Multi-objective pattern and feature selection by a genetic algorithm

Hisao Ishibuchi

Dept. of Industrial Engineering
Osaka Prefecture University
1-1 Gakuen-cho, Sakai, Osaka 599-8531, JAPAN
E-mail: hisaoi@ie.osakafu-u.ac.jp
Phone: +81-722-54-9350

Tomoharu Nakashima

Dept. of Industrial Engineering
Osaka Prefecture University
1-1 Gakuen-cho, Sakai, Osaka, 599-8531, JAPAN
E-mail: nakashi@ie.osakafu-u.ac.jp
Phone: +81-722-54-9351

Abstract

This paper discusses a genetic-algorithm-based approach for selecting a small number of representative instances from a given data set in a pattern classification problem. The genetic algorithm also selects a small number of significant features. That is, instances and features are simultaneously selected for finding a compact data set. The selected instances and features are used as a reference set in a nearest neighbor classifier. Our goal is to improve the classification performance (i.e., generalization ability) of our nearest neighbor classifier by searching for an appropriate reference set. In this paper, we first describe the implementation of our genetic algorithm for instance and feature selection. Next we discuss the definition of a fitness function in our genetic algorithm. Then we examine the classification performance of nearest neighbor classifiers designed by our approach through computer simulations on artificial data sets and real-world data sets.

1. INTRODUCTION

Genetic algorithms (Holland, 1975) have been successfully applied to various problems (Goldberg, 1989). Genetic algorithms can be viewed as a general-purpose optimization technique in discrete search spaces. They are suitable for complex problems with multi-modal objective functions. Their application to instance selection was proposed by (Kuncheva, 1995) for designing nearest neighbor classifiers. In her approach, the classification performance of selected instances was maximized by a genetic algorithm. A penalty term with respect to the number of selected instances was added to the fitness function of her genetic algorithm in (Kuncheva, 1997) for maximizing the classification ability and minimizing the size of nearest neighbor classifiers. In the design of nearest neighbor classifiers, genetic algorithms were also used for selecting features in (Siedlecki and Sklansky, 1989) and finding an appropriate weight of each feature in (Kelly, Jr. and Davis, 1991, and Punch et al., 1993).

In the nearest neighbor classification (Cover and Hart, 1967), each new instance is classified by its nearest neighbor in a reference set. Usually all the given instances are used as the reference set for classifying new instances. For decreasing the number of instances in the reference set and improving its classification performance, various approaches to instance selection have been proposed (for example, see Hart, 1968, Dasarathy, 1994, and Chaudhuri et al., 1994). Some of those approaches intended to find the minimum reference set that can correctly classify all the given instances. The main advantage of the genetic-algorithm-based approach in (Kuncheva, 1997) is its flexibility in the handling of the tradeoff between the classification ability and the size of reference sets. The tradeoff is handled by weight values with respect to these two objectives in the fitness function. This means that the genetic algorithm does not always search for the reference set that can correctly classify all the given instances. Much smaller reference sets with slightly inferior classification ability can be found if the weight value for the size of reference sets is large.

In our former work (Ishibuchi and Nakashima, 1999, 2000), we proposed a GA-based approach to the design of compact reference sets with high classification ability by instance and feature selection. Our approach used several ideas such as instance selection (Kuncheva, 1997), feature selection (Siedlecki and Sklansky, 1989), and biased mutation probabilities (Ishibuchi et al., 1997). In this paper, we examine two definitions of a fitness function in our genetic algorithm for instance and feature selection. Our fitness function is basically defined by the classification performance of a reference set, the number of selected instances, and the number of selected features. One definition, which was used in our former work, is based on the classification results on the given instances by a reference set. This definition is to find compact reference sets that can correctly classify almost all the given instances. In the other definition, the classification of each instance is performed by a reference set excluding

that instance (as in the leaving-one-out procedure). That is, each instance in the reference set is not selected as its own nearest neighbor in the calculation of the fitness function. This definition of the fitness function is to find compact reference sets with high generalization ability. The same idea as the second definition has been used in some instance selection methods (Wilson, 1972 and Kuncheva, 1995, 1997).

2. GENETIC ALGORITHMS

2.1 CODING

Let us assume that m labeled instances $\boldsymbol{x}_p = (x_{p1}, ..., x_{pn})$, $p = 1, 2, ..., m$ are given from c classes in an n-dimensional pattern space where x_{pi} is the value of the i-th feature in the p-th instance. Our task is to select a small number of representative instances together with a few significant features for designing a compact nearest neighbor classifier with high classification ability. Let P_{ALL} be the set of the given m instances: $P_{\mathrm{ALL}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m\}$. We also denote the set of the given n features as $F_{\mathrm{ALL}} = \{f_1, f_2, ..., f_n\}$ where f_i is the label of the i-th feature. Let F and P be the set of selected features and the set of selected instances, respectively, where $F \subseteq F_{\mathrm{ALL}}$ and $P \subseteq P_{\mathrm{ALL}}$. We denote the reference set as $S = (F, P)$.

For handling our instance and feature selection problem by genetic algorithms, every reference set $S = (F, P)$ is coded as a binary string of the length $(n + m)$ as

$$S = a_1 a_2 \cdots a_n s_1 s_2 \cdots s_m , \tag{1}$$

where a_i denotes the inclusion ($a_i = 1$) or the exclusion ($a_i = 0$) of the i-th feature f_i, and s_p denotes the inclusion ($s_p = 1$) or the exclusion ($s_p = 0$) of the p-th instance $\boldsymbol{x}_p$. The feature set F and the instance set P are obtained by decoding the string S as follows:

$$F = \{ f_i \mid a_i = 1, i = 1, 2, ..., n\} , \tag{2}$$

$$P = \{\boldsymbol{x}_p \mid s_p = 1, p = 1, 2, ..., m\} . \tag{3}$$

2.2 FITNESS FUNCTION

In our nearest neighbor classification with the reference set $S = (F, P)$, the nearest neighbor $\boldsymbol{x}_{\hat{p}}$ of a new instance $\mathbf{x}$ is found from the instance set P as

$$d_F(\boldsymbol{x}_{\hat{p}}, \boldsymbol{x}) = \min\{d_F(\boldsymbol{x}_p, \boldsymbol{x}) \mid \boldsymbol{x}_p \in P\} , \tag{4}$$

where $d_F(\boldsymbol{x}_p, \boldsymbol{x})$ is the distance between $\boldsymbol{x}_p$ and $\boldsymbol{x}$, which is defined by the feature set F as

$$d_F(\boldsymbol{x}_p, \boldsymbol{x}) = \sqrt{\sum_{i \in F} (x_{pi} - x_i)^2} . \tag{5}$$

When the instance set P does not include any instance, the classification of new instances is always rejected. The classification is also rejected when the feature set F does not include any feature.

In our instance and feature selection problem, the number of selected instances and the number of selected features are to be minimized, and the classification performance of the reference set $S = (F, P)$ is to be maximized. Thus our problem is formulated as follows:

$$\text{Minimize } |F|, \text{ minimize } |P|, \quad \text{and maximize } \mathit{Performance}(S) , \tag{6}$$

where $|F|$ is the number of features in F, $|P|$ is the number of instances in P, and $\mathit{Performance}(S)$ is a performance measure of the reference set $S = (F, P)$. The performance measure is defined based on the classification results of the given m instances by the reference set $S = (F, P)$.

In our former work (Ishibuchi and Nakashima, 1999, 2000), we defined the performance measure $\mathit{Performance}(S)$ by the number of correctly classified instances by $S = (F, P)$. Each instance $\boldsymbol{x}_q$ ($q = 1, 2, ..., m$) was classified by its nearest neighbor $\boldsymbol{x}_{\hat{p}}$, which is defined as

$$d_F(\boldsymbol{x}_{\hat{p}}, \boldsymbol{x}_q) = \min\{d_F(\boldsymbol{x}_p, \boldsymbol{x}_q) \mid \boldsymbol{x}_p \in P\} . \tag{7}$$

We denote this performance measure for the reference set S as $\mathit{Performance}_{\mathrm{A}}(S)$. It should be noted that the following formulation corresponds to the instance selection problem for finding the minimum consistent set that can correctly classify all the given instances (for example, see Wilson, 1972 and Dasarathy, 1994):

$$\text{Minimize } |P| \text{ subject to } \mathit{Performance}_{\mathrm{A}}(S) = m , \tag{8}$$

where $S = (F_{\mathrm{ALL}}, P)$ and $P \subseteq P_{\mathrm{ALL}}$. From the comparison between (6) and (8), we can see the difference between our task and the instance selection problem for the minimum consistent set.

In the definition of the performance measure in (Kuncheva, 1995, 1997) for the instance selection, when an instance $\boldsymbol{x}_q$ was included in the reference set, $\boldsymbol{x}_q$ was not selected as its own nearest neighbor. In the context of instance and feature selection, this means that the nearest neighbor $\boldsymbol{x}_{\hat{p}}$ of $\boldsymbol{x}_q$ is selected as follows:

$$d_F(\boldsymbol{x}_{\hat{p}}, \boldsymbol{x}_q) = \begin{cases} \min\{d_F(\boldsymbol{x}_p, \boldsymbol{x}_q) \mid \boldsymbol{x}_p \in P\}, & \text{if } \boldsymbol{x}_q \notin P, \\ \min\{d_F(\boldsymbol{x}_p, \boldsymbol{x}_q) \mid \boldsymbol{x}_p \in P - \{\boldsymbol{x}_q\}\}, & \text{if } \boldsymbol{x}_q \in P. \end{cases} \tag{9}$$

We denote the performance measure defined in this manner as $Performance_{\mathrm{B}}(S)$. The instance selection problem discussed in (Kuncheva, 1995) can be written as

$$\text{Maximize } Performance_{\mathrm{B}}(S), \tag{10}$$

where $S = (F_{\mathrm{ALL}}, P)$ and $P \subseteq P_{\mathrm{ALL}}$.

These two definitions of the performance measure are different only in the classification of instances included in the reference set. When a small number of instances are selected and included in the reference set (e.g., 1/30 of the given instances), these two definitions are almost the same because most instances are classified in the same manner. Thus it seems that we will obtain almost the same results from these two definitions. This expectation is examined by computer simulations in the next section.

The fitness value of the reference set $S = (F, P)$ is defined by the weighted sum of our three objectives as

$$\begin{aligned} fitness(S) = W_{Performance} \cdot Performance(S) \\ - W_F \cdot |F| - W_P \cdot |P|, \end{aligned} \tag{11}$$

where $W_{Performance}$, W_F, and W_P are user definable non-negative weights. Since the three objectives in our instance and feature selection problem in (6) are combined into the single scalar fitness function in (11), a single-objective genetic algorithm is used for searching for a single solution. Of course, it is possible to use multi-objective genetic algorithms for searching for multiple non-dominated solutions as in (Ishibuchi et al., 1997).

2.3 BASIC ALGORITHM

We use a genetic algorithm for maximizing the fitness function in (11). In our genetic algorithm, first a number of binary strings (say, N_{pop} strings) of the length $(n+m)$ are randomly generated. Next a pair of strings are randomly selected from the current population to generate two strings by crossover and mutation. The selection, crossover, and mutation are iterated to generate N_{pop} strings. The newly generated N_{pop} strings are added to the current population to form an enlarged population of the size $2 \cdot N_{\mathrm{pop}}$. The next population is constructed by selecting the best N_{pop} strings from the enlarged population. The population update is iterated until a pre-specified stopping condition is satisfied. Our genetic algorithm is written as follows:

Step 1 (Initialization):
Randomly generate N_{pop} strings of the length $(n+m)$.

Step 2 (Genetic Operations):
Iterate the following procedures $N_{\mathrm{pop}}/2$ times for generating N_{pop} strings.

1) Randomly select a pair of strings from the current population.
2) Apply a crossover operation to the selected pair of strings for generating two offspring. In computer simulations of this paper, we use the uniform crossover.
3) Apply a mutation operation to each bit value of the two strings generated by the crossover operation. The mutation operation changes the bit value from 1 to 0 or from 0 to 1.

Step 3 (Generation Update):
Add the newly generated N_{pop} strings in Step 2 to the current population of the N_{pop} strings to form an enlarged population of the size $2 \cdot N_{\mathrm{pop}}$. Select the best N_{pop} strings with the largest fitness values from the enlarged population to form the next population.

Step 4 (Termination Test):
If a pre-specified stopping condition is not satisfied, return to Step 2. Otherwise end the algorithm.

Our genetic algorithm is different from the standard implementation (Goldberg, 1989) in the selection and generation update procedures. In our algorithm, the selection of parent strings for the crossover is performed randomly. The selection of good strings is performed in the generation update procedure. In this sense, the generation update procedure of our genetic algorithm can be viewed as a selection procedure for generating a mating pool from which parent strings are randomly selected. We adopted this implementation according to the first attempt of the application of genetic algorithms to instance selection in (Kuncheva, 1995, 1997). We also examined a more standard implementation based on the roulette wheel selection with the linear scaling and a single elite string. Simulation results of these two implementations were almost the same. So we only report simulation results by the above implementation.

2.4 ILLUSTRATION BY SIMPLE EXAMPLE

Let us illustrate our approach to instance and feature selection by a simple numerical example in Fig. 1 (a) where 30 instances from each class are given. We artificially generated this simple example with 60 instances only for illustration purpose. In Fig. 1 (a), the classification boundary is drawn by the nearest neighbor classification using all the given instances. We used our genetic algorithm with the parameter specifications:

String length: 62 (2 features and 60 instances),
Population size: $N_{\mathrm{pop}} = 50$,
Crossover probability: 1.0,

Mutation probability: 0.01,
Stopping condition: 1000 generations,
Weight values: $W_{Performance} = 10$; $W_F = 1$; $W_P = 1$,
Performance measure: $Performance_{\mathrm{A}}(S)$.

Our genetic algorithm selected 11 instances and the two features (i.e., no feature was removed). The selected instances are shown in Fig. 1 (b) together with the classification boundary generated by them. In Fig. 1 (b), all the given instances are correctly classified. Since we used the large weight value (i.e., $W_{Performance} = 10$) for the performance measure $Performance_{\mathrm{A}}(S)$, we had a 100% classification rate on the given instances by the selected reference set. In Fig. 2 (a), we show a simulation result by the performance measure $Performance_{\mathrm{B}}(S)$. The other parameters including the weight values were specified in the same manner as in Fig. 1 (b). In Fig. 2 (a), a single instance is misclassified by the selected nine instances. The classification boundary in Fig. 2 (a) was drawn by the selected instances. In the case of the second definition of the performance measure, each instance in the reference set is not classified by itself when the fitness value is evaluated. Thus the inclusion of misclassified instances in the reference set does not always improve the performance measure. On the contrary, the inclusion of misclassified instances always corrects the classification of those patterns in the case of the first definition. The difference between these two definitions is also discussed in the next section through computer simulations.

Our genetic algorithm with different weight values generates different reference sets. For example, three instances in Fig. 2 (b) were selected by our genetic algorithm with $W_{Performance} = 0.5$ (The other parameters were the same as in Fig. 1 (b)). In Fig. 2 (b), three instances are misclassified by the selected instances. Since a larger weight value (i.e., $W_P = 1$) is assigned to the number of selected instances (i.e., $|P|$) than the number of correctly classified instances (i.e., $Performance_{\mathrm{A}}(S)$), a reference set with a 100% classification rate on the given instances does not always have the maximum fitness value. As a result, our genetic algorithm selected the reference set in Fig. 2 (b), which can not correctly classify all the given instances. In computer simulations with small weight values for the performance measure, the second feature f_2 (i.e., x_2-axis in Fig. 2 (b))) was often removed. Actually, it was not selected by our genetic algorithm with $W_{Performance} = 0.5$ in 25 out of 30 independent trials.

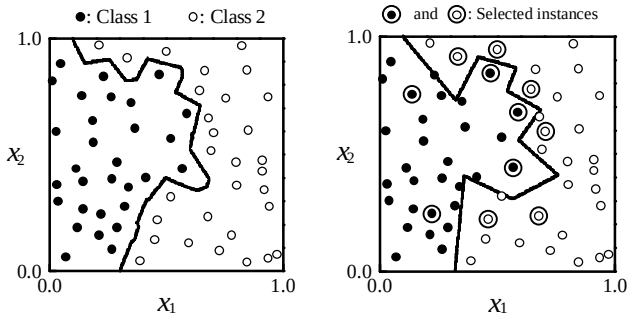

Figure 1: Given instances and selected instances.

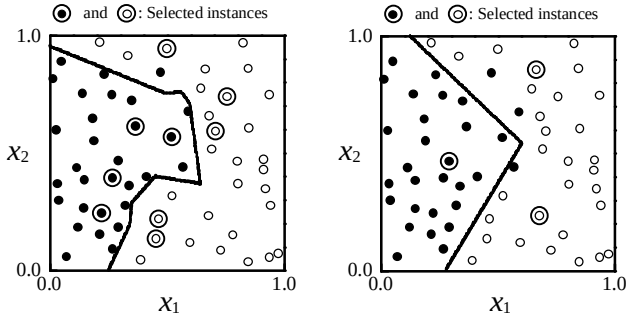

Figure 2: Example of selected instances that can not correctly classify all the given instances.

2.5 BIASED MUTATION

As we can see from the coding mechanism of each reference set into a binary string, our instance and feature selection method is computationally intensive. The string length is $(n+m)$ where n is the number of features and m is the number of instances. Thus the size of the search space is 2^{n+m}, which is terribly large especially when the number of given instances is large. Since the number of features is usually much smaller than the number of instances in many real-world pattern classification problems, we concentrate on how to effectively decrease the number of selected instances by our genetic algorithm in this subsection.

Let us examine the effect of the crossover and mutation on the number of instances included in each string. Since the crossover just exchanges bit values between two parent strings, the total number of selected instances in the parent strings is exactly the same as that in their offspring. This means that the crossover does not change the number of instances on the average. Of course, strings with fewer instances are more likely to survive the generation update due to the definition of the fitness function. Thus the average number of selected instances in each population gradually decreases by iterating the

generation update.

On the contrary, the mutation tends to increase the average number of selected instances. We illustrate this fact using simple numerical calculation. Let m_1 be the number of instances included in a string before the mutation. We also denote the number of excluded instances in the string by m_0 where $m_0 + m_1 = m$ (m is the number of given instances). Among the m_1 instances included in the string, the mutation removes $p_{\mathrm{m}} \cdot m_1$ instances from the string on the average where p_{m} is the mutation probability. At the same time, the mutation adds some instances to the string by changing some 0's to 1's. The expected value of the number of added instances is $p_{\mathrm{m}} \cdot m_0$. Thus the expected value of the number of selected instances after the mutation is calculated as

$$\hat{m}_1 = m_1 - p_{\mathrm{m}} \cdot m_1 + p_{\mathrm{m}} \cdot m_0 . \quad (12)$$

Since a small number of instances are to be selected from a large number of given instances in our instance and feature selection, m_1 should be much smaller than m_0 and m. For example, let us assume that we have a binary string with 10 instances out of 1000 instances (i.e., $m = 1000$, $m_1 = 10$, and $m_0 = 990$). In this case, the expected value $\hat{m}_1$ of the number of selected instances after the mutation is calculated as follows:

$$\hat{m}_1 = 108 \text{ when } p_{\mathrm{m}} = 0.1, \quad (13)$$
$$\hat{m}_1 = 19.8 \text{ when } p_{\mathrm{m}} = 0.01, \quad (14)$$
$$\hat{m}_1 = 10.98 \text{ when } p_{\mathrm{m}} = 0.001. \quad (15)$$

From these calculations, we can see that large mutation probabilities prevent our genetic algorithm from decreasing the number of selected instances.

For demonstrating the effect of the mutation on the number of selected instances, we applied our genetic algorithm to a numerical example with 500 instances from each of two classes (i.e., 1000 instances in total). In this numerical example, we generated 500 instances from each class using the normal distribution $N(\mu_k, \Sigma_k)$ where μ_k is a mean vector, Σ_k is a covariance matrix, and k is the class label ($k = 1, 2$). We specified μ_k and Σ_k as follows:

$$\mu_1 = (0,1), \ \mu_2 = (1,0), \ \Sigma_1 = \Sigma_2 = \begin{pmatrix} 0.3^2 & 0 \\ 0 & 0.3^2 \end{pmatrix}. \quad (16)$$

In our genetic algorithm, we examined three specifications of the mutation probability for instance selection: $p_{\mathrm{m}} = 0.1,\ 0.01,\ 0.001$. The other parameters were specified in the same manner as in Fig. 1 (b). Simulation results are shown in Table 1 where the CPU time was measured by a PC with a Pentium II 400MHz processor. From this table, we can see that the large mutation probability prevented our genetic algorithm from finding compact reference sets. We can also see that the larger the size of reference sets is, the longer the computation time is.

Table 1. Simulation results on pattern classification problems with 1000 instances.

Mutation	$Performance_A(S)$	$\lvert P \rvert$	CPU time
0.1	9657.0	341.0	317.8 (min.)
0.01	9971.0	26.0	108.3 (min.)
0.001	9978.0	16.7	82.1 (min.)

Our trick for effectively decreasing the number of selected instances is to bias the mutation probability (Ishibuchi and Nakashima, 1999, 2000). In the biased mutation, a much larger probability is assigned to the mutation from "$s_p = 1$" to "$s_p = 0$" than the mutation from "$s_p = 0$" to "$s_p = 1$". That is, we use two different mutation probabilities $p_{\mathrm{m}}(1 \to 0)$ and $p_{\mathrm{m}}(0 \to 1)$ for instance selection (i.e., for the last m bits of each binary string). Since the number of features is usually much smaller than the number of instances in many real-world pattern classification problems, we use the standard unbiased mutation for feature selection. That is, the mutation probability p_{m} is not biased for the first n bits of each binary string.

In the same manner as in the above computer simulations with the unbiased mutation, we applied our genetic algorithm with the biased mutation to the pattern classification problem with 1000 instances. The three mutation probabilities were specified as $p_{\mathrm{m}}(1 \to 0) = 0.1$, $p_{\mathrm{m}}(0 \to 1) = 0.001$, and $p_{\mathrm{m}} = 0.1$. The following average results were obtained from three independent trials.

$Performance_{\mathrm{A}}(S)$=9967.0, $|P|$= 4.3, CPU time: 49 min.

From these results, we can see our genetic algorithm with the biased mutation can efficiently search for compact reference sets.

3. PERFORMANCE EVALUATION

3.1 DATA SETS

We used six data sets: two data sets were artificially generated using normal distributions, and the others were real-world data sets used in the literature. In our computer simulations, we applied our genetic algorithm to each data set after normalizing given attribute values to real

numbers in the unit interval [0,1]. In the nearest neighbor classification based on the Euclidean distance, such normalization may be essential for handling data sets including features with different magnitudes. Each data set is briefly described in the following.

Data Set I from Normal Distributions with Small Overlap: We generated a two-class pattern classification problem in the unit square $[0,1]\times[0,1]$. For each class, we generated 50 instances using the normal distribution $N(\mu_k,\Sigma_k)$ where μ_k and Σ_k were specified as follows:

$$\mu_1=(0,1),\ \mu_2=(1,0),\ \Sigma_1=\Sigma_2=\begin{pmatrix}0.4^2 & 0\\ 0 & 0.4^2\end{pmatrix}. \quad (17)$$

Data Set II from Normal Distributions with Large Overlap: We generated a two-class pattern classification problem in the same manner as in the above data set using larger variances. We specified the normal distribution of each class as follows:

$$\mu_1=(0,1),\ \mu_2=(1,0),\ \Sigma_1=\Sigma_2=\begin{pmatrix}0.6^2 & 0\\ 0 & 0.6^2\end{pmatrix}. \quad (18)$$

Iris Data: The iris data set is one of the most commonly used data sets in the literature. This data set consists of 150 instances with four features from three classes (50 instances from each class). The best result reported in (Weiss and Kulikowski, 1991) was a 2.0% error rate on test data (i.e., unseen instances) by linear discriminants.

Appendicitis Data: The appendicitis data set consists of 106 instances with eight features from two classes. Since one feature has some missing values, we used seven features as in (Weiss and Kulikowski, 1991) where ten classification methods were examined by the leaving-one-out procedure for the appendicitis data. The best result reported in their book was a 10.4% error rate on test data by a machine learning technique.

Cancer Data: The cancer data set consists of 286 instances with nine features from two classes. This data set was also used in (Weiss and Kulikowski, 1991) for evaluating the performance of ten classification methods by random resampling where 70% of given instances were used as training data. The best result reported in their book was a 22.9% error rate on test data by a machine learning technique.

Wine Data: The wine data set consists of 178 instances with 13 features from three classes, which is available from the machine learning database in the University of California, Irvine. This data set was used in (Corcoran and Sen, 1994) for evaluating the performance of their genetics-based machine learning algorithm.

3.2 PERFORMANCE ON TRAINING DATA

We applied our genetic algorithm to the six data sets using the following parameter specifications:

Population size: $N_{\text{pop}}=50$,
Crossover probability: 1.0,
Mutation probability: $p_{\text{m}}=0.01$ for feature selection,
$p_{\text{m}}(1\to 0)=0.1,\ p_{\text{m}}(0\to 1)=0.01$ for instance selection,
Stopping condition: 500 generations,
Weight values: $W_{Performance}=5;\ W_F=1;\ W_P=1$,
Performance measure:
$Performance_{\text{A}}(S)$ or $Performance_{\text{B}}(S)$.

All the given instances were used as training data in this subsection. The aim of computer simulations in this subsection is to compare the two definitions of the performance measure. Our genetic algorithm was applied to each data set 30 times for calculating average results. Average simulation results over 30 trials are summarized in Table 2 and Table 3 where each figure in parentheses denotes the number of given features or given instances in each data set.

Table 2. Simulation results on training data using the first performance measure.

Data set	Features	Instances	Classification
Data Set I	1.9 (2)	14.5 (100)	96.7%
Data Set II	1.8 (2)	31.0 (100)	94.4%
Iris	2.0 (4)	6.1 (150)	99.4%
Appendicitis	3.3 (7)	16.0 (106)	97.5%
Cancer	5.1 (9)	54.3 (286)	89.2%
Wine	6.3 (13)	5.9 (178)	100%

Table 3. Simulation results on training data using the second performance measure.

Data set	Features	Instances	Classification
Data Set I	2.0 (2)	6.2 (100)	92.3%
Data Set II	1.8 (2)	12.3 (100)	80.9%
Iris	2.6 (4)	7.6 (150)	94.2%
Appendicitis	3.2 (7)	4.4 (106)	91.8%
Cancer	2.9 (9)	27.2 (286)	81.3%
Wine	6.6 (13)	7.3 (178)	99.9%

In Table 2 and Table 3, we used the first definition $Performance_{\text{A}}(S)$ and the second definition $Performance_{\text{B}}(S)$ of the performance measure, respectively. The first definition directly evaluates the

classification ability on training data for the evolution of reference sets. As a result, we obtained higher classification rates on training data in Table 2 than Table 3. Such higher classification rates were realized by selecting much more instances for constructing reference sets in the case of data sets with large overlaps such as Data Set II, the appendicitis data, and the cancer data. On the other hand, in Table 3, the generalization ability of each reference set on unseen data was estimated in our genetic algorithm by the second definition. Thus the classification rates on training data in Table 3 are inferior to those in Table 2.

From the projection of the iris data into the x_3-x_4 plane, we can see that these two features are important for the classification purpose of the iris data. These two features were selected by our genetic algorithm in 29 out of the 30 trials in Table 2. In Table 3, $\{f_3, f_4\}$ were selected in 16 trials, and $\{f_2, f_3, f_4\}$ were selected in the other 14 trials.

3.3 PERFORMANCE ON TEST DATA

In the previous subsection, we demonstrated that our genetic algorithm can select a small number of instances together with only significant features. It was also shown that the selected reference sets can correctly classify almost all the given instances. While we examined classification rates on training data in the previous subsection, the performance of classification systems should be evaluated by classification rates on test data (i.e., unseen instances). In this subsection, we examine the generalization ability of selected reference sets.

Since the first two data sets were artificially generated from the given normal distributions, we can generate unseen instances from the same normal distributions. In our computer simulations, 1000 instances (500 from each class) were generated as test data. That is, a reference set selected from 100 instances was examined on 1000 instances at each trial. This procedure was iterated 50 times for Data Set I and Data Set II. For the iris data and the appendicitis data, we used the leaving-one-out (LV1) procedure as in (Weiss and Kulikowski, 1991). The LV1 procedure was iterated ten times for the iris data and the appendicitis data. For the cancer data and the wine data, we used the 10-fold cross-validation (10CV) procedure. In the 10CV procedure, the given instances are divided into ten subsets of the same size. One subset is used as test data, and the other subsets are used as training data. This is iterated ten times so that all the subsets are used as test data. The 10CV procedure was employed ten times for the cancer data and the wine data.

We used the same parameter values of our genetic algorithm as in the previous subsection. We examined the two definitions of the performance measure (i.e., $Performance_{\mathrm{A}}(S)$ and $Performance_{\mathrm{B}}(S)$). Simulation results are summarized in Table 4. For comparison, we also examined the generalization ability of the original data sets before instance and feature selection. From Table 4, we can see that the generalization ability was improved by the use of $Performance_{\mathrm{B}}(S)$ in many data sets. The improvement of the generalization ability is clear in the appendicitis data and the cancer data with large overlaps between different classes. On the contrary, we can not observe such clear improvement in the iris data and the wine data. Those data sets, for which we obtained high classification rates on test data by the original nearest neighbor classifiers, do not have large overlaps between different classes in the pattern spaces. From the comparison between the two definitions of the performance measure, we can see that higher classification rates on test data were obtained by the second definition $Performance_{\mathrm{B}}(S)$ than the first definition $Performance_{\mathrm{A}}(S)$. This is because the first definition is based on the classification ability on training data. That is, the evolution of reference sets guided by the first definition tends to overfit to training data. As a result, instance and feature selection based on the first definition is not likely to improve the generalization ability of nearest neighbor classifiers.

Let us briefly compare the simulation results in Table 4 with those by ten classification methods reported in (Weiss and Kulikowski, 1991). The 96.9% classification rate for the iris data set in Table 4 is better than six methods (e.g., 96.7% by the back-propagation algorithm). The 85.7% classification rate for the appendicitis data is better than five methods. This result is almost the same as the reported result 85.8% by the back-propagation algorithm. The 73.6% classification rate for the cancer data is better than eight methods (e.g., 71.5% by the back-propagation algorithm).

Table 4. Average classification rates on test data.

Data set	Original data	$Perf_{\mathrm{A}}(S)$	$Perfe_{\mathrm{B}}(S)$
Data Set I	81.3%	80.2%	84.7%
Data Set II	60.6%	60.2%	64.8%
Iris	95.3%	96.9%	94.2%
Appendicitis	80.2%	77.0%	85.7%
Cancer	65.3%	68.3%	73.6%
Wine	95.3%	94.8%	96.5%

4. CONCLUSIONS

In this paper, we discussed instance and feature selection for the design of compact nearest neighbor classifiers. Through computer simulations on artificially generated data sets and real-world data sets, we demonstrated that a small number of instances were selected together with only significant features by our genetic algorithm. That is, our genetic algorithm can simultaneously perform instance selection and feature selection. We also demonstrated that the generalization ability of nearest neighbor classifiers was improved by the use of the selected instances and features in some data sets. This improvement was clear in the case of data sets with large overlaps between different classes. We examined two performance measures used for calculating the fitness value of each reference set in our genetic algorithm. One performance measure was defined by the classification performance of the reference set on given instances. That is, the classification performance was evaluated by classifying all the given instances by the reference set. This performance measure is suitable for finding the minimum reference set that can correctly classify all the given instances. In the evaluation of the other performance measure, the classification of all the given instances is performed in a different manner. The point is that every instance included in the reference set is not used as its own nearest neighbor. In this manner, the generalization ability of each reference set is evaluated in the execution of our genetic algorithm. As shown by our computer simulations, this performance measure is suitable for selecting compact reference sets with high generalization ability from data sets with large overlaps between different classes.

References

Chaudhuri, D., Murthy, C. A., and Chaudhuri, B. B. 1994. Finding a subset of representative points in a data set. *IEEE Trans. on Systems, Man, and Cybernetics*, 24: 1416-1424.

Corcoran, A. L. and Sen, S. 1994. Using real-valued genetic algorithms to evolve rule sets for classification. *Proc. of 1st IEEE International Conference on Evolutionary Computation*. Orlando, FL. 120-124.

Cover, T. M. and Hart, P. E. 1967. Nearest neighbor pattern classification. *IEEE Trans. on Information Theory*, 13: 21-27.

Dasarathy, B. V. 1994. Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design. *IEEE Trans. on Systems, Man, and Cybernetics*, 24: 511-517.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA, Addison-Wesley.

Hart, P. 1968. The condensed nearest neighbor rule. *IEEE Trans. on Information Theory*, 14: 515-516.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, University of Michigan Press.

Ishibuchi, H., Murata, T., and Turksen, I. B. 1997. Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89: 135-150.

Ishibuchi, H. and Nakashima, T. 1999. Evolution of reference sets in nearest neighbor classification, in B. McKay et al.(eds.) *Lecture Notes in Artificial Intelligence 1585: Simulated Evolution and Learning (2nd Asian-Pacific Conference on Simulated Annealing, Canberra, 1998, Selected Papers)*. 82-89.

Ishibuchi, H. and Nakashima, T. 2000. Pattern and feature selection by genetic algorithms in nearest neighbor classification. *International Journal of Advanced Computational Intelligence* (to appear).

Kelly, Jr., J. D. and Davis, L. 1991. Hybridizing the genetic algorithm and the *k* nearest neighbors classification algorithm. *Proc. of 4th International Conference on Genetic Algorithms*, University of California, San Diego, 377-383.

Kuncheva, L. I. 1995. Editing for the *k*-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16: 809-814.

Kuncheva, L. I. 1997. Fitness functions in editing *k*-NN reference set by genetic algorithms. *Pattern Recognition*, 30: 1041-1049.

Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., and Enbody, R. 1993. Further research on feature selection and classification using genetic algorithms. *Proc. of 5th International Conference on Genetic Algorithms*, University of Illinois at Urbana-Champaign, 557-564.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group. 1986. *Parallel Distributed Processing*. Cambridge, MA. MIT Press.

Siedlecki, W. and Sklansky, J. 1989. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10: 335-347.

Weiss, S. M. and Kulikowski, C. A. 1991. *Computer Systems That Learn*. San Mateo, CA. Morgan Kaufmann.

Wilson, D. L. 1972. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. on Systems, Man, and Cybernetics*, 2: 408-420.