Advances in Intelligent Systems Research, volume 133
2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE2016)

hi-RF: Incremental Learning Random Forest for Large-Scale Multi-class Data Classification

Tingting Xie*, Changjian Wang and Yuxing Peng
National Lab for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, China, 410073
*Corresponding author

***Abstract*—In recent years, dynamically growing data and large-scale data classification research. Most traditional methods struggle to balance the precision and computational burden when data and its number of classes increased. However, some methods are with weak precision, and the others are time-consuming. In this paper, we propose an incremental learning method, namely, heterogeneous incremental Nearest Class Mean Random Forest (hi-RF), to handle this issue. It is a heterogeneous method that either replaces trees or updates trees leaves in the random forest adaptively, to reduce the computational time in comparable performance, when data of new classes arrive. Specifically, to keep the accuracy, one proportion of trees are replaced by new NCM decision trees; to reduce the computational load, the rest trees are updated their leaves probabilities only. Most of all, out-of-bag estimation and out-of-bag boosting are proposed to balance the accuracy and the computational efficiency. Fair experiments were conducted and demonstrated its comparable precision with much less computational time.**

Keywords-large scale multi-class classification; Incremental Learning; random forest; heterogeneous incremental Nearest Class Mean Random Forest

I. INTRODUCTION

With data increasingly available in every second, automatic classification has attracted wide attention in both research and industry. Though there are thousands of classes in this dataset, new visual classes and data grow dynamically in practice. To retrain a model from scratch when new data arrives is very time-consuming and requires full access to the original training data.

Among the state of the art solutions, Random Forest (RF)[23], [24] has been tested as a successful representative for large-scale image classification given its performance of high efficiency and accuracy. As a standard supervised learning method, RF training usually assumes that the number of class is fixed and the class distribution is fixed [21], which is unable to handle dynamic growing data and classes. On-line variants of RF [9], [16]were proposed to address one aspect of this issue-the class distribution is fixed. They assume the numbers of classes as well as the class labels are known beforehand, so they cannot handle new classes of data.

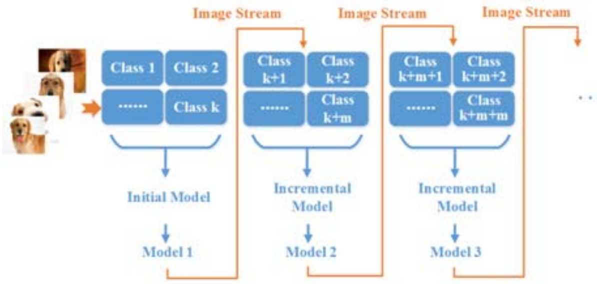

FIGURE I. INCREMENTAL LEARNING CONCEPT

The problem, that a few classes are available at the beginning and new classes of data arrive consequentially, is called incremental learning problem. The conceptual structure of the incremental learning is illustrated in Figure. I. Previous methods often re-use sub-trees in incremental learning and only handle data from one new class. In order to achieve high accuracy, they have to re-use a large proportion of subtrees, which is very time-consuming. Their stiff procedure (i.e., they can only handle the arriving data of one class) and high computational load limit the generalization. To deal with these problems, we propose a large-scale multi-class data classification method-Heterogeneous Incremental Nearest Class Mean Random Forest (hi-RF). The method can handle multiple new classes of data and reduce computational cost in matchable accuracy.

In hi-RF, Rolling Release NCM decision trees (RRN) was presented to integrate new classes. For a proportion of trees in RF, RRN retrained them with new NCM decision trees [3], based on their out-of-bag error [13] comparing with a particular threshold, once fresh data arrives. The training dataset is the subset of the new and the old training samples for each NCM decision tree, so it is convenient to add upcoming samples and keep the accuracy. Because of that, we can address the issue of multiple new classes. To reduce the computational cost, Regenerate leaves probabilities (RLP) was advanced for the rest proportion of trees in RF. RLP only updates the probabilities of leaf node, but not modifies any part of the decision tree. Its negligible time cost reduces the computing load greatly. To balance the accuracy and computational cost, it is very important to select a particular threshold to separate RF into two parts. The threshold selection is based on Out-Of-Bag estimation (OOB estimation), which is calculated by out-of-bag error. Most time, we usually limit the proportion of re-training by decreasing the threshold as small as possible to alleviate the computational burden with comparable accuracy. During the procedure of rolling release new data, to make OOB estimation balance the trees better and more stability to new data, Out-Of-Bag boosting (OOB boosting) is proposed. OOB boosting increases the out-of-bag error of trees updated by RLP, so there is more chance to retrain the trees as the next time new data added, and it preserves the stability of trees and maintains the accuracy.

In this work, the contributions of the paper are as follows. First, RRN was adopted to retrain the decision tree based on their out-of-bag error in a rolling manner, thus providing freedom to integrate multiple classes of data. Second, RLP was presented to reduce the computational load. As a sequence, it performs pretty good on the challenging large-scale ImageNet datasets, because the computational load is a little more than half of that for re-training all the trees. Experiments showed that, compared with off-line random forest, the computational time of hi-RF is a little more than half of the off-line, with the accuracy loss 2.70%. Third, OOB estimation is used to balance computational load and accuracy by adaptively cutting down the proper proportion of decision trees, re-training trees and updating trees. Besides, OOB boosting helps a lot to maintain the trees stability in the incremental procedure.

This paper is organized as follows: we firstly summarize related works in Section II. Section III demonstrates the whole working procedure of hi-RF. Experiments and brief discussion will be presented in Section IV and we will conclude with a summary in Section V.

II. RELATED WORK

Since ImageNet appeared several years ago, many researchers are working on data classification with RF. Compared to other classifiers, RF have performed very well in classification, visual tracking [6], feature detection [11], and even in cancer detection from mass proteomic pictures [5]. RF have been a good candidate for computer vision for several reasons. Firstly, they run fast both in training and classification. Secondly, they can be easily parallelized, thus they are always considered for distributed computing and GPU acceleration. Additionally, RF have inherently hierarchical structure, so they can be made locally modification in deep layers, which only affect part of the data in negligible cost [3], [4]. Above all, compared to SVM and other ensemble methods, RF are naturally multi-class, more robust against noises and more suitable for generalization [14], [15].

Researchers have proposed dimensionality reduction methods to address fine-grained classification. [21] is a particular one, and it effectively reduce subspace size as well as improve classification performance. [22] adopts sparse coding to train a dictionary of visual words and then convert SIFT descriptors into sparse vectors. However, the methods above were applied to off-line learning, so they cannot process data grown continuously.

Many papers used RF in on-line mode to address the problem of large data. They often combine on-line bagging and on-line decision trees with random feature selection, but they are either memory intensive because of storing all the data in every node [16] or have to discard important information if parent nodes change. Besides, there is some researchers focus on improving the accuracy. Because of the hierarchy structure, error can be propagated further down to the tree. While some methods solve this problem, they combine decision trees with ideas from neural networks [9], but they often lose the $O(\log n)$ evaluation time because samples are propagated to all nodes. [6] proposed a novel on-line algorithm that has neither of the problems. The algorithm allows discarding entire trees by out-of-bag-error and consecutively growing of new trees. Nevertheless, in practice, data streams might obtain new classes, and they cannot adapt to this problem.

In incremental learning, algorithms was established with competitive accuracy to off-line RF without retraining the whole forests. The approach [2] learns a discriminative metric on the initial set of classes, and classifies samples simply based on the nearest class mean. Adding a new class means inserting its mean in the pool of classes, leading to a negligible computational burden. They could not update the structure of forests by itself when a new class added, which will finally lead to suboptimal performance. Conversely, we propose to update the structure of hi-RF to integrate new classes. While in [3], [4], they proposed a novel and almost perfect method to solve the problem, but they can only process data of one class added. That is to say, it is hard to process new classes in batch, which limits their usability.

Transfer learning is also along with incremental learning, while it intends to reduce the amount of labeled data required to learn a new class [20]. Incremental learning is different from it for two reasons. Firstly, transfer learning is limited to one-vs-all classification, Secondly, the intention of incremental learning is to add a new class efficiently instead of exploiting the knowledge from previous classes to reduce the amount of annotation necessary for good performance.

In this paper, we demonstrate a heterogeneous approach, inspired by on-line bagging and nearest class mean. Usually, there are trees containing too much noise, and they hamper the accuracy. It is necessary to replace trees in poor performance. And the simple use of class means as centroids makes it efficient to train a decision tree, and it results in high performance with much less cost. Finally, hi-RF is proposed to achieve better performance.

III. HETEROGENEOUS INCREMENTAL NEAREST CLASS MEAN RANDOM FOREST

Random Forest (RF) is an ensemble classifier that aggregates many decision trees. It does a popularity vote of individual tree when predicting a class. This character makes RF suitable to incremental learning.

Heterogeneous incremental Nearest Class Mean Random Forest (hi-RF) is a new way to modify the random forest for incrementally integrated data. It is constructed by Nearest Class Mean decision trees (NCM decision trees) [3], [4]. When new data arrives, hi-RF processes the trees in the forest in two ways according a standard, which is calculated by OOB estimation and OOB boosting, which will be described in Ш-A. For each tree, if it does not satisfy the standard, it will be retrained with bootstrap samples, which combined previous and fresh data. Otherwise, it will be updated with the leaves node probabilities changed only.

ALGORITHM I. HETEROGENEOUS INCREMENTAL NEAREST CLASS MEAN RANDOM FOREST (hi-FR)

```
Require:
  Previous model,m;
  Mumber of decision trees in $m, s$ ;
  The set of out-of-bag for each tree, $O$ ;
  Old training data, $D_o$ ;
  New training data, $D_n$ ;
Ensure:
  New model, $M$ ;
1:  for each time new data arriving do
2:    $threshold \leftarrow OOB\_estimation(O)$
3:    for each tree $T_i$ in $m$ do
4:      if $T_i$ does not reach the threshold then
5:        $T_i \leftarrow Retraining(D^o, D^n)$
6:      else
7:        $T_i \leftarrow Updating(D^o, D^n)$
8:      end if
9:    end for
10:   $O \leftarrow OOB\_boosting(O)$
11:   $M \leftarrow Bagging(T_1, T_2, \dots, T_s)$
12: end for
13: return $M$
```

TABLE I. VARIABLE DESCRIPTIONS

Notation	Description
k	the number of features
x	a feature vector, $x = \{x_1, x_2, \dots, x_k\}$
$y, \hat{y}$	the actual and the predicted label
(x, y)	a sample
c, C	a single class, the number of trees
s	the number of trees in RF
δ	the particular threshold
t	the computing time
acc	Accuracy for RF
T	The whole RF of decisong tree, $T = \{T_1, T_2, \dots, T_s\}$
O	the distribution of out-of-bag error rates, $O = \{o_1, o_2, \dots, o_s\}$
D, D^o, D^n	All,old,new training data, bootstrap sample set is, $D_i\,(i \in 1, 2, \dots, s\}$

The former procedure of retraining is called RRN and the latter of updating is named RLP. The whole process is depicted in Algorithm I, especially, RRN can be presented by line 2:4, and RLP can be presented by 2,5:6, which two will be described in section 3 and 4.

To make mathematical model understood easier, we represent variable description in Table I . hi-RF is aggregated by s versions of decision tree, and each tree is denoted $T_i(i \in \{1,2,\cdots, s\})$[26]. The aggregation averages over the versions and does a popularity vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set D^o and new arriving dataset D^n, and using $D = D^o + D^n$ as new learning sets, which is represented by $D_i(i \in \{1, 2, ..., s\})$. The accuracy of each tree is measured by out-of-bag error rates, whose distribution for all the trees is O and out-of-bag error rate for each tree is $o_i(i \in \{1, 2, ..., s\})$. Besides, we establish a standard value δ to measure the performance of one tree.

ALGORITHM II. OOB ESTIMATION

```
Require:
  The whole RF, $T$ ;
  Bootstrap samples for each tree, $D_i(i \in \{1, 2, ..., s\})$ ;
  Old training data, $D_o$;
  New training data, $D_n$ ;
Ensure:
  The threhold, $\delta$ ;
for each tree $T_i$ in $T$ do
  $D = D^o + D^n$
  $D^l = D - D_i$          // $D^l$ :left-out sample set
  for $(x, y)$ in $D^l$ do
    $\hat{y} \leftarrow T_i(x)$
    if $\hat{y} = y$ then
      $I\{\hat{y} = y\} = 1$          //
      $I\{\hat{y} = y\}$ :loss function
    else
      $I\{\hat{y} = y\} = 0$
    end if
  end for
  $o_i = \frac{\sum_{(x,y) \in D^l} I\{\hat{y} = y\}}{|D^l|}$          // calculate the out-of-bag error for $T_i$
end for
$O \sim N(\mu, \sigma^2)$
$(\mu, \sigma^2) \leftarrow MaxLikelihoodEstimation(O, \mu, \sigma^2)$
$\delta = \mu$
return $T_i$
```

A. *OOB Estimation*

OOB estimation is to calculate a threshold δ to decide if a tree should be retrained or updated only. The threshold can be regarded as a standard representation of an average tree in RF. If the out-of-bag error rate of a single tree is less than δ , it can be regarded as a good tree with pretty good performance and we preserve its structure by updating the leaves node probabilities only, which is called RLP. If not, it will be replaced by a new tree, and the procedure of replacing is called RRN. The whole procedure is presented in Algorithm2.

The most important thing in OOB estimation is to calcaulate the out-of-bag error of each tree. Each tree is constructed using bootstrap samples from training set, so the left-out samples can be used to measure the performance of it [13]. Therefore, the out-of-bag error rate of each tree can be computed, and the particular threshold can be estimated according to the mean value of out-of-bag error's Gaussian distribution. We will show the computing procedure in detail as following.

The most important thing is to calculate the out-of-bag error rate for each tree. As mentioned above, D is the whole data set, and $D_i (i \in \{1, 2, ..., s\})$ is the bootstrap samples for T_i. Thus, the out-of-bag error rate of T_i can be calculated by Equation 1.

$$o_i = \frac{\sum_{(x,y)\in D^l} I\{\hat{y} = y\}}{|D^l|} \tag{1}$$

I is a loss function, i.e., when a sample is arriving, if the predicted label is the same as actual label, the loss will be \$0\$, and the loss will be 1 otherwise. The equation can be described by Equation 2 blow.

$$I\{\hat{y} = y\} = \begin{cases} 0, & if\ \hat{y} = y \\ 1, & otherwise \end{cases} \tag{2}$$

Then, achieve the mean value based on the distribution of out-of-bag error. As we know, are independent and have identical distribution, so the list $o_i (i \in \{1, 2, ..., s\})$ of them, presented by O, is satisfied Gaussian distribution [29], i.e., $O \sim N(\mu, \sigma^2)$, in which μ is the expected value, and σ^2 is the deviation. Hence, δ can be represented by the expected value μ, that is, $\delta = \mu$. While μ can be estimated by $O=\{o_1,o_2,...,o_s\}$ through maximum likelihood estimation[30]. Assuming that the probability of $o < o_i$ is $p(o_i)$, we can achieve it by Equation3.

$$p(o_i) = \frac{1}{\sqrt{2\sigma^2\pi}} \bullet e^{-\frac{(o_i-\mu)^2}{2\sigma^2}} \tag{3}$$

According to maximum likelihood [reference paper], we should maximize all the probability above, which can be represented by likelihood function 4.

$$L(o_1, o_2, ..., o_s; \mu, \sigma) = \prod_{i=1}^{s} p(o_i) = \prod_{i=1}^{s} \frac{1}{\sqrt{2\sigma^2\pi}} \bullet e^{-\frac{(o_i-\mu)^2}{2\sigma^2}} \tag{4}$$

Next step, maximizing the $logL(o_1, o_2, \ldots, o_s; \mu, \sigma)$ is to make the gradient of μ and σ euqal to zero. With these, we can calculate the threshold δ throughout μ.

However, the data will be rolling released more than once, so keep the stability of the model is very important. It is convinced that the trees updated by RLP is not that reliable, because this approach does not change the splitting function or size of the trees. Therefore, when forest need to be updated the second time or more, the trees are retrained by RRN or RLP will be well balanced. To make a balance, OOB boosting is proposed to do that. It boosts the out-of-bag-error rate of trees updated by RRN, which helps us to retrain the tree updated by RLP compared the one updated by RRN with almost the same accuracy.

The boosting out-of-bag-error rate o is decided by a learning rate α and $tanh(o)$, which is decipted in Equation 5.

$$o = o + \alpha * tanh(o) \tag{5}$$

ALGORITHM III. ROLLING RELEASE NCM DECISION TREE (RRN)

```
Require:
  The previous random forest, $T^o$;
  The out-of-bag error for each tree, $O_i$;
  The threshold, $\delta$;
  All training data, $D$;
Ensure:
  The new random forest, $T^n$;
for each tree $T_i$ in $T^o$ do
  if $o_i > \delta$ then
    $Dascarding(T_i)$
    $D_i \leftarrow Bootstrap(D)$
    // The growing a new NCM decision tree
    $T_i \leftarrow Growing(D_i)$ :
    if all the $(x, y)$ in $D_i$ has the same label $k$ or reach the max Depth then
      return classes probabilities P
    else
      // $K$ is a random subset of the classes observed in $D_n$
      $K \leftarrow ClassesSubset(D_i)$
      class centroids $\theta_n \leftarrow CalClassCentroids(D_n)$
      $K_{left}, K_{right} \leftarrow ChooseBestFeature(D, \theta_n)$ according to Information gain
      $D_{left}, D_{right} \leftarrow SplitDataSet(D, K_{left}, K_{right}, \theta_n)$
      build subtree:
      $T_{left} = Growing(D_{left}), T_{right} = Growing(D_{right})$
      $T \leftarrow K_{left} : T_{left} + K_{right} : T_{right}$
    end if
    return $T$
  end if
end for
$T^n \leftarrow \{T_1, T_2, \ldots, T_s\}$
```

B. *RRN*

Rolling release NCM decision trees (RRN) presented in this paper is a rolling method to incrementally incorporate new classes of data unlimitedly. It modifies the current classifier to a new one with trees re-constructed by learning new bootstrapping samples when new data arrives. RRN is described in Algorithm 3 and will be explained in detail as following.

Each time new data arrives, we calculate a threshold according to the OOB estimation. For each tree, if its OOB error is less than the threshold, it means this tree does not reach the average level. Thus, it will be discarded and replaced by a new NCM decision tree [3], [4]. The training samples of a new tree are bootstrapping samples from the new and old data. When all the trees were judged, RRN outputs a new bagging forest to make data classification. The growing of a NCM decision tree is illustrated in function $Growing$.

The construction of constructing a NCM decision tree is an iterative process. Once all the samples in a node share the same class or this tree has reach the max depth, the algorithm will return the leaves node probabilities of classes. Each iterative procedure consists of 3 functions, they will be described in detail as follows.

1) Cal class centroids

Since performance of a decision tree is heavily depended on splitting function, NCM decision trees take class centroid instead of best feature. Supposing samples D_n arrive at node \$n\$, we can randomly select a subset K of classes observed in D_n . For each class K in K , D_n^k is the subset of D_n of class $k \in K$ and the corresponding class centroid θ_n^k can be calculated by Equation 6.

$$\theta_n^k = \frac{1}{|D_n^k|} \bullet \sum_{i \in D_n^k} x_i \tag{6}$$

2) Choose best feature

After calculating the class centroids in node n , they are randomly as signed to left or right child node and form two class sets K_{left} and K_{right} . This is the split function of data, which is represented by $f_n : x \to \{0,1\}$. According to it, D_n can be divided into D_{left} and D_{right} . In this paper, information gain is taken to measure the split. First, information entropy of D_n , D_{left} and D_{right} are computed as E_n , E_{left} and E_{right} . Second, calculate the information gain described in Equation 7.

$$Gain(D_n, f) = E_n - \sum_{i \in \{left,right\}} E_i \tag{7}$$

Assuming the random splitting function space is F_n , so we can choose the best one with the most information gain as Equation 8.

$$f_n = \underset{f \in F_n}{\operatorname{argmax}} \, Gain(D_n, f) \tag{8}$$

From mentioned above, a few comparisons is needed at each node, thus reducing the time cost to resort to expensively learn best feature and offering non-linear classification at node level. While that is important to large-scale data classification, and the performance of NCM classifier is comparable to that of linear SVMs which obtain current state-of-the-art performance [2], so we take use of NCM classifier instead of decision trees.

3) Split data set

As we choose the best split function, the class centroids can be distributed to the left or right node. Thus, we can split the data into two datasets according to the distance between data and class centroids, which is defined by Equation 9.

$$\hat{y} = \underset{k \in K}{\operatorname{argmin}} \, || x - \theta_n^k ||^2 \tag{9}$$

To meet the requirement of constantly growing datasets, we have to update the trees continuously, which is similar with rolling release. For rolling release NCM decision trees, there are two ways to explain rolling release. First, when data is arriving, in the procedure of bagging, we judge all the trees and update one by one. Second, data growing will not stop, so each time data rolling falls, we rolling release the whole forest. It is also another rolling release.

As we all know, trees are generated by training samples, if you want to change the data, you must update the tree. Once concerned with extending, cutting, modifying any sub-tree of the previous tree, data category and distribution will be the bottleneck. To avoid that, we just grow a new entire tree without influencing the trees before. Also, the training data is the bootstrapping dataset of the original and present data, it makes our models suit to incorporate new classes of data unlimitedly.

ALGORITHM IV. REGRENERATE LEAVES PROBABILITIES (RLP)

```
Require:
  The previous random forest, $T^o$;
  The out-of-bag error for each tree, $O_i$;
  The threshold, $\delta$;
  All training data, $D$;
Ensure:
  The new random forest, $T^n$;
for each tree $T_i$ in $T^o$ do
  if $o_i > \delta$ then
    $Dismiss(T_i, leavesprobabilities)$
    $D_i \leftarrow Bootstrap(D)$
    $K \leftarrow Classesset(D_i)$
    // The updating of a decision tree
    Define $T_i \leftarrow Updating(T_i, D_i)$ :
    $n\_node \leftarrow NumberOfLeavesNode(T_i)$
    $\{D_i^1, D_i^2, \ldots, D_i^{n\_node}\} \leftarrow Fall(T_i, D_i)$
    for $D_i^j \in \{D_i^1, D_i^2, \ldots, D_i^{|K|}\}$ do
      for $k$ in $K$ do
        $D_i^j(k) \leftarrow DatasetOfClassK(D_i^j, k)$
        $P_i^j(k) = \frac{|D_i^j(k)|}{|D_i^j|}$
      end for
      $P_i^j = \{P_i^j(1), P_i^j(2), \ldots, P_i^j(|K|)\}$
    end for
    $P_i = \{P_i^1, P_i^2, \ldots, P_i^{n\_node}\}^T$
    update $P_i$ to $T_i$
    EndDefine
  end if
  $T^n \leftarrow \{T_1, T_2, \ldots, T_s\}$
end for
```

C. RLP

ReGenerate leaves probabilities (RLP) is a light weight method to update the probabilities of the leaf node without structure modification of a decision tree, which contributes a lot to computational cost reduction. The whole procedure is described in Algorithm 4 and will be explained in detail as following.

RLP also takes use of the threshold generated by OOB estimation. For each tree in RF, if the out-of-bag error is more than threshold means the versions of trees reach the average level and are robust in learning information. Thus, RLP is to update these trees. Usually, once a decision tree is constructed, the split function in each intern node is determined and the probabilities of leaves is generated. RLP is proposed to update that.

Since the splitting function is unchanged, we can put the new data into the tree T_i and get a new data set of class in each leaf node j, whose number for a tree is n_node. Supposing the input data is D_i and class set K, then the data set of class is $\{D_i^j(1), D_i^j(2), \ldots, D_i^j(|K|)\}$. Therefore, the probabilities of class k in node j P_i^j can be described in Equation 10.

$$P_i^j(k) = \frac{|D_i^j(k)|}{|D_i^j|} \tag{10}$$

Thus the probabilities of node j is $P_i^j = \{P_i^j(1), P_i^j(2), \ldots, P_i^j(|K|)\}$, and we draw a matrix of leaf probabilities of T_i that is $P_i = \{P_i^1, P_i^2, \ldots, P_i^{n\_node}\}^T$. Finally, we update the leaf probabilities P_i of T_i and achieve a new tree which can make more classifications without training time aparting from traversing the training data once.

D. Computational Efficiency

NCM has been used for large-scale image classification in [2], and shown its excellent performance on it. Besides, NCM also reduces time cost when used in RF. As we know, the feature space in a traditional decision tree in RF is $\sqrt{k}$ (It should be $floor(\sqrt{k})+1$, but we leave $floor$ out to make it easy to read), where k is the number of features. However, the NCM decision tree takes centroid centers as split function and it is splitted randomly, so it is a constant time to split the feature space. Therefore, NCM decision tree also outperforms other decision trees like C4.5 [28] for its time-saving.

When new class of data arriving, off-line mode requires us to retrain a new RF, which hinders the computational efficiency greatly. Supposing that, the training time of training a NCM tree is t_{ncm}, the whole computation complexity t_{off} for s trees is Equation 11.

$$t_{off} = s * t_{ncm} \tag{11}$$

While for hi-RF, trees in RF are separated into two groups by the threshold: $T^1 = \{T_{11}, T_{12}, \ldots, T_{1n_1}\}$ (RRN) and $T^2 = \{T_{21}, T_{22}, \ldots, T_{2n_2}\}$ (RLP), and always $n_2 < \frac{s}{2} < n_1$. For trees in T^1, they are retrained a new decision tree, and the computational time can be represented by t_1. For trees in T^2, RLP only updates their leaf probabilities, and their computational time can be decipted by t_2. Therefore, the training time of hi-RF t_{hi-RF} is showed in Equation 12.

$$t_{hi-RF} \approx n_1 * t_{ncm} \tag{12}$$

While t_2 is equals to the testing time of all the input data D, t_1 is the training time of D, so we can draw a conclusion as described in Equation 13.

$$t_{hi-RF} = n_1 * t_1 + n_2 * t_2 \tag{13}$$

In a conclusion, we can infer that the training time of hi-RF is up to the number of RRN trees. In practice, n_1 is always less than $\frac{s}{2}$, so the training time of hi-RF is much less than off-line random forest.

IV. EXPERIMENT

In this section, we take Large Scale Visual Recognition Challenge 2010 (ILSVRC2010) for evaluation [17]. It contains 96833 training samples, 15000 test samples, and 15000 validation samples in 100 classes and each sample was represented by 4096 features. We use features extracted by AlexNet [25] in 10 classes. While the data in other categories is not available in public and the feature extracting is beyond the scope of this paper, we will not talk about anything about it. To achieve better performance, the original data are normalized.

Since on-line random forest and previous incremental methods either can not handle new classes of data or can just handle one new classes of data, so we cannot make a comparison with them. Therefore, experiments are conducted to compare the accuracy and computational cost of the novel hi-RF with its off-line counterpart on ILSVRC2010. From the result, it demonstrates its suitability on the large scale multi-class incremental data classification. When compared with off-line mode, it only needs much less than the off-line computational with approximately accuracy. Besides, we conduct a serial experiments to verify the stability and effectiveness of hi-RF in the senario with incrementally increasing data.

A. *Data Pre-processing*

To obtain nice results, data pre-processing is often of vital importance when concerned with exploratory data classification or building a good and robust prediction model. In this paper, we normalized the input data and the accuracy improvement is more than 5%. As mentioned in section III, the centroid of each class k is θ_k and the deviation can be calculated as σ. Then, all the samples which can be presented by $(x_1, x_2, \ldots, x_m)$ was normalized by Equation 14 below.

$$x_k = \frac{x_k - \theta_k}{\sigma} \tag{14}$$

B. *Computational Efficiency and Accuracy*

In this section, computational efficiency and accuracy are evaluated in nine different proportions of the class number of incremental data and the original data. We cannot even try all kinds of proportions, so we take the above nine as representations, and we take 50 trees constructing the RF. After the procedure hi-RF, the nine situations have 100 classes data, while the original data is n, $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$. Accuracy, training time and testing time are compared between hi-RF and off-line method, which is the baselene, while hi-RF has n classes of original data and off-line random forest retrain 50 trees with all the data. The final results are showed in Figure II.

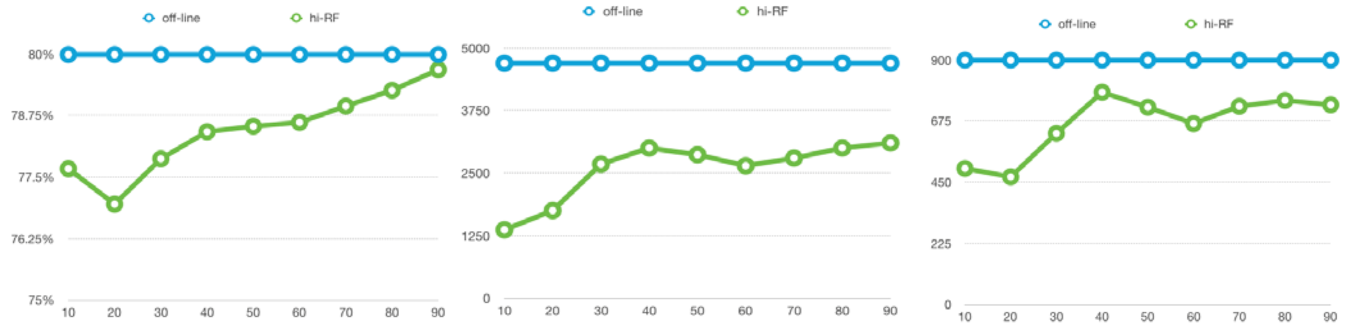

FIGURE II. COMPARISON AMONG BASELINE HI-RF ,WHILE THE ORIGINAL CLASS NUMBER IS RANGE FROM 10 TO 90, AND THE ADDED DATA RANGE FROM 90 TO10,AND THE FINAL DATA CLASSIS 100 A) ACCURACY B) TRAINING TIME C) TESTING TIME

Accuracy. Figure II(a) plots the accuracy for the baseline and our approach. It shows that when the original data is small, the accuracy of hi-RF is not that satisfied to baseline. Especially when the original data is 20, the accuracy of final 100 classes data is down to the mountain valley 76.95%. Nonetheless, after that, the accuracy is increased with the number of the initial classes. It ranges from 76.95% to 79.43%, while the accuracy of off-line forest is 79.65%. The class number of original data is closer to all the data afer added, the result is more accurate. We can also infer that the difference ranges from 0.22% to 2.70%, thus showing the stability regardless of the original class number.

Training time. Figure II(b) plots the training time for the baseline and our approach. The least training time is 1365.6, and the most one is 2396.12, while the off-line computational time is 3733.23. Although the training time is theoretically half the baseline, the complexity of the tree's structure for updating the leaf node probabilities creates an additional overhead. When the structure of trees are simple just as initial number is less than 40, the training time is less than half off-line computational time, and more than half otherwise.

Testing time. Figure II(c) plots the testing time for the baseline and our approach. The final testing time is varied with the trees complexity, the curve increases for the trees complexity is increases sharply and gently increases for the trees' structure is stable.

Finally, we report the accuracy, training time and testing time of hi-RF and baseline on 100 classes incrementally added from n, $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$. From massive experiments, it is proved practically that hi-RF requires a little more than half computational time of the baseline mode with negligible loss of accuracy. It is not sensitive to the original classes.

C. Stability

In practices, multiple classes can appear in batches, so to verify the stability in different scenario, the number of simultaneous classes to add is an interesting parameter to study. According to the sparse batch and intensive batch, the experiment designs to three part with different incremental strategy, i.e., different batch size. The initial random forest was constructed by 10 classes of training data, and new arriving data is increased in batch. The batch size in the three group of experiments are 1, 5, 10, and the number of classes of final data is 20, 50, 100. Experiments were conducted between hi-RF and off-line random forest as baseline, especially, hi-RF with using OOB boosting and without using OOB boosting were also compared to have a better understanding of its stability when applying OOB boosting to it, which is showed in 3, 4, 5.

1) Step size equals to 1

Comparison with step size 1 among baseline, hi-RF with OOB boosting and OOB unboosting is showed in Figure III. The data increases so slow that the hi-RF with OOB boosting or OOB un-boosting accuracy keeps pace with the off-line mode. While the training time and testing time curve trend keeps similar with 2(b) and 2(c).

2) Step size equals to 5

Figure IV shows the above three comparisons. The curve trend is also the same with 3. So we will not show it in detail.

3) Step size equals to 10

Figure V shows the above three comparisons. It said that, hi-RF is very stable, although the step size is big. OOB boosting works a little to the hi-RF without OOB boosting, because the accuracy in green is always higher than the curve in blue.

In a conclusion, the accuracy loss of hi-RF with OOB boosting and without OOB boosting keep pace with that in last section, and shows that the stability even the initial model is training by few classes of data. Besides, it seems that hi-RF with OOB boosting applied is a little more stable and accurate than hi-RF without OOB boosting applied.

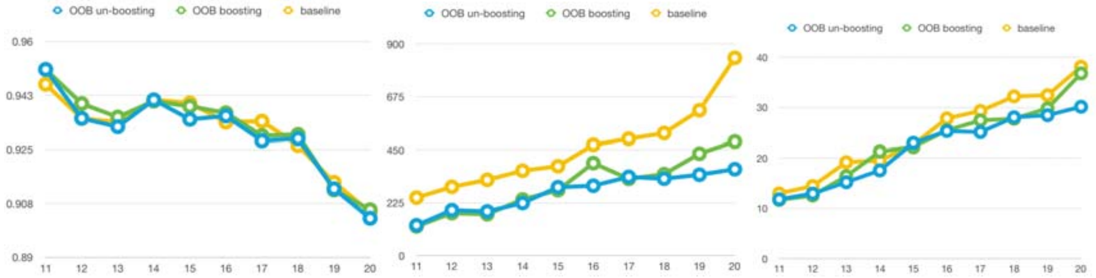

FIGURE III. COMPARISON AMONG BASELINE HI-RF WITH OOB BOO,STING AND OOB UNBOOSTING,WHILE THE ORIGINAL CLASS NUMBER IS RANGE FROM 10 TO 20,AND THE STEP SIZE IS 1 A) ACCURACY B) TRAINING TIME C) TESTING TIME

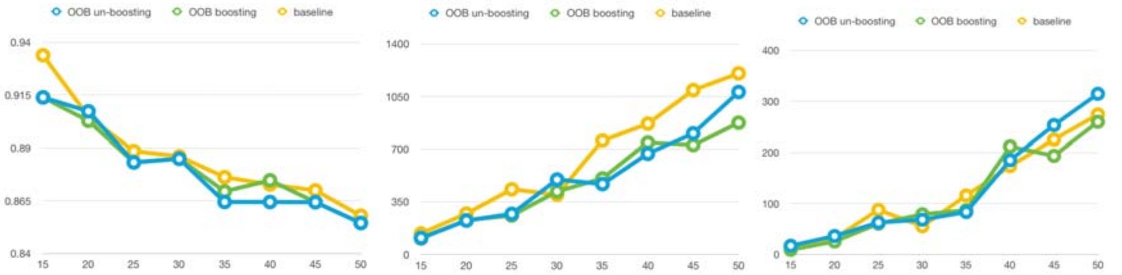

FIGURE IV. COMPARISON AMONG BASELINE HI-RF WITH OOB BOO,STING AND OOB UNBOOSTING,WHILE THE ORIGINAL CLASS NUMBER IS RANGE FROM 10 TO 50,AND THE STEP SIZE IS 5 A) ACCURACY B) TRAINING TIME C) TESTING TIME

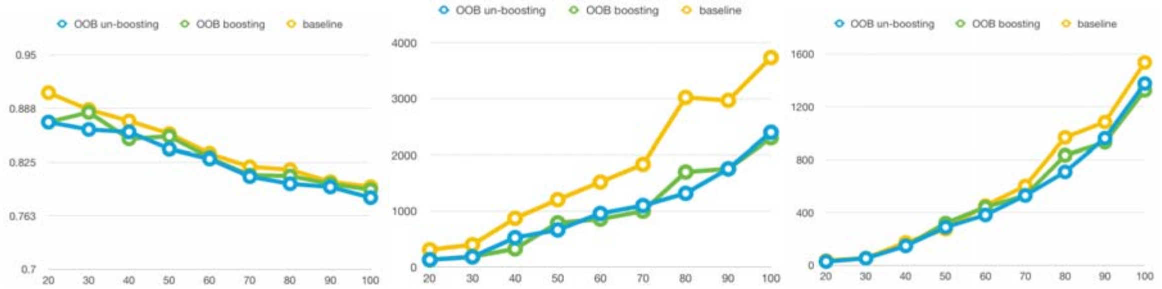

FIGURE V. COMPARISON AMONG BASELINE HI-RF WITH OOB BOO,STING AND OOB UNBOOSTING,WHILE THE ORIGINAL CLASS NUMBER IS RANGE FROM 10 TO 100,AND THE STEP SIZE IS 10 A) ACCURACY B) TRAINING TIME C) TESTING TIME

V. CONCLUSION

In this paper, we describe a hi-RF which integrates new classes gracefully for large-scale multi-class data classification. Extensive experiments were performed and showed that it preserved the overall accuracy with much less computational cost. Therefore, we can implement this method to scenario when few training samples are available at the beginning, and it can improve performance with least cost.

Each tree in a forest is built and tested independently from other trees. Hence the overall training and testing procedures can be performed in parallel later. Although the feature space is large, only a fraction of it will actually useful. Therefore, we will design a paralleled algorithm to make use of the informative features only.

ACKNOWLEDGEMENT

This work is sponsored in part by the National Basic Research Program of China (973) under Grant No.2016YFB1000101, and the National Natural Science Foundation of China under Grant No. 61402514.

REFERENCES

[1] Deng J, Berg A C, Li K, et al. What does classifying more than 10,000 image categories tell us?[C]//European conference on computer vision. Springer Berlin Heidelberg, 2010: 71-84.

[2] Mensink T, Verbeek J, Perronnin F, et al. Distance-based image classification: Generalizing to new classes at near-zero cost[J]. IEEE transactions on pattern analysis and machine intelligence, 2013, 35(11): 2624-2637.

[3] Ristin M, Guillaumin M, Gall J, et al. Incremental learning of NCM forests for large-scale image classification[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014: 3654-3661.

[4] Ristin M, Guillaumin M, Gall J, et al. Incremental Learning of Random Forests for Large-Scale Image Classification[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 38(3): 490-503.

[5] Barrett J H, Cairns D A. Application of the random forest classification method to peaks detected from mass spectrometric proteomic profiles of cancer patients and controls[J]. Statistical Applications in Genetics and Molecular Biology, 2008, 7(2).

[6] Saffari A, Leistner C, Santner J, et al. On-line random forests[C]//Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on. IEEE, 2009: 1393-1400.

[7] Elgawi O H, Hasegawa O. Online incremental random forests[C]//Machine Vision, 2007. ICMV 2007. International Conference on. IEEE, 2007: 102-106.

[8] Oza N C. Online bagging and boosting[C]//2005 IEEE international conference on systems, man and cybernetics. IEEE, 2005, 3: 2340-2345.

[9] Basak J. Online adaptive decision trees[J]. Neural computation, 2004, 16(9): 1959-1981.

[10] Dantone M, Gall J, Fanelli G, et al. Real-time facial feature detection using conditional regression forests[C]//Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012: 2578-2585.

[11] Yang H, Patras I. Privileged information-based conditional regression forest for facial feature detection[C]//Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International Conference and Workshops on. IEEE, 2013: 1-6.

[12] Yang H, Patras I. Sieving regression forest votes for facial feature detection in the wild[C]//Proceedings of the IEEE International Conference on Computer Vision. 2013: 1936-1943.

[13] Breiman L. Out-of-bag estimation[R]. Technical report, Statistics Department, University of California Berkeley, Berkeley CA 94708, 1996b. 33, 34, 1996.

[14] Gall J, Yao A, Razavi N, et al. Hough forests for object detection, tracking, and action recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2011, 33(11): 2188-2202.

[15] Schulter S, Leistner C, Roth P M, et al. On-line Hough Forests[C]//BMVC. 2011: 1-11.

[16] Utgoff P E, Berkman N C, Clouse J A. Decision tree induction based on efficient tree restructuring[J]. Machine Learning, 1997, 29(1): 5-44.

[17] Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

[18] Gall J, Lempitsky V. Class-specific hough forests for object detection[M]//Decision forests for computer vision and medical image analysis. Springer London, 2013: 143-157.

[19] Gall J, Lempitsky V. Class-specific hough forests for object detection[M]//Decision forests for computer vision and medical image analysis. Springer London, 2013: 143-157.

[20] Gao J, Ling H, Hu W, et al. Transfer learning based visual tracking with gaussian processes regression[C]//European Conference on Computer Vision. Springer International Publishing, 2014: 188-203.

[21] Xu B, Ye Y, Nie L. An improved random forest classifier for image classification[C]//Information and Automation (ICIA), 2012 International Conference on. IEEE, 2012: 795-800.

[22] Tang F, Lu H, Sun T, et al. Efficient image classification using sparse coding and random forest[C]//Image and Signal Processing (CISP), 2012 5th International Congress on. IEEE, 2012: 781-785.

[23] Ho T K. Random decision forests[C]//Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on. IEEE, 1995, 1: 278-282.

[24] Ho T K. The random subspace method for constructing decision forests[J]. IEEE transactions on pattern analysis and machine intelligence, 1998, 20(8): 832-844.

[25] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[26] Breiman L. Bagging predictors[J]. Machine learning, 1996, 24(2): 123-140.
[27] Quinlan J R. Induction of decision trees[J]. Machine learning, 1986, 1(1): 81-106.
[28] Quinlan J R. C4. 5: programs for machine learning[M]. Elsevier, 2014.
[29] Hazewinkel M. Normal distribution[J]. Encyclopedia of Mathematics, 2001, 4.
[30] Aldrich J. RA Fisher and the making of maximum likelihood 1912-1922[J]. Statistical Science, 1997, 12(3): 162-176.