Boosted Random Forest

Yohei Mishina, Masamitsu Tsuchiya and Hironobu Fujiyoshi
Department of Computer Science, Chubu University, 1200 Matsumoto-cho, Kasugai, Aichi, Japan

Keywords: Boosting, Random Forest, Machine Learning, Pattern Recognition.

Abstract: The ability of generalization by random forests is higher than that by other multi-class classifiers because of the effect of bagging and feature selection. Since random forests based on ensemble learning requires a lot of decision trees to obtain high performance, it is not suitable for implementing the algorithm on the small-scale hardware such as embedded system. In this paper, we propose a boosted random forests in which boosting algorithm is introduced into random forests. Experimental results show that the proposed method, which consists of fewer decision trees, has higher generalization ability comparing to the conventional method.

1 INTRODUCTION

Random forest(Breiman, 2001) is a multi-class classifier that is robust against noise, has high discrimination performance, and is capable of training and classifying at high speed. It is therefore attracting attention in many fields, including computer vision, pattern recognition, and machine learning(Amit and Geman, 1997), (Lepetit and p. Fua, 2006), (J. Shotton and Cipolla, 2008), (J. Shotton et al., 2011), (Gall et al., 2011). Random forest controls loss of generalization in training due to overfitting by introducing randomness in bagging and feature selection(Ho, 1998) when constructing an ensemble of decision trees. Each decision tree in random forest is independent, so high speed can be attained by parallel processing in tree training and classification. Boosting(Freund and Schapire, 1995) is a typical ensemble training algorithm that is used to sequentially construct classifiers for random forest that involve independent decision trees. Boosting is an ensemble training algorithm that combines weak learners, which individually have low discriminating performance, to construct a classifier of higher discriminating performance. Boosting generally attains high discrimination by sequential training of classifiers in which the training sample for which the previous classifier produced classification errors is used to train the subsequent classifier to produce correct classification. However, boosting tends to overfit the training sample. Random forest, on the other hand, uses randomness in the construction of the decision trees, thus avoiding overfitting to the training sample. For that reason, a large number of decision trees must be constructed to obtain high generality. However, increasing the number of decision trees increases the memory requirements, so that approach is not suited to implementation on small-scale hardware such as embedded system. We therefore propose a boosted random forest method, in which a boosting algorithm is introduced in random forest. The proposed method constructs complementary classifiers by successive decision tree construction and can yield classifiers with smaller decision trees while maintaining discrimination performance.

2 RANDOM FOREST

Random forest is an ensemble training algorithm that constructs multiple decision trees. It suppresses overfitting to the training samples by random selection of training samples for tree construction in the same way as is done in bagging(Breiman, 1996),(Breiman, 1999), resulting in construction of a classifier that is robust against noise. Also, random selection of features to be used at splitting nodes enables fast training, even if the dimensionality of the feature vector is large.

2.1 Training Process

In the training of random forest, bagging is used to create sample sub sets by random sampling from the training sample. One sample set is used to construct one decision tree. At splitting node n, sample set $\mathcal{S}_n$ is split into sample sets $\mathcal{S}_l$ and $\mathcal{S}_r$ by comparing the

Mishina Y., Tsuchiya M. and Fujiyoshi H..
Boosted Random Forest.
DOI: 10.5220/0004739005940598
In *Proceedings of the 9th International Conference on Computer Vision Theory and Applications* (VISAPP-2014), pages 594-598
ISBN: 978-989-758-004-8

value of feature quantity x_i with a threshold value τ. The splitting function of the splitting node selects combinations that can partition the most samples from among randomly selected features $\{\mathbf{f}_k\}_{k=1}^{K}$ and threshold $\{\tau_h\}_{h=1}^{H}$ for each class. The recommended number of feature selections, K, is the square root of the feature dimensionality. The evaluation function used for selecting the optimum combination is the information gain, ΔG. The splitting processing is repeated recursively until a certain depth is reached or until the information gain is zero. A leaf node is then created and the class probability $P(c|l)$ is stored.

2.2 Classification Process

An unknown sample is input to all of the decision trees, and the class probabilities of the leaf nodes arrived at are output. The class that has the largest average of the class probabilities obtained from all of the decision trees, $P_t(c|\mathrm{x})$, according to Eq. (1) is the classification decision.

$$P(c|\mathrm{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathrm{x}) \tag{1}$$

2.3 Number of Decision Trees and Discriminating Performance

Random forest achieves generality by using a large number of decision trees for ensemble training. However, it is difficult to obtain the optimum number of trees for training, so there are many redundant decision trees that consume a large amount of memory.

3 BOOSTED RANDOM FOREST

Random forest is robust against noise and has high generality because of random training. However, it requires many decision trees, as using fewer decision trees reduces performance. It therefore cannot maintain generality when implemented on small-scale hardware. For that reason, boosting is introduced to random forest. The purpose of using boosting is to maintain generality even with a small number of decision trees by using the fact that sequential training constructs complementary decision trees for the training samples.

3.1 Training Process

The proposed training algorithm involves one procedure for when the sample weighting is updated and another procedure for when there is no

Algorithm 1: Proposed method.

Require: Training samples $\{\mathbf{x}_1, y_1, w_1\}, \ldots, \{\mathbf{x}_N, y_N, w_N\}$;
$\mathbf{x}_i \in \mathcal{X}, y_i \in \{1, 2, \ldots, M\}, w_i$
Init: Initialize sample weight w_i:
$w_i^{(1)} \Leftarrow \frac{1}{N}$.
Run:
for $t = 1 : T$ **do**
Make subset $\mathcal{S}_t$ from training samples.
$\Delta G^{max} \Leftarrow -\infty$.
for $k = 1 : K$ **do**
Random sampling from feature f_k.
for $h = 1 : H$ **do**
Random sampling from threshold τ_h.
Split $\mathcal{S}_n$ into $\mathcal{S}_l$ or $\mathcal{S}_r$ by f_k and τ_h.
Compute information gain ΔG:
$\Delta G = E(\mathcal{S}_n) - \frac{|\mathcal{S}_l|}{|\mathcal{S}_n|}E(\mathcal{S}_l) - \frac{|\mathcal{S}_r|}{|\mathcal{S}_n|}E(\mathcal{S}_r)$.
if $\Delta G > \Delta G_{max}$ **then**
$\Delta G_{max} \Leftarrow \Delta G$
end if
end for
end for
if $\Delta G_{max} = 0$ or reach a maximum depth **then**
Store the probability distribution $P(c|l)$ to leaf node.
else
Generating a split node recursively.
end if
if Finished training of decision tree **then**
Estimate class label $\hat{y}_i$:
$\hat{y}_i = \arg\max_c P_t(c|l)$.
Compute error rate of decision tree ε_t:
$\varepsilon_t = \sum_{i:y_i \neq \hat{y}_i}^{\mathrm{N}} w_i^{(t)} \;/\; \sum_{i=1}^{\mathrm{N}} w_i^{(t)}$.
Compute weight of decision tree α_t:
$\alpha_t = \frac{1}{2}\log\frac{(\mathrm{M}-1)(1-\varepsilon_t)}{\varepsilon_t}$
if $\alpha > 0$ **then**
Update weight of training sample $w_{i,t+1}$:
$$w_i^{(t+1)} = \begin{cases} w_i^{(t)}\exp(\alpha_t) & \text{if } y_i \neq \hat{y}_i \\ w_i^{(t)}\exp(-\alpha_t) & \text{otherwise.} \end{cases}$$
else
Reject a tree
end if
end if
end for

updating. First, a training sample of size N, $\{\mathbf{x}_1, y_1, w_1\}, \ldots, \{\mathbf{x}_N, y_N, w_N\}$, that have a feature of dimension d and class labels $y \in M$ are prepared. The training sample weight, w, is initialized to $\frac{1}{N}$. Sample sets are created by random sampling from the training sample. Decision trees are constructed using the sample sets in the same way as in random forest. Proposed algorithm 1 is described in above.

3.1.1 Node Splitting

The flow of the proposed method is illustrated in Fig. 1. The splitting function selects combinations of randomly-prepared features and thresholds that have

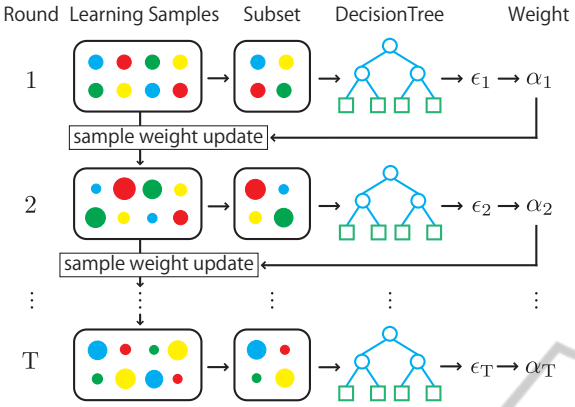

Figure 1: Training algorithm of the proposed method.

the highest information gain. The information gain ΔG is computed by

$$\Delta G = E(\mathcal{S}_n) - \frac{|\mathcal{S}_l|}{|\mathcal{S}_n|}E(\mathcal{S}_l) - \frac{|\mathcal{S}_r|}{|\mathcal{S}_n|}E(\mathcal{S}_r), \quad (2)$$

where $\mathcal{S}_n$ is sample set at node n, $\mathcal{S}_l$ is sample set at left child node, $\mathcal{S}_r$ is sample set at right child node, and $E(\mathcal{S})$ is entropy computed by

$$E(\mathcal{S}) = -\sum_{j=1}^{M} P(c_j)\log P(c_j). \quad (3)$$

In calculating the information gain, the samples are prioritized by largest weight and the probability of class c_j, $P(c_j)$, is calculated using the weight of sample i, w_i computed using

$$P(c_j) = \sum_{i \in S \wedge y_i = c_j} w_i \,/\, \sum_{i \in S} w_i, \quad (4)$$

where, S is the sample set that arrived at node. A leaf node is created when recursive splitting has developed the decision tree to a certain depth or when the information gain of a sample set that has reached a node is zero. The leaf node stores the class probability $P(c)$ obtained with Eq. (4).

3.1.2 Decision Tree Weighting

In the same way as for multi-class boosting(Kim and Cipolla, 2008), (Saberian and Vasconcelos, 2008), the decision tree weight, α_t, is calculated by

$$\alpha_t = \frac{1}{2}\log\frac{(\mathrm{M}-1)(1-\varepsilon_t)}{\varepsilon_t}, \quad (5)$$

where, ε_t is the error rate of the decision tree and M is the number of classes. The expected value for the successful classification rate in random classification is $\frac{1}{M}$. If the classification error rate exceeds $1-\frac{1}{M}$, the value of α is negative in Eq. (5) and the decision tree is discarded. The training sample is classified by the constructed decision trees and the error rate is calculated from the weights of the incorrectly classified samples as

$$\varepsilon_t = \sum_{i:y_i \neq \hat{y}_i}^{\mathrm{N}} w_i^{(t)} \,/\, \sum_{i=1}^{\mathrm{N}} w_i^{(t)}. \quad (6)$$

3.1.3 Updating Training Sample Weights

Decision trees that easily correct classification of the samples that have been incorrectly classified in the next step are constructed by making the weights of incorrectly classified samples large as

$$w_i^{(t+1)} = \begin{cases} w_i^{(t)} \exp(\alpha_t) & \text{if } y_i \neq \hat{y}_i \\ w_i^{(t)} \exp(-\alpha_t) & \text{otherwise,} \end{cases} \quad (7)$$

where $\hat{y}_i$ is estimated class label using

$$\hat{y}_i = \arg\max_c P_t(c|l). \quad (8)$$

After updating the training sample weights, the weights are normalized to N. Constructing the decision trees and updating the training sample weights in that way is repeated to obtain T decision trees and T weighted decision trees. After all decision trees have been constructed, the decision tree weights are normalized.

3.2 Classification Process

An unknown sample is input to all of the decision trees as shown in Fig. 1, and the class probabilities that are stored in the arrived-at leaf nodes are output. Than, the outputs of the decision trees, $P_t(c|x)$, are weight-averaged using the decision tree weights as

$$P(c|\mathrm{x}) = \frac{1}{T}\sum_{t=1}^{T} \alpha_t P_t(c|\mathrm{x}). \quad (9)$$

The class that has the highest probability $\hat{y}$ is output as the classification result by

$$\hat{y} = \arg\max_c P(c|\mathrm{x}). \quad (10)$$

4 EXPERIMENTAL RESULTS

To show the effectiveness of the proposed method, the number of nodes are compared for the proposed method and the conventional method at the same level of generalization ability. For the proposed method, we investigated procedures with and without sequential sample weight updating.

4.1 Data Set

The evaluation experiments used four data sets, Pendigits, Letter, Satelite, and Spambase, from those published by the UCI Machine Learning Repository as a set of machine training algorithm benchmarks. The data sets are described briefly in Table 1.

Table 1: Data sets.

Dataset	Training	Tests	Class	Dim.
Pendigits	7,494	3,498	10	16
Letter	10,000	10,000	26	16
Satelite	4,435	2,000	6	36
Spamebase	3,221	1,380	2	57

4.2 Training Parameter

In this experiment, the depth of the decision trees for training parameters was fixed at 5, 10, 15, and 20. We compared the minimum value of the miss rate by changing the number of decision trees. The number of candidates for the split function was 10 times the square root of the number of feature dimensions.

4.3 Experimental Results

The error ratio for each dataset of the conventional random forest(RF) and for the proposed method(boosted random forest with or without sample weight updating, BRF, BRF w/o updating) are shown in Table 2, 3, and 4 respectively. The miss rate of the proposed method was lower than that of the random forest when the number of depth for decision trees was shallower. In contrast, when the number of depth for decision trees was deeper, the miss rate of the proposed method was equal to or lower than that of the random forest. It is clear that the classification performance of the boosted random forest with updating sample weight is superior. Figure 2 shows the miss rate for each depth in the case of the "Pendigits" dataset. From Fig. 2, we realized that the random forest requires more depth in order to obtain a higher classification performance than the proposed method. In contrast, the proposed "boosted random forest" method does not require more depth because it has been trained by choosing the split function with difficult training samples at upper nodes, which have a heavier sample weight.

Table 2: Error rate by random forest [%].

Depth	Pen	Letter	Satelite	Spame	Ave.
5	8.38	22.95	13.05	6.88	12.82
10	3.97	7.25	9.95	5.58	6.69
15	3.66	6.40	9.30	5.14	6.13
20	3.69	6.20	9.10	4.71	5.92

Table 3: Error rate by boosted random forest w/o updating [%].

Depth	Pen	Letter	Satelite	Spame	Ave.
5	8.18	22.75	12.95	6.81	12.67
10	3.92	7.50	10.05	5.58	6.76
15	3.66	6.20	9.40	5.07	6.08
20	3.72	6.10	9.10	4.64	5.89

Table 4: Error rate by boosted random forest [%].

Depth	Pen	Letter	Satelite	Spame	Ave.
5	3.72	11.45	10.70	4.57	7.61
10	2.55	5.00	8.35	4.13	5.01
15	2.69	4.55	8.25	3.77	4.81
20	2.66	4.40	8.20	3.55	4.70

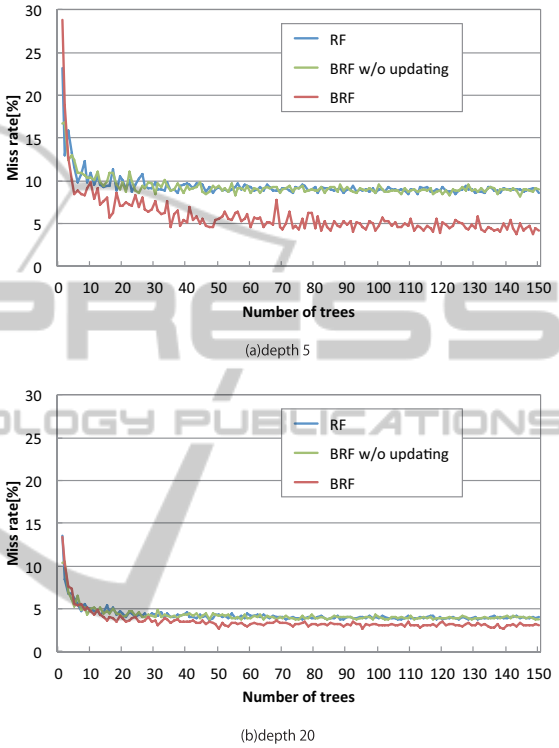

Figure 2: Generalization error of pendigits.

4.4 Memory Usage for Decision Trees

For the implementation of decision trees on small-scale hardware, less memory is better. Therefore, in this section we compare the amount of memory needed for each method. For each node, the total memory of a split function is 11 bytes, of which the selected feature dimension is 1 byte, the threshold is

2 bytes, and the pointer for a child node is 8 bytes. For each leaf node, total memory is estimated by the number of class bytes. Thus, we estimate the amount of memory B required for decision trees by

$$B = \sum_{t=1}^{T} (N_{s,t} \times 11 + N_{l,t} \times M), \qquad (11)$$

where the number of trees is T, number of split nodes is $N_{s,t}$, number of leaf nodes is $N_{l,t}$ and number of classes is M.

Figure 3 shows the amount of memory for each method with minimum error rate and the reduction ratio of the proposed method compared to the random forest. The amount of memory required by the

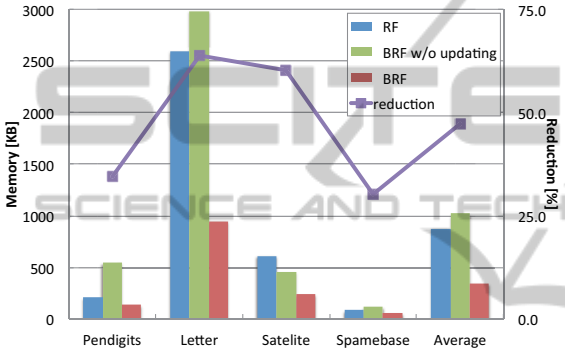

Figure 3: Amount of memory and reduction rate.

proposed "boosted random forest" method is significantly reduced while maintaining the higher classification performance. Due to sequential training, the boosted random forest consists of complementary decision trees that enable the final classifier to be constructed in favor of those instances misclassified by previous decision trees.

5 CONCLUSIONS

We have proposed a boosted random forest in which a boosting algorithm is introduced to a conventional random forest. The boosted random forest maintains a high classification performance, even with fewer decision trees, because it constructs complementary classifiers through sequential training by boosting. Experimental results show that the total memory required by the boosted random forest is 47% less than that of the conventional random forest. It is thus suited to implementation in low-memory, small-scale hardware applications such as embedded systems. Our future work includes experimental evaluation for image recognition problems that are currently difficult to classify.

REFERENCES

Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. In *Neural Computation*. MIT Press.

Breiman, L. (1996). Bagging predictors. In *Machine Learning*. Springer.

Breiman, L. (1999). Using adaptive bagging to debias regressions. In *Technical Report*. Statistics Dept. UCB.

Breiman, L. (2001). Random forests. In *Machine learning*. Springer.

Freund, Y. and Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*. Springer.

Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. (2011). Hough forests for object detection, tracking, and action recognition. In *Pattern Analysis and Machine Intelligence*. IEEE.

Ho, T. K. (1998). The random subspace method for constructing decision forests. In *Pattern Analysis and Machine Intelligence*. IEEE.

J. Shotton, a. A. F., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition*. IEEE.

J. Shotton, M. J. and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Computer Vision and Pattern Recognition*. IEEE.

Kim, T.-K. and Cipolla, R. (2008). Mcboost: Multiple classifier boosting for perceptual co-clustering of images and visual features. In *Advances in Neural Information Processing Systems*.

Lepetit, V. and p. Fua (2006). Keypoint recognition using randomized trees. In *Pattern Analysis and Machine Intelligence*. IEEE.

Saberian, M. and Vasconcelos, N. (2008). Multiclass boosting: Theory and algorithms. In *Advances in Neural Information Processing Systems*.