Prototype Generation Using Multiobjective Particle Swarm Optimization for Nearest Neighbor Classification

Weiwei Hu and Ying Tan, *Senior Member, IEEE*

***Abstract*—The nearest neighbor (NN) classifier suffers from high time complexity when classifying a test instance since the need of searching the whole training set. Prototype generation is a widely used approach to reduce the classification time, which generates a small set of prototypes to classify a test instance instead of using the whole training set. In this paper, particle swarm optimization is applied to prototype generation and two novel methods for improving the classification performance are presented: 1) a fitness function named error rank and 2) the multiobjective (MO) optimization strategy. Error rank is proposed to enhance the generation ability of the NN classifier, which takes the ranks of misclassified instances into consideration when designing the fitness function. The MO optimization strategy pursues the performance on multiple subsets of data simultaneously, in order to keep the classifier from overfitting the training set. Experimental results over 31 UCI data sets and 59 additional data sets show that the proposed algorithm outperforms nearly 30 existing prototype generation algorithms.**

***Index Terms*—Error rank, multiobjective (MO) optimization, nearest neighbor (NN) classification, particle swarm optimization (PSO), prototype generation.**

I. Introduction

NEAREST neighbor classification is a classic and widely used supervised learning algorithm. To classify an instance (i.e., a test instance), this algorithm first gets a certain number of instances from the training set which are the nearest to the test instance. These instances are called nearest neighbors (NNs) of the test instance. The majority class of the NNs is assigned to the test instance.

The simplest version of NN classification is 1NN. 1NN just gets one nearest instance in the training set for a test instance. The class of the nearest instance is assigned to the test instance.

The main problem faced by NN classification is its low time and space efficiency when classifying a test instance. The classifier has to store the whole training set in the memory. For each test instance, the classifier has to search the whole training set for the NNs, which is very time-consuming.

Prototype generation [1] is a very effective method to overcome the problem faced by NN classification. It generates a small set of prototypes based on the training set and just gets the NNs from the small set of prototypes, rather than from the whole training set. This approach needs much less space to store the prototypes, and searching the small set is much faster than searching the whole training set.

Traditional prototype generation methods include learning vector quantization [2], bootstrap technique [3], and mixtures of Gaussians [4]. Evolutionary computation is a new approach to generate prototypes. As a kind of optimization method, evolutionary computation is used to search for the best positions of prototypes by regarding the classification performance as the fitness function. Evolutionary computation-based prototype generation algorithms usually work better than nonevolutionary algorithms, since evolutionary algorithms are good at global optimization and they generally generate and evaluate tens of thousands of candidate solutions when searching for the best set of prototypes.

This paper proposes two novel techniques for evolutionary computation-based prototype generation, in order to further improve the performance of NN classification. We found that particle swarm optimization (PSO) is able to achieve better classification performance when using the two proposed techniques. Therefore, we use PSO-based prototype generation scheme as the basic framework in this paper.

PSO is a well-known evolutionary computation algorithm to search for the optimal solution of a specific problem [5]. PSO maintains a swarm of moving particles which represent candidate solutions. Each particle renews its velocity and position according to its best position and the swarm's best position found so far at each iteration. By this way, a particle can communicate its information with other particles in the swarm. Such collaborative searching strategy makes PSO good at global search. PSO has become one of the most popular optimization algorithms for continuous real value-based problems and has been applied to a variety of engineering fields.

The two main contributions in this paper are as follows.

1) A fitness function named error rank is proposed to enhance the generalization ability. The traditional fitness function error rate only considers whether the training

Manuscript received April 10, 2015; revised August 11, 2015; accepted September 20, 2015. Date of publication October 19, 2015; date of current version November 15, 2016. This work was supported in part by the Natural Science Foundation of China under Grant 61375119, Grant 61170057, and Grant 60875080, and in part by the National Key Basic Research Development Plan (973 Plan) Project of China under Grant 2015CB352302. This paper was recommended by Associate Editor S. Yang. *(Corresponding author: Ying Tan.)*

The authors are with the Key Laboratory of Machine Perception, and the Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: weiwei.hu@pku.edu.cn; ytan@pku.edu.cn).

Digital Object Identifier 10.1109/TCYB.2015.2487318

instances are wrongly classified, while the proposed error rank utilizes the likelihood information of NN classification when calculating the fitness function. Such detailed information makes error rank perform better than error rate.

2) The multiobjective (MO) optimization strategy for prototype generation is proposed to keep the NN classifier from overfitting the training set. The MO optimization refers to optimizing a problem with multiple objective functions. For example Rosales-Pérez *et al.* [6] proposed to minimize the error rate and the number of prototypes simultaneously for prototype generation. The MO optimization strategy proposed in this paper is different from their approach since we focus on the problem of weakening overfitting. In the proposed strategy, the training set is divided into several subsets and the performance metric (e.g., error rate and error rank) is calculated on each subset. Then each subset's performance metric is regarded as an objective function of MO PSO. The resulting prototypes will have better classification performance simultaneously on multiple subsets and the effect of overfitting can be weakened.

The proposed prototype generation algorithm is evaluated on 31 UCI data sets via cross validation. We use the Friedman aligned (FA) ranks test and the Holm's test to compare the proposed algorithm with 16 existing prototype generation algorithms. The experimental results demonstrate that the proposed method is better than these comparison algorithms. In addition, we have run the proposed algorithm on the 59 data sets used in [1]. This paper has compared the experimental performance of 25 existing prototype generation algorithms on the 59 data sets and several other papers have also reported the results of their proposed algorithms on the 59 data sets. Our method achieves higher average accuracy than all of these prototype generation algorithms.

This paper is organized as follows. Section II reviews the related work. A brief introduction to PSO and the framework of applying PSO to generate prototypes are given in Section III. The novel fitness function error rank is introduced in Section IV. Section V proposes the MO optimization strategy to weaken the effect of overfitting. Experimental results are presented in Section VI. Section VII concludes this paper.

II. Related Work

Many prototype generation algorithms have been proposed in recent years. In this section, we categorize the prototype generation algorithms into three classes: 1) the data condensation approach; 2) the prototype tuning approach; and 3) the evolution-based approach.

The data condensation algorithms such as vector quantization are the immediate ways to generate prototypes.

Vector quantization is a classic approach to reduce the size of a set of vectors, which has been used to generate prototypes from the training set. Xie *et al.* [7] quantized the training data for each class and applied the KNN rule to the resulting alphabets.

Self-organizing map (SOM) is an effective vector quantization algorithm [2]. SOM maps each instance in the training set to a neuron which represents a prototype. During the learning phase of SOM, different neurons compete with each other, and the winner neuron together with its neighboring neurons will update their weights. After the learning phase, the weights of each neuron are regarded as the vector of a prototype. Li *et al.* [8] used SOMs to get the initial prototypes and then fine-tuned these prototypes iteratively.

Lozano *et al.* [4] obtained the prototypes from the Gaussian mixture model. For each class, they estimated the probability density of the training data through the mixture of Gaussians. The mean vector of each Gaussian distribution is regarded as a prototype.

Most data condensation-based prototype generation algorithms just regard prototype generation as a general data condensation problem and do not take the characteristics of NN classification into consideration. These algorithms usually perform more poorly than other prototype generation algorithms which introduce some strategies to improve the performance of NN classification.

The prototype tuning approach obtains the prototypes by merging, moving, deleting, or relabeling the training instances. This tuning process usually involves certain heuristic mechanisms to guarantee the classification performance according to the characteristics of the NN rule.

Chang [9] continuously merged the closest training instances of the same class until the accuracy of the NN classifier decreases.

Hamamoto *et al.* [3] adopted the bootstrap technique to merge the training instances. They first selected a training instance and then found several NNs of it. The linear combination of the instance and its neighbors with random weights, or the mean vector of the instance and its neighbors, can be regarded as a prototype. They repeated this progress to obtain multiple prototypes.

Geva and Sitte [10] first selected some training instances from the training set as the initial prototypes and then adjusted their positions to improve the accuracy. When a training instance is misclassified, its nearest prototype will be moved away from it, and its nearest prototype of the same class will be moved toward it.

Koplowitz and Brown [11] used an editing scheme to improve the performance of NN classification. The training instances can be deleted or relabeled according to their consistencies with neighbors.

Decaestecker [12] regarded prototype generation as an optimization problem by defining a cost function whose arguments are the positions of prototypes and then optimizing the cost function. The cost function is based on the difference between the observed and the desired probabilities. Gradient descent and simulated annealing were used to optimize the cost function.

It can be seen that prototype tuning approach generally introduces some human-defined rules to improve the classification performance when tuning the prototypes. For example, Geva and Sitte [10] defined the rule to move the prototype away from the misclassified training instances and toward the

correctly classified ones. They expected this rule would benefit the NN classification. Even though the human-defined rules have a certain correlation with the final classification performance, it is difficult to establish a clear relationship between the rules and the final performance. The rules cannot directly guarantee that the classification performance is optimal.

Evolution-based prototype generation is a kind of performance-driven approach. It directly regards the classification performance as the objective function and uses evolutionary computation to optimize it on the training set.

Fernández and Isasi [13] proposed an evolutionary approach to search for the best prototypes. They encoded the positions and classes of all the prototypes as a single vector which represents an individual of the evolutionary algorithm. The mutation operator, reproduction operator, fight operator (i.e., using information from other prototypes), move operator, and die operator are designed to evolve an optimal set of prototypes.

Cervantes *et al.* [14], [15] encoded each prototype as a particle of PSO, and an NN classifier consists of a set of particles. To obtain a good combination of prototypes, this method allows inserting new particles into the swarm and deleting particles from the swarm. Each particle has a local fitness function which is related to the distances between the particle and instances in the training set.

Nanni and Lumini [16] used a single particle to represent a set of prototypes. They regarded the error rate on the training set as the fitness function. Multiple NN classifiers are generated by repeatedly running PSO and a vote rule is used to combine these classifiers, in order to obtain better classification performance.

Triguero *et al.* [17] proposed a hybrid model which combines the prototype selection method and the position adjustment method. At first, the prototype selection method selects a set of prototypes from the training set. Then the position adjustment method (e.g., differential evolution and PSO) uses this set to initialize its population, while the conventional position adjustment method initializes its population randomly. This initialization strategy reduces the search space to a local region where a potential optimal solution may lie, and therefore it is more probable to obtain an excellent solution.

For big data applications, Triguero *et al.* [18] used MapReduce to accelerate prototype generation. The large data set is partitioned into a number of subsets, and each subset is processed by a computing unit. The computing unit will reduce the subset to a small prototype set by a prototype generation algorithm. Finally, the prototype sets from all of the computing units are concatenated to a single set and some similar prototypes may be merged.

Rosales-Pérez *et al.* [6] and Escalante *et al.* [19] adopted the MO evolutionary algorithm to generate prototypes. Their algorithm has two objective functions to minimize the error rate on the training set and the number of prototypes. They also combined prototype generation with feature selection in another paper [20], by adding a third objective function: the number of selected features.

By directly pursuing the classification performance, evolution-based prototype generation is able to result in better accuracy. The experiment section of this paper presents the accuracies of many prototype generation algorithms. Most of the top algorithms are based on evolutionary computation.

However, evolution-based approach may suffer from overfitting. The fitness function of evolution-based prototype generation is the classification performance on the training set. After the optimization of evolutionary algorithm, the resulting NN classifier may overfitting the training data and cannot generalize well on the unseen test set. In this paper, we will introduce two novel strategies to enhance the generation ability and weaken the effect of overfitting.

Algorithm 1 Particle Swarm Optimization

```
Generate a swarm of particles with positions and velocities initialized at random.
Evaluate the fitness functions of all the particles.
while maximal number of iterations not reached do
    for all particle in the swarm do
        Update the velocity of the particle.
        Update the position of the particle.
        Evaluate the fitness function of the particle.
        Update the particle's best position if needed.
        Update the swarm's best position if needed.
    end for
end while
return the best solution found in the iteration process.
```

III. Particle Swarm Optimization and its Application to Prototype Generation

This section first introduces the framework of PSO and the basic concept of MO PSO and then demonstrates how to use PSO for prototype generation.

A. Framework of PSO

PSO simulates the social behavior of the flying bird flock to search for the optimal solution of a specific problem in multidimensional solution space. In other words, PSO aims at finding the minimal or maximal value of a multivariable function.

PSO maintains a swarm of particles where each particle represents a candidate solution of the problem and particles are able to communicate information with each other to behavior cooperatively. Each particle is associated with a position vector which represents its coordinate in the solution space, and a velocity vector is used to update the position vector. PSO conducts the optimization process through an iterative approach as Algorithm 1.

PSO first generates a swarm of particles randomly. At each iteration, each particle updates its velocity and position according to the following formulas:

$$\mathbf{v} = w\mathbf{v} + c_1 r_1(\mathbf{b} - \mathbf{x}) + c_2 r_2\left(\mathbf{b_g} - \mathbf{x}\right) \tag{1}$$

$$\mathbf{x} = \mathbf{x} + \mathbf{v} \tag{2}$$

where $\mathbf{x}$ and $\mathbf{v}$ represent the current position and velocity of the particle, $\mathbf{b}$ and $\mathbf{b_g}$ represent the best solution found by the

particle and the global swarm so far respectively, w, c_1, and c_2 are hyper-parameters of PSO, and r_1 and r_2 are random numbers between 0 and 1. After updating the particle's position, the fitness function need to be recalculated. If the new position is better than the particle's best solution $\mathbf{b}$, $\mathbf{b}$ needs to be updated, and so does the global best solution $\mathbf{b_g}$.

Formula (1) plays a key role in PSO. Under the influence of the last two items in (1), the particle will move toward the optimal solutions found so far with a certain probability, which is beneficial to the local search of PSO. The last item enables the particles in the whole swarm to move toward the global best solution $\mathbf{b_g}$ with a certain probability, making the swarm to behave socially and collaboratively. The random numbers in (1) gives the particle a certain probability to explore the global solution space. Due to such local, global, and collaboration mechanisms, PSO exhibits strong ability to search the multidimensional solution space, and usually it is able to find a satisfying solution for complex problems.

B. Multiobjective PSO

Single-objective (SO) PSO uses a single fitness function to evaluate a particle. In many applications, there are more than one metric to evaluate a particle and people want to find a particle which is excellent over all of the metrics. For example, a feature selection algorithm usually tries to improve the classification performance and reduce the size of the selected feature set simultaneously. When evaluating a particle, Xue *et al.* [21] defined two metrics to minimize: 1) the number of selected features and 2) the error rate on the training set, while Nag and Pal [22] used three metrics: 1) false positive; 2) false negative; and 3) the size of the selected feature set. Under many circumstances, different metrics may conflict with each other. In the feature selection example if the feature set size is overly reduced, the error rate usually increases.

Multiobjective PSO is developed to deal with the case of multiple metrics. Each metric is called an objective function and the goal of MO PSO is to minimize[1] all the objective functions, as

$$\text{minimize } \boldsymbol{f}(\boldsymbol{x}) = <f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})> \tag{3}$$

where n represents the number of objective functions and $f_i(\boldsymbol{x})$ is the ith objective function of particle $\boldsymbol{x}$, $i = 1, 2, \ldots, n$.

Given two particles $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, if the objective function values of $\boldsymbol{x}_1$ are all not larger than those of $\boldsymbol{x}_2$ and there is at least one objective function under which $\boldsymbol{x}_1$ has a smaller value than $\boldsymbol{x}_2$, we say that $\boldsymbol{x}_1$ dominates $\boldsymbol{x}_2$. Different objective functions usually conflict with each other; decreasing one objective function usually leads to an increment in another objective function. Therefore, there usually does not exist a single particle which is optimal simultaneously over all of the objective functions. The solution of MO optimization is a set of particles called Pareto-optimal set. This set is nondominated; no particle in this set dominates another one. Besides, no particle in the whole solution space can dominate any particle in this set.

[1]An optimization problem may refer to minimizing or maximizing the objective function(s). In this paper, we only take minimization as an example.

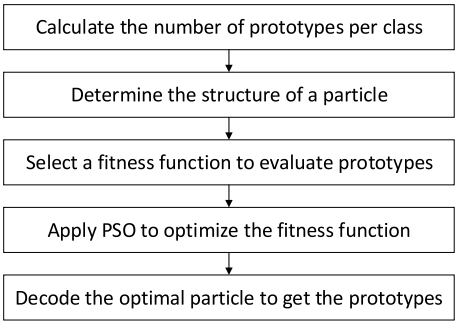

Fig. 1. PSO for prototype generation.

Multiobjective PSO tries to find the Pareto optimal set as exactly as possible. Many approaches have been proposed to implement MO PSO [23]. An external archive is usually used to store the nondominated solutions during the iterative process of PSO. Different researchers have defined different strategies to maintain the archive. These strategies specify when to add a particle in the swarm to the archive, when to remove bad particles from the archive, and how to keep the diversity of the archive.

C. PSO for Prototype Generation

The flowchart of applying PSO to generate prototypes is shown in Fig. 1.

In this paper, we adopt a fixed reduction strategy for prototype generation; we should determine the number of prototypes at first. Given a training set with N instances and C classes, let an instance have D attributes, and the cth class have N_c training instances, $c = 1, 2, \ldots, C$. For a prespecified reduction rate γ, $(1-\gamma)*N$ prototypes will be generated based on the training set. We keep the proportion of the prototype quantities among different classes the same as the proportion of the training instance quantities among different classes. Therefore, the number of prototypes in class c is

$$P_c = \max\{1, (1-\gamma) * N_c\}. \tag{4}$$

We ensure that there is at least one prototype in each class in (4).

Then the structure of a particle should be defined. Assuming the prototypes in the cth class are $\mathbf{p}_{c;1}, \mathbf{p}_{c;2}, \ldots, \mathbf{p}_{c;P_c}$, $\mathbf{p}_{c;j}$ is a D-dimensional vector where each dimension represents an input attribute, $j = 1, 2, \ldots, P_c$. We merge these vectors from the cth class into a single one which is denoted as $\mathbf{p}_c$. The particle of PSO is represented as $<\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_C>$; each particle is a classifier consisting of the prototypes from all the C classes.

Next we need to select a fitness function to evaluate the particle. The prototypes represented by the particle are used to classify the training set according to the NN rule, and the performance metric such as error rate and error rank proposed in this paper is regarded as the fitness function.

After defining the structure of a particle and the fitness function, PSO is able to optimize the prototype generation problem and will produce an optimal particle. At last, the optimal particle is decoded according to its structure to obtain the final prototypes.

In this paper, we use 1NN rule to classify an instance, since the generated prototype set is generally quite small, and 1NN rule is used by most other prototype generation papers.

IV. Error Rank

This section proposes a fitness function named error rank to overcome the shortcoming of the widely used fitness function error rate.

A. Motivation of Error Rank

The error rate on the training set is the commonly used fitness function in evolution-based prototype generation. Minimizing the error rate on the training set is identical to maximizing the accuracy on the training set.

Error rate (i.e., the number of misclassified instances divided by the total number of instances) only considers whether an instance is correctly classified or not, disregarding its likelihood of belonging to a specific class. Assuming that there are two prototypes with classes c_1 and c_2, respectively, and a training instance with class c_1, the distances from the training instance to the two prototypes are dis_1 and dis_2. If $\text{dis}_1 > \text{dis}_2$, the training instance will be misclassified by 1NN rule. Let $\varepsilon = \text{dis}_1 - \text{dis}_2$, then $\varepsilon > 0$. Even through 1NN cannot classify the instance correctly, the smaller ε is, the better generalization ability the classifier has due to the potential pattern hidden in the data. However, error rate cannot reflect this potential information since it only uses the sign of ε and discards its numerical value. If two particles have the same error rate, PSO cannot tell which particle is better in terms of generalization ability.

We propose error rank to overcome the above shortcoming of error rate by taking full advantage of the potential information. Error rank is defined to measure the classification loss of binary classification (i.e., classification between just two classes). Therefore, if the data set contains more than two classes we have to convert multiclass classification to binary classification. The one-against-one method [24] is used to perform multiclass classification. The classes of the data set are organized into pairs and for a C-class problem there will be $C(C-1)/2$ pairs of classes. A binary NN classifier is trained for each pair of classes by just using the data from the two corresponding classes, and there will be $C(C-1)/2$ binary NN classifiers. Each binary classifier votes for a specific class when classifying an instance. The class with the most votes is regarded as the predicted class of the instance.

In binary classification, the classifier need to classify data between classes c_1 and c_2. Given a training instance $\mathbf{y}_i$, its nearest prototype in class c_1 is denoted as $\mathbf{p}_{1;j_1}$, and its nearest prototype in class c_2 is denoted as $\mathbf{p}_{2;j_2}$. We introduce a discriminant value d_i for the instance to evaluate the difference between its distances from the two nearest prototypes

$$d_i = \text{distance}\left(\mathbf{y}_i, \mathbf{p}_{1;j_1}\right) - \text{distance}\left(\mathbf{y}_i, \mathbf{p}_{2;j_2}\right). \tag{5}$$

Algorithm 2 Calculation of Error Rank

Input: N labeled training instances and their discriminant values $d_0, d_1, \ldots, d_{N-1}$.

Output: the fitness function value *er* of error rank.

1: Sort all the instances by their discriminant values, let $d_0 \leq d_1 \leq ... \leq d_{N-1}$.
2: Get the index of the first non-negative value in the ordered sequence, denote it as z.
3: **for** $i = 0 \rightarrow N-1$ **do**
4: $\quad r_i = \begin{cases} (i-z)/z, i < z \\ (i-z)/(N-z), i \geq z \end{cases}$.
5: $\quad r_i = sigmoid(r_i)$.
6: **end for**
7: **return** $er = \sum_{i \in misclassified\ \ instances} abs(r_i)$.

In this paper, we use the Euclidean distance to measure the distance between an instance and a prototype since it is the most used distance metric in prototype generation literatures.

The discriminant value d_i can be used to determine the class of $\mathbf{y}_i$ according to 1NN rule. If $d_i < 0$, $\mathbf{y}_i$ belongs to class c_1, and the smaller d_i is, the more confidently $\mathbf{y}_i$ belongs to class c_1. If $d_i \geq 0$, $\mathbf{y}_i$ belongs to class c_2, and the larger d_i is, the more confidently $\mathbf{y}_i$ belongs to class c_2. The absolute value of the discriminant value indicates the confidence of belonging to a specific class. Minimizing the sum of absolute discriminant values of misclassified instances will result in a better classifier than just minimizing the proportion of misclassified instances, as discriminant values contain more detailed information.

The discriminant values of different instances may differ greatly. In such case, some extremely large discriminant values will dominate the sum of absolute discriminant values of misclassified instances. To eliminate the effect of the uneven distribution of discriminant values, the ranks of discriminant values are used to rescale the discriminant values. The ranks are evenly distributed values and no extreme discriminant values could dominate the sum of misclassified instances.

B. Definition of Error Rank

The proposed error rank minimizes the ranks of misclassified instances to improve the generalization ability, as shown in Algorithm 2.

N training instances are first sorted by their discriminant values in ascending order. Then all the instances in the sequence are numbered from 0 to $N-1$, as shown in Fig. 2. The class of an instance is determined by whether its discriminant value is negative or non-negative, and therefore the first non-negative value in the ordered sequence separates these instances. The index of the first non-negative value is denoted as z. The algorithm calculates a rank value r_i for each instance which indicates its distance from the first non-negative instance. The return value of Algorithm 2 is the sum of absolute rank values of all the misclassified instances, which is regarded as the fitness function value.

1) Rank Distance Normalization: Step 4 of Algorithm 2 uses $(i-z)$ to measure the distance between the ith instance and

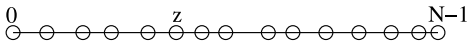

Fig. 2. Ordered sequence of instances.

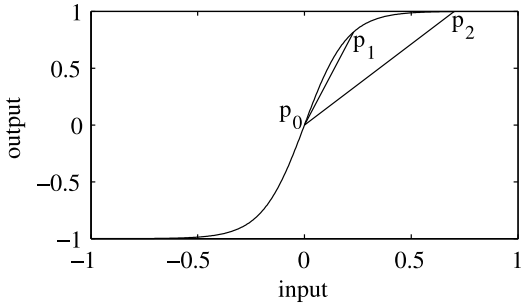

Fig. 3. Sigmoid function to enlarge the input value. p_1 has a larger magnitude of enlargement than p_2 since the absolute value of the input of p_1 is smaller.

the first non-negative instance, $i = 0, 1, \ldots, N-1$. We call this distance the rank distance. If the distribution of instances over the two classes is unbalanced (e.g., in Fig. 2, there are much more non-negative instances than negative instances according to the discriminant values), the average absolute value of rank distances of the minority class is much smaller than that of the majority class. Thus PSO will pay more attention to the majority class and may ignore the minority class. To eliminate the problem caused by imbalance, the rank distance $(i - z)$ is divided by the number of instances in the corresponding class predicted by the discriminant value (i.e., z or $N - z$).

2) Emphasizing Instances With Small Rank Distances: PSO tunes the positions of prototypes to fit the misclassified instances by minimizing their absolute values of rank distances. Generally speaking, fitting the instances with small absolute values of rank distances is more imperative than fitting the instances with large absolute values. A small absolute value of a misclassified instance is not far from zero, and therefore a small reduction will bring it to zero or very near to zero. Instances in the neighborhood of this instance are more likely to be correctly classified. While the same reduction in a large absolute value will not generate a small absolute value, and therefore the performance of the classifier will not improve that much. To fit an instance with a large absolute value, a large reduction is required, while the same reduction would fit more instances with small absolute values. For example decreasing one instance from 1.0 to 0 requires the same reduction as decreasing ten instances from 0.1 to 0. Fitting more misclassified instances will improve the performance on a wider region in the vector space and therefore it will bring a better generalization ability. This is why fitting the instances with small absolute values is more imperative.

PSO minimizes the sum of the rank distances' absolute values of the misclassified instances, but it should pay more attention to the instances with small absolute values. In step 5 of Algorithm 2, a special case of sigmoid function illustrated in Fig. 3 is used to emphasize these instances.

The sigmoid function performs a nonlinear mapping from the input to the output. When we divide the output value of the sigmoid function by its input value, we get the input's magnitude of enlargement. From the curve in Fig. 3, we can see that an input with a small absolute value has a large magnitude of enlargement, while an input with a large absolute value has a small magnitude of enlargement. For example, the magnitude of enlargement of p_1 (i.e., the slope of $\overline{p_0 p_1}$) is larger than that of p_2 (i.e., the slope of $\overline{p_0 p_2}$), as the input's absolute value of p_1 is smaller than that of p_2. In other words, a large weight is assigned to any misclassified instance with a small absolute value when calculating the fitness function, and therefore PSO is able to pay more attention to instances with small absolute values.

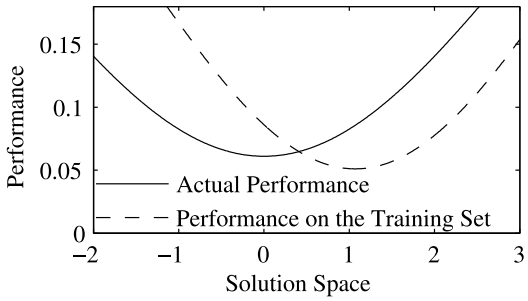

Fig. 4. Actual performance and the performance on the training set of a classifier over the solution space.

Compared with error rate, error rank takes the likelihood of misclassification into consideration by introducing the discriminant values and rescaling them, rather than just judges whether the training instances are misclassified. Under the condition of same error rate, smaller error rank means better generalization ability for a particle (i.e., an NN classifier). In this way, PSO is able to factor the generalization ability into its optimization objective.

V. Multiobjective Optimization Strategy for Learning

A common problem faced by classification is overfitting. A classifier (i.e., a set of prototypes) trained on a fixed training set may perform well on the training set but show a poor performance on the test set if it learns too much about the variation of the training set and misses the training data's underlying nature. Fig. 4 illustrates the difference between the actual performance and the performance on the training set. The horizontal axis represents the parameters of a classifier which constitute the solution space of PSO, i.e., the positions of all the prototypes. The multidimensional parameters are visualized as a 1-D horizontal axis in the figure. The vertical axis represents the performance metric of a classifier such as error rate and error rank, and smaller vertical value means better performance. The parameters under which the training set achieves the optimal performance have a certain distance from the parameters which generate the optimal actual performance.

This section proposes a novel method to deal with overfitting. The whole training set is divided into several subsets and the performance metric is calculated on each subset. This process results in several fitness functions for a single particle.

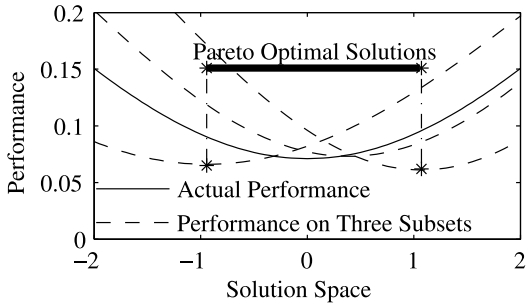

Fig. 5. Actual performance and the performance on three subsets. The Pareto-optimal solutions lie in the horizontal range labeled by the bold line.

Multiobjective PSO is used to optimize these fitness functions simultaneously by regarding each fitness function as an objective function. The MO optimization tries to search for solutions which are excellent on multiple subsets and therefore the classifier will have better generalization ability to weaken the effect of overfitting.

The MO optimization usually finds a set of Pareto-optimal solutions which are nondominated, as shown in Fig. 5. The performance curves of three subsets are around the actual performance curve and the Pareto-optimal solutions lie around the actual optimal solution.

If MO PSO gets more than one Pareto-optimal solution, we just choose the best one to classify a test instance. The Pareto-optimal solution with the minimum average objective function value is regarded as the best one.

Another advantage of the MO optimization strategy is its intrinsic parallelism. Fitness function evaluation is the most time-consuming component of the whole algorithm, especially when the training set is extremely large. However, the calculations of the fitness functions on the multiple subsets are independent. Therefore, such calculations can be distributed to different computing units, in order to accelerate the prototype generation algorithm. The parallel implementation will be very useful when people are dealing with large-scale problems.

VI. Experiments

We ran the proposed algorithm and 16 existing prototype generation algorithms on 31 UCI data sets. This section will make a detailed experimental analysis based on these results. Besides, we give the proposed algorithm's experimental results on the 59 data sets used by Triguero *et al.* [1] at the end of this section, and compare our results with the published results of 28 existing prototype generation algorithms.

A. Experimental Setup

We have used 31 common data sets from the UCI Machine Learning Repository [25] to test the performance of the proposed algorithm. The number of attributes, the number of classes, and the number of instances of each data set are listed in Table I. Some data sets contain missing values. For symbolic attributes, we used the most common values to replace missing values, and for numerical attributes we used the average values [26].

TABLE I
Detailed Information of the 31 Data Sets. #At Represents the Number of Attributes, #Cl Represents the Number of Classes, and #In Represents the Number of Instances

dataset	#At	#Cl	#In	dataset	#At	#Cl	#In
anneal	39	6	898	ionosphere	35	2	351
audiology	70	24	226	iris	5	3	150
autos	26	7	205	kr-vs-kp	37	2	3196
balance-scale	5	3	625	labor	17	2	57
breast-cancer	10	2	286	lymph	19	4	148
breast-w	10	2	699	primary-tumor	18	22	339
colic	23	2	368	segment	20	7	2310
colic.ORIG	28	2	368	sick	30	2	3772
credit-a	16	2	690	sonar	61	2	208
credit-g	21	2	1000	soybean	36	19	683
diabetes	9	2	768	splice	62	3	3190
glass	10	7	214	vehicle	19	4	846
heart-c	14	5	303	vote	17	2	435
heart-h	14	5	294	waveform-5000	41	3	5000
hepatitis	20	2	155	zoo	18	7	101
hypothyroid	30	4	3772				

Tenfold cross validation has been used to obtain the accuracies of the proposed method and the comparison algorithms. Since many prototype generation algorithms are stochastic, we ran such algorithms ten times and took the average accuracies.

The parameters of the proposed algorithm and MO PSO are tuned by experimental study. The 31 data sets cannot be used to tune the parameters because such tuning will result in overfitting. We chose another ten data sets to perform the tuning, and the average size of the ten chosen data sets is similar to that of the 31 UCI data sets.

After the parameter tuning process, the number of subsets was set to two, and therefore there are two objective functions for the MO optimization. The reduction rate was set to 0.99. The MO PSO algorithm OMOPSO [27] implemented in the Java package jMetal [28], [29] was adopted as the MO optimizer. The swarm size and the archive size were both set to 100, and the number of iterations was set to 250. All the input attributes were normalized to the interval $[0, 1]$ before training the NN classifier.

B. Comparison Algorithms

To evaluate the effectiveness of error rank and the MO optimization strategy, the presence and absence of them were both tested. Different configurations of these methods are summarized as follows.

1) *MO/Rank:* Error rank + MO optimization.
2) *MO/Rate:* Error rate + MO optimization.
3) *SO/Rank:* Error rank + SO PSO.
4) *SO/Rate:* Error rate + SO PSO.

MO/rank uses error rank as the fitness function and adopts the MO optimization strategy. MO/rate uses error rate and the MO optimization strategy. These two comparison algorithms are used to evaluate the improvement of error rank. SO/rank and SO/rate calculate the fitness function on the whole training set and just use SO PSO to optimize the classifier. These two comparison algorithms will show the effectiveness of the MO optimization strategy.

We also compared the proposed algorithm with 16 existing prototype generation algorithms. The comparison prototype generation algorithms are AMPSO [14], [15], BTS3 [3], DSM [10], GENN [11], HYB [30], IPLDE [31], LVQPRU [8],

MixtGauss [4], MSE [12], PNN [9], PSCSA [32], PSO [16], RSP3 [33], SGP [34], SSMASFLSDE [17], and VQ [7]. The introduction of these algorithms can be found in Section II.

A java software named KEEL [35], [36] has implemented these prototype generation algorithms. Thus this paper used KEEL to get the performance of these algorithms. For each algorithm, we adopted the parameters used by its original authors. For the parameters that the original authors did not provide, we used the default parameters offered by KEEL, since KEEL's default parameters were carefully tuned [37].

The parameters we choose for different algorithms are listed below. Some algorithms need too many parameters and we just give the important parameters here. The algorithms without key parameters are not listed.

1) *AMPSO:* Swarm size = 5, MaxIter = 300, C1 = 1.0, C2 = 1.0, C3 = 0.5, VMax = 1, Winertia = 0.1, Xfactor = 0.1, ProbR = 0.1, and ProbD = 0.1.
2) *IPLDE:* MaxIter = 1000, iterSFGSS = 8, iterSFHC = 20, ScalingFactor = 0.5, CrossOverRate = 0.9, tau3 = 0.03, and tau4 = 0.07.
3) *DSM:* Iterations = 100, percentage respect training size = 10, and alpha 0 = 0.1.
4) *HYB:* Iterations of search = 200, iterations of optimal search = 1000, percentage prototypes generated = 10, alpha 0 = 0.1, percent of set in training partition = 80, type of initial selection = SVM, and CNN parameter $K = 1$.
5) *LVQPRU:* Number of iterations = 100, percentage of prototypes per class = 10, percentage respect training size = 10, iterations of lvq2_1 = 100, alpha_0 = 0.1, windowWidth = 0.2, and number of neighbors = 2.
6) *MSE:* Number of neighbors = 3, number of initial centroid = 10, gradient step = 0.5, and initial temperature = 100.
7) *PSCSA:* Number of neighbors $k = 1$, hyperMutation rate = 2, clonal rate = 10, mutation rate = 0.008, stimulation threshold = 0.89, and alpha = 0.4.
8) *PSO:* Number of neighbors = 1, swarm size = 20, particle size = 5, MaxIter = 250, C1 = 3, C2 = 1, VMax = 0.1, Wstart = 2.5, and Wend = 1.
9) *RSP3:* Number of subsets = 0 and Subset choice = diameter.
10) *SGP:* Method = 1 and Rmin = 5.
11) *SSMASFLSDE:* Population size = 30, number of SSMA evaluations = 10 000, cross probability per bit = 0.5, mutation probability per bit = 0.001, population size2 = 40, MaxIter = 500, iterSFGSS = 8, iterSFHC = 20, Fl = 0.1, and Fu = 0.9.
12) *VQ:* Iterations = 100, percentage of prototypes generated = 10, alpha 0 = 0.1, and size of neighborhood KNN = 1.

What is more, we used the 1NN without prototype reduction implemented in KEEL as a baseline algorithm.

C. Hypothesis Test

The hypothesis test technique has been used to further analyze the superiority of the proposed algorithm. Since the accuracies of different data sets generally do not follow a parametric distribution, we used the nonparametric test here. The FA ranks test is a popular nonparametric hypothesis test method to compare multiple algorithms, and the Holm's test performs $1 \times N$ comparisons as a *post-hoc* procedure of the FA ranks test, which gives the adjusted p-values (Holm APVs) [38]. The ranking calculated by FA test reflects the relative performance of a classifier among all the classifiers. We ran each algorithm ten times over the 31 data sets, and therefore the hypothesis test was conducted over 310 samples. We performed the hypothesis test on the accuracies of the classifiers. The Holm's test was conducted with a level of significance of $\alpha = 0.05$ and took MO/rank as the control method.

TABLE II
AVERAGE ACCURACY (IN PERCENTAGE, ALONG WITH THE STANDARD DEVIATION IN THE BRACKETS), FA RANKING, HOLM APV, REDUCTION RATE (IN PERCENTAGE), AND TRAINING TIME (IN SECONDS) OF DIFFERENT ALGORITHMS

Algorithm	Accuracy	FA Ranking	Holm APV	Reduction	Time
MO/Rank	83.39(3.06)	1415.0	-	99.0	109.21
MO/Rate	81.95(3.81)	1827.1	6.34E-03	99.0	80.82
SSMASFLSDE	81.41(3.71)	1863.0	6.01E-03	95.9	116.16
SO/Rank	81.11(3.87)	2160.7	2.35E-06	99.0	72.73
GENN	81.09(2.42)	2171.6	2.16E-06	14.1	1.05
SO/Rate	80.35(4.14)	2403.4	2.93E-10	99.0	58.37
PSO	79.77(3.91)	2511.9	2.22E-12	94.4	21.07
RSP3	80.05(0.00)	2517.4	1.98E-12	75.7	9.40
MSE	79.30(3.68)	2659.5	1.34E-15	90.6	1.58
MixtGauss	78.13(1.89)	3062.8	8.77E-27	95.4	12.77
1NN	76.87(0.00)	3097.0	7.86E-28	0.0	0.00
HYB	78.17(2.28)	3131.9	6.24E-29	51.5	5.93
LVQPRU	77.42(4.58)	3660.0	6.06E-49	90.3	0.39
IPLDE	74.97(4.72)	3670.3	2.36E-49	97.7	3.12
PNN	76.54(4.54)	3795.0	7.48E-55	92.0	492.88
DSM	76.08(5.35)	4201.3	6.81E-75	89.6	0.02
BTS3	75.45(4.99)	4266.7	2.19E-78	90.1	0.17
VQ	74.23(5.53)	4695.6	1.75E-103	89.6	0.21
PSCSA	69.10(6.30)	4948.7	6.37E-120	98.2	6.67
AMPSO	70.44(7.12)	5097.6	3.70E-130	92.2	204.34
SGP	69.11(0.00)	5209.0	4.39E-138	94.9	0.11

D. Experimental Results

Table II gives each algorithm's average accuracy, FA Ranking, Holm APV, reduction rate, and the average training time over each fold of data set. Since many algorithms involve randomness, executing them repeatedly on the same data set will result in different accuracies. Therefore for each data set we calculated, the standard deviation of the accuracy over the ten times of repeated execution. The standard deviation in Table II is the average standard deviation over the 31 data sets. The algorithms without involving randomness such as 1NN have zero standard deviation.

The algorithms in this table are sorted by FA Ranking in ascending order. The smaller FA Ranking is, the better overall performance the corresponding algorithm has.

We can see that the proposed algorithm MO/rank achieves the best performance. Holm APVs show that MO/rank significantly outperforms all of the other prototype generation algorithms.

We will analyze the results in detail in the following sections.

1) Experimental Analysis of Error Rank: In this section, we analyze the difference between error rank and error rate.

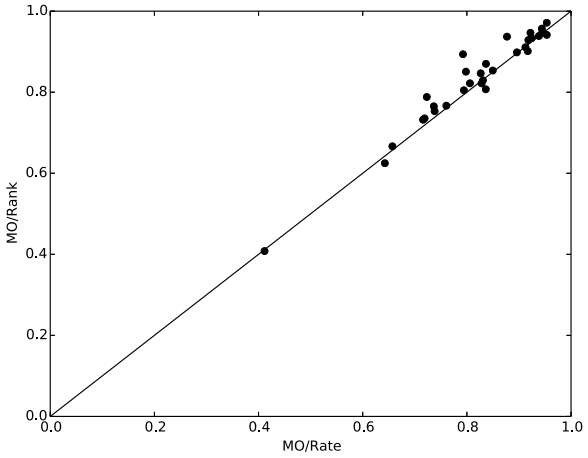

Fig. 6. Accuracy of MO/rank versus MO/rate over each data set.

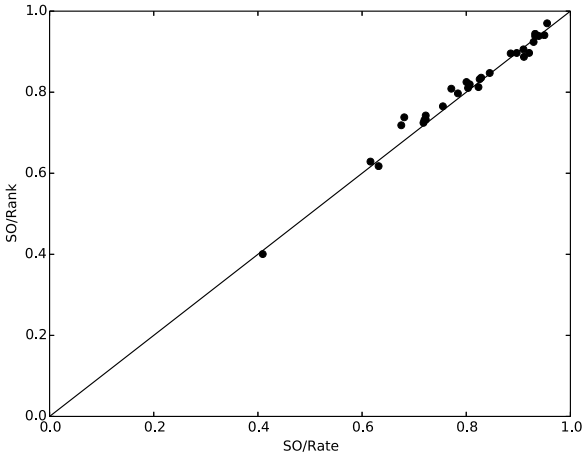

Fig. 7. Accuracy of SO/rank versus SO/rate over each data set.

When using the MO optimization strategy, the average accuracy is 83.39% for error rank, while error rate only achieves 81.95%. Error rank and error rate have the average accuracies of 81.11% and 80.35% under the SO PSO (i.e., SO/rank and SO/rate). These results indicate that error rank exhibits better classification performance than error rate under different circumstances.

We plot the accuracy of MO/rank and MO/rate over each data set in Fig. 6. The vertical axis represents the accuracy of MO/rank, while the horizontal axis represents the accuracy of MO/rate. Each dot in the figure represents one of the 31 data sets used in this paper. A diagonal from (0, 0) to (1, 1) is plotted in the figure. If a dot falls above the diagonal, MO/rank gets higher accuracy than MO/rate over the corresponding data set, and vice versa.

Most of the dots in Fig. 6 fall above the diagonal, while the other dots fall near the diagonal. No dots fall far below the diagonal. We have checked the numerical values for the 31 data sets. On 23 data sets does error rank achieve better accuracies than error rate. For the "splice" data set, the accuracy of error rank exceeds error rate by 10%. There are five data sets whose accuracies of error rank exceed error rate by at least 3%. In the worst case for error rank, its accuracy falls behind error rate by 2.9% on the data set "lymph." On three other data sets, it falls behinds error rate by about 1%.

There is a point near (0.4, 0.4) which stays far away from other points. This data set is "primary tumor," which has 22 classes and 339 instances. Due to the small size and the large number of classes, both algorithms cannot achieve high accuracies.

Fig. 7 gives the comparison between error rank and error rate under the SO optimization strategy. Most of the data sets get better results when using error rank. Just a few data sets get worse results, but the gaps between error rank and error rate are very small (i.e., at most 2%) on these data sets. Compared with Fig. 6, the superiority of error rank under the MO optimization strategy is more remarkable than that under the SO optimization strategy.

From the above analysis, we can conclude that using error rank as the fitness function shows better overall performance than using error rate.

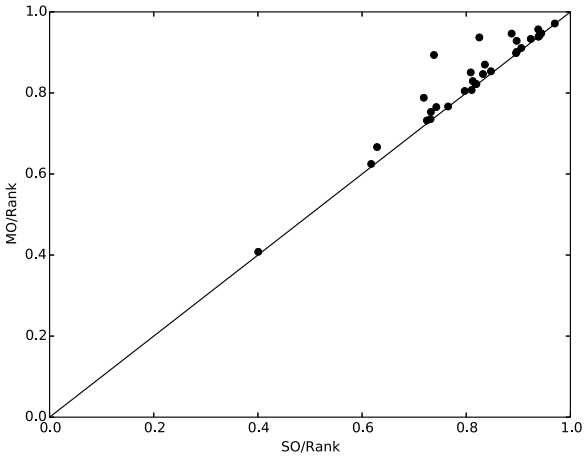

Fig. 8. Accuracy of MO/rank versus SO/rank over each data set.

2) Experimental Analysis of Multiobjective Optimization Strategy: Table II shows that when using error rank as the fitness function, the average accuracy of the MO optimization strategy (i.e., MO/rank) achieves 83.39%, while the SO optimization strategy (i.e., SO/rank) only reaches 81.11%. In respect of error rate, MO/rate obtains an average accuracy of 81.95%, and SO/rate's average accuracy is 80.35%. Therefore, the MO optimization strategy has higher average accuracy than the SO optimization strategy under both fitness functions.

The accuracy of the MO optimization strategy and the SO optimization strategy over each data set when using error rank as the fitness function is shown in Fig. 8. The MO optimization strategy outperforms the SO optimization strategy over most of the 31 data sets. For the data sets splice and "kr-vs-kp," the MO optimization strategy's accuracies exceed the SO one by 15.6% and 11.2%, respectively. Over no data sets does the MO optimization strategy fall behind the SO optimization strategy.

Fig. 9 illustrates the 91 Pareto-optimal solutions obtained by MO/rank for the "sick" data set in a particular fold of the ten-fold cross validation. The Pareto-optimal solutions distribute along a line with a negative slope, which implies the conflict of the two objective functions; decreasing error rank on one subset will cause an increment in the other one.

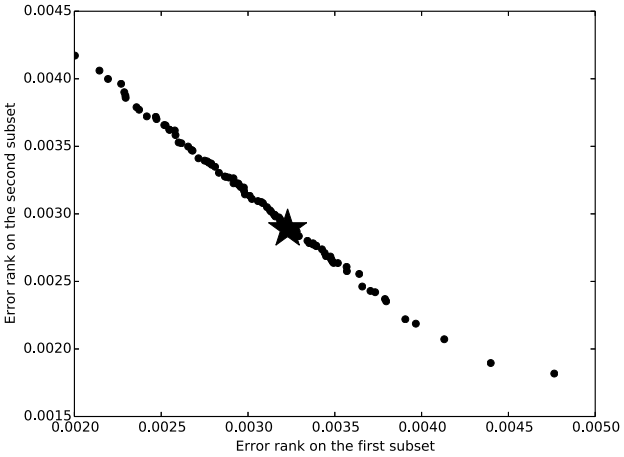

Fig. 9. Pareto-optimal solutions obtained by MO/rank for the sick data set. The star represents the solution with the minimum average error rank over the two subsets.

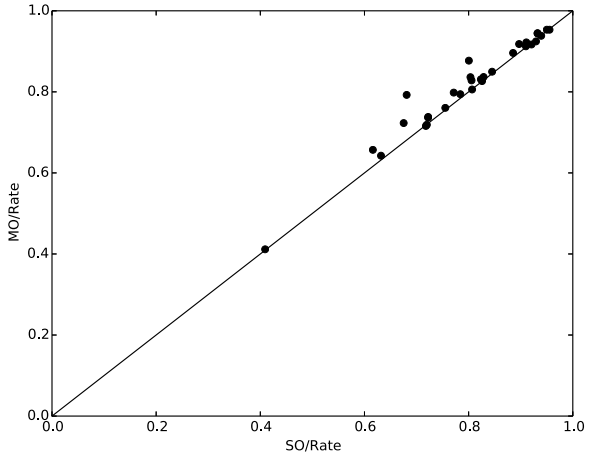

Fig. 10. Accuracy of MO/rate versus SO/rate over each data set.

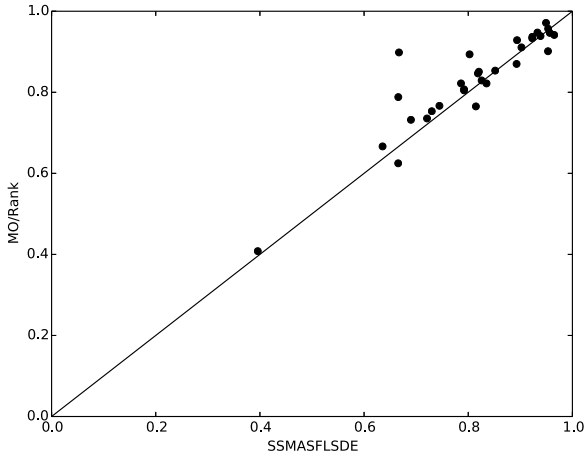

Fig. 11. Accuracy of MO/rank versus SSMASFLSDE over each data set.

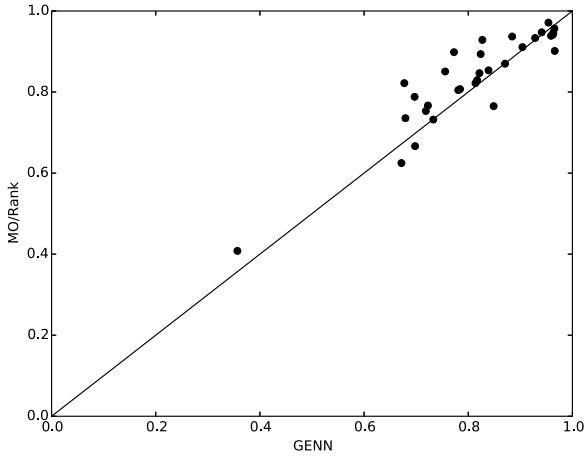

Fig. 12. Accuracy of MO/rank versus GENN over each data set.

The star in Fig. 9 is the solution with the minimum average error rank over the two subsets, which is chosen as the final solution to classify test instances in this paper. This solution lies in the middle of the line and is not biased toward a particular subset. Such characteristic is very beneficial to the final classification.

Fig. 10 gives the comparison under the fitness function error rate. Once again, the MO optimization strategy works better on most of the data sets.

We can see that in most cases, the MO optimization strategy outperforms the SO optimization strategy, which means pursuing the performance on several subsets is better than just on the whole training set. The MO optimization strategy is a useful way to reduce the effect of overfitting.

3) Experimental Comparison With Existing Prototype Generation Algorithms: This section compares the proposed algorithm with 16 existing prototype generation algorithms.

Table II gives the average accuracy of each algorithm and the results of hypothesis test. The average accuracy of MO/rank is 83.39%, while the highest average accuracy among the 16 existing prototype generation algorithms is 81.41%, which is achieved by SSMASFLSDE. MO/rank has the smallest FA ranking, and the Holm APVs of the 16 existing prototype generation algorithms are all not larger than 0.05. This means MO/rank significantly outperforms these algorithms with a level of significance of $\alpha = 0.05$. The standard deviation of MO/rank is relatively low among all of these algorithms, which means it is a relatively stable algorithm.

The best two algorithms among the 16 existing prototype generation algorithms are SSMASFLSDE and GENN. Figs. 11 and 12 compare MO/rank with these two algorithms over each data set, respectively.

The two figures give similar results. When compared with both SSMASFLSDE and GENN, MO/rank gets higher accuracies on most of the data sets, while on just a few data sets does it get a little lower accuracies. There are 23 and 21 data sets on which MO/rank functions better than SSMASFLSDE and GENN, respectively.

In Fig. 11, there is a data set on which MO/rank functions much better than SSMASFLSDE. The data set is "labor," which only contains 57 instances. On this data set, the average accuracies of SSMASFLSDE and MO/rank over the ten runs are 66.7% and 89.8%, respectively. Such poor performance of SSMASFLSDE is caused by overfitting, since this data set is quite small. On the training set, SSMASFLSDE's average

accuracy over the ten runs is 99.7%, and MO/rank has an average accuracy of 98.8% on the training set. The optimization objective of SSMASFLSDE is to directly improve the accuracy on the training set, while MO/rank uses the fitness function error rank and the MO optimization strategy to improve the generalization ability and weaken the effect of overfitting. MO/rank does not perform as well as SSMASFLSDE on the training set, but it significantly outperforms SSMASFLSDE on the test set.

It can be seen that the combination of error rank and the MO optimization strategy yields an excellent prototype generation algorithm which exhibits great potential in real-world applications.

4) Training Time Consumption: The last column of Table II gives each algorithm's average training time over each fold of data set.

Evolutionary algorithm-based approaches such as our methods and SSMASFLSDE consume much more time than nonevolutionary approaches such as GENN and MSE. Evolutionary algorithm needs tens of thousands of fitness function evaluations. That is, the calculation over the training set will be made for tens of thousands of times, resulting in the low-time efficiency.

However, the huge time consumption of evolutionary prototype generation algorithms will pay off. Many evolutionary prototype generation algorithms work better than nonevolutionary ones. The top four algorithms in Table II are all based on evolutionary algorithms.

Among evolutionary prototype generation algorithms, MO/Rank has moderate time consumption. SSMASFLSDE which takes second place in Table II costs a bit more time than MO/Rank.

The fitness function error rank has higher time consumption than error rate under both optimization strategies. The fitness function error rank needs to sort all the instances and do some mathematical manipulation for each instance, which consumes certain time.

The MO optimization strategy also consumes a bit more time than the SO optimization strategy as it needs to randomly partition the training set, while this procedure requires many random numbers whose generation process is time-consuming. Besides, MO PSO spends additional time on maintaining its external archive.

E. Comparison With 28 Prototype Generation Algorithms on 59 Data Sets Offered by Triguero et al.

Triguero *et al.* [1] did an experimental study on prototype generation and gave 25 algorithms' results on 59 data sets. Rosales-Pérez *et al.* [6], [20] and Escalante *et al.* [19] also used the 59 data sets to validate their three proposed MO prototype generation algorithms (i.e., EMOPG, EMOPG+FS, and MOGP).

Compared with the UCI data sets used above, there are some larger data sets in the 59 data sets. The 59 data sets are categorized into two parts by Triguero *et al.*; data sets with less than 2000 instances are categorized into the small data sets, and data sets with more than 2000 instances are categorized into the large data sets. The detailed information can be found in their paper. Some prototype generation algorithms are extremely time-consuming and they only gave the results on the small data sets for these algorithms.

TABLE III
AVERAGE ACCURACY AND REDUCTION RATE (IN PERCENTAGE) OF DIFFERENT ALGORITHMS ON THE SMALL DATA SETS. THE DATA EXCEPT MO/RANK ARE CITED FROM [1], [6], [19], AND [20]

Algorithm	Accuracy	Reduction	Algorithm	Accuracy	Reduction
MO/Rank	78.35	98.5	LVQPRU	69.97	95.0
GENN	75.64	18.6	LVQTC	69.81	95.5
ICPL2	75.60	83.7	SGP	69.49	95.1
PSO	75.01	94.9	MixtGauss	69.32	95.5
MOGP	74.81	98.3	AMPSO	69.03	94.3
EMOPG+FS	74.26	97.3	DSM	68.10	94.9
GMCA	73.51	69.8	PNN	67.86	94.5
1NN	73.26	0.0	Chen	67.70	95.2
RSP3	73.25	73.3	LVQ3	67.63	94.9
Depur	72.96	35.3	PSCSA	66.82	98.6
MSE	72.37	95.2	AVQ	66.72	97.6
MCA	72.19	85.7	BTS3	66.26	95.2
EMOPG	71.73	98.1	VQ	65.49	94.9
ENPC	71.67	72.2	POC	64.93	60.7
HYB	71.53	42.8			

The results they gave only come from a single run of the prototype generation algorithms. Many prototype generation algorithms involve randomness and different runs of an algorithm on the same data may result in quite different results. Table II shows that on average a data set's accuracy has a standard deviation of about 2%–7% over the ten repeated runs. Therefore, the accuracy of a data set from a single run is not stable.

However, we found that the average accuracy over many data sets remains stable across different runs. Based on the previous experiments on the 31 UCI data sets, we analyzed the average accuracy to check its stability. For each algorithm which involves randomness, we averaged the accuracies of the 31 data sets from each run and obtained ten average accuracies from the ten times of running. Then the standard deviation of the ten average accuracies was calculated for the algorithm. It is found that different algorithms' standard deviations range from 0.2% to 0.5%; the standard deviations are quite small.

Therefore, the average accuracy over a number of data sets is a stable performance metric for single-run experiments, and it is reliable to use this metric here.

The average size of the small data sets is smaller than that of the 31 UCI data sets used above, while the average size of the large data sets is larger than it. Therefore, we empirically changed the reduction rate to 98.5% and 99.5% for the small and the large data sets, respectively.

Tables III and IV give the average accuracy and reduction rate of MO/rank on the small data sets and the large data sets, respectively, along with the results of 28 other prototype generation algorithms from [1], [6], [19], and [20].

From the two tables, we can see that MO/rank achieves the best average accuracy on both the small and the large data sets. The average accuracy of MO/rank is higher than that of the second place algorithm by about 2.7% and 2.4% for the two types of data sets, respectively. These results have validated the experimental superiority of the proposed algorithm again.

TABLE IV
AVERAGE ACCURACY AND REDUCTION RATE (IN PERCENTAGE) OF DIFFERENT ALGORITHMS ON THE LARGE DATA SETS. THE DATA EXCEPT MO/RANK ARE CITED FROM [1], [6], [19], AND [20]

Algorithm	Accuracy	Reduction	Algorithm	Accuracy	Reduction
MO/Rank	84.20	99.5	RSP3	75.56	81.0
EMOPG+FS	81.82	98.4	AMPSO	74.10	98.0
EMOPG	81.34	99.5	BTS3	73.99	98.0
GENN	81.33	15.8	LVQPRU	73.56	98.0
MOGP	81.33	99.4	DSM	73.41	98.0
1NN	80.60	0.0	LVQ3	73.18	98.0
ENPC	80.29	82.1	Mixtgauss	73.18	98.0
Depur	80.04	27.1	VQ	73.16	98.0
PSO	80.00	98.0	LVQTC	70.56	99.8
MSE	76.74	99.4	PSCSA	67.07	99.9
Chen	76.21	98.0	AVQ	65.18	99.8
HYB	76.18	57.3	SGP	60.86	98.2

The three MO prototype generation algorithms EMOPG, EMOPG+FS, and MOGP perform ordinarily on the small data sets, but they achieve high average accuracies on the large data sets. However, they still cannot outperform MO/rank since they just focus on the tradeoff between accuracy and reduction rate, while MO/rank pays more attention to improving the generalization ability and overcoming overfitting.

VII. CONCLUSION

The traditional NN algorithm has a high time and space complexity in the classification phase, and its performance is often influenced by noisy training data. Using a small set of prototypes generated from the large training set will overcome these shortcomings. Evolutionary computation is a well-known approach to generate prototypes, and many algorithms adopting this approach work better than other kinds of prototype generation algorithms. This paper applies PSO to generate prototypes and proposes error rank and the MO optimization strategy to further improve the classification performance.

Error rank utilizes the ranks of misclassified instances to calculate the fitness function, while the traditional fitness function error rank just considers the proportion of misclassified instances. The ranks provide more potential information about the pattern of the data, which is helpful to improve the classifier's generalization ability.

The MO optimization strategy splits the training set into several subsets, and it optimizes the classification performance simultaneously on the multiple subsets. Traditional evolution-based prototype generation algorithms pursues the classification performance on the single training set, and this process may lead to severe overfitting. The MO optimization strategy is able to weaken the effect of overfitting since it aims to perform well on multiple sets.

Experiments conducted on 31 UCI data sets have shown the effectiveness of the proposed algorithm. Error rank functions better than error rate and the MO optimization strategy outperforms the traditional SO optimization strategy. On the whole, the combination of error rank and the MO optimization strategy outperforms 16 existing prototype generation algorithms. We also compared the proposed algorithm with 28 algorithms on 59 additional data sets. The proposed algorithm achieves higher average accuracy than the other comparison algorithms.

REFERENCES

[1] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 1, pp. 86–100, Jan. 2012.

[2] T. Kohonen, "The self-organizative map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.

[3] Y. Hamamoto, S. Uchimura, and S. Tomita, "A bootstrap technique for nearest neighbor classifier design," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 1, pp. 73–79, Jan. 1997.

[4] M. Lozano *et al.*, "Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces," *Pattern Recognit.*, vol. 39, no. 10, pp. 1827–1838, 2006.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. Perth, WA, Australia, 1995, pp. 1942–1948.

[6] A. Rosales-Pérez, H. J. Escalante, C. A. C. Coello, J. A. Gonzalez, and C. A. Reyes-Garcia, "An evolutionary multi-objective approach for prototype generation," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 1100–1107.

[7] Q. Xie, C. A. Laszlo, and R. K. Ward, "Vector quantization technique for nonparametric classifier design," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 12, pp. 1326–1330, Dec. 1993.

[8] J. Li, M. Manry, C. Yu, and D. R. Wilson, "Prototype classifier design with pruning," *Int. J. Artif. Intell. Tools*, vol. 14, nos. 1–2, pp. 261–280, 2005.

[9] C.-L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. Comput.*, vol. 23, no. 11, pp. 1179–1184, Nov. 1974.

[10] S. Geva and J. Sitte, "Adaptive nearest neighbor pattern classifier," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 318–322, Mar. 1991.

[11] J. Koplowitz and T. Brown, "On the relation of performance to editing in nearest neighbor rules," *Pattern Recognit.*, vol. 13, no. 3, pp. 251–255, 1981.

[12] C. Decaestecker, "Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing," *Pattern Recognit.*, vol. 30, no. 2, pp. 281–288, 1997.

[13] F. Fernández and P. Isasi, "Evolutionary design of nearest prototype classifiers," *J. Heuristics*, vol. 10, no. 4, pp. 431–454, 2004.

[14] A. Cervantes, I. M. Galvan, and P. Isasi, "An adaptive Michigan approach PSO for nearest prototype classification," in *Nature Inspired Problem-Solving Methods in Knowledge Engineering* (LNCS 4528), J. Mira and J. A. lvarez, Eds. Berlin, Germany: Springer, 2007.

[15] A. Cervantes, I. M. Galván, and P. Isasi, "AMPSO: A new particle swarm method for nearest neighborhood classification," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 39, no. 5, pp. 1082–1091, Oct. 2009.

[16] L. Nanni and A. Lumini, "Particle swarm optimization for prototype reduction," *Neurocomputing*, vol. 72, nos. 4–6, pp. 1092–1097, 2009.

[17] I. Triguero, S. García, and F. Herrera, "Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification," *Pattern Recognit.*, vol. 44, no. 4, pp. 901–916, 2011.

[18] I. Triguero, D. Peralta, J. Bacardit, S. Garcia, and F. Herrera, "A combined MapReduce-windowing two-level parallel scheme for evolutionary prototype generation," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 3036–3043.

[19] H. J. Escalante *et al.*, "MOPG: A multi-objective evolutionary algorithm for prototype generation," *Pattern Anal. Appl.*, pp. 1–15, Feb. 2015.

[20] A. Rosales-Pérez, J. A. Gonzalez, C. A. Coello-Coello, C. A. Reyes-Garcia, and H. J. Escalante, "Evolutionary multi-objective approach for prototype generation and feature selection," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Berlin, Germany: Springer, 2014, pp. 424–431.

[21] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 1656–1671, Dec. 2013.

[22] K. Nag and N. R. Pal, "A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification," *IEEE Trans. Cybern.*, to be published.

[23] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *Int. J. Comput. Intell. Res.*, vol. 2, no. 3, pp. 287–308, 2006.

[24] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002.

[25] K. Bache and M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[26] J. Grzymala-Busse, L. Goodwin, W. Grzymala-Busse, and X. Zheng, "Handling missing attribute values in preterm birth data sets," in *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing* (LNCS 3642), D. Ślęzak, J. Yao, J. Peters, W. Ziarko, and X. Hu, Eds. Berlin, Germany: Springer, 2005, pp. 342–351.
[27] M. R. Sierra and C. A. C. Coello, "Improving PSO-based multi-objective optimization using crowding, mutation and ϵ-dominance," in *Evolutionary Multi-Criterion Optimization* (LNCS 3410), C. C. Coello, A. H. Aguirre, and E. Zitzler, Eds. Berlin, Germany: Springer, 2005, pp. 505–519.
[28] J. J. Durillo, A. J. Nebro, and E. Alba, "The jMetal framework for multi-objective optimization: Design and architecture," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Barcelona, Spain, 2010, pp. 1–8.
[29] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 760–771, 2011.
[30] S.-W. Kim and A. J. Oomenn, "A brief taxonomy and ranking of creative prototype reduction schemes," *Pattern Anal. Appl.*, vol. 6, no. 3, pp. 232–244, 2003.
[31] I. Triguero, S. García, and F. Herrera, "IPADE: Iterative prototype adjustment for nearest neighbor classification," *IEEE Trans. Neural Netw.*, vol. 21, no. 12, pp. 1984–1990, Dec. 2010.
[32] U. Garain, "Prototype reduction using an artificial immune model," *Pattern Anal. Appl.*, vol. 11, nos. 3–4, pp. 353–363, 2008.
[33] J. S. Sanchez, "High training set size reduction by space partitioning and prototype abstraction," *Pattern Recognit.*, vol. 37, no. 7, pp. 1561–1564, 2004.
[34] H. A. Fayed, S. R. Hashem, and A. F. Atiya, "Self-generating prototypes for pattern classification," *Pattern Recognit.*, vol. 40, no. 5, pp. 1498–1509, 2007.
[35] J. Alcalá-Fdez, "KEEL: A software tool to assess evolutionary algorithms for data mining problems," *Soft Comput.*, vol. 13, no. 3, pp. 307–318, 2009.
[36] J. Alcalá *et al.*, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Mult. Valued Log. Soft Comput.*, vol. 17, nos. 2–3, pp. 255–287, 2011.
[37] J. Derrac, J. Luengo, J. Alcala-Fdez, A. Fernandez, and S. Garcia, "Using KEEL software as a educational tool: A case of study teaching data mining," in *Proc. 7th Int. Conf. Next Gener. Web Serv. Pract. (NWeSP)*, Salamanca, Spain, 2011, pp. 464–469.
[38] J. Derraca, S. Garcíab, D. Molinac, and F. Herreraa, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, 2011.

Weiwei Hu received the Bachelor of Science degree from the Beijing Institute of Technology, Beijing, China, in 2012. He is currently pursuing the Ph.D. degree with the Key Laboratory of Machine Perception (Ministry of Education), and the Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing, both in computer science.

His current research interests include machine learning, neural networks, and computer information security.

Ying Tan (SM'XX) is a Full Professor and a Ph.D. Advisor with the School of Electronics Engineering and Computer Science, Peking University (PKU), Beijing, China, and the Director of Computational Intelligence Laboratory with PKU. He is the Inventor of fireworks algorithm. His current research interests include computational intelligence, swarm intelligence, machine learning and data mining, and their applications to information security.

Dr. Tan serves as the Editor-in-Chief of the *International Journal of Computational Intelligence and Pattern Recognition* and the Associate Editor of the IEEE TRANSACTION ON CYBERNETICS, the IEEE TRANSACTION ON NEURAL NETWORKS AND LEARNING SYSTEMS, and the *International Journal of Swarm Intelligence Research*. He served as an Editor of Springer's Lecture Notes on Computer Science for over 13 volumes and a Guest Editor of several referred journals, including *Information Science*, *Softcomputing*, *Neurocomputing*, *Natural Computing*, and the IEEE/ACM TRANSACTIONS ON BIOINFORMATICS AND COMPUTATIONAL BIOLOGY. He is the Founding General Chair of the series International Conference on Swarm Intelligence from 2010 to 2015 and the Joint General Chair of the first and second BRICS CCI, and the Program Committee Co-Chair of the IEEE WCCI in 2014.