Neurocomputing 102 (2013) 98–110

Contents lists available at SciVerse ScienceDirect

Neurocomputing

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

Evolutionary extreme learning machine ensembles with size control

Dianhui Wang*, Monther Alhamdoosh

Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, VIC 3086, Australia

ARTICLE INFO

Available online 9 June 2012

Keywords:
Learner model ensembles
Extreme learning machines
Evolutionary computation
Generalization capability
Robustness

ABSTRACT

Ensemble learning aims to improve the generalization power and the reliability of learner models through sampling and optimization techniques. It has been shown that an ensemble constructed by a selective collection of base learners outperforms favorably. However, effective implementation of such an ensemble from a given learner pool is still an open problem. This paper presents an evolutionary approach for constituting extreme learning machine (ELM) ensembles. Our proposed algorithm employs the model diversity as fitness function to direct the selection of base learners, and produces an optimal solution with ensemble size control. A comprehensive comparison is carried out, where the basic ELM is used to generate a set of neural networks and 12 benchmarked regression datasets are employed in simulations. Our reporting results demonstrate that the proposed method outperforms other ensembling techniques, including simple average, bagging and adaboost, in terms of both effectiveness and efficiency.

1. Introduction

Performances of learning systems are largely dependent on training examples and the initial settings of model parameters. For data regression using neural networks, each well-trained neural network demonstrates different generalization performance and robustness property with respect to various uncertainties due to the local minima phenomena. This uncertain nature makes the learning-based modeling approach less favorable [1–3]. Over the past years, ensembling framework has received a considerable attention in machine learning community, and many excellent researches on this topic have been reported in the literature [1–11]. The main idea behind ensembling learning is to minimize the risk of modeling error through integrating several base learner models, which can be generated by using either different subsets of training examples or the whole training dataset but with different settings in learning process. Usually, the averaging operator is applied in model fusion for data regression problems [2].

Basically, building ensemble models involves two steps: training the ensemble components individually/collectively and then combining these learnt models to produce aggregate predictions. While some methods use a fixed dataset to train all components [1,2], others use variable learning sets. For instance, bootstrap aggregating ensemble (*Bagging*) generates different training sets by repetitive resampling of a given dataset [4]. Similarly, *Boosting* ensemble method populates its component models incrementally by giving misclassified examples from previous models more chances to be selected in the training of subsequent models [5,6]. Boosting usually performs better than bagging, but it is very sensitive to noisy data. Similar to bagging, boosting may not use all training examples. For effective utilization of training examples, a dataset set can be partitioned into m subsets and one of the subsets is picked out circularly while the remaining $(m-1)$ subsets are used to train ensemble components [3]. This cross-validation procedure is also used to estimate ensembles generalization errors and decide the ensemble sizes [1–3]. Differently, the negative correlation learning was proposed to simultaneously train ensemble components in order to guarantee that the prediction errors among components are negatively correlated and the resulting ensemble has better generalization [7]. Once ensemble components are well tuned, they are merged together by simple or weighted averaging for regression tasks [3] or by majority voting or winner-takes-all for classification tasks [1,2]. The averaging weights in the ensemble model can be assigned by either heuristic means or minimizing a cost function [8,9].

It has been shown that an ensemble is considered to be well constructed if some disagreement metrics of the base learner models can be maximized or minimized [3,10]. In practice, however, it is still quite challenging to optimize ensembles in terms of size, averaging weights and candidate selection. Perrone and Cooper came up with the idea of selecting ensemble base models [2]. They defined a correlation matrix between the outputs of individual neural networks to analytically remove duplicated networks from population. Two networks are

* Corresponding author. Tel.: +61 3 9479 3034; fax: +61 3 9479 3060.
E-mail address: dh.wang@latrobe.edu.au (D. Wang).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2011.12.046

considered as being redundant if their corresponding rows in the correlation matrix are collinear, i.e., their dot product is above a preset threshold. Alternatively, the top L networks ranked according to their mean square errors (MSEs) on the testing dataset are selected for ensembling [2]. Unfortunately, the top ranked L models are not always the best combination for ensembling. To select right models from a given pool of networks to form an optimal ensemble, genetic algorithms (GAs) have been employed by some researchers. Yao and Liu [8] encoded the whole networks population as flip-flop binary chromosomes so that 1 bit represents one network. Eventually, the networks corresponding to ones are collected for ensembling. In different manner, the averaging weights of initial networks ensemble were represented as real chromosomes and finally the networks weighted below a preset threshold were removed, e.g., GASEN [11]. Although the robustness of these approaches was not well defined and evaluated, they had raised a strong motivation for achieving selective ensembles with less capacity and higher diversity [8,10,11].

In principle, the base learner models for data regression can be any types of neural networks provided that they are universal approximators. However, the learner models used in the ensembling framework should be relatively reliable and easily generated. Over the past decades, numerical optimization techniques-based training algorithms, such as error back-propagation (BP) algorithm and its variations, have been widely used to train neural networks. Unfortunately, they suffer from varying generalization performance caused by weight initialization and/or overfitting, intensive human intervention, and unbearably slow learning speed. Indeed, the model unreliability and the computational complexity block neural network ensembles to achieve highly [1–3]. ELM, proposed by Huang et al. [12–14], learns network parameters analytically and it has been shown in many occasions that ELM produces better performance than BP and Support Vector Machines (SVMs) in terms of learning speed, reliability and generalization [15]. These unique features motivate some researchers to explore ELM ensembles [16–19]. Chen et al. [16] proposed simple averaging based ensemble that elects ELM models from a population of models according to a product index defined for each model by the correlation between the ambiguity and the MSE values. Product Index based Excluding (PIEx) ensemble retains the base models that have product indices higher than the mean product index of the initial population. In the same context, cross-validation has been utilized to optimize ELM ensembles based on the norms of the output weights of base models. EN-ELM starts with $R\times K$ ELM models generated from R-fold cross-validation over K iterations, and then the first half of models are selected, according to their output weights norms, to constitute the ensemble [17]. In fact, this method is based on an understanding that the smaller the norm of network weights is, the better the generalization capability may be. Heeswijk et al. introduced an adaptive ELM ensemble in which the averaging weights are updated and the component ELM models themselves are retrained when new example is presented [18]. Experimental results show that the performance of their method is comparable with those obtained by the state-of-the art methods for stationary and non-stationary time series predictions, but with less computational cost. However, the ensemble size and the weights update rate have to be preset. Recently, a modified version of Adaboost.RT has utilized ELM to build boosting ensembles [20]. Instead of experimentally assigning the threshold parameter ϕ of Adaboost.RT, it has been tuned adaptively in accordance with the error rate of consecutive learners [19]. Clearly, this modification made the Adaboost.RT less sensitive to the value of ϕ. Compared with other learning methods, this learning scheme is faster, less user-driven and more generalized.

In the light of the merits of ELM for fast generating better performed neural networks (single hidden layer), we propose an evolutionary approach for implementing ELM ensembles, named as EE-ELM. Our proposed EE-ELM starts with a pool of ELM models generated from a fixed dataset, and then selects the best ensemble of ELM models through iterative process that takes the advantage of genetic algorithms in finding optimal solutions. The key characteristics of EE-ELM lie in presenting a new encoding scheme for the potential ensembles such that the ensemble size can be controlled during the evolutionary process. Also, unlike bagging and boosting ensembles, the EE-ELM is generated without partitioning the training dataset. Empirical data indicates that our proposed algorithm can produce an improved performance comparing to simple, bagging, Adaboost.RT, top-K, and random-K ensembles. This study reconfirms a statement of “many could be better than all”, and it is observed that some weak base models can be selected to contribute to the EE-ELM model. Our last observation from simulations tells that the sizes of ELM ensembles for the 12 testing datasets vary from 3 to 8. Such small sized ensembles will result in a fast modeling tool, which will be significant and useful to some domain applications, e.g., process control and intelligent image processing.

The remainder of this paper is organized as follows. Section 2 briefly reviews ELM basics and ensemble learning theory for data regression. Section 3 details our algorithm thoroughly. Section 4 evaluates our proposed algorithm with 12 benchmarked datasets and comprehensive comparisons. Section 5 concludes our work with some remarks on further research.

2. Related work

2.1. Extreme learning machine basics

For N arbitrary distinct samples (x_i,t_i), where $x_i=[x_{i1},x_{i2},\ldots,x_{in}]^T\in\mathbf{R}^n$ and $t_i=[t_{i1},t_{i2},\ldots,t_{im}]^T\in\mathbf{R}^m$, a standard SLFN with $\tilde{N}$ hidden neurons and an activation function $g(x)$ is mathematically modeled as

$$\sum_{i=1}^{\tilde{N}}\beta_i g(a_i,b_i,x_j)=o_j,\quad j=1,\ldots,N, \tag{1}$$

where $a_i=[a_{i1},a_{i2},\ldots,a_{in}]^T$ is the weight vector connecting the ith hidden neuron and the input neurons; $\beta_i=[\beta_{i1},\beta_{i2},\ldots,\beta_{im}]^T$ is the weight vector connecting the ith hidden neuron and the output neurons; and b_i is the bias of the ith hidden neuron. The standard SLFNs can approximate these N samples with zero error if there exist β_i, w_i and b_i such that

$$\sum_{i=1}^{\tilde{N}}\beta_i g(a_i,b_i,x_j)=t_j,\quad j=1,\ldots,N. \tag{2}$$

The above N equations can be rewritten in a matrix form

$$H\beta=T, \tag{3}$$

where

$$H=\begin{bmatrix} g(a_1,b_1,x_1) & \cdots & g(a_{\tilde{N}},b_{\tilde{N}},x_1)\\ \vdots & \cdots & \vdots\\ g(a_1,b_1,x_N) & \cdots & g(a_{\tilde{N}},b_{\tilde{N}},x_N)\end{bmatrix}_{N\times\tilde{N}},$$

$$\beta=\begin{bmatrix}\beta_1^T\\ \vdots\\ \beta_{\tilde{N}}^T\end{bmatrix}_{\tilde{N}\times m}\quad\text{and}\quad T=\begin{bmatrix}t_1^T\\ \vdots\\ t_{\tilde{N}}^T\end{bmatrix}_{\tilde{N}\times m},$$

here the matrix H is called the hidden layer output matrix; the ith column of H is the ith hidden neuron output vector with respect to the inputs $x_1,x_2,\ldots,x_N$.

Since the number of distinct training examples is usually much greater than the number of hidden neurons ($N \gg \tilde{N}$), the H matrix is rectangular and tuning the parameters a_i,b_i,β_i ($i=1,\ldots,\tilde{N}$) in order to obtain a unique solution for Eq. (3) is almost impossible. Huang et al. proposed a fast solution, ELM, from function approximation perspective [12]. Their idea is very simple, i.e. the input weights and biases are randomly set and fixed, then followed by solving the linear parameter vector β in Eq. (3). Such a solution can be obtained by the Moor–Penrose generalized inverse, i.e.,

$$\hat{\beta} = H^{\dagger}T, \tag{4}$$

where $H^{\dagger}=(H^TH)^{-1}H^T$ is the pseudoinverse matrix of the hidden layer output matrix H. As analyzed by Huang et al., ELM can reach good generalization performance by ensuring two properties of learning: the smallest norm of weights besides the smallest squared error on the training samples, while the gradient-based algorithms focuses on the later property only. The main steps involved in ELM are summarized in Algorithm 1.

Algorithm 1. Build basic ELM models.

```
Require: Training dataset, hidden layer size $\tilde{N}$, and an activation function $g: \mathbf{R} \mapsto \mathbf{R}$, default is Sigmoid.
Ensure: Trained SLFN.
1: Assign the input weights and biases randomly.
2: Compute the hidden layer output matrix $H$ using the training dataset.
3: Compute the pseudoinverse matrix $H^{\dagger}$ of $H$.
4: Set the output weights $\hat{\beta} = H^{\dagger} \cdot T$.
5: return ELM model.
```

2.2. Review of learner model ensembles

The difficulty of estimating optimal parameters is inherent in neural networks since ever and has made creating neural networks for real applications more like handcraft than science. Hansen and Salamon [1] proposed the neural network ensembles to reduce the impact of uncontrolled weaknesses exposed in learning neural models. Instead of using one network to approximate a function, multiple networks are combined such that wrong decisions cancel each other and reduce the overall prediction error [2]. In this section, we mathematically define ensemble generalization error for regression models and explain how ensemble ambiguity and diversity are important for building more reliable ensemble learners. All formulas can be generalized for classification problems.

2.2.1. Notation

Given an ensemble of N neural networks and a training dataset $D=\{(x_i,t_i)\in \mathbf{R}^n \times \mathbf{R};\ i=1,\ldots,M\}$, the output of the jth network on the kth data example is designated by $f_j(x_k)$ and the ensemble collective output is given by

$$f(x_k) = \sum_{j=1}^{N} w_j f_j(x_k), \tag{5}$$

where w_j is the averaging weight of the jth component network and reflects the contribution of this component in the final ensemble decision. In other words, the averaging weights must satisfy the following constraints:

$$\sum_{j=1}^{N} w_j = 1 \quad \text{and} \quad 0 \le w_j \le 1. \tag{6}$$

The learning error of the jth component network and the learning error of ensemble on the kth data sample are given, respectively, as follows:

$$E_j(x_k) = (f_j(x_k) - t_k)^2, \tag{7}$$

$$E(x_k) = (f(x_k) - t_k)^2. \tag{8}$$

Then the learning error of the jth component network and that of ensemble on the whole training dataset are formulated as follows:

$$E_j = \sum_{k=1}^{M} E_j(x_k), \tag{9}$$

$$E = \sum_{k=1}^{M} E(x_k). \tag{10}$$

Substituting Eqs. (8) and (5) in Eq. (10) gives the ensemble loss function as follows:

$$E = \sum_{k=1}^{M}\left(\sum_{j=1}^{N} w_j f_j(x_k) - t_k\right)^2. \tag{11}$$

Practically, small values for the quadratic loss function given in Eq. (11) do not truly mean better performance and that the ensemble well generalizes the rules hidden in the training dataset. This leads to further analysis on Eq. (11) in the next subsection.

2.2.2. Characterizations on ensemble generalization

Krogh and Vedelsby [3] showed that identical networks contribute to the ensemble generalization as one network and the disagreement among ensemble components can be quantified. Therefore, they derived a more reliable estimate for the ensemble generalization error given in Eq. (11) by introducing the ensemble ambiguity concept. The theoretical framework of ensemble ambiguity derivation is based on the bias-variance decomposition. Given a validation dataset $D_0=\{(x_i,t_i)\in \mathbf{R}^n \times \mathbf{R};\ i=1,\ldots,L\}$ and an ensemble of size N, the ambiguity of the jth component network on a sample x_k is defines as

$$A_j(x_k) = (f_j(x_k) - f(x_k))^2, \tag{12}$$

then the *ensemble ambiguity* on an example x_k is

$$\overline{A}(x_k) = \sum_{j=1}^{N} w_j A_j(x_k). \tag{13}$$

From Eq. (13) and using Eqs. (6)–(8) it can be concluded that

$$\overline{A}(x_k) = \sum_{j=1}^{N} w_j E_j(x_k) - E(x_k). \tag{14}$$

The first term of Eq. (14) is called the weighted average of the component networks' errors on an input x_k and denoted by $\overline{E}(x_k)$. This yields

$$E(x_k) = \overline{E}(x_k) - \overline{A}(x_k). \tag{15}$$

Considering the whole validation dataset D_0, Eqs. (12), (13) and (15) become

$$A_j = \sum_{k=1}^{L} A_j(x_k), \tag{16}$$

$$\overline{A} = \sum_{j=1}^{N} w_j A_j, \tag{17}$$

$$\overline{E} = \sum_{j=1}^{N} w_j E_j, \tag{18}$$

$$E = \overline{E} - \overline{A}, \tag{19}$$

where $\overline{E}$ is the weighted average of individual generalization errors and $\overline{A}$ is the weighted average of individual ambiguities.

Eq. (19) describes the relationship between ensemble ambiguity and ensemble generalization error on a validation dataset. Experiments showed that maximizing the ensemble ambiguity improves the ensemble generalization [3]. Moreover, it can be easily noted from Eq. (19) that the ensemble generalization error is always smaller than the average of ensemble components' errors i.e. $E \leq \overline{E}$. This emphasizes that network ensembles perform better than the average performance of their individual networks and they perform equally when the ensemble components are strongly biased toward common region in the input space [3]. Note that this is true under the assumption of mutual independence of individual errors with zero mean [2]. Alternatively, the ensemble generalization error is derived from Eq. (19) in terms of the distances among ensemble components' outputs, called *ensemble diversity*, instead of using the ambiguity term, as follows:

$$E = \sum_{i=1}^{N} w_i E_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j D_{ij}, \tag{20}$$

$$E = \overline{E} - \overline{D}, \tag{21}$$

where E_i is the generalization error of the ith component, given by Eq. (9); $\overline{D}$ is the ensemble diversity; and D_{ij} is the distance between the ith component and the jth component, given by

$$D_{ij} = \sum_k (f_i(x_k) - f_j(x_k))^2. \tag{22}$$

From Eq. (21) it is obvious that the ensemble accuracy increases in proportional with the distances among its components' outputs. However, since the ensemble components' outputs are not independent and the general mean of individual errors is not zero, a symmetric *correlation matrix* between ensemble components' outputs was defined to ensure ensemble components disagreement [2,11], as follows:

$$C_{ij} = \sum_k (f_i(x_k) - t_k)(f_j(x_k) - t_k); \quad i,j = 1, \ldots, N. \tag{23}$$

This yields the following target function:

$$\hat{E} = \sum_{i,j} w_i w_j C_{ij} \tag{24}$$

subject to

$$\sum_j w_j = 1 \quad \text{and} \quad 0 \leq w_j \leq 1.$$

From Eq. (24), it is clearly shown that minimizing the target function $\hat{E}$ ensures great variance among ensemble constitutes.

In practice, the diversity, ambiguity and correlation matrix are metrics used to properly select the ensemble base models and guarantee maximum disagreement within ensemble. Unlike traditional ensemble methods that use all base models in a pool [1,3,5,6], an ensemble of few base models wisely selected from the pool could perform better and represent the whole pool in a compact structure [2,8,10,11]. Most proposed methods in the literature use genetic algorithms (GAs) along with the disagreement metrics to find the best ensemble [8,10,11]. However, specifying the putative ensemble size is still an open problem. It is worthy noted that GASEN algorithm, proposed by Zhou et al. [11], encodes the ensemble averaging weights as floating point chromosomes and selects the ensemble components solely based on the evolved weights. This encoding scheme lacks proof of effectiveness and logic.

3. Evolutionary selection of base learner models

3.1. Problem formulation

Taking the advantage of fast generation of ELM models, our ensemble method utilizes the genetic algorithm optimization capabilities to select a subset of models from a pool of ELM models and uses simple approach to confine the ensemble size. Algorithm 2 shows the pseudocode of our proposed approach. Given a pool of P neural networks (SLFN) already learned with ELM on a fixed training dataset $D_t = \{(x_i, t_i) \in \mathbf{R}^n \times \mathbf{R} (i = 1, \ldots, M)\}$, our algorithm increases the ensemble size K as long as expanding the ensemble is beneficial. In other words, our algorithm first finds the best ensemble of two ELM models Ens_2 then it finds the best ensemble of three ELM models Ens_3. When the generalization error e_3 of Ens_3 is smaller than the generalization error e_2 of Ens_2, the search scope is enlarged to find the best ensemble of four ELM models Ens_4 and so on. The ensemble size increment stops and consequently the algorithm converges when the ensemble generalization error increases due to expansion i.e. the algorithm reaches a global minimum for the ensemble cost function or the maximum size of possible ensembling is attained. For each iteration, we use Genetic Algorithm (GA) to search for the best ensemble of size K that *minimizes* the fitness function given by Eq. (20). Moreover, the GA individuals are evaluated on an unseen validation dataset $D_v = \{(x_i, t_i) \in \mathbf{R}^n \times \mathbf{R} (i = 1, \ldots, L)\}$. The evolutionary selection process is thoroughly explained in Section 3.2.

Algorithm 2. Evolutionary selection based ensemble of ELM models.

```
Require: Training dataset $D_t$, validation dataset $D_v$, ELM pool size $P$, maximum ensemble size $S$, the number of hidden neurons of the base ELM models $\tilde{N}$, GA population size $GA_p$, maximum number of GA generations $GA_g$, GA mutation rate $GA_m$, GA crossover rate $GA_c$.
Ensure: EE-ELM model.
1: for $j = 1$ to $P$ do
2:     $Pool[j] \leftarrow$ Build ELM model invoking Algorithm 1 with $(D_t, \tilde{N})$.
3: end for
4: $K \leftarrow 2$
5: $e_1 \leftarrow \infty$
6: while $K \leq S$ do
7:     $Ens_K \leftarrow$ Find the best ensemble of $K$ models drawn from $Pool$ using GA (Algorithm 3).
8:     $e_K \leftarrow$ Evaluate $Ens_K$ on $D_v$.
9: if $e_K \leq e_{K-1}$ then
10:         $K \leftarrow K+1$
11:     else
12:         break
13:     end if
14: end while
15: return $Ens_{K-1}$
```

3.2. Algorithm description

Genetic algorithms (GAs), proposed by Holland [21], are search algorithms inspired from the natural evolution laws: selection and reproduction. GAs aim at finding heuristic solutions for optimization problems. Several neural network ensembles utilized genetic algorithms to select component networks from a population of networks [8,11] or maximize the diversity of component networks and ensemble prediction accuracy [10]. However, using genetic algorithms for both selecting ensemble components and ensuring high diversity has not been tried before. A genetic algorithm evolves an initial population of chromosomes over a finite number of generations until it converges toward a solution which is expected to be optimal. Each population represents a set of possible solutions in the problem solutions space. Instantiating individuals of evolutionary population with problem solutions requires an encoding and evaluation schemes that define how to map problem solutions to genetic chromosomes and vice versa. Each chromosome consists of a sequence of genes encoded as floating or binary objects that denote solution features. Mapping chromosomes to problem solutions is performed through a fitness function which considers chromosome characteristics to measure its goodness. Superior individuals are only selected to mate and survive for the next generation. A pair of selective individuals reproduce new offsprings through two genetic operations: crossover and mutation. Crossover operation combines the genetic material of two good individuals to produce two new individuals while mutation operator only alters particular genes in certain chromosomes to maintain genetic diversity of chromosomes.

As for our ensemble method, genetic algorithm is used to select the best subset of ELM models from a pool in such a way that reflects better performance than the performance of any other ensemble. The subset cardinality K is fixed and set by our main algorithm. Practically, the GA helps to reduce ensemble size and avoid thorough search in the networks subsets space. In the next subsections, we explain the implementation of GA in order to fit our proposed ensemble approach. Fig. 1 and Algorithm 3 illustrate the whole evolutionary process conformed in our learning system.

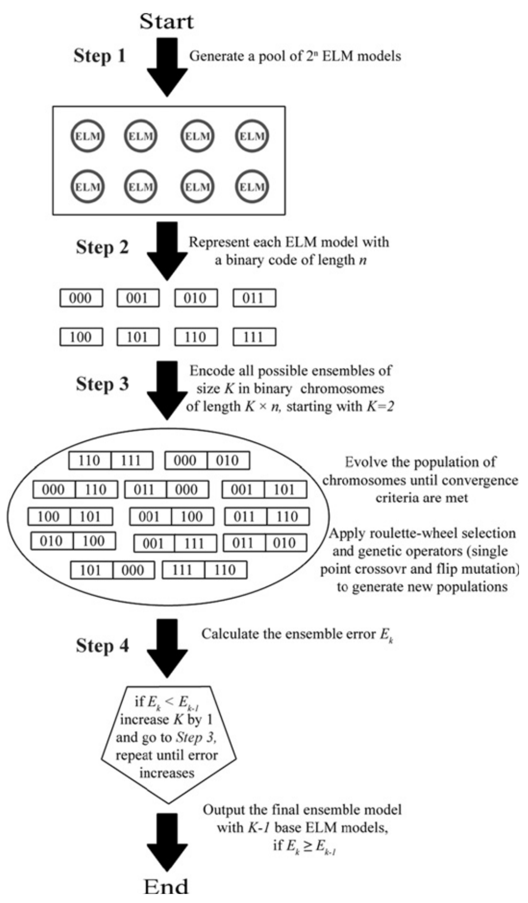
Start
Step 1
Generate a pool of 2ⁿ ELM models
ELM
ELM
ELM
ELM
ELM
ELM
ELM
ELM
Step 2
Represent each ELM model with a binary code of length n
000
001
010
011
100
101
110
111
Step 3
Encode all possible ensembles of size K in binary chromosomes of length K × n, starting with K=2
110 111
000 010
000 110
011 000
001 101
100 101
001 100
011 110
010 100
001 111
011 010
101 000
111 110
Evolve the population of chromosomes until convergence criteria are met
Apply roulette-wheel selection and genetic operators (single point crossovr and flip mutation) to generate new populations
Step 4
Calculate the ensemble error E_k
if E_k < E_k-1 increase K by 1 and go to Step 3, repeat until error increases
Output the final ensemble model with K-1 base ELM models, if E_k ≥ E_k-1
End

Fig. 1. The general scheme of our EE-ELM algorithm.

Algorithm 3. Selecting the best ensemble of K ELM models from a pool.

```
Require: Pool of ELM models, putative ensemble size $K$, $GA_p$, $GA_g$, $GA_m$, $GA_c$, fitness function $f(x)$.
Ensure: Optimal ELM ensemble of size $K$.
1: $n \leftarrow$ Calculate number of binary bits needed to encode pool size $P$.
2: Initialize a random population of $GA_p$ binary chromosomes (of lengths $n \times K$) that encode putative ensembles, denote it $P_{current}$.
3: $i \leftarrow 1$
4: while ($i \le GA_g$) do
5:     Evaluate all individuals in $P_{current}$ using $f(x)$.
6:     Select fit individuals from $P_{current}$ and make a breeding pool.
7:     Generate a new population $P_{new}$ using genetic operators considering $GA_m$ and $GA_c$.
8:     Evaluate all individuals in $P_{new}$ using $f(x)$.
9:     Replace least-fit individuals in $P_{new}$ with best-fit individuals from $P_{current}$.
10:    $P_{current} \leftarrow P_{new}$
11:    if $\min_{x \in P_{current}} f(x) = \max_{x \in P_{current}} f(x)$ then
12:        break
13:    end if
14:    $i \leftarrow i+1$
15: end while
16: return Best-fit individual of $P_{current}$.
```

3.2.1. *Encoding ensembles*

Since the purpose of using genetic algorithms is to find the best ELM ensemble of size K, chromosomes representation must stick to this property in the evolved solutions. To maintain this size property in our encoding scheme, the pool size is enforced to be a power of 2 i.e. $P = 2^n$ where $n = 3, 4, \ldots$ so that each ELM model in the pool can be encoded in a binary sequence of size n. Subsequently, each chromosome is represented by a binary string of length $K \times n$ which maps to a possible ensemble in the ELM ensembles space of size K. For the sake of clarity, assume a pool of eight ELM models and $K=2$, the pool models are identified by 3-bit sequences as follows: $\{net_1 = 000,\ net_2 = 0001,\ net_3 = 010,\ net_4 = 011,\ net_5 = 100,\ net_6 = 101,\ net_7 = 110,\ net_8 = 111\}$ and the chromosomes of one population are randomly constructed as follows: $\{ch_1 = 000101,\ ch_2 = 111011, \ldots, ch_{GA_p} = 111011\}$. For each chromosome in this population, the first three bits encode for the first base model and the next bits encode for the second base model.

3.2.2. *Fitness function*

To select the parents that produce offsprings for next generation, we use a fitness function to evaluate population individuals and a selection procedure to draw good ones. The fitness function is the same as the ensemble generalization error given in Eq. (20) where uniform weights are set for ensemble components i.e. $f(x) = (1/K)\sum_{i=1}^{K} E_i - (1/2K^2)\sum_{i=1}^{K}\sum_{j=1}^{K} D_{ij}$. To evaluate a chromosome, it is decomposed into its constituting ELM models of which an ensemble is created. For instance, given a chromosome 101011, $P=8$ and $K=2$, the corresponding ensemble is composed of the ELM model number 5(101) and ELM model number 4(011) from the base learners pool. Our algorithm seeks the optimal subset of models that minimizes the fitness function $f(x)$. It is clearly noticed that the optimal solution will have less average individual error and higher average diversity.

After scoring all the chromosomes of existing population, individuals are repeatedly selected for breeding until the new population is saturated. We use fitness proportionate selection (Roulette–Wheel) procedure for this purpose. First, the fitness values of all individuals are normalized to obtain a selection probability distribution $P_i = f_i / \sum_{j=1}^{GA_p} f_j$ $(i=1,\ldots,GA_p)$ where f_i is the fitness of the ith individual. Second, a cumulative distribution function $F(P_i) = \sum_{P_j \le P_i} P_j$ is computed over P_i; population individuals are sorted by descending order of selection probabilities P_i and cumulative scores $F(P_i)$ are assigned to the ordered individuals by summing up P_i with all preceding selection probabilities. Eventually, a random value $R \in [0,1]$ is generated and the first individual whose cumulative score $F(P_i)$ is greater than R is selected. It is worthy mentioned that chromosomes encoding ensembles with duplicated ELM models are considered bad individuals and therefore given less chances to be selected for reproduction. If it happens, the fitness score is multiplied by the number of repeated models.

3.2.3. Reproduction

Crossover and mutation are the basic genetic operations for breeding new generation. In this paper, two selected parents are recombined using *single point* crossover in which all genes beyond a randomly pointed gene are swapped between the two parent chromosomes resulting in two offsprings. This genetic variation occurs with a crossover probability GA_c that decides whether parents will stay themselves for the next generation without change or they will exchange genetic materials. After applying crossover, the offspring chromosomes are vulnerable to be changed by mutation operator. We use *flip-flop single point* mutation which turns over each bit with a probability GA_m. Rationally, these two genetic operators cause the average fitness of individuals to be improved in the next generation since fit individuals are more likely to be selected for breeding. However, they may badly affect the new offsprings and generate undesired chromosomes i.e. models may appear repeatedly in the same ensemble. For example, given two chromosomes {101100, 100101}, $P=8$ and $K=2$, if crossover occurs at the second bit (from right), the new offsprings will be {101101,100100}. Similarly, given a chromosome 101111, if mutation happens at the second bit, the new offspring will be 101101. Under such circumstances, the chromosomes are treated as bad individuals for evolution.

3.2.4. Selecting elitists

After generating new population using selection and genetic operations, some selective individuals in the old population might be more informative than poor individuals in the new population. In fact, the evolution process becomes more stable and converges earlier if superior individuals are allowed to stay in the next generations without any genetic variation. The standard elitism approach replaces the worst N_e individuals in the new population with the best N_e individuals from the old population where $N_e < GA_p$. In our algorithm, however, after the reproduction phase, old and new population individuals are combined together and then the best-fit GA_p individuals in this pool are selected to constitute new population. This mechanism allows smooth improvement in the average fitness of chromosomes through sequential generations.

4. Performance evaluation

4.1. Datasets

Twelve datasets have been used to analyze the performance of our method on regression problems. Among those datasets, 10 are used for functions approximation and two are real datasets used for predicting continuous variables. A summary of all datasets along with their target functions are presented in Table 1.

Table 1
Summary of datasets used in our study.

Dataset	Function	Variables	Size
2-d Mex. Hat	$y = \mathrm{sinc}\lvert x\rvert = \frac{\sin\lvert x\rvert}{\lvert x\rvert}$	$x \sim \mathbf{U}[-2\pi, 2\pi]$	5000
3-d Mex. Hat	$y = \mathrm{sinc}\sqrt{x_1^2+x_2^2} = \frac{\sin\sqrt{x_1^2+x_2^2}}{\sqrt{x_1^2+x_2^2}}$	$x_i \sim \mathbf{U}[-4\pi, 4\pi]$	3000
Friedman #1	$y = 10\sin(\pi x_1 x_2) + 20(x_3-0.5)^2 + 10x_4 + 5x_5$	$x_i \sim \mathbf{U}[0,1]$	5000
Friedman #2	$y = \sqrt{x_1^2 + \left(x_2x_3 - \left(\frac{1}{x_2x_4}\right)\right)^2}$	$x_1 \sim \mathbf{U}[0,100]$ $x_2 \sim \mathbf{U}[40\pi, 560\pi]$ $x_3 \sim \mathbf{U}[0,1]$ $x_4 \sim \mathbf{U}[1,11]$	5000
Friedman #3	$y = \tan^{-1}\frac{x_2x_3 - \frac{1}{x_2x_4}}{x_1}$	$x_1 \sim \mathbf{U}[0,100]$ $x_2 \sim \mathbf{U}[40\pi, 560\pi]$ $x_3 \sim \mathbf{U}[0,1]$ $x_4 \sim \mathbf{U}[1,11]$	3000
Gabor	$y = \frac{\pi}{2}\exp[-2(x_1^2+x_2^2)]\cos[2\pi(x_1+x_2)]$	$x_i \sim \mathbf{U}[0,1]$	3000
Multi	$y = 0.79 + 1.27x_1x_2 + 1.56x_1x_4 + 3.42x_2x_5 + 2.06x_3x_4x_5$	$x_i \sim \mathbf{U}[0,1]$	4000
Plane	$y = 0.6x_1 + 0.3x_2$	$x_i \sim \mathbf{U}[0,1]$	1000
Polynomial	$y = 1 + 2x + 3x^2 + 4x^3 + 5x^4$	$x \sim \mathbf{U}[0,1]$	3000
sinC	$y = \frac{\sin(x)}{x}$	$x_i \sim \mathbf{U}[0, 2\pi]$	3000
Cal. Housing	House prices estimation Eight continuous inputs $[x_1,\ldots,x_8]$	$x_i \sim \mathbf{U}[0,1]$	20 640
Ozone	Ozone concentration estimation Eight continuous inputs $[x_1,\ldots,x_8]$	$x_i \sim \mathbf{U}[0,1]$	330

4.1.1. Functions approximation

The same 10 datasets used by Zhou et al. [11] have been selected to justify the function approximation abilities of our ensemble method. These functions are 2-d Mexican Hat, 3-d Mexican Hat, Friedman #1, Friedman #2, Friedman #3, Gabor, Multi, Plane, Polynomial, and SinC. They have been widely used in testing various learning algorithms. The domain variables of these functions are randomly generated from a uniform distribution between x and y, denoted as $U[x,y]$ in Table 1. The size of each dataset and the constraints on its variables are listed in Table 1. The input and output variables of Friedman#2 and Friedman#3 were normalized for ease of computations and comparison. A random noise in the range $[-0.2,0.2]$ was added to all examples during training in order to reliably measure the learning capabilities of tested methods on function approximation problems. However, testing and validation examples are noise-free.

4.1.2. California Housing

California Housing is a dataset obtained from the StatLib repository and contains 20 640 observations for predicting the price of houses in California. Each observation is defined on nine variables which consist of eight continuous inputs (median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude) and one continuous output (median house value). The input and output features were normalized. This dataset was used to test the ELM performance on function approximation [12].

4.1.3. Ozone

Ozone is a dataset obtained from the Department of Statistics, University of California and contains 360 meteorological cases to study the Ozone pollution at Upland in California. The raw dataset contains 12 continuous input variables and one dependent output variable which is the ozone concentration measured by ppm (parts per million volume air concentration). We dropped four irrelevant input variables and removed all cases that had missing values. As a result, we ended up with 330 observations and each with eight input variables (Sandburg air force base temperature, inversion base height, Daggett pressure gradient, visibility, Vandenburg 500 mbar height, humidity, inversion base temperature and wind speed). This information is given through one year and measured every single day. Data normalization is also applied.

4.2. Parameters setting and experimental protocol

Different methods have been widely presented in the literature to build neural network ensembles. In this paper, we compare the performance of our algorithm with three popular ensemble approaches: simple average ensemble, bagging, and boosting. *Simple average* ensemble [1] basically trains population of base models on a fixed dataset then combines them all together using uniform averaging while *bagging* [4] uses bootstrap sampling to train the ensemble base models on different datasets though. On the other hand, *boosting* resamples the training datasets of ensemble components according to an adaptive probability distribution while the averaging weights are altered to be in reverse proportion to the error of corresponding components. Although there are different variations of boosting for regression problems, we use Adaboost.RT [20], as a recent boosting method, to compare boosting with our ensemble approach. In all methods we use ELM as a base learner to populate ensemble individuals.

Experimentally, different datasets and different ensemble methods require different parameters to be optimized. There are six parameters to be tuned for our ensemble method EE-ELM (GA_g, GA_p,GA_c, GA_m, base learner size, and pool size), two parameters for simple average ensemble (base learner size and ensemble size), three parameters for bagging (base learner size, ensemble size, and bootstrap sampling size), and four parameters for Adaboost.RT (base learner size, ensemble size, subsampling size, and threshold ϕ for weighting incorrect predictions). The pool size of our method and the ensemble size in other methods have been set equally to $2^7=128$ base models in order for good comparison between all methods. Different pool sizes have been tried and 128 has been found fair enough for our study. The number of hidden neurons in base models has been selected for each dataset after exhaustive search in the range [5,100]. In fact, pruning ELM-based methods can be used to optimize base learner size e.g. OP-ELM [22]. Similarly, the threshold parameter ϕ has been searched for each dataset in the range [0.01,0.5]. In bagging and boosting, roughly 40% of the training set has been frequently resampled to constitute training examples for ensemble components. In addition to that, crossover rate GA_c, mutation rate GA_m, population size GA_p, and maximum number of generations GA_g of the genetic algorithm have been fixed identically for all datasets to 0.95, 0.02, 200, and 150, respectively. It is worthy mentioned that the simple average ensemble was constructed from the initial ELM models pool of our EE-ELM algorithm. All parameters can be tuned further to obtain better accuracies. However, we are only highlighting the selection property and the relative performance of EE-ELM against other ensemble methods. Table 2 summarizes the parameters values in our experiments for each dataset.

The ensemble methods in this paper were assessed using 10-fold cross-validation procedure in which each dataset is randomly partitioned into 10 subsets that initiate 10 runs for each experiment. For each run, one subset is used for validation, one subset is used for testing and the remaining subsets are used for training. The performance computations from all folds are collected and averaged over 10. Due to the randomness of ELM and cross-validation partitioning and for more reliable results, each experiment was simulated 10 times and the results from all simulations were averaged. Note that four-fold cross-validation was used for the Ozone dataset only. In all ensemble methods, the same number of hidden neurons has been used for all ensemble individual networks. The simulations were executed on a computer of two processors, each one with 2.0 GHz frequency, and 4.0 GB of RAM.

4.3. Performance evaluation metrics

To evaluate the performance of different ensemble methods, we calculate the Root Mean Square Error (RMSE) of ensemble

Table 2
Summary of parameters setting for each dataset.

Dataset	Net size[a]	ϕ[b]	Sample size[c]
2-d Mex. Hat	10	0.1	2000
3-d Mex. Hat	100	0.4	1200
Friedman #1	100	0.01	2000
Friedman #2	20	0.1	2000
Friedman #3	95	0.1	1200
Gabor	45	0.1	1200
Multi	60	0.01	1600
Plane	5	0.1	400
Polynomial	5	0.01	1200
sinC	5	0.1	1200
Cal. Housing	60	0.1	8000
Ozone	25	0.1	75

[a] The size of ensemble base models.
[b] The threshold parameter in Adaboost.RT method.
[c] The sampling size in Bagging and Adaboost.RT.

Table 3
Comparison of testing RMSE of EE-ELM and other ensembles.

Dataset	Ensemble method	Ensemble RMSE		Individual RMSE		
		Mean	Std. Dev.	Mean	Std. Dev.	Best
2-d Mex. Hat	Adaboost.RT	0.0062	0.0003	0.0116	0.0026	0.0058
	Bagging	0.0060	0.0004	0.0104	0.0022	0.0054
	Simple	0.0058	0.0003	0.0063	0.0012	0.0050
	Random-*K*	0.0060	0.0003	0.0063	0.0006	0.0056
	Top-*K*	0.0052	0.0004	0.0052	0.0001	0.0050
	EE-ELM	0.0049	0.0004	0.0059	0.0007	0.0052
3-d Mex. Hat	Adaboost.RT	0.0666	0.0003	0.1105	0.0205	0.0807
	Bagging	0.0686	0.0006	0.0874	0.0049	0.0752
	Simple	0.0691	0.0006	0.0764	0.0034	0.0669
	Random-*K*	0.0704	0.0007	0.0760	0.0027	0.0722
	Top-*K*	0.0648	0.0008	0.0708	0.0025	0.0674
	EE-ELM	0.0639	0.0006	0.0711	0.0027	0.0675
Friedman #1	Adaboost.RT	0.1635	0.0021	0.5299	0.1547	0.2562
	Bagging	0.2036	0.0010	0.3012	0.0350	0.2126
	Simple	0.2045	0.0009	0.2871	0.0327	0.2051
	Random-*K*	0.2192	0.0061	0.2862	0.0304	0.2401
	Top-*K*	0.1579	0.0020	0.2276	0.0148	0.2056
	EE-ELM	0.1554	0.0022	0.2322	0.0185	0.2056
Friedman #2	Adaboost.RT	0.0090	0.0003	0.0227	0.0053	0.0134
	Bagging	0.0089	0.0003	0.0212	0.0053	0.0124
	Simple	0.0087	0.0005	0.0173	0.0057	0.0092
	Random-*K*	0.0106	0.0008	0.0170	0.0049	0.0112
	Top-*K*	0.0080	0.0004	0.0102	0.0006	0.0092
	EE-ELM	0.0069	0.0005	0.0138	0.0031	0.0100
Friedman #3	Adaboost.RT	0.0881	0.0018	0.2672	0.1584	0.0749
	Bagging	0.0602	0.0007	0.0776	0.0057	0.0652
	Simple	0.0605	0.0009	0.0644	0.0027	0.0584
	Random-*K*	0.0615	0.0009	0.0645	0.0022	0.0614
	Top-*K*	0.0598	0.0009	0.0622	0.0017	0.0598
	EE-ELM	0.0598	0.0009	0.0627	0.0021	0.0599
Gabor	Adaboost.RT	0.0418	0.0029	0.0805	0.0701	0.0278
	Bagging	0.0218	0.0005	0.0364	0.0068	0.0235
	Simple	0.0213	0.0005	0.0256	0.0059	0.0161
	Random-*K*	0.0227	0.0010	0.0257	0.0049	0.0194
	Top-*K*	0.0163	0.0006	0.0169	0.0006	0.0162
	EE-ELM	0.0161	0.0005	0.0180	0.0015	0.0164
Multi	Adaboost.RT	0.0165	0.0003	0.0373	0.0039	0.0282
	Bagging	0.0168	0.0003	0.0335	0.0033	0.0258
	Simple	0.0165	0.0003	0.0236	0.0027	0.0183
	Random-*K*	0.0176	0.0003	0.0236	0.0024	0.0204
	Top-*K*	0.0160	0.0003	0.0201	0.0010	0.0185
	EE-ELM	0.0154	0.0002	0.0223	0.0021	0.0193
Plane	Adaboost.RT	0.0088	0.0010	0.0183	0.0058	0.0063
	Bagging	0.0083	0.0009	0.0156	0.0048	0.0053
	Simple	0.0084	0.0007	0.0096	0.0021	0.0065
	Random-*K*	0.0093	0.0008	0.0101	0.0019	0.0081
	Top-*K*	0.0071	0.0008	0.0070	0.0003	0.0066
	EE-ELM	0.0070	0.0007	0.0086	0.0013	0.0072
Polynomial	Adaboost.RT	0.0058	0.0005	0.0139	0.0068	0.0041
	Bagging	0.0056	0.0006	0.0105	0.0052	0.0033
	Simple	0.0055	0.0006	0.0072	0.0054	0.0048
	Random-*K*	0.0071	0.0016	0.0072	0.0024	0.0052
	Top-*K*	0.0050	0.0005	0.0048	0.0000	0.0048
	EE-ELM	0.0047	0.0005	0.0065	0.0013	0.0052
sinC	Adaboost.RT	0.0076	0.0004	0.0162	0.0076	0.0058
	Bagging	0.0071	0.0002	0.0071	0.0071	0.0050
	Simple	0.0071	0.0005	0.0111	0.0072	0.0053
	Random-*K*	0.0096	0.0015	0.0115	0.0051	0.0070
	Top-*K*	0.0054	0.0006	0.0055	0.0002	0.0053
	EE-ELM	0.0047	0.0006	0.0088	0.0025	0.0063
Cal. Housing	Adaboost.RT	0.1289	0.0004	0.1408	0.0107	0.1289
	Bagging	0.1284	0.0003	0.1336	0.0050	0.1275
	Simple	0.1282	0.0002	0.1312	0.0027	0.1273
	Random-*K*	0.1286	0.0002	0.1309	0.0020	0.1286
	Top-*K*	0.1281	0.0005	0.1302	0.0018	0.1281
	EE-ELM	0.1280	0.0007	0.1306	0.0022	0.1283
Ozone	Adaboost.RT	0.1117	0.0017	0.1522	0.0227	0.1144

Table 3 (*continued*)

Dataset	Ensemble method	Ensemble RMSE		Individual RMSE		
		Mean	Std. Dev.	Mean	Std. Dev.	Best
	Bagging	0.1120	0.0022	0.1519	0.0223	0.1146
	Simple	0.1103	0.0029	0.1157	0.0044	0.1057
	Random-*K*	0.1111	0.0026	0.1150	0.0037	0.1102
	Top-*K*	0.1102	0.0024	0.1134	0.0029	0.1098
	EE-ELM	0.1094	0.0029	0.1138	0.0033	0.1095

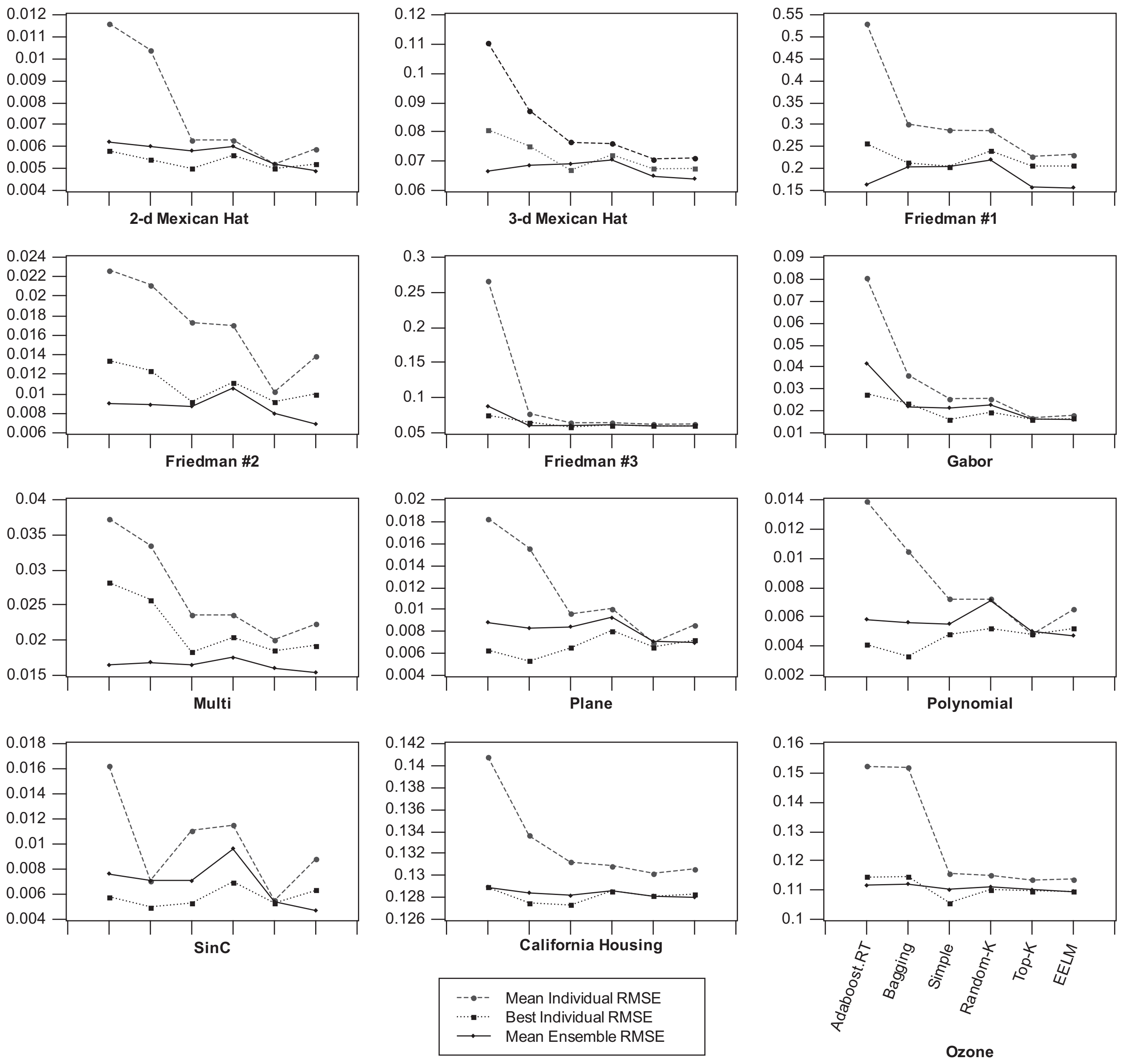

Fig. 2. Comparison of testing RMSE of EE-ELM and other methods. These graphs also compare the mean ensemble RMSE with the mean and least RMSE of ensemble base models.

using the following equation:

$$E_{ens} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left(\sum_{j=1}^{N} w_j f_j(x_i) - t_i\right)^2}, \tag{25}$$

where M is the number of tested examples and N is the number of ensemble component networks. To emphasize the importance of ensemble learning and deeply investigate our algorithm characteristics, some statistics (mean, standard deviation, and least) are also calculated for the RMSE of individual ELM models which is given by

$$E(j) = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(f_j(x_i) - t_i)^2}. \tag{26}$$

Apparently, the smaller the values of E_{ens} and $E(j)$, the better the estimated performance of ensemble and base models, respectively. Moreover, we compare the generalization error of our EE-ELM method with the *top-K* and *random-K* ensembles to prove the selection ability of EE-ELM. Top-*K* ensemble is composed of the first top *K* base models in a pool which are selected according to their validation RMSE while random-*K* ensemble is composed of *K* models randomly selected from a pool. In both ensembles, *K* is determined by our algorithm.

On the other hand, we investigate the robustness of our algorithm in terms of base learners capacity because it plays an important role in improving the generalization capabilities of ensemble component models and consequently supporting the ensemble learning. Practically, we tried different sizes of ELM models from the range [5, 100] hidden neurons to deeply understand the influence of base learners structure on our EE-ELM algorithm performance.

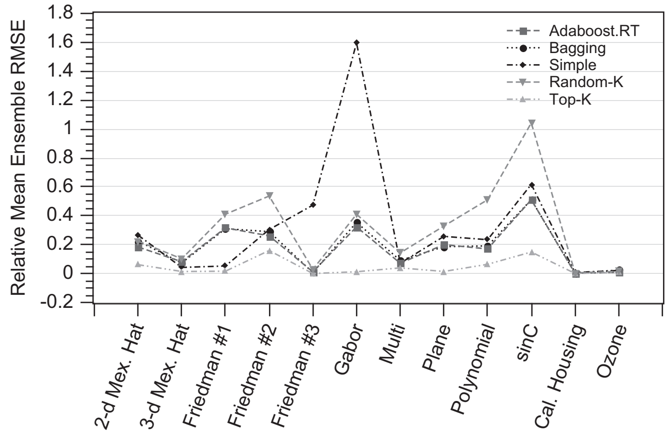

Fig. 3. The relative mean RMSE of Adaboost.RT, bagging, simple, random-*K* and top-*K* ensembles in proportion to EE-ELM mean RMSE.

4.4. Results and discussion

Table 3 and Figs. 2 and 3 compare the performance of our method against Adaboost.RT, bagging, simple average, random-*K* and top-*K* ensembles using the combination of parameter values given in Table 2. In Table 3, mean indicates the average results over 10-fold cross validation and 10 runs and Std. Dev. indicates the standard deviation of the results from the 10×10 experiments. It is shown that the generalization error of our EE-ELM ensemble is less than the generalization error of other ensembles in all function approximation problems (2-d Mexican Hat, 3-d Mexican Hat, Friedman #1, Friedman #2, Friedman #3, Gabor,

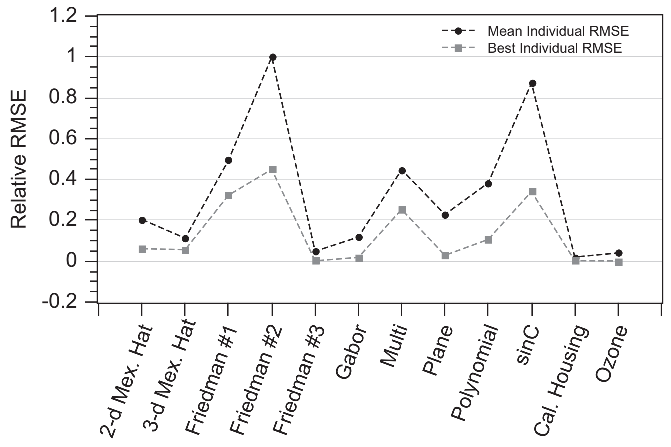

Fig. 4. The relative mean RMSE and relative best RMSE of EE-ELM individual models in proportion to EE-ELM mean RMSE.

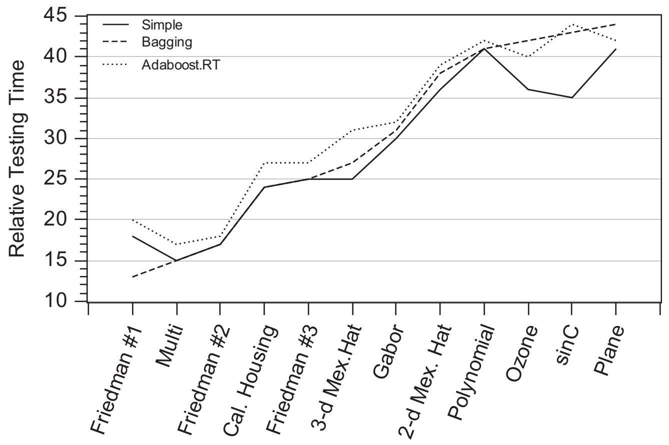

Fig. 6. The relative average testing time of simple, bagging, Adaboost.RT ensembles in proportion to EE-ELM testing time.

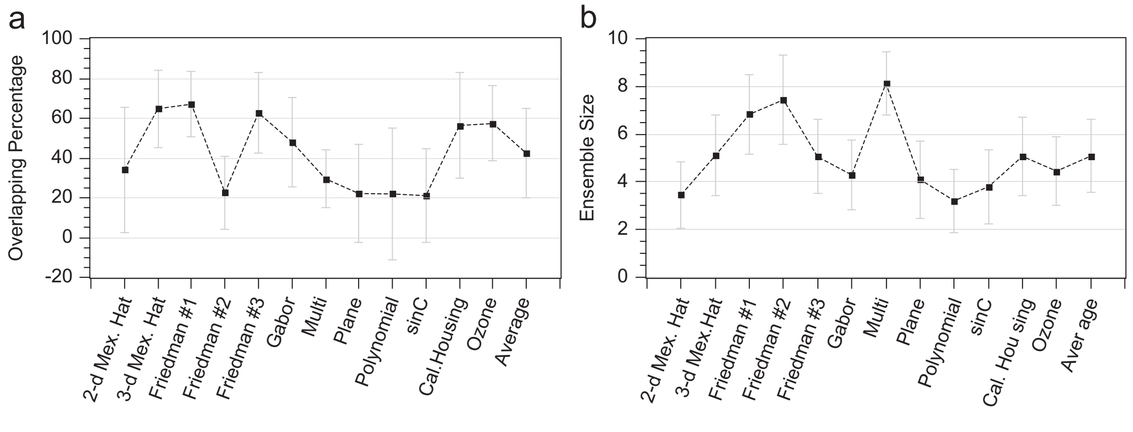

Fig. 5. (a) Mean and standard deviation of overlapping percentage (%) between top-*K* base models and EE-ELM base models. (b) Mean and standard deviation of the sizes of EE-ELM models. The last tick on *x*-axis "Average" denotes the average values overall datasets. (a) Overlapping percentages. (b) Ensemble sizes.

Multi, Plane, Polynomial, and SinC in Fig. 2). Moreover, the performance of our method is better than other ensemble methods when tested on real datasets (California Housing and Ozone in Fig. 2). Interestingly, the RMSE of EE-ELM ensemble is less than the RMSEs of top-*K* and random-*K* ensembles for all datasets. This ensures that our algorithm properly selects the ensemble individuals and it is not biased by the base models generalization errors (see Top-*K* and Random-*K* curves in Fig. 3). From Table 3, by comparing the mean individual RMSE and the mean ensemble RMSE of simple ensemble with the mean ensemble RMSE of our EE-ELM ensemble, we conclude that our ensembling approach outperforms the traditional ELM algorithm and draws good representative ensemble for the initial ELM models pool. It is also shown that EE-ELM is quite stable and more reliable than traditional ELM (compare Std. Dev. values of simple and EE-ELM rows in Table 3) and comparable in reliability with other ensemble methods (compare Std. Dev. values of Ensemble RMSE section in Table 3).

Fig. 4 depicts that EE-ELM ensemble outperforms the best individual model in the final ensemble and intuitively holds RMSE less than the mean individual RMSE. Moreover, Fig. 5(a) shows that the best *K* performed models (on validation dataset) in the initial ELM models pool do not necessarily constitute the whole EE-ELM ensemble. The overlapping between the models of EE-ELM ensemble and the models of top-*K* ensemble does not exceed 70% in most cases and in some cases (Friedman #2, Plane, Polynomial and SinC) it is around 20%. Briefly, the average overlapping percentage over all datasets is less than 50% of the top-*K* models. The diversity term in

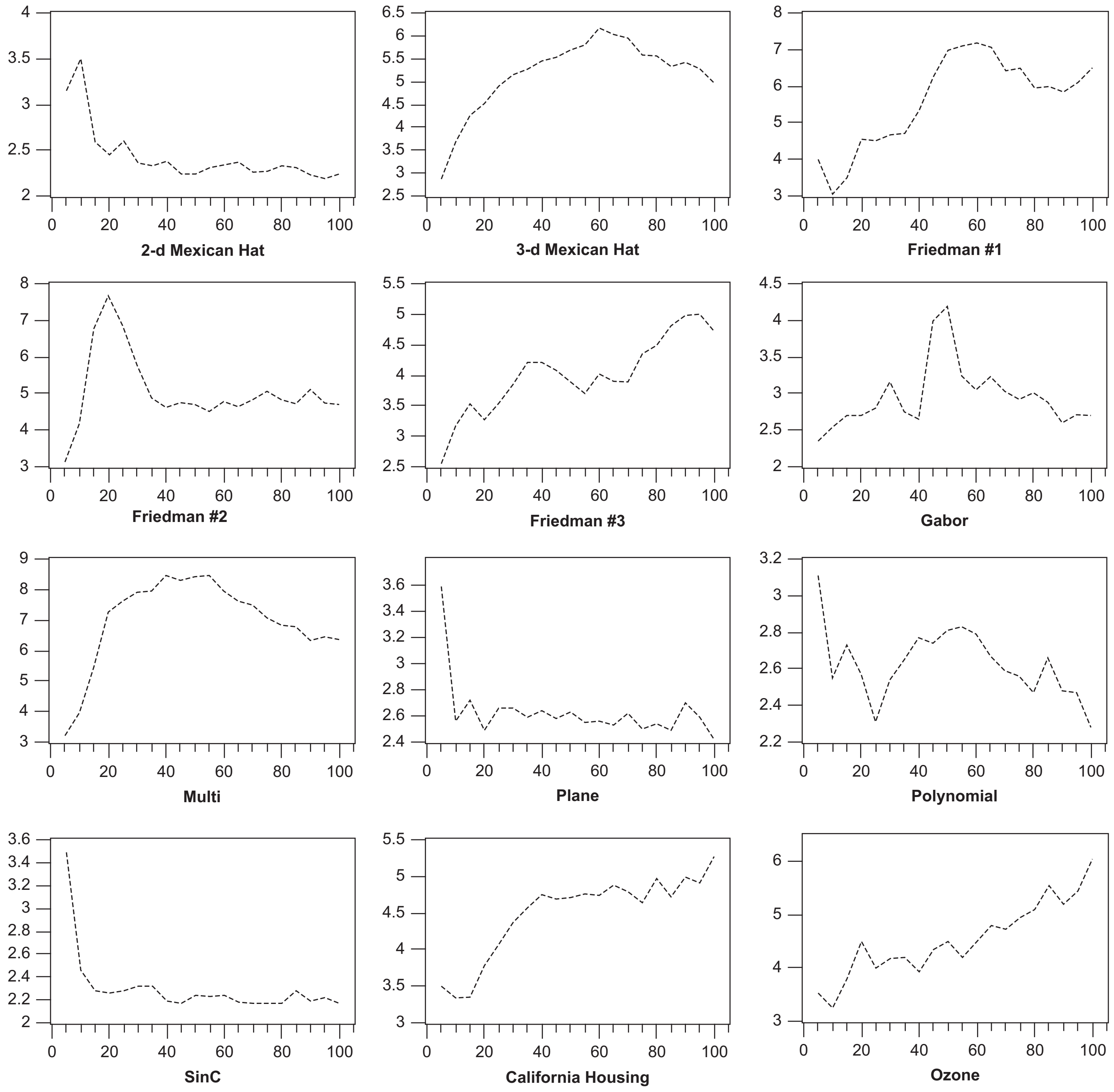

Fig. 7. Average sizes of ensembles created by our EE-ELM algorithm for different sizes of base learners for all datasets.

the fitness function helps our algorithm to select models which might be weak when they are single but give better performance when coupled with other models. Note that the standard deviation of overlapping percentage is greater than the mean overlapping percentage in three datasets (see the error bars of Plane, Polynomial and SinC in Fig. 5(a)). This means that in most experiments there was no overlapping between the top-*K* base models and the EE-ELM base models.

From Fig. 5(b), it is easily noticed that our algorithm reduces the number of ensemble components by roughly 95% comparing with simple, bagging and adaboost ensembles which lack the selection property. Fig. 5(b) also shows that the average size of ensembles is less than five component models in most cases and not more than eight components in other cases (Friedman #1, Friedman #2 and Multi). In summary, the average size of EE-ELM models generated by our algorithm over all datasets is 5.08 ± 1.54.

Moreover, Fig. 6 shows that EE-ELM is much faster than other ensemble methods in practice. The testing time of EE-ELM is at least 10 times lower than the testing time of other ensemble methods and for some datasets it reaches up to 45 times. In fact, the great difference in testing time is due to the great reduction of ensemble size in EE-ELM resulting in very high throughput.

Eventually, Figs. 7 and 8 show the performance characteristics of our EE-ELM algorithm for different sizes of base learners. It is illustrated in Fig. 7 that the EE-ELM model size steadily changes

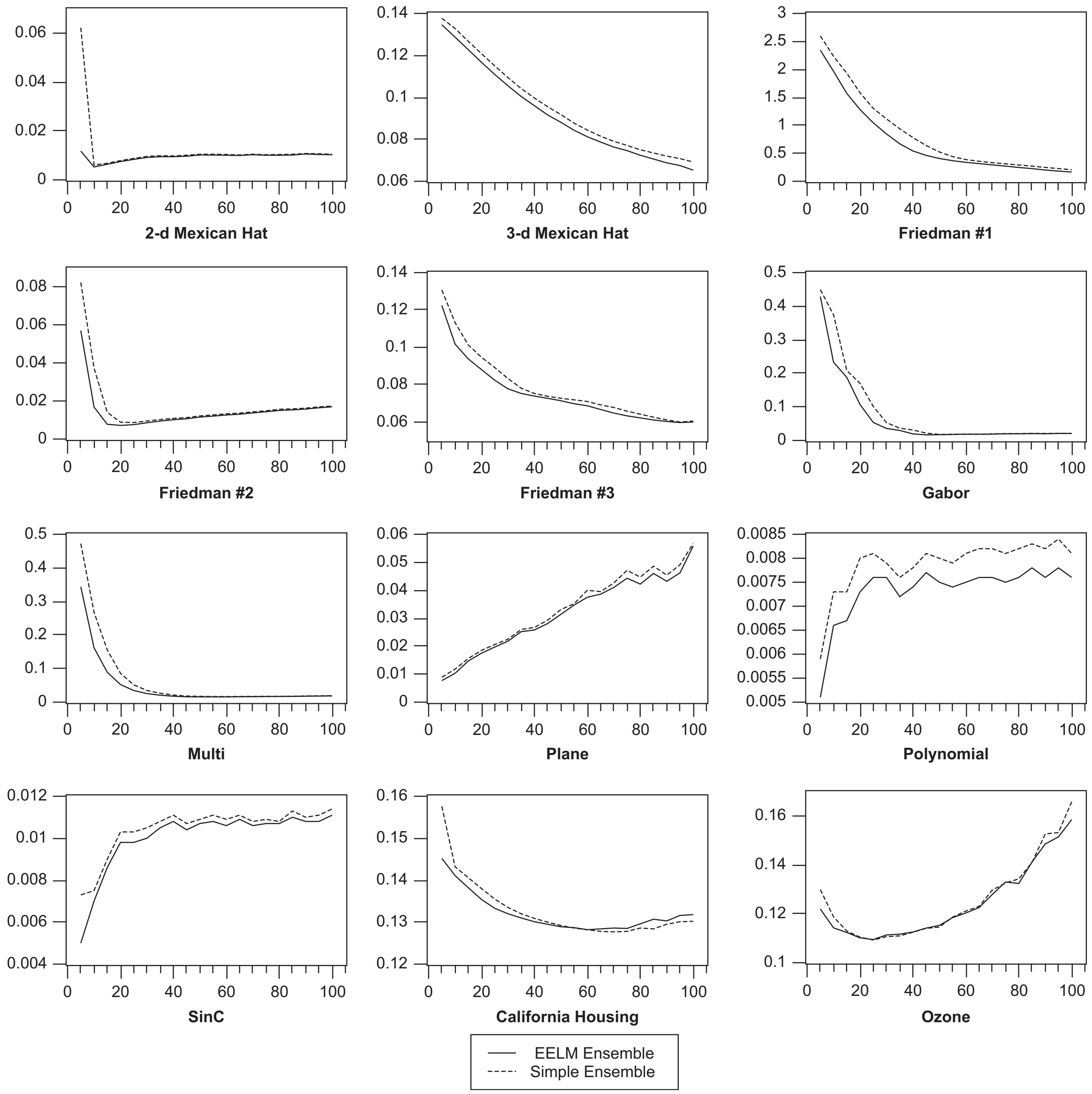

Fig. 8. Average testing RMSE of EE-ELM ensemble and simple ensemble for different sizes (number of hidden neurons) of base learners.

between two and six base models while the number of hidden neurons of base learners changes from 5 to 100 neurons. However, three cases (Friedman #1, Friedman #2 and Multi) only show larger upper bounds for EE-ELM models sizes; up to eight components. Therefore, the selection property of our algorithm is preserved regardless of the base learner structure. On the other hand, EE-ELM still performs better than simple ensemble as the number of hidden neurons increases, see Fig. 8. It is also worthy noted that overfitting sometimes occurs and consequently the testing RMSE of simple and EE-ELM ensembles degrades when the base learners size gets higher than a specific size (see 2-d Mexican Hat, Friedman #2, Gabor, Multi, Plane, Polynomial, SinC, California Housing and Ozone in Fig. 8). As a result, we need properly structured base learners with suitable number of hidden units to achieve good ensembling performance.

5. Conclusions

This paper reviews the basics of ensemble learning and explains three complexity terms (ambiguity, diversity and correlation matrix), proposed in the literature, for quantifying the disagreement among ensemble models. It has been observed from our experiments that the diversity term works well for building our ensemble model. A new evolutionary ELM-based ensemble (EE-ELM) is proposed that takes the advantage of ELM in generating models instantly and utilizes genetic algorithms (GAs) to select the best combination of ELM models from a given pool. Our proposed evolutionary scheme searches for the best ensembles with size control on the base models. Performance of our proposed EE-ELM has been sufficiently evaluated by 12 benchmark datasets for regression problems, where a comprehensive comparison with three traditional ensemble methods (simple average, bagging and Adaboost.RT) was given. Another two ensembles, named top-*K* and random-*K*, were coined to well examine the characteristics of our method. Some of the parameters of the studied methods were set empirically and others were tuned using a specific protocol explained earlier. Moreover, the robustness of our ensemble method was analyzed along with the number of hidden neurons of base learner models. The RMSE was used to estimate the generalization error of all methods.

In a nutshell, our proposed EE-ELM produces better performance than the traditional ensemble methods and the experimental random-*K* and top-*K* ensembles. Moreover, the overlapping percentage between the models of top-*K* ensemble and the models of EE-ELM ensemble does not exceed 50% in average, which implies the power of our ensembles encoding scheme in the selection process. EE-ELM also reduces the number of component models by roughly 95% comparing with other traditional ensembling approaches. Eventually, the performance characteristics of EE-ELM does not change with respect to the architecture of base learner models.

References

[1] L. Hansen, P. Salamon, Neural network ensembles, IEEE Trans. Pattern Anal. Mach. Intell. 12 (10) (1990) 993–1001.
[2] M.P. Perrone, L.N. Cooper, When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, Technical Report A121062, Brown University, Institute for Brain and Neural Systems, January 1993.
[3] A. Krogh, J. Vedelsby, Neural network ensembles, cross validation, and active learning, in: Advances in Neural Information Processing Systems, MIT Press, 1995, pp. 231–238.
[4] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.
[5] R.E. Schapire, The strength of weak learnability, Mach. Learn. 5 (1990) 197–227.
[6] Y. Freund, Boosting a weak learning algorithm by majority, in: Proceedings of the Third Annual Workshop on Computational Learning Theory, Morgan Kaufmann, 1990, pp. 202–216.
[7] Y. Liu, X. Yao, Ensemble learning via negative correlation, Neural Networks 12 (10) (1999) 1399–1404.
[8] X. Yao, Y. Liu, Making use of population information in evolutionary artificial neural networks, IEEE Trans. Syst. Man Cybern. 28 (3) (1998) 417–425.
[9] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241–259.
[10] D.W. Opitz, J W. Shavlik, Generating accurate and diverse members of a neural network ensemble, in: Advances in Neural Information Processing Systems, MIT Press, 1996, pp. 535–541.
[11] Z.-H. Zhou, J. Wu, W. Tang, Ensembling neural networks: many could be better than all, Artif. Intell. 137 (1–2) (2002) 239–263.
[12] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of IEEE International Joint Conference on Neural Networks, vol. 2, 2004, pp. 985–990.
[13] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489–501.
[14] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Networks 17 (4) (2006) 879–892.
[15] G.-B. Huang, D. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. Cybern. 2 (2011) 107–122.
[16] H. Chen, H. Chen, X. Nian, P. Liu, Ensembling extreme learning machines, in: Advances in Neural Networks, Lecture Notes in Computer Science, vol. 4491, Springer, Berlin, Heidelberg, 2007, pp. 1069–1076.
[17] N. Liu, H. Wang, Ensemble based extreme learning machine, IEEE Signal Process. Lett. 17 (8) (2010) 754–757.
[18] M. Heeswijk, Y. Miche, T. Lindh-Knuutila, P.A. Hilbers, T. Honkela, E. Oja, A. Lendasse, Adaptive ensemble models of extreme learning machines for time series prediction, in: Proceedings of the 19th International Conference on Artificial Neural Networks, Springer-Verlag, 2009, pp. 305–314.
[19] H.-X. Tian, Z.-Z. Mao, An ensemble ELM based on modified Adaboost.RT algorithm for predicting the temperature of molten steel in ladle furnace, IEEE Trans. Autom. Sci. Eng. 7 (1) (2010) 73–80.
[20] D.L. Shrestha, D.P. Solomatine, Experiments with Adaboost.RT, an improved boosting scheme for regression, Neural Comput. 18 (2006) 1678–1710.
[21] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, USA, 1992.
[22] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: optimally pruned extreme learning machine, IEEE Trans. Neural Networks 21 (1) (2010) 158–162.

Dianhui Wang received the Ph.D. degree from Northeastern University, Shenyang, China, in 1995. From 1995 to 2001, he worked as a Postdoctoral Fellow with Nanyang Technological University, Singapore, and a Researcher with The Hong Kong Polytechnic University, Hong Kong, China. He is currently a Reader and Associate Professor with the Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, Victoria, Australia. He is also associated with the State Key Laboratory of Synthetical Automation of Process Industries, Northeastern University, China. His current research interests include data mining and computational intelligence systems for bioinformatics and engineering applications.

Monther Alhamdoosh received his Bachelor Degree of Informatics Engineering in 2008 from University of Aleppo, Syrian Arab Republic, and a Master Degree in Bioinformatics in 2010 from University of Bologna, Italy. He is doing his Ph.D. in the Department of Computer Science and Computer Engineering, La Trobe University, Australia. His current research interests include computational intelligence systems for bioinformatics.