Multi-layer Heterogeneous Ensemble with Classifier and Feature Selection

Tien Thanh Nguyen
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Nang Van Pham
School of Electrical Engineering, Hanoi University of Science and Technology
Vietnam

Manh Truong Dang
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Anh Vu Luong
School of Information and Communication Technology, Griffith University
Australia

John McCall
School of Computing Science and Digital Media, Robert Gordon University
Aberdeen, UK

Alan Wee Chung Liew
School of Information and Communication Technology, Griffith University
Australia

ABSTRACT
Deep Neural Networks have achieved many successes when applying to visual, text, and speech information in various domains. The crucial reasons behind these successes are the multi-layer architecture and the in-model feature transformation of deep learning models. These design principles have inspired other sub-fields of machine learning including ensemble learning. In recent years, there are some deep homogenous ensemble models introduced with a large number of classifiers in each layer. These models, thus, require a costly computational classification. Moreover, the existing deep ensemble models use all classifiers including unnecessary ones which can reduce the predictive accuracy of the ensemble. In this study, we propose a multi-layer ensemble learning framework called MUlti-Layer heterogeneous Ensemble System (MULES) to solve the classification problem. The proposed system works with a small number of heterogeneous classifiers to obtain ensemble diversity, therefore being efficiency in resource usage. We also propose an Evolutionary Algorithm-based selection method to select the subset of suitable classifiers and features at each layer to enhance the predictive performance of MULES. The selection method uses NSGA-II algorithm to optimize two objectives concerning classification accuracy and ensemble diversity. Experiments on 33 datasets confirm that MULES is better than a number of well-known benchmark algorithms.

CCS CONCEPTS
• Computing methodologies → Ensemble methods; • Mathematics of computing → Evolutionary Algorithms;

KEYWORDS
Ensemble method, deep learning, multiple classifiers, ensemble of classifiers, feature selection, classifier selection

1 INTRODUCTION
In recent years “deep learning” has become one of the most popular terms in the research community. Deep Neural Networks (DNNs) have shown outstanding achievements in many supervised learning tasks for visual, text, and speech information [13]. In computer vision, for instance, Convolutional Neural Network (CNN), a DNN, significantly outperforms traditional machine learning algorithms on the large-scale ImageNet classification task [11]. Zhou and Feng summarized three crucial reasons for the success of DNNs, including layer-by-layer processing, in-model feature transformation, and sufficient model complexity [30]. From the training data, DNNs generates the new training input at each layer in which the new training features reflect the different high-level abstract representations of the original data. This multi-layer design brings out different aspects of the original data in comparison to the flat networks i.e. single-hidden layer networks or traditional machine learning models which only work on the original data. Moreover, DNNs are designed with many hyper-parameters in many layers, resulting in high model complexity. Such high complexity is needed to explore the large training data.

DNNs are built based on neural networks by using backpropagation technique to train multi-layer with differentiable nonlinear modules. Recently, some efforts are being done to extend deep learning models with many layers of ensemble of classifiers [3, 21, 24, 30]. An ensemble is a collection of classifiers whose prediction are combined so as to achieve better performance than its constituent members. The key success of ensemble learning results from the diverse exploration of the original data using different hypothesis i.e. classifiers. By constructing deep ensemble models, we expect to
benefit from both the deep learning and ensemble learning mechanisms. Although these deep ensemble models achieve superior classification performance on many datasets, the existing ensemble requires a large number of classifiers, resulting in huge memory usage and expensive computation. Moreover, these deep ensemble models do not perform classifier selection for each layer. In fact, the presence of some poor classifiers in each layer could degrade the system performance. We addressed these limitations by designing a multi-layer ensemble model with ensemble selection.

This paper aims to design an effective ensemble-based supervised learning system for classification inspired by the representation learning using layer-by-layer processing of DNNs. To achieve this goal, the objectives below have been specified:

- Design a novel ensemble model involving a multi-layer architecture with diverse classifiers in each layer (called MULES). By combining the advantages of ensemble learning and deep learning, the proposed system is expected to perform well in many classification tasks.
- Investigate an approach to simultaneously select classifiers and the features in each layer. The proposed approach thus can overcome the limitation of existing deep ensemble models by selecting only the subset of suitable classifiers and features for each layer.
- Propose fitness measures for MULES by considering not only classification accuracy but also the ensemble diversity generated by classifiers in each layer.
- Compare the performance of MULES to several existing ensemble methods and deep ensemble models.

In Section 2, we briefly introduce ensemble learning for classification and some recent developments in Evolutionary Algorithms for ensemble systems. In Section 3, we give a detail description of the general architecture for the MULES. 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 Ensemble Learning

Ensemble learning explores the original data by using different hypothesis, i.e. classifiers. There are two strategies to generate the set of classifiers: heterogeneity or homogeneity [2, 4, 19]. In homogeneous ensemble, many classifiers are obtained by training one learning algorithm on many different training sets obtained from the original one. Homogeneous ensemble uses many classifiers to ensure that the classification error converges to its asymptotic value. This thus requires huge memory storage for classifiers and high computational budgets for classification. Among many homogeneous ensemble frameworks, Random Forest and XgBoost are the top-performance methods [4, 8]. The heterogeneous ensemble, in contrast, uses a small number of different learning algorithms on the training data to generate diverse classifiers. The research on heterogeneous ensemble focuses on designing combining algorithms that effectively combine the predictions of the classifiers. Some examples of combining algorithms are Bayesian-based method with Mixture of Gaussians [16] and Information Granularity-based method [19].

Ensemble methods based on a multi-layer architecture is nowadays becoming a popular trend in ensemble system design. Such systems involve more than one layer of ensembles. Some examples of two layers of ensembles are two-layer heterogeneous ensemble [17], and a two-layer ensemble of Random Subspace and Bagging [29]. Viola and John [25] proposed a cascade model as a sequence of binary classifiers. If one classifier outputs positive value, the data is transmitted to the next classifier. A deep ensemble learning model with more than two layers called geForest was introduced by Zhou and Feng [30] with four random forests in each layer. Utkin et al. [24] optimized geForest by considering the weights of the trees in the same forest when averaging their predictions. These weights are found by minimizing a loss function on the training data based on the Euclidean distance between the weighted average vector and the crisp distribution vector based on the class labels of the training instances. Qi et al. [21] introduced a deep model with ensemble of SVM classifiers in each layer. The parameters of the models including the kernel functions of SVM classifiers, the number of classifiers, and the weights of features are found by using AdaBoost. In deep ensemble model in [3], the training input for the layer is formed by concatenating the original training data and all predictions of classifiers from the layer to the layer. The optimal hyper-parameters of the proposed model, including the number of classifiers in each layer, the number of layers, and the parameters for classifiers in each layer, are found by an Evolutionary Algorithm.

2.2 Selection Problem with Evolutionary Algorithms

In the past years, many applications of Evolutionary Algorithm (EA) have been proposed to select the optimal subset of classifiers or features to improve the ensemble performance. Nguyen et al. [15] proposed the encoding of both the classifiers and six combining methods in a chromosome. The final optimal set of combining methods is combined once again using the OWA operator. Chen et al. [5] used Ant Colony Optimization (ACO) to find the optimal set of classifiers in the ensemble system. A combining algorithm is chosen from a given set of algorithms by using the uniform distribution. Wang et al. [26] used NSGA-II [6] to search for the optimal set of classifiers generated by training a regression tree on 100 new training sets. The new training data was obtained by applying the random subspace and bootstrap resampling techniques on the original training set.

EA is also widely applied to feature selection which aims to remove redundant and irrelevant attributes to improve predictive performance and efficiency [28]. In ensemble learning, some EA-based feature selection methods have been proposed to select features for classifiers or to select predictions of classifiers for the combining algorithms. Kim and Cho [9] encoded the feature selection methods that will be used for each learning algorithm to learn the classifiers. The optimal solution is then obtained by using GA. In their extended version, the chromosomes in GA encodes the weight for each feature-classifier pair for the weighted sum combining rule. Bacaule et al. [1] built the homogeneous ensemble system of SVM classifiers in which the parameters of each classifier i.e. the regularization constant and the kernel width and the features used by each classifier are encoded via the binary encoding scheme. Nguyen et al. [18] used ACO to simultaneously select the optimum from the
predictions of classifiers and the set of combining algorithms to improve ensemble performance.

Recently, some Evolutionary Algorithms-based approaches have been proposed for DNNs optimization in terms of their topology and hyper-parameters [14]. In [27], Genetic Algorithm was used to optimize the network structures of CNNs. The CNN network was organized in several states including ordered nodes. A binary encoding scheme was used in each stage to encode the connections between the nodes insides. In [23], the three different building blocks: the convolutional layer, the pooling layer, and the fully connected layers in a CNN were encoded in one chromosome for evolution. The individuals are organized in variable-length encoding so as to reach the optimal depth of architecture. In [10], the hyper-parameters of DenseNet, a CNN, including the number of blocks, the number of layers in each block, and the growth rate was optimized by considering two objectives, namely computation accuracy and computational cost. The taxonomy of Evolutionary Algorithms to optimize DNNs can be found in [14].

3 PROPOSED SYSTEM

3.1 Deep Heterogeneous Ensemble System

Let $D = \{(x_n, y_n)\}$, $|D| = N$ be the training data, where $x_n \in \mathbb{R}^D$ is the $D$-feature vector of the training instance; $y_n \in \mathcal{Y} = \{y_m\}$, $|\mathcal{Y}| = M$ be its corresponding label that belongs to the label set with $M$ labels, $\mathcal{K} = \{\mathcal{K}_k\}$ be the set of learning algorithms $|\mathcal{K}| = K$. In supervised learning (i.e. classification or regression), we aim to learn a hypothesis $h$ (i.e., classifier) for the unknown relationship $g : x_n \rightarrow y_n$ and use this hypothesis to assign a label for each unlabeled instance. By using an ensemble system, we train an ensemble of classifiers (EnC) on $D$ to obtain several different hypotheses $\{\mathcal{h}_i\}$ for $g$. These hypotheses are then combined by a combining algorithm $C : \vec{h} = C(\vec{h}_i)$ for final decision making.

In this study, we propose MULES to solve the classification problems. The proposed system has multiple layers including different classifiers in each layer. The classifiers in the first layer train on the original training data and generate the new input training data for the second layer. The classifiers in the next layer train on the new training data generated by the preceding layer and generate the new training data for the subsequent layer. A layer thus can be viewed as the feature generator which generates features for the next layer. The combining algorithm then trains on the predictions of the classifiers in the last layer for collaboration prediction. In fact, this model is based on the idea of DNNs in which the data is passed through several layers in the learning system. However, it is different from DNNs since the information in the MULES is processed under the feed-forward mechanism in which the information is only passed from one layer to the next layer and does not involve back-propagation like in neural network methods. Moreover, several different learning algorithms will be used in each layer, whereas DNNs only work with layers of neurons. An illustration of MULES with a 3 layer-architecture and several different classifiers in each layer is shown in Figure 1. The outputs of the Decision Tree, Naïve Bayes, and Random Forest classifier in the first layer are combined with the original training data to generate the input for the second layer. A similar scheme is used at the second layer with two classifiers (SVM and Decision Tree) and in the third layer with two classifiers (LDA and SVM), before the output of the third layer is combined for the final prediction.

Two following questions arise from the proposed MULES:

- How to grow the deep model?
- How to combine the predictions of the last layer to obtain the final prediction?

To grow the deep model of MULES, it is needed to generate the input data for one layer. In this study, we concatenate the original training data with the predictions of classifiers of the $i^{th}$ layer to form the new input data for the $(i + 1)^{th}$ layer. We use $T_i$-fold cross-validation on the input data for the $i^{th}$ layer to generate the predictions for the training data. In cross-validation, the input for the $i^{th}$ layer is divided into $T_i$ disjoint parts in which the cardinality of each part is nearly similar. The predictions for observation in each part will be obtained by using classifiers trained on the other parts. An observation thus will be predicted one time. We obtain the predictions of the classifiers output the predictions in the form of probability [19]:

$$\sum_{m=1}^{M} p_{i,k,m}(x_n) = 1; k = 1, ..., K; n = 1, ..., N \quad (1)$$

Let $\mathcal{L}_1$ denotes the new data generated by the $i^{th}$ layer as the input for the $(i + 1)^{th}$ layer $(i = 1, 2, ...) \text{ and } \mathcal{L}^{(i)} = D$. In this study, we propose to generate $\mathcal{L}_i$ by concatenating the original training data and the prediction of each classifier for each observation. The augmented features, i.e. predictions and original features is expected to improve the discriminative characteristic of the training data [30]. In detail, the predictions for the training set at the $i^{th}$ layer is given as a $N \times (MK)$ matrix $P^{(i)}(x_n) = [P_{1,1}^{(i)}(x_n), P_{1,2}^{(i)}(x_n), ..., P_{K,M}^{(i)}(x_n)]$ while the original training data is given as a $N \times D$ matrix. The concatenation operator will generate a new input for the $(i + 1)^{th}$ layer in the form of a $N \times (D + MK)$ matrix including the original training features and predictions of all instances arranged in the order as in Eqn. (3).

$$\mathcal{L}^{(i)} = \left[\mathcal{L}^{(i)}(x_1) ... \mathcal{L}^{(i)}(x_N)\right]^T \quad (2)$$

$$\mathcal{L}^{(i)}(x_n) = [x_{n1}, ..., x_{nD}; P_{1,1}^{(i)}(x_n), P_{1,2}^{(i)}(x_n), ..., P_{K,M}^{(i)}(x_n)] \quad (3)$$

In ensemble systems, an algorithm is used to combine the predictions of classifiers for the collaborated prediction. In this study, we used the Sum Rule method for combining [10]. Sum Rule summarizes the predictions of each instance with regard to each class label and assigns the instance to the class label associated with the maximum value. The combined prediction on an instance $x$ at the $i^{th}$ layer is given by:

$$x \in y_t \text{ if } t = \text{argmax}_{m=1, ..., M} \left\{ \sum_{k=1}^{K} p_{k,m}^{(i)}(x) \right\} \quad (4)$$

The classification process works in a straightforward manner where a test instance $x$ is fed forward through the layers to finally obtain
At each layer of MULES, the original training data is concatenated with the original training data in a shape of a \((D + MK)\)-vector. The prediction vector of the last layer is fed into the combiner to generate the predicted label.

To keep the figure simple, we do not show the concatenation between training data and the prediction of each classifier to create the input for the next layer.

**Figure 1:** An illustration of the proposed deep heterogeneous ensemble system.

### 3.2 Simultaneous Classifier and Feature Selection Approach

At each layer of MULES, the original training data is concatenated to the predictions of classifiers to generate the input for the next layer. The prediction of a classifier can be viewed as scaled data from feature domain to probability domain. The new data in the probability domain may have better discriminative characteristics than the original training data in case of correct predictions where observations that belong to the same class will have similar prediction results and stay close together in the probability domain. However, in cases of wrong predictions, the discriminative characteristic of data will be mitigated. Concatenating the original training data and the classifier’s predictions can therefore result in better or worse discriminative characteristics than the original training data. In this work, we develop a simultaneous classifier and feature selection method for each layer of MULES in which the system performance can be improved by either removing some inconsistent classifiers in each layer or selecting the most suitable features for each selected classifier when training on the input data.

We design a two-part encoding representation for the simultaneous classifier and feature selection method. The first part \([h_k^{(i)}, k = 1, \ldots, K]\) of the proposed encoding \(E^{(i)}\) in Eqn. (5) is the encoding of the \(K\) classifiers and the second part \([f_{kd}^{(i)}, k = 1, \ldots, K; d = 1, \ldots, d_i]\) is the encoding of the features used by each classifier. In both parts, each gene in the encoding get two binary values \([0,1]\) showing which classifiers is absence or presence in each layer (Eqn. (6)) or which features will be used by a classifier (Eqn. (7)).

\[
E^{(i)} = \begin{bmatrix}
h_1^{(i)} & \ldots & h_K^{(i)} \\
f_{11}^{(i)} & \ldots & f_{1d_i}^{(i)} \\
f_{21}^{(i)} & \ldots & f_{2d_i}^{(i)} \\
\vdots & \ddots & \vdots \\
f_{K1}^{(i)} & \ldots & f_{Kd_i}^{(i)}
\end{bmatrix}
\]  

(5)

where \(d_i\) is the number of training features for layer \(i\)

\[
h_k^{(i)} = \begin{cases} 
1, & \text{\(k^{th}\) classifier is selected at layer \(i\)} \\
0, & \text{otherwise}
\end{cases}
\]  

(6)

\[
f_{kd}^{(i)} = \begin{cases} 
1, & \text{\(d^{th}\) feature is used by \(k^{th}\) classifier at layer \(i\)} \\
0, & \text{otherwise}
\end{cases}
\]  

(7)

In this study, the model selection is conducted at each layer to obtain the optimal set of classifiers and their features. The purpose of classifiers in all layers except the last one is to generate the input data for the next layer. Thus, the attention on only prediction accuracy is not enough to ensure the generation of effective input data. Here we consider bi-objective optimization for the model selection problem. The first objective is the accuracy of the classification task on the validation set \(V\):

\[
\max_{E^{(i)}} \left\{ \frac{1}{|V|} \sum_{n=1}^{|V|} [\tilde{h}^{(i)}(x_n) = y_n] \right\}
\]  

(8)

where \(\tilde{h}^{(i)}\) is the combining model using Sum Rule at the \(i^{th}\) layer, \(|\cdot|\) denotes the cardinality of a set, and \([\cdot] = 1\) if the condition is true, otherwise equal 0.

The second objective concerns the ensemble diversity generated by classifiers in each layer. It is widely recognized that diversity is an important factor to be considered when designing an ensemble system \([12, 20, 30]\). Kuncheva et al. [12] studied ten pairwise and non-pairwise diversity measures and examined the relationship between accuracy and diversity. Diversity was considered when designing an evaluation measure for ensemble selection (aka ensemble pruning) [20]. In this study, we measure the diversity of
classifiers in each layer by using $Q$-statistic computed on the predictive outcomes of two classifiers [12]:

$$Q_{ij} = \frac{N_{00}N_{11} - N_{10}N_{01}}{N_{00}N_{11} + N_{10}N_{01}}$$ (9)

in which $N_{ij}$ denotes the number of samples which are correctly classified by both classifiers, $N_{0i}$ denotes the number of samples correctly classified by the $i^{th}$ classifier but are not correctly classified by the $j^{th}$ classifier. $N_{00}$ and $N_{01}$ is inversion of $N_{11}$ and $N_{10}$, respectively.

The $Q$-statistic diversity of the $i^{th}$ layer is computed by averaging the diversity of pairwise of selected classifiers. It is noted that the lower the value of the $Q$-statistic, the higher the value of the ensemble diversity. The second objective, which aims to maximize the ensemble diversity, is given by:

$$\min_{E_{00}} \left\{ \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} Q_{ij} \right\}$$ (10)

in which $m$ is the number of selected classifiers.

### 3.3 Algorithms

In the training process, MULES receives the inputs including the training data $\mathcal{D}$, the validation data $\mathcal{V}$, the learning algorithms $\mathcal{K}$ and early stopping rounds $T_{stop}$. In each layer, MULES searches for the optimal subset of classifiers and features using bi-objective optimization. Among the Evolutionary Algorithms introduced in solving the multi-objective optimization problems, the non-dominated sorting genetic algorithm II (NSGA-II) is one of the most popular and effective methods [26]. NSGA-II was designed with the elitism and diversity preserving characteristics so as to find the Pareto-optimal solution (a set of non-dominated solutions) [6]. In this study, we use NSGA-II in each layer to solve the combinational optimization problem given in (8) and (10) (line 4). Since there is no clear relationship between accuracy and diversity of an ensemble [12], the use of NSGA-II makes these objectives to be considered separately, thus maintaining the richness of both criteria in the evolution process. The final result is still the prediction accuracy as it is the main objective of interests [22, 26]. Therefore, we simply choose the chromosome with the best accuracy from the last generation of NSGA-II as the final selected individual.

Based on the encoding of the selected individual, we get the set of selected classifiers $\mathcal{H}^{(i)}$ and their associated features $\mathcal{F}^{(i)}$. The selected algorithms $\mathcal{K}^{(i)}$ and $\mathcal{F}^{(i)}$ is then used on $\mathcal{L}^{(i-1)}$ with the Cross-Validation procedure to generate the predictions for the training instances at the $i^{th}$ layer i.e. a $N \times (MK)$ matrix $\mathcal{P}^{(i)}$. In line 8, $\mathcal{P}^{(i)}$ is concatenated with the original training data $\mathcal{L}^{(0)}$ to form the input training data $\mathcal{L}^{(i)}$ for the $(i+1)^{th}$ layer.

In this study, we evaluate the predictive performance of MULES on $\mathcal{V}$ on each layer to automatically determine the number of layers. In line 9, we used the selected $\mathcal{H}^{(i)}$ and the selected features for each classifier $\mathcal{F}^{(i)}$ on $\mathcal{V}_{i-1}$ to obtain the predictions $\mathcal{P}_{\mathcal{V}}^{(i)}$ for instances in the validation set at the $i^{th}$ layer. As mentioned in (4), we apply Sum Rule on $\mathcal{P}_{\mathcal{V}}^{(i)}$ to obtain the predicted class label $\mathcal{Y}$ (line 10). By comparing $\mathcal{Y}$ and the ground truth of class labels of the instances in $\mathcal{V}$, we can calculate the classification error rates of the $i^{th}$ layer on $\mathcal{V}$ (line 11). The predictions $\mathcal{P}^{(i)}_{\mathcal{V}}$ is also concatenated with the original validation set $\mathcal{V}$ to obtain $\mathcal{V}_{i}$ for the evaluation at the $(i+1)^{th}$ layer.

We use a checkpoint to save the current best result and the number of layers when MULES enhances its performance on the validation set (line 13-16). After a specific number of layers, if the classification error on the validation part does not improve, we stop growing new layers and then use the checkpoint to choose the optimal number of layers.

In the testing process, an unlabeled instance is pass through all layers in MULES until reaching the last layer. In the $i^{th}$ layer, by referencing $\mathcal{F}^{(i)}$, we can choose the features of the test instance for each selected classifier. The predictions of the selected classifiers for the test instance are concatenated to the original features to form the new training data for the next layer. We use Sum Rule on the predictions of the selected classifiers at the last layer to assign a class label for the test instance.
Algorithm: MULES
Input:
- Training data \( D = \{(x_n, y_n), n = 1...N\} \)
- Validation data \( V = \{(x_i, y_i), i = 1...|V|\} \)
- \( K \) learning algorithm \( K \)
- Early stopping rounds \( T_{stop} \)
Output: List of selected classifiers and their associated features at layer \( i \)
1. Initialize \( L^{(0)} = D, V_0 = V, i = 0, \text{best\_error\_model} = 1.0 \)

2. while True do
   3. \( i++ / \text{Train layer } i \) ;
   4. Apply NSGA-II with (8) and (10) to obtain optimal encoding \( E \) for layer \( i \) ;
   5. Get the set of selected classifier \( H^{(i)} \) and their associated features \( F^{(i)} \) from \( E \) ;
   6. Get the selected algorithm \( K^{(i)} \) ;
   7. \( p^{(i)} = \text{Cross-Validation}(K^{(i)}, F^{(i)}, L^{(i-1)}) \); 
   8. \( L^{(i)} = p^{(i)} \cup L^{(0)} \);
   9. \( p_{V_i}^{(i)} = \text{predict}(H^{(i)}, F^{(i)}, V_{(i-1)}) \);
   10. \( Y = \text{Sum\_Rule\_predict}(p_{V_i}^{(i)}) \) by (4) ;
   11. \( \text{error} = \text{Loss\_function}(Y, V) \);
   12. \( V^{(i)} = p_{V_i}^{(i)} \cup V_0 \);
   13. if error \( < \text{best\_error\_model} \) then
      14. \( \text{best\_error\_model} = \text{error} ;
      15. \( T_{layer} = i ;
      16. end
      17. if error does not decrease after \( T_{stop} \) layers then
         18. Break
      19. end
   20. end
21. Return \( H^{(i)} \text{and} F^{(i)} , i = 1, ..., T_{layers} ;

4 EXPERIMENTAL STUDIES
4.1 Configurations
MULES was constructed with five learning algorithms: \( K \) Nearest Neighbor (KNN, \( K \) was set to 5), Logistic Regression, Naïve Bayes (Bernoulli distribution was used), Random Forest (with 200 estimators), and Decision Tree. All these methods were implemented from the scikit-learn library with default parameters. We followed the experiments in [30] in which 80% of labeled data is used for the training part and the remainder is used for the validation part. We used the 2-fold Cross-Validation in one layer to generate the predictions for the training part. The layer growing process is stopped if the classification error rate on the validation part does not improve after 5 layers. For NSGA-II algorithm using in each layer, the maximum number of generations was set to 100 and the population size was set to 50.

We used some well-known benchmark algorithms to evaluate the performance of MULES: Random Forest (with 2000 trees), gcForest (4 forests and 500 trees in each forest), XgBoost (with 2000 trees), and Multi-Layer Perceptron (MLP). As the performance of MLP significantly depends on the network structure, we performed grid search on different parameters and reported the best result for the comparison. For MLP, we followed the experiments in [30] in which it was constructed with different configurations: input-30-20-output, input-50-30-output, and input-70-50-output.

We used the Friedman test to test the null hypothesis that all methods perform equally on all 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 for pairwise comparison on all datasets [7]. The experiments were conducted on 33 datasets selected from various sources such as the UCI Machine Learning Repository and OpenML.

4.2 Experimental Results
Comparison to baselines: The prediction error rates of MULES and five benchmark algorithms are presented in Table 1. Some observations can be made:

- Based on the Friedman test, the null hypothesis was rejected with the P-Value. The Nemenyi test shows that MULES is better than XgBoost, gcForest, and MLP.
- MULES achieves the lowest average rank among all methods (rank value 1.76). On the 33 datasets, MULES ranks first in 16 cases (48.5%) and ranks second in 13 cases (39.4%). Although MULES performs poorly on 4 datasets, i.e. it ranks fourth on the Breast-Cancer and Twonorm datasets and ranks third on the Cleveland and Spambase datasets, the prediction error rates of MULES and the first rank method are not significant differences (0.0488 vs. 0.0244) of gcForest on Breast-Cancer dataset, for example.
- Random Forest and XgBoost rank second and third with average rank value 2.59 and 3.12, respectively. Random Forest ranks first on 9 datasets while XgBoost ranks first on 4 datasets. Random Forest is better than MULES on only two datasets Hayes-Roth (0.1250 vs. 0.1667) and Wine_white (0.3014 vs. 0.3449). In contrast, MULES is better than Random Forest on 6 datasets namely Chess-krvk (about 6% better), Embryonal (more than 10% better), Hill-valley (about 30% better), Madelon (about 10% better), Tic-tac-toe and Vehicle (about 5% better).
- gcForest is worse than MULES in our experiment. One some datasets such as Chess-krvk, Electricity, Hill-valley, and Isolde, gcForest performs poorly and by far worse than MULES. gcForest ranks first on three datasets namely Breast-Cancer, Marketing, and Cleveland, but the differences in comparison to classification results of MULES are not significant.
- MLP is the poorest method in our experiment. Although MLP was run with different configurations and we reported the best result for comparisons, its performance is by far worse than MULES. On some datasets like Ring, Satimage, and...
Table 2: An example of the obtained configuration for the validation data of some datasets. Moreover, MULES generated going deeply can improve the performance of the deep model on MULES generated 4 layers of ensemble of classifiers. This means on its prediction performance on the validation set. On average, MULES can automatically determine the number of layers based on its prediction performance on the validation set. Like gcForest, the comparisons between the number of layers generated by gcForest and MULES on the experimental datasets. Like gcForest, the comparisons between the number of layers generated by gcForest and MULES on Tic-tac-toe dataset.

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Features</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>6 original features</td>
<td>0110101111</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>5 original features</td>
<td>0100101111</td>
</tr>
<tr>
<td>Random Forest (200)</td>
<td>6 original features</td>
<td>0110101111</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>7 original features + 5 predictions</td>
<td>0011111111</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 4 predictions</td>
<td>0100101000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>6 original features + 2 predictions</td>
<td>0011010100</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 8 predictions</td>
<td>1000110000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>3 original features + 6 predictions</td>
<td>0100100010</td>
</tr>
<tr>
<td>Random Forest (200)</td>
<td>5 original features + 1 prediction</td>
<td>1000101111</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>7 original features + 5 predictions</td>
<td>0011111111</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 4 predictions</td>
<td>0100101000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>6 original features + 2 predictions</td>
<td>0011010100</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 8 predictions</td>
<td>1000110000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>3 original features + 6 predictions</td>
<td>0100100010</td>
</tr>
<tr>
<td>Random Forest (200)</td>
<td>5 original features + 1 prediction</td>
<td>1000101111</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>7 original features + 5 predictions</td>
<td>0011111111</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 4 predictions</td>
<td>0100101000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>6 original features + 2 predictions</td>
<td>0011010100</td>
</tr>
<tr>
<td>KNN(5)</td>
<td>4 original features + 8 predictions</td>
<td>1000110000</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>3 original features + 6 predictions</td>
<td>0100100010</td>
</tr>
</tbody>
</table>

Table 2: An example of the obtained configuration for MULES on Tic-tac-toe dataset.

Vehicle, the classification error rates of MLP are significantly higher than those of MULES.

The number of layers and configuration: Figure 4 presents the comparisons between the number of layers generated by gcForest and MULES on the experimental datasets. Like gcForest, MULES can automatically determine the number of layers based on its prediction performance on the validation set. On average, MULES generated 4 layers of ensemble of classifiers. This means going deeply can improve the performance of the deep model on the validation data of some datasets. Moreover, MULES generated more number of layers than gcForest, 4 compared to 3.7 on average. Exceptionally, both methods generate only one layer on 9 and 6 datasets. In these cases, going deeply with the new proposed input data does not bring benefits to the ensemble system.

We further analyze the benefits of the multi-layer architecture in MULES. Figure 5 shows the reductions of the prediction error rates on the test data of MULES through layers in the deep model. On Hayes-Roth dataset, for example, where MULES generates the model with 2 layers, the prediction error rate reduces from 0.1875 to 0.1667. On Tic-Tac-Toe dataset, the prediction error rate reduces from 0.1042 to 0.059 from the first layer to second layer, and continue to reduce to 0.0486 at the third layer and to 0.0208 at the fourth layer. This figure demonstrates the advantages of layer-by-layer
MULES can obtain high prediction accuracy and effectiveness in resource usage in terms of memory and computation requirements.

**Classification time:** Although MULES takes much higher training time than gcForest, the classification time of MULES is lower than gcForest. On Tic-tac-toe dataset, for example, MULES used 3154.86 second for training process compared to only 311.78 of gcForest. Meanwhile, gcForest used 0.62 second to classify all test instances while MULES only used 0.26 second.

MULES obtains a subset of classifiers and their features in each layer. From the obtained configuration for Tic-tac-toe dataset in Table 2, after the evolution process, only 3 and 4 classifiers were kept in the first and second layer. The third and fourth layer only have 2 classifiers. Therefore, only 11 classifiers are maintained in MULES to classify instances on this dataset. That makes MULES takes less time during classification. In contrast, gcForest used 4 Random Forests involving 500 trees in each forest. MLP also takes high computation for the training process as its configuration is somehow problem-dependent that requires a procedure to search for the optimal one. It is noted that like population-based Evolution Algorithms, NSGA-II can be implemented in parallel. This can further reduce the training time of MULES.

### 5 CONCLUSIONS

In summary, we introduced a Multi-Layer Heterogeneous Ensemble System (MULES) inspired by the layer-by-layer processing of DNNs. MULES includes several layers of the ensemble of different classifiers in which the classifiers in one layer train on the new training data generated by the preceding layer. The new training data for one layer is the concatenation of the predictions of the classifiers in the preceding layer and the original training data. We train a combining algorithm on the predictions of classifiers in the last layer for the final collaborated prediction. Since the ensemble in each layer can contain the unnecessary classifiers which increase the prediction error of the ensemble, we propose an Evolutionary Algorithm-based method to select the optimal set of classifiers and their features on each layer. The optimization problem is considered under two objectives concerning the prediction error and ensemble diversity. We used NSGA-II, a popular and effective multi-objective evolutionary algorithm, to solve this optimization problem.

Experiments on 33 datasets confirm that MULES is better than MLP, Random Forest, gcForest, and XgBoost in terms of predictive performance and efficiency.

Two solutions to enhance the performance of MULES could be implemented in the future. First, in this study we concatenated the original training data and the predictions of classifiers in one layer to generate the input data for the next layer. In general, MULES can generate multiple layers and its prediction accuracy becomes better on the deeper layers. The exception occurred on 6 datasets where going deeply does not achieve any improvements for the classification process. A new input data is needed to populate the deep model in these cases. Second, we used NSGA-II to search for the set of classifiers and their features in each layers. Parallel implementation of NSGA-II can be used in MULES to reduce its training time.