Improving Genetic Classifiers with a Boosting Algorithm

Bo Liu
College of Computer and Information Engineering, Guangxi University, Nanning, China, 530004
ddxllb@yahoo.com.cn

Bob McKay, Hussein A.Abbass
School of Computer Science, ADFA, University of New South Wales, ACT 2600, Australia
rim@cs.adfa.edu.au , abbass@cs.adfa.edu.au

Abstract : This paper presents a boosting genetic algorithm for classification rule discovery. The method is based on the iterative rule learning approach to genetic classifiers. The boosting mechanism increases the weight of those training instances that are not classified correctly by the new rules, so that in the next iteration the algorithm focuses the search on those rules that capture the misclassified or uncovered instances. We show that the boosted genetic classifier has higher accuracy for prediction, or from an alternative and perhaps more important perspective, uses less computational resources for similar accuracy, than the original genetic classifier.

1 Introduction

Classification rule discovery is a process of extracting models and patterns for each class or category from a set of training examples, which can be used to classify future examples. A range of methods have been applied in classification, including decision tree induction, Bayesian theory, neural networks, rough set theory, genetic algorithms (GA) etc. Genetic algorithms are an optimization approach based on the mechanics of natural selection and genetics. Several genetic algorithm methods for data mining have been proposed in the literature [2][1].

Although genetic algorithms form an extremely powerful optimization technique, their efficacy is dependent on the ability to do a large number of evaluations in a reasonable amount of time. When the database contains a large number of examples with many features, it may take considerable execution time and require an enormous amount of memory to find a satisfactory solution. In addition, genetic algorithms can't guarantee to find an optimum solution.

Boosting is an ensemble learning method, i.e. it constructs a set of classifiers and then classifies new data by taking a weighted vote of their predictions. It can theoretically be used to reduce the error of any weak learning algorithm, which need only be a little bit better than random guessing [6]. So far, there has not been much research on using the boosting technique in evolutionary learning. To our knowledge, the method has been used to boost genetic programming in [6] and [7], and to generate fuzzy rules in [9] and [11].

In this paper, we apply the boosting algorithm in a GA classifier. Compared to other learning methods, the advantage of boosting is its proven ability to improve the accuracy of a weak learner; in the GA context, it permits speed-up through reducing the population size and number of generations. In other words, we can reduce the computational requirements for a GA to reach a given level of performance, by weakening the GA and using it within a boosting algorithm. The rest of the paper is organized as follow. Section 2 introduces the boosting algorithm. Section 3 describes the boosting genetic algorithm for classification. Section 4 presents experimental results obtained by the boosted GA classifier and the original GA classifier . Finally, section 5 draws some conclusions.

2 Boosting Algorithm

The boosting algorithm was proposed and developed by Freund and Schapire (1995, 1996, 1997, 1998). According to [10], boosting is a method of finding a highly accurate hypothesis by combining many "weak" hypotheses, each of which is only moderately accurate. It manipulates the training examples to generate multiple hypotheses. In each iteration, the learning algorithm uses different weights on the training examples, and it returns a hypothesis h_t. The weighted error of h_t is computed and applied to update the weights on the training examples . The result of the change in weights is to place more weight on training examples that were misclassified by h_t, and less weight on examples that were correctly classified. The final classifier is constructed by weighted vote of the individual classifiers.

Our method is based on the ADABOOST.M1 algorithm proposed by Freund and Schapire in [3]. ADABOOST iteratively generates a robust final hypothesis by giving increased weight to mis-classified training samples from previous learning rounds. An algorithm is given in Figure 1. Ordinarily, the distribution D will be set to be uniform initially, so that D(i)=1/N. The algorithm maintains a set of weights w^t over the training examples. On iteration t, a distribution p^t is computed by normalizing these weights. This distribution is fed to the weak learner, which generates a hypothesis h_t, that assigns one of the k possible labels to each. The error of h_t is computed by $\| h_t(x_i) \neq y_i \|$, where $\| h_t(x_i) \neq y_i \|$ is 1 if $h_t(x_i) \neq y_i$ and 0

otherwise. The final hypothesis h_f, for given instances x, outputs the label y that maximizes the sum of the weights of the weak hypotheses predicting that label. In AdaBoost, a key step is choosing a new distribution on the training examples based on the old distribution and the errors made by the present weak hypothesis.

Input: sequence of N examples $< (x_1,y_1),\ldots,(x_N,y_N) >$ with labels $y_i \Gamma Y=\{1,\ldots, k\}$;
distribution D over the examples;
weak learning algorithm WeakLearn;
number of iterations T.

Initialize the weight vector: $\Gamma=D(i)$ for i=1,…,N.

Do for t=1,2,…,T

1.Set

2.Call WeakLearn, providing it with the distribution p^t;
get back a hypothesis h_t: X→Y.

3.Calculate the error of h_t: $\epsilon_t=\Gamma$
If $\epsilon_t>1/2$, then set T=t-1 and abort loop.

4.Set $\beta_t= \epsilon_t/(1- \epsilon_t)$.

5.Set the new weights vector to be

Output the hypothesis

$h_f(x)=\arg$

Figure 1 Algorithm AdaBoost.M1

3 Boosting GA for Classification

3.1 GA for Classification Rule Discovery

In our work, we designed the genetic algorithm for classification rule discovery shown in Figure 2. In the GA system, each population consists of a number of genotypes, and each genotype corresponds to a chromosome, which includes a set of strings. Each string represents a rule, for example, for Boolean attributes, "If ((not A1) and A2) then C1" can be coded as "011". More generally, f an attribute has k values, $\log_2 k$ bits will be required to code the attribute.

The GA starts with some random populations of rules, and thereafter generates successive populations using the following basic operators: one-point crossover, bit mutation, and selection according to a fitness function. A population composed of a set of candidate rules is kept, and gradually improved by constructing new fitter rules until stopping criteria are satisfied. Finally, a solution is selected from the best population, having the maximum fitness value. The fitness function is defined as equation (1).

$$F=\frac{TP+TN}{number\ \ of\ \ training\ \ examples} \qquad (1)$$

Where

·*TP* is the number of examples covered by the rule that have the target class.

·*TN* is the number of examples that are not covered by the rule and that do not have the target class.

3.2 Boosting GA Classifier

The aim of boosting is to take a weak learner and, through iteration, produce a strong learner. Of course, it would be computationally wasteful to simply apply boosting to a computationally expensive, but relatively strong, learner such as a GA. The aim of this work, then, is to investigate whether reducing the computational cost of GAs by reducing their population size and number of generations (thus producing a weak learner), and combining them with boosting, represents a useful trade-off. The overall

procedure of the boosting GA classifier is depicted in figure 3.

The boosting genetic algorithm is detailed in Figure 4. The function of boosting is to repeatedly apply a weak learning algorithm on various distributions of the training data and to aggregate the individual classifiers into a single overall classifier. Initially, all training examples are uniformly weighted. After each iteration, the distribution of training examples is changed, based on the error that the current classifier exhibits on the training set. The weight w(i) specifies the relative importance of the i-th training example

For each target-class
Generate initial populations of rule conditions at random
Repeat
Evaluate each population on the training set using a fitness function
Applying genetic operators to generate new populations，such as
Select the best in population P_i to P_j;
Crossover P_i individuals with P_j individuals;
Mutation P_i
Until number of generations
Generate rule set from the best population
End for

Figure 2 GA for Classification Rule Discovery

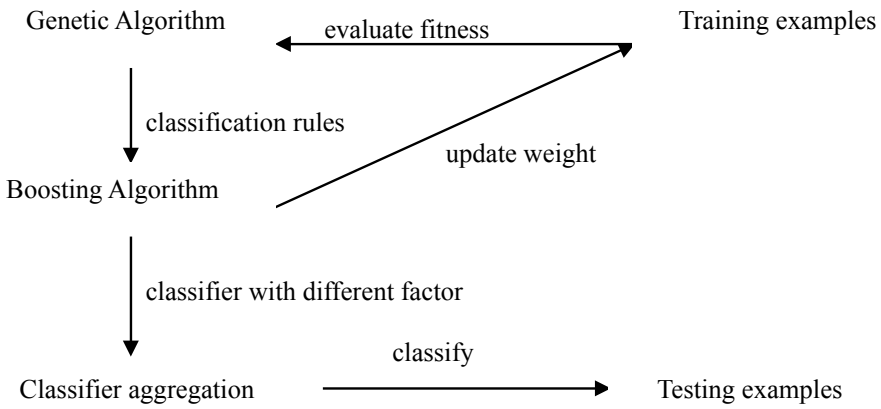


Figure 3 Procedure of Boosting GA Classifier

The fitness function in GA(S, D_t) in Figure 4 is different from that depicted in equation (1). The fitness function considers the distribution of each training example, as defined in equation (2).

$$F=\frac{\sum_{k\in TPSet} D_t(k)+\sum_{k\in TNSet} D_t(k)}{\sum_{k\in S} D_t(k)} \quad (2)$$

Where

·*S* is training set;

·*TPSet* includes instances covered by the rule that have target class;

·*TNSet* includes instances that are not covered by the rule and that do not have the target class.

4 Experiment Results

We conducted experiments on two data sets from the UCI repository [5]. One is the Wisconsin breast cancer database, which contains 699 instances, 9 integer-valued attributes and 2 classes (malignant and benign). Instances containing missing values were omitted from the experiment. The other is the Tic_tac_toe endgame database, which contains 958 instances, 9 numeric-valued attributes and 2 classes (won and lost). These learning problems have been widely studied, and a number of algorithms have achieved high performance on them – higher than the results reported in this paper. However the aim of this paper is to understand the effects of combining GAs with boosting, rather than to achieve optimality on these particular datasets, on the basis that the relative

performance would be expected to extend to more difficult problems on which GAs relatively well.

In the experiments, each database is divided into 10 mutually exclusive and exhaustive partitions. In each experiment, a different partition is used as the test set and the other nine partitions are used as the training set (i.e. 10-fold cross-validation)..

The evolutionary parameter settings used in the experiment are listed in Table 1.

We use a training set to generate some rule sets, then predict the class of instances in an independent test set. Supposing Num_C is the total number of instances correctly classified and Num_UC is the total number of instances incorrectly classified, the accuracy is computed by means of the following formula:

$$\text{Accuracy} = \frac{Num_C}{Num_C + Num_{UC}} \qquad (3)$$

The average accuracy, on the test sets, is reported as the prediction accuracy of the discovered rule set. The results comparing the mean accuracy of GA and BoostGA are shown in Table 2.

Input: a training set S={(x_1, y_1), (x_2, y_2),…, (x_m, y_m)}; x_i ¿ X,
y_i ¿ Y={1, 2, …k};
GA(S,D) : a GA algorithm for classification using a distribution D on S;
T : rounds of boosting

Initialize the weight of instance (x_i, y_i): $w_1(i)=1/m$ for i=1 ,…, m.

For t=1, 2, …, T

$$D_t(i)= \frac{w_t(i)}{\sum_{i=1}^{m} w_t(i)}$$

Run GA(S,D_t)

Get a set of rules denoted as R_t: X→Y.

Calculate the error of R_t : $\epsilon(R_t)= \sum_{i=1}^{m} D_i(t) \| R_t(x_i) \neq y_i \|$

If $\epsilon(R_t) >1/2$ and k>2 then T=t-1 and abort loop

Else

Set $\beta(t)=\epsilon(R_t) / (1-\epsilon(R_t))$

Update the weight of each instance to be

$$w_{t+1}(i)=w_t(i)* \beta(t)^{1- \| R_t(x_i) \neq y_i \|}$$

End if

End for

Output the class of instance x_k

$$R_f(x_k)=\arg \max_{y \in Y} \sum_{t=1}^{T} \log(1/\beta(t)) * \| R_t(x_k) = y \|$$

Figure 4 The GABoost Algorithm for Classification

Table 1 GA and BoostGA Parameter Settings

Parameter	GA	BoostGA
Rounds of boosting	1	3
Number of generations	20	5
Population size	1000	500
Crossover rate	0.8	0.8
Mutation rate	0.1	0.1
Copy rate	0.1	0.1
Number of rules(for each class)	10	10

Table 2 Mean Prediction Accuracy of GA and BoostGA

	Accuracy		Evaluations	
Data set	GA	BoostGA	GA	BoostGA

Breast Cancer	93.28%	95.18%	20,000	7,500
Tic-tac-toe	73.13%	86.75%	20,000	7,500

The table shows that BoostGA generates a higher classification accuracy than GA. We also conducted one-tailed student's t-tests for ten group experiment results. The significance level is 0.04 for the Breast Cancer Database and 0.004 for the Tic-tac-toe. The difference of the two methods is statistically significant in both cases.

We can also compare computational requirements of the two methods. This is best measured by comparing the number of fitness evaluations required, since measurement of actual computational times can be dependent on external variables. The comparison is given in table 2. The measure is calculated by equation (4).

Evaluations = Rounds of boosting ¿ Number of generations ¿ Population size (4)

Table 3 Mean Classification Accuracy of GA Classifiers in Boosting

	Accuracy (best fitness)	
Rounds of boosting	Breast cancer	Tic-tac-toe
1	0.7233	0.6904
2	0.7566	0.6615
3	0.7748	0.6990

In BoostGA, we use boosting rounds to compensate for reducing the generations and populations in the GA. The result is that fewer than half the fitness evaluations (7,500 vs 20,000) are required to produce significantly better results.

Table 3 shows the best fitness in each round of boosting. It can be observed that the "weak" GA learner results in a rule set which has moderate accuracy (just greater than 0.6) in the boosting procedure, conforming to the requirement of the boosting algorithm.

Figure 5 shows an interesting phenomenon in choosing BoostGA parameters,

(a) for the Wisconsin breast cancer data set and
(b) for the Tic_tac_toe data set.

To explore the effects of different versions of the weak learner, we adjusted the number of generations (the other parameters are as set in Table 1), obtaining the corresponding accuracies. As the number of generations increases, although the best fitness of every generation is higher than in table 3, the correct prediction rate of the ensemble classifier is not increased.

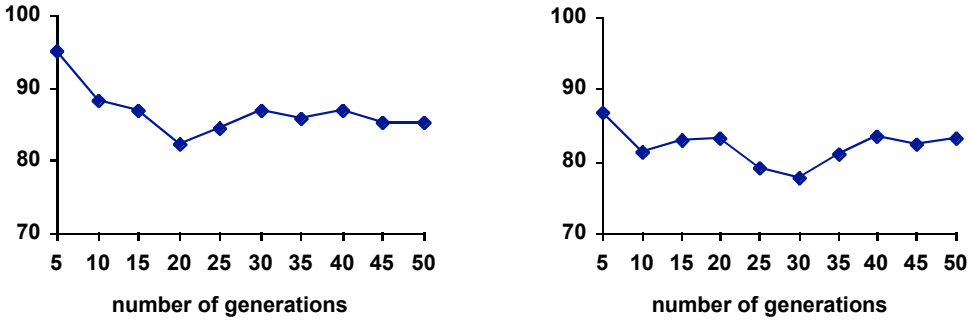
BoostGA experimental results from varying the boosting rounds from 1 to 10 (leaving the other parameters as in Table 1) are depicted in figure 6, and results varying the population size from 100 to 1000 (leaving the other parameters as in Table 1) are depicted in figure 7. We can conclude that it doesn't require many boosting rounds to obtain a significantly better accuracy, and that increasing the population size above those from Table 1 does not usefully affect the accuracy.

5 Conclusion

In the paper we applied boosting to genetic algorithms for rule discovery. It raises interesting issues not only in classification, but also in genetic algorithms. The method has been tested on publicly available databases. We compare the boosted algorithm with the original genetic algorithm used in rule discovery. Experimental results have demonstrated that the boosted genetic algorithm is able to classify the test set with higher accuracy and fewer fitness evaluations than the original genetic algorithm.

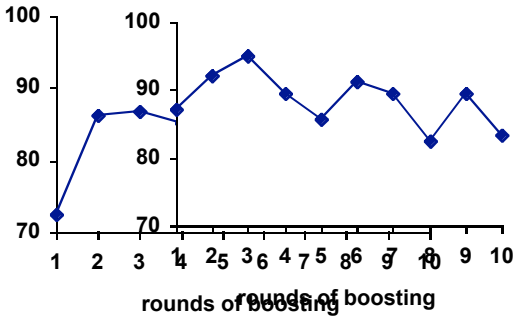
The proposed algorithm can also be used as a distributed classification technique for efficiently integrating specialized classifiers learned over distributed homogenous databases that cannot be merged at a single location.

Future work will include the application of the proposed algorithm to real-world databases, and the application of boosting to parallel evolutionary system.



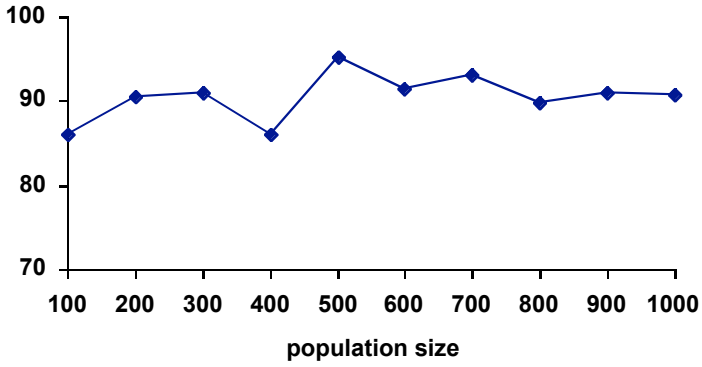
(a) (b)

Figure 5 Test Set Accuracy (%) versus Different Number of Generations in BoostGA

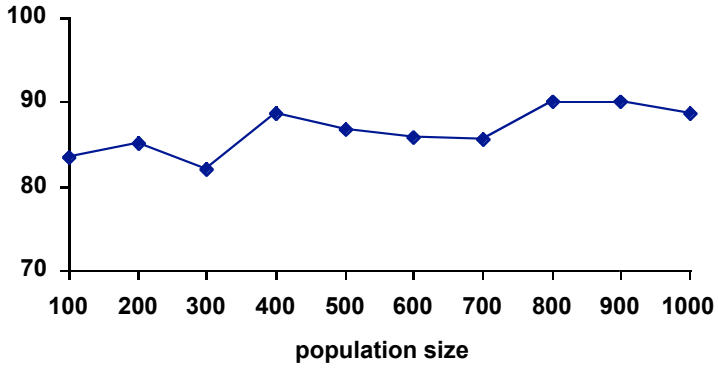


(a) **(b)**

Figure 6 Test Set Accuracy(%) versus Number of Rounds in BoostGA



(a)



(b)

Figure 7 Test Set Accuracy (%) versus Population Sizes in BoostGA

Bibliography

1. J.L.Álvarez, J.Mata, J.C.Riquelme, Mining Interesting Regions Using an Evolutionary Algorithm, Proceedings of the 17th Symposium on Applied Computing , (2002) 498-502
2. Siddhartha Bhattacharyya, Evolutionary Algorithms in Data Mining:Multi-Objective Performance Modeling for Direct Marketing, Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (2000) 465-473
3. Yoav Freund, Robert E.Schapire, A Decision-theoretic Generalization of On-line Learning and an Application to Boosting, Tech.rep., AT&T Bell Laboratories, Murray Hill, NJ, (1995)
4. Yoav Freund, Robert E.Schapire, Experiments with a New Boosting Algorithm, Proceeding of 13th international Conference on Machine Learning, (1996) 148-156
5. Hettich, S. and Bay, S.D., The UCI KDD Archive, http://kdd.ics.uci.edu. (1999)
6. Hitoshi Iba, Bagging, Boosting, and Bloating in Genetic Programming, Proceedings of the Genetic and Evolutionary Computation Conference, volume 2, (1999) 1053-1060
7. Gregory Paris, Denis Robilliard, and Cyril Fonlupt: Applying Boosting Techniques to Genetic Programming, Proceeding of the 6th Intelligent Artificial Evolution, (2001) 315-326.
8. Robert E.Schapire, Using Output Codes to Boost Multiclass Learning Problems, Proceedings of the Fourteenth International Conference on Machine Learning, (1997) 313-321
9. L.Junco, L.Sanchez, Using the Adaboost Algorithm to Induce Fuzzy Rules in Classification Problems, Proceeding of Spanish Conference for Fuzzy Logic and Technologies, (2000) 297-301
10. Robert E.Schapire, Yoram Singer: Improved Boosting Algorithms Using Confidence-rated Predictions, Proceedings of the 11th Annual Conference on Computational Learning Theory, (1998) 80-91

Frank Hoffmann, Boosting a Genetic Classifier, IFSA/NAFIPS, (2001)