Computing and Informatics, Vol. 32, 2013, 1–22

AN INTELLIGENT GENETIC ALGORITHM FOR MINING CLASSIFICATION RULES IN LARGE DATASETS

P. Vivekanandan

Department of Computer Science and Engineering
Park College of Engineering and Technology
Coimbatore, India
e-mail: anandpvivek@yahoo.co.in

M. Rajalakshmi

Department of Computer Science and Engineering and Information Technology
Coimbatore Institute of Technology
Coimbatore, India
e-mail: raji_nav@yahoo.com

R. Nedunchezhian

Department of Information Technology
Sri Ramakrishana College of Engineering
Coimbatore, India
e-mail: rajuchezhian@gmail.com

Communicated by Vladimír Kvasnička

Abstract. Genetic algorithm is a popular classification algorithm which creates a random population of candidate solutions and makes them to evolve into a suitable accurate solution for a given problem by processing them iteratively for several generations. During each generation the training data set is accessed by the genetic algorithm only for the population member's fitness calculation and no other extra knowledge about the problem domain is extracted from the training data set. Even the domain knowledge stored in the chromosome code of the population may be

lost in the future generations due to genetic operations. All the genetic operations like crossover and mutation are probability based and they do not depend upon the domain knowledge. This phenomenon makes the genetic algorithm to converge slowly. This paper proposes a genetic algorithm which tries to gain maximum knowledge in between the generations and store them in the form of knowledge chromosomes. The gained knowledge is used to make predictions about the search space and to guide the search process to an area with potential solutions in the subsequent generations. This makes the genetic algorithm to converge quickly which in turn reduces the learning cost. The experiments show that the run time is reduced considerably when compared with the state-of-the-art evolutionary algorithm.

Keywords: Classification, genetic algorithm (GA), knowledge discovery, scalability

1 INTRODUCTION

Genetic algorithm (GA) is a search technique purely based on the natural evolution process. It is widely used by the machine learning and data mining community for the classification rule discovery [1, 2, 3, 4, 5, 6, 7] process. It has been proved that the performance of GA [6] is comparable with the other classification methodologies like neural networks, decision trees etc.

During the rule discovery process, the entire GA based algorithms described in the literature [7] access the training data set repeatedly for the potential candidate rules fitness calculation. Based on the extracted fitness value, the candidate rules are processed using the genetic operators to generate new child candidate rules which have more accurate predictive capabilities than their parents. It can be noted that during the fitness calculation process no other extra knowledge other than the fitness value of the rules is extracted from the training data set. However, there are more possibilities for extracting domain knowledge from the training data set, since during the GA iterations the training data set records are individually examined and are accessed repeatedly. If more knowledge can be extracted then such extracted knowledge can be used to guide the search process quickly into promising areas with accurate solutions and this will reduce the computational cost, particularly when the training data size is very large and complex.

The GA proposed in this paper extracts maximum knowledge from the training data set during each generation and stores them as Knowledge Chromosomes. In addition to the fitness value, the Knowledge Chromosomes are also used by the GA operators to process the candidate rules. The proposed method is just an ordinary GA which extracts and uses domain knowledge without any extra effort and it does not provide any new memetic operators [8] as done by memetic computing algorithms or replace any genetic operators [9] as done by estimation of distribution algorithms. In fact the proposed method can be combined and implemented with any evolutionary algorithm.

In this paper, Section 2 describes the literature review and Section 3 introduces the general features of GA for classification and the historical knowledge that can be extracted between generations. The new proposed GA is described in Section 4, while Section 5 describes the simulation and results. Finally, Section 6 concludes the paper.

2 LITERATURE REVIEW

Genetic algorithm is purely based on natural evolutionary process and was introduced by Holland [10] in 1975. Slowly it gained wide recognition and several important classification algorithms like Learning Classifier Systems (LCS) [11, 12] and Genetics-Based Machine Learning (GBML) [13] emerged from it. Such evolutionary based methods can be classified into two main approaches based on their chromosome representation and learning methods. Algorithms like GABIL [11], GIL [14] and Hidder [15] employ a single chromosome to represent the whole solution and each chromosome will contain multiple rules. This type of approach is called Pittsburgh approach. Second type of approach is called Michigan approach in which each chromosome is a fixed length rule and the whole population is the solution for the classification problem. XCS proposed by Wilson et al. [12] applies Michigan style of population formation. All the above algorithms design a classification model based on small data sets.

However, today data sets are large, complex and several techniques have been proposed to enhance the GA performance and also to reduce their computational cost. One of the important techniques is parallel processing [18, 19] like GA_PVMINER [18] which divides the training data set into multiple parts and they are distributed to multiple processors. Each processor generates the rules for each data part independently and then the rules generated by the processors are again shared between all the processors for their fitness calculation.

Incremental learning [20, 21 and 22] is another way of reducing the computational cost and is widely used for mining large data sets. The initial algorithm was GAssist system [21] which is a Pittsburgh style classifier. It divides the training data into several non overlapping strata and uses different strata for different GA iterations in a sequence. The system is an enhancement of LCS and produces highly efficient and compact solutions. Later the authors of GAssist created a new algorithm called BioHEL (Bioinformatics-oriented hierarchical evolutionary learning) [16, 17, 22] in order to enhance the scalability of GA and make it suitable for classifying large scale bioinformatics data sets. BioHEL applies iterative rule learning (IRL) approach [23], which learns the rule one by one and after learning a rule it excludes those records which the rule predicts correctly from the training data set and starts the next iteration. Attribute List Knowledge Representation (ALKR) [24] is an advanced version of BioHEL which identifies relevant attributes for each rule and constructs the rules only based on the identified relevant attributes and excludes all other non relevant attributes from the rules. To add and remove

attributes from a rule ALKR introduces generalization and specializations genetic operators.

Classification algorithms that are employed for data mining are based on the above described evolutionary learning methods with slight modifications in rule representation and fitness calculation [7]. The next section describes the general features of the GA classification with respect to Data Mining perspective.

3 GENERAL FEATURES OF CLASSIFICATION GA

Classification rules can be considered as a kind of prediction rules where the rule antecedent ("IF part") is the conjunction of conditions, say A (conjunction of attribute value pairs $A_1, A_2 \ldots A_n$) and the consequent part C (THEN part) is the class label; an individual rule can be represented as

$$A1 \wedge A2 \wedge \ldots \wedge A_n \text{ THEN } C. \tag{1}$$

In the Pittsburg approach a single chromosome contains a set of rules which forms the solution to the problem and in the Michigan approach a chromosome represents a single individual rule.

The GA classification process starts from a population of randomly generated candidate rules and the rules are processed in sequential steps, such that an accurate solution gradually emerges. The sequential steps are called generations. Each generation goes through the selection (or testing) phase and the reproduction phase. During the selection phase, candidate rules accuracy (fitness) is evaluated using the training data set.

3.1 Historical Knowledge

Generally large amount of knowledge can be captured from the training data set in between the generations of the GA process, but normally they are not captured and such kind of knowledge is called historical knowledge.

To understand what kind of knowledge can be gained between generations, consider a multi class classification problem with three attributes a $\{a_1, a_2, a_3\}$, b $\{b_1, b_2, b_3\}$ and c $\{c_1, c_2, c_3\}$ and a class label C $\{C_1, C_2, C_3\}$ (Table 1). Let the training set consist of six records as described in Table 2

Attribute name	Values
a	a_1, a_2, a_3
b	b_1, b_2, b_3
c	c_1, c_2, c_3
C (class label)	C_1, C_2, C_3

Table 1. Example problem

Attribute values	Class
a_1 b_2 c_3	C_1
a_1 b_2 c_3	C_1
a_3 b_3 c_1	C_2
a_3 b_3 c_1	C_2
a_2 b_1 c_2	C_3
a_2 b_1 c_2	C_3

Table 2. Training set

Let GA be used to build a classifier based on the training data set described in Table 2 for the problem described in Table 1. The classifier will contain classification rules for classes C_1, C_2 and C_3. Let the rules described by Equations (2) and (3) be two candidate rules of class C_1 and C_3 for the above example problem that are generated during the GA process.

$$a = a_1 \text{ and } b = b_2 \text{ and } c = c_3 \text{ then class} = C_1 \tag{2}$$

$$a = a_2 \text{ and } b = b_1 \text{ and } c = c_1 \text{ and class} = C_3 \tag{3}$$

During the fitness calculation process prediction accuracy of both rules is tested by matching them with the records in the training data set (Table 2). The first rule (Equation (2)) predicts all the records in the training set of class C_1 correctly; therefore it has high accuracy and this makes it a potential solution for the problem; but the second rule does not predict any of the class C_3 records of the training set correctly and so it has a low accuracy. In an ordinary GA process, because of high accuracy the first rule will be treated as seed to generate the child population and due to low accuracy the second rule will be treated as an unfit solution which may be discarded or mutated with a high probability. The parts of the high accuracy rule will be duplicated and parts of the low accuracy rule will be destroyed fully or partially.

However, during the fitness calculation process, if we examine the attribute values of the less accurate second rule (Equation 3) separately we can clearly extract some extra knowledge which is embedded with in it as described below:

1. When the value of both attribute a ($a = a_2$) and b ($b = b_1$) of the rule are examined separately, they have high confidence (Definition 1) and so they will have high prediction accuracy for class C_3. This is due to the fact that attributes a and b of class C_3 records in the training set (Table 2) have these values and so they can be a part of a future potential solution for class C_3. They both will form a better building block (schemata) and class C_3 rules must be encouraged to have these values for their a and b attributes in the future generations of the GA process. When examined separately, the value of attribute c ($c = c_1$) of the rule has low confidence (Definition 1) for class C_3 because no record of class C_3 in the training set (Table 2) has this value for the attribute c. During the mutation of class C_3 rules in the subsequent generations, part of the rule

corresponding to attribute c must be prevented from taking this value for better convergence.

2. To determine the predictive capability of the rule, training records of the classes other than the rule class (class C_3) must also be examined. So while comparing the rule with the records of class C_2, it can be identified that the value of attribute c ($c = c_1$) of the rule has high confidence with respect to class C_2 since many of the c attributes of the particular class records in the training set (Table 2) has this value. It can be a part of a potential solution for class C_2. It can be further identified that a's ($a = a2$) and b's ($b = b_1$) values of the rule can be considered as low confidence (Definition 1) for class C_2 because attributes a and b of class C_2 records in the training set do not have those values. As discussed earlier, in the future generations Class C_2 candidate rule attributes can be encouraged to have only high confidence values and must be prevented from having the low confidence values during the GA process.
3. In the same way, it can also be identified that the values of a, b and c of the same rule have low confidence with respect to class C_1 and attributes of its candidate rules must be prevented from taking these values in the further iterations.

The above knowledge can be extracted and stored between generations and if done, it can be used to control the GA operators in the further stages of the GA. By controlling the GA operators the search process can be driven into areas where the solution exists and can be prevented from entering into a non productive area. The proposed algorithm in the next section describes how to extract and store such type of historical knowledge during the GA iterations.

Definition 1. Confidence of an attribute value (with respect to a class): Capability of an attribute value of being a potential building block for a classs future rules. High confidence indicates that it predicts the corresponding class records with high accuracy and will be a part of future rule. Low confidence indicates that it has poor predictive accuracy and so it will not be a part of the future solution of the corresponding class.

4 PROPOSED GENETIC ALGORITHM

The proposed method has two major components, namely Population Space and Knowledge Space, and three major functions, namely Knowledge Extraction function, Genetic Process and Knowledge Injection function.

Population Space contains the potential solutions for the given problem. In the Population Space there will be a separate independent sub-population for all the classes of the problem. The Knowledge Space will contain a Knowledge Chromosome and a set of chromosomes called Best Chromosomes for all the classes. The Knowledge Extraction function extracts maximum knowledge from the candidate chromosomes and the training data set. Based on the extracted knowledge it upgrades the Knowledge Space. The Genetic Process applies genetic operators to

the population and produces a new child population for the next generation. The Knowledge Injection function checks the population and modifies them based on the Knowledge Chromosomes so that the evolution process is guided in a correct direction.

The pseudo code for the proposed algorithm is described below. In the following subsections it describes the process of the methodology is described in detail.

Intelligent_GA()

Generate the initial population $\mathrm{POP}_1, \mathrm{POP}_2, \ldots, \mathrm{POP}_n$ for all classes C_1 to C_n separately // n-number of classes

Create and Initialize the belief space (Knowledge Chromosomes $\mathrm{K}_1, \mathrm{K}_2$–$\mathrm{K}_n$ and a set Best Chromosomes, $\mathrm{BEST}_1, \mathrm{BEST}_2$–$\mathrm{BEST}_n$ for all the classes C_1 to C_n)

// BEST_n is the set of best chromosomes for class n.

// K_n is Knowledge chromosomes for class n.

Repeat

For each Sub Population POP_i

do

Apply Knowledge Extraction function and update the Knowledge Space.

Apply Genetic Process to the chromosomes in the population and generate new child population.

Apply Knowledge Injection function to the child population

Until the end condition is reached.

Produce the BEST chromosomes of all the classes as the final solution.

The end condition can be either a fixed number of generations or until there is no increase in the overall fitness value of the population for some fixed number of generations.

4.1 Population Space

The Population Space contains sub-population for all the classes of the problem. Each chromosome of the sub-population represents a classification rule for that particular class. Binary encoding is employed to represent a chromosome. Let there be n input attributes and one target (class) attribute. The chromosome contains $n + 1$ genes. Each gene in the chromosome represents an input attribute and the last gene represents the class attribute. Let an attribute take m different values. It will be encoded by using $m + 1$ binary bits. The m bits starting from the second bit are used to represent each value of the attribute. The first bit is called flag bit and it is used to indicate whether the particular attribute forms a part of the classification rule or not. The population of a class can be considered as a set of chromosomes representing the best solution at that instant. In the proposed GA the rules of all the classes are generated separately and so the last gene representing the class attribute can be optional and if present in the rule it will not have the flag bit.

Consider a classification problem with three attributes $\{a, b, c\}$ and one target attribute C. Each attribute a, b and c can take any one of the three values from the domain set $\{(a_1, a_2, a_3), (b_1, b_2, b_3), (c_1, c_2, c_3)\}$ and the target attribute will take any value from the domain set $\{(C_1, C_2, C_3)\}$. The example chromosome coding and its meaning is shown in Table 3.

Attribute	a	b	c	C
Code	1010	0001	1100	100
Rule	If $a = a_2$ and $c = c_1$ Then $C = C_1$			

Table 3. Chromosome coding

4.2 Knowledge Space

Knowledge Chromosomes are created in the Knowledge Space for all the classes of the given problem. In the Knowledge Chromosome a gene is created for all the attributes of the problem. The gene contains a set of Knowledge Variables (or Knowledge Bits) representing all the values that are present in the domain of the corresponding attribute. Each Knowledge Bit takes any one of the three values from the set $\{1, 0, \#\}$ and it measures the confidence (Definition 1) of that particular attribute domain value with respect to the corresponding Knowledge Chromosome class. If the Knowledge Bit is set to 1 (good) then the domain value corresponding to that attribute can be considered to have high confidence (Definition 1) with respect to that particular class; if it is set to 0 (bad) then the particular domain value can be considered to have low confidence with respect to the particular class. If it is set to # (unknown) then the confidence status of the particular domain value is unknown. Initially all the Knowledge Bits of the Knowledge Chromosome of all the classes are set to the unknown (#) value and then will be modified to 1 or 0 after each generation based on the knowledge extracted in that particular generation.

Let 001 0#1 10# be a Knowledge Chromosome of class C_1 for the example problem described in Table 1. It has three genes and each gene corresponds to the three attributes a, b and c of the problem. Each gene of the Knowledge Chromosome will have a representation for all the domain values of the corresponding attribute indicating their confidence level with respect to its class (in this case it is for class C_1). The first gene in the example indicates that the third domain value a_3 of the attribute 'a' has the required confidence and the first two domain values a_1 and a_2 do not have the required confidence for class C_1. Similarly, the second gene indicates that, for the attribute 'b', the first domain value b_1 does not have the required confidence and the third value b_3 has the required confidence. From the third gene we can understand that for the attribute 'c', the first value c_1 has the required confidence and the second value c_2 does not have the required confidence for class C_1. This knowledge stored in the Knowledge Chromosome can be used

during the GA process like mutation and crossover to generate accurate rules and thus making the operators more productive.

Knowledge Space also stores all the best chromosomes that are generated after each generations for all the classes ($BEST_1$–$BEST_n$) and at the end of the processes they form a solution to the problem.

4.3 Knowledge Extraction Function

Below the Knowledge Extraction process is described:
For each chromosome in the population
do
    Create the Confidence Matrix for the chromosome
    Calculate the Gene Confidence of all the genes of the chromosome
        with respect to the classes using the Confidence Matrix.
    Update the Knowledge Chromosomes of all the classes based
        on the Gene confidence of all the genes of the chromosome.
end do
Identify the potential candidate chromosomes and add it to the BEST part
    of the Knowledge Space.

4.3.1 Creating Confidence Matrix

Let the problem domain have m attributes (a, b, c–) and n classes (C_1, C_2, ..., C_n). Then the Confidence Matrix is a m row and n column matrix which is created for all the chromosomes in the population. Each row is associated with one of the gene attribute values which make the chromosome and each column associated with one of the n classes (C_1 to C_n) of the problem. The matrix cell stores the number of training records in the training data set that has the attribute value corresponding to the row of the cell and the class label corresponding to the column of the cell.

All the cell values of the Confidence Matrix of a chromosome are calculated based on the training data set provided by comparing the records in the data set with the chromosome. Initially all the cell values of the Confidence Matrix are initialized to zero.

Consider the training set shown in Table 4 for a problem with three attributes (a, b and c) and three classes (C_1, C_2 and C_3). Also consider a candidate chromosome as described by Equation (4).

$$a = a_1 \text{ and } b = b_2 \text{ and } c = c_3 \text{ then class} = C_1 \tag{4}$$

The Confidence Matrix for the above chromosome is shown in Table 5 which has three rows (for the genes $a = a_1$, $b = b_2$ and $c = c_3$) and three columns corresponds to the class labels C_1, C_2 and C_3, respectively. It is calculated based on the training set shown in Table 4.

Attribute values	Class
a_1 b_2 c_3	C1
a_1 b_2 c_3	C1
a_3 b_3 c_1	C2
a_3 b_3 c_1	C2
a_2 b_1 c_2	C3
a_2 b_1 c_2	C3

Table 4. Training data set

Attribute value/class	(C_1)	(C_2)	(C_3)
$a = a_1$	2	0	0
$b = b_2$	2	0	0
$c = c_3$	2	0	0

Table 5. Confidence matrix

4.3.2 Calculating the Gene Confidence of a Chromosome

After calculating the Confidence Matrix of a chromosome and based on its cell values the Gene Confidences of all the genes (Attribute value pair) of the chromosome are calculated not only with respect to the chromosome class but also with respect to all the other classes For example, Gene Confidence of gene $a = a_1$ of the chromosome represented by Equation (4) (Confidence Matrix shown in Table 5) is calculated not only with respect to the chromosome class C1 but also with respect to the other two classes C_2 and C_3.

Gene Confidence of a gene (Cg_{ij}) at position i (i^{th} row in the Confidence Matrix) of a chromosome with respect to a class C_j (class represented in j^{th} column of Confidence Matrix) is calculated using Equation (5).

$$Cg_{ij} = \frac{v_{ij}}{\sum_{z=1}^{n} v_{iz}} \tag{5}$$

In the equation, z ranges from 1 to n (number of classes) and v_{iz} is the cell value at location i, z (i^{th} row z^{th} column) of the Confidence Matrix corresponding to that particular chromosome and v_{ij} describes the cell value in the same Confidence Matrix of the corresponding chromosome at location i, j (i^{th} row, j^{th} column).

For the example chromosome $a = a_1$, $b = b_2$ and $c = c_3$ then class $= C_1$ (Equation (4)) the Gene Confidence of its genes ($a = a_1$, $b = b_1$ and $c = c_1$) with respect to class C_1 based on its Confidence Matrix (Table 5) is calculated as described below:

Gene Confidence of the gene $(a = a_1)$ with respect to class $C_1 = 2/(2 + 0 + 0) = 1$

Gene Confidence of the gene $(b = b_2)$ with respect to class $C_1 = 2/(2 + 0 + 0) = 1$

Gene Confidence of the gene $(c = c_3)$ with respect to class $C_1 = 2/(2 + 0 + 0) = 1$

Similarly, all the Gene Confidence values of the genes of the chromosome with respect to the other two classes C_2 and C_3 other than the chromosome class are calculated.

4.3.3 Updating the Knowledge Chromosome

If the Gene Confidence of a gene of a chromosome with respect to a class is greater than the confidence threshold (above 0.7) provided by the user then the corresponding Knowledge Bit of the particular gene in the Knowledge Chromosome of the corresponding class is set to 1 and if it is less than the proposed threshold, then the Knowledge Bit will be set to zero. Knowledge Chromosomes of all the classes are altered similarly based on the genes of the chromosome under consideration. Let the initial knowledge chromosome for class C_1 be as follows:

a	b	c
# # #	# # #	# # #

Table 6. Initial knowledge chromosome for class C_1

Let the confidence threshold Cg be 0.7. After calculating the gene confidence of each gene present in the example chromosome "$a = a_1$ and $b = b_2$ and $c = c_3$ of class $= C_1$" (Equation (4)) the Knowledge Chromosome of the class C_1 indicated in Table 6 will be updated as follows:

a	b	c
1 # #	# 1 #	# # 1

Table 7. Knowledge chromosome of class C_1 after modification

Modifications to the Knowledge Chromosomes for class C_2 and C_3 are made in the same manner.

4.3.4 Best Chromosomes

Best Chromosomes for each class are selected based on the Chromosome predictive accuracy. The Predictive Accuracy is made up of two terms namely Confidence Factor and Comprehensibility. The Confidence Factor (Equation (6)) of all the chromosomes in the population is calculated using their own Confidence Matrix.

$$\text{Confidence-Factor (CF)} = (|A \,\&\, C| - 1/2)/|A| \tag{6}$$

where $|A|$ describes the number of examples satisfying all the conditions in the antecedent A. $|A \,\&\, C|$ describes the number of examples that satisfy both antecedent A and consequent C. Intuitively, this metric measures both the antecedent and consequent cases which hold out of all cases where the antecedent holds. The term $1/2$ is subtracted to penalize the rules covering few training examples [2].

The standard way of measuring Comprehensibility is to count the number of conditions in the rule. If a rule has at most L conditions, the Comprehensibility of the rule can be defined as follows:

$$\text{Comprehensibility (CM)} = (L - x)/(L - 1). \tag{7}$$

where x is the length of the corresponding rule.

The overall Predictive Accuracy (PA) of the chromosome is computed as the arithmetic weighted mean of CM and CF. The predictive accuracy is given by

$$PA = W_1 \cdot CF + W_2 \cdot CM \tag{8}$$

where W_1 and W_2 (usually $W_1 = 0.6$ and $W_2 = 0.4$) are the weights defined by the user.

Chromosomes whose predictive accuracy above a threshold (value above 0.7) are considered as the best solution and are copied to the BEST set of the Knowledge Space. After the GA process the chromosomes in the BEST part of the Knowledge Space are produced as output.

4.4 Genetic Process

After Knowledge Extraction process the genetic process is applied to the population to create a new population for the next generation. The genetic process has four functions, namely fitness calculation, crossover, mutation, Insert and Remove operators and elitism selection. The genetic operators are applied one by one and the new population is created.

4.4.1 Calculating the Fitness of Chromosomes and Population

The fitness of the individual chromosomes is their predictive accuracy calculated as described in Section 4.3.4.

Overall fitness of the population with r chromosomes is

$$FP = \sum_{j=1}^{r} F_j \tag{9}$$

where F_j is the fitness of the j^{th} chromosome.

Average fitness of the population is

$$F_{\text{avg}} = \frac{FP}{r} \tag{10}$$

4.4.2 Crossover

Selection process for crossover is based on applying the roulette wheel rule. Two best chromosomes are selected for the crossover process and the crossover employed is a

multipoint crossover which produces a single offspring. Each gene value of the first chromosome is compared with the corresponding gene value of the other chromosome and one is selected among them for creating the offspring. The selection probability is based on their Knowledge Chromosome values. If the values are equal then one of the competing genes is selected with equal probability. If they are different then the gene with the highest knowledge value has selection probability of 0.7 while the other gene has the selection probability of 0.3.

4.4.3 Mutation

In general, mutation operation is used to divert the search to a new area and in a classification GA; the mutation process is just changing the attribute value of a rule to another random value selected from the domain of the attribute with a probability called mutation probability.

Let there be a problem with three multi dimensional solution spaces area_1, area_2 and area_3 shown in Figure 1 and let area_3 have the potential solution for the problem. Let us assume that the GA search for the solution was started from area_1 and by mutation it has been diverted to area_2. If area_2 is unpromising, then again mutation must be applied to all the chromosomes in the population with high mutation probability to divert the search to a new area. It is likely that the new area may be an unexplored area like area_3 which may contain the solution or it can be an explored area like area_1. Searching an explored area again will increase the learning cost. As the knowledge obtained from the previous generation (i.e. area_1 is an unproductive area) is saved in the Knowledge Chromosome, it can be used to guide the mutation process so that searching an unpromising area repeatedly can be prevented.

To prevent the search from entering into a non productive area and to make a deep search in the area which has the potential solution, the proposed GA applies mutation with a varying mutation probability (P_m) which is based on the Knowledge Chromosome values. Mutation probability P_m ranges between Pl (lower bound) to P_u (upper bound). If the value of the chromosome gene is considered as unfit (i.e. its corresponding Knowledge Chromosome value is 0) then its gene mutation probability is P_u (maximum mutation probability) and, in turn, if Knowledge Chromosome value of the same gene is 1 then the mutation probability will be P_l (Minimum mutation probability). On the other hand, if its Knowledge Chromosome value is # then the mutation will be an adaptive mutation and its value P_m will be based on three quantities, namely the fitness value of the particular chromosome i (F_i), the overall fitness of the population (FP), the average fitness of the population (F_{avg}) and the fitness value of a chromosome in the population whose fitness value is the maximum ($FC_{\max}$). If the fitness F_i is less than the average fitness (F_{avg}) of the population then it is mutated with high probability, and if it is greater than the average fitness of the population it is mutated with a lower probability. The calculation of F_i, FP and F_{avg} is described in Section 4.4.1.

Let V_i be the value the knowledge chromosome of a particular chromosome gene. Then

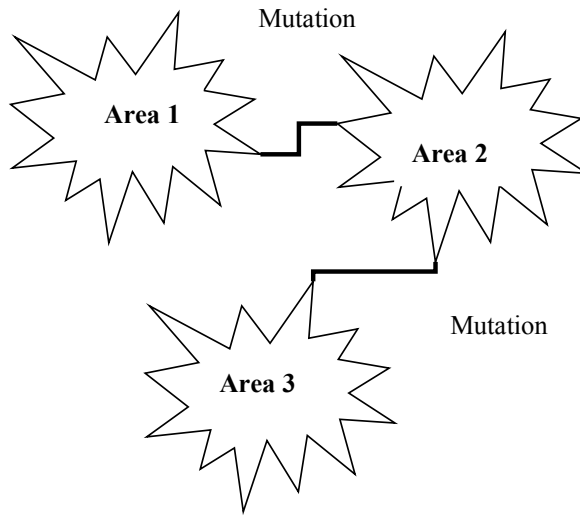

Fig. 1

if $V_i = 0$ then $P_m = P_u$
if $V_i = \#$ then
    if $F_i > F_{\text{avg}}$ then $P_m = P_l + P_u \frac{FC_{\max} - F_i}{FC_{\max} - F_{\text{avg}}}$
    else $P_m = P_u$ if $F_i \leq F_{\text{avg}}$
if $V_i = 1$ then $P_m = P_l$

Mutation probability range $[P_l, P_u]$ is fixed by the user. The normal range can be $[0.01, 0.5]$.

4.4.4 Insert and Remove Operators

Insert and Remove operators [30, 31] are used to control the size of the rule. Insert operator activates the gene of a chromosome by setting its flag bits which are not set with a varying probability P_i and Remove operator deactivates a gene of a chromosome by resetting the flag bits which are set with a varying probability P_r. Both the probabilities ranges from 0 to 0.3 based on the number of genes that take part in the rule.

4.4.5 Elitism Selection

After every generation, Best Elite percentage (5 %) of chromosomes is considered as elite and is copied to the next generation unaltered.

4.5 Knowledge Injection Function

After completion of an iteration of the genetic process, it has to be ascertained that the search process is examining an area where there is high possibility of finding an optimal solution and the search must also be made to avoid the areas which have been explored already and found unfit. For this purpose the following two operations are performed based on the Knowledge Chromosome values.

1. If the value of an attribute of the Knowledge Chromosome of a class is 0 and if the attribute of some chromosomes of the sub population of that class has that value then all those attribute values are compulsorily mutated to another value.
2. If the value of an attribute in the Knowledge Chromosome is 1 and no corresponding chromosomes attribute in the sub population of the class has that value, then the particular value is compulsorily injected into the population by altering sufficient percentage of low fitness chromosomes corresponding attribute gene to have that value.

5 SIMULATION

5.1 Description of the Dataset

The simulation was performed using the 23 data sets obtained from the UCI machine repository (`http://www.ics.uci.edu/`). These data sets are normally used as benchmark data sets for evaluating the algorithms that perform classification task. The data sets are given in Table 8. The continuous attributes of the data sets are discretized into 5 equal bins for experimental purpose.

5.1.1 Settings and Parameters

The experiments have been performed on a Pentium 4 (256 MB main memory) with Windows XP as the operating system, and the proposed Intelligent Genetic Algorithm (IGA) was developed using Java. Some related parameters are fixed as follows: $P_c = 0.8$, $P_m = 0.2$, $P_i = [0, 0.3]$, $P_r = [0, 0.3]$, $W_1 0.7$, and $W_2 0.3$. The population size for each class is 50 chromosomes. The experiment is repeated until there is no improvement in the fitness for 10 generations.

5.1.2 Experiment 1

The results of IGA are compared with the recent Michigan style classification GA (CGA) proposed by Xian-Jun Shi et al. [4] which is a typical example for common classification methods that are used to mine large data sets in the data mining domain. The fitness of the algorithm is calculated as described in Section 4.4.1 and its parameters are set similar to IGA implementation (Section 5.1.2) so that performance of both algorithms can be compared The results are also compared with

the standard state of art classification algorithms like C4.5 decision tree algorithm and Nave Bayes (NB) [32] classifier algorithm.

5.1.3 Experiment 2

To check the parallel implementation performance of the proposed algorithm the parallel implementation of the proposed method is done using four processors and its performance is compared with PV_MINER [18] algorithm which is a famous GA based classification algorithm for data mining tasks. PV_MINER is also implemented by using a parallel setup with 4 processors and the settings and parameters for both algorithms are similar to those described in Experiment 1. In the proposed method the data set is divided into four equal parts and distributed to all the four processors for GA process. After completion of the process the rules generated are again distributed to all the processors for fitness calculation.

Index	Dataset name	Number of instances	Number of attributes	Number of classes
1	Network intrusion	49 270	41	5
2	USPS Data	9 298	256	10
3	Nursery	12 960	8	5
4	Solar Flare	1 389	10	6
5	Yeast Database	1 484	8	10
6	Car	1 728	6	4
7	Image segmentation	2 310	19	7
8	Thyroid	3 772	28	4
9	Page blocks	5 473	10	5
10	Optical Digits	5 620	64	10
11	Satimage	6 435	36	6
12	Isolet Spoken Letter	7 797	617	26
13	LED Display	10 000	7	10
14	Waveform	10 000	21	3
15	Pen Digits	10 992	16	10
16	Australian sign language	12 546	8	3
17	Letter	20 000	16	26
18	Poker	25 000	10	10
19	Chess (King RootKing)	28 056	6	18
20	Shuttle	58 000	9	7
21	Connect-4	67 557	42	3
22	Adult	48 842	14	2
23	German credit data	1 000	20	2

Table 8. Data set description

		Average Classification Error (%)			
Index	DataSet name	C4.5	NB	CGA	IGA
1	Network intrusion	2.3	2.4	3.1	2.1
2	USPS Data	26.4	24.1	21.9	21.2
3	Nursery	9.0	9.1	13.2	11.4
4	Solar Flare	10.2	7.5	9.0	7.2
5	Yeast Database	31.9	31.77	34.9	33.2
6	Car	5.2	6.2	6.4	7.2
7	Image segmentation	22.7	18.6	21.1	21.8
8	Thyroid	5.0	3.2	3.7	3.4
9	Page blocks	12.0	11.2	11.3	10.5
10	Optical Digits	18.6	16.7	12.8	11.8
11	Satimage	14.6	13.1	10.1	10.8
12	Isolet Spoken Letter	22.9	21.4	15.6	15.2
13	LED Display	11.6	11.5	12.6	12.3
14	Waveform	12.3	11.2	8.8	8.0
15	Pen Digits	12.4	11.8	12.6	11.5
16	Australian sign language	11.4	10.6	11.3	11.1
17	Letter	12.7	12.1	11.6	11.0
18	Poker	52.5	48.0	49.5	44.1
19	Chess	32.2	31.0	30.6	29.6
20	Shuttle	4.2	4.6	7.1	5.8
21	Connect-4	27.9	26.1	24.5	23.8
22	Ipums-la-99	16.4	12.2	18.1	17.9
23	German Credit data	19.2	18.1	19.1	18.4

Table 9. Classification error

Method	C4.5	NB	CGA	IGA
C4.5	0/0/23	4/19/0	8/15/0	6/17/0
NB	**19/4/0**	0/0/23	**15/8/0**	10/13/0
CGA	**15/8/0**	8/15/0	0/0/23	2/21/0
IGA	**17/6/0**	**13/10/0**	**21/2/0**	0/0/23

Table 10. Win/lose/tie records of rival algorithms with regard to their learning efficiency across 23 datasets

5.2 Performance

To compare the average classification error of all four algorithms described in Experiment 1, a 10-fold validation test is performed. The training data is divided into 10 parts of which 9 parts are used to generate the classification model and one part is used as test data. The average of the results is shown in Table 9. The win/lose/tie (w/l/t) record is calculated for each pair of the methods for which the experiment is performed; it represents the number of data sets in which an algorithm, wins, loses or ties when compared with the other algorithm regarding accuracy. Such

		Total Run Time (s)	
Index	Data Set name	CGA	IGA
1	Network intrusion	3 940	1 660.3
2	USPS Data	835.2	350.6
3	Nursery	315.6	175
4	Solar Flare	125.7	75.8
5	Yeast Database	126.6	73.0
6	Car	105.8	65.52
7	Image segmentation	219.5	93.8
8	Thyroid	355.6	160.8
9	Page blocks	435.3	240.8
10	Optical Digits	555.5	225.6
11	Sat image	610.8	300.6
12	Isolet Spoken Letter	855.6	425.0
13	LED Display	750.4	390.0
14	Waveform	850.5	415.14
15	Pen Digits	875.2	420.2
16	Australian sign language	1 010.4	575.44
17	Letter	2 040.2	1 020.4
18	Poker	2 120.1	1 075.4
19	Chess (King Root King)	2 115.2	1 025.4
20	Shuttle	4 628.1	2 525.94
21	Connect-4	6 355.1	3 080.28
22	Adult	1 136.1	480.4
23	German Credit data	118.6	65.64

Table 11. Run time

(w/l/t) record is calculated for all algorithms with respect to run time. A two-tailed binomial sign test can be applied to wins versus losses. If its result is less than the critical level of 0.05, the wins against losses are statistically significant and the winning algorithm has advantage over the loser. The statistical w/l/t records for classification error and run time for all the methods are listed in Tables 9 and 10, respectively.

From the results in Tables 9 and 10, it can be identified that the error rated of the proposed methods is lower when compared with all other methods. Traditional algorithms like C4.5 NB perform well for data sets with lower number of attributes and classes. Standard classification GA and proposed Intelligent GA performance is far better for complex data sets with more attributes and class labels. Performance of the proposed method is also better when compared with the standard classification GA. This is due to the fact that the proposed method extracts the domain knowledge during every iteration which helps understand the search domain characteristic fully and the proposed method performs search only in areas with good solution which will generate rules with higher accuracy.

		Average Classification Error (%)		Total Run Time (s)	
Index	DataSet name	PV_MINER	IGA	PV_MINER	IGA
1	Network intrusion	3.1	2.1	2 626.6	1 106.8
2	USPS Data	23.9	21.2	556.8	233.7
3	Nursery	13.2	11.4	83.7	36.6
4	Solar Flare	9.0	7.2	77.1	30.5
5	Yeast Database	34.9	33.2	77.7	23.3
6	Car	6.4	6.2	83.8	30.3
7	Image segmentation	21.1	21.8	146.3	62.5
8	Thyroid	3.7	3.8	237.0	107.2
9	Page blocks	11.3	10.5	290.2	147.2
10	Optical Digits	12.8	11.8	370.3	190.4
11	Sat image	10.1	10.8	407.2	207.0
12	Isolet Spoken Letter	15.6	15.2	570.4	283.3
13	LED Display	12.6	12.3	500.2	253.3
14	Waveform	8.8	8.0	567	290.0
15	Pen Digits	12.4	11.5	583.4	313.46
16	Australian sign language	11.3	11.1	673.6	386.9
17	Letter	11.9	11.0	1 360.1	713.6
18	Poker	51.2	44.1	1 413.4	716.9
19	Chess (King Root King)	30.6	29.6	1 410.1	750.2
20	Shuttle	7.1	5.8	3 085.4	1 483.9
21	Connect-4	27.5	25.8	4 236.7	2 053.5
22	Ipums-la-99	18.3	17.9	757.4	320.2
23	German Credit data	19.1	18.4	99.0	37.0

Table 12. Comparison of parallel execution of the proposed method with PV_MINER

The run time of the proposed method and its counterpart CGA is presented in Table 11. The proposed method is faster than its counterpart by almost 80 % to 130 %. For complex domains with large attributes and classes the proposed method converges much faster. This is again due to the fact that the search is guided and not random. Parallel implementation of the proposed method and the PV_MINER algorithm is compared in Table 12. The results show that, although implemented in parallel the proposed method will have good accuracy and run time when compared to its counterpart.

6 CONCLUSION

Genetic Algorithm is popularly used for classification rule discovery in data mining. The learning cost associated with the rule discovery process is very high. This is due to the fact that except the knowledge embedded in the gene of a chromosome no other domain knowledge is used by GA in the search process. In this paper it

has been demonstrated that GA can extract considerble domain knowledge from the training data set between generations. An efficient way of storing the gained knowledge and utilizing it in the subsequent generations has been proposed in this paper and this makes the GA process to converge quickly. It has been proved experimentally that learning cost of the proposed algorithms is lower than that of the state-of-the-art GA based algorithms and its accuracy is comparable with other classification methods. In future, attempts will be made to make GA much faster and use it to mine large data sets like data streams.

REFERENCES

[1] Noda, E.—Freitas, A. A.—Lopes, H. S.: Discovering Interesting Prediction Rule With a Genetic Algorithm. Proceedings of the 1999 Congress on Evolutionary Computation, Volume 2.

[2] Dehuri, S.—Mall, R.: Predictive and Comprehensible Rule Discovery Using a Multi-Objective Genetic Algorithm. Knowledge Based Systems, Vol. 19, 2006, pp. 413–421.

[3] Guan, S. U.—Zhu-Collard, F.: An Incremental Approach to Genetic-Algorithms Based Classification. IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol. 35, 2005, No. 2, pp. 227–239.

[4] Shi, X.-J.—Lei, H.: A Genetic Algorithm-Based Approach for Classification Rule Discover. Proceedings of 2008 IEEE International Conference on Information Management, Innovation Management and Industrial Engineering, pp. 175–178.

[5] Kwedlo, W.—Kretowski, M.: Discovery of Decision Rules from Databases: An Evolutionary Approach. Second European PKDD '98 Symposium, Nantes, France, pp. 23–26.

[6] Yang, L.—Widyantoro, D. H.—Ioerger, T.—Yen, J.: An Entropy-Based Adaptive Genetic Algorithm for Learning Classification Rules. Proceedings of the 2001 Congress on Evolutionary Computation, pp. 790–796.

[7] Freitas, A. A.: A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery. In: A. Ghosh and S. Tsutsui (Eds.): Advances in Evolution Comput., Springer-Verlag 2002, pp. 819–845.

[8] Bacardit J.—Krasnogor, N.: Performance and Efficiency of Memetic Pittsburgh Learning Classifier Systems. Evolutionary Computation (EC), Vol. 17, 2009, No. 3, pp. 307–342.

[9] Bacardit, J.—Krasnogor, N.: Smart Crossover Operator With Multiple Parents for a Pittsburgh Learning Classifier System. In GECCO 06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM Press 2006, pp. 1441–1448.

[10] Holland, J. H.: Adaptation in Natural and Artificial Systems. University of Michigan Press 1975.

[11] De Jong, K. A.—Spears, W. M.: Learning Concept Classification Rules Using Genetic Algorithms. Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann 1991, pp. 651–656.

[12] Wilson, S. W.: Classifier Fitness Based on Accuracy. Evolutionary Computation, Vol. 3, 1995, No. 2, pp. 14–17.

[13] Freitas, A. A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag 2002.

[14] Janikow, C. Z.: A Knowledge-Intensive Genetic Algorithm for Supervised Learning. Machine Learning, Vol. 13, 1993, No. 2-3, pp. 189–228.

[15] Aguilar-Ruiz, J. S.—Riquelme, J. C.—Toro, M.: Evolutionary Learning of Hierarchical Decision Rules. IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol. 33, 2003, No. 2, pp. 324–331.

[16] Bacardit, J.—Krasnogor, N.: Empirical Evaluation of Ensemble Techniques for a Pittsburgh Learning Classifier System. 9th International Workshop on Learning Classifier Systems (IWLCS 2006), Springer, Lecture Notes in Artificial Intelligence (2006).

[17] Bacardit, J.—Krasnogor, N.—Biohe, N.: Bioinformatics-Oriented Hierarchical Evolutionary Learning. Nottingham Reprints, University of Nottingham 2006.

[18] Araujo, D. L. A.—Lopes, H. S.—Freitas, A. A.: A Parallel Genetic Algorithm for Rule Discovery in Large Databases. Proc. IEEE Systems, Man and Cybernetics Conference, Volume 3, Tokyo 199, pp. 940–945.

[19] Wilson, R.: Scalable Parallel Genetic Algorithms. Artificial Intelligence Review, Vol. 16, 2004, pp. 153–168.

[20] Giráldez, R.—Aguilar-Ruiz, J. S.—Santos, J. C. R.: Knowledge-Based Fast Evaluation for Evolutionary Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C, Vol. 35, 2005, No. 2, pp. 254–261.

[21] Bacardit, J.: Pittsburgh Genetics-Based Machine Learning in the Data Mining Era: Representations, Generalization, and Run Time. Ph. D. thesis, Ramon Llull University, Barcelona, Spain 2004.

[22] Bacardit, J.—Stout, M.—Hirst, J. D.—Sastry, K.—Llorà, X.—Krasnogor, N.: Automated Alphabet Reduction Method with Evolutionary Algorithms for Protein Structure Prediction. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM Press 2007, pp. 346–353.

[23] Venturini, G.: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts. In: Brazdil, P. B. (Ed.): Machine Learning: ECML-93 – Proc. of the European Conference on Machine Learning, Springer-Verlag, Berlin, Heidelberg 1993, pp. 280–296.

[24] Bacardit, J.—Burke, E. K.—Krasnogor, N.: Improving the Scalability of Rule-Based Evolutionary Learning. Memetic Computing, Vol. 1, 2009, No. 1, pp. 55–67.

P. Vivekanandan is currently working as a Professor in Department of Computer Science and Engineering, Park College of Engineering and Technology, Coimbatore, Tamilnadu, India. He has more than twelve years of teaching experience. He obtained his B. E. (computer science and engineering) from Bharathiar University, Coimbatore, India and his M. Tech (distributed computing systems) from Pondicherry University, Pondicherry, India. At present he is also a research scholar of Anna University, India. His research interests include knowledge discovery and data mining, soft computing and distributed computing. He has published many research papers in national/international conferences and journals. He has attended several seminars and workshops in the past ten years. He has also organized several symposiums and workshops. He has guided more than 20 UG projects. He is a life member of ISTE and also a member of Computer Society of India.

M. Rajalakshmi received her B. E. (CSE) degree from Bharathiar University, M. E. (CSE) degree from PSG College of Technology and PhD. from Anna University, Chennai. She is working as an Associate Professor in the Department of Computer Science and Engineering & Information Technology at Coimbatore Institute of Technology, Coimbatore, India. She has 18 years of teaching and research experience. Her areas of interest include data mining, distributed computing, data structures and algorithms and database management systems. She is a life member of ISTE. She guided many B.E and M. E. projects.

R. Nedunchezhian is currently working as Professor and Head of Department of IT, Sri Ramakrishna Engineering College, Coimbatore. Previously he worked as the Vice-Principal of Kalaignar Karunanidhi Institute of Technology, Coimbatore. He also served as Research Coordinator of the Institute and Head of Computer Science and Engineering Department (PG) at Sri Ramakrishna Engineering College, Coimbatore. He has more than 20 years of experience in research and teaching. He has obtained his B. E., M. E. and Ph. D. degrees in computer science and engineering. He has guided numerous UG, PG and M. Phil projects and conducted several sponsored conferences and workshops funded by private and government agencies. Recently he has obtained funding from AICTE for conducting research in data mining. He is guiding Ph. D. scholars of Anna University, Bharathiar University and Manonmaniam Sundaranar University. His research interests include knowledge discovery and data mining, soft computing, distributed computing, and information security. He has published 2 books, 50 research papers in international journals, 13 research papers in international conferences and 10 in national conferences. He has produced two Ph. D.s, and one more scholar has submitted her thesis recently. He is a Life Member of Advanced Computing and Communication Society and Indian Society for Technical Education. He is a reviewer for several international journals/conferences.