Neurocomputing 73 (2010) 3273–3283

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

# A new wrapper feature selection approach using neural network

Md. Monirul Kabir [a], Md. Monirul Islam [b], Kazuyuki Murase [c,*]

[a] Department of System Design Engineering, University of Fukui, Fukui 910-8507, Japan
[b] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka 1000, Bangladesh
[c] Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, and Research and Education Program for Life Science, University of Fukui, Fukui 910-8507, Japan

ARTICLE INFO

*Article history:*
Received 5 December 2008
Received in revised form
9 November 2009
Accepted 2 April 2010
Communicated by M.T. Manry
Available online 21 May 2010

*Keywords:*
Feature selection
Wrapper approach
Neural networks
Correlation information

ABSTRACT

This paper presents a new feature selection (FS) algorithm based on the wrapper approach using neural networks (NNs). The vital aspect of this algorithm is the automatic determination of NN architectures during the FS process. Our algorithm uses a constructive approach involving correlation information in selecting features and determining NN architectures. We call this algorithm as constructive approach for FS (CAFS). The aim of using correlation information in CAFS is to encourage the search strategy for selecting less correlated (distinct) features if they enhance accuracy of NNs. Such an encouragement will reduce redundancy of information resulting in compact NN architectures. We evaluate the performance of CAFS on eight benchmark classification problems. The experimental results show the essence of CAFS in selecting features with compact NN architectures.



## 1. Introduction

Feature selection (FS) is a search process or technique used to select a subset of features for building robust learning models, such as neural networks and decision trees. Some irrelevant and/or redundant features generally exist in the learning data that not only make learning harder, but also degrade generalization performance of learned models. For a given classification task, the problem of FS can be described as follows: given the original set, $G$, of $N$ features, find a subset $F$ consisting of $N'$ relevant features where $F \subset G$ and $N' < N$. The aim of selecting $F$ is to maximize the classification accuracy in building learning models. The selection of relevant features is important in the sense that the generalization performance of learning models is greatly dependent on the selected features [8,33,42,46]. Moreover, FS assists for visualizing and understanding the data, reducing storage requirements, reducing training times, and so on [7].

A number of FS approaches exists that can be broadly classified into three categories: the filter approach, the wrapper approach, and the hybrid approach [21]. The filter approach relies on the characteristics of the learning data and selects a subset of features without involving any learning model [5,11,22,26]. In contrast, the wrapper approach requires one predetermined learning model and selects features with the aim of improving the generalization performance of that particular learning model [9,12,26]. Although the wrapper approach is computationally expensive than the filter approach, the generalization performance of the former approach is better than the later approach [7]. The hybrid approach attempts to take advantage of the filter and wrapper approaches by exploiting their complementary strengths [13,43].

In this paper we propose a new algorithm, called constructive approach for FS (CAFS) based on the concept of the wrapper approach and sequential search strategy. As a learning model, CAFS employs a typical three layered feed-forward neural network (NN). The proposed technique combines the FS with the architecture determination of the NN. It uses a constructive approach involving correlation information in selecting features and determining network architecture. The proposed FS technique differs from previous works (e.g., [6,9,11–13,22,43,46]) on two aspects.

First, CAFS emphasizes not only on selecting relevant features for building learning models (i.e., NNs), but also on determining appropriate architectures for the NNs. The proposed CAFS selects relevant features and the NN architectures simultaneously using a constructive approach. This approach is quite different from existing work [1,8,11,12,22,26,35,42,46]. The most common practice is to choose the number of hidden neurons in the NN randomly, and then selects relevant features for the NN automatically. Although the automatic selection of relevant features is a good step in improving the generalization performance, the random selection of hidden neurons affects the generalization performance of NNs. It is well known that the performance of any NN is greatly dependent on its architecture [14,15,18,19,23,39,48]. Thus determining both hidden neurons'

* Corresponding author. Tel: +81 0 776 27 8774; fax: +81 0 776 27 8420.
*E-mail address:* murase@synapse.his.u-fukui.ac.jp (K. Murase).

0925-2312/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.04.003

number and relevant features automatically provide a novel approach in building learning models using NNs.

Second, CAFS uses correlation information in conjunction with the constructive approach for selecting a subset of relevant features. The aim of using correlation information is to guide the search process. It is important here to note that CAFS selects features when they reduce the error of the NN. The existing wrapper approaches do not use correlation information to guide the FS process (e.g. [1,8,12,31]). Thus they may select correlated features resulting in increase of redundant information among selected features. Consequently, it will increase the computational requirements of NN classifiers.

The rest of this paper is organized as follows. Section 2 describes some related works about FS. Our proposed CAFS is discussed elaborately in Section 3. Section 4 presents the results of our experimental studies including the experimental methodology, experimental results, the comparison with other existing FS algorithms, and the computational complexity of different stages. Finally, Section 5 concludes the paper with a brief summary and a few remarks.

## 2. Previous work

The wrapper based FS approach has received a lot of attention due to its better generalization performance. For a given dataset $G$ with $N$ features, the wrapper approach starts from a given subset $F_0$ (an empty set, a full set, or any randomly selected subset) and searches through the feature space using a particular search strategy. It evaluates each generated subset $F_i$ by applying a learning model to the data with $F_i$. If the performance of the learning model with $F_i$ is found better, $F_i$ is regarded as the current best subset. The wrapper approach then modifies $F_i$ by adding or deleting features to or from $F_i$ and the search iterates until a predefined stopping criterion is reached. As CAFS uses the wrapper approach in selecting features for NNs, this section briefly describes some FS algorithms based on the wrapper approach using NNs. In addition, some recent works that do not use NN and/or concern with the wrapper approach are also described.

There are a number of ways by which one can generate subsets and progress the search process. For example, search may start with an empty set and successively add features (e.g. [9,31,34]), called sequential forward selection (SFS), or a full set and successively remove features [1,8,12,38,42], called sequential backward selection (SBS). In addition, search may start with both ends in which one can add and remove features simultaneously [3], called bidirectional selection. Alternatively, a search process may also start with a randomly selected subset (e.g. [25,44]) involving a sequential or bidirectional strategy. The selection of features involving a sequential strategy is simple to implement and fast. However, the problem of such a strategy is that once a feature is added (or deleted) it cannot be deleted (or added) latter. This is called *nesting effect* [33], a problem may encounter by SFS and SBS. In order to overcome this problem, the floating search strategy [33] that can re-select the deleted features or can delete the already-added features is effective. The performance of this strategy has been found to be very good compared with other search strategies and, furthermore, the floating search strategy is computationally much more efficient than a FS method branch and bound [30].

In some studies (e.g., [42,46]), the goodness of a feature is computed directly as the value of a loss function. The cross entropy with a penalty function is used in these algorithms as a loss function. In [42], the penalty function encourages small weights to converge to zero or prevents the weights to converge to large values. On the other hand, in [46], the penalty function forces a network to keep the derivative of its neurons' transfer functions from becoming low. The aim of such a restriction is to reduce output sensitivity to the input changes. In practice, these approaches need extra computation and time to converge as they augment the cost function by an additional term. In another study [45], three FS algorithms are proposed for multilayer NNs and multiclass support vector machines, using mutual information between class labels and classifier outputs as an objective function. These algorithms employ backward elimination and direct search in selecting a subset of salient features. As the backward search strategy is involved in these algorithms, the searching time might be longer specifically for high dimensional datasets [20].

Pal and Chintalapudi [35] proposed a novel FS technique in which each feature was multiplied by an attenuation function prior to its entry for training an NN. The parameters of the attenuation function were learned by the back-propagation (BP) learning algorithm, and their values varied between zero and one. After training, the irrelevant and salient features can be distinguished by the value of attenuation function that is close to zero and one, respectively. On basis of this FS technique, a number of algorithms have been proposed that combines fuzzy logic and genetic programming in selecting relevant features for NNs [4] and decision trees [27], respectively. In addition, Rakotomamonjy [37] proposed new FS criteria derived from Support Vector Machines and were based on the generalization error bounds sensitivity with respect to a feature. The effectiveness of these criteria was tested on several problems.

Ant colony optimization, a meta-heuristic approach is also used in some studies (e.g. [2, 43]) for selecting salient features. A number of artificial ants are used to iteratively construct feature subsets. In each iteration, an ant would deposit a certain amount of pheromone proportional to the quality of the feature subset in solving a given dataset. In [43], the feature subset is evaluated using the error of an NN, it is evaluated using mutual information is used in [2].

A FS technique, called automatic discoverer of higher order correlations (ADHOC) [40], has been proposed that consists of two steps. In the first step, ADHOC identifies spurious features by constructing a profile for each feature. In the second step, it uses genetic algorithms to find a subset of salient features. Some other studies (e.g. [13, 32]) also use genetic algorithms in FS. In [13], a hybrid approach for FS has been proposed that incorporates the filter and wrapper approaches in a cooperative manner. The filter approach involving mutual information is used here as local search to rank features. The wrapper approach involving genetic algorithms is used here as global search to find a subset of salient features from the ranked features. In [32], two basic operations, i.e., deletion and addition are incorporated that seek the least significant and most significant features for making stronger local search during FS.

In [47], a FS algorithm has been proposed which first converts a $C$ class classification problem in $C$ two-class classification problems. It means the examples in a training set are divided into two classes (say, $C_1$ and $C_2$). For finding feature subset of each binary classification problem, the FS algorithm then integrates features in SFS manner for training a support vector machine.

Most of the FS algorithms try to find a subset of salient features by measuring contribution of features, while others use a penalty term in the objective function. In addition, a sort of heuristics functions in conjunction with ant colony optimization algorithms, fuzzy logic, genetic algorithms or genetic programming are used to find a subset of salient features. The main problem in most of these algorithms is that they do not optimize the configuration of the classifier in selecting features. This issue is important in the sense that it affects the generalization ability of NN classifiers.

## 3. A constructive approach for feature selection (CAFS)

The proposed CAFS uses a wrapper approach in association with incremental training to find a feature subset from a set of available features. In order to achieve a good generalization performance of a learning model (i.e., NN), CAFS determines automatically the number of hidden neurons of the NN during the FS process. The same incremental approach is used in determining hidden neurons. The proposed technique starts with a minimum number of features and hidden neurons. It then adds features and hidden neurons in an incremental fashion one by one. The proposed CAFS uses one simple criterion to decide when to add features or hidden neurons.

The major steps of CAFS are summarized in Fig. 1, which are explained further as follows:

*Step* 1: Divide the original feature set into two different subsets or groups equally as shown in Fig. 2. The most correlated $N/2$ features are put in one group called *similar group S*, while the least correlated $N/2$ features in the other group called *dissimilar group D*. Here $N$ is the total number of features in the original feature set.

*Step* 2: Choose a three layered feed-forward NN architecture and initialize its different parameters. Specifically, the number of neurons in the input and hidden layers is set to two and one, respectively. The number of output neurons is set to the number of categories in the given dataset. Randomly initialize all connection weights of the NN within a small range.

*Step* 3: Select one feature from the $S$ or $D$ group according to feature addition criterion. Initially, CAFS selects two features: one from the $S$ group and one from the $D$ group. The selected feature is the least correlated one among all the features in the $S$ or $D$ group.

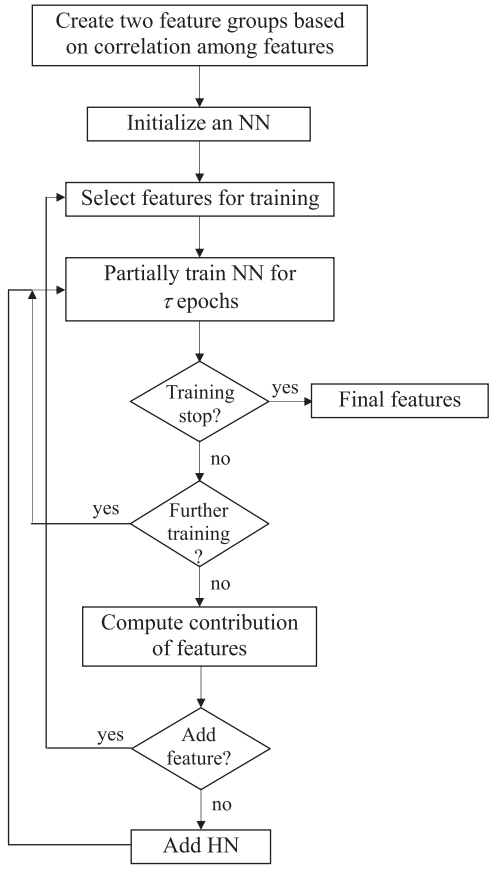


**Fig. 1.** Flowchart of CAFS. NN and HN refer to neural network and hidden neuron, respectively.

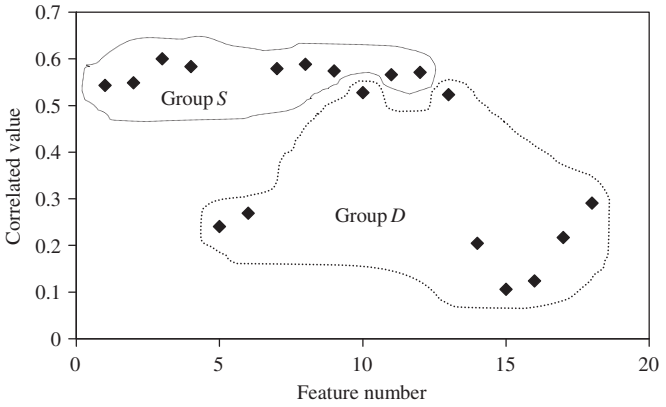


**Fig. 2.** Scenario of the grouping of features for vehicle dataset. Here, $S$ and $D$ groups contain similar and dissimilar features, respectively.

*Step* 4: Partially train the NN on the training set for a certain number of training epochs using the BP learning algorithm [41]. The number of training epochs,$\tau$, is specified by the user. Partial training, which was first used in conjunction with an evolutionary algorithm [48], means that the NN is trained for a fixed number of epochs regardless whether it has converged or not.

*Step* 5: Check the termination criterion to stop the training and FS process in building a NN. If this criterion is satisfied, the current NN architecture with the selected features is the outcome of CAFS for a given dataset. Otherwise continue. In this work, the error of the NN on the validation set is used in the termination criterion. The error, $E$, is calculated as

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{c=1}^{C}(o_c(p) - t_c(p))^2 \tag{1}$$

where $o_c(p)$ and $t_c(p)$, respectively, are the actual and target responses of the $c$th output neuron for the validation pattern $p$. The symbols $P$ and $C$ represent the total number of validation patterns and of output neurons, respectively.

*Step* 6: Check the training progress to determine whether further training is necessary. If the training error reduces by a predefined amount, $\varepsilon$, after the training epochs $\tau$, it is assumed that the training process is progressing well; thus further training is necessary and go to the Step 4. Otherwise, go to the next step for adding a feature or hidden neuron. The reduction of training error can be described as

$$E(t) - E(t+\tau) > \varepsilon, \quad t = \tau, 2\tau, 3\tau, \ldots \tag{2}$$

*Step* 7: Compute the contribution of the previously added feature in the NN. The proposed CAFS computes the contribution based on the classification accuracy ($CA$) of the validation set. The $CA$ can be calculated as

$$CA = 100\left(\frac{P_{vc}}{P_v}\right) \tag{3}$$

where $P_{vc}$ refers to the number of validation patterns correctly classified by the NN and $P_v$ is the total number of patterns in the validation set.

*Step* 8: Check the criterion for adding a feature in the NN. If the criterion is satisfied, go to Step 3 for adding one feature according to the FS criterion. Otherwise continue.

*Step* 9: Since the criterion for further training and feature addition is not satisfied, the performance of the NN can only be improved by increasing its information processing capability. The proposed CAFS, therefore, adds one hidden neuron to the NN and go to Step 4 for training.

It is now clear that the idea behind CAFS is straightforward, i.e., minimize the training error and maximize the validation

accuracy. To achieve this objective, CAFS uses a constructive approach to find a salient feature subset and to determine NN architectures automatically. Although other approaches such as pruning [39] and regularization [10] could be used in CAFS, the selection of an initial NN architecture in these approaches is difficult [18,19]. It is, however, very easy in case of the constructive approach. For example, the initial network architecture in the constructive approach can be consisted of an input layer with one feature, a hidden layer with one neuron and an output layer with $C$ neurons, one neuron for each class. If this minimal architecture cannot solve the given task, features and hidden neurons can be added one by one. Due to the simplicity of initialization, the constructive approach is used widely in multi-objective learning tasks [19,24]. The following section gives more details about different components of our proposed algorithm.

### 3.1. Grouping of features

Grouping of features is actually a division of features into groups of similar objects. Different practitioners used to entitle the task of grouping by clustering. In [28], it is found that solving the feature selection task by clustering algorithm requires large number of groups of original features. It can be done by dividing the input features with assigning user-specified thresholds. On the other hand, proper consciousness is necessary for handling the multiple grouping of features, otherwise, superiority might fall.

The aim of grouping in CAFS is to find relationships between features so that the algorithm can select distinct and informative features for building robust learning models. Correlation is one of the most common and useful statistics that describes the degree of relationship between two variables. A number of criteria have been proposed in statistics to estimate correlation. In this work, CAFS uses the best known *Pearson product–moment correlation coefficient* to measure correlation between different features of a given training set. The correlation coefficient $r_{ij}$ between two features $i$ and $j$ is

$$r_{ij}=\frac{\sum_p(x_i-\bar{x}_i)(x_j-\bar{x}_j)}{\sqrt{\sum_p(x_i-\bar{x}_i)^2\sqrt{\sum_p(x_j-\bar{x}_j)^2}}} \tag{4}$$

where $x_i$ and $x_j$ are the value of features $i$ and $j$, respectively. The variables $\bar{x}_i$ and $\bar{x}_j$ represent the mean values of $x_i$ and $x_j$, averaged over $p$ examples. If the features $i$ and $j$ are completely correlated, i.e., exact linear dependency exist, then $r_{ij}$ would be 1 or $-1$. If $i$ and $j$ are completely uncorrelated then $r_{ij}$ would be 0. After computing the correlation coefficient for all possible combinations of features, CAFS computes the correlation of each feature and arranges features in descending order. The correlation, $Cor_i$, of any feature $i$ is

$$Cor_i=\frac{\sum_{j=1}^{N}|r_{ij}|}{N-1} \quad \text{if} \quad i\neq j \tag{5}$$

where $N$ is the number of features used in representing a given dataset. Finally, CAFS creates two groups. One group contains the first $N/2$ features and we call it *similar* ($S$) *group*, while the other group contains the remaining $N/2$ features and we call it as *dissimilar* ($D$) *group*. The first feature in the $S$ group and the last feature in the $D$ are the most and least correlated features in the dataset.

### 3.2. Termination of NN training

Since CAFS adds features and hidden neurons one by one during the training process of a NN, the training error would reduce as the training process progresses. However, the objective of CAFS is to improve generalization ability of the NN. This means the training error may not be a right choice to be used for terminating the training process of the NN. Generally, a separate dataset, called the validation set, is widely used for termination. It is assumed that the validation error gives an unbiased estimate because the validation data are not used for modifying the weights of the NN.

In order to achieve good generalization ability, CAFS uses validation error in its termination criterion. This criterion is very simple and straightforward. CAFS measures validation error after every $\tau$ epochs of training, called *strip*. It terminates training when the validation error increases by a predefined amount ($\lambda$) for $T$ successive times, which are measured at the end of each of $T$ successive strips [36]. Since the validation increases not only once but $T$ successive times, it can be assumed that such increases indicate the beginning of the *final over fitting* not just the *intermittent*. The termination criterion can be expressed as

$$E(\tau+i)-E(\tau)>\lambda, \quad i=1,2,\ldots,T \tag{6}$$

where $\tau$ and $T$ are positive integer number specified by the user. Our algorithm, CAFS, tests the termination criterion after every $\tau$ epochs of training and stops training when the condition described by Eq. (6) is satisfied. On the other hand, if the condition is not satisfied even though the required number of features and hidden neurons has already been added to the NN then CAFS tests the $CA$ of NN on validation set for some $\tau$ epochs according to Eq. (3). If it does not increase significantly, then the training is stopped automatically. In this work, the value of $T$ is chosen as 3.

### 3.3. Feature addition and selection

The proposed CAFS uses a straightforward criterion for deciding when to add a feature to an existing NN. It employs a simple criterion for such additions. This criterion decides the addition of features based on their $CA$ on the validation set. The reason for using the validation data also here is to emphasis on selecting salient features for achieving good generalization. The $CA$ can be calculated according to Eq. (3) and the corresponding feature addition criterion can be described as

$$CA(t+\tau)>CA(t), \quad t=\tau,2\tau,3\tau,\ldots. \tag{7}$$

Our CAFS tests the feature addition criterion after every $\tau$ epochs and adds one feature to the feature subset if the criterion is satisfied. The most important point in the feature addition process is the selection so that the added features can improve accuracy of classifiers. As we have already seen, CAFS has divided the original features into $S$ and $D$ groups based on their correlation. The main philosophy of CAFS is to find distinct and informative features to reduce redundancy in learning. To achieve this goal, CAFS puts emphasis on the features of the $D$ group because they are less correlated than those of the $S$ group. The way CAFS selects features from two groups in every feature addition step can be described as follows.

The proposed algorithm first adds the most distinct (i.e., least correlated) feature from the available features in the $D$ group and then trains the network for a certain number of epochs. It considers the addition of a feature successful, when such an addition increases accuracy of the network on the validation set. CAFS attempts to add the second most distinct feature from the $D$ group and this process repeats until the $D$ group is empty. For further addition, CAFS adds features from the $S$ group in the same manner. Such a selection process is called here as the "regular selection".

In course of regular selection, when adding a feature from the $D$ group fails to improve accuracy of the network, CAFS tries to improve accuracy by increasing the network processing power. The proposed algorithm, therefore, adds a hidden neuron to the existing network architecture and trains the modified architecture for a certain number of $\tau$ epochs. If the network accuracy

does not improve yet, the added feature is considered irrelevant and CAFS restores the previous network. To improve accuracy of the network, CAFS adds the most distinct feature that is available in the $S$ group and trains the network for a certain number of epochs. On the other hand, since the addition of a feature from the $D$ group does not improve accuracy, it is reasonable to think that the distinct nature of features is not sufficient and the network may need some general information about a given learning problem. To achieve this objective, CAFS adds a feature from the $S$ group that maintains more correlation than any feature from the $D$ group. The feature from the $S$ group therefore can be considered as the source of providing general information about the problem. It should be kept in mind that when the addition of a feature from the $S$ group is found successful, CAFS adds the next feature from the $D$ group because the proposed technique emphasis on the distinct nature of features. On the other hand, if the addition of the feature from the $S$ group is found unsuccessful, it is assumed that the network still requires the distinct informative features about the dataset. Thus CAFS adds the next feature from the $D$ group with the aim of improving accuracy of the network. We call such a selection process here as the "irregular selection". In order to complete the FS process, CAFS proceeds by following the regular selection and irregular selection strategies simultaneously until all elements of the $S$ and $D$ groups are empty.

### 3.4. Computational complexity

Rigorous analysis of computational complexity helps to understand the actual computational cost of an algorithm. As Kudo and Sklansky [17] showed such an analysis in the form of big-O notation, we are inspired to compute the computational cost of our CAFS. The following few paragraphs present the computational complexity of CAFS to show that the inclusion of different techniques does not increase computational complexity of training NNs.

(i). *Correlation computation*: In this work, we use *Pearson product–moment correlation coefficient technique* to compute correlation using Eq. (5). If the total features of a given learning problem is $N$, the cost of computing correlation is $O(N^2 \times P_t)$, where $P_t$ denotes the number of examples in the training set. It is important here to note that this cost requires only once, i.e., before starting the training process of an NN.

(ii). *Partial training*: We use standard BP [41] for training. Each epochs of BP takes $O(W)$ computations for training one example. Here, $W$ is the number of weights in the current NN. Thus training all examples in the training set for $\tau$ epochs needs $O(\tau \times P_t \times W)$ computations.

(iii). *Termination criterion*: The termination criterion employed in CAFS for stopping training of the NN uses both training and validation errors. Since the training error is computed as a part of the training process, the termination criterion takes $O(P_v \times W)$ computations, where $P_v$ denotes the number of examples in the validation set. Since $P_v < P_t$, $O(P_v \times W) < O(\tau \times P_t \times W)$.

(iv). *Further training*: Our CAFS uses Eq. (2) to check whether further training for the added feature is necessary. The evaluation of Eq. (2) takes a constant computation $O(1)$, since the error values used in Eq. (2) have already evaluated during training.

(v). *Contribution computation*: CAFS computes the contribution of the added feature using Eq. (3). This computation takes $O(P_v)$ operations, which is less than $O(\tau \times P_t \times W)$.

(vi). *Feature addition criterion*: CAFS uses Eq. (7) for determining whether a feature is to be added. Since CAFS evaluates the accuracy of the validation set in the previous step, the evaluation of Eq. (7) involving such accuracy requires constant computation $O(1)$.

(vii). *Adding a feature*: This operation takes $O(h)$ computations for initializing the connection weights of the newly added feature, where $h$ is number of hidden neurons in the current network and $O(h) \ll O(\tau \times P_t \times W)$.

(viii). *Adding a hidden neuron*: The computational cost for adding a hidden neuron is $O(N_1+C)$ for initializing its connection weights, where $N_1$ is the number of added input features and $C$ is the number neurons in the output layer. It is also noted that $O(N_1+C) < O(\tau \times P_t \times W)$.

All the above mentioned computation is done for a partial training consisting of $\tau$ epochs. In general, CAFS needs several, say $M$, such partial trainings. Thus the total computational cost of CAFS for training a total of $T$ epochs ($T=\tau \times M$) is $O(N^2 \times P_t)+O(\tau \times M \times P_t \times W)$. However, in practice, the first term, i.e., $N^2 \times P_t$ is much less than the second one. Hence the total computational cost CAFS is $O(\tau \times M \times P_t \times W)$, which is same for training a fixed network architecture using BP [41]. It is clear that the incorporation of several techniques in CAFS does not increase its computational cost.

## 4. Experimental studies

This section presents CAFS's performance on eight well-known benchmark classification datasets. The datasets that have been used to evaluate the performance of CAFS are diabetes, breast cancer, glass, vehicle, hepatitis, horse colic, ionosphere, and splice junction. Detailed descriptions of these datasets are available at UCI machine learning repository. The characteristics of the datasets and their partitions are summarized in Table 1, which shows a considerable diversity in the number of examples, features, and classes among datasets. The above mentioned datasets were used widely in many previous studies, and they represent some of the most challenging datasets in the NN and machine learning [36,48]. Experimental details, results, the roles of architecture determination and grouping of features in CAFS and finally the comparisons with other works are described in this context.

### 4.1. Experimental setup

In this study, we follow the benchmarking methodology and suggestions in doing experiments. All datasets were partitioned into three sets: a training set, a validation set and a testing set. The number of examples in these sets is shown in the Table 1. The training and testing sets were used to train NNs and to evaluate the classification accuracy of trained NNs, respectively. The validation set was used to stop the training process of CAFS. It has been known that the experimental results may vary significantly for different partitions of the same data collection, even when the number of examples in each set is the same [36]. It is necessary to know precise specification of the partition in order to replicate an experiment or conduct fair comparisons. For all datasets, the first $P_1$ examples were used for the training set, the following $P_2$ examples for the validation set, and the final $P_3$ examples for the testing set. It should be kept in mind that such partitions do not represent the optimal one in practice.

We preprocessed the datasets found from UCI database by rescaling input attribute values between 0 and 1 with a linear function. The outputs were encoded by the 1-of-$c$ representation of $c$ classes. The most commonly used winner-takes-all method was used for selecting the NN output. One bias neuron with a fixed input of 1 was used for the hidden and output layers.

The hidden and output neuron functions were defined by the logistic sigmoid function $\phi(x)=1/(1+\exp(-x))$.

There are some parameters in CAFS which need to be specified by the user. These are described as follows. The initial connection weights for an NN were randomly chosen in the range between $-1.0$ and 1.0. The learning rate and momentum term for training of NN were chosen as 0.1–0.15 and 0.8–0.9, respectively. The number of partial training epochs ($\tau$) of NN was chosen between 5 and 70. The training error threshold value ($\varepsilon$) for diabetes, cancer, glass, vehicle, hepatitis, horse, ionosphere, and splice datasets was set to 0.003, 0.0001, 0.13, 0.04, 0.035, 0.001, 0.008, and 0.02, respectively. In addition, the validation error threshold value ($\lambda$) for diabetes, glass, vehicle, and ionosphere was set to 0.0 while 0.001, 0.0045, 0.02, and 0.0008 were set for cancer, hepatitis, horse, and splice datasets, respectively. The initial weight values, learning rate, and momentum term are the parameters of the well known BP algorithm [41]. According to the suggestions of many previous studies [8,15,16,36] and after some preliminary runs these values were set. They were not meant to be optimal. We conducted two sets of experiments to investigate the essence of FS. In one set, we used CAFS that selects salient feature subset and hidden neurons during the learning process of a given dataset. We call this set of experiment as CAFS with FS. In the other set, the FS part of CAFS was not used. We call this set of experiment as CAFS without FS.

**Table 1**
Characteristics and partitions of different benchmark classification datasets.

| Dataset | Number of | | | | |
|---|---|---|---|---|---|
| | Features | Output classes | Training examples | Validation examples | Testing examples |
| Diabetes | 8 | 2 | 384 | 192 | 192 |
| Cancer | 9 | 2 | 349 | 175 | 175 |
| Glass | 9 | 6 | 108 | 53 | 53 |
| Vehicle | 18 | 4 | 424 | 211 | 211 |
| Hepatitis | 19 | 2 | 77 | 39 | 39 |
| Horse | 21 | 2 | 172 | 86 | 86 |
| Ionosphere | 34 | 2 | 175 | 88 | 88 |
| Splice | 60 | 3 | 1584 | 793 | 793 |

### 4.2. Experimental results

Tables 2 and 3 show the results of CAFS over 30 independent runs on eight classification datasets and Fig. 3 shows FS and architecture determination process for the glass dataset. The classification accuracy (*CA*) in Table 2 refers to the percentage of exact classifications produced by trained NNs on the testing set of a classification dataset. The average run time, i.e., CPU time (Table 3) for each dataset includes the time taken for computing the correlation of each feature, training the NN with adding features and hidden neurons, and computing contribution of added features. The following observations can be made from Tables 2 and 3 and Fig. 3:

(i). It can be observed that CAFS was able to select a smaller number of features for solving different datasets. For example, CAFS selected on average 5.80 features from a set of 8 features in solving the diabetes dataset. It also selected on average 4.26 features from a set of 60 features in solving the splice dataset. In fact, CAFS selected a very small number of features for datasets with more features. The feature reduction in such datasets was several orders of magnitude (Table 2).

(ii). The positive effect of FS is evident on the hidden neurons of NNs. For example, NNs produced by CAFS for the cancer dataset had on average 1.36 hidden neurons, while those produced by CAFS without FS had on average 2.50 hidden neurons. However, CAFS with FS took a smaller number of hidden neurons for solving all the datasets except only ionosphere. This is reasonable in the sense that the reduction of feature may sometime make a problem harder to solve, thereby an NN may require more hidden neurons. Since we do not know in advance whether the reduction of features would make a problem

**Table 2**
Performance of CAFS for different classifications datasets.

| Dataset | CAFS | No. of feature | | No. of HNs | | No. of connections | | Class. acc. (%) | |
|---|---|---|---|---|---|---|---|---|---|
| | | Mean | SD | Mean | SD | Mean | SD | Mean | SD |
| Diabetes | Without FS | 8.00 | 0.00 | 2.90 | 0.94 | 32.00 | 9.43 | 75.95 | 0.95 |
| | With FS | 5.80 | 0.90 | 1.53 | 0.71 | 15.46 | 7.26 | 76.18 | 1.32 |
| Cancer | Without FS | 9.00 | 0.00 | 2.50 | 0.67 | 30.50 | 7.37 | 98.17 | 0.30 |
| | With FS | 6.33 | 0.47 | 1.36 | 0.54 | 14.36 | 4.62 | 98.76 | 0.70 |
| Glass | Without FS | 9.00 | 0.00 | 3.06 | 1.65 | 40.73 | 18.17 | 74.02 | 6.65 |
| | With FS | 4.73 | 0.81 | 1.70 | 0.64 | 18.50 | 4.86 | 76.91 | 2.26 |
| Vehicle | Without FS | 18.0 | 0.00 | 1.80 | 0.90 | 41.00 | 18.18 | 72.68 | 4.55 |
| | With FS | 2.70 | 0.64 | 1.30 | 0.52 | 11.30 | 3.31 | 74.56 | 0.76 |
| Hepatitis | Without FS | 19.0 | 0.00 | 1.20 | 0.47 | 28.20 | 9.99 | 69.57 | 5.32 |
| | With FS | 2.06 | 0.35 | 1.03 | 0.17 | 7.26 | 1.43 | 79.40 | 0.46 |
| Horse | Without FS | 21.0 | 0.00 | 3.10 | 1.46 | 74.30 | 33.77 | 84.10 | 1.95 |
| | With FS | 8.10 | 1.16 | 2.60 | 0.98 | 29.76 | 11.86 | 83.41 | 3.65 |
| Ionosphere | Without FS | 34.0 | 0.00 | 1.30 | 0.93 | 49.80 | 33.70 | 95.56 | 1.76 |
| | With FS | 6.73 | 1.93 | 1.76 | 0.84 | 19.20 | 10.12 | 96.55 | 1.82 |
| Splice | Without FS | 60.0 | 0.00 | 3.50 | 1.66 | 221.0 | 103.4 | 74.84 | 2.55 |
| | With FS | 4.26 | 1.61 | 1.80 | 0.83 | 16.46 | 9.02 | 77.78 | 3.43 |

The results were averaged over 30 independent runs. Here SD and HN refer to standard deviation and hidden neurons, respectively.

**Table 3**
Average run time for different datasets over 30 independent runs.

| Datasets | Diabetes | Cancer | Glass | Vehicle | Hepatitis | Horse | Ionosphere | Splice |
|---|---|---|---|---|---|---|---|---|
| Time (min:s) | 0:25.76 | 0:17.24 | 0:22.13 | 0:6.97 | 0:2.35 | 2:08 | 0:11.91 | 33:43 |

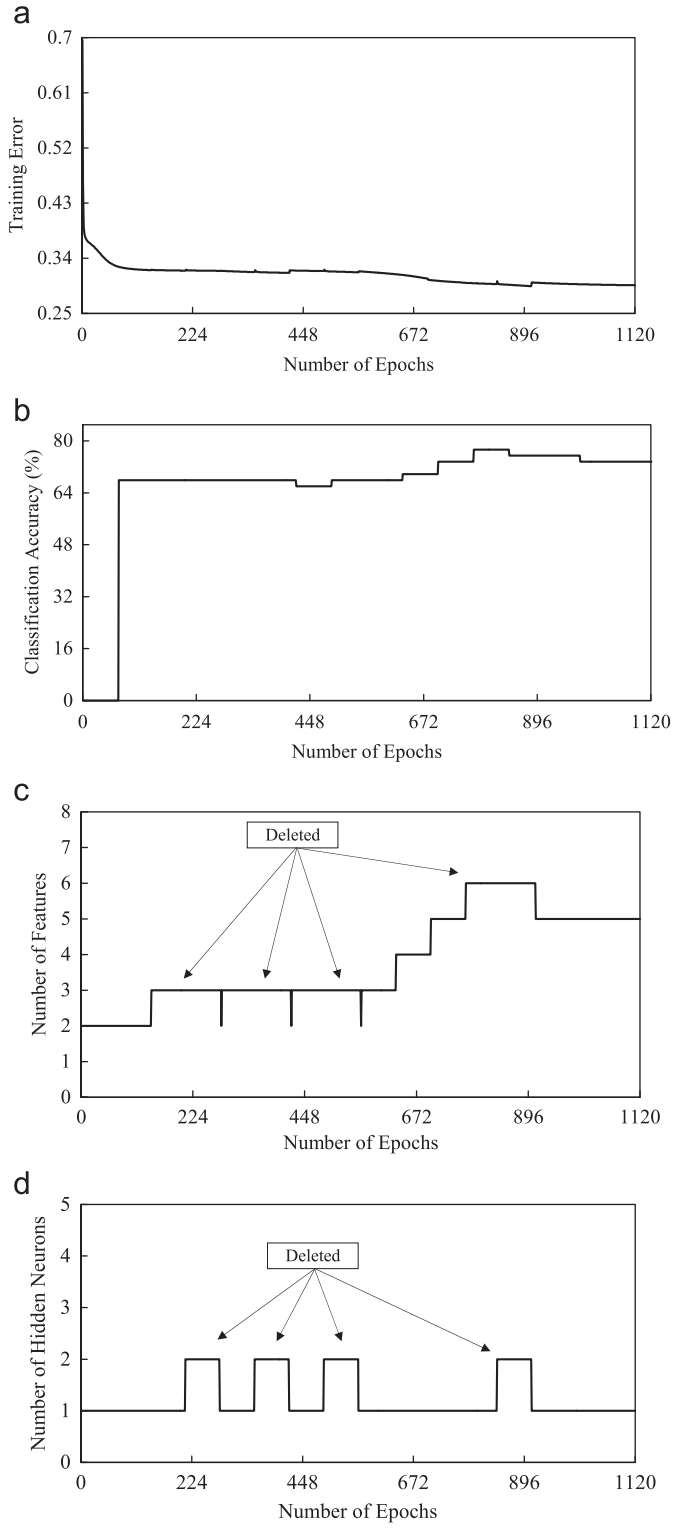


**Fig. 3.** Training process of CAFS for the glass dataset: (a) training error, (b) classification accuracy on validation set, (c) the feature addition process, and (d) the hidden neuron addition process.

harder, it is very difficult to decide the number of hidden neurons in NNs for solving given datasets. One solution to this dataset is the automatic determination of hidden neurons during the FS process, which is adopted in our proposed CAFS.

(iii). It can be seen that CAFS with FS produced compact NN architectures for solving all eight datasets we used in this work. For example, NNs produced by CAFS with FS for the hepatitis dataset had on average 7.26 connections, while that produced by CAFS without FS had on average 28.20 connections. The reduction of network architecture is several orders of magnitude for most of the datasets (Table 2). The positive effect of compact architectures is clearly visible on the classification accuracy of produced NNs. For example, for the glass dataset, the average *CA* of NN produced by CAFS with FS was 76.91%, while it was 74.02% for the NN produced by CAFS without FS. The only exception is the horse dataset where CAFS without FS showed better accuracy than CAFS with FS. It is well known that compact NN architectures are beneficial for achieving good classification accuracy.

(iv). It can be seen that the average run time for all datasets is less or slightly more than 1 min in most of the datasets (Table 3). For the splice dataset, CAFS, however, took on average about 33 min which is much larger than other datasets. This is natural because the splice dataset consists of a large number features and examples (Table 1).

(v). It can be seen that training error on training set converged up to a certain limit as the training process progresses (Fig. 3(a)). However, there are some instances where the training error goes up. This is due to the addition of some irrelevant features and unnecessary hidden neurons that hamper the classification accuracy of the validation set or keep similar (Fig. 3(b)). Therefore, CAFS deletes such irrelevant features (Fig. 3(c)) with corresponding unnecessary hidden neurons (Fig. 3(d)) to generate a compact and salient feature subset.

In order to understand the essence of selected features, we measured information gain [29] and the frequency of features. The information gain $IG(P, N_i)$ of a feature $N_i$ relative to a collection of examples $P$ is defined as

$$\mathrm{IG}(P,N_i) \equiv Entropy(P) - \sum_{s \in \mathrm{Values}(N_i)} \frac{P_s}{P} Entropy(P_s), \tag{8}$$

where $\mathrm{Values}(N_i)$ is the set of all possible values for the feature $N_i$, and $P_s$ is the subset of $P$ for which $N_i$ has value $s$, i.e., $P_s = (p \in P | N_i(p) = s)$. It should be noted that $P_s$ is the number of sub-examples for a particular value of $s$ among the total number of examples $P$ of the given dataset. The frequency of a feature $N_i$ can be defined as

$$fr = \frac{H}{R} \tag{9}$$

where $R$ is the total number of simulation runs and $H$ is the number of times a particular feature is selected in all runs. Table 4 shows that information gain and frequency of features for diabetes, cancer, and glass datasets. It can be seen in Table 4 that CAFS selected features 1, 2, 6, 7, and 8 of the diabetes dataset vary frequently. This is why the frequency of selection for these

**Table 4**
Frequency (*fr*) of the features selected by CAFS and their information gain (*IG*) for diabetes, cancer, and glass datasets.

| Dataset | | Feature number | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Diabetes | *fr* | 0.63 | 1.0 | 0.37 | 0.37 | 0.50 | 0.93 | 1.00 | 1.00 | – |
| | *IG* | 0.06 | 0.37 | 0.09 | 0.15 | 0.32 | 0.48 | 0.76 | 0.19 | – |
| Cancer | *fr* | 1.00 | 0.30 | 0.00 | 1.00 | 1.00 | 1.00 | 0.03 | 1.00 | 1.00 |
| | *IG* | 0.43 | 0.65 | 0.63 | 0.45 | 0.5 | 0.52 | 0.54 | 0.46 | 0.2 |
| Glass | *fr* | 0.33 | 0.03 | 1.00 | 0.80 | 0.10 | 0.67 | 0.80 | 0.00 | 1.00 |
| | *IG* | 2.1 | 1.93 | 1.46 | 1.77 | 1.95 | 1.34 | 1.91 | 0.51 | 0.42 |

features is one or nearly one. It can be seen that the information gain of features 2, 6, and 7 was high, while that of features 1 and 8 was low. These results indicate that CAFS can select features that contain good amount of information. It should be kept in mind that the salient features selected by CAFS cannot match completely with respect to the information gain. This is reasonable in the sense that information gain is a statistical procedure, while CAFS selects features during learning based on their performance for different experimental conditions.

### 4.3. Effect of architecture determination

The results presented in Table 2 show that the essence of using FS and architecture determination in the same learning process. It is, however, not clear the effects of architecture determination in the whole system of CAFS. To observe such effects, we conducted a new set of experiments. The setup of these experiments was exactly the same as those described previously. The only difference was that CAFS did not determine here network architectures during the FS process. Rather, CAFS used fixed sized network architectures like conventional approaches. It is noted that CAFS used 15 hidden neurons individually for diabetes, cancer, glass, vehicle, hepatitis, and horse datasets while 20 and 25 hidden neurons were used in ionosphere and splice datasets, respectively.

Table 5 shows the average results of our new experiments over 30 independent runs. The positive effect of determining network architecture during the FS process is clearly visible from these results. For example, for the vehicle dataset, the classification accuracies of CAFS without architecture determination and CAFS were 70.85% and 74.56%, respectively. The similar results can be found for the other datasets as well. *t*-test was used here to determine whether the performance difference between CAFS and CAFS without architecture determination is statistically significant or not. It was found that CAFS was significantly better than CAFS without architecture determination at 95% confidence level for the all datasets except cancer, hepatitis, and horse dataset.

### 4.4. Effect of grouping of features

The essence of using FS in CAFS can be seen in Table 2, but the effect of grouping in the FS process of CAFS is not clear. Therefore, a new set of experiments has been carried out to observe such effects. The setup of these experiments was exactly the same as those described before. The only difference was that CAFS did not divide the original feature set into two groups before training. Rather, CAFS kept the features in the same group as the original feature set and selected features one by one randomly as the conventional forward FS approach.

**Table 5**
Effect of the architecture determination and grouping of features on the classification accuracy of CAFS for different classification datasets.

| Dataset | Classification accuracy (%) | | | | | |
|---|---|---|---|---|---|---|
| | CAFS with fixed architecture | | CAFS without grouping | | CAFS | |
| | Mean | SD | Mean | SD | Mean | SD |
| Diabetes | 71.04 | 3.84 | 73.50 | 3.27 | 76.18 | 1.32 |
| Cancer | 98.39 | 0.78 | 96.83 | 1.08 | 98.76 | 0.70 |
| Glass | 75.47 | 3.08 | 74.08 | 4.46 | 76.91 | 2.26 |
| Vehicle | 70.85 | 5.70 | 74.31 | 1.81 | 74.56 | 0.76 |
| Hepatitis | 74.51 | 1.62 | 72.19 | 4.97 | 79.40 | 0.46 |
| Horse | 82.67 | 3.96 | 84.61 | 4.44 | 83.41 | 3.65 |
| Ionosphere | 94.62 | 2.70 | 89.50 | 11.70 | 96.55 | 1.82 |
| Splice | 72.74 | 1.93 | 81.22 | 3.77 | 77.78 | 3.43 |

Table 5 also shows the average results of our new experiments over 30 independent runs. The positive effect of grouping of features can clearly be understood from these results. For example, for the ionosphere dataset, the classification accuracy of CAFS with and without grouping of features was 96.55% and 89.50%, respectively. A similar classification improvement for CAFS with grouping of features was also observed for the other datasets except horse and splice. The performance of CAFS was also found very consistent, i.e., low standard deviation (SD), under different experimental set-ups. Furthermore, *t*-test shows that the classification accuracy of CAFS with grouping was significantly better than that of CAFS without grouping at 95% confidence level for the all datasets, except the vehicle, horse, and splice datasets.

### 4.5. Comparison with other works

The obtained results of CAFS on several benchmark classification datasets have been compared here with the results of different FS algorithms. Table 6 shows the results of CAFS and seven other algorithms, i.e., neural network feature selector (NNFS) [42], MLP-based FS method (MLPFS) [8], incremental approach to contribution-based FS (ICFS) [9], artificial neural net input gain measurement approximation (ANNIGMA) [12], hybrid genetic algorithm for FS (HGAFS) [13], genetic programming for FS (GPFS) [27], and automatic discoverer of higher-order correlations (ADHOC) [40]. We used two parameters for comparisons. They are the number of selected features and classification accuracy.

The aforementioned eight FS techniques represent a wide range of FS techniques. The five FS techniques, i.e., CAFS, NNFS, MLPFS, ANNIGMA, and ICFS use NNs as classifiers. In the remaining three techniques, GPFS and ADHOC use decision trees as classifiers, while HGAFS uses support vector machine. An

**Table 6**
Comparison among CAFS, NNFS [42], ICFS [9], MLPFS [8], ANNIGMA [12], HGAFS [13], GPFS [27], and ADHOC [40] for the diabetes, cancer, glass, vehicle, and ionosphere datasets. "–" means not available.

| Dataset | | CAFS | NNFS | ICFS | MLPFS | ANNIGMA | HGAFS | GPFS | ADHOC |
|---|---|---|---|---|---|---|---|---|---|
| Diabetes | No. of features | 5.80 | 2.03 | 2.50 | – | 5.20 | – | – | 3.00 |
| | Class. acc. (%) | 76.18 | 74.30 | 78.79 | – | 77.80 | – | – | 71.20 |
| Cancer | No. of features | 6.33 | 2.70 | 5.00 | 8.00 | 5.80 | – | 2.23 | – |
| | Class. acc. (%) | 98.76 | 94.10 | 98.25 | 89.40 | 96.5 | – | 96.84 | – |
| Glass | No. of features | 4.73 | – | 4.50 | 8.00 | – | 5.00 | – | 4.00 |
| | Class. acc. (%) | 76.91 | – | 65.19 | 44.10 | – | 65.51 | – | 70.50 |
| Vehicle | No. of features | 2.70 | – | – | 13.00 | – | 11.00 | 5.37 | 7.00 |
| | Class. acc. (%) | 74.56 | – | – | 74.60 | – | 76.36 | 78.45 | 69.6 |
| Ionosphere | No. of features | 6.73 | – | – | 32.00 | 9.00 | 6.00 | – | – |
| | Class. acc. (%) | 96.55 | – | – | 90.60 | 90.20 | 92.76 | – | – |

important component of any FS technique is the searching strategy used for finding a set of salient features. The NNFS, ANNIGMA and MLPFS use a backward selection strategy in finding salient features, while CAFS and ICFS use a forward selection strategy. The ADHOC and HGAFS use a global search strategy genetic algorithm in finding salient features, while GPFS uses genetic programming, a variant of genetic algorithm. The NNFS uses training set, validation set and testing set, while ANNIGMA, ICFS, and ADHOC use only the training set and testing set. The two algorithms MLPFS and GPFS use 10-fold cross-validation. A similar method, i.e., *k*-fold cross-validation is used in HGAFS where *k* refers to the value ranging 2–10 depending on the given dataset scale. The above-mentioned algorithms not only use different data partitions but also employ different number of independent runs in measuring average performances. For example, CAFS and ANNIGMA used 30 runs; ICFS used 20 runs; MLPFS, GPFS, and ADHOC used 10 runs. It is important to note that no more information regarding the number of runs was mentioned in the literature of NNFS and HGAFS.

It can be seen that NNs produced by CAFS achieved the best classification accuracy among all other algorithms for three out of five datasets. For the remaining two datasets, CAFS achieved one as a third best and the next one as the fourth best, while ICFS and GPFS achieved the best classification accuracy for one dataset each. In terms of features, CAFS selected the smallest number of features for one out of five datasets and the second smallest for one dataset, i.e., next to HGAFS.

It can importantly be said that FS improves the performance of classifiers by ignoring the irrelevant features from the original feature set. An important task in such a process is to capture necessary information in selecting salient features; otherwise, the performance of classifiers might be degraded. For example, for the diabetes dataset, NNFS selected the smallest feature subset consisting of 2.03 features but it achieved lower *CA*. On the other hand, CAFS selected a bulky feature subset that provides better *CA* compared to others for the cancer dataset. In fact, the results presented for other algorithms presented in Table 6 indicate that the smallest or largest feature subset does not guarantee the best or worst *CA*.

Some good aspects of CAFS amplify the performances which can be summarized as follows. First, in course of FS process CAFS automatically determines the architecture of the NN classifier, while the other approaches (e.g., ICFS, MLPFS, and ANNIGMA) use bulky and randomly selected NN architectures. It has been known that the automatic selection of NN architecture is beneficial for achieving good classification accuracy [14,15,18,19,23,39,48]. Second, CAFS utilizes correlation information in order to capture general and special characteristics of a given dataset, so that a classifier can learn all necessary information about the dataset. The proposed CAFS therefore selects features from both *S* and *D* groups. This kind of technique is not used in other FS techniques; thereby they may select redundant features or may not capture necessary information to learn a dataset in a better way. Third, CAFS uses mainly three user-specified parameters: number of training epochs $\tau$, training error threshold $\varepsilon$, and validation error threshold $\lambda$. In contrast, other FS approaches use many user-specified parameters.

One drawback of CAFS is that it uses a sequential search strategy in selecting salient features and hidden neurons of NNs. Although the sequential search strategy is simple to implement and fast in producing results, it ignores the completeness and thus there are risks in losing optimal feature subsets [21] and hidden neurons. This may be the main reason that the performance of CAFS in terms of selected features and classification accuracy was not better in some cases.

## 5. Conclusions

A desirable aspect of using NNs is their good classification accuracies, which is greatly dependent not only on the selected features of datasets but also on the architecture of NNs. Although a number of techniques exist to select salient features for solving datasets efficiently, most of them do not pay attention on the architecture of NNs. They use a predefined, fixed and manually determined NN architectures. This manual determination scheme may hurt classification accuracy of NNs. In this paper, we propose a new a wrapper based FS technique, CAFS, using NNs. The idea behind CAFS is to put emphasis on the simultaneous selection of input features and NN architectures automatically. The proposed CAFS divides the input features into two groups based on their correlation. It then uses a constructive approach to select a set of distinct features from two groups and to determine the architecture of NNs. The reason for using the two groups and constructive approach is to reduce the number of user-specified parameters in the whole system of CAFS.

The extensive experiments reported in this paper have been carried out to evaluate the essence of using different schemes of CAFS and to compare CAFS with other FS techniques. Eight benchmark classification datasets were used in our experimental studies. The positive effect of using the constructive approach and grouping of features based on correlation is observed (Table 5). The comparison of CAFS with other FS techniques showed that CAFS was better or comparable. Since CAFS uses a sequential approach in selecting a set of salient features and the number of hidden neurons for NN classifiers, it may suffer from the so-called

*nesting effect* [33]. This is because the features once selected cannot be discarded in the scheme of CAFS. One way to avoid this effect is to incorporate a global search procedure such as genetic algorithms, evolutionary programming, and genetic programming. The main challenges in such incorporation are the use of a minimum number of user-specified parameters in the whole scheme of CAFS. This could be an interesting future research topic.

## Acknowledgements

Supported by grants to KM from the Japanese Society for Promotion of Sciences, the Yazaki Memorial Foundation for Science and Technology, and the University of Fukui.

## References

[1] S. Abe, Modified backward feature selection by cross validation, in: Proceedings of the European Symposium on Artificial Neural Networks 2005, pp. 163–168.
[2] A. Al-Ani, Feature subset selection using ant colony optimization, International Journal of Computational Intelligence 2 (1) (2005) 53–58.
[3] R. Caruana, D. Freitag, Greedy attribute selection, Proceedings of the 11th International Conference of Machine Learning, Morgan Kaufmann, USA, 1994.
[4] D. Chakraborty, N.R. Pal, A neuro-fuzzy scheme for simultaneous feature selection and fuzzy rule-based classification, IEEE Transactions on Neural Networks 15 (1) (2004) 110–123.
[5] M. Das, H. Liu, Feature selection for clustering, Proceedings of Pacific-asia Conference on Knowledge Discovery and Data Mining (2000) 110–121.
[6] K. Dunne, P. Cunningham, F. Azuaje, Solutions to instability problems with sequential wrapper-based approaches to feature selection, Journal of Machine Learning Research, 2002.
[7] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Journal of Machine Learning Research 3 (2003) 1157–1182.
[8] E. Gasca, J.S. Sanchez, R. Alonso, Eliminating redundancy and irrelevance using a new MLP-based feature selection method, Pattern Recognition 39 (2006) 313–315.
[9] S. Guan, J. Liu, Y. Qi, An incremental approach to contribution-based feature selection, Journal of Intelligence Systems 13 (1) (2004).
[10] F. Girosi, M. Jones, T. Poggio, Regularization theory and neural networks architectures, Neural Computation 7 (2) (1995) 219–269.
[11] M.A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in: Proceedings of the 17th International Conference on Machine Learning, 2000.
[12] C. Hsu, H. Huang, D. Schuschel, The ANNIGMA-wrapper approach to fast feature selection for neural nets, IEEE Transactions on Systems Man, and Cybernetics—Part B: Cybernetics 32 (2) (2002) 207–212.
[13] J. Huang, Y. Cai, X. Xu, A hybrid genetic algorithm for feature selection wrapper based on mutual information, Pattern Recognition Letters 28 (2007) 1825–1844.
[14] Y. Hirose, K. Yamashita, S. Hijiya, Back-propagation algorithm which varies the number of hidden units, Neural Network 4 (1) (1991) 61–66.
[15] M.M. Islam, K. Murase, A new algorithm to design compact two hidden-layer artificial neural networks, Neural Network 14 (9) (2001) 1265–1278.
[16] M.M. Islam, X. Yao, K. Murase, A constructive algorithm for training cooperative neural network ensembles, IEEE Transactions on Neural Networks 14 (4) (2003) 820–834.
[17] M. Kudo, J. Sklansky, Comparison of algorithms that select features for pattern classifiers, Pattern Recognition 33 (2000) 25–41.
[18] T.Y. Kwok, D.Y. Yeung, Constructive algorithms for structure learning in feed-forward neural networks for regression problems, IEEE Transactions on Neural Networks 8 (1997) 630–645.
[19] T.Y. Kwok, D.Y. Yeung, Objective functions for training new hidden units in constructive neural networks, IEEE Transactions on Neural Network 8 (5) (1997) 1131–1148.
[20] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1–2) (1997) 273–324.
[21] H. Liu, Lei Tu, Toward integrating feature selection algorithms for classification and clustering, IEEE Transactions on Knowledge and Data Engineering 17 (4) (2005) 491–502.
[22] Y. Lei, H. Liu, Feature selection for high-dimensional data: a fast correlation-based filter solution, in: Proceedings of the 20th International Conference on Machine Learning (ICML), 2003.
[23] M. Lehtokangas, Modeling with constructive back propagation, Neural Network 12 (4–5) (1999) 707–716.
[24] M. Lehtokangas, Modified cascade-correlation learning for classification, IEEE Transactions on Neural Networks 11 (2000) 795–798.
[25] C. Lai, M.J.T. Reinders, L. Wessels, Random subspace method for multivariate feature selection, Pattern Recognition Letters 27 (2006) 1067–1076.
[26] K. Michalak, H. Kwasnicka, Correlation-based feature selection strategy in neural classification, in: Proceedings of the 6th International Conference on Intelligent Systems Design and Applications (ISDA), 2006.
[27] D.P. Muni, N.R. Pal, J. Das, Genetic programming for simultaneous feature selection and classifier design, IEEE Transactions on Systems Man, and Cybernetics–Part B: Cybernetics 36 (1) (2006) 106–117.
[28] P. Mitra, C.A. Murthy, S.K. Pal, Unsupervised feature selection using feature similarity, IEEE Transaction on Pattern Analysis and Machine Intelligence 24 (3) (2002).
[29] T.M. Mitchell, Machine learning, McGraw-Hill, 1997, pp. 55–61.
[30] P.M. Narendra, K. Fukunaga, A branch and bound algorithm for feature selection, IEEE Transactions on Computers C-26 (9) (1977) 917–922.
[31] V. Onnia, M. Tico, J. Saarinen, Feature selection method using neural network, Internationall. Conference on Image Processing 1 (2001) 513–516.
[32] I. Oh, J. Lee, B. Moon, Hybrid genetic algorithms for feature selection, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (11) (2004) 1424–1437.
[33] P. Pudil, J. Novovicova, J. Kittler, Floating search methods in feature selection, Pattern Recognition Letters 15 (11) (1994) 1119–1125.
[34] H. Peng, F. Long, C. Ding, Overfitting in making comparisons between variable selection methods, Journal of Machine Learning Research 3 (2003) 1371–1382.
[35] N.R. Pal, K. Chintalapudi, A connectionist system for feature selection, International Journal of Neural, Parallel and Scientific Computation 5 (1997) 359–381.
[36] L. Prechelt, PROBEN1-A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Faculty of Informatics, University of Karlsruhe, 1994.
[37] A. Rakotomamonjy, Variable selection using SVM-based criteria, Journal of Machine Learning Research 3 (2003) 1357–1370.
[38] E. Romero, J.M. Sopena, Performing feature selection with multilayer perceptrons, IEEE Transactions on Neural Networks 19 (3) (2008) 431–441.
[39] R. Reed, Pruning algorithms—a survey, IEEE Transactions on Neural Networks 4 (5) (1993) 740–747.
[40] M. Richeldi, P. Lanzi, ADHOC: a tool for performing effective feature selection, in: Proceedings of the Eighth IEEE Conference on Tools with Artificial Intelligence (ICTAI) (1996) 102–105.
[41] D.E. Rumelhart, J. McClelland, in: Parallel Distributed Processing, MIT Press, 1986.
[42] R. Setiono, H. Liu, Neural network feature selector, IEEE Transactions on Neural Networks vol. 8 (1997).
[43] R.K. Sivagaminathan, S. Ramakrishnan, A hybrid approach for feature subset selection using neural networks and ant colony optimization, Expert Systems with Applications 33 (2007) 49–60.
[44] D.J. Straceezzi, P.E. Utgoff, Randomized variable elimination, Journal of Machine Learning Research 5 (2004) 1331–1362.
[45] V. Sindhwani, S. Rakshit, D. Deodhare, D. Erdogmus, J.C. Principe, P. Niyogi, Feature selection in MLPs and SVMs based on maximum output information, IEEE Transactions on Neural Networks 15 (4) (2004) 937–948.
[46] A. Verikas, M. Bacauskiene, Feature selection with neural networks, Pattern Recognition Letters 23 (2002) 1323–1335.
[47] L. Wang, N. Zhou, F. Chu, A general wrapper approach to selection of class-dependent features, IEEE Transactions on Neural Networks 19 (7) (2008) 1267–1278.
[48] X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks, IEEE Transactions on Neural Networks 8 (3) (1997) 694–713.

**Md. Monirul Kabir** received the BE degree in Electrical and Electronic Engineering from Bangladesh Institute of Technology (BIT), Khulna, now Khulna University of Engineering and Technology (KUET), Bangladesh, in 1999. He received the ME degree in the department of Human and Artificial Intelligent Systems from the University of Fukui, Japan, in 2008. Now, he is pursuing the Ph.D. degree in the Department of System Design Engineering from the same university.

He was an assistant programmer from 2002 to 2005 at the Dhaka University of Engineering and Technology (DUET), Bangladesh. His research interest includes data mining, artificial neural networks, evolutionary approaches, and swarm intelligence.

**Md. Monirul Islam** received the BE degree from the Bangladesh Institute of Technology (BIT), Khulna, now Khulna University of Engineering and Technology (KUET), Bangladesh, in 1989. He received the ME degree from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 1996, and the Ph.D. degree from the University of Fukui, Japan, in 2002. He was a Lecturer and Assistant Professor from 1989 to 2002 at BIT, Khulna. He moved to BUET as an Assistant Professor of Computer Science and Engineering in 2003, where he is now a Professor. He has worked as a visiting Associate Professor supported by the Japanese Society for promotion of Sciences (JSPS) at University of Fukui in 2007–2009. His major research interests include evolutionary robotics, evolutionary computation, neural networks, machine learning, pattern recognition, and data mining. He has more than 80 refereed publications. He won the First Prize in The Best Paper Award Competition of the Joint 3rd International Conference on Soft Computing and Intelligent Systems and 7th International Symposium on advanced Intelligent Systems.

**Kazuyuki Murase** is a Professor at the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui, Japan, since 1999. He received ME in Electrical Engineering from Nagoya University in 1978, Ph.D. in Biomedical Engineering from Iowa State University in 1983. He Joined as a Research Associate at Department of Information Science of Toyohashi University of Technology in 1984, as an Associate Professor at the Department of Information Science of Fukui University in 1988, and became the professor in 1992. He is a member of The Institute of Electronics, Information and Communication Engineers (IEICE), The Japanese Society for Medical and Biological Engineering (JSMBE), The Japan Neuroscience Society (JSN), The International Neural Network Society (INNS), and The Society for Neuroscience (SFN). He serves as a Board of Directors in Japan Neural Network Society (JNNS), a Councilor of Physiological Society of Japan (PSJ) and a Councilor of Japanese Association for the Study of Pain (JASP).