VEGAS: A Variable Length-Based Genetic Algorithm for Ensemble Selection in Deep Ensemble Learning

Kate Han[1], Tien Pham[2], Trung Hieu Vu[3], Truong Dang[1], John McCall[1], and Tien Thanh Nguyen[1(✉)]

[1] School of Computing, Robert Gordon University, Aberdeen, UK
t.nguyen11@rgu.ac.uk
[2] College of Science and Engineering, Flinders University, Adelaide, Australia
[3] School of Electronics and Telecommunications, Hanoi University of Science and Technology, Hanoi, Vietnam

Abstract. In this study, we introduce an ensemble selection method for deep ensemble systems called VEGAS. The deep ensemble models include multiple layers of the ensemble of classifiers (EoC). At each layer, we train the EoC and generates training data for the next layer by concatenating the predictions for training observations and the original training data. The predictions of the classifiers in the last layer are combined by a combining method to obtain the final collaborated prediction. We further improve the prediction accuracy of a deep ensemble model by searching for its optimal configuration, i.e., the optimal set of classifiers in each layer. The optimal configuration is obtained using the Variable-Length Genetic Algorithm (VLGA) to maximize the prediction accuracy of the deep ensemble model on the validation set. We developed three operators of VLGA: roulette wheel selection for breeding, a chunk-based crossover based on the number of classifiers to generate new offsprings, and multiple random points-based mutation on each offspring. The experiments on 20 datasets show that VEGAS outperforms selected benchmark algorithms, including two well-known ensemble methods (Random Forest and XgBoost) and three deep learning methods (Multiple Layer Perceptron, gcForest, and MULES).

Keywords: Deep learning · Ensemble learning · Ensemble selection · Classifier selection · Ensemble of classifiers · Genetic algorithm.

1 Introduction

In recent years, deep learning has emerged as a hot research topic because of its breakthrough performance in diverse learning tasks. For instance, in computer vision, Convolutional Neural Network (CNN), a deep neural network (DNN), significantly outperforms traditional machine learning algorithms on the image classification task on some large scale datasets. Despite its many successes, there are

N. T. Nguyen et al. (Eds.): ACIIDS 2021, LNAI 12672, pp. 168–180, 2021.
https://doi.org/10.1007/978-3-030-73280-6_14

some limitations of DNNs. Firstly, these deep models are usually very complex with many parameters and can only be trained on specially-designed hardware. Secondly, DNNs require a considerable amount of labeled data for the training process. When the cost of labeled data is too prohibitive, deep models might not bring about the expected gains in performance. It is well-recognized that there are many learning tasks where DNNs are poorer than traditional machine learning methods, especially state-of-the-art methods like Random Forest and XgBoost [12].

Meanwhile, ensemble learning is developed to obtain a better result than using single classifiers. By using an ensemble of classifiers (EoC), the poor predictions of several classifiers are likely to be compensated by those of others, which boosts the performance of the whole ensemble. Ensemble systems have been applied in many areas such as computer vision, software engineering, and bioinformatics. Traditionally, ensemble systems have been constructed with one layer of EoC. A combining algorithm combines the predictions of EoC for the final collaborated prediction [10].

The term *"deep learning"* makes the crowd only think of DNNs, which include multiple layers of parameterized differentiable nonlinear modules. In 2014, Zhou and Feng introduced a deep ensemble system called gcForest, including several layers of Random Forest-based classifiers [22]. The introduction of gcForest has shown that DNN is only a subset of deep models or deep learning models can be constructed with multiple layers of non-differentiable learning modules. Experiments on some popular datasets have shown that deep ensemble models outperform not only DNNs but also several state-of-the-art ensemble methods [15,22].

The *predictive performance* and *computational/storage efficiency* of ensemble systems can be further improved by obtaining a subset of classifiers that performs competitively to the whole ensemble. This research topic is known as ensemble selection (aka ensemble pruning or selective ensemble), an ensemble design stage to enhance ensemble performance based on searching for an optimal EoC from the whole ensemble. In this study, we introduce an ensemble selection method to improve the performance and efficiency of deep ensemble models. A configuration of the deep model is encoded in the form of binary encoding, showing which classifiers are selected or not. It is noted that the length of the proposed encoding depends on the number of layers and the number of classifiers in each layer that we use to construct the deep model. The configuration of the deep model thus is given in variable-length encoding. To find the optimal set of classifiers, we consider an optimization problem by maximizing the classification accuracy of the deep ensemble system on the validation data. In this work, we develop VEGAS: a Variable-length Genetic Algorithm (VLGA) to solve this optimization problem of the ensemble selection. The main contributions of our work are as follows:

- We propose a classifier selection approach for deep ensemble system
- We propose to encode classifiers in all layers of the deep ensemble system in a variable-length encoding

– We develop a VLGA to search for the optimal number of layers and the optimal classifiers.
– We experimentally show that VEGAS is better than some well-known benchmark algorithms on several datasets.

2 Background and Related Work

2.1 Ensemble Learning and Ensemble Selection

Ensemble learning refers to a popular research topic in machine learning in which multiple classifiers are combined to obtain a better result than using single classifiers. Two main stages in building an ensemble system are somehow to generate diverse classifiers and then combine them to make a collaborated decision. For the first stage, we train a learning algorithm on multiple training sets generated from the original training data [11] or train different learning algorithms on the original training data [10] to generate EoC. For the second stage, a combining method works on the predictions of the generated classifiers for the final decision. Based on experiments on diverse datasets, Random Forest [1] and XgBoost [3] were reported as the top-performance ensemble methods.

Inspired by the success of DNNs in many areas, several ensemble systems have been constructed with a number of layers of EoCs. Each layer receives outputs of the subsequent layer as its input training data and then outputs the training data for the next layer. The first deep ensemble system called gcForest was proposed in 2014, including many layers of two Random Forests and two Completely Random Tree Forests working in each layer. After that, several deep ensemble systems were introduced such as deep ensemble models of incremental classifiers [7], an ensemble of SVM classifiers with AdaBoost in finding model parameters [17], and deep ensemble models for multi-label learning [21]. Nguyen et al. [15] proposed MULES, a deep ensemble system with classifier and feature selection in each layer. The optimization problem was considered under bi-objectives: maximizing classification accuracy and diversity of EoC in each layer.

Meanwhile, ensemble selection is an additional intermediate stage of the ensemble design process that aims to select a subset of classifiers from the ensemble to achieve higher *predictive performance* and *computational/storage efficiency* than using the whole ensemble. Ensemble selection can be formulated as an optimization problem that can be solved by either Evolutionary Computation methods or greedy search approach. Nguyen et al. [13] used Ant Colony Optimisation (ACO) to search for the optimal combining algorithm and the optimal set from the predictions for the selected classifier in the ensemble system. Hill climbing search-based ensemble pruning, on the other hand, greedily selects the next configuration of selected classifiers around the neighborhood of the current configuration. Two crucial factors of the hill-climbing search-based ensemble pruning strategy are the searching direction and the measure to evaluate the different branches of the search process [16]. For an EoC, the measures determine the best single classifier to be added to or removed from the ensemble

to optimize either performance or diversity of the ensemble. Examples of evaluation measures are *Accuracy, Mean Cross-Entropy, F-score, ROC area* [16], and *Concurrency* [2]. The direction of the search process can be conducted in the forward selection which starts from the empty EoC and adds a base classifier in sequence, or backward selection which prunes a base classifier from the whole set of classifiers until reaching the optimal subset of classifiers. Dai et al. [4] introduced the Modified Backtracking Ensemble Pruning algorithm to enhance the search processing in the backtracking method. The redundant solutions in the search space are reduced by using a binary encoding for the classifiers.

2.2 Variable Learning Encoding in Evolutionary Computation

There are some variable length encoding-based algorithms introduced recently. In [19], a VLGA is proposed to search for the best CNN structure. Each chromosome consists of three types of units corresponding to convolutional layers, pooling layers, and full connection layers. Later in [20], a non-binary VLGA was also proposed to search for the best CNN structure. This variable-length encoding strategy used different representations for different layer types. A skipping layer consists of two convolutional layers and one skipper connection; its encoding is the number of feature maps of the two convolutional layers within this skip layer. The encoding of the pooling layers is the pooling operation type, i.e. mean pooling or maximum pooling.

For applications, in [18], the GA with a variable-length chromosome was used to solve the path optimization problems. The path optimization problem is modeled as an abstract graph. Each chromosome is a set of nodes consisting of a feasible solution and therefore has a length equal to node amount. In [6], a GA with variable length chromosomes was also used to solve path planning problems for the autonomous mobile robot. Each chromosome is a position set that represents a valid path solution. The length of the chromosome is the number of the intermediate nodes. In [9], the proposed VLGA was also implemented to solve the multi-objective multi-robot path planning problems.

3 Proposed Method

3.1 Ensemble Selection for Deep Ensemble Systems

Let $\mathcal{D}$ be the training data of N observations $\{(\mathrm{x}_n, \hat{y_n}\})$, where x_n is the D-feature vector of the training instance and y_n be its corresponding label. True label $\hat{y_n}$ belongs to label set $\mathcal{Y}, |\mathcal{Y}| = M$. We aim to learn a hypothesis $\mathbf{h}$ (i.e., classifier) to approximate unknown relationship between the feature vector and its corresponding label $g : \mathrm{x}_n \rightarrow \hat{y_n}$ and then use this hypothesis to assign a label for each unlabeled instance. We also denote $\mathcal{K} = \mathcal{K}_k$ as the set of K learning algorithms. In deep ensemble learning consisting of s layers, we train an EoC $\{\mathbf{h}_k^{(i)}\}$ on i^{th} layer ($i = 1, \ldots, s$ and $k = 1, \ldots, K$) and then use a combining

algorithm C on the hypothesis of the s^{th} layer $\tilde{\mathbf{h}} = C\left\{\mathbf{h}_k^{(s)}, k = 1, \ldots, K\right\}$ for final decision making.

A deep ensemble system has multiple layers where each of them consists of EoCs. In the first layer, we obtain EoC $\left\{\mathbf{h}_k^{(1)}, k = 1, \ldots, K\right\}$ by training K learning algorithms on the original training data $\mathcal{D}$. The first layer also generates input data for the second layer by using the Stacking algorithm with the set of learning algorithms $\mathcal{K}$. Specifically, $\mathcal{D}$ is divided into T_1 disjoint parts in which the cardinality of each part is nearly similar. For each part, we train classifiers on its complementary and use these classifiers to predict for observations of this part. Thus, each observation in $\mathcal{D}$ will be tested one time. For observation x_n, we denote $p_{k,m}^{(1)}(\mathrm{x}_n)$ is the prediction of the k^{th} classifier in the first layer that observation belongs to the class label y_m. The predictions in terms of M class labels are given in the form of probability: $\sum_{m=1}^{M} p_{k,m}^{(1)}(\mathrm{x}_n = 1; k = 1, \ldots, K$ and $n = 1, \ldots, N$. We denote $P^{(1)}(\mathrm{x}_n) = \left[p_{1,1}^{(1)}(\mathrm{x}_n), p_{1,2}^{(1)}(\mathrm{x}_n), \ldots, p_{K,M}^{(1)}(\mathrm{x}_n)\right]$ is a (MK) prediction vector of the EoC in the first layer for x_n. The prediction vectors for all observations in $\mathcal{D}$ is given in the form of a $N \times (MK)$ matrix.

$$\mathcal{P}_1 = \left[P^{(1)}(\mathrm{x}_1)P^{(1)}(\mathrm{x}_2)\ldots P^{(1)}(\mathrm{x}_N)\right]^T \tag{1}$$

We denote $\mathcal{L}_1$ denotes the new data generated by the 1^{st} layer as the input for the 2^{nd} layer. Normally, $\mathcal{L}_1$ is created by concatenating the original training data and the predictions classifiers as below:

$$\mathcal{L}_1 = \mathcal{D} \bigoplus \mathcal{P}_1 \tag{2}$$

in which $\bigoplus$ denotes the concatenation operator between two matrices $\mathcal{D}$ of size $N \times D + 1$ and $\mathcal{P}_1$ of size $N \times (MK)$. Thus $\mathcal{L}_1$ is obtained in the form of a $N \times (D + MK + 1)$ matrix including D features of original data, MK features of predictions, and ground truth of observations. A similar process conducts on the next layers until reaching the last layer in which at the i^{th} layer, we train the EoC of K classifiers $\mathbf{h}_k^{(i)}, k = 1, \ldots, K$ on the input data $\mathcal{L}_{i-1}$ generated by $(i-1)^{th}$ layer and generate input data $\mathcal{L}_i$ for the $(i+1)^{th}$ layer

$$\mathcal{L}_i = \mathcal{D} \bigoplus \mathcal{P}_i \tag{3}$$

The predictions of EoC of the last layer i.e. s^{th} layer are combined for the collaborated decision. In this study, we use the Sum rule for combining [14]. For an instance x, the Sum rule summarizes the predictions of EoC of the last layer concerning each class label. The label associated with the maximum value is assigned to this instance as follows:

$$\tilde{\mathbf{h}} : \mathrm{x} \in y_t \text{ if } t = argmax_{m=1,\ldots,M} \left\{\sum_{k=1}^{K} p_{k,m}^{(s)}(\mathrm{x})\right\} \tag{4}$$

In the classification process, each unseen instance is fed forward through the layers until reaching the last layer. The predictions of K classifier at the last

layer i.e. $P^{(s)}(.) = \left[p_{1,1}^{(s)}(.), p_{1,2}^{(s)}(.), \ldots p_{K,M}^{(s)}(.)\right]$ are combined by Sum Rule in (4) to obtain the predicted label.

It is recognized that there is existing a subset of EoC that performs better than using the whole ensemble. Moreover, storing a subset of the ensemble will save the computational cost and storage cost. In this study, we propose an ensemble selection approach for deep ensemble systems. We propose to encode classifiers in the deep ensemble system using binary encoding in (5), showing which classifiers are presented or absent. For a deep ensemble system of s layers, since there are K classifiers in each layer, the encoding associated with the model of s layers has $s \times K$ binary elements. It is noted that the length of the proposed encoding is not fixed and depends on the number of layers that we use to construct the deep model. If the number of layers is chosen by $1 \leq s \leq S$, we have S groups of encoding with the lengths of $\{K, 2 \times K, \ldots, S \times K\}$. By using these groups of encoding, we aim to search for the optimal number of layers and the optimal set of classifiers in each layer for the deep ensemble system.

$$\mathbf{h}_k^{(i)} = \begin{cases} 1, & k^{th} \text{ classifier at } i^{th} \text{ layer is selected} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

3.2 Optimization Problems and Algorithm

We consider optimization for the model selection problem. The objective is maximizing the accuracy of the classification task on a validation set $\mathcal{V}$:

$$max_E \left\{ \frac{1}{|\mathcal{V}|} \sum_{n=1}^{|\mathcal{V}|} \left\| \tilde{\mathbf{h}}_E(\mathrm{x}_n) = \hat{y}_n \right\| \right\} \tag{6}$$

where $\tilde{\mathbf{h}}_E$ is the combining model using the Sum Rule in (4) associated with encoding E, $|.|$ denotes the cardinality of a set, and$\|.\|$ is equal 1 if the condition is true, otherwise equal 0. In this study, we develop a VLGA to solve this optimization problem. Genetic Algorithm (GA) is a search heuristic inspired by Charles Darwin's theory of natural evolution. It is widely recognized that GA commonly generates high-quality solutions for search problems [8]. Three operators of GA are considered in this study:

Selection: We apply the roulette wheel selection approach to select a pair of individuals for breeding. The probability of choosing an individual from a population is proportional to its fitness as an individual has a higher chance of being chosen if its fitness is higher than those of others. Probability of choosing individual i^{th} is equal to:

$$\mathrm{p}_i = \frac{f_i}{\sum_{j=1}^{popSize} f_j} \tag{7}$$

where f_i is the accuracy of the deep ensemble model with the corresponding configuration of the i^{th} individual and $popSize$ is the size of the current generation.

Crossover: We define the probability P_c for the crossover process in which crossover occurs if the generated crossover probability is smaller than P_c. Here

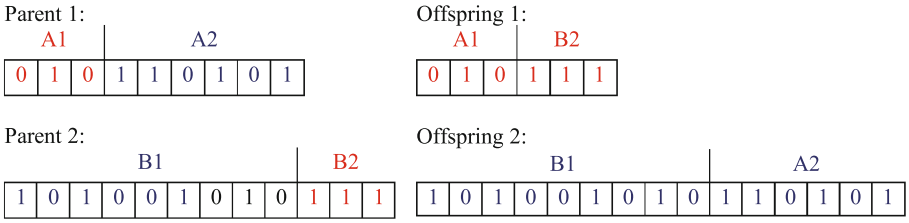

Fig. 1. The illustration of chunk-based crossover operator

we develop a chunk-based crossover operator to generate new offsprings. As mentioned before, since there are at most K classifiers in each layer of a deep ensemble system with s layers, each chromosome is given in the form of s-chunk in which the chunk size is K. On two selected parents with s_1 and s_2 layers, we generate two random numbers r_1 and r_2 which are the multiple of s_1 and s_2 i.e. $r_1 \in \{K, 2 \times K, \ldots, s_1 \times K\}$ and $r_2 \in \{K, 2 \times K, \ldots, s_2 \times K\}$. r_1 and r_2 will divide each parent into two parts. Each parent exchanges its tail with the other while retains its head. After crossover is performed, we have two new offspring chromosomes. We illustrate in Fig 1 how chunk-based crossover works on a deep ensemble model with 3 classifiers in each layer. Parent 1 encodes a 3-layer deep ensemble model, while parent 2 encodes a 4-layer deep ensemble model. On parent 1 and 2, two random numbers are generated as $r_1 = 3$ and $r_1 = 9$. By retaining heads and exchanging tails on these parents, we obtain two new offsprings, the first one encodes a 2-layer deep ensemble, and the second encodes a 5-layer deep ensemble. By using this crossover operator, we can generate the offsprings with different sizes compared to those of their parents, thus improving exploration of the searching process.

Mutation: Mutation operators introduce genetic diversity from one generation in a population to the next generation. It also prevents the algorithm from falling into local minima or maxima by making the population of chromosomes different from each other. We define the probability P_m for the mutation process in which mutation occurs if the generated mutation probability is smaller than P_m. In this study, we propose to apply a multiple point-based mutation operator on an offspring. First, we generate several random numbers which show the position of mutated genes in a chromosome. The values of these mutated genes will be flipped, i.e., from 0 to 1 or 1 to 0. By doing this way, we obtain a new offspring, which may change entirely from the previous one; consequently, GA can escape from local minima or maxima and reach a better solution.

The pseudo-code of VLGA is present in Algorithm 1. The algorithm gets the inputs including the training data $\mathcal{D}$, the validation data $\mathcal{V}$ and some parameters for the evolutionary process (the population size $popSize$, the number of generations $nGen$, crossover probability P_c and mutation probability P_m) We first randomly generate a population with $popSize$ individuals and then calculate the fitness of each individual on $\mathcal{V}$ by using Algorithm 2 (Step 1 and 2).

The probabilities for individual selection are computed by using (7) in Step 3. Two selected parents will bread a pair of offsprings if they satisfy the crossover check (Step 7–8). These offsprings will pass through mutation in which some random positions of them are changed if mutation occurs (Step 15–21). We also calculate the fitness of each offspring on $\mathcal{V}$ by using Algorithm 2 before adding them to the population. The step 6–23 are repeated until we generate a new $popSize$ offsprings. From the population of $2 \times popSize$ individuals, we keep $popSize$ best individuals for the next generation. The algorithm runs until it reaches the number of generations. We select the candidate from the last population, which is associated with the best fitness value as the solution of the problem.

Algorithm 2 aims to calculate the fitness and deep model generation associated with an encoding. The algorithm inputs training data $\mathcal{D}$, validation data $\mathcal{V}$, an encoding E, and the number of T-folds. From the configuration of E, we can obtain the number of layers and which classifiers are selected in each layer. On the i^{th} layer, we do two steps (i) train selected classifiers at the on whole $\mathcal{L}_{i-1}$denoted by $\{h_k^{(i)}\}$ (Step 4) and (ii) generate training data for the $(i+1)^{th}$ layer by using T-fold Cross-Validation and concatenation operator between prediction data and original training data (Step 7–14). The classifier $\{h_k^{(i)}\}$ predicts on $\mathcal{V}_{i-1}$ which is the prediction matrix for observations of $\mathcal{V}$ at the $(i-1)^{th}$ layer, to obtain the prediction $\mathcal{P}_i(\mathcal{V})$ (Step 6). $\mathcal{P}_i(\mathcal{V})$ is also concatenated with $\mathcal{V}$ to obtain the validation data for the $(i+1)^{th}$ layer. After running through the last layer i.e. the s^{th} layer, we apply the Sum Rule on the prediction $\mathcal{P}_s(V)$ to obtain the fitness value of E. We also obtain the classifiers $\{h_k^{(i)}\}$ associated with E.

In the classification process, we assign the class label to an unlabeled test sample. In each layer, the input test data will be predicted by classifiers and then be concatenated with the original test sample to generate new test data for the next layer. The combining function in (4) is applied to the outputs of classifiers of the last layer to give the final prediction.

4 Experimental Studies

4.1 Experimental Settings

We conducted the experiments on the 20 datasets collected from different sources such as the UCI Machine Learning Repository and OpenML. We used 5 classifiers in each layer of VEGAS in which these classifiers were generated by using learning algorithms: Naïve Bayes classifiers with Gaussian distribution, XgBoost with 200 estimators, Random Forest with 200 estimators, and Logistic Regression. We used the 5-fold Cross-Validation in each layer to generate the new training data for the next layer. 20% of the training data is used for validation purposes [22]. For VLGA, the maximum number of generations was set to 50, the population size was set to 100, and the crossover and mutation probability was set to 0.9 and 0.1, respectively.

Algorithm 1. Variable length Genetic Algorithm

Require: Training data $\mathcal{D}$, Validation data $\mathcal{V}$ population size: $popSize$, number of generations $nGen$, crossover probability: P_c, mutation probability: P_m
Ensure: Optimal configuration
Randomly generate population
Calculate fitness on $\mathcal{V}$ of each individual using Algorithm 2
Calculate selection probabilities by (7)
for $i \leftarrow 1, nGen$ **do**
  while $currentpopulationsize < 2 \times popSize$ **do**
    Select a pair of individuals based on selection probabilities
    Generate a random number $r_c \in [0, 1]$
    if $r_c \leq P_c$ **then**
      Generate two random number r_1, r_2 which are multiple of K
      Divide parents to head and tail based on r_1 and r_2
      Swap tails of two parents to create two new offsprings
    else
      Continue
    end if
    Generate a random number $r_m \in [0, 1]$
    if $r_m \leq P_m$ **then**
      for each offspring **do**
        Generate random number r of mutation points
        Flip the binary value associated with mutation points
      end for
    end if
    Calculate the fitnesses of two new offsprings using Algorithm 2
    Add two new offsprings to the population
  end while
  Keep $popSize$ best individuals for the next generation
  Calculate selection probabilities by (7)
end for
Return individual (encoding and associated deep model) with the best fitness from the last generation

VEGAS was compared to some algorithms, including the ensemble methods and deep learning models. Three well-known ensemble methods were used as the benchmark algorithms: Random Forest, XgBoost, and Rotation Forest. All these methods were constructed by using 200 learners. Three deep learning models were compared with VEGAS: gcForest (4 forests with 200 trees in each forest) [22], MULES [15], and Multiple Layer Perceptron (MLP). For MULES, we used parameter settings like in the original paper [15]. It is noted that the performance of MLP significantly depends on the network structure. To ensure a fair comparison, we experimented with MLP on a number of different network configurations: input-30-20-output, input-50-30-output, and input-70-50-output by referencing the experiments [22]. We then reported the best performance of MLP among all configurations and used this result to compare with VEGAS.

We used Friedman test to compare performance of experimental methods on experimental datasets. If the P-Value of this test is smaller than a significant threshold, e.g. 0.05, we reject the null hypothesis and conduct the Nemenyi post-hoc test to compare each pair of methods [5].

4.2 Comparison to the Benchmark Algorithms

Table 1 shows the prediction accuracy of VEGAS and the benchmark algorithms. Based on the Friedman test, we reject the null hypothesis that all methods per-

Algorithm 2. Fitness calculation and deep model generation on an encoding

Require: Training Data $\mathcal{D}$, validation data $\mathcal{V}$, and encoding E, number of T-fold
Ensure: the fitness value of E

Get number of layer s and selected classifiers in each layer from E
$\mathcal{L}_0 = \mathcal{D}, \mathcal{V}_0 = \mathcal{V}$
for $i \leftarrow 1, s$ **do**
  Train selected classifiers at the i^{th} layer on whole $\mathcal{L}_{i-1} : \{h_k^{(i)}\}$
  $\mathcal{P}_i = \emptyset$
  Use $\{h_k^{(i)}\}$ to predict for $\mathcal{V}_{i-1}$ to obtain $\mathcal{P}_i(\mathcal{V})$
  for $t \leftarrow 1, T$ %generate the running data for the next layer **do**
    $\mathcal{L}_{i-1} = \bigcup_{j=1}^{T} \mathcal{L}_{i-1}^{(j)}, \mathcal{L}_{i-1}^{(j_1)} \bigcap \mathcal{L}_{i-1}^{(j_2)} = \emptyset, \left|\mathcal{L}_{i-1}^{(j_1)}\right| \approx \left|\mathcal{L}_{i-1}^{(j_2)}\right|,$
    $1 \leq j_1, j_2 \leq T, j_1 \neq j_2$
    for all $\mathcal{L}_{i-1}^{(j)}$ **do**
      Train Selected Classifiers on $\mathcal{L}_{i-1} - \mathcal{L}_{i-1}^{(j)}$
      Use these classifiers to predict on $\mathcal{L}_{i-1}^{(j)}$ to obtain $\mathcal{P}_i^{(j)}$
      $\mathcal{P}_i = \mathcal{P}_i \bigcup \mathcal{P}_i^{(j)}$
    end for
    $\mathcal{L}_i = \mathcal{L}_0 \bigoplus \mathcal{P}_i$
  end for
  $\mathcal{V}_i = \mathcal{V}_0 \bigoplus \mathcal{P}_i(\mathcal{V})$
end for
Using combing method (4) on $\mathcal{P}_i(\mathcal{V})$
Calculate fitness f of E by using (6)
Return f and $\{h_k^{(i)}\} i = 1, \ldots, s$

form equally. The Nemenyi test in Fig 2 shows that VEGAS is better than all benchmark algorithms. In detail, VEGAS performs the best among all methods on 15 datasets. VEGAS ranks second on 5 datasets, and the prediction accuracy of VEGAS and the first rank method are not significant differences (0.9610 vs 0.9756 of gcForest on the Breast-cancer dataset, for example). The outstanding performance of VEGAS over the benchmark algorithms comes from (i) the use of multi-layer architecture over one-layer ensembles (ii) the use of ensemble selection to search for optimal configuration for VEGAS on each dataset.

Surprisingly, Random Forest ranks higher than the other benchmark algorithms in our experiment. Random Forest ranks the first on two datasets Hayes-Roth and Wine white (about 2% better than VEGAS for prediction accuracy). In contrast, VEGAS is significantly better than Random Forest on some datasets such as Hill-valley (about 30% better), Sonar (about 6% better), Vehicle (about 6% better), Tic-Tac-Toe (about 8% better).

MULES and XgBoost rank the middle in our experiment. VEGAS outperforms MULES on all datasets. MULES looks for optimal EoC of each layer by considering two objectives: maximising accuracy and diversity. Meanwhile, VEGAS learns the optimal configuration for all layers of the deep ensemble. It demonstrates the efficiency of the optimisation method, i.e. VLGA of VEGAS. For MLP, although we ran the experiments on its 3 different configurations and reported its best result for the comparison, this method is worse than VEGAS on up to 18 datasets. gcForest is worst among all methods on the experimental datasets. On some datasets such as Conn-bench-vowel, Hill-valley, Sonar, Texture, Tic-Tac-Toe, Vehicle, and Wine-white, gcForest performs poorly and by far worse than VEGAS.

Table 1. The classification accuracy and ranking of all methods

	gcForest	MLP	Random forest	XgBoost	MULES	VEGAS
Artificial	0.7905 (2)	0.6476 (6)	0.7619 (3)	0.7571 (4)	0.7238 (5)	**0.8000 (1)**
Breast-cancer	**0.9756 (1)**	0.9512 (5.5)	0.9561 (3)	0.9512 (5.5)	0.9512 (4)	0.9610 (2)
Cleveland	0.6000 (2.5)	0.5000 (6)	0.6000 (2.5)	0.5556 (5)	0.5889 (4)	**0.6111 (1)**
Conn-bench-vowel	0.6289 (6)	0.8365 (4)	0.9119 (2)	0.8428 (3)	0.8050 (5)	**0.9245 (1)**
Hayes-roth	0.8333 (3)	0.7708 (6)	**0.8750 (1)**	0.8125 (5)	0.8333 (4)	0.8542 (2)
Hill-valley	0.5852 (6)	0.8942 (3)	0.6593 (5)	0.6786 (4)	0.9766 (2)	**0.9849 (1)**
Iris	0.9556 (4)	**1.0000 (1)**	0.9333 (5.5)	0.9333 (5.5)	0.9556 (3)	0.9778 (2)
Led7digit	0.7067 (3)	0.7200 (2)	0.6933 (4)	0.6600 (5)	0.4400 (6)	**0.7333 (1)**
Musk1	0.8462 (3)	0.8252 (6)	0.8462 (4)	0.8322 (5)	0.8671 (2)	**0.8951 (1)**
Musk2	0.9515 (6)	0.9798 (3)	0.9773 (4)	0.9929 (2)	0.9727 (5)	**0.9939 (1)**
Newthyroid	0.9692 (4)	0.9846 (2.5)	0.9846 (2.5)	0.9538 (5)	0.9385 (6)	**1.0000 (1)**
Page-blocks	0.9464 (6)	0.9629 (4)	0.9695 (2)	0.9683 (3)	0.9568 (5)	**0.9720 (1)**
Pima	0.7229 (5)	0.7316 (4)	0.7403 (2)	0.7359 (3)	0.6840 (6)	**0.7576 (1)**
Sonar	0.8095 (6)	0.8889 (2)	0.8413 (3.5)	0.8413 (3.5)	0.8254 (5)	**0.9048 (1)**
Spambase	0.9392 (5)	0.9356 (6)	0.9551 (2)	0.9522 (3)	0.9508 (4)	**0.9594 (1)**
Texture	0.8436 (6)	**0.9933 (1)**	0.9752 (5)	0.9848 (4)	0.9855 (3)	0.9915 (2)
Tic-Tac-Toe	0.8368 (6)	0.9271 (4)	0.9236 (5)	**1.0000 (1.5)**	0.9792 (3)	**1.0000 (1.5)**
Vehicle	0.7283 (5)	0.6890 (6)	0.7638 (4)	0.7717 (3)	0.8189 (2)	**0.8307 (1)**

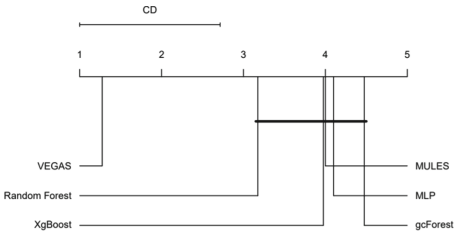

Fig. 2. The Nemenyi test result

4.3 Discussions

VEGAS takes higher training time compared to two deep ensemble models i.e. gcForest and MULES. On the Tic-Tac-Toe dataset, for example, gcForest used only 311.78 s for the training process compared to 15192.08 of VEGAS (100 individuals in each generation). Meanwhile, MULES (50 individuals in each generation) used 3154.86 s for its training process. It is noted that the training time of VEGAS can reduce based on either parallel implementation or early stopping of VLGA. Figure 3 shows the average and global best of the fitness function in the generations of VEGAS on 4 selected datasets. It is observed that the global bests converge quickly after several generations. On Balance and Artificial dataset, for example, their global bests converge after 8 iterations. Thus, the training time on some datasets can reduce by an early stopping based on the convergence of the global best.

On the other hand, although VEGAS creates more layers than gcForest and MULES (6.4 vs 3.8 and 2.75 on average), the classification time of VEGAS is

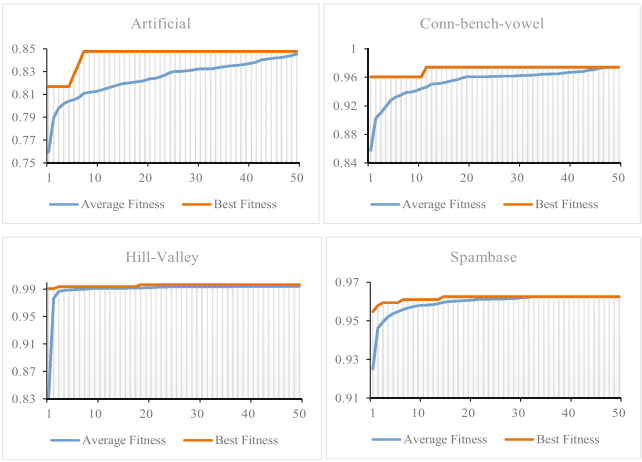

Fig. 3. The average and global best of the fitness function in the generations of VLGA on 4 datasets

competitive to those of gcForest and MULES. That is because on some datasets, VEGAS selects a small number of classifiers in each layer. On the Tic-Tac-Toe dataset, VEGAS takes only 0.016 s for classification with 4 layers and 6 classifiers in total. Meanwhile, gcForest (6 layers and 4800 classifiers in total) used 0.62 s to classify all test instances, and MULES (4 layers with 11 classifiers in total) used 0.26 s with its selected configuration [15].

5 Conclusions

The deep ensemble models have further improved the predictive accuracy of one-layer ensemble models. However, the appearance of unsuitable classifiers in each layer reduces predictive performance and the *computational/storage efficiency* of the deep models. In this study, we have introduced an ensemble selection method for the deep ensemble systems called VEGAS. We design the deep ensemble system involving multiple layers of the EoC. The training data is populated through layers by concatenating the predictions of classifiers in the subsequent layer and the original training data. The predictions of the classifiers in the last layer are combined by a combining method to obtain the final collaborated prediction. We proposed the VLGA to search for the optimal configuration, which maximizes the prediction accuracy of the deep ensemble model on each dataset. Three operators of VLGA were considered in this study, namely selection, crossover, and mutation. The experiments on 20 datasets show that VEGAS is better than both well-known ensemble methods and other deep ensemble methods.

References

1. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)

2. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: Proceedings of the 21st ICML (2004)
3. Chen, T., Guestrin, C.: Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD, pp. 785–794 (2016)
4. Dai, Q., Liu, Z.: Modenpbt: a modified backtracking ensemble pruning algorithm. Appl. Soft Comput. **13**(11), 4292–4302 (2013)
5. Demsar, J.: Statistical comparisons of classifers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
6. Lamini, C., Benhlima, S., Elbekri, A.: Genetic algorithm based approach for autonomous mobile robot path planning. Procedia Comput. Sci. **127**, 180–189 (2018)
7. Luong, V., Nguyen, T., Liew, A.: Streaming active deep forest for evolving data stream classification. arXiv preprint arXiv:2002.11816 (2020)
8. Mitchell, M.: An Introduction to Genetic Algorithms. MIT press, New York (1998)
9. Nazarahari, M., Khanmirza, E., Doostie, S.: Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm. Expert Syst. Appl. **115**, 106–120 (2019)
10. Nguyen, T., Dang, T., Baghel, V., Luong, V., McCall, J., Liew, A.: Evolving interval-based representation for multiple classifier fusion. Knowledge-based systems, p. 106034 (2020)
11. Nguyen, T., Dang, T., Liew, A., Bezdek, J.C.: A weighted multiple classifier framework based on random projection. Inf. Sci. **490**, 36–58 (2019)
12. Nguyen, T., et al.: Deep heterogeneous ensemble. Australian J. Intell. Inf. Process. Syst. **16**(1), (2019)
13. Nguyen, T., Luong, V., Liew, A., McCall, J., Van Nguyen, M., Ha, S.: Simultaneous meta-data and meta-classifier selection in multiple classifier system. In: Proceedings of the GECCO, pp. 39–46 (2019)
14. Nguyen, T., Pham, C., Liew, A., Pedrycz, W.: Aggregation of classifiers: a justifiable information granularity approach. IEEE Trans. Cybern. **49**(6), 2168–2177 (2018)
15. Nguyen, T., Van, N., Dang, T., Luong, V., McCall, J., Liew, A.: Multi-layer heterogeneous ensemble with classifier and feature selection. In: Proceedings of the GECCO, pp. 725–733 (2020)
16. Partalas, I., Tsoumakas, G., Vlahavas, I.: An ensemble uncertainty aware measure for directed hill climbing ensemble pruning. Mach. Learn. **81**(3), 257–282 (2010)
17. Qi, Z., Wang, B., Tian, Y., Zhang, P.: When ensemble learning meets deep learning: a new deep support vector machine for classification. Knowl.-Based Syst. **107**, 54–60 (2016)
18. Qiongbing, Z., Lixin, D.: A new crossover mechanism for genetic algorithms with variable-length chromosomes for path optimization problems. Expert Syst. Appl. **60**, 183–189 (2016)
19. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Evolving deep convolutional neural networks for image classification. IEEE Trans. Evol. Comput, **24**(2), 394–407 (2019)
20. Sun, Y., Xue, B., Zhang, M., Yen, G.G., Lv, J.: Automatically designing CNN architectures using the genetic algorithm for image classification. IEEE Trans. Cybern. (2020)
21. Yang, L., Wu, X.Z., Jiang, Y., Zhou, Z.H.: Multi-label learning with deep forest. arXiv preprint arXiv:1911.06557 (2019)
22. Zhou, Z.H., Feng, J.: Deep forest. arXiv preprint arXiv:1702.08835 (2017)