PAPER *Special Section on Optimization and Learning Algorithms of Small Embedded Devices and Related Software/Hardware Implementation*

Boosted Random Forest

Yohei MISHINA[†a)], ***Nonmember***, **Ryuei MURATA**[†b)], ***Student Member***, **Yuji YAMAUCHI**[†c)], **Takayoshi YAMASHITA**[†d)], ***and*** **Hironobu FUJIYOSHI**[†e)], ***Members***

SUMMARY Machine learning is used in various fields and demand for implementations is increasing. Within machine learning, a Random Forest is a multi-class classifier with high-performance classification, achieved using bagging and feature selection, and is capable of high-speed training and classification. However, as a type of ensemble learning, Random Forest determines classifications using the majority of multiple trees; so many decision trees must be built. Performance increases with the number of decision trees, requiring memory, and decreases if the number of decision trees is decreased. Because of this, the algorithm is not well suited to implementation on small-scale hardware as an embedded system. As such, we have proposed Boosted Random Forest, which introduces a boosting algorithm into the Random Forest learning method to produce high-performance decision trees that are smaller. When evaluated using databases from the UCI Machine learning Repository, Boosted Random Forest achieved performance as good or better than ordinary Random Forest, while able to reduce memory use by 47%. Thus, it is suitable for implementing Random Forests on embedded hardware with limited memory.

key words: *Boosting, Random Forest, machine learning, pattern recognition*

1. Introduction

Machine learning refers to learning techniques that analyze data, obtain useful knowledge, rules, and evaluation criteria from it, and perform highly accurate classification. It has wide ranging applications and has been used in many fields, including classification, identification, image analysis, voice recognition, and market prediction. With such a record, practical implementations of machine learning in various other fields are also anticipated.

Some typical machine learning techniques include Neural Networks [1], Boosting [2], Support Vector Machines [3], and Random Forests [4]. Random Forests is a multi-class classifier that is resistant to noise, has strong classification capabilities, and is fast both in training and classification. For these reasons, it has received much attention in many fields, including computer vision, pattern recognition, and machine learning [5]–[9]. Random Forests builds an ensemble of decision trees incorporating a component of randomness. This enables it to control the drop in generalization performance due to over-training. Since each decision tree in Random Forests is independent, each tree can be processed in parallel during learning and classification. It is thus well-suited to hardware implementation because processes can be parallelized easily. Random Forests also incorporates randomness when building decision trees, so overfitting the learning sample does not occur. Conversely, it has the disadvantage that many decision trees are needed to achieve good generalization performance. Increasing the number of decision trees requires larger amounts of memory. So, in order to implement it on small-scale hardware, memory use must be reduced. This research eliminates this bottleneck to implementing a Random Forest that is both accurate and compact.

To do so, we propose the Boosted Random Forests technique, which combines a boosting algorithm with Random Forest. The proposed technique trains complementary discriminators by building decision trees successively. This makes it possible to build classifiers with fewer decision trees while still maintaining classification performance.

1.1 Related Works

Random Forest
Random Forest is a multi-class classifier algorithm that introduces randomness through bagging and feature selection and is easily parallelized. However, to achieve good generalization performance, a large number of trees must be built, requiring large amounts of memory. Random ferns [11] is a method that reduces the memory used for decision trees. It reduces memory use by using the same decision function for branch nodes at the same level. However, simply reducing the number of nodes results in a drop in classification performance. With Alternating Decision Forests [12], weightings are updated for successive samples to make classification of training data easier overall. Nodes at the same level in each decision tree are also trained in parallel for each level. As a result, compared to a Random Forest with the same tree height, it has been shown to achieve better classification accuracy. This enables shallower decision trees with the same accuracy to be built, suggesting that Random Forest sizes can be reduced by making correlations among decision trees.

Manuscript received November 27, 2014.
Manuscript revised May 15, 2015.
Manuscript publicized June 22, 2015.

[†]The authors are with the Department of Computer Science, Chubu University, Kasugai-shi, 487–8501 Japan.
a) E-mail: mishi@vision.cs.chubu.ac.jp
b) E-mail: mryua@vision.cs.chubu.ac.jp
c) E-mail: yuu@vision.cs.chubu.ac.jp
d) E-mail: yamashita@cs.chubu.ac.jp
e) E-mail: hf@cs.chubu.ac.jp

DOI: 10.1587/transinf.2014OPP0004

Boosting
Boosting is an ensemble learning algorithm that builds a classifier sequentially [13]–[17]. Like Bagging, Boosting is an ensemble learning algorithm that builds a classifier with higher classification performance by combining weak classifiers with low performance. Bagging combines classifiers independently trained using bootstrap sampling. In contrast, Boosting conducts successive training by adjusting the weightings for samples so that a training sample that was misclassified by the previous classifier will be correctly classified by the next one. By training complementary classifiers, Boosting can achieve higher accuracy with fewer weak classifiers. In this research, we create interrelation among the decision tries by treating each decision tree as a weak discriminator and adjusting the weightings of misclassified samples when training successive decision trees.

1.2 Overview of Our Approach

Our objective in this paper is to implement Random Forest using small-scale hardware. The following two points constitute the contribution of our method.

1. Reduced memory use
   Memory use can be reduced by reducing the number of trees (i.e. nodes) in the Random Forest. To reduce the number of trees, independent trees are taken as weak classifiers and a boosting algorithm is introduced so they complement each other. This makes it possible to decrease memory use while avoiding a drop in performance.
2. Easily parallelizable algorithm
   By using Random Forest as a framework with simple and easily parallelizable structure, classification can be parallelized in hardware easily, which is another advantage of this method.

2. Random Forest

Random Forest is an ensemble training algorithm that constructs multiple decision trees as shown in Fig. 1. 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 [18], [19], 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 value of feature quantity x_i

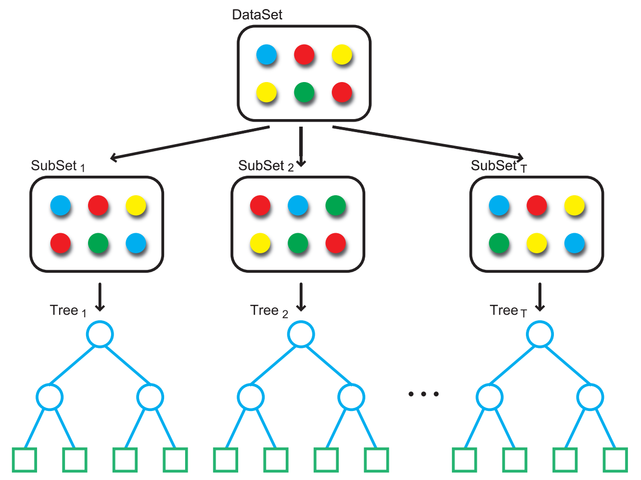


Fig. 1 Random Forest structure. Random Forest generates random sampling subsets from the training set. Duplication and omission of selected samples is permitted. Decision trees are built using these subsets. This is repeated for the number of trees.

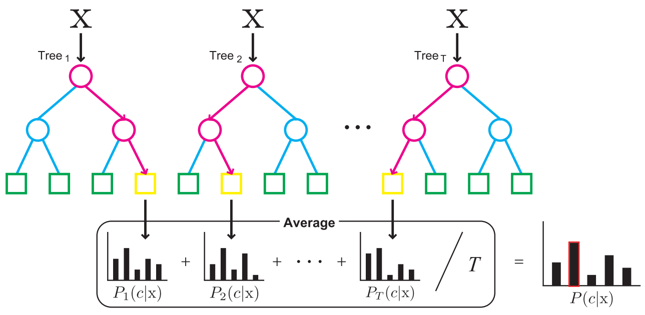


Fig. 2 Classification process for Random Forest. For new input samples, Random Forest averages from the output leaf node of each tree to compute class probabilities.

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 as shown in Fig. 2, 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 updating. First, a training sample of size $\{\mathbf{x}_1, y_1, w_1\}, \ldots, \{\mathbf{x}_N, y_N, w_N\}$, 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.

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 $\tau_h$.
        Split $\mathcal{S}_n$ into $\mathcal{S}_l$ or $\mathcal{S}_r$ by $f_k$ and $\tau_h$.
        Compute information gain $\Delta G$:
        $\Delta G = E(\mathcal{S}_n) - \frac{|\mathcal{S}_l|}{|\mathcal{S}_n|}E(\mathcal{S}_l) - \frac{|\mathcal{S}_r|}{|\mathcal{S}_n|}E(\mathcal{S}_r)$.
        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 $\epsilon_t$:
        $\epsilon_t = \sum_{i:y_i \neq \hat{y}_i}^{N} w_i^{(t)} \Big/ \sum_{i=1}^{N} w_i^{(t)}$.
      Compute weight of decision tree $\alpha_t$:
        $\alpha_t = \frac{1}{2}\log\frac{(M-1)(1-\epsilon_t)}{\epsilon_t}$
      Update weight of training sample $w_{i,t+1}$:
        $w_i^{(t+1)} = \begin{cases} w_i^{(t)}\exp(\alpha_t) & \text{if } y_i \neq \hat{y}_i \\ w_i^{(t)}\exp(-\alpha_t) & \text{otherwise.} \end{cases}$
    end if
  end for
```

3.1.1 Node Splitting

The flow of the proposed method is illustrated in Fig. 3. The splitting function selects combinations of randomly-prepared features and thresholds that have the highest information gain. The information gain ΔG is computed by

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

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

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

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

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

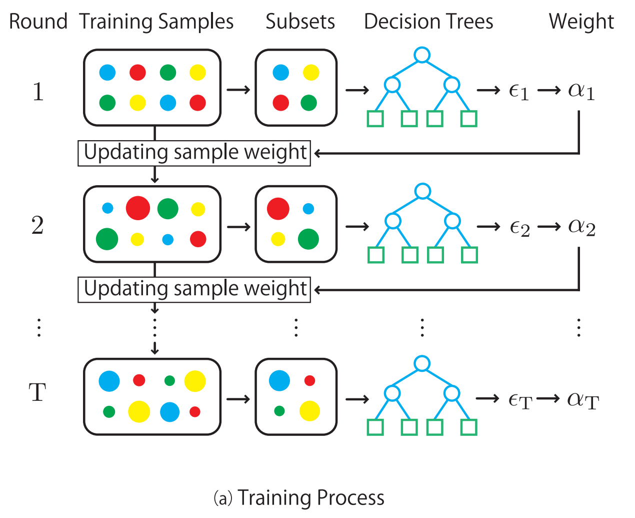
where, S is the sample set that arrived at node. A leaf node is created when recursive splitting has developed the decision tree to a certain depth or when the information gain of a sample set that has reached a node is zero. The leaf node stores the class probability $P(c)$ obtained with Eq. (4).

3.1.2 Decision Tree Weighting

In the same way as for multi-class boosting [17], [20], the decision tree weight, α_t, is calculated by

$$\alpha_t = \frac{1}{2}\log\frac{(M-1)(1-\epsilon_t)}{\epsilon_t}, \tag{5}$$

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



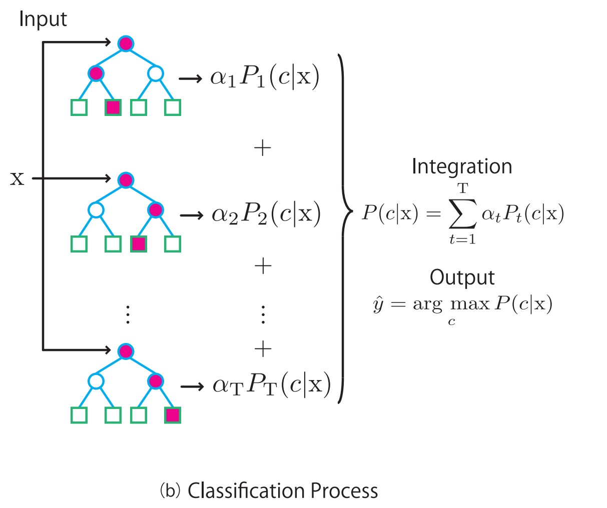

Input
x
$\alpha_1 P_1(c|x)$
$\alpha_2 P_2(c|x)$
$\alpha_T P_T(c|x)$
Integration
$P(c|x) = \sum_{t=1}^{T} \alpha_t P_t(c|x)$
Output
$\hat{y} = \arg\max_c P(c|x)$
(b) Classification Process

Fig. 3 Training algorithm of the proposed method. In the training process, successive decision trees are built with a boosting algorithm that adds weightings to the training samples. This enables generalization performance to be maintained while using a small number of decision trees. In the classification process, as with Random Forest, class probabilities are calculated by averaging the output leaf nodes for the unknown sample from each of the trees. However, care must be taken in totaling the weights from each tree.

the error rate is calculated from the weights of the incorrectly classified samples as

$$\epsilon_t = \sum_{i:y_i \neq \hat{y}_i}^{N} w_i^{(t)} \Big/ \sum_{i=1}^{N} w_i^{(t)}. \tag{6}$$

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

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

Table 1 Data sets.

Data set	Training	Tests	Class	Feature dimensions
Pendigits	7,494	3,498	10	16
Letter	10,000	10,000	26	16
Satellite	4,435	2,000	6	36
Spam base	3,221	1,380	2	57
Iris	75	75	3	4

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

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

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. 3, and the class probabilities that are stored in the arrived-at leaf nodes are output. Then, the outputs of the decision trees, $P_t(c|x)$, are weight-averaged using the decision tree weights as

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

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

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

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 five data sets, Pendigits, Letter, Satellite, Spam base, and Iris, from those published by the UCI Machine Learning Repository [21] as a set of machine training algorithm benchmarks. The data sets are described briefly in Table 1.

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 data set 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 Tables 2, 3, and 4 respectively.

For Pen, Letter, Satellite, Spam base data sets which have enough number of training samples, 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. For Fig. 4 shows the miss rate for each depth in the case of the "Pendigits" data set. From Fig. 4, 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.

For Iris data set which has small number of training samples, there is no significant changes between the Random Forest and the Boosted Random Forest for classification performance. This is because 5 of depth is enough for representing diversity of the training samples in small scaled problem.

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

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

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 5 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 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.

Table 2 Error rate by Random Forest [%].

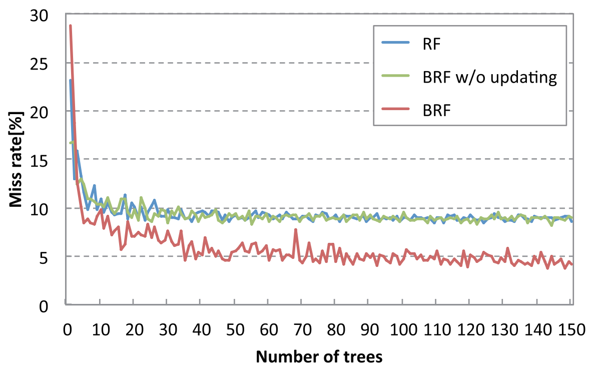
Depth	Pen	Letter	Satellite	Spam	Iris	Average
5	8.38	22.95	13.05	6.88	2.67	10.79
10	3.97	7.25	9.95	5.58	2.67	5.88
15	3.66	6.40	9.30	5.14	2.67	5.43
20	3.69	6.20	9.10	4.71	2.67	5.27

Table 3 Error rate by Boosted Random Forest w/o updating [%].

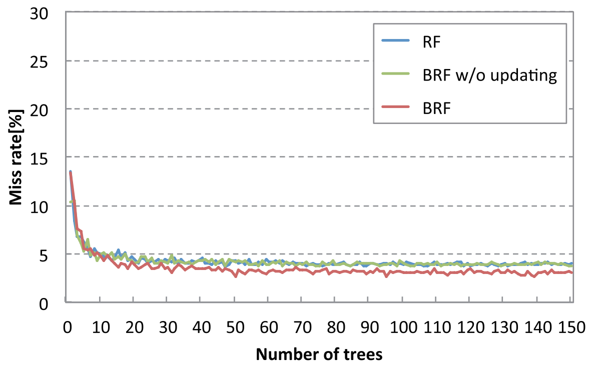
Depth	Pen	Letter	Satellite	Spam	Iris	Average
5	8.18	22.75	12.95	6.81	2.67	10.67
10	3.92	7.50	10.05	5.58	2.67	5.94
15	3.66	6.20	9.40	5.07	2.67	5.40
20	3.72	6.10	9.10	4.64	2.67	5.24

Table 4 Error rate by Boosted Random Forest [%].

Depth	Pen	Letter	Satellite	Spam	Iris	Average
5	3.72	11.45	10.70	4.57	2.67	6.62
10	2.55	5.00	8.35	4.13	2.67	4.54
15	2.69	4.55	8.25	3.77	2.67	4.38
20	2.66	4.40	8.20	3.55	2.67	4.76



(a)depth 5



(b)depth 20

Fig. 4 Generalization error of Pendigits. (a) Shows the miss rates for numbers of trees with depth 5, while (b) shows miss rates for trees of depth 20. The blue lines show results for Random Forest, the green lines for Boosted Random Forest without updating the weightings, and the red lines for Boosted Random Forest.

Table 5 Computational cost for training and classification process.

	Training time [msec/tree]	Classification time [μsec/sample]
Random Forest	13.88	23
Boosted Random Forest w/o updating	41.66	33
Boosted Random Forest	14.04	29

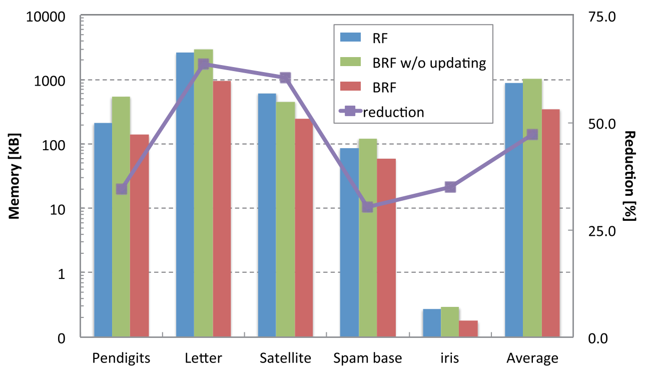


Fig. 5 Amount of memory and reduction rate. The left axis shows memory use for each of the methods to achieve comparable classification rates, while the right axis shows reduction rates.

4.5 Computational Cost for Decision Trees

Table 5 shows computational cost for training and classification process. The training time means the average time of constructing a decision tree. The classification time means the average time for computing a final classification score of decision trees. From Table 5, we see that the processing time of the proposed method for training takes more time than the conventional Random Forest, because the proposed method needs an extra process for updating weights. On the other hand, the processing time of the proposed method for classification process is less than that of the conventional Random Forest. This is because the proposed method can construct efficient decision trees by introducing boosting algorithm into training.

5. Conclusion

We have proposed a Boosted Random Forest in which a boosting algorithm is introduced to a conventional Random Forest. With a conventional Random Forest, decision trees are built independently, so classifiers cannot be built to complement each other. Thus, if the performance of each decision tree in the Random Forest is limited, classification performance drops. In contrast to this, 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. For the Letter data set, which used the most memory in these experiments, and implementing on the Cyclone III FPGA EP3C120 image processing board from Altera Corp., Random Forest used approximately 66% of the total RAM of 3,888 KB, while Boosted Random Forest could be implemented using approximately 24% of memory. Random Forest can also process in parallel, so implementations can replicate hardware resources as many times as is permissible to increase speed through parallel processing. 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

[1] D.E. Rumelhart, G.E. Hintont, and R.J Williams, “Learning representations by back-propagating errors,” Nature, vol.323, no.6088, pp.533–536, 1986.

[2] Y. Freund and R.E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” Computational Learning Theory, Lecture Notes in Computer Science, vol.904, pp.23–37, Springer, 1995.

[3] C. Cortes and V. Vapnik, “Support-vector networks,” Machine Learning, vol.20, no.3, pp.273–297, 1995.

[4] L. Breiman, “Random forests,” Machine Learning, vol.45, no.1, pp.5–32, 2001.

[5] Y. Amit and D. Geman, “Shape quantization and recognition with randomized trees,” Neural Computation, vol.9, no.7, pp.1545–1588, 1997.

[6] V. Lepetit and P. Fua, “Keypoint recognition using randomized trees,” IEEE Trans. Pattern Anal. Mach. Intell., vol.28, no.9, pp.1465–1479, 2006.

[7] J. Shotton, M. Johnson, and R. Cipolla, “Semantic texton forests for image categorization and segmentation,” IEEE Conference on Computer Vision and Pattern Recognition, pp.1–8, 2008.

[8] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, “Real-time human pose recognition in parts from single depth images,” Machine Learning for Computer Vision, Studies in Computational Intelligence, vol.411, pp.119–135, Springer, 2013.

[9] J. Gall, A. Yao, N. Razavi, L. van Gool, and V. Lempitsky, “Hough forests for object detection, tracking, and action recognition,” IEEE Trans. Pattern Anal. Mach. Intell., vol.33, no.11, pp.2188–2202, 2011.

[10] T.K. Ho, “The random subspace method for constructing decision forests,” IEEE Trans. Pattern Anal. Mach. Intell., vol.20, no.8, pp.832–844, 1998.

[11] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, “Fast keypoint recognition using random ferns,” IEEE Trans. Pattern Anal. Mach. Intell., vol.32, no.3, pp.448–461, 2010.

[12] S. Schulter, P. Wohlhrt, C. Leistner, A. Saffari, P.M. Roth, and H. Bischof, “Alternating decision forests,” IEEE Conference on Computer Vision and Pattern Recognition, pp.508–515, 2013.

[13] Y. Freund and R. Schapire, “Experiments with a new boosting algorithm,” Proc. Thirteenth International Conference on Machine Learning, pp.148–156, 1996.

[14] R. Schapire and Y. Singer, “Improved boosting algorithms using confidence-rated predictions,” Machine Learning, no.37, pp.297–336, 1999.

[15] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regres-

sion: A statistical view of boosting," Technical report, Department of Statistics, Stanford University, 1998.

[16] J. Zhu, H. Zou, S. Rosset, and T. Hastie,"Multi-class adaboost," Statistics and Its Interface, pp.349–360, 2009.

[17] M. Saberian and N. Vasconcelos, "Multiclass boosting: Theory and algorithms," NIPS2011, 2011.

[18] L. Breiman, "Bagging predictors," Machine Learning, vol.24, no.2, pp.123–140, 1996.

[19] L. Breiman, "Using adaptive bagging to debias regression," Technical Report 547, Statistics Dept. UCB, 1999.

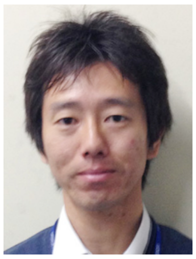
[20] T.-K. Kim and R. Cipolla, "MCBoost: Multiple classifier boosting for perceptual co-clustering of images and visual features," Advances in Neural Information Processing Systems (NIPS), Vancouver, Canada, Dec. 2008.

[21] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/

Yohei Mishina received the M.S. from Department of Computer Science, Chubu University in 2014. His research interests include computer vision, pattern recognition and machine learning.

Ryuei Murata received the B.S. from Department of Computer Science, Chubu University in 2014. The same year, he has been enrolled in the M.S. His research interests include computer vision, pattern recognition and machine learning. He is a member of the IEICE.

Yuji Yamauchi received the Ph.D. from Department of Computer Science, Chubu University in 2012. From 2012 to 2014 he was a post-doctoral fellow at the Chubu University. In 2011, he was a visiting student at Robotics Institute, Carnegie Mellon University. He was a Fellowship of the Japan Society for the Promotion of Science from 2010 to 2012. His research interests include computer vision and pattern recognition. He is a member of the IEEE, the IEICE and the IPSJ.



Takayoshi Yamashita received his Ph.D. degree from Department of Computer Science, Chubu University, Japan in 2011. He worked in OMRON Corporation from 2002 to 2014. He is a lecturer of the Department of Computer Science, Chubu University, Japan since 2014. He current research interests include object detection, object tracking, human activity understanding, pattern recognition and machine learning. He is a member of the IEEE, the IEICE and the IPSJ.

Hironobu Fujiyoshi received his Ph.D. in Electrical Engineering from Chubu University, Japan, in 1997. From 1997 to 2000 he was a post-doctoral fellow at the Robotics Institute of Carnegie Mellon University, Pittsburgh, PA, USA, working on the DARPA Video Surveillance and Monitoring (VSAM) effort and the humanoid vision project for the HONDA Humanoid Robot. He is now a professor of the Department of Robotics, Chubu University, Japan. From 2005 to 2006, he was a visiting researcher at Robotics Institute, Carnegie Mellon University. His research interests include computer vision, video understanding and pattern recognition. He is a member of the IEEE, the IEICE, and the IPSJ.