Contents lists available at ScienceDirect

Knowledge-Based Systems

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

DEFEG: Deep Ensemble with Weighted Feature Generation

Anh Vu Luong [a], Tien Thanh Nguyen [b], Kate Han [c], Trung Hieu Vu [d], John McCall [b], Alan Wee-Chung Liew [a,*]

[a] School of Information and Communication Technology, Griffith University, Australia
[b] School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, UK
[c] Salford Business School, The University of Salford, Manchester, UK
[d] School of Electronics and Telecommunications, Hanoi University of Science and Technology, Viet Nam

ARTICLE INFO

Article history:
Received 20 September 2020
Received in revised form 29 April 2023
Accepted 28 May 2023
Available online 2 June 2023

Keywords:
Ensemble method
Deep learning
Multiple classifiers
Ensemble of classifiers
Random forest
Feature generation

ABSTRACT

With the significant breakthrough of Deep Neural Networks in recent years, multi-layer architecture has influenced other sub-fields of machine learning including ensemble learning. In 2017, Zhou and Feng introduced a deep random forest called gcForest that involves several layers of Random Forest-based classifiers. Although gcForest has outperformed several benchmark algorithms on specific datasets in terms of classification accuracy and model complexity, its input features do not ensure better performance when going deeply through layer-by-layer architecture. We address this limitation by introducing a deep ensemble model with a novel feature generation module. Unlike gcForest where the original features are concatenated to the outputs of classifiers to generate the input features for the subsequent layer, we integrate weights on the classifiers' outputs as augmented features to grow the deep model. The usage of weights in the feature generation process can adjust the input data of each layer, leading the better results for the deep model. We encode the weights using variable-length encoding and develop a variable-length Particle Swarm Optimization method to search for the optimal values of the weights by maximizing the classification accuracy on the validation data. Experiments on a number of UCI datasets confirm the benefit of the proposed method compared to some well-known benchmark algorithms.

1. Introduction

Deep Neural Networks (DNNs) nowadays has become a powerful supervised machine learning technique. While being applied to various domains for visual and text information, the design concept of DNNs has influenced other sub-fields of machine learning. By going deeply through layer-by-layer processing, the machine learning models are likely to get better prediction ability. Recently, ensemble methods based on multi-layer architecture have become a new trend in the design of ensemble learning systems. Starting with the simple combination between two layers of ensemble methods like in RotBoost [1], cascade model of Boosting [2], and two-layer heterogeneous ensemble [3], some deeper ensemble systems have been introduced [4–8]. Experiments in [4] have confirmed the benefit of multi-layer architecture to the ensemble system as the deep ensemble systems offer not only higher classification performance than the one-layer ensemble systems and DNNs on some datasets but also fewer number of hyper-parameters than DNNs.

In 2017, Zhou and Feng [4] introduced a deep random forest (called gcForest, see Fig. 1), a learning architecture involving several layers of Random Forest-based classifiers. gcForest involves multiple layers of two Random Forest and two Completely Random Tree Forests (also Random Forest in which only one feature is considered when looking for the best split for node samples) in each layer. Each tree in a forest is made to output the class distribution vector which indicates the probabilities that a sample belongs to the class labels. The prediction vector of a forest is the average of the class distribution vectors of all the trees inside. The prediction vectors of the four forests are concatenated with the original training data at each layer as the new input training features for the subsequent layer. gcForest is better than Multi-Layer Perceptron (MLP) and well-known ensemble methods on diverse types of datasets in the authors' experiments [4].

gcForest uses the concatenation of the original training data and the predictions of classifiers in one layer as input to the next layer. It was assumed that the deeper we grow the model, the better the classification accuracy. However, our experiments show that the classification accuracy of gcForest could reduce when going to deeper layers. It means that either the input features do not improve data discriminative ability or going deeper in

* Corresponding author.
E-mail address: a.liew@griffith.edu.au (A.W.-C. Liew).

https://doi.org/10.1016/j.knosys.2023.110691

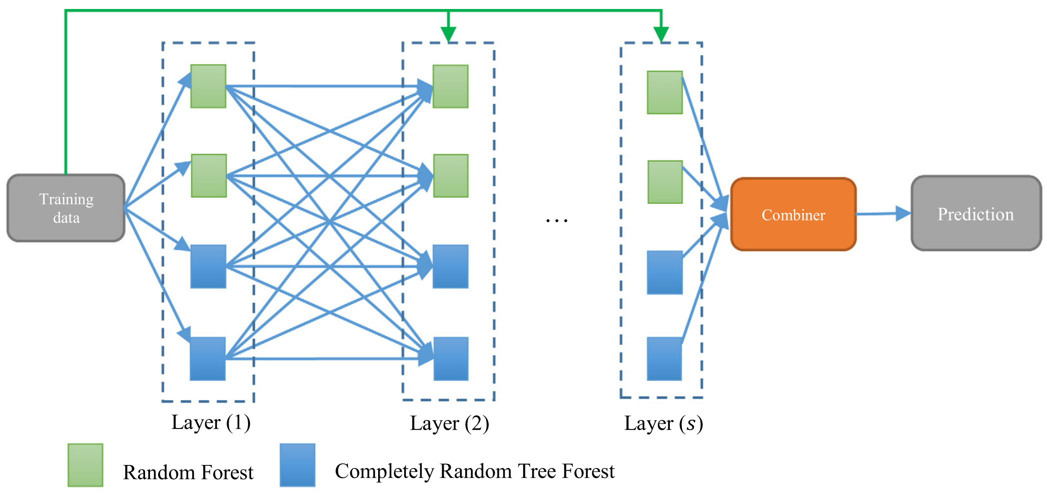


Fig. 1. The deep random forest. Fig. 1 shows the model of deep random forest called gcForest including multi-layers of two Random Forests and two Completely Random Tree Forests. The predictions of the forests are grouped and then are concatenated with the original training data (green arrows) to form the input features for the next layer. The prediction of the last layer is aggregated with a combiner to obtain the final classification.

gcForest does not provide any benefit compared to the one-layer ensemble models.

In this study, we introduce a novel deep ensemble learning model called Deep Ensemble with Weighted Feature Generation called DEFEG to address the limitations of gcForest. We propose a new approach to generate the input features for each layer using a weighted feature generation module that integrates weights on the classifiers' outputs. Like the gcForest, DEFEG involves several different classifiers in each layer. The difference between gcForest and DEFEG is the way to grow the deep learning model. Specifically, the predictions of the classifiers are integrated with different weights which shows the different contributions of the classifiers to the predictions. The number of weights depends on the number of layers in the deep model. We propose two feature generation approaches: weighted predictions themselves or the concatenation between weighted predictions and the original features as the input features for the subsequent layer. The optimal values of weights are obtained by maximizing the classification accuracy on the validation data. In this work, we develop a variable-length Particle Swarm Optimization (VLPSO) to solve the optimization problem using a variable-length encoding for the weights.

The main contributions of our work are as follows:

- We propose an approach to generate new augmented features to grow the deep ensemble models, based on the weighted predictions.
- We propose to encode the weights for all layers in a variable-length encoding
- We develop a VLPSO algorithm to search for the optimal weights.
- We experimentally show that DEFEG is better than some well-known benchmark algorithms on a number of datasets.

In Section 2, we briefly introduce deep ensemble learning, some variants of Particle Swarm Optimization algorithms, and some recent developments of Evolutionary Computation for the ensemble systems. In Section 3, we give a detailed description of the proposed method. Experimental studies on a number of datasets are provided in Section 4, followed by conclusions in Section 5.

2. Background and related work

2.1. Deep ensemble learning

Ensemble learning is a sub-field of machine learning in which a collection of classifiers is combined for the collaborative decision. Two main stages need to be considered when designing an ensemble system namely ensemble generation and ensemble integration. In the ensemble generation, multiple classifiers are generated by using homogeneous strategy (training a learning algorithm on multiple training sets generated from the original training data) [9,10] or heterogeneous strategy (training different learning algorithms on the original training data) [11,12]. The predictions of the selected classifiers are aggregated by a combining method in the ensemble integration stage to obtain the collaborated prediction. Several top-performance methods for the classification problem have been reported including Random Forest [13], XgBoost [14], and Rotation Forest [15].

Recently, there are a number of interests to the ensemble generation inspired by the success of DNNs. Instead of using only one layer like in traditional ensemble models, the ensemble systems were made to train deeply through multiple layers. The first deep ensemble system was proposed by Zhou and Feng [4] (called gcForest) with two Random Forests and two Completely Random Tree Forests working in each layer. Kim et al. [5] introduced a deep ensemble of Support Vector Machine (SVM) classifiers with an uncertainty checker in each layer. If the uncertainty in the predictions of the SVM classifiers in one layer is less than a threshold, these predictions are combined to obtain the final classification, otherwise, they are fed into the next layer. Utkin et al. [6] proposed a weight average approach for gcForest when combining the class distribution vectors of the trees in each forest. In this way, each tree is associated with a weighted vector for its class distribution vector. The weight vectors of the trees in a forest in one layer are found by minimizing the distance between the class label vector in a binary encoding scheme and the weighted prediction vector of this forest. To reduce the number of weight vectors when using too many trees in a forest, the authors proposed to group the class distribution vectors of the trees and only set a weight vector for each group. Qi et al. [7] introduced a deep ensemble model including an ensemble of SVM classifiers in each layer. In each layer, the authors used AdaBoost to find the parameters of the models including the kernel functions of the SVM classifiers, the number of classifiers, and the weights of the features. Chen et al. [8] proposed a feature generation approach to grow the deep ensemble model in which the predictions of all previous layers are concatenated to the original features as the input feature for the subsequent layer. Nguyen et al. [16] introduced MULES, a deep ensemble framework in which each of its layers is optimized by classifier and feature selection. The authors modeled the selection process at each layer as a bi-objective optimization problem, where classification accuracy and diversity of the ensemble in each layer are two objectives to be maximized. Dang et al. proposed a two-layer ensemble of deep learning models for the problem of medical image segmentation [17]. In this method, the pixel-wise predictions for each training image produced by the first layer's models are used as augmented data for training models in the second layer. Predictions made by the second layer are then combined to form the final output using a weighted approach that gives each model a different weight, and these weights can be optimized by solving linear regression problems. In [18], Dang et al. introduced another multi-layer ensemble of deep learning models for medical image segmentation, which contains a new last layer's combining strategy based on swarm intelligence. The authors addressed the high time complexity issue of the optimization process by employing a surrogate-based approach. Recently, Luong et al. applied the multi-layer ensemble architecture to the data stream setting to tackle the problem of insect stream classification [19]. The authors also made use of dynamic GA to develop an online ensemble selection method, aiming to find the optimal subset of classifiers at each layer. Last but not least, a multi-layer ensemble system comprised of gated recurrent unit (GRU), bidirectional long short-term memory (BiLSTM), and support vector machine (SVM) is used in [20] to predict RNA-binding proteins after employing a number of biological techniques to extract useful features, applying autoencoder for feature selection, and making use of SMOTE-ENN to balance the samples.

2.2. Evolutionary Computation for ensemble systems

Evolutionary Computation (EC) have been widely applied to improve the performance of ensemble systems in terms of *optimization for selection* and *optimization for adaptation*. In the first category, an EC method is used to search for the optimal subset of classifiers, original features, combining algorithms, or predictions (called meta-data) to obtain better performance than using the whole set of classifiers or features. In [21], the optimal set of classifiers was obtained based on searching with a hybrid Genetic Algorithm (GA). The hybrid GA involves a local search procedure to make the offspring likely be improved before going to the replacement stage. In [22], Wang et al. built an ensemble of classifiers by training decision tree on 100 new training sets generated by using the random subspace and bootstrap resampling techniques on the original training data. The non-dominated sorting genetic algorithm II (NSGA-II), a multi-objection optimization algorithm then was used to search for the optimal set of classifiers in terms of two objectives: maximizing classification accuracy and minimizing ensemble complexity i.e. the number of selected classifiers. In [12], Nguyen at el. proposed an ensemble selection method by comparing confidence in each classifier's predictions to an associated confidence threshold. The optimal values of confidence thresholds of classifiers are found by using Artificial Bee Colony algorithm. In [23], Mendialdua et al. used Estimation of Distribution Algorithm (EDA), an stochastic optimization method belonging to the class of Evolutionary Algorithm, to search for the optimal set of classifiers. In [24], both classification accuracy and ensemble diversity were addressed in the selection of classifiers and combining algorithms. The solution was obtained by using NSGA-II. In [25], the authors addressed the performance of ensemble on the imbalanced datasets in searching of ensemble of classifiers among the set of 20 classifiers. In [26], Nguyen et al. proposed to simultaneously select the optimal predictions and combining algorithms from the whole ensemble by using Ant Colony Optimization.

Meanwhile, in optimization methods for adaptation, the parameters of the classifiers or the combining algorithms are tuned by using an EC method to adapt to each specific classification task. In [27], GA was used to find the optimal weights which are the contributions of the learning algorithms on each training instance. These learning algorithms then train on the weighted training data to obtain the ensemble of classifiers. In [28], Genetic Programming and PSO were used in searching for the prediction function as the combining algorithm and in searching for the threshold for the prediction functions in different feature spaces, respectively. PSO was used to find the optimal representation in the form of vector [29] or vector of intervals [30] for each class labels on the meta-data as the basis for the combining algorithm on the classifiers' prediction.

There are also methods to handle both selection and adaptation. For example, in [31], the selection and adaptation approach for an ensemble of SVM classifiers were conducted based on the searching on chromosomes of encoding for classifiers, features, and parameters and weights of classifiers. In [32], the feature selection methods were encoded to indicate which features were used for each learning algorithm to train the classifiers. The optimal solution then was obtained by using GA. The detail of EC techniques for ensemble systems can be found in [29].

2.3. Particle swarm optimization and comprehensive learning

As mentioned, Evolutionary Computation is a popular approach to search for the optimized configuration for ensemble systems. Particle Swarm Optimization (PSO) [33], a kind of swarm algorithm inspired by the emergent motion of a flock of birds searching for food, can be used for this task. This algorithm performs a local search within each particle and global search among the whole swarm at the same time. For a U-dimension optimization problem, a particle in typical PSO is defined by $\boldsymbol{x}_i = \left(x_i^1, x_i^2, \ldots, x_i^U\right)$; $\boldsymbol{v}_i = \left(v_i^1, v_i^2, \ldots, v_i^U\right)$ in which $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ are the position and velocity of the ith particle. Each particle's velocity is updated based on its current velocity, local best position, and global best position. This ensures each particle is learned from the whole swarm search result while it performs its own search.

$$v_i^u \leftarrow a \times v_i^u + c_1 \times r_1 \times \left(pbest_i^u - x_i^u\right) + c_2 \times r_2 \times \left(gbest^u - x_i^u\right) \quad (1)$$

Then each particle's position is updated based on its velocity and its current position

$$x_i^u \leftarrow x_i^u + v_i^u \quad (2)$$

Three learning parameters (a, c_1, c_2) are defined to control the speed convergence of the search: a is the inertia weight which is used to control the velocity speeding rate, c_1 is an acceleration constant used to control the learning rate of the particle's local best while c_2 is an acceleration constant used to control the learning rate of the swarm global best. $pbest_i^u$ is the uth dimension of ith particle's best position and $gbest^u$ is the uth dimension of the swarm's best position. r_1 and r_2 are two random number drawn from a uniform distribution over [0, 1]. Until now, various variants of PSO aiming to enhance the performance of original one can be found in the literature [34].

It is believed that the original PSO can easily get trapped by local optimum as all particles learn from the global best position even if the current global best is far from the global optimum [35]. In 2006, Liang et al. proposed a variant of PSO called Comprehensive Learning PSO (CLPSO) [35] in which each particle is designed to learn from all particles' local best position. Specifically, each particle with U-dimension will also have a U-dimension exemplar vector $\boldsymbol{e}_i = \left(e_i^1, e_i^2, \ldots, e_i^U\right)$ for comprehensive learning. The exemplar vector is introduced for a particle to learn from the local best (*pbest*) of itself as well as all the other particles. For example, a particle with the position (0.13, 0.43, 0.22, 0.74, 0,11), the velocity (0.48, 0.25, 0.52, 0.13, −0.15), and the exemplar (6, 8, 4, 8, 4) would learns/updates the 3rd dimension position value based on the 3rd dimension position value of the 4th particle's *pbest*. A particle is randomly assigned with an exemplar vector after initialization. The exemplar will be renewed if a particle's *pbest* stop improving after a number of iterations. For each dimension of a certain particle, two different particles will be selected randomly and the better one will be assigned as the exemplar for the updated particle on the corresponding dimension [35,36]. Therefore, only one acceleration of constant c is needed. The updated equation is given by:

$$v_i^u \leftarrow a \times v_i^u + c \times r_1 \times \left(pbest^u_{\boldsymbol{e}_i^u} - x_i^u\right) \tag{3}$$

in which c is an acceleration constant used to control the learning rate of the exemplars' local best, $pbest^u_{\boldsymbol{e}_i^u}$ is the uth dimension of particle's best position referring to the uth dimension of exemplar $\boldsymbol{e}_i$. Later in 2014, Yu et ca. proposed an Enhanced CLPSO algorithm (ECLPSO) [37]. Targeting the low solution accuracy disadvantage of CLPSO, they proposed two approaches. First, they used a perturbation term to update each particle's velocity to improve the exploitation of the swarm. Normative knowledge about dimensional bounds of personal best positions is used to appropriately activate the perturbation-based exploitation. Second, they changed the particles' learning probabilities from the initial pre-determined based on particle ranking to adaptative to speed up convergence. Lynn et al. [38] proposed a heterogeneous comprehensive learning particle swarm optimization algorithm (HCLPSO). The approach aims at enhancing both exploration and exploitation during the searching process. Specifically, the swarm is divided into two subswarm with different focus: exploration and exploitation. The exploration group exemplars are generated based on its own subswarm particles local best experience while the exploitation group exemplars are calculated based on the entire swarm particles local best experience.

In this paper, we introduce a novel deep ensemble model, DEFEG, which addresses the limitation of gcForest's performance. Our model addresses the research gap of gcForest by enhancing the input feature's discriminative ability and making performance gain possible from going deeper in layer in the ensemble. We propose a new approach to generating input features for each layer using a weighted feature generation module that integrates weights on the classifiers' outputs. We encode the weights using variable-length encoding and develop a variable-length Particle Swarm Optimization (VLPSO) method to search for the optimal values of the weights by maximizing the classification accuracy on the validation data.

3. Proposed system

3.1. General description

We denote $\mathcal{D} = \left\{\left(\mathbf{x}_n, \hat{y}_n\right)\right\}$ as the training data with N instances, where $\mathbf{x}_n = \{x_{nd} | d = 1, \ldots, D\} \in \mathcal{R}^D$ is the D-dimension feature vector of the training instance, $\hat{y}_n \in \mathcal{Y} = \{y_m, m = 1, \ldots, M\}$ is its corresponding label, $\mathcal{K} = \{\mathcal{K}_k, k = 1, \ldots, K\}$ as the set of K learning algorithms. In each layer, we train the set of classifier $\mathcal{H}^{(i)} = \left\{h_k^{(i)}, k = 1 \ldots K\right\}$. The classifiers in the first layer of DEFEG receive the original training features and outputs the predictions for the training instances. These predictions are integrated with the weights of classifiers then will be used as augmented features for the second layer. The process continues until reaching the last layer where the predictions of classifiers in this layer (e.g. sth layer) are combined by a combining algorithm C: $\tilde{h} = C\left\{\mathcal{H}^{(s)}\right\}$ for the final prediction. The proposed method DEFEG is illustrated in Fig. 2:

There are two questions concerning the proposed model of DEFEG:

- How to generate the predictions for the training observations and then integrate these predictions with the weights of classifiers to create the input data for the subsequence layer?
- How to combine the predictions of classifiers of the last layer to obtain the final prediction?

For the first question, we use the Cross Validation procedure with K learning algorithms to obtain the predictions for the training instances. In detail, in the first layer, we use T_1-fold Cross Validation procedure on the original training data $\mathcal{D} = \mathcal{L}_0$ to divide the training data into T_1 disjoint parts that are nearly equal in cardinality. We use one part as testing while the others as training for K learning algorithms that make each training instance be used for training $(T_1 - 1)$ times and for the testing 1 time. After running through all T_1 folds, we obtain the predictions of the classifiers in the first layer for the training instances. The same T_i-fold Cross Validation procedure is conducted at the ith layer for prediction generation. The kth classifiers trained on the complementary of T_i in the Cross Validation procedure at the ith layer gives support for the hypothesis that $\mathbf{x}_n \in T_i$ belongs to class label y_m which is denoted by $p_{k,m}^{(i)}(\mathbf{x}_n)$. For M class labels, all predictions will give in a vector $\left\{p_{k,m}^{(i)}(\mathbf{x}_n)\right\}$, $m = 1, \ldots, M$ which shows the probabilities $\mathbf{x}_n$ belongs to each class label [12,29].

Let $W^{(i)} = \left\{w_{km}^{(i)}\right\}$ $i = 1, \ldots, s$; $1 \leq s \leq S_{max}$ denotes the integrated weights on the prediction vector in which $w_{km}^{(i)} \in [0, 1]$ is the weight (i.e. the contribution) of classifier $h_k^{(i)}$ to the integration associated with the class label y_m. In this study, we propose using $W^{(i)}$ to generate features for the $(i+1)$ th layer. The weights of classifiers in a layer is an approach to change the structure of the prediction data with the aim of making deep model become better after going through the subsequent layer. Table 1 presents the feature generation model based on weighted integration in which we integrate each prediction with its associated weights and also shows the combining result associated with the class labels. The output of the feature generation model is a weighted integration vector $\mathbf{P}^{(i)}(\mathbf{x}_n)$ for each training instance $\mathbf{x}_n$.

The prediction vector for $\mathbf{x}_n$ based on the weights is given by:

$$\begin{aligned}&\mathbf{P}^{(i)}(\mathbf{x}_n)\\&= \left[w_{11}^{(i)} p_{1,1}^{(i)}(\mathbf{x}_n), \ldots, w_{1M}^{(i)} p_{1,M}^{(i)}(\mathbf{x}_n), \ldots, w_{K1}^{(i)} p_{K,1}^{(i)}(\mathbf{x}_n), \ldots, w_{KM}^{(i)} p_{K,M}^{(i)}(\mathbf{x}_n)\right]\end{aligned} \tag{4}$$

We propose two approaches to generate input training data for one layer. The first approach is to concatenate $\mathbf{P}^{(i)}(\mathbf{x}_n)$ to the original training features. The input training data associated with $\mathbf{x}_n$ for the $(i+1)$ th layer is given by:

$$\mathcal{L}_i(\mathbf{x}_n) = \left[x_{n1}, x_{n2}, \ldots, x_{nD}, \mathbf{P}^{(i)}(\mathbf{x}_n), \hat{y}_n\right] \tag{5}$$

The second approach is to use $\mathbf{P}^{(i)}(\mathbf{x}_n)$ only as the training data associated with $\mathbf{x}_n$ for the $(i+1)$ th layer

$$\mathcal{L}_i(\mathbf{x}_n) = \left[\mathbf{P}^{(i)}(\mathbf{x}_n), \hat{y}_n\right] \tag{6}$$

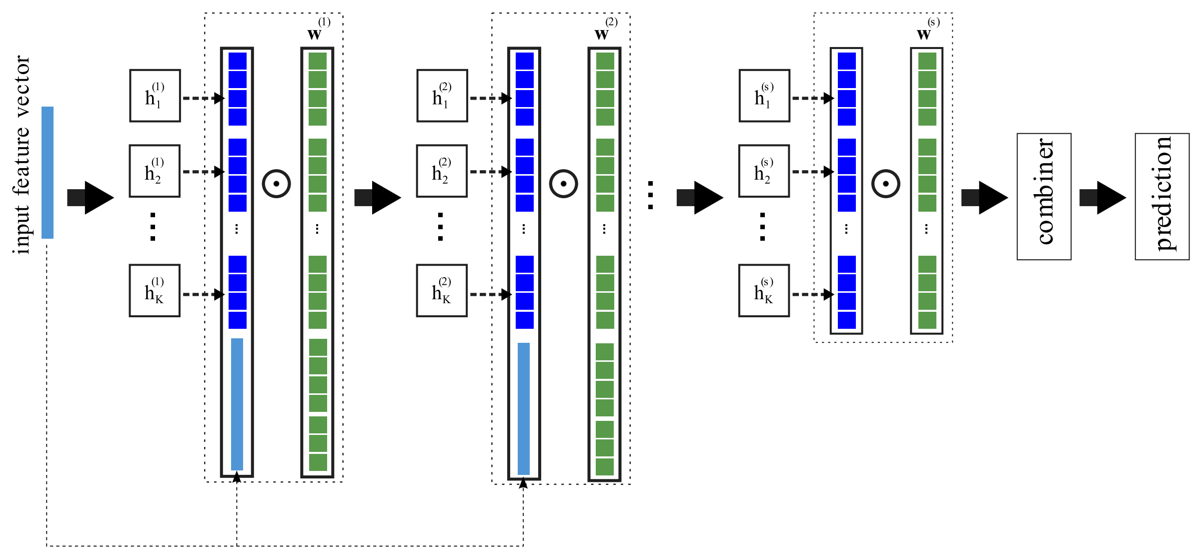


Fig. 2. The proposed method DEFEG.

Table 1
The weighted integration on the predictions of *i*th layer.

	Class 1	Class 2	...	Class m	...	Class M
Classifier 1	$p_{1,1}^{(i)}(\mathbf{x}_n)\ w_{11}^{(i)}$	$p_{1,2}^{(i)}(\mathbf{x}_n)\ w_{12}^{(i)}$	...	$p_{1,m}^{(i)}(\mathbf{x}_n)\ w_{1m}^{(i)}$	...	$p_{1,M}^{(i)}(\mathbf{x}_n)\ w_{1M}^{(i)}$
Classifier 2	$p_{2,1}^{(i)}(\mathbf{x}_n)\ w_{21}^{(i)}$	$p_{2,2}^{(i)}(\mathbf{x}_n)\ w_{22}^{(i)}$	...	$p_{2.m}^{(i)}(\mathbf{x}_n)\ w_{2m}^{(i)}$	...	$p_{2,M}^{(i)}(\mathbf{x}_n)\ w_{2M}^{(i)}$
...	...	...		...		...
Classifier $\boldsymbol{k}$	$p_{k,1}^{(i)}(\mathbf{x}_n)\ w_{k1}^{(i)}$	$p_{k,2}^{(i)}(\mathbf{x}_n)\ w_{k2}^{(i)}$	...	$p_{k,m}^{(i)}(\mathbf{x}_n)\ w_{km}^{(i)}$	...	$p_{k,M}^{(i)}(\mathbf{x}_n)\ w_{kM}^{(i)}$
...	...	...		...		...
Classifier $\boldsymbol{K}$	$p_{K,1}^{(i)}(\mathbf{x}_n)\ w_{K1}^{(i)}$	$p_{K,2}^{(i)}(\mathbf{x}_n)\ w_{K2}^{(i)}$	...	$p_{K,m}^{(i)}(\mathbf{x}_n)\ w_{Km}^{(i)}$	...	$p_{K,M}^{(i)}(\mathbf{x}_n)\ w_{KM}^{(i)}$
Combining	$\frac{1}{K}\sum_{k=1}^{K} w_{k1}^{(i)}P_{k,1}^{(i)}(\mathbf{x}_n)$	$\frac{1}{K}\sum_{k=1}^{K} w_{k2}^{(i)}P_{k,2}^{(i)}(\mathbf{x}_n)$		$\frac{1}{K}\sum_{k=1}^{K} w_{km}^{(i)}P_{k,m}^{(i)}(\mathbf{x}_n)$		$\frac{1}{K}\sum_{k=1}^{K} w_{kM}^{(i)}P_{k,M}^{(i)}(\mathbf{x}_n)$

In fact, $\mathbf{P}^{(i)} = \left\{\mathbf{P}^{(i)}(\mathbf{x}_n)\right\}$ can be treated as the augmented features which would change the structure of the prediction data with the values of the weights. The input data of all training instances are concatenated to form the new training data $\mathcal{L}_i$ for the $(i+1)$ th layer in the shape of an $N \times (M \times K + D)$ matrix in case of using the first approach in (5) or an $N \times (M \times K)$ matrix in case of using the second approach in (6)

$$\mathcal{L}_i = [\mathcal{L}_i(\mathbf{x}_1), \ldots, \mathcal{L}_i(\mathbf{x}_n), \ldots, \mathcal{L}_i(\mathbf{x}_N)]^{\mathrm{T}} \tag{7}$$

The combining algorithm C: $\tilde{h} = C\left\{\mathcal{H}^{(s)}\right\}$ works on the predictions of the classifiers in the last layer to obtain the final predicted label. Assume for a model with s layer, a class label is assigned to sample $\mathbf{x}$ by getting one associated with the maximum of the weighted average on the predictions of the classifiers in the last layer:

$$\tilde{\boldsymbol{h}}(\mathbf{x}) : \mathbf{x} \in \mathrm{y} \text{ if } \mathrm{y} = \arg \max_{y_m, m=1..M} \frac{1}{K}\sum_{k=1}^{K} w_{km}^{(s)}P_{k,m}^{(s)}(\mathbf{x}) \tag{8}$$

Algorithm 1 and 2 present the training and classification process of DEFEG. The training process gets the training data, K learning algorithms, and weights for feature generation and outputs the set of classifiers in each layer. In *i*th layer, we first train the set of classifiers $\mathcal{H}^{(i)} = \left\{h_k^{(i)}\right\}$ by using K learning algorithms on the whole input training data $\mathcal{L}^{(i-1)}$ of this layer in Step 4–6. The following steps aim to generate input training data for the $(i+1)$ th layer. In step 7, the input training data $\mathcal{L}^{(i-1)}$ is divided into T_i disjoin nearly equal parts. For each part $\mathcal{L}_j^{(i-1)}$, we train classifiers on its complementary $\mathcal{L}^{(i-1)} - \mathcal{L}_j^{(i-1)}$ and then use these classifiers to predict for its observations. These predictions of all parts are joined together in Step 11 so that we obtain the predictions for all training observations at *i*th layer. In Step 14, predictions of all training observations are integrated with the weights to obtain weighted predictions $\mathbf{P}^{(i)}$ in a $N \times (M \times K)$ matrix. The input training data for the $(i+1)$ th layer is generated by concatenating $\mathbf{P}^{(i)}$ to the original training data $\mathcal{L}^{(0)} = \mathcal{D}$ (5) or by using $\mathbf{P}^{(i)}$ only (6).

The classification process meanwhile gets an unseen sample $\mathbf{x}$, the set of classifiers in each layer, and the weights as the input data. For *i*th layer, we use the set of classifiers in this layer to predict on $\mathcal{L}^{(i-1)}(\mathbf{x})$ (Step 5). All predictions then are integrated with the weights $W^{(i)}$ to create the weighted prediction $\mathbf{P}^{(i)}(\mathbf{x})$ (4). $\mathbf{P}^{(i)}(\mathbf{x})$ will be used with or without $\mathbf{x}$ to generate the test data for the $(i+1)$ th layer (Step 8). The classification process runs through layers in the model until reaching the last layer i.e. sth layer. Finally, the predictions of classifiers in the last layer are combined by using weights $W^{(s)}$ to obtain the predicted label (Step 11).

Algorithm 1: Training process of DEFEG

```
Training Process
Input:
    - Training data $\mathcal{D} = \{(x_n, \hat{y}_n), n = 1 \ldots N\}$
    - $K$ learning algorithms $\mathcal{K} = \{\mathcal{K}_k\}$
    - Weight $W^{(i)} = \{w_{km}^{(i)}\}\ i = 1, \ldots, s$
Output:
    - List of classifiers $\mathcal{H}^{(i)} = \{h_k^{(i)}\}\ i = 1, \ldots, s$
1. Initialize  $\mathcal{L}^{(0)} = \mathcal{D}$;  $i = 1$
2. While $i \le s$:
3.      $predictions = \emptyset$;
    %Train classifiers for $i^{th}$ layer
4.      For $k = 1 \ldots K$
5.          $h_k^{(i)} = Training(\mathcal{K}_k, \mathcal{L}^{(i-1)})$;
6.      End For
    %Generate input data for $(i+1)^{th}$ layer
7.      $\mathcal{L}^{(i-1)} = \mathcal{L}_1^{(i-1)} \cup \mathcal{L}_2^{(i-1)} \cup \ldots \cup \mathcal{L}_{T_i}^{(i-1)}$; $\mathcal{L}_j^{(i-1)} \cap \mathcal{L}_l^{(i-1)} = \emptyset$; $j \ne l$
        $\left|\mathcal{L}_j^{(i-1)}\right| \approx \left|\mathcal{L}^{(i-1)}\right| / T_i$;
8.      For each $\mathcal{L}_j^{(i-1)}$
9.          For $k = 1 \ldots K$
10.             $c_k^{(i)} = Training\left(\mathcal{K}_k, \mathcal{L}^{(i-1)} - \mathcal{L}_j^{(i-1)}\right)$;
11.             $predictions = predictions \cup predict\left(c_k^{(i)}, \mathcal{L}_j^{(i-1)}\right)$
12.         End For
13.     End For
14.     $\mathbf{P}^{(i)} = Weighted\_features(predictions, W^{(i)})$;
15.     $\mathcal{L}^{(i)} = \mathbf{P}^{(i)} \cup \mathcal{L}^{(0)}$ (5) or $\mathcal{L}^{(i)} = \mathbf{P}^{(i)}$ (6)
16.     $i{+}{+}$;
17.   End While
18.   Return $\mathcal{H}^{(i)} = \{h_k^{(i)}\}\ i = 1, \ldots, s$
```

Algorithm 2: Classification process of DEFEG

```
Input:
    - Unseen data x
    - List of classifiers $\mathcal{H}^{(i)} = \{h_k^{(i)}\}\ i = 1, \ldots, s$
    - Weight $W^{(i)} = \{w_{km}^{(i)}\}\ i = 1, \ldots, s$
Output:
    - Predicted label
1. $\mathcal{L}^{(0)}(\mathbf{x}) = \mathbf{x}; i = 1$;
2. While $i \le s$:
3.      $prediction = \emptyset$
4.      For $k = 1 \ldots K$
5.          $prediction = prediction \cup Predict\left(h_k^{(i)}, \mathcal{L}^{(i-1)}(\mathbf{x})\right)$
6.      End
7.      $\mathbf{P}^{(i)}(\mathbf{x}) = Weighted\_features(prediction, W^{(i)})$;
8.      $\mathcal{L}^{(i)}(\mathbf{x}) = \mathbf{P}^{(i)}(\mathbf{x}) \cup \mathcal{L}^{(0)}(\mathbf{x})$ or $\mathcal{L}^{(i)}(\mathbf{x}) = \mathbf{P}^{(i)}(\mathbf{x})$
9.      $i{+}{+}$;
10. End
11. $y = Weighted\_predict(P^{(s)}(\mathbf{x}), W^{(s)})$ by (8)
12. Return $y$
```

3.2. Proposed encoding

One question that arises from this model is to search for the weights $\mathcal{W} = \{W^{(i)}\}$. In this work, $\mathcal{W}$ is found by maximizing the classification accuracy of the ensemble on a validation set $\mathcal{V}$. We define the empirical fitness function on $\mathcal{V}$ by comparing the prediction hypothesis for $\mathbf{x} \in \mathcal{V}$ i.e. $\widetilde{\boldsymbol{h}}(\mathbf{x})$ to its ground truth $\hat{y}$. The optimization problem based on the fitness function computed on $\mathcal{V}$ is given by:

$$\max_{\mathcal{W}=\{W^{(i)}\}} \left\{ \frac{1}{|\mathcal{V}|} \sum_{\mathbf{x}\in|\mathcal{V}|} [\![\widetilde{\boldsymbol{h}}(\mathbf{x}) = \hat{y}]\!] \right\} \tag{9}$$

s.t $w_{km}^{(i)} \in [0, 1]$, $W^{(i)} = \{w_{km}^{(i)}\}$
$i = 1 \ldots s; k = 1 \ldots K; m = 1 \ldots M$

in which $\widetilde{\boldsymbol{h}}$ is the combining model in the last layer, $|\cdot|$ denotes the cardinality of a set, and $[\![.]\!]$ is equal 1 if the condition is true, otherwise equal 0.

We propose using Evolutionary Computation-based approach in solving the optimization problem in (9) since it is an effective approach to deal with non-differentiable, discontinuous, or multi-model objective functions involved in the optimization problems. In this way, the proposed encoding for the weights in DEFEG is presented in Fig. 3 in which we use the variable-length numerical encoding for the weights of s layers. Since the weights of each layer are given in the shape of $M \times K$ vector (see Table 1), the chromosome associated with the model of s layers has $s \times K \times M$ numerical elements: $w_{km}^{(i)} \in [0, 1]$. The proposed encoding is divided into two parts: the first part including the weights for layers from 1 to $(s-1)$: $W^{(i)} = \{w_{km}^{(i)}\}$ $i = 1, \ldots, s-1$ and the second part is the combining weight for the last layer $W^{(s)} = \{w_{km}^{(s)}\}$ $k = 1, \ldots, K; m = 1, \ldots, M$. The weights $W^{(i)}$ $i = 1, \ldots, s-1$ is used to generate augmented features for the subsequent layer while the weight $W^{(s)}$ is used for the combining method in (8) to obtain a predicted class label. It is noted that the length of the chromosome is not fixed and depends on the number of layers that we use to construct the deep model. Here the number of layers is chosen by $1 \le s \le maxS$ in which $maxS$ is the maximum number of layers. Based on this encoding, we can obtain not only the optimal weights for feature generation and combining but also the optimal number of layers for the deep model.

3.3. Variable-length PSO

VLPSO is a research topic that has emerged recently as an efficient tool to solve optimization problems with numerical variable-length encodings [36]. Some VLPSO methods have been introduced in the literature. Wang et al. [39] proposed to use VLPSO to search for the optimal structure of CNN and the parameters in each layer. The proposed searching works on fixed-length particles and then uses IP encoding/decoding to realize the variable-length search. Another example of the VLPSO can be found in [36] which aimed to solve the feature selection problem. In this study, we develop a VLPSO algorithm to solve the optimization problem of (9) based on the studies of comprehensive learning PSO [35] and the VLPSO for feature selection [36].

VLPSO representation: Each particle is represented with a triple: a position vector, a velocity vector, and an exemplar vector. The position vector is decoded into the weights $\mathcal{W}$ for fitness evaluation. The velocity $\boldsymbol{v}_i = \left(v_i^1, v_i^2, \ldots, v_i^{s\times M\times K}\right)$ is used to update the position vector for the searching purpose. By using

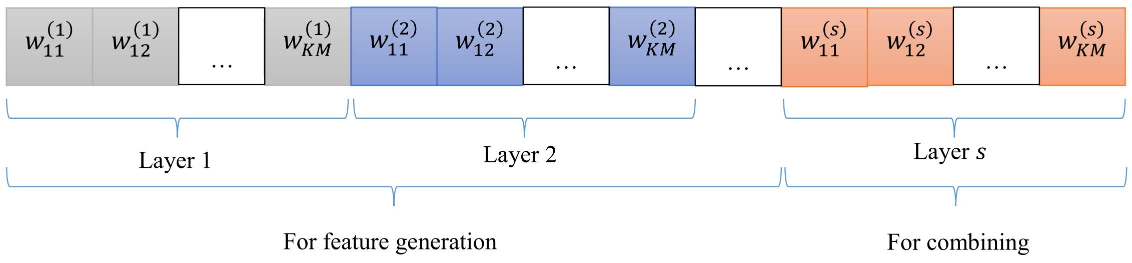


Fig. 3. Proposed encoding for the DEFEG.

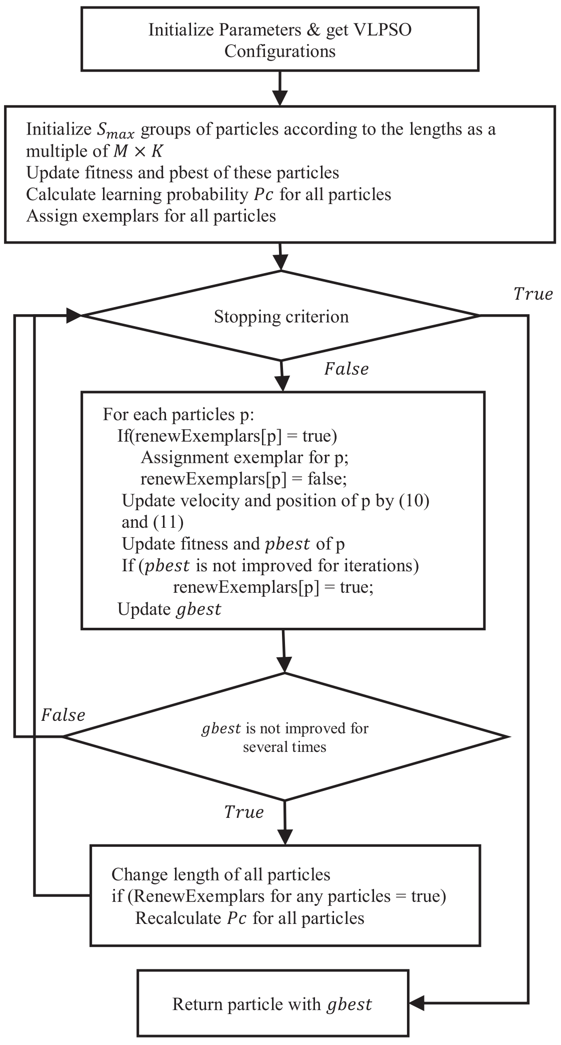

Initialize Parameters & get VLPSO Configurations
Initialize S_{max} groups of particles according to the lengths as a multiple of $M \times K$
Update fitness and pbest of these particles
Calculate learning probability Pc for all particles
Assign exemplars for all particles
Stopping criterion
True
False
For each particles p:
If(renewExemplars[p] = true)
Assignment exemplar for p;
renewExemplars[p] = false;
Update velocity and position of p by (10) and (11)
Update fitness and $pbest$ of p
If ($pbest$ is not improved for iterations)
renewExemplars[p] = true;
Update $gbest$
$gbest$ is not improved for several times
False
True
Change length of all particles
if (RenewExemplars for any particles = true)
Recalculate Pc for all particles
Return particle with $gbest$

Fig. 4. The flowchart of VLPSO.

two update Eqs. (2) and (3), we have:

$$v_i^u \leftarrow v_i^u \times a + c \times r_1 \times \left(pbest_{e_i^u}^u - w_i^u\right) \tag{10}$$

$$w_i^u \leftarrow w_i^u + v_i^u \tag{11}$$

Adaptive learning probability: In order to maintain the converging speed, comprehensive learning is controlled by a learning probability Pc. This allows particle with worse performance to have more chance to learn from other particles' $pbest$ while particles with better performance learn more from their own $pbest$. The learning probability is calculated as below [36]:

$$Pc^{ith} = 0.05 + 0.45 * \frac{e^{\frac{10*(rank(i)-1)}{(number\_of\_particles-1)}}}{e^{10} - 1} \tag{12}$$

in which $rank(i)$ is the performance ranking of the ith corresponding particle. For this particle, it has a probability Pc^{ith} to learn from other particles while probability (1-Pc^{ith}) to learn from its own $pbest$. Specifically, when assigning or renewing exemplars, with a probability (1-Pc^{ith}) the exemplar will be the corresponding particle's index while with a probability Pc^{ith} the exemplar will be randomly picked from other particles.

Particle groups division: The VLPSO aims to search for the best structure of the deep ensemble model and the weights simultaneously. In order to identify the best structure (number of layers), the number of particles ($PopSize$) is equally divided into a number of groups which is set to S_{max}. To ensure all the particle candidates be feasible, we initialized all the particles length as a multiple of $M \times K$. In detail, we first generate a set of particle length by randomly picking up a value in the group size set of $[M \times K, 2 \times M \times K, \ldots, S_{max} \times M \times K]$. The proposed VLPSO will enable the search of different layers at the same time. For example, if the maximum layer number S_{max} is set to 5 and $M \times K = 12$, we will divide all the particles into 5 groups and initialized them with 5 different lengths [12,17,29].

Length changing: The length of all particle division will be changed when the global best ($gbest$) stops changing after a certain number of iteration. When the length changing happens, the current maximum length will be reduced by $M \times K$, and all the particle divisions' size will be updated proportionally to the current maximum length except the group having the same length with the $gbest$ size. For example, if the current $gbest$ particle's size is Lg and the maximum particle length is $LMax$, the maximum size will be updated to $LMax - M \times K$ and the particles group size will be updated to the proportion of $LMax$ except for the particle groups that have size Lg. After the update, the particle length updated to lower than the minimum size $M \times K$ will be set to $M \times K$ to ensure the solution is feasible.

There are two differences between the proposed VLPSO and the VLPSO for feature selection in [36]. The first difference is in the initialization phase in which to ensure the particles encoding is feasible to construct the deep ensemble model, all the particle lengths are initialized as a multiple of $M \times K$. Meanwhile, the number of groups was set to the maximum number of layers S_{max}. Thus S_{max} groups with the lengths $[M \times K, 2 \times M \times K, \ldots, S_{max} \times M \times K]$ are initialized (see Fig. 4). The second difference comes from the length changing procedure. The VLPSO in [36] was developed to handle the feature selection problem in which the number of retained features should be chosen as small as possible, resulting in the quick reduction of the particles' length in the

Table 2
Summary of notations.

Notations	Mathematical meaning
N	The number of training instances
$\mathcal{D} = \left\{(\mathbf{x}_n, \hat{y}_n)\right\}, n = 1, \ldots N$	The training data
D	The dimension of feature vector
$\mathbf{x}_n = \{x_{nd} \mid d = 1, \ldots, D\} \in \mathcal{R}^D$	The feature vector of the nth training instance
M	The number of classes
$\hat{y}_n \in \mathcal{Y} = \{y_m, m = 1, \ldots, M\}$	The label of the nth training instance
K	The number of learning algorithms
$\mathcal{K} = \{\mathcal{K}_k, k = 1, \ldots, K\}$	The set of K learning algorithms
$\mathcal{H}^{(i)} = \left\{h_k^{(i)}, k = 1 \ldots K\right\}$	The set of K classifiers trained from $\mathcal{K}$
s	The number of layers
$maxS$	The maximum number of layers
C	A combining algorithm
$p_{k,m}^{(i)}(\mathbf{x}_n)$	The support for the hypothesis that $\mathbf{x}_n$ belongs to class label y_m given by the kth classifier at the ith layer
$W^{(i)} = \left\{w_{km}^{(i)}\right\}$	The integrated weights on the prediction vector, where $w_{km}^{(i)}$ is the weight of classifier $h_k^{(i)}$ to the integration associated with the class label y_m.
$\mathcal{W} = \left\{W^{(i)}\right\}$	The weights
$\mathbf{P}^{(i)}(\mathbf{x}_n)$	The prediction vector of the ith layer for $\mathbf{x}_n$ based on weights
$\mathcal{L}_i$	The training data for the $(i+1)$ th layer
$\boldsymbol{x}_i = \left(x_i^1, x_i^2, \ldots, x_i^U\right)$	The position of the ith particle in PSO
$\boldsymbol{v}_i = \left(v_i^1, v_i^2, \ldots, v_i^U\right)$	The velocity of the ith particle in PSO
$\mathbf{e}_i = \left(e_i^1, e_i^2, \ldots, e_i^U\right)$	The U-dimension exemplar vector of a particle
a	The inertia weight which is used to control the velocity speeding rate in PSO
c	The acceleration constant used to control the learning rate of the exemplars' local best
r_1 and r_2	Two random number drawn from a uniform distribution over [0, 1]
Pc	The learning probability to control comprehensive learning
$PopSize$	The number of particles
S_{max}	The number of initial groups of particles
$pbest_i$	The ith particle's best position
$gbest$	The swarm's best position
Lg	The current *gbest* particle's size
$LMax$	The maximum particle length

algorithm. For DEFEG, we only reduce from the maximum length to one-layer size less every time the length changing procedure is triggered.

The flowchart of proposed VLPSO is presented in Fig. 4. First, we initialize S_{max} groups of particles according to lengths of the multiple of $M \times K$ as mentioned above. For each particle, based on its a position vector (i.e. weights), we use Algorithm 1 to train the deep ensemble model including an ensemble of classifiers in each layer. The deep ensemble model then will be used to predict class labels for the observations in the validation set. Since the ground truth of validated observations is known in advance, we can calculate the fitness value of the particle as the classification accuracy of the deep ensemble model on the validation data. We also update *pbest* of these particles and calculate learning probability Pc for all particles by using (12). In this study, we use the algorithm introduced in [36] to assign exemplars for all particles.

The VLPSO algorithm runs until reaching the last iterations. In each iteration, for each particle, we do three tasks (i) check whether its exemplar vector needs to be renewed by using a renewExemplars flag (ii) update its velocity and position by using (10) and (11) and (iii) calculate its fitness and update its *pbest*. If *pbest* is not improved after a number of iterations, we mark renew for its exemplar in the next iteration by setting its renewExemplars flag to true. Finally, we update the value of *gbest*. The length changing procedure is conducted if *gbest* is not improved for several times. The VLPSO algorithm returns the particle associated with *gbest* choosing from the last iteration.

The summary of notations used in this section is shown in Table 2.

4. Experimental studies

4.1. Configurations

We designed the experiments to address two questions:

- Which is a suitable feature generation approach for DEFEG?
- Does DEFEG perform better than several well-known benchmark algorithms?

For the first question, we compared the performance of DEFEG with two proposed approaches in generating input features for the subsequent layer introduced in (5) and (6). This would help to choose a suitable one for practice. We experimented and compared the performances of DEFEG using 5 classifiers in each layer: Naïve Bayes (using Gaussian distribution to approximate the likelihood distribution), Random Forest (using 200 classifiers), XgBoost (using 200 classifiers), K Nearest Neighbors (K was set to 5), and Logistic Regression. These classifiers were implemented from the scikit-learn library.

To design the experiment for the second question, we compared DEFEG to four well-known ensemble methods namely Random Forest [13], Completely Random Tree Forest [4], Rotation Forest [15], and XgBoost [14] in which we used 200 classifiers for each method by referencing [2,12]. We also compared DEFEG to two multiple layer models namely gcForest and Multiple Layer Perceptron (MLP). In each layer of gcForest, we used 2 Random Forests and 2 Completely Random Tree Forests like the experiments in [4]. Since the performance of MLP significantly depends on the network structure, we experimented with these methods on a number of different network configurations and then reported the best result among all configurations for comparison.

Table 3
The experimental datasets.

Dataset name	# of training observations	# of testing observations	# of classes	# of dimensions
Abalone	2 921	1 253	3	8
Artificial	490	210	2	10
Australian	483	207	2	14
Balance	437	188	3	4
Breast-Tissue	74	32	6	9
Bupa	241	104	2	6
Cleveland	207	90	5	13
Conn-bench-vowel	369	159	11	10
Electricity	31 718	13 594	2	8
Embryonal	42	18	2	7129
Fertility	70	30	2	9
Hayes-roth	112	48	3	4
Heart	189	81	2	13
Hill_Valley	1 696	728	2	100
Leukemia	50	22	2	7129
Madelon	1 400	600	2	500
Magic	13 314	5 706	2	10
Musk1	333	143	2	166
Musk2	4 618	1 980	2	166
Newthyroid	150	65	3	5
Page-Blocks	3 830	1 642	5	10
Penbased	7 694	3 298	10	16
Phoneme	3 782	1 622	2	5
Pima	537	231	2	8
Ring	5 180	2 220	2	20
Satimage	4 504	1 931	6	36
Skin_NonSkin	171 539	73 518	2	3
Sonar	145	63	2	60
Spambase	3 220	1 381	2	57
Svmguide2	273	118	3	20
Texture	3 850	1 650	10	40
Tic-tac-toe	670	288	2	9
Twonorm	5 180	2 220	2	20
Vehicle	592	254	4	18
Vertebral	217	93	3	6
Waveform_w_noise	3 500	1 500	3	40
Waveform_wo_noise	3 500	1 500	3	21
Wdbc	398	171	2	30
Wine_red	1 119	480	6	11
Wine_white	3 428	1 470	7	11

MLP was constructed with different configurations: input-30-20-output, input-50-30-output, and input-70-50-output by referencing the experiments in [4]. Also, all the network configurations use ReLU as activation function, and their weights are found by minimizing the log-loss function using Adam algorithm. DEFEG was compared to two deep learning models, namely DeepFM [40] and WideDeep [41], for tabular data. WideDeep learning model aims to leverage the benefits of both wide linear models and deep neural networks. Specifically, WideDeep employs a wide component of a generalized linear model to effectively memorize sparse feature interactions using cross-product feature transformations. That wide component is trained jointly with a feed-forward neural network, to generalize previously hidden feature interactions via low dimensional embeddings. DeepFM meanwhile was proposed by combining the advantages of factorization-machine-based recommendation and feature learning in deep models, to exploit both low- and high-order feature interactions. Essentially, DeepFM jointly trains an FM component, which models order-1 and order-2 feature interactions, and a deep component, which is employed to capture high-order feature interactions. In this study, we used DeepTables library (https://deeptables.readthedocs.io/en/latest/index.html) with default parameters to train WideDeep and DeepFM.

For DEFEG, we used the 5-fold Cross-Validation in each layer to generate the predictions for the training data. We followed the experiments of gcForest by choosing 20% of the training data for validation [4]. We also initialized the parameters for the VLPSO. The maximum number of layers was set to 5. The maximum number of iterations and the population size were set to 50. The inertia weight a is updated at the tth iteration as [29]:

$$a = a_{max} - \frac{(a_{max} - a_{min})\, t}{maxT} \tag{13}$$

in which $a_{max} = 0.9$, $a_{min} = 0.5$. Meanwhile, c was set to 1.49445 based on the experiments in [29]. The length changing frequency was set to 2/3 of the maximum number of iterations. Finally, the number of iterations used to update the exemplars was set to 10. Although careful task-specific tuning may bring better performance, here we used the same settings for experiments on all datasets to show that DEFEG performs better than the benchmark algorithms even with the simple settings.

The DEFEG and benchmark algorithms experimented on 40 datasets collected from UCI Machine Learning Repository. The information of the datasets is present in Table 3.

4.2. Experimental results

Comparing two different feature generation approaches: We first compared the performance of DEFEG with two feature generation approaches given in (5) and (6) for classification accuracy and F1 score, respectively. We denoted DEFEG (Concatenation) for the approach using the concatenation between the weighted predictions and the original training data in (5) and DEFEG (Prediction) for the approach using the weighted predictions only in (6). Figs. 5 and 6 show the performance of DEFEG on 40 experimental datasets. In general, DEFEG (Concatenation)

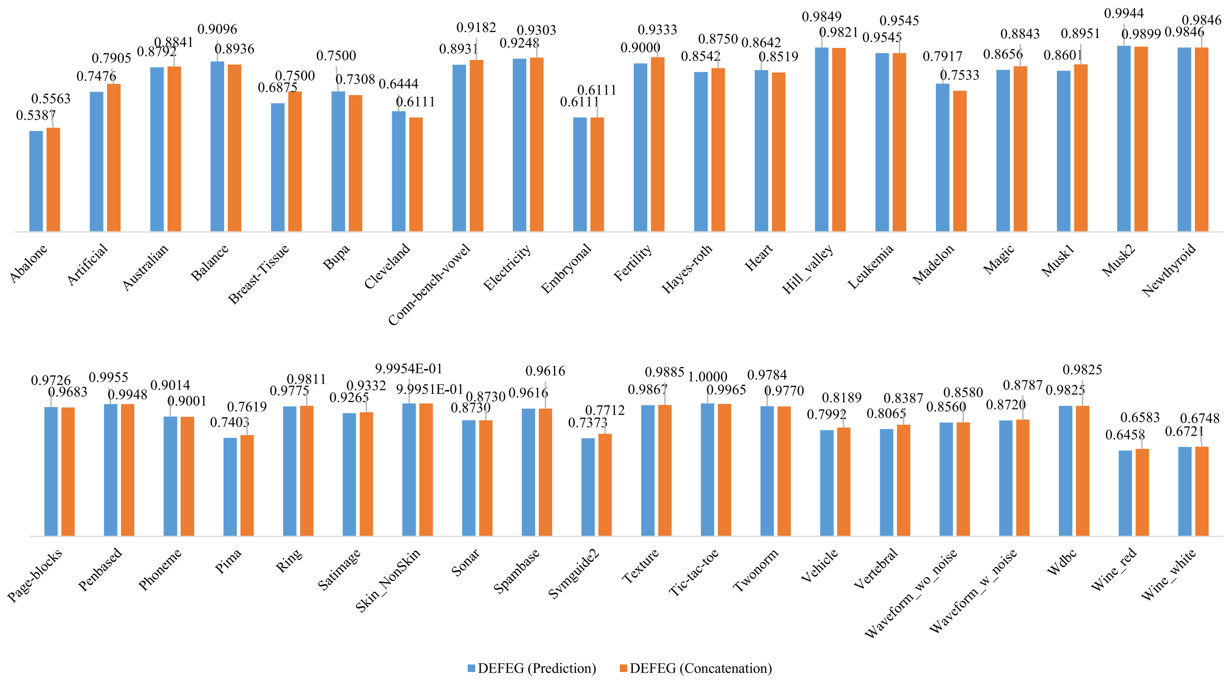


Fig. 5. The comparison between DEFEG (Prediction) and DEFEG (Concatenation) on classification accuracy.

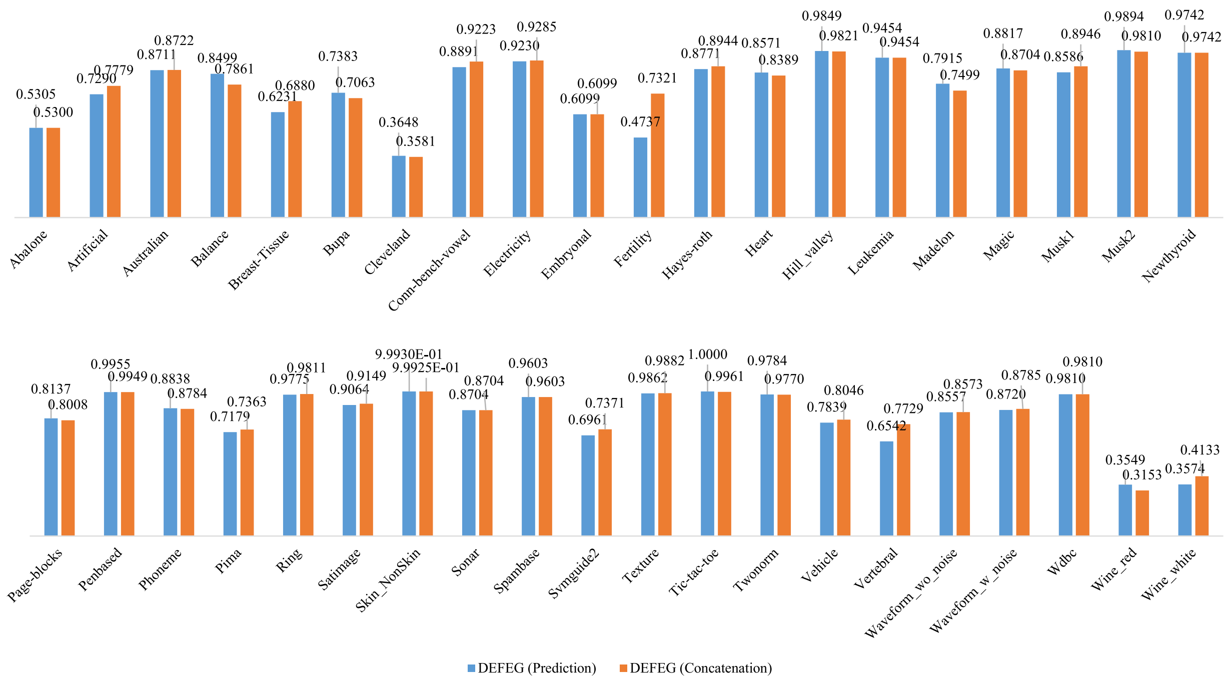


Fig. 6. The comparison between DEFEG (Prediction) and DEFEG (Concatenation) on F1 score.

is slightly better than DEFEG (Prediction) for both performance metrics.

For classification accuracy, the outstanding performances of DEFEG (Concatenation) over DEFEG (Prediction) are reported on 21 datasets while two approaches perform equally on 6 datasets. On 6 datasets namely Artificial, Breast-Tissue, Fertility, Musk1, Svmguide2, and Vertebral, DEFEG (Concatenation) is better than DEFEG (Prediction) for more than 3%. Meanwhile, DEFEG (Prediction) is significantly outperformed DEFEG (Concatenation) on only 2 datasets namely Cleveland (3.3% better, 0.6444 vs. 0.6111) and Madelon (3.8% better, 0.7917 vs. 0.7533).

For F1 score, DEFEG (Concatenation) is better than DEFEG (Prediction) on 18 datasets in which the significant differences in the performance are presented on 8 datasets (more than 3% better). For example, on Fertility and Vertebral dataset, F1 scores of DEFEG (Concatenation) are more than 25% and 11% higher than those of DEFEG (Prediction). DEFEG (Concatenation) meanwhile is equal and worse than DEFEG (Prediction) on 5 datasets and 17

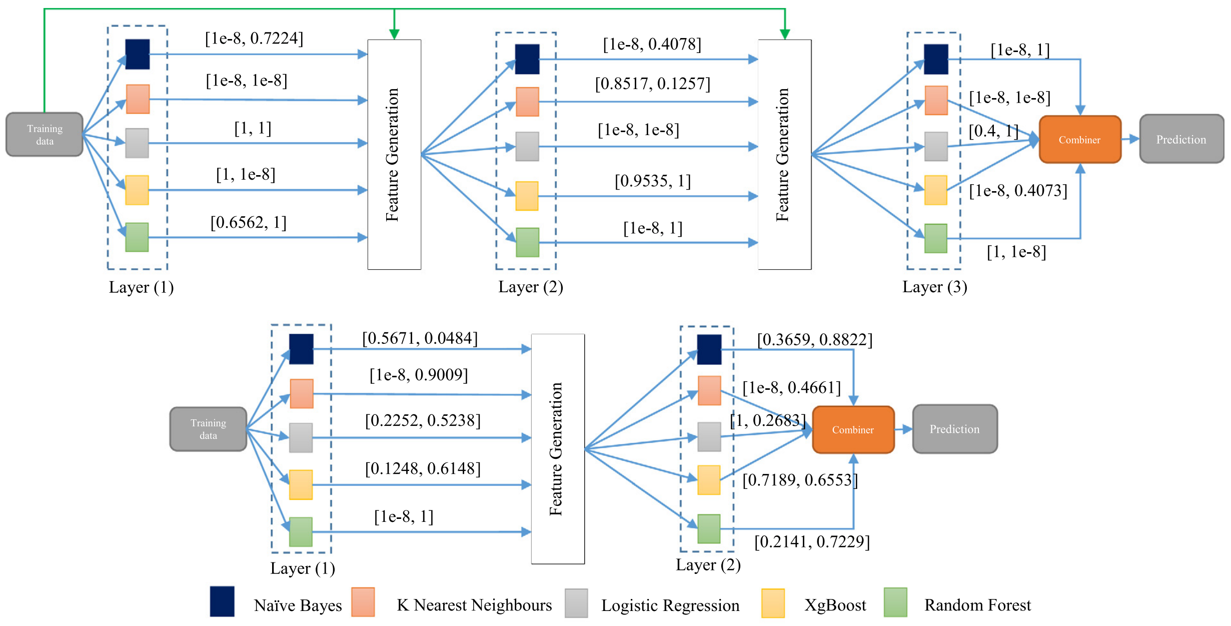


Fig. 7. The illustration of obtained configurations of DEFEG with two feature generation approaches for Musk2 dataset. * The above figure is the configuration obtained by DEFEG (Concatenation) (the green line from the training data shows this concatenation). The below figure is the configuration obtained by using DEFEG (Prediction).

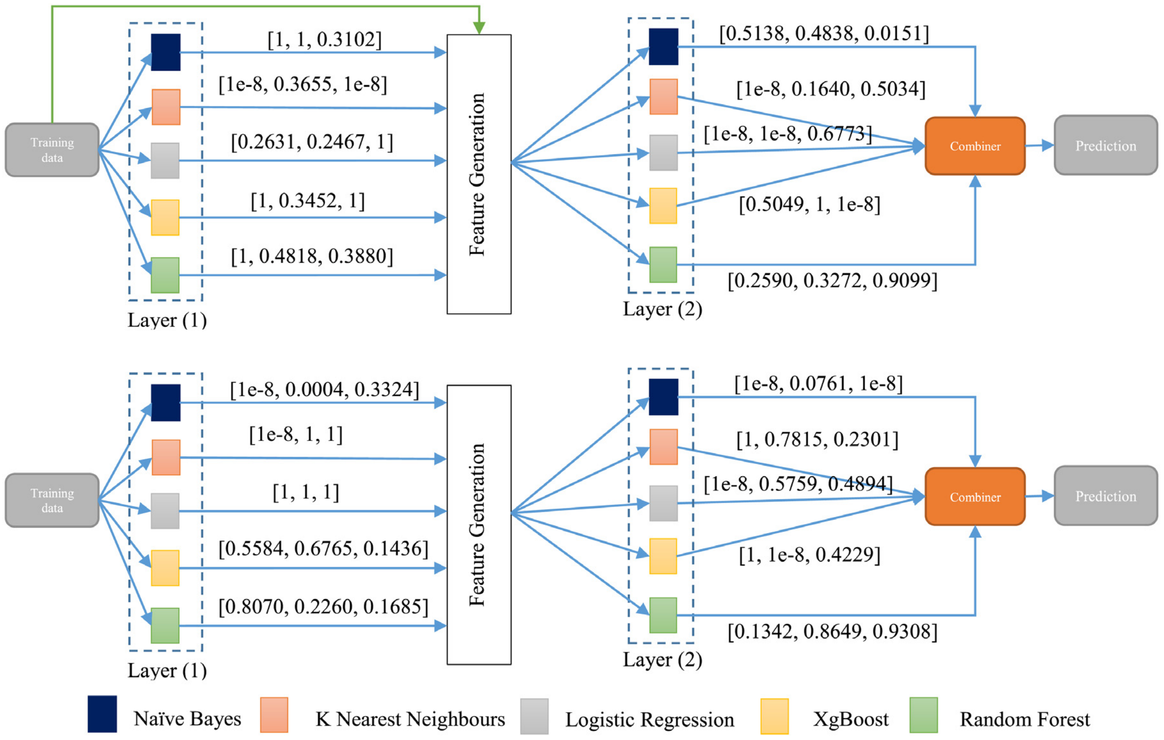


Fig. 8. The illustration of obtained configurations of DEFEG with two feature generation approaches for Hayes-roth dataset. * The above figure is the configuration obtained by DEFEG (Concatenation) (the green line from the training data shows this concatenation). The below figure is the configuration obtained by using DEFEG (Prediction).

datasets, respectively. DEFEG (Prediction) is significantly better than DEFEG (Concatenation) on some datasets like Balance (6.4% better, 0.8499 vs. 0.7861), Bupa (3.2% better, 0.7383 vs. 0.7063), Madelon (4.2% better, 0.7915 vs. 0.7499), and Wine_red (3.96% better, 0.3549 vs. 0.3153).

Figs. 7 and 8 show the obtained configurations of two feature generation approaches on Musk2 and Hayes-roth dataset. For 2-class label Musk2 dataset, we acquired a 3-layer model with DEFEG (Concatenation) and a 2-layer model with DEFEG (Prediction). In the first layer of the model of DEFEG (Concatenation), the

Table 4
The classification accuracy of DEFEG and the benchmark algorithms.

Data	Completely Random Tree Forest	gcForest	Random Forest	XgBoost	MLP	Rotation Forest	DEFEG	DeepFM	WideDeep
Abalone	0.5235 (9)	0.5387 (5)	0.5323 (6)	0.5499 (4)	0.5611 (1)	0.5515 (3)	0.5563 (2)	0.5243 (8)	0.5299 (7)
Artificial1	0.7143 (6)	0.7905 (2)	0.7905 (2)	0.7619 (5)	0.6476 (7)	0.781 (4)	0.7905 (2)	0.5 (9)	0.6238 (8)
Australian	0.8599 (6)	0.8792 (4)	0.8889 (1)	0.8744 (5)	0.7005 (8)	0.8841 (2.5)	0.8841 (2.5)	0.5072 (9)	0.8068 (7)
Balance	0.8564 (5)	0.8564 (5)	0.8085 (8)	0.8457 (7)	0.9734 (1)	0.8564 (5)	0.8936 (2)	0.7287 (9)	0.8883 (3)
Breast-tissue	0.7188 (3.5)	0.6875 (6)	0.7188 (3.5)	0.7188 (3.5)	0.625 (7)	0.7188 (3.5)	0.75 (1)	0.1563 (8)	0.125 (9)
Bupa	0.7308 (2)	0.7212 (4)	0.7308 (2)	0.7019 (6.5)	0.7115 (5)	0.7019 (6.5)	0.7308 (2)	0.5769 (8)	0.4615 (9)
Cleveland	0.6222 (2.5)	0.6222 (2.5)	0.6333 (1)	0.5889 (5)	0.5 (7)	0.5333 (6)	0.6111 (4)	0.1 (9)	0.1333 (8)
Conn-bench-vowel	0.6478 (5)	0.6415 (6)	0.6101 (7)	0.8239 (4)	0.8365 (3)	0.956 (1)	0.9182 (2)	0.1258 (9)	0.1698 (8)
Electricity	0.7725 (9)	0.7998 (7)	0.7745 (8)	0.8529 (3)	0.805 (4)	0.8955 (2)	0.9303 (1)	0.8049 (5)	0.8035 (6)
Embryonal	0.5 (7)	0.5 (7)	0.3889 (9)	0.5 (7)	0.6111 (2.5)	0.5556 (4.5)	0.6111 (2.5)	0.5556 (4.5)	0.6667 (1)
Fertility	0.9 (5.5)	0.9 (5.5)	0.9 (5.5)	0.9 (5.5)	0.9 (5.5)	0.9333 (1.5)	0.9333 (1.5)	0.8667 (9)	0.9 (5.5)
Hayes-roth	0.75 (6)	0.7292 7)	0.8333 (2.5)	0.8125 (4)	0.7708 (5)	0.8333 (2.5)	0.875 (1)	0.625 (8)	0.375 (9)
Heart	0.8148 (4.5)	0.8272 (3)	0.8025 (6.5)	0.7531 (9)	0.8642 (1)	0.8025 (6.5)	0.8519 (2)	0.7901 (8)	0.8148 (4.5)
Hill-valley	0.544 7)	0.5824 (5)	0.5755 (6)	0.6126 (4)	0.8942 (3)	0.978 (2)	0.9821 (1)	0.4698 (9)	0.5124 (8)
Leukemia	0.7727 (8.5)	0.9091 (4.5)	0.9545 (1.5)	0.9091 (4.5)	0.9091 (4.5)	0.9091 (4.5)	0.9545 (1.5)	0.8182 (7)	0.7727 (8.5)
Madelon	0.5433 7)	0.635 (4)	0.62 (5)	0.7 (3)	0.545 (6)	0.8133 (1)	0.7533 (2)	0.5417 (8)	0.5317 (9)
Magic	0.8267 (8)	0.8375 (7)	0.8228 (9)	0.8735 (3)	0.8396 (6)	0.8796 (2)	0.8843 (1)	0.8531 (5)	0.8547 (4)
Musk1	0.7762 (9)	0.8531 (3)	0.8042 (8)	0.8322 (6)	0.8252 (7)	0.8462 (4.5)	0.8951 (1)	0.8462 (4.5)	0.8811 (2)
Musk2	0.8449 (9)	0.951 (7)	0.8944 (8)	0.9768 (4)	0.9798 (3)	0.9808 (2)	0.9899 (1)	0.9652 (6)	0.9717 (5)
Newthyroid	0.9538 (6.5)	0.9692 (3.5)	0.9538 (6.5)	0.9692 (3.5)	0.9846 (1.5)	0.9538 (6.5)	0.9846 (1.5)	0.9538 (6.5)	0.7385 (9)
Page-blocks	0.9214 (9)	0.9452 (6)	0.9525 (5)	0.9683 (2.5)	0.9629 (4)	0.9714 (1)	0.9683 (2.5)	0.9348 (8)	0.9379 (7)
Penbasd	0.8472 (6)	0.8957 (5)	0.8357 (7)	0.9897 (4)	0.99 (3)	0.9955 (1)	0.9948 (2)	0.7216 (8)	0.6595 (9)
Phoneme	0.8002 (7)	0.8181 (5)	0.8033 (6)	0.857 (4)	0.8631 (3)	0.8859 (2)	0.9001 (1)	0.7879 (8)	0.7707 (9)
Pima	0.7446 (3.5)	0.7359 (6)	0.7576 (2)	0.7186 (9)	0.7316 (7.5)	0.7403 (5)	0.7619 (1)	0.7446 (3.5)	0.7316 (7.5)
Ring	0.9446 7)	0.9635 (6)	0.927 (8)	0.9707 (3)	0.8432 (9)	0.968 (4)	0.9811 (1)	0.9649 (5)	0.9721 (2)
Satimage	0.8493 (6)	0.8695 (4)	0.8576 (5)	0.912 (3)	0.8364 (7)	0.9259 (2)	0.9332 (1)	0.8276 (8)	0.8141 (9)
Skin_Nonskin	0.9655 (8.5)	0.9829 (7)	0.9655 (8.5)	0.9977 (4)	0.99914 (3)	0.99955 (1)	0.99951 (2)	0.9837 (6)	0.9868 (5)
Sonar	0.8254 (8.5)	0.8254 (8.5)	0.8413 (6)	0.8413 (6)	0.8889 (1)	0.8413 (6)	0.873 (3)	0.873 (3)	0.873 (3)
Spambase	0.929 6)	0.9385 (4)	0.9254 (8)	0.9544 (2)	0.9356 (5)	0.9493 (3)	0.9616 (1)	0.9261 (7)	0.9232 (9)
Svmguide2	0.7458 (3)	0.6949 (9)	0.7373 (6)	0.7458 (3)	0.7034 (8)	0.7373 (6)	0.7712 (1)	0.7458 (3)	0.7373 (6)
Texture	0.8 (7)	0.8436 (5)	0.8345 (6)	0.9812 (4)	0.9933 (1)	0.9927 (2)	0.9885 (3)	0.6394 (9)	0.6448 (8)
Tic-Tac-Toe	0.7153 (9)	0.816 (5)	0.8056 (6.5)	0.9861 (2)	0.9271 (4)	0.9375 (3)	0.9965 (1)	0.7951 (8)	0.8056 (6.5)
Twonorm	0.9662 (8)	0.9671 (7)	0.9604 (9)	0.9734 (5)	0.9707 (6)	0.9811 (1)	0.977 (3)	0.9761 (4)	0.9779 (2)
Vehicle	0.689 (6.5)	0.7283 (4)	0.7126 (5)	0.7756 (3)	0.689 (6.5)	0.815 (2)	0.8189 (1)	0.622 (9)	0.6378 (8)
Vertebra	0.7957 (7.5)	0.7957 (7.5)	0.8495 (2)	0.828 (5)	0.7957 (7.5)	0.871 (1)	0.8387 (3.5)	0.7957 (7.5)	0.8387 (3.5)
Waveform_wo_Noise	0.8267 (7)	0.8473 (4)	0.838 (6)	0.8487 (3)	0.8433 (5)	0.86 (1)	0.858 (2)	0.75 (9)	0.778 (8)
Waveform_w_Noise	0.8087 (7)	0.8573 (4)	0.8447 (5)	0.8667 (3)	0.8367 (6)	0.872 (2)	0.8787 (1)	0.772 (8)	0.74 (9)
Wdbc	0.9649 (6.5)	0.9649 (6.5)	0.9591 (8)	0.9766 (3)	0.9532 (9)	0.9766 (3)	0.9825 (1)	0.9766 (3)	0.9708 (5)
Wine_red	0.575 (6)	0.5854 (5)	0.5938 (4)	0.6375 (3)	0.5646 (9)	0.6583 (1.5)	0.6583 (1.5)	0.5688 (8)	0.5708 (7)
Wine_white	0.5299 (8)	0.5707 (4)	0.5524 (6)	0.6122 (3)	0.5163 (9)	0.685 (1)	0.6748 (2)	0.5571 (5)	0.5517 (7)
Average	0.7729	0.7969	0.7898	0.8250	0.8082	0.8507	0.8650	0.7068	0.7121
Average ranking	6.575	5.2625	5.625	4.275	5.0375	2.9875	1.725	6.9875	6.525

weights associated with the predictions of K Nearest Neighbors are [1e−8, 1e−8] which means the predictions of this classifiers are set to 0 in the input training data for the second layer. All predictions of Logistic Regression are used in generating the training data for the second layer but after the second layer, because the weight vector of Logistic Regression is [1e−8, 1e−8] which means the predictions of this classifier are set to 0 in the training data for the third layer.

On the 3-class Hayes-roth dataset, both models of DEFEGs have 2 layers with different weights. For example, in the model of DEFEG (Concatenation), the weight vector of Naïve Bayes in the first layer is [1, 1, 0.3102] for 3 class labels, that means the predictions for 1st and 2nd class label are used with original values while 31.02% of the predictions for 3rd class label are used to generate new training data. Meanwhile, in the model of DEFEG (Prediction), the weight vector of Naïve Bayes in the first layer is [1e−8, 0.0004, 0.3324] for 3 class labels, that means only 33.24% of the predictions for 3rd class label is used to generate new training data while the predictions for 1st and 2nd class labels are set to 0.

The choosing of DEFEG (Concatenation) or DEFEG (Prediction) thus is somewhat data-dependent. Although DEFEG (Concatenation) is slightly better than DEFEG (Prediction) in general, using DEFEF (Prediction) may bring benefits concerning the training time in some cases. By concatenating the original training data and weighted predictions in DEFEG (Concatenation), the dimension of new training data for the subsequent layer would be $D+M\times K$ which is much higher than those of DEFEG (Prediction) in cases the original data is given in high dimensions i.e. D is a large number, resulting in high training cost. On Musk 2, for example, the training process of DEFEG (Concatenation) and DEFEG (Prediction) takes 7 and 2.5 h, respectively.

In the next section, we used the experimental results of DEFEG (Concatenation) to compare to those of the benchmark algorithms.

Comparison with benchmark algorithms: Tables 4 and 5 shows the classification accuracy, F1 score, and related ranking of DEFEG and the benchmark algorithms on the experimental datasets. For classification accuracy, DEFEG is better than all benchmark algorithms except Rotation Forest based on the Nemenyi posthoc test (see Fig. 9). DEFEG ranks first with the rank value 1.725 . Rotation Forest ranks second with a rank value of 2.9875, followed by XgBoost (rank value 4.275) and MLP (rank value 5.0375). DeepFM and Completely Random Tree Forest are the two poorest methods in our experiment with rank value 6.9125 and 7.0875 respectively.

In detail, DEFEG ranks first or shares the first ranking on 21 datasets (52.5%) and ranks second or shares the second-ranking

Table 5
The F1 score of DEFEG and the benchmark algorithms.

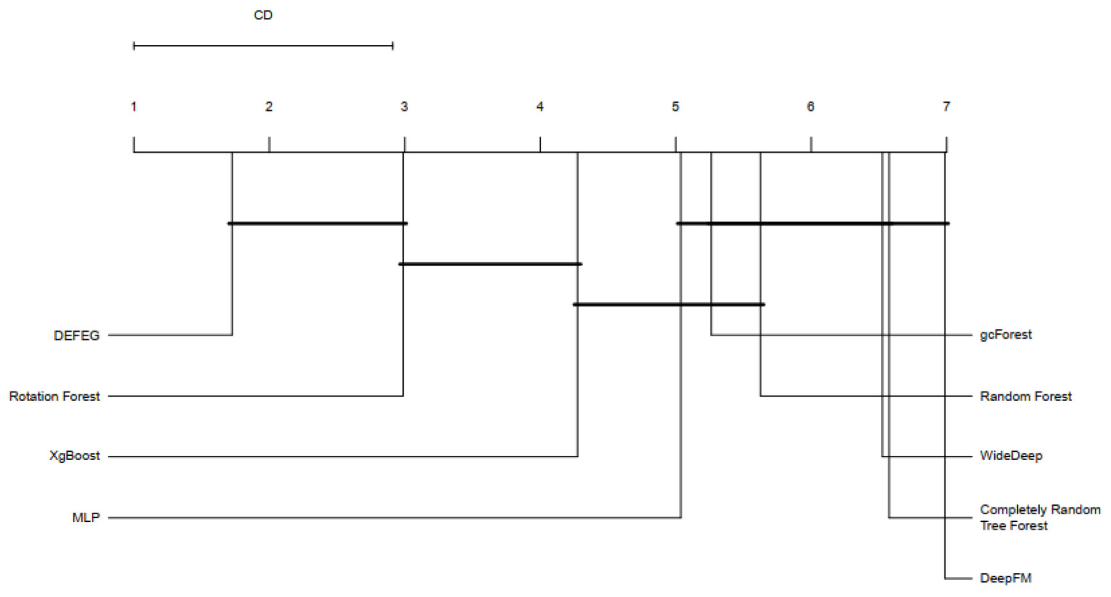
Data	Completely Random Tree Forest	gcForest	Random Forest	XgBoost	MLP	Rotation Forest	DEFEG	DeepFM	WideDeep
Abalone	0.4586 (8)	0.5052 (5)	0.4801 (6)	0.5343 (3)	0.5456 (2)	0.5522 (1)	0.53 (4)	0.4371 (9)	0.4654 (7)
Artificial	0.65 (6)	0.7608 (4)	0.769 (2)	0.7464 (5)	0.6 (7)	0.7654 (3)	0.7779 (1)	0.4986 (9)	0.5654 (8)
Australian	0.848 (6)	0.8704 (4)	0.882 (1)	0.8669 (5)	0.6635 (8)	0.8778 (2)	0.8722 (3)	0.3826 (9)	0.7972 (7)
Balance	0.6004 (6)	0.6985 (3)	0.5664 (9)	0.597 (8)	0.9578 (1)	0.5996 (7)	0.7861 (2)	0.626 (4)	0.6217 (5)
Breast-Tissue	0.6989 (3)	0.6389 (6)	0.7037 (1)	0.6989 (3)	0.5781 (7)	0.6989 (3)	0.688 (5)	0.045 (8)	0.037 (9)
Bupa	0.6986 (3)	0.6899 (7)	0.7063 (1.5)	0.6937 (5)	0.6916 (6)	0.6956 (4)	0.7063 (1.5)	0.3659 (9)	0.3786 (8)
Cleveland	0.2418 (7)	0.2559 (5)	0.2571 (4)	0.3016 (2)	0.2456 (6)	0.272 (3)	0.3581 (1)	0.0367 (9)	0.0689 (8)
Conn-bench-vowel	0.6356 (6)	0.6442 (5)	0.6134 (7)	0.8208 (4)	0.8288 (3)	0.9539 (1)	0.9223 (2)	0.0621 (9)	0.0863 (8)
Electricity	0.7532 (9)	0.7961 (5.5)	0.7571 (8)	0.8483 (3)	0.8002 (4)	0.8925 (2)	0.9285 (1)	0.7961 (5.5)	0.7937 (7)
Embryonal	0.3333 (9)	0.4109 (7)	0.3378 (8)	0.4582 (5)	0.5786 (3)	0.5325 (4)	0.6099 (2)	0.4462 (6)	0.6667 (1)
Fertility	0.4737 (6.5)	0.4737 (6.5)	0.4737 (6.5)	0.6727 (3)	0.6296 (4)	0.7321 (1.5)	0.7321 (1.5)	0.4643 (9)	0.4737 (6.5)
Hayes-roth	0.7889 (5)	0.7718 (6)	0.8561 (3)	0.8262 (4)	0.7674 (7)	0.8593 (2)	0.8944 (1)	0.5957 (8)	0.295 (9)
Heart	0.797 (5)	0.8056 (3)	0.7817 (7)	0.7271 (9)	0.8571 (1)	0.791 (6)	0.8389 (2)	0.7765 (8)	0.8052 (4)
Hill_valley	0.5435 (7)	0.5803 (5)	0.5755 (6)	0.6122 (4)	0.8942 (3)	0.978 (2)	0.9821 (1)	0.4283 (8)	0.3462 (9)
Leukemia	0.6508 (9)	0.8854 (5)	0.9454 (1.5)	0.8854 (5)	0.9018 (3)	0.8854 (5)	0.9454 (1.5)	0.7412 (7)	0.727 (8)
Madelon	0.5377 (7)	0.6248 (4)	0.6196 (5)	0.6996 (3)	0.4949 (9)	0.8129 (1)	0.7499 (2)	0.5417 (6)	0.5299 (8)
Magic	0.7898 (9)	0.8177 (6)	0.7959 (8)	0.8551 (3)	0.8119 (7)	0.8634 (2)	0.8704 (1)	0.8313 (4)	0.8271 (5)
Musk1	0.7652 (9)	0.8513 (3)	0.798 (8)	0.8292 (6)	0.8209 (7)	0.8434 (4.5)	0.8946 (1)	0.8434 (4.5)	0.8803 (2)
Musk2	0.458 (9)	0.8968 (7)	0.7163 (8)	0.9538 (4)	0.9606 (3)	0.9617 (2)	0.981 (1)	0.9273 (6)	0.9431 (5)
Newthyroid	0.926 (6.5)	0.9453 (3.5)	0.926 (6.5)	0.9453 (3.5)	0.9742 (1.5)	0.926 (6.5)	0.9742 (1.5)	0.926 (6.5)	0.4734 (9)
Page-blocks	0.4631 (7)	0.6302 (6)	0.6745 (5)	0.7875 (3)	0.7746 (4)	0.8302 (1)	0.8008 (2)	0.3517 (9)	0.3891 (8)
Penbased	0.8442 (6)	0.8961 (5)	0.8318 (7)	0.9897 (4)	0.99 (3)	0.9955 (1)	0.9949 (2)	0.683 (8)	0.6474 (9)
Phoneme	0.7474 (7)	0.7822 (5)	0.7566 (6)	0.828 (4)	0.8395 (3)	0.8615 (2)	0.8784 (1)	0.734 (8)	0.71 (9)
Pima	0.6989 (6)	0.723 (3)	0.7255 (2)	0.6952 (7)	0.6741 (8)	0.7164 (4)	0.7363 (1)	0.7078 (5)	0.6617 (9)
Ring	0.9445 (7)	0.9635 (6)	0.9266 (8)	0.9707 (3)	0.8429 (9)	0.968 (4)	0.9811 (1)	0.9649 (5)	0.9721 (2)
Satimage	0.7927 (6)	0.8322 (4)	0.8092 (5)	0.8932 (3)	0.792 (7)	0.9092 (2)	0.9149 (1)	0.7196 (8)	0.7121 (9)
Skin_NonSkin	0.9481 (8.5)	0.9745 (7)	0.9481 (8.5)	0.9965 (4)	0.9987 (3)	0.99932 (1)	0.99925 (2)	0.9756 (6)	0.9804 (5)
Sonar	0.8225 (8)	0.8209 (9)	0.838 (6.5)	0.8393 (5)	0.8871 (1)	0.838 (6.5)	0.8704 (4)	0.8722 (2.5)	0.8722 (2.5)
Spambase	0.926 (6)	0.9364 (4)	0.9225 (8)	0.9529 (2)	0.9338 (5)	0.9476 (3)	0.9603 (1)	0.9237 (7)	0.9212 (9)
Svmguide2	0.5834 (7)	0.6017 (6)	0.6157 (5)	0.6825 (2)	0.4729 (9)	0.6535 (4)	0.7371 (1)	0.5241 (8)	0.6602 (3)
Texture	0.787 (7)	0.8378 (5)	0.8258 (6)	0.9804 (4)	0.9929 (1)	0.9925 (2)	0.9882 (3)	0.5561 (8)	0.5501 (9)
Tic-tac-toe	0.5328 (9)	0.7715 (5)	0.7383 (6)	0.9841 (2)	0.9138 (4)	0.9259 (3)	0.9961 (1)	0.7085 (8)	0.7305 (7)
Twonorm	0.9662 (8)	0.9671 (7)	0.9604 (9)	0.9734 (5)	0.9707 (6)	0.9811 (1)	0.977 (3)	0.9761 (4)	0.9779 (2)
Vehicle	0.649 (7)	0.7037 (4)	0.6692 (6)	0.7639 (3)	0.6742 (5)	0.799 (2)	0.8046 (1)	0.5404 (9)	0.5633 (8)
Vertebral	0.6706 (9)	0.6809 (7)	0.766 (3)	0.7395 (4)	0.7025 (6)	0.813 (1)	0.7729 (2)	0.6755 (8)	0.7271 (5)
Waveform_wo_noise	0.8206 (7)	0.8464 (4)	0.8365 (6)	0.8481 (3)	0.8427 (5)	0.8592 (1)	0.8573 (2)	0.7473 (9)	0.7742 (8)
Waveform_w_noise	0.8025 (7)	0.8571 (4)	0.8439 (5)	0.8667 (3)	0.8367 (6)	0.872 (2)	0.8785 (1)	0.766 (8)	0.7373 (9)
Wdbc	0.9618 (7)	0.962 (6)	0.9556 (8)	0.9749 (2)	0.949 (9)	0.9747 (3.5)	0.981 (1)	0.9747 (3.5)	0.9685 (5)
Wine_red	0.2107 (9)	0.262 (5)	0.2517 (8)	0.3535 (2)	0.2577 (7)	0.3608 (1)	0.3153 (3)	0.265 (4)	0.2584 (6)
Wine_white	0.1666 (9)	0.2314 (6)	0.2005 (8)	0.3273 (3)	0.217 (7)	0.4136 (1)	0.4133 (2)	0.239 (4)	0.2363 (5)
Average	0.6747	0.7301	0.7164	0.7755	0.7541	0.8051	0.8225	0.6169	0.6208
Average ranking	7.0875	5.2125	5.825	3.8875	5.0125	2.7125	1.775	6.9125	6.575

on 14 datasets (35%). On the datasets where DEFEG ranks first, the significant differences between the classification of DEFEG and that of the second rank methods are presented on some datasets like Breast-Tissue (3.12%, 0.75 vs. 0.7188), Electricity (3.48%, 0.9303 vs. 0.8955), Hayes-roth (4.17%, 0.875 vs. 0.8333). DEFEG performs poorly compared to the first rank method on 4 datasets: Balance (0.8936 vs. 0.9734 of MLP), Conn-bench-vowel (0.9182 vs. 0.956 of Rotation Forest), Madelon (0.7533 vs. 0.8133 of Rotation Forest), and Vertebral (0.8387 vs. 0.871 of Rotation Forest). This demonstrates the advantage of DEFEG comparing to the state-of-the-art ensemble methods and multi-layer based methods.

Rotation Forest is the second-best method in our experiment. This ensemble method ranks first or shares the first rank on 11 datasets. Based on the Nemenyi test on the experimental datasets, Rotation Forest is better than MLP, gcForest, Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM. Rotation Forest creates an ensemble of classifiers on many new training data generated from the subset of the original one with PCA transformation. This ensemble generation makes new training data richness of diversity, resulting in high performance on several datasets. Meanwhile, although there is no difference between Rotation Forest and DEFEG based on the Nemenyi test, DEFEG is significantly better than Rotation Forest on many datasets, for example, more than up to 5% better on Cleveland, Heart, Leukemia, Musk1, and Tic-tac-toe.

XgBoost does not rank first on any datasets although this ensemble method ranks third in our experiment. Moreover, it is noted that XgBoost does not win DEFEG on any datasets. The significant difference in the classification accuracy of XgBoost and DEFEG is presented on many datasets such as Balance (0.8936 vs. 0.8453), Conn-bench-vowel (0.9182 vs. 0.8239), Electricity (0.9303 vs. 0.8529), Embryonal (0.6111 vs. 0.5), Hayes-roth (0.875 vs. 0.84125), Heart (0.8519 vs. 0.7531), Hill_valley (0.9821 vs. 0.6126), Leukemia (0.9545 vs. 0.9091), Madelon (0.7533 vs. 0.7), Musk1 (0.8951 vs. 0.8322), Phoneme (0.9001 vs. 0.857), Pima (0.7619 vs. 0.7186), Vehicle (0.8189 vs. 0.7756), and Wine_white (0.6748 vs. 0.6122).

MLP is poorer than DEFEG even though we conducted the experiment on the set of parameters to report the best result. On the 5 datasets (Artificial, Australian, Breast-Tissue, Cleveland, and Ring), MLP is significantly lower classification accuracy than



DEFEG is better than XgBoost, MLP, gcForest, Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM

Rotation Forest is better than MLP, gcForest, Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM

XgBoost is better than WideDeep, Completely Random Tree Forest, and DeepFM

Fig. 9. The Nemenyi test result on classification accuracy. DEFEG is better than XgBoost, MLP, gcForest, Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM Rotation Forest is better than MLP, gcForest, Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM XgBoost is better than WideDeep, Completely Random Tree Forest, and DeepFM.

not only DEFEG but also the other methods. DEFEG is only significantly poorer than MLP on the Balance dataset while it is far better than MLP on many datasets.

gcForest's rank is higher than those of Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM in our experiment. Although gcForest is better than or equal to its members i.e. Random Forest and Completely Random Tree Forest on 27 datasets, respectively, it is by far worse than DEFEG. The outstanding performance of DEFEG over gcForest is from the proposed augmented features in which the weighted integration is done on the predictions to generate better-augmented features for the subsequent layer. By using the VLPSO algorithm, we can obtain the optimal weights that make classification accuracy on the validate set be better, highly resulting in improving performance on test data. This demonstrates the advantages of weighted feature generation of DEFEG compared to the multi-layer model of gcForest.

Random Forest, WideDeep, Completely Random Tree Forest, and DeepFM are four poorest methods in the experiment. Completely Random Tree Forest yielded the lowest classification accuracy among all methods on 9 datasets. The poor performance of Completely Random Tree Forest can be explained based on its mechanism in which only one feature is considered when looking for the best split for node samples that reduce the flexibility of the tree generation procedure. Random Forest yielded the lowest classification accuracy among all methods on only 4 datasets while this ensemble method ranks first on 3 datasets namely Australian, Cleveland, and Leukemia. It is noticed that not only Random Forest and Completely Random Tree Forest but also gcForest have difficulties in some datasets likes Conn-bench-vowel, Electricity, Embryonal, Madelon, Penbased, Phoneme, Satimage, Texture, Tic-tac-toe, Hill_valley in comparison to their competitors. For example, on the Hill_valley dataset where Random Forest and Completely Random Tree Forest yields the smallest classification accuracies among all methods, those accuracies are by far worse than that of DEFEF (0.544 of Completely Random Tree Forest and 0.5755 of Random Forest vs. 0.9821 of DEFEG). Although the usage of deep ensemble models gcForest can improve the performance (0.5824 of gcForest), the classification accuracy is still low.

WideDeep and DeepFM performed poorly in the experiments in which they ranked the seventh and nineth among all methods respectively. While WideDeep ranks first on Embryonal dataset and ranks second on 3 datasets (Musk1, Ring, and Twonorm), DeepFM does not rank first or second on any datasets. DEFEG outperformed WideDeep and DeepFM on 32 datasets. Some significant differences between the performances of DEFEG and WideDeep and DeepFM are on Breast-Tissue (0.75 of DEFEG vs 0.1563 and 0.125 of WideDeep and DeepFM, respectively), Cleveland (0.6111 of DEFEG vs.0.1 and 0.1333 of WideDeep and DeepFM, respectively), and Conn-bench-vowel (0.9182 of DEFEG vs. 0.1258 and 0.1698 of WideDeep and DeepFM, respectively). By contrast, WideDeep is better than DEFEG on 2 datasets only (0.6667 vs. 0.6111 on Embryonal dataset, and 0.9779 vs. 0.977 on Twonorm dataset) while DeepFM does not win DEFEG on any datasets.

For F1 score, it is observed a similar pattern in Nemenyi test result like the comparison on classification accuracy mentioned above. In detail, DEFEF is better than all benchmark algorithms except Rotation Forest based on the Nemenyi test (see Fig. 10). DEFEG ranks first with rank value 1.775 and ranks first or shares rank first on up to 22 datasets. WideDeep, DeepFM, and Completely Random Tree Forest continue to be three poorest methods in our experiments. There are two differences in Figs. 9 and 10 (i) for F1 score, XgBoost is better than not only Completely Random Tree Forest, WideDeep, and DeepFM like the Nemenyi test result concerning accuracy but also Random Forest and (ii) while the ranks of 7 methods (DEFEF, Rotation Forest, XgBoost, MLP, gcForest, Random Forest, and WideDeep) are similar for

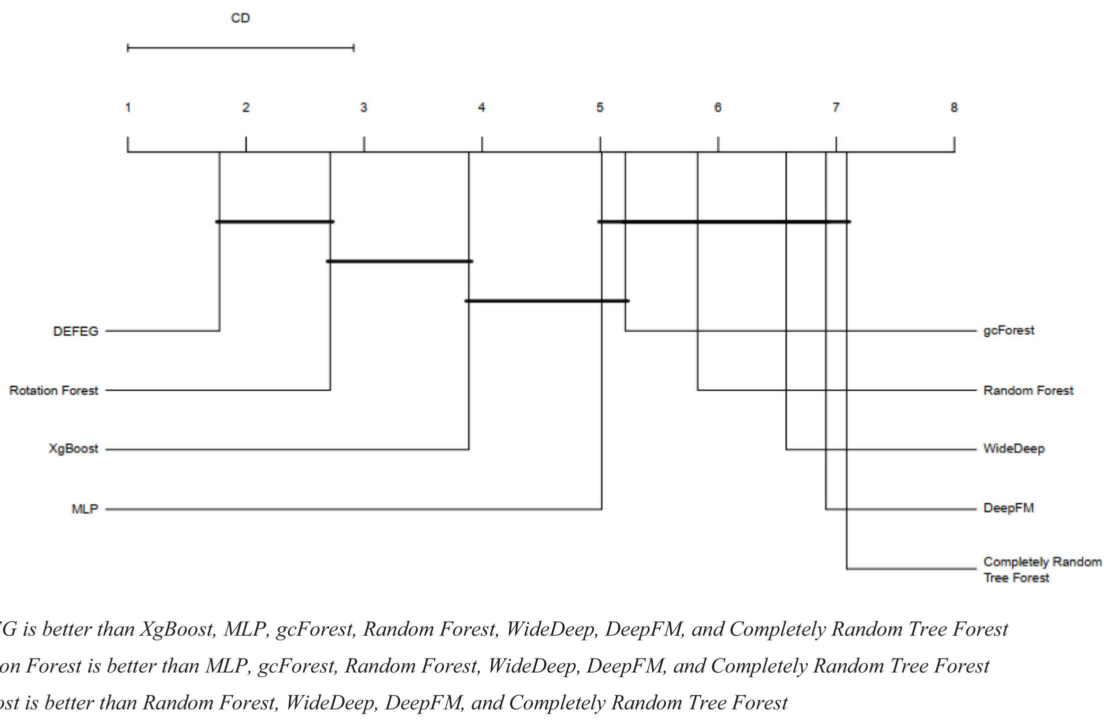


Fig. 10. The Nemenyi test result on F1 score. DEFEG is better than XgBoost, MLP, gcForest, Random Forest, WideDeep, DeepFM, and Completely Random Tree Forest Rotation Forest is better than MLP, gcForest, Random Forest, WideDeep, DeepFM, and Completely Random Tree Forest XgBoost is better than Random Forest, WideDeep, DeepFM, and Completely Random Tree Forest.

both accuracy and F1 score, the Completely Random Tree Forest is the poorest methods when F1 score was considered. The average ranking and the ranking on each dataset once again show the advantage of DEFEG.

We note some differences in the experimental results concerning F1 score in comparison to the experimental results concerning classification accuracy. On 3 datasets, although the ranking of DEFEG concerning classification accuracy is very good, the ranking of DEFEG concerning F1 score is at the middle. In detail, on Abalone, Breast-Tissue, and Wine_red dataset, the rankings of DEFEG concerning classification accuracy are 2nd, 1st, and 1.5th, respectively while the rankings of DEFEG concerning F1 score are up to 4th, 5th, and 3rd, respectively. In contrast, on Cleveland dataset, DEFEG ranks the first concerning F1 score but ranks 4th concerning classification accuracy. Since we optimized the classification accuracy only on the validation set to find the integrated weights, the comparison between DEFEG and the benchmark algorithms concerning the classification accuracy shows a greater number of better results than those concerning the F1 score. We address future work to optimize both these two performance metrics for deep ensemble systems.

DEFEG continues be significantly better than the second rank method on several datasets such as Cleveland (0.3581 vs. 0.3016 of XgBoost), Electricity (0.9285 vs. 0.8925 of Rotation Forest), Hayes-roth (0.8944 vs. 0.8593 of Rotation Forest), and Svmguide2 (0.7371 vs. 0.6825 of XgBoost).

To sum up, DEFEG is better than the benchmark algorithms for both performance metrics on the experiments on 40 datasets. The outstanding of DEFEG over the benchmark algorithms may come from (i) the advantage of multi-layer architecture over one-layer ensembles like Random Forest and Completely Random Forest (ii) the advantage of integrated weights on the predictions in feature generation over simple concatenation in gcForest.

The number of layers and computational complexity: Fig. 11 shows the comparison between the number of layers generated by gcForest and DEFEG. On average, gcForest generates more layers than DEFEG, 3.5 vs. 2.25. GcForest can determine the number of layers based on the performance on the validation set, i.e. it stops growing the new layer if the classification error on the validation set does not improve after a specific number of layers. In DEFEG, the maximum number of layers was set to 5, and in the length changing procedure, we shorten the number of layers by 1 if the global best does not improve after a specific number of iterations. Thus, the average number of layers in gcForest is more than that of DEFEG in our experiments. The classification time depends on the number of layers as the more layer, the higher the computational complexity during classification.

It is noted that DEFEG takes much higher training time to search for the optimal weights for generating the augmented features on the predictions of classifiers. However, the training time of DEFEG can be reduced based on the stopping criterion in VLPSO. In Fig. 12, the fitness function's average and global best in VLPSO iterations show that DEFEG's global best converges quickly on datasets such as Musk2, Conn-bench-vowel, Bupa, Hayes-roth, and Penbased, while on datasets like Spambase, Vehicle, Waveform_w_noise, and Electricity, global best convergence happens after around 20 iterations. On some datasets, such as Artificial, Waveform_wo_noise, and Madelon, global best convergence is slower. Thus, early stopping based on the convergence of the global best can effectively reduce training time on certain

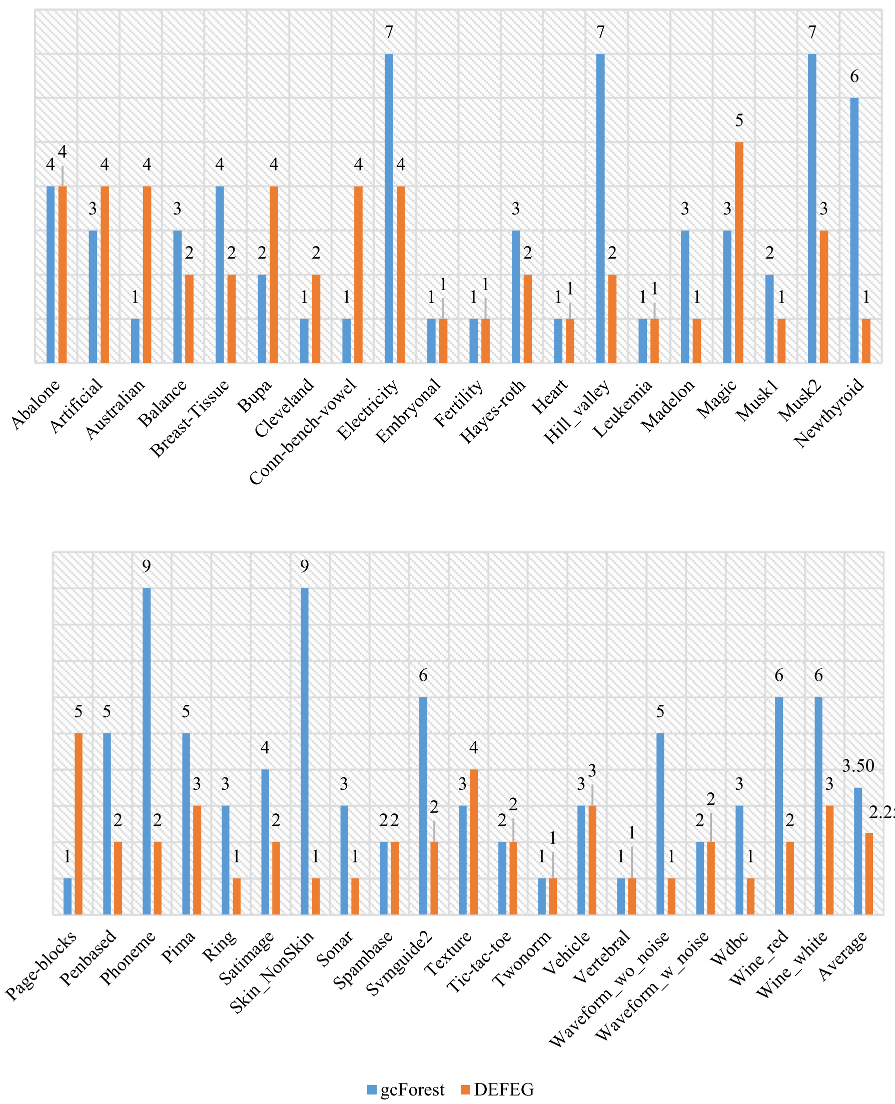


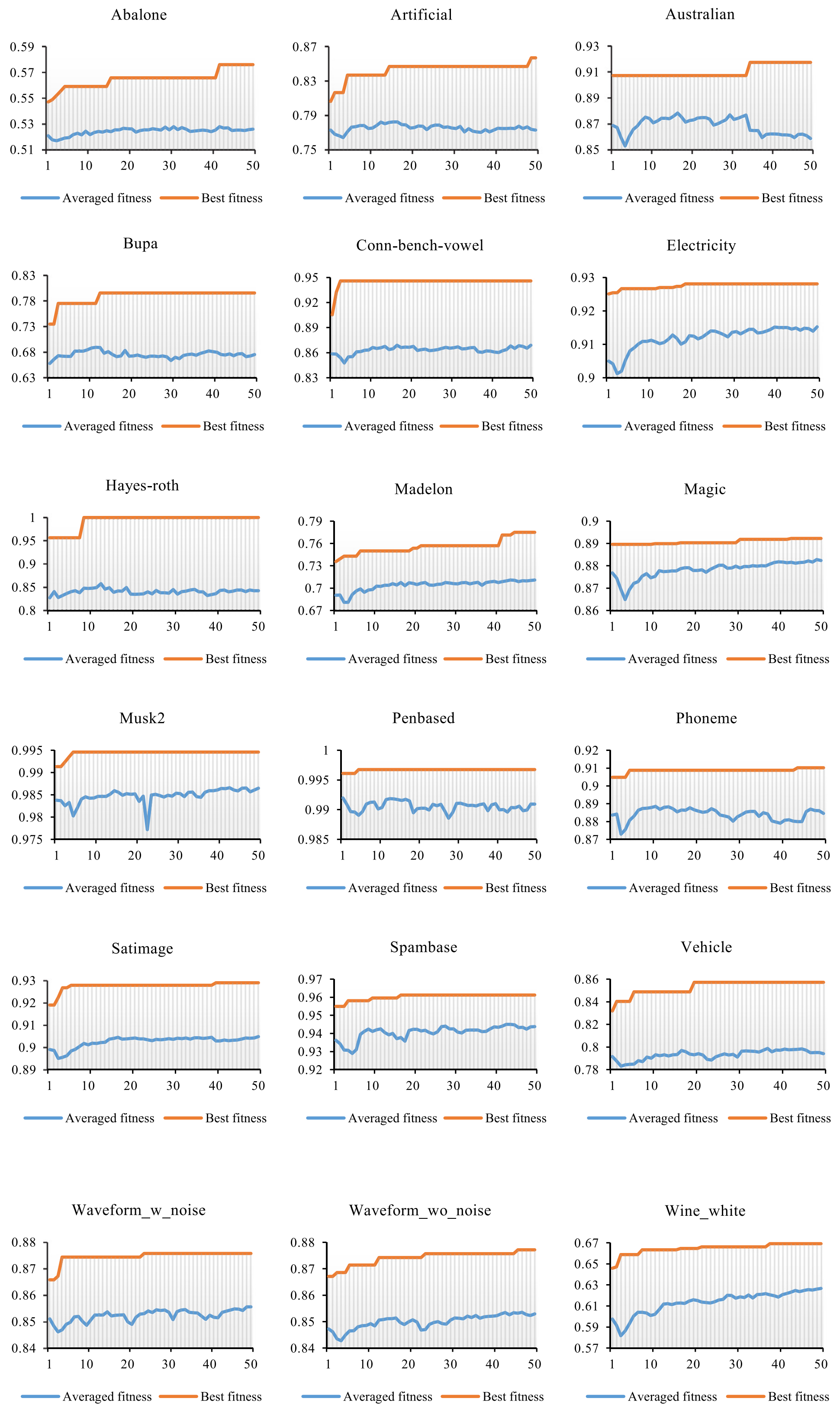
Fig. 11. The number of layers of gcForest and DEFEG on experimental datasets.

datasets. Theoretically, the training time complexity of DEFEG is $s \times \mathcal{O}(VLPSO) \times \mathcal{O}(Particle\ Evaluation) = s \times (nIters \times nPop) \times (K \times t_1)$, where *nIters* is the number of iterations in VLPSO, *nPop* is the population size of VLPSO, s is the average number of layers of VLPSO's configurations, K is the number of classifiers at each layer, and t_1 is the average training time complexity of base classifiers. In comparison, the training time of gcForest is faster by $nIters \times nPop$ times.

However, DEFEG's classification time is competitive with that of gcForest. The theoretical time complexities of DEFEG and gcForest are $\mathcal{O}(s \times K \times t_2)$, where s is the number of layers, K is the number of classifiers at each layer, and t_2 is the average testing time complexity of base classifiers. On the datasets where the two methods have the same number of layers, their time complexities for the classification phase are nearly the same. Experimental results also confirm the similar classification times of the two methods, for example, 0.5 vs. 0.38 s on Abalone (4 layers), 0.28 vs. 0.23 s on Spambase (2 layers), 0.07 vs. 0.09 s on Tic-tac-toe (2 layers), and 0.12 vs. 0.14 s on Vehicle (3 layers), for DEFEG and gcForest, respectively.

5. Conclusions

In this study, we proposed DEFEG, a deep ensemble model with multiple layers of different classifiers in each layer for the classification problem. The main difference between DEFEG and gcForest lies in the feature generation module. While gcForest concatenates the original features with the outputs of classifiers to generate the input features for the subsequent layer, DEFEG integrates weights on the classifiers' outputs as augmented features to grow the deep model, which can be viewed as some kind of boosting. This approach allows for the adjustment of the input data of each layer, leading to better results for the deep model. In contrast, the input features in gcForest do not necessarily ensure better performance when going through layer-by-layer architecture. In addition, gcForest uses a homogeneous ensemble in its layers, while DEFEG employs a heterogeneous ensemble at each layer, allowing for greater diversity in the predictions of each layer. Another key innovation in DEFEG is the use of variable-length encoding to encode the weights for all layers. This encoding method enables the algorithm to handle different



** x-axis is the number of iterations, y-axis is the value of fitness function as the classification accuracy on the validation set*

Fig. 12. The average and global best of the fitness function in the iterations of VLPSO. * *x*-axis is the number of iterations, *y*-axis is the value of fitness function as the classification accuracy on the validation set.

numbers of classifiers in each layer as well as different number of layers, providing more flexibility in the model architecture. We also developed a new variable-length Particle Swarm Optimization (VLPSO) algorithm to search for the optimal values of the weights by maximizing the classification accuracy on the validation data.

DEFEG is better than the benchmark algorithms including gcForest, XgBoost, Random Forest, Completely Random Tree Forest, MLP, and Rotation Forest in our experiments. In detail, DEFEG outperforms the XgBoost, MLP, gcForest, Random Forest, and Completely Random Tree Forest based on the Nemenyi post-hoc test for both classification accuracy and F1 score. DEFEG ranks first on average and ranks first or share the first ranking on more than 20 datasets. These results demonstrate the advantage of the proposed augmented features to grow the deep model in our study comparing to the gcForest as well as the advantage of the multi-layer architecture of DEFEG comparing to the one-layer ensemble like Random Forest and Completely Random Tree Forest.

In future work, a plausible approach to improve DEFEG is to speed up its training time. In particular, the problem of expensive training time of the VLPSO optimization during the objective function evaluation can be relieved using a surrogate-based method [42]. A surrogate on the objective function can be trained with some beginning populations, and then it is used to predict values of the objective function of each particle in the subsequent populations.

For reproducibility, we have made the code available on [to be provided upon publication of this work].

CRediT authorship contribution statement

Anh Vu Luong: Conceptualization, Methodology, Software, Writing – original draft. **Tien Thanh Nguyen:** Methodology, Writing – original draft. **Kate Han:** Software. **Trung Hieu Vu:** Software. **John McCall:** Writing – review & editing. **Alan Wee-Chung Liew:** Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Publicly available dataset was used in this research. Code will be made available after acceptance.

References

[1] C.-X. Zhang, J.-S. Zhang, RotBoost: A technique for combining rotation forest and AdaBoost, Pattern Recognit. Lett. 29 (10) (2008) 1524–1536.
[2] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: Proceedings of CVPR, 2001.
[3] T.T. Nguyen, A.W.C. Liew, M.T. Tran, T.T.T. Nguyen, M.P. Nguyen, Fusion of classifiers based on a novel 2-stage model, in: International Conference on Machine Learning and Cybernetics, 2014, pp. 60–68.
[4] Z.-H. Zhou, J. Feng, Deep forest: Towards an alternative to deep neural networks, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), AAAI Press, Melbourne, Australia, 2017, pp. 3553–3559.
[5] S. Kim, Z. Yu, R.M. Kil, M. Lee, Deep learning of support vector machines with class probability output networks, Neural Netw. 64 (2015) 19–28.
[6] L.V. Utkin, M.S. Kovalev, A.A. Meldo, A deep forest classifier with weights of class probability distribution subsets, Knowl. Based Syst. 173 (2019) 15–27.
[7] Z. Qi, B. Wang, Y. Tian, P. Zhang, When ensemble learning meets deep learning: A new deep support vector machine for classification, Knowl. Based Syst. 107 (2016) 54–60.
[8] B. Chen, H. Wu, W. Mo, I. Chattopadhyay, H. Lipson, Autostacker: A compositional evolutionary system, in: Proceeding of GECCO, 2018.
[9] M.F. Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 9 (2014) 3133–3181.
[10] T.T. Nguyen, M.T. Dang, A.W.-C. Liew, J.C. Bezdek, A weighted multiple classifier framework based on random projection, Inform. Sci. 490 (2019) 36–58.
[11] T.T. Nguyen, M.P. Nguyen, X.C. Pham, A.W.C. Liew, W. Pedrycz, Combining heterogeneous classifiers via granular prototypes, Appl. Soft Comput. 73 (2018) 795–815.
[12] T.T. Nguyen, A.V. Luong, M.T. Dang, A.W.-C. Liew, J. McCall, Ensemble selection based on classifier prediction confidence, Pattern Recognit. 100 (2020).
[13] L. Breiman, Random forest, Mach. Learn. 45 (2001) 5–32.
[14] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[15] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: A new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (10) (2006) 1619–1630.
[16] T.T. Nguyen, N. Van Pham, M.T. Dang, A.V. Luong, J. McCall, A.W.C. Liew, Multi-layer heterogeneous ensemble with classifier and feature selection, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 725–733.
[17] T. Dang, T.T. Nguyen, J. McCall, E. Elyan, C.F. Moreno-García, Two layer ensemble of deep learning models for medical image segmentation, 2021, arXiv preprint arXiv:2104.04809.
[18] T. Dang, A.V. Luong, A.W.C. Liew, J. McCall, T.T. Nguyen, Ensemble of deep learning models with surrogate-based optimization for medical image segmentation, in: 2022 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2022, pp. 1–8.
[19] A.V. Luong, T.T. Nguyen, A.W.C. Liew, Streaming multi-layer ensemble selection using dynamic genetic algorithm, in: 2021 Digital Image Computing: Techniques and Applications, DICTA, IEEE, 2021, pp. 1–8.
[20] Q. Wei, Q. Zhang, H. Gao, T. Song, A. Salhi, B. Yu, Deepstack-RBP: Accurate identification of RNA-binding proteins based on autoencoder feature selection and deep stacking ensemble classifier, Knowl.-Based Syst. 256 (2022) 109875.
[21] Y.-W. Kim, I.-S. Oh, Classifier ensemble selection using hybrid genetic algorithms, Pattern Recognit. Lett. 29 (2008) 796–802.
[22] Y. Wang, D. Wang, N. Geng, Y. Wang, Y. Yin, Y. Jin, Stacking-based ensemble learning of decision trees for interpretable prostate cancer detection, Appl. Soft Comput. 7 (2019) 188–204.
[23] I. Mendialdua, A. Arruti, E. Jauregi, E. Lazkano, B. Sierra, Classifier subset selection to construct multi-classifiers by means of estimation of distribution algorithms, Neurocomputing 157 (2015) 46–60.
[24] R. Mousavi, M. Eftekhari, A new ensemble learning methodology based on hybridization of classifier ensemble selection approach, Appl. Soft Comput. 37 (2015) 652–666.
[25] M.N. Haque, N. Noman, R. Berretta, P. Moscato, Heterogeneous ensemble combination search using genetic algorithm for class imbalanced data classification, PLOS1 (2016) http://dx.doi.org/10.1371/journal.pone.0146116.
[26] T.T. Nguyen, A.V. Luong, T.M.V. Nguyen, T.S. Ha, A.W.C. Liew, J. McCall, Simultaneous meta-feature and meta-classifier selection in multiple classifier system, in: GECCO, 2019.
[27] X. Wang, H. Wang, Classification by evolutionary ensembles, Pattern Recognit. 39 (2006) 595–607.
[28] S. Ali, A. Majid, Can-evo-ens: Classifier stacking based evolutionary ensemble system for prediction of human breast cancer using amino acid sequences, J. Biomed. Inform. 54 (2015) 256–269.
[29] T.T. Nguyen, M.D. Dang, V.A. Baghel, A.V. Luong, J. McCall, A.W.C. Liew, Evolving interval-based representation for heterogeneous classifier fusion, Knowl. Based Syst. (2020).
[30] T.T. Nguyen, A.V. Luong, M.T. Dang, L.P. Dao, T.T.T. Nguyen, A.W.C. Liew, J. McCall, Evolving an optimal decision template for combining classifiers, in: Proceeding of ICONIP, 2019, pp. 608–620.
[31] C.A.A. Padilha, D.A.C. Barone, A.D.D. Neto, A multi-level approach using genetic algorithms in an ensemble of least squares support vector machines, Knowl. Based Syst. 106 (2016) 85–95.
[32] K.-J. Kim, S.-B. Cho, An evolutionary algorithm approach to optimal ensemble classifiers for DNA microarray data analysis, IEEE Trans. Evol. Comput. 12 (3) (2008) 377–388.
[33] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Vol. IV, 1995, pp. 1942–1948, http://dx.doi.org/10.1109/ICNN.1995.488968.

[34] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Math. Probl. Eng. (2015).

[35] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.

[36] B. Tran, B. Xue, M. Zhang, Variable-length particle swarm optimization for feature selection on high-dimensional classification, IEEE Trans. Evol. Comput. 23 (3) (2019) 473–487.

[37] X. Yu, X. Zhang, Enhanced comprehensive learning particle swarm optimization, Appl. Math. Comput. 242 (2014) 265–276.

[38] N. Lynn, P.N. Suganthan, Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation, Swarm Evol. Comput. 24 (2015) 11–24.

[39] B. Wang, Y. Sun, B. Xue, M. Zhang, A hybrid GA-PSO method for evolving architecture and short connections of deep convolutional neural networks, https://arxiv.org/abs/1903.03893.

[40] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, DeepFM: a factorization-machine based neural network for CTR prediction, 2017, arXiv preprint arXiv:1703.04247.

[41] H.T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye ..., H. Shah, Wide & deep learning for recommender systems, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, 2016, pp. 7–10.

[42] T. Dang, A.V. Luong, A.W.-C. Liew, J. McCall, T.T. Nguyen, Ensemble of deep learning models with surrogate-based optimization for medical image segmentation, in: Proceeding of IEEE Congress on Evolutionary Computation, CEC, 2022.