Using real-valued genetic algorithm to evolve rule sets for classification

Conference Paper · July 1994

DOI: 10.1109/ICEC.1994.350030 · Source: IEEE Xplore

2 authors, including:

Sandip Sen
University of Tulsa
276 PUBLICATIONS **4,263** CITATIONS

SEE PROFILE

Using Real-Valued Genetic Algorithms to Evolve Rule Sets for Classification *

Arthur L. Corcoran Sandip Sen

Abstract— **In this paper, we use a genetic algorithm to evolve a set of classification rules with real-valued attributes. We show how real-valued attribute ranges can be encoded with real-valued genes and present a new uniform method for representing don't cares in the rules. We view supervised classification as an optimization problem, and evolve rule sets that maximize the number of correct classifications of input instances. We use a variant of the Pitt approach to genetic-based machine learning system with a novel conflict resolution mechanism between competing rules within the same rule set. Experimental results demonstrate the effectiveness of our proposed approach on a benchmark wine classifier system.**

I. Introduction

Genetic algorithms (GAs) have proved to be robust, domain independent mechanisms for numeric and symbolic optimization[7]. Our previous work has demonstrated effective genetic-based rule learning in discrete domains [13]. In the real world, however, most classification problems involve real-valued features. To develop a classification mechanism that can be applied to a wide variety of real-world problems, in this paper, we combine ideas from our previous work with recent advances in real-valued parameter optimization [6, 9] in GAs to build classification systems.

The physical world abounds with problems that require classification of observed instances into a set of target classes. The goal of a supervised classification system is to generate a set of rules from a set of preclassified training instances, which can then be used to classify future instances. An accurate rule induction mechanism improves quality of classification and greatly reduces the cost of providing such classification. Application domains include medical diagnosis, fault detection in electro-mechanical devices, evaluating credit applications, predicting crop quality, etc.

*This research has been partially supported by OCAST Grant AR2-004 and Sun Microsystems, Inc. Both authors are with The Department of Mathematical and Computer Sciences, The University of Tulsa, 600 South College Avenue, Tulsa, Oklahoma 74104-3189.

In this paper, we formulate the supervised classification problem as a function optimization problem. The goal of the optimization problem is to develop a rule set that maximizes the number of correct classifications of training set instances. A GA is used to evolve structures representing sets of classification rules. Each rule in a rule set matches only a subset of all possible input instances, while the set of rules together is expected to cover the whole input space. Whereas previous approaches to using GAs for evolving rule sets for supervised classification rely on the assumption of discrete domains [2], we are particularly interested in the more difficult problems involving continuous variables. Since real genes lead to an explosion of the search space over binary genes, we believe it is a challenge and a contribution to show that GAs can be used to develop rule sets with continuous variables.

This paper is organized as follows: Section II provides an introduction to genetic-based machine learning systems. Section III describes how genetic algorithms can be used to optimize the classification rule set. This includes how the problem can be encoded into a representation suitable for the genetic algorithm and how the fitness function is designed. Section IV contains experimental results which were obtained when a genetic algorithm was used to evolve a continuous classification rule set for a benchmark problem. Finally, Section V presents a brief summary and conclusions based on our results.

II. Genetic-Based Machine Learning Systems

Genetic-Based Machine Learning (GBML) systems are rule-based systems which determine class membership of input instances from a set of attributes. That is, a particular instance is examined based on several attributes. The attributes values could have been derived from laboratory test results or from environmental sensor readings. A classifier system matches the set of attributes corresponding to an instance against a set of rules to determine the class membership of the instance. These domain-independent classification mechanisms are particularly useful in problem domains for which there is no known precise model to determine the class, or for which determining a precise model is impractical.

There are two principal schools of thought in designing

GBML systems [3]. The first school of researchers proposes to use genetic algorithms to evolve individual rules, a collection of which comprises the classification expertise of the system. This approach to building classifier systems was originally proposed by John Holland at the University of Michigan, and hence is referred to as the Michigan approach [8]. The other school of thought has been popularized by Ken De Jong and Steve Smith [14] from the University of Pittsburgh, and is therefore referred to as the Pitt approach to building classifier systems. In this approach, genetic algorithms are used to evolve structures, each of which represent a complete set of rules for classification. So, each structure in the population in the Pitt approach correspond to the entire set of rules in the Michigan approach.

While debate rages on which of these two is the better approach to building classifier systems, the Pitt approach seems to be better suited at batch-mode learning (where all training instances are available before learning is initiated) and for static domains, and the Michigan approach is more flexible to handle incremental-mode learning (training instances arrive over time) and dynamically changing domains. We have chosen the Pitt approach in this paper to take advantage of the availability of all training data before learning is initiated.

III. Proposed approach

In this section we present our approach to using GAs with real genes to evolve classification rule sets. As mentioned above, we use the Pitt approach of evolving rule sets by GAs. In the following, we first describe the semantics of real-valued genes as rule conditions, and then present the evaluation function for chromosomes in the GA population.

A. Encoding of the problem

The choice of an efficient representation is one of the most important issues in designing a genetic algorithm. In our GA, each chromosome represents a set of classification rules. Each of the classification rules is composed of a set of A attributes and a class value. Each attribute in the rule has two real variables which indicate the minimum and maximum in the range of valid values for that attribute. A *don't care* condition occurs when the maximum value is less than the minimum value. The class value can be either an integer or a real. This manner of representing don't cares allow us to eliminate the need for special don't care symbols, and also obviates the need to enforce the validity of rules after applying operators modifying the rule sets. When each attribute value in an input instance is either contained in the range specified for the corresponding attribute of a rule or the rule attribute is a don't care, the rule matches the instance and the class value indicates the membership class of the instance. We use a fixed-length chromosome for the GA, where each chromosome is made up of a fixed number (n) of rules encoded as above. The length of a chromosome is thus $n(2A + 1)$.

B. Fitness function design and conflict resolution

Given an appropriate encoding of the problem, the next step in the design of the genetic algorithm is the design of the fitness function. The fitness function evaluates a chromosome to determine its fitness relative to the other chromosomes in the population. The GA uses this information during reproduction to evolve better chromosomes over generations.

Since each chromosome in our representation comprises an entire classification rule set, the fitness function must measure the collective effectiveness of the rule set. A rule set is evaluated by its ability to correctly predict class memberships of the instances in a pre-classified training set. The fitness of a particular chromosome is simply the percentage of test instances correctly classified by the chromosome's rule set. Each rule in the rule set is compared with each of the instances. If the rule matches the instance, the rule's class prediction is compared with the actual known class for the instance. Since there is more than one rule, it is possible for multiple rules to match a particular instance and predict different classifications. When matching rules do not agree on classification, some form of conflict resolution must be used.

Conflict resolution is perhaps the most important issue to consider in the design of the objective function. We decided to use a form of weighted voting. A history of the number of correct and incorrect matches is maintained for each rule. Each matching rule votes for its class by adding the number of correct and incorrect matches to the corresponding variables for its class. The winning class is determined by examining the total number of correct and incorrect matches assigned to each class. The first objective is to select the class with the least number of incorrect matches by rules predicting that class. If there is a tie between two classes on this metric, the second objective is to select the class with the greatest number of correct matches. If there is still a tie between two or more classes, one of the classes is arbitrarily In either case, the winning class is compared with the actual known class. If they are the same, this contributes to the percentage of correct answers for this classifier set. Finally, the number of correct or incorrect matches is incremented appropriately for each of the matching rules. The above conflict resolution mechanism is a modification of that used in our previous work [13]. We use a democratic approach of pooling votes instead of using the class predicted by the best individual as determined by performance history.

C. Crossover and mutation

The purpose of recombination operators such as crossover and mutation in a genetic algorithm is to obtain new points in the search space. Since our chromosomes are composed of nonhomogeneous alleles, it is necessary to use modified versions of the standard operators. We designed three operators: modified simple (one point) crossover, creep mutation, and simple random mutation. Simple crossover was modified so that the random crossover point always falls on a rule boundary. Creep mutation is applied to each attribute's minimum and maximum value with a probability denoted as the *creep rate*. Attribute values which are to be mutated are incremented or decremented by the *creep fraction*, which is a fraction of the valid range for the attribute value. Since it is possible for the range to differ greatly between any two attributes, the amount of change applied may be quite different even though the creep fraction is the same. In addition, it is equally likely that the attribute value will be incremented or decremented, and the value is adjusted if necessary so it will fall in the valid range for the attribute. Note, creep mutation makes it possible for the range specified in the chromosome for the attribute to change from a normal range to a don't care or vice versa. Creep mutation is intended to occur often: on the order of half of the attribute values are creeped in each recombination. We chose a high frequency of the creep operator because of the complexity of searching a space consisting of a large number of real-valued genes. Finally, simple random mutation replaces randomly selected attribute range values or class values with random values from the appropriate valid range. As in creep mutation, simple random mutation can change valid range specifications to don't cares and vice versa. Unlike creep mutation, simple random mutation can also change the class values. Simple random mutation is intended to be applied infrequently: on the order of once per recombination.

D. Benchmark classification problem

We selected a suitable benchmark classification problem based on the large collection of test data at *The UCI Repository of Machine Learning Databases* [10]. The test data we selected was for wine classification [5]. This data consists of 178 instances, each consisting of 13 real-valued attributes and one of three classifications. The data has been obtained from chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. The chemical analysis determined the quantities of 13 constituents found in each of the three types of wine. The data contains 59 instances from the first class, 71 from the second class, and 48 from the third class. The source of the data is Forina *et al.* [4] and it has recently been used by Aeberhard *et al.* [11, 12].

Best	Average	Worst	Variance	Standard Deviation
100%	99.5%	98.3%	4.524e-05	6.726e-03

Table 1: Results obtained by the GA

Method	Classification Rate
GA (best)	100%
RDA	100%
GA (average)	99.5%
QDA	99.4%
LDA	98.9%
1NN	96.1%

Table 2: Results obtained using various methods

IV. Experimental Results

We developed a genetic algorithm for wine classification using LibGA [1]. LibGA allowed us to easily write our objective function and recombination operators using our problem encoding. Our GA used an elitist generational population model without any overlap of the populations. Fixed length chromosomes were used with 60 rules per chromosome rule set. Since each rule has two genes corresponding to each of 13 attributes and another gene for its class prediction, the length of a rule is 27. With 60 rules, this results in a total chromosome length of 1620. The population size was chosen to be 1500. Uniform random selection was used to reduce the selection pressure since roulette selection resulted in premature convergence (at the start of the run all rule sets performed very poorly as few rules matched any instance, we may use biased initialization to address this problem in our future work). Our modified simple (one point) crossover operator was used with crossover probability of one. Creep mutation was used with creep rate of 0.5 and creep amount of 0.01. This meant that half of the attribute values were modified in each recombination, and that the amount of creep was one percent of the span between the minimum and maximum values in the range. Simple random mutation occurred with a mutation rate of 1, which in LibGA meant one mutation per recombination.

Table 1 summarizes the results obtained by the genetic algorithm over ten independent runs. The numbers indicate the best, average, and worst classification percentage over the ten runs as well as the variance and standard deviation. Exactly half of the runs obtained a perfect 100% classification rate. Two of the runs obtained a 98.3% classification rate and the other three obtained a 99.4% classification rate. Thus, the average classification rate was 99.5%.

Table 2 summarizes the results that Aeberhard [12] ob-

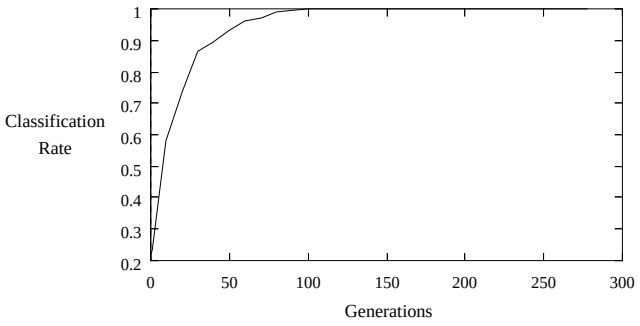

Figure 1: Convergence Profile for the GA

tained using various methods on the wine classification problem as well as the results obtained by the best and average genetic algorithm. All of the techniques that Aeberhard tested used the leave-one-out technique. Technique 1NN used Z-transformed data. Table 2 shows that our GA-based classification approach was able to cover the training set extremely well.

Figure 1 shows the convergence profile for one of the runs of the genetic algorithm. The figure plots classification rate by generation of the GA. In this case, the GA was able to obtain a perfect classification rate in less than 100 generations.

The best rule set developed by the GA has some interesting characteristics. Approximately a third of the rule set actually matches any input instance, and these rules are perfect in all but a couple of cases, i.e., they predict correctly whenever they match. This means that conflict resolution is not necessary once the best rule set has been evolved. Conflict resolution mechanism is, however, important to evaluate fitnesses of less than perfect rule sets, and as such is essential to prevent premature convergence of the GA. The two-thirds of the rule set that does not match any input are also necessary for the GA to search the space of classification rules. We observed significant decrease in performance with much smaller rule set sizes. The large percentage of non-matching rules in the best rule set, however, suggests the possible benefits of using variable size rule sets in the GA population.

V. Conclusions

In this paper, we have shown how a genetic algorithm can be used to evolve rule sets with real variables for supervised classification problems. Our GA used the Pitt approach to encode the rule set and a weighted voting scheme for conflict resolution. Attributes were represented in the rules by real valued ranges and don't cares were represented by a range where the minimum value was greater than the maximum value. We used three structure modification operators: modified simple (one point) crossover, creep mutation, and simple random mutation. We tested our GA on a benchmark wine classification problem. Our results show the GA is capable of developing an effective rule set to accurately classify almost all instances. This suggests the promise of using GAs to optimize rule sets for classification problems with real attributes.

In the future, we would like to extend our GA to use variable length rules and add a selection pressure to reduce the size of the rules. In this way we can obtain both compact and optimal rule sets. We plan to apply our approach to a number of different supervised classification problems and compare corresponding results with other promising machine learning approaches.

Acknowledgements

This research has been partially supported by OCAST Grant AR2-004. The authors also wish to acknowledge the support of Sun Microsystems, Inc. Special thanks to P. M. Murphy and D. W. Aha for making *The UCI Repository of Machine Learning Databases* available and to M. Forina for making the wine recognition data available.

References

[1] A. L. Corcoran and R. L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In E. Deaton, K. M. George, H. Berghel, and G. Hedrick, editors, *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–118, New York, 1993. ACM Press.

[2] K. A. De Jong and W. M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, pages 651–656, 1991.

[3] K. A. DeJong. Genetic-algorithm-based learning. In Y. Kodratoff and R. Michalski, editors, *Machine Learning, Volume III*. Morgan Kaufmann, Los Alamos, CA, 1990.

[4] M. Forina et al. Parvus - an extendible package for data exploration, classification and correlation. Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy, July 1991.

[5] M. Forina et al. Wine recognition database. Available via anonymous ftp from *ics.uci.edu* in directory */pub/machine-learning-databases/wine*, 1992.

[6] D. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, 5:139–168, 1991.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[8] J. H. Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell, and T. M. Mitchell, editors, *Machine Learning, an artificial intelligence approach: Volume II.* Morgan Kaufmann, Los Alamos, CA, 1986.

[9] J. D. Kelly and L. Davis. A hybrid genetic algorithm for classification. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 645–650, 1991.

[10] P. M. Murphy and D. W. Aha. *UCI Repository of Machine Learning Databases* [machine-readable data repository]. Maintained at The Department of Information and Computer Science, The University of California at Irvine, 1992.

[11] D. C. S. Aeberhard and O. de Vel. The classification performance of RDA. Technical Report 92-01, Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland, 1992. Also submitted to Journal of Chemometrics.

[12] D. C. S. Aeberhard and O. de Vel. Comparison of classifiers in high dimensional settings. Technical Report 92-02, Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland, 1992. Also submitted to Technometrics.

[13] S. Sen. Improving classification accuracy through performance history. In *Proc. 5th Intl Conf on Genetic Algorithms*, 1993.

[14] S. F. Smith. *A learning system based on genetic adaptive algorithms.* PhD thesis, University of Pittsburgh, 1980. (Dissertation Abstracts International, 41, 4582B; University Microfilms No. 81-12638).