Efficient Feature Selection Method Using Contribution Ratio by Random Forest

Ryuei Murata*||, Yohei Mishina†||, Yuji Yamauchi‡||, Takayoshi Yamashita§|| and Hironobu Fujiyoshi¶||
*Email: mryua@vision.cs.chubu.ac.jp
†Email: mishi@vision.cs.chubu.ac.jp
‡Email: yuu@vision.cs.chubu.ac.jp
§Email: yamashita@cs.chubu.ac.jp
¶Email: hf@cs.chubu.ac.jp
||Chubu University, Kasugai-shi, Aichi, Japan

Abstract—**In the field of image recognition, a high-dimensional feature vector is often used to construct a classifier. This presents a problem, however, since using a large number of features can slow down training and degrade model readability. To alleviate this problem, sequential backward selection (SBS) has come to be used as a method for selecting an effective number of features for classification. However, as a type of wrapper method, SBS iteratively constructs and evaluates classifiers when selecting features, which is computationally intensive. In this study, we define the contribution ratio of features by random forest and use it to create an efficient feature selection method. We performed an evaluation experiment to compare the proposed method with SBS and found that the former could significantly reduce feature selection time for the same dimension reduction rate.**

I. Introduction

Advances in computer technology have made it possible to process massive amounts of data at high speed, and there is now a trend toward using such large amounts of data to construct classifiers in fields like computer vision and pattern recognition. A standard problem in image recognition, is that a high-dimensional feature vector is often used to construct a classifier despite the fact that some of those features do not contribute to classification ability. Using a large number of features can degrade learning speed and learning-model readability. Another problem is that a large amount of memory can be consumed during the classification process when using a high-dimensional feature vector thereby increasing computational costs. Against this background, a number of methods for selecting features have been proposed. Conventional feature selection methods can be divided into filtering that selects features on the basis of an evaluation function like entropy and wrapper methods that select features according to generalization error. Filters select features by an evaluation function using training samples, which makes for high-speed processing. This approach, however, requires the design of an optimal evaluation function for each target dataset. A wrapper method, on the other hand, first constructs a classifier from a subset of features and evaluates the classification ability of that classifier. It then changes the feature subset and constructs another classifier. For each classifier so constructed, the method evaluates its generalization ability. Finally, it selects the subset corresponding to the classifier with the highest generalization ability as the one composed of features with the highest classification ability. Since a wrapper method selects features acin acording to generalization ability, it is better than filtering in reducing the number of feature dimensions while maintaining classification ability. On the other hand, a wrapper method must construct a classifier for each subset, which can make the time required for feature selection huge. The purpose of the study presented here is to achieve efficient feature selection while maintaining classification ability without the need for iteratively constructing and evaluating classifiers as in a wrapper method. For the classifier of this study, we use a random forest [1], which has been attracting attention in fields such as computer vision and pattern recognition. Random forest is a learning algorithm that combines decision-tree learning and ensemble learning to construct a multitude of decision trees. It features high-speed learning robust to noise by also incorporating random learning. In this study, we define the degree of contribution that each feature in the constructed decision trees makes to classification ability as “contribution ratio” and reduce the number of feature dimensions based on the contribution ratios of the features in question. Contribution ratios can be calculated after constructing the classifier only once, which means that feature selection can be performed faster than a wrapper method that must repeatedly construct classifiers.

II. Existing feature selection methods

Selecting a combination of features with high classification ability improves classifier readability, shortens feature extraction time at classification time, and reduces the amount of memory consumed. Feature selection involves the following optimization problem: from among K features, select d features having high generalization ability and the highest dimension reduction rate. Feature selection methods can generally be divided into filtering and wrapper methods. Filtering is capable of high-speed processing since it selects features according to an evaluation function like entropy using training samples. On the other hand, it must select an optimal evaluation function tailored to each target dataset and learning model. A wrapper method starts out by constructing a classifier from a subset of features and evaluating its classification ability. It then changes the feature subset and constructs another classifier. For each classifier so constructed, it evaluates its generalization ability, and it selects the subset that constructs the classifier with the highest generalization ability as the one having features with the highest classification ability.

A. Wrapper methods

A wrapper method evaluates a constructed classifier on the basis of generalization ability and consequently has a high dimension reduction rate compared to filtering. Typical wrapper methods for solving this optimization problem are round robin, sequential forward selection (SFS), sequential backward selection (SBS), and plus-s minus-r. These methods are outlined below.

1) Round robin: The round robin technique, which is also called an exhaustive search, searches through all combinations of ${}_KC_d$ features. Among all feature selection methods, this is the only one that can guarantee an optimal solution, but computational complexity increases explosively for large values of K and d, the number of feature dimensions and number of selected features, respectively. As a result, the round robin method is not a practical approach when dealing with a large number of features.

2) Sequential forward selection: The SFS method proposed by Whitney [2] first constructs classifiers each with a different feature and selects the feature of the classifier with the highest classification rate as the first feature. It then constructs new classifiers by combining that feature with a feature not yet selected and treats the new feature that results in a classifier with the highest classification rate as the second feature. This type of sequential addition is repeated to search for the third feature and beyond until d features are found. The SFS method has the lowest computational cost among wrapper methods and has been used as an executable feature selection method.

3) Sequential backward selection: In contrast to SFS, the SBS method proposed by Marill [3] sequentially constructs classifiers for each subset of features achieved by removing one feature at a time from the previous set and evaluates those classifiers according to an evaluation value such as classification error rate. Once a classifier with the highest evaluation value has been constructed, the excluded feature is considered to make no contribution to classification ability and is therefore deleted. The flow of the SBS technique is shown in Fig.1.

4) Plus-s minus-r: The plus-s minus-r method proposed by Stearns [4] is an optimization method that uses the SFS and SBS methods in an alternate manner. It features the ability to delete a feature that has already been selected. First, given feature set X_k consisting of k-selected features, the technique applies SFS s times to Xk to select features and obtain the feature set X_{k+s}. Next, it applies SBS to reduce the number of features and obtain the feature set X_{k+s-r}. It then repeats this application of SFS and SBS until the number of features reaches the target value d. In general, $s > r$, so the plus-s minus-r technique may be used, for example, with $s = 2andr = 1$.

B. Problems with wrapper methods

A wrapper method selects features by evaluating classification ability, and can therefore select features in a stable manner without having to be optimized for each dataset or learning model as in filtering. On the other hand, a wrapper method must train as many classifiers as there are features when selecting features. In other words, classifiers must be repeatedly trained and evaluated in order to select features, which means that processing time can escalate as the number of feature dimensions increase.

Algorithm 1 Random Forest training algorithm

```
Require: Training samples$\{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}$;
         $\mathbf{x}_i \in \mathcal{X}, y_i \in \{1, 2, \ldots, M\}$
Init: Initialize sample weight $w_i$:
  $w_i^{(1)} \Leftarrow \frac{1}{N}$.
Run:
  for $t = 1 : T$ do
    Create subset $\mathcal{I}_t$ from training samples.
    $\Delta E_{max} \Leftarrow -\infty$.
    for $k = 1 : K$ do
      Select $f_k$ one dimensionally and randomly from features.
      for $h = 1 : H$ do
        Select threshold $\tau_h$ randomly.
        Use $f_k$ and $\tau_h$ to split sample-set $\mathcal{I}_n$ into $\mathcal{I}_l$ and $\mathcal{I}_r$.
        Calculate information gain $\Delta E$:
        $\Delta E = E(\mathcal{I}_n) - \frac{|\mathcal{I}_l|}{|\mathcal{I}_n|}E(\mathcal{I}_l) - \frac{|\mathcal{I}_r|}{|\mathcal{I}_n|}E(\mathcal{I}_r)$.
        if $\Delta E > \Delta E_{max}$ then
          $\Delta E_{max} \Leftarrow \Delta E$
        end if
      end for
    end for
    if $\Delta E_{max} = 0$ or max depth $D$ has been reached, then
      save class probability $P(c|l)$ in leaf node.
    else
      Continue splitting.
    end if
  end for
```

III. RANDOM FOREST

In this section, we describe the Random Forest method for feature selection using contribution ratio. Random Forest is a training algorithm for constructing a multiclass classifier having a multiple-decision-tree structure. It features a safeguard against overtraining by incorporating bootstrapping the same as in the “ bagging ” approach [5] as well as high-speed learning even for a high-dimensional feature vector by incorporating random feature selection [6]. Thanks to these features, the Random Forest method has come to be used in semantic segmentation [7], character recognition [8], object recognition [9],[10] and human pose estimation [11] in the field of computer vision.

A. Training

The Random Forest method creates subsets from training samples and constructs a classifier having a multiple-decision-tree structure. Each decision tree consists of split nodes and leaf nodes: split nodes are repeatedly constructed and leaf nodes are created when splitting can no longer be performed on the basis of a certain criterion. The training algorithm is shown in Algorithm 1. Here, T denotes number of trees and D the maximum tree depth. A subset is created from input consisting of training-sample set $\mathcal{I} = \{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_N, y_N\}, \mathbf{x}_i \in \mathcal{X}, y_i \in \{1, 2, \ldots, C\}$. It is selected randomly from training-sample set $\mathcal{I}$ with sample overlapping allowed. The algorithm constructs a decision tree using one subset. A split node in a decision tree saves a splitting function that uses feature f_k and threshold τ_h to split the sample either left or right. Based on the

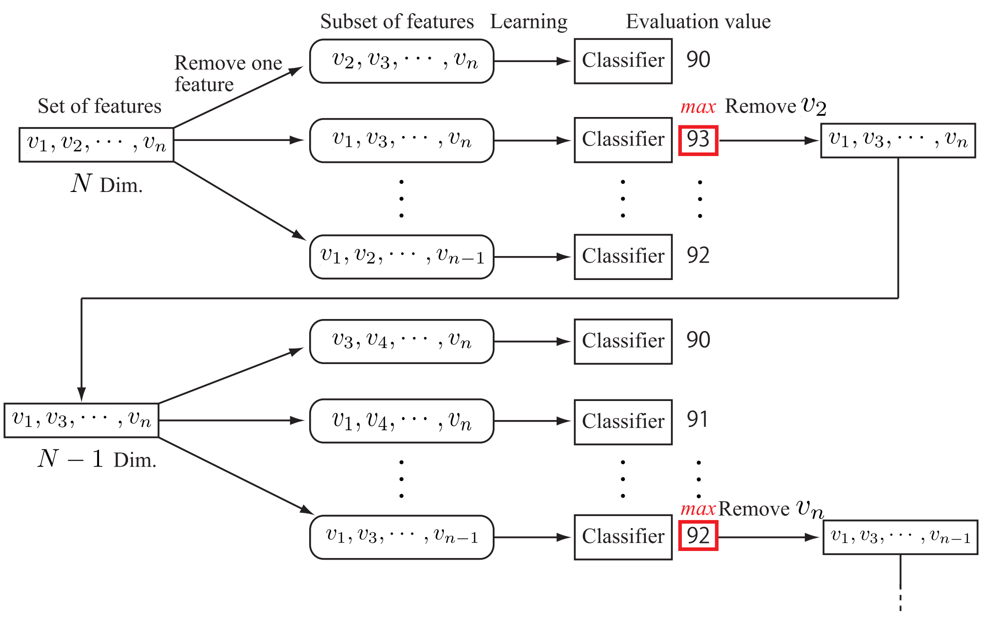


Fig. 1. Flow of the SBS method.

number of feature selections K and the number of threshold selections H, the splitting function selects a candidate randomly $K \times H$ times and settles on the candidate with the highest information gain ΔE. Sample set $\mathcal{I}_n$ that has arrived at a certain split node n branches into $\mathcal{I}_l$ and $\mathcal{I}_r$ according to candidate f_k and τ_h of the splitting function. Then, using $\mathcal{I}_l$ and $\mathcal{I}_r$, the algorithm calculates information gain ΔE by Eq. (1). Information gain is the result of subtracting from the entropy of the existing node the entropies of its child nodes. It represents the extent to which information gain has decreased by the splitting function. Small child-node entropy results in large information gain and a splitting function that can split a category well.

$$\Delta E = E(\mathcal{I}_n) - \frac{|\mathcal{I}_l|}{|\mathcal{I}_n|}E(\mathcal{I}_l) - \frac{|\mathcal{I}_r|}{|\mathcal{I}_n|}E(\mathcal{I}_r). \tag{1}$$

Here, function $E(I)$ represents information entropy calculated by Eq. (2).

$$E(\mathcal{I}) = -\sum_{i=1}^{n} p(c_i)\log p(c_i). \tag{2}$$

Here, $p(c_i)$ represents probability of class c_i determined by the occurrence rate of the teacher signal in the training sample. Repeating this process splits the sample set, and once the information gain becomes 0 or the process reaches maximum depth D, the algorithm creates leaf node l and calculates occurrence probability $P(c|l)$ of each category from the sample set that the process has arrived at. Each decision tree is constructed in this manner.

B. *Classification*

We here describe the Random Forest classification algorithm. The algorithm begins by inputting an unknown sample $\mathbf{x}$ into each of the decision trees created by the training algorithm. It then transverses each decision tree by branching left or right at each node by the splitting function and outputs class probability $P_t(c|\mathbf{x})$ saved at the leaf node arrived at. Then, as shown by Eq. (3), the algorithm calculates the average value of the outputs from all decision trees in the random forest.

$$P(c|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{x}). \tag{3}$$

Final output $\hat{y}$ obtained by Eq. (4) determines the class with the highest probability.

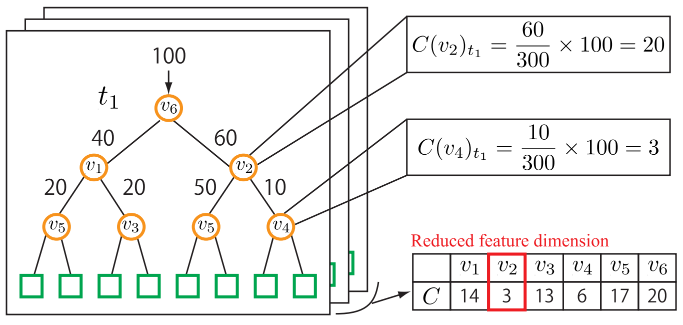
$$\hat{y} = \arg\max_{c} P(c|\mathbf{x}). \tag{4}$$

IV. Proposed Method

This study proposes efficient feature selection requiring no iterative construction and evaluation of classifiers. The proposed method selects features by using the degree to which each feature contributes to the classification ability of the constructed classifier (contribution ratio) as an evaluation criterion. Contribution ratios can be calculated once a random forest has been constructed, which means that feature selection can be performed faster than a wrapper method that repeatedly constructs classifiers. Here, we propose a sequential-calculation type and a batch-calculation type of feature selection using the contribution ratio concept. The sequential-calculation technique calculates feature contribution ratios after reducing the feature set by one dimension for the case that features are reduced one dimension at a time by backward selection. The batch-calculation technique, in contrast, calculates feature contribution ratios once a random forest has been constructed using all features. Features are then selected on the basis of those calculated contribution ratios.

A. *Calculation of contribution ratios*

The method for calculating contribution ratios is shown in Fig. 2. A feature dimension selected at a split node situated on an upper level of a decision tree splits many training samples and can therefore be considered to make a large contribution



$C(v_2)_{t_1} = \frac{60}{300} \times 100 = 20$
$C(v_4)_{t_1} = \frac{10}{300} \times 100 = 3$
100
t_1
v_6
40
60
v_1
v_2
20
20
50
10
v_5
v_3
v_5
v_4
Reduced feature dimension
v_1 v_2 v_3 v_4 v_5 v_6
C 14 3 13 6 17 20

Fig. 2. Reduced feature dimension.

Algorithm 2 Feature selection by sequential calculation of contribution ratios

```
Require: Number of trees : $T$
Run:
  for $k = 1 : K$ do
    Construct random forest using existing features
    for $i = 0 : I^{(k)}$ do
      Calculate contribution ratio $C(i)$ for each feature from the constructed random forest:
      $C(i) = \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_j^J S_{t,j}\delta[f_j, i]}{\sum_j^J S_{t,j}} \times 100$
    end for
    if the termination condition is not satisfied then
      delete the feature with the lowest contribution ratio.
    else
      Terminate feature selection.
    end if
  end for
```

to the classification ability of the classifier. With this in mind, we define contribution ratio $C(i)$ by random forest by Eq. (5) using the number of samples that have arrived at the split node in question.

$$C(i) = \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_j^J S_{t,j}\delta[f_j, i]}{\sum_j^J S_{t,j}} \times 100. \tag{5}$$

Here, $\delta[f_j, i]$ represents the delta function, which returns 1 if the dimension number selected at node j is the same as i and 0 otherwise. Accordingly, the numerator in the expression of Eq. (5) represents the total number of samples $S_{t,j}$ split at split node j at which feature dimension i was selected, and the denominator represents the total number of samples $S_{t,j}$ split at all split nodes in that decision tree. This equation therefore calculates the contribution ratio of that feature for each of the T constructed trees and calculates the average of those values to give the overall contribution ratio of that feature.

B. Feature selection by sequential calculation using contribution ratio

The sequential-calculation type of feature selection using contribution ratios is shown in Algorithm 2. The proposed method first constructs a random forest and calculates the contribution ratio of each feature. It then evaluates the constructed classifier and decides whether to terminate feature selection. If the condition for termination is not satisfied and feature-dimension reduction continues, the algorithm deletes the feature with the lowest contribution ratio. The flow of feature selection by sequential calculation is shown in Fig. 3(a).

Algorithm 3 Feature selection by batch calculation of contribution ratios

```
Require: Number of trees : $T$
Run:
  Construct random forest using all features.
  for $i = 0 : I^{(k)}$ do
    Calculate contribution ratio $C(i)$ of each feature from the constructed random forest:
    $C(i) = \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_j^J S_{t,j}\delta[f_j, i]}{\sum_j^J S_{t,j}} \times 100$
  end for
  for $k = 1 : K$ do
    if the termination condition is not satisfied then
      delete the feature with the lowest contribution ratio.
    else
      Terminate feature selection.
    end if
  end for
```

TABLE I. OUTLINE OF DATASETS.

Dataset Name	No.of Samples	No.of Classes	No.of Features
Pendigits	14988	10	16
Waveform	5000	3	21
Spambase	4601	2	57
Optdigits	5620	10	64

C. Feature selection by batch calculation using contribution ratio

Feature selection by batch calculation of contribution ratios is shown in Algorithm 3 and the flow of feature selection by batch calculation is shown in Fig. 3(b). This type of feature selection performs dimension reduction based on contribution ratios calculated from all features. Its computational cost is small compared to feature selection by sequential calculation since it does not have to calculate contribution ratios every time a random forest is constructed.

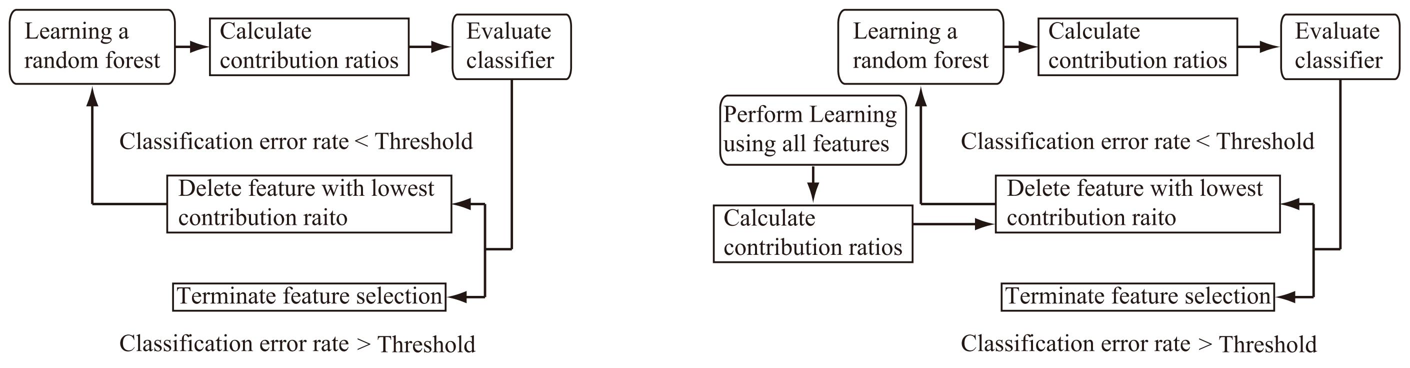
V. EVALUATION EXPERIMENT

We performed an experiment to demonstrate the effectiveness of the proposed method by comparing it with the SBS method. The condition for terminating the experiment was increase in the classification error rate by 10% compared to that at the time of training and evaluating the classifier using all features. The evaluation method measured generalization ability by 3-hold cross-validation.

1) Datasets: In this evaluation experiment, we used datasets from the UCI Machine Learning Repository [12], which is a collection of benchmark datasets released by the University of Californian at Irvine for assessing machine learning algorithms. We used, in particular, the Pendigits, Waveform, Spambase, and Optdigits datasets in this repository as listed in Table 1.

A. Training parameters

The random-forest training parameters used in this experiment are listed in Table 2. These parameters were used so that


Learning a random forest
Calculate contribution ratios
Evaluate classifier
Classification error rate < Threshold
Delete feature with lowest contribution raito
Terminate feature selection
Classification error rate > Threshold
Perform Learning using all features
Calculate contribution ratios

(a) Sequential calculation

(b) Batch calculation

Fig. 3. Flow of feature selection.

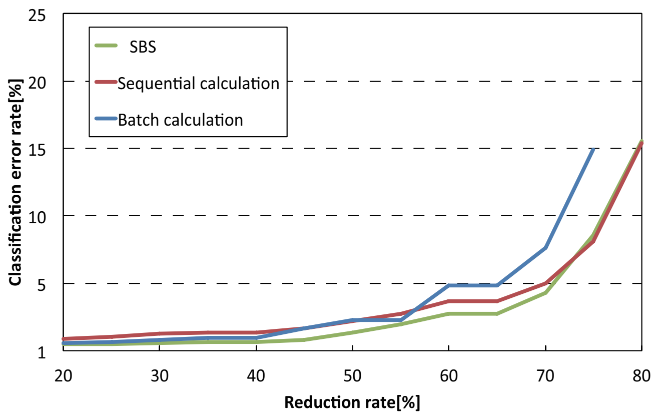


Fig. 4. Pendigits classification error rate versus reduction rate.

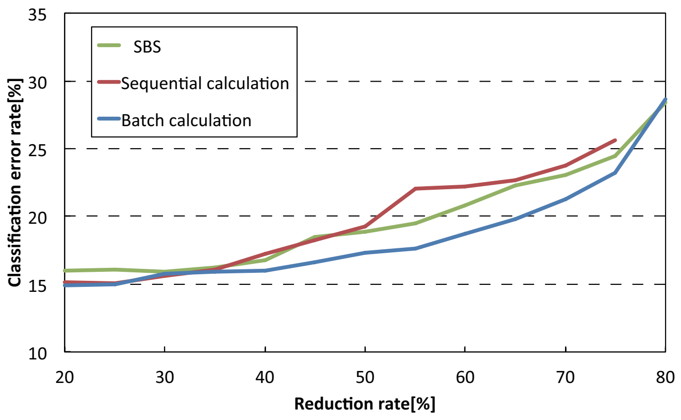


Fig. 5. Waveform classification error rate versus reduction rate.

error would be minimized in the results of classifier training and evaluation after changing tree depth and number of trees when using all features to 10 - 30 and 10 - 200, respectively, in ten rounds of feature selection. As for fixed parameters applied to all datasets, the number of times to perform feature selection was set to the square root of the number of feature dimensions, the threshold number of times to perform feature selection was set to 10, and the subset ratio was set to 1.0.

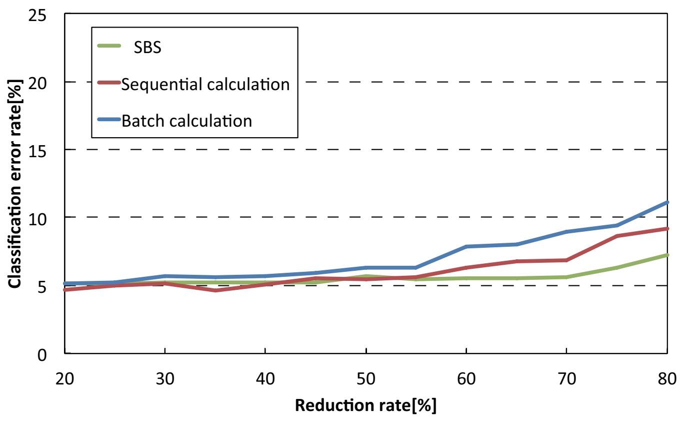


Fig. 6. Spambase classification error rate versus reduction rate.

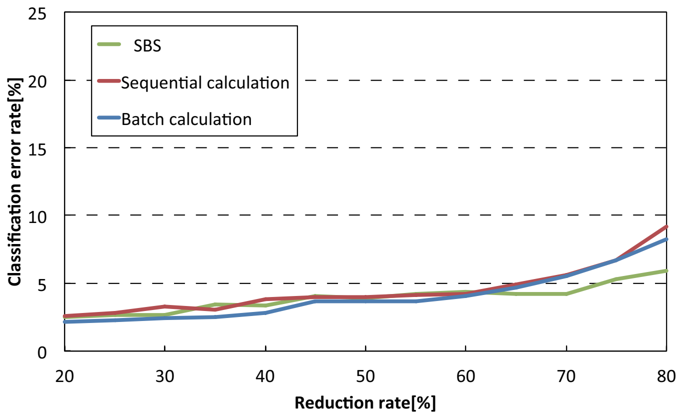


Fig. 7. Optdigits classification error rate versus reduction rate.

B. Evaluation results

The classification error rate versus dimension reduction rate for each dataset is shown in Figs. 4, 5, 6, and 7. These results show that the proposed method can perform feature selection at a classification error rate essentially equivalent to that of the SBS method for the same dimension reduction rate. Here, we consider that the sequential-calculation type of feature selection has a reduction rate higher than that of

TABLE II. Random-forest training parameters.

Dataset Name	Depth	No. of Trees
Pendigits	20	80
Waveform	30	150
Spambase	30	90
Optdigits	20	110

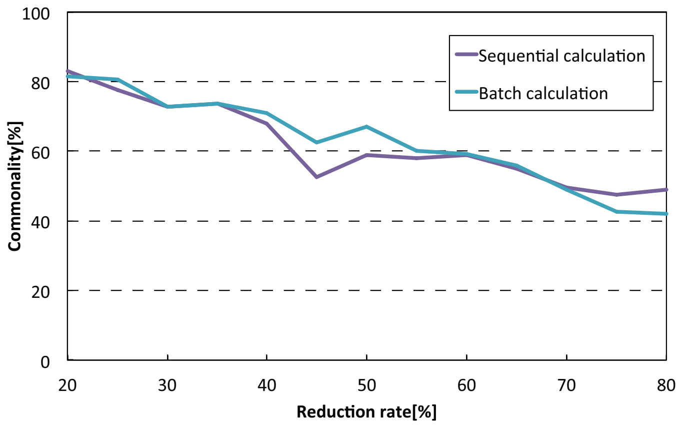


Fig. 8. Commonality in selected features between SBS and proposed method.

the lump-calculation type of feature selection since the former evaluates classifiers while taking fluctuation in features into account. Commonality in selected features between SBS and the proposed method is shown in Fig. 8. Although commonality between features selected by SBS and features selected by the proposed method drops for each feature reduction, it can be seen from these results that about 50% of features selected by the proposed method are the same as those selected by SBS at a reduction rate of 80%. The processing time for reducing the feature set by one dimension for each dataset is shown in Fig. 9. Feature selection time is significantly shortened by the proposed method compared to SBS. Since the SBS method must train as many classifiers as there are features, N number of features means that a classifier must be constructed $\mathcal{O}(N)$ times. In contrast, the proposed method can calculate contribution ratios from one classifier, which means that a classifier must be constructed $\mathcal{O}(1)$ times thereby shortening the time required for feature selection. In other words, the difference in feature selection time between SBS and the proposed method becomes large for datasets with a large number of features.

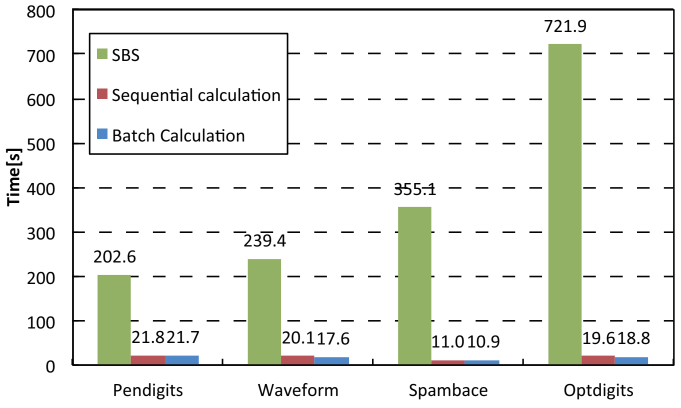


Fig. 9. Time required for feature selection.

VI. Conclusion

In this paper, we defined the contribution that each feature makes to classification ability as “contribution ratio” and proposed a method of selecting features based on contribution ratio. The results of an evaluation experiment showed that the proposed method achieves a feature-dimension reduction rate equivalent to that of the conventional SBS method. Furthermore, compared to SBS that must construct $\mathcal{O}(N)$ number of classifiers for N number of feature dimensions, the proposed method constructs a classifier only $\mathcal{O}(1)$ times, which significantly shortens the time required for feature selection. In future research, we plan to study a method for selecting an optimal combination of features by using a decision-tree structure and evaluating co-occurrence among features.

References

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

[2] A. W. Whitney, “A direct method of nonparametric measurement selection,” *IEEE Trans. Comput.*, vol. 20, no. 9, pp. 1100–1103, Sep. 1971. [Online]. Available: http://dx.doi.org/10.1109/T-C.1971.223410

[3] T. Marill and D. Green, “On the effectiveness of receptors in recognition systems,” *IEEE Trans. Inf. Theor.*, vol. 9, no. 1, pp. 11–17, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1963.1057810

[4] S. D. Stearns, “On selecting features for pattern classifiers.” in *Proceedings of the 3rd International Conference on Pattern Recognition (ICPR 1976)*, Coronado, CA, 1976, pp. 71–75.

[5] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1023/A:1018054314350

[6] T. K. Ho, “The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

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

[8] Y. Amit and D. Geman, “Shape quantization and recognition with randomized trees,” *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.7.1545

[9] V. Lepetit and P. Fua, “Keypoint recognition using randomized trees,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 9, pp. 1465–1479, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2006.188

[10] J. Gall, A. Yao, N. Razavi, L. V. Gool, and V. Lempitsky, “Hough forests for object detection, tracking, and action recognition,” *Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[11] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, “Real-time human pose recognition in parts from single depth images,” *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

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