Using Genetic Algorithms for Data Mining Optimization in an Educational Web-Based System

Behrouz Minaei-Bidgoli and William F. Punch

Genetic Algorithms Research and Applications Group (GARAGe)
Department of Computer Science & Engineering
Michigan State University
2340 Engineering Building
East Lansing, MI 48824
{minaeibi,punch}@cse.msu.edu
http://garage.cse.msu.edu

Abstract. This paper presents an approach for classifying students in order to predict their final grade based on features extracted from logged data in an education web-based system. A combination of multiple classifiers leads to a significant improvement in classification performance. Through weighting the feature vectors using a Genetic Algorithm we can optimize the prediction accuracy and get a marked improvement over raw classification. It further shows that when the number of features is few; feature weighting is works better than just feature selection.

1 Statement of Problem

Many leading educational institutions are working to establish an online teaching and learning presence. Several systems with different capabilities and approaches have been developed to deliver online education in an academic setting. In particular, Michigan State University (MSU) has pioneered some of these systems to provide an infrastructure for online instruction. The research presented here was performed on a part of the latest online educational system developed at MSU, the *Learning Online Network with Computer-Assisted Personalized Approach* (*LON-CAPA*).

In LON-CAPA[1], we are involved with two kinds of large data sets: 1) educational resources such as web pages, demonstrations, simulations, and individualized problems designed for use on homework assignments, quizzes, and examinations; and 2) information about users who create, modify, assess, or use these resources. In other words, we have two ever-growing pools of data.

We have been studying data mining methods for extracting useful knowledge from these large databases of students using online educational resources and their recorded paths through the web of educational resources. In this study, we aim to an-

[1] See http://www.lon-capa.org

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 2252–2263, 2003.

swer the following research questions: Can we find *classes* of students? In other words, do there exist groups of students who use these online resources in a *similar* way? If so, can we identify that class for any individual student? With this information, can we *help* a student use the resources better, based on the usage of the resource by other students in their groups?

We hope to find similar patterns of use in the data gathered from LON-CAPA, and eventually be able to make predictions as to the most-beneficial course of studies for each learner based on their present usage. The system could then make suggestions to the learner as to how to best proceed.

2 Map the Problem to Genetic Algorithm

Genetic Algorithms have been shown to be an effective tool to use in data mining and pattern recognition. [7], [10], [6], [16], [15], [13], [4]. An important aspect of GAs in a learning context is their use in pattern recognition. There are two different approaches to applying GA in pattern recognition:

1. Apply a GA directly as a classifier. Bandyopadhyay and Murthy in [3] applied GA to find the decision boundary in N dimensional feature space.

2. Use a GA as an optimization tool for resetting the parameters in other classifiers. Most applications of GAs in pattern recognition optimize some parameters in the classification process. Many researchers have used GAs in feature selection [2], [9], [12], [18]. GAs has been applied to find an optimal set of feature weights that improve classification accuracy. First, a traditional feature extraction method such as Principal Component Analysis (PCA) is applied, and then a classifier such as k-NN is used to calculate the fitness function for GA [17], [19]. Combination of classifiers is another area that GAs have been used to optimize. Kuncheva and Jain in [11] used a GA for selecting the features as well as selecting the types of individual classifiers in their design of a Classifier Fusion System. GA is also used in selecting the prototypes in the case-based classification [20].

In this paper we will focus on the second approach and use a GA to optimize a combination of classifiers. Our objective is to *predict* the students' final grades based on their web-use features, which are extracted from the homework data. We design, implement, and evaluate a series of pattern classifiers with various parameters in order to compare their performance on a dataset from LON-CAPA. Error rates for the individual classifiers, their combination and the GA optimized combination are presented.

2.1 Dataset and Class Labels

As test data we selected the student and course data of a LON-CAPA course, PHY183 (Physics for Scientists and Engineers I), which was held at MSU in spring semester 2002. This course integrated 12 homework sets including 184 problems, all of which are online. About 261 students used LON-CAPA for this course. Some of students dropped the course after doing a couple of homework sets, so they do not have any final grades. After removing those students, there remained 227 valid samples. The grade distribution of the students is shown in Fig 1.

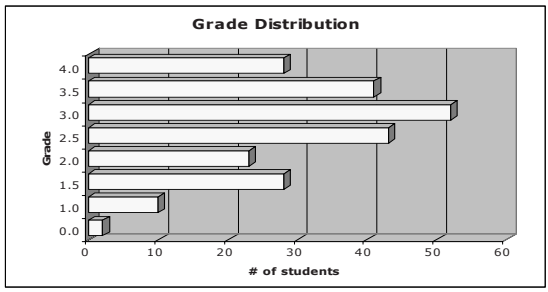

Fig. 1. Graph of distribution of grades in course PHY183 SS02

We can group the students regarding their final grades in several ways, 3 of which are:

1. Let the 9 possible class labels be the same as students' grades, as shown in table 1
2. We can label the students in relation to their grades and group them into three classes, "*high*" representing grades from 3.5 to 4.0, "*middle*" representing grades from 2.5 to 3, and "*low*" representing grades less than 2.5.
3. We can also categorize the students with one of two class labels: "*Passed*" for grades higher than 2.0, and "*Failed*" for grades less than or equal to 2.0, as shown in table 3.

Table 1. Selecting 9 class labels regarding to students' grades in course PHY183 SS02

Class	Grade	Student #	Percentage
1	0.0	2	0.9%
2	0.5	0	0.0%
3	1.0	10	4.4%
4	1.5	28	12.4%
5	2.0	23	10.1%
6	2.5	43	18.9%
7	3.0	52	22.9%
8	3.5	41	18.0%
9	4.0	28	12.4%

Table 2. Selecting 3 class labels regarding to students' grades in course PHY183 SS02

Class	Grade	Student #	Percentage
High	Grade >= 3.5	69	30.40%
Middle	2.0 < Grade < 3.5	95	41.80%
Low	Grade <= 2.0	63	27.80%

Table 3. Selecting 2 class labels regarding to students' grades in course PHY183 SS02

Class	Grade	Student #	Percentage
Passed	Grade > 2.0	164	72.2%
Failed	Grade <= 2.0	63	27.80%

We can predict that the error rate in the first class grouping should be higher than the others, because the sample size of the grades over 9 classes differs considerably. It is clear that we have less data for the first three classes in the training phase, and so the error rate would likely be higher in the evaluation phase.

2.2 Extractable Features

An essential step in doing classification is selecting the features used for classification. Below we discuss the features from LON-CAPA that were used, how they can be visualized (to help in selection) and why we normalize the data before classification.

The following features are stored by the LON-CAPA system:

1. Total number of correct answers. (Success rate)
2. Getting the problem right on the first try, vs. those with high number of tries. (Success at the first try)
3. Total number of tries for doing homework. (Number of attempts before correct answer is derived)
4. Time spent on the problem until solved (more specifically, the number of hours until correct. The difference between time of the last successful submission and the first time the problem was examined). Also, the time at which the student got the problem correct relative to the due date. Usually better students get the homework completed earlier.
5. Total time spent on the problem regardless of whether they got the correct answer or not. (Difference between time of the last submission and the first time the problem was examined).
6. Participating in the communication mechanisms, vs. those working alone. LON-CAPA provides online interaction both with other students and with the instructor. Where these used?
7. Reading the supporting material before attempting homework vs. attempting the homework first and then reading up on it.

8. Submitting a lot of attempts in a short amount of time without looking up material in between, versus those giving it one try, reading up, submitting another one, and so forth.
9. Giving up on a problem versus students who continued trying up to the deadline.
10. Time of the first log on (beginning of assignment, middle of the week, last minute) correlated with the number of tries or number of solved problems. A student who gets all correct answers will not necessarily be in the successful group if they took an average of 5 tries per problem, but it should be verified from this research.

In this paper we focused on the first six features in the PHY183 SS02 dataset that we have chosen for the classification experiment.

2.3 Classifiers

Pattern recognition has a wide variety of applications in many different fields, such that it is not possible to come up with a single classifier that can give good results in all the cases. The optimal classifier in every case is highly dependent on the problem domain. In practice, one might come across a case where no single classifier can classify with an acceptable level of accuracy. In such cases it would be better to pool the results of different classifiers to achieve the optimal accuracy. Every classifier operates well on different aspects of the training or test feature vector. As a result, assuming appropriate conditions, combining multiple classifiers may improve classification performance when compared with any single classifier [4].

The scope of this survey is restricted to comparing some popular non-parametric pattern classifiers and a single parametric pattern classifier according to the error estimate. Six different classifiers using the LON-CAPA datasets are compared in this study. The classifiers used in this study include *Quadratic Bayesian classifier, 1-nearest neighbor (1-NN)*, *k-nearest neighbor (k-NN)*, *Parzen-window, multi-layer perceptron (MLP),* and *Decision Tree.*[2] These classifiers are some of the common classifiers used in most practical classification problems. After some preprocessing operations were made on the dataset, the error rate of each classifier is reported. Finally, to improve performance, a combination of classifiers is presented.

2.4 Normalization

Having assumed in Bayesian and Parzen-window classifiers that the features are normally distributed, it is necessary that the data for each feature be normalized. This ensures that each feature has the same weight in the decision process. Assuming that the given data is Gaussian distributed, this normalization is performed using the mean and standard deviation of the training data. In order to normalize the training data, it is necessary first to calculate the sample mean μ, and the standard deviation σ of

[2] The first five classifiers are coded in MATLAB™ 6.0, and for the decision tree classifiers we have used some available software packages such as C5.0, CART, QUEST, and CRUISE.

each feature, or column, in this dataset, and then normalize the data using the equation (1)

$$x_i = \frac{x_i - \mu}{\sigma} \quad (1)$$

This ensures that each feature of the training dataset has a normal distribution with a mean of zero and a standard deviation of one. In addition, the kNN method requires normalization of all features into the same range. However, we should be cautious in using the normalization before considering its effect on classifiers' performances.

2.5 Combination of Multiple Classifiers (CMC)

In combining multiple classifiers we want to improve classifier performance. There are different ways one can think of combining classifiers:

- The simplest way is to find the overall error rate of the classifiers and choose the one which has the least error rate on the given dataset. This is called an *offline CMC*. This may not really seem to be a CMC; however, in general, it has a better performance than individual classifiers.
- The second method, which is called *online CMC*, uses all the classifiers followed by a vote. The class getting the *maximum votes* from the individual classifiers will be assigned to the test sample. This method intuitively seems to be better than the previous one. However, when tried on some cases of our dataset, the results were not better than the best result in previous method. So, we changed the rule of majority vote from "*getting more than 50% votes*" to "*getting more than 75% votes*". This resulted in a significant improvement over offline CMC.

Using the second method, we show in table 4 that CMC can achieve a significant accuracy improvement in all three cases of 2, 3, and 9-classes. Now we are going to use GA to find out that whether we can maximize the CMC performance.

3 Optimizing the CMC Using a GA

We used GAToolBox[3] for MATLAB to implement a GA to optimize classification performance. Our goal is to find a population of best weights for every feature vector, which minimize the classification error rate.

The feature vector for our predictors are the set of six variables for every student: Success rate, Success at the first try, Number of attempts before correct answer is derived, the time at which the student got the problem correct relative to the due date, total time spent on the problem, and the number of online interactions of the student both with other students and with the instructor.

[3] Downloaded from `http://www.shef.ac.uk/~gaipp/ga-toolbox/`

We randomly initialized a population of six dimensional weight vectors with values between 0 and 1, corresponding to the feature vector and experimented with different number of population sizes. We found good results using a population with 200 individuals. The GA Toolbox supports binary, integer, real-valued and floating-point chromosome representations. Real-valued populations may be initialized using the Toolbox function *crtrp*. For example, to create a random population of 6 individuals with 200 variables each: we define boundaries on the variables in *FieldD* which is a matrix containing the boundaries of each variable of an individual.

```
FieldD = [ 0 0 0 0 0 0;  % lower bound
           1 1 1 1 1 1]; % upper bound
```

We create an initial population with `Chrom = crtrp(200, FieldD)`, So we have for example:

```
Chrom = 0.23 0.17 0.95 0.38 0.06 0.26
        0.35 0.09 0.43 0.64 0.20 0.54
        0.50 0.10 0.09 0.65 0.68 0.46
        0.21 0.29 0.89 0.48 0.63 0.89
             ..................
```

We used the simple genetic algorithm (SGA), which is described by Goldberg in [9]. The SGA uses common GA operators to find a population of solutions which optimize the fitness values.

3.1 Recombination

We used "*Stochastic Universal Sampling*" [1] as our selection method. A form of stochastic universal sampling is implemented by obtaining a cumulative sum of the fitness vector, *FitnV*, and generating *N* equally spaced numbers between 0 and sum(FitnV). Thus, only one random number is generated, all the others used being equally spaced from that point. The index of the individuals selected is determined by comparing the generated numbers with the cumulative sum vector. The probability of an individual being selected is then given by

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N_{ind}} f(x_i)} \tag{2}$$

where $f(x_i)$ is the fitness of individual x_i and $F(x_i)$ is the probability of that individual being selected.

3.2 Crossover

The crossover operation is not necessarily performed on all strings in the population. Instead, it is applied with a probability *Px* when the pairs are chosen for breeding. We selected $Px = 0.7$. There are several functions to make crossover on real-valued matrices. One of them is *recint*, which performs intermediate recombination between pairs of individuals in the current population, OldChrom, and returns a new popula-

tion after mating, NewChrom. Each row of OldChrom corresponds to one individual. *recint* is a function only applicable to populations of real-value variables. Intermediate recombination combines parent values using the following formula [14]:

$$Offspring = parent1 + Alpha \times (parent2 - parent1) \quad (3)$$

Alpha is a Scaling factor chosen uniformly in the interval [-0.25, 1.25]

3.3 Mutation

A further genetic operator, mutation is applied to the new chromosomes, with a set probability *Pm*. Mutation causes the individual genetic representation to be changed according to some probabilistic rule. Mutation is generally considered to be a background operator that ensures that the probability of searching a particular subspace of the problem space is never zero. This has the effect of tending to inhibit the possibility of converging to a local optimum, rather than the global optimum.

There are several functions to make mutation on real-valued population. We used *mutbga,* which takes the real-valued population, OldChrom, mutates each variable with given probability and returns the population after mutation, *NewChrom = mutbga(OldChrom, FieldD, MutOpt)* takes the current population, stored in the matrix OldChrom and mutates each variable with probability by addition of small random values (size of the mutation step). We considered 1/600 as our mutation rate. The mutation of each variable is calculated as follows:

$$Mutated\ Var = Var + MutMx \times range \times MutOpt(2) \times delta \quad (4)$$

where delta is an internal matrix which specifies the normalized mutation step size; MutMx is an internal mask table; and MutOpt specifies the mutation rate and its shrinkage during the run. The mutation operator *mutbga* is able to generate most points in the hypercube defined by the variables of the individual and the range of the mutation. However, it tests more often near the variable, that is, the probability of small step sizes is greater than that of larger step sizes.

3.4 Fitness Function

During the reproduction phase, each individual is assigned a fitness value derived from its raw performance measure given by the objective function. This value is used in the selection to bias towards more fit individuals. Highly fit individuals, relative to the whole population, have a high probability of being selected for mating whereas less fit individuals have a correspondingly low probability of being selected. The error rate is measured in each round of cross validation by dividing "the total number of misclassified examples" into "total number of test examples". Therefore, our *fitness function* measures the error rate achieved by CMC and our objective would be to maximize this performance (minimize the error rate).

4 Experiment Results

Without using GA, the overall results of classifiers' performance on our dataset, regarding the four tree-classifiers, five non-tree classifiers and CMC are shown in the Table 4. Regarding individual classifiers, for the case of 2-classes, kNN has the best performance with 82.3% accuracy. In the case of 3-classes and 9-classes, CART has the best accuracy of about 60% in 3-classes and 43% in 9-Classes. However, considering the combination of non-tree-based classifiers, the CMC has the best performance in all three cases. That is, it achieved 86.8% accuracy in the case of 2-Classes, 71% in the case of 3-Classes, and 51% in the case of 9-Classes.

Table 4. Comparing the Error Rate of all classifiers on PHY183 dataset in the cases of 2-Classes, 3-Classess, and 9-Classes, using 10-fold cross validation, **without GA**

		Performance %		
Classifier		**2-Classes**	**3-Classes**	**9-Classes**
Tree Classifier	**C5.0**	80.3	56.8	25.6
	CART	81.5	**59.9**	**33.1**
	QUEST	80.5	57.1	20.0
	CRUISE	81.0	54.9	22.9
Non-tree Classifier	**Bayes**	76.4	48.6	23.0
	1NN	76.8	50.5	29.0
	KNN	**82.3**	50.4	28.5
	Parzen	75.0	48.1	21.5
	MLP	79.5	50.9	-
	CMC	**86.8**	**70.9**	**51.0**

For GA optimization, we used 200 individuals in our population, running the GA over 500 generations. We ran the program 10 times and got the averages, which are shown, in table 5. In every run 500×200 times the fitness function is called in which we used 10-fold cross validation to measure the average performance of CMC. So every classifier is called 3×10^6 times for the case of 2-classes, 3-classes and 9-classes. Thus, the time overhead for fitness evaluation is critical. Since using the MLP in this process took about 2 minutes and all other four non-tree classifiers (Bayes, 1NN, 3NN, and Parzen window) took only 3 seconds, we omitted the MLP from our classifiers group so we could obtain the results in a reasonable time.

Table 5. Comparing the CMC Performance on PHY183 dataset Using GA and without GA in the cases of 2-Classes, 3-Classess, and 9-Classes, 95% confidence interval.

	Performance %		
Classifier	2-Classes	3-Classes	9-Classes
CMC of 4 Classifiers without GA	83.87 ± 1.73	61.86 ± 2.16	49.74 ± 1.86
GA Optimized CMC, Mean individual	94.09 ± 2.84	72.13 ± 0.39	62.25 ± 0.63
Improvement	**10.22 ± 1.92**	**10.26 ± 1.84**	**12.51 ± 1.75**

The results in Table 5 represent the mean performance with a two-tailed t-test with a 95% confidence interval. For the improvement of GA over non-GA result, a P-value indicating the probability of the Null-Hypothesis (There is no improvement) is also given, showing the significance of the GA optimization. All have $p<0.000$, indicating significant improvement. Therefore, using GA, in all the cases, we got more than a 10% mean individual performance improvement and about 12 to 15% mean individual performance improvement. Fig. 2 shows the best result of the ten runs over our dataset. These charts represent the population mean, the best individual at each generation and the best value yielded by the run.

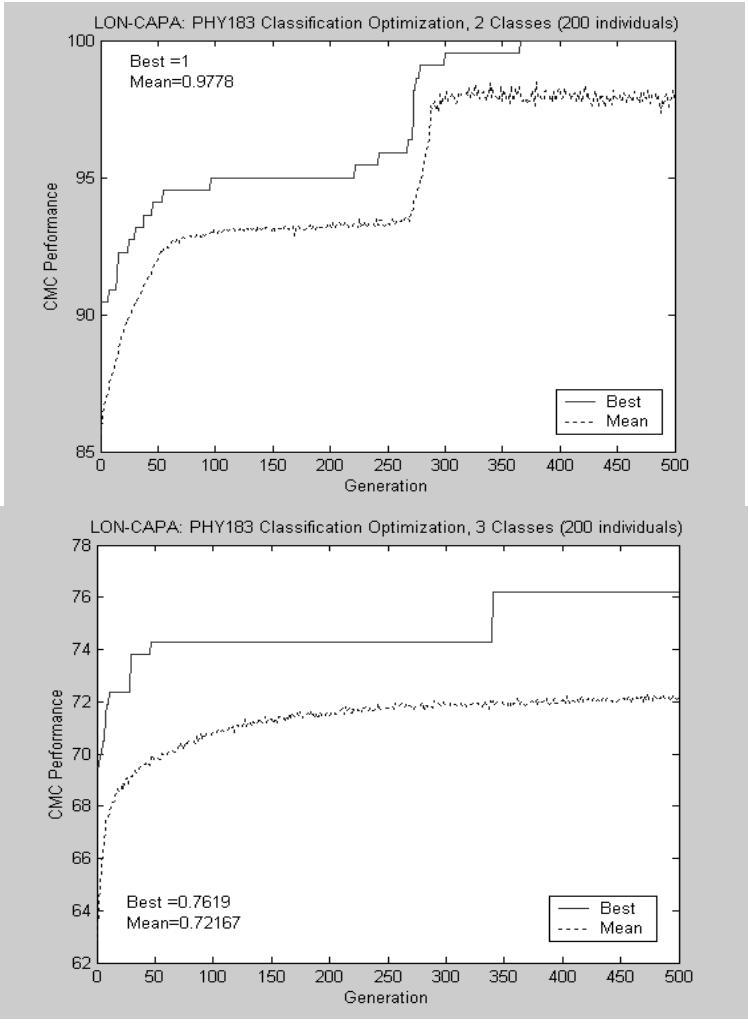

Fig. 2. Graph of GA Optimized CMC performance in the case of 2, and 3-Classes

Finally, we can examine the individuals (weights) for features by which we obtained the improved results. This feature weighting indicates the importance of each feature for making the required classification. In most cases the results are similar to Multiple Linear Regressions or tree-based software that use statistical methods to measure feature importance. Table 6 shows the importance of the six features in the 3-classes case using the Entropy splitting criterion. Based on entropy, a statistical property called *information gain* measures how well a given feature separates the training examples in relation to their target classes. Entropy characterizes *impurity* of an arbitrary collection of examples S at a specific node N. In [5] the impurity of a node *N* is denoted by *i(N)* such that:

$$Entropy(S) = i(N) = -\sum_{j} P(\omega_j)\log_2 P(\omega_j) \tag{5}$$

where $P(\omega_j)$ is the fraction of examples at node *N* that go to category ω_j.

Table 6. Feature Importance in 3-Classes Using Entropy Criterion

Feature	Importance %
Total_Correct _Answers	**100.00**
Total_Number_of_Tries	**58.61**
First_Got_Correct	**27.70**
Time_Spent_to_Solve	**24.60**
Total_Time_Spent	**24.47**
Communication	**9.21**

The GA results also show that the "Total number of correct answers" and the "Total number of tries" are the most important features for the classification. The second column in table 6 shows the percentage of feature importance.

5 Conclusions and Future Work

Four classifiers were used to segregate the students. A combination of multiple classifiers leads to a significant accuracy improvement in all 3 cases. Weighing the features and using a genetic algorithm to minimize the error rate improves the prediction accuracy at least 10% in the all cases of 2, 3 and 9-Classes. In cases where the number of features is low, the feature weighting worked much better than feature selection. The successful optimization of student classification in all three cases demonstrates the merits of using the LON-CAPA data to *predict* the students' final grades based on their features, which are extracted from the homework data.

We are going to apply Genetic Programming to produce many different combinations of features, to extract new features and improve prediction accuracy. We plan to use Evolutionary Algorithms to classify the students and problems directly as well. We also want to apply Evolutionary Algorithms to find Association Rules and Dependency among the groups of problems *(Mathematical, Optional Response, Numerical, Java Applet, and so forth)* of LON-CAPA homework data sets.

Acknowledgements. This work was partially supported by the National Science Foundation under ITR 0085921.

References

1. Baker, J. E.: Reducing bias and inefficiency in the selection algorithm, Proceeding ICGA 2, Lawrence Erlbuam Associates, Publishers, (1987) 14–21
2. Bala J., De Jong K., Huang J., Vafaie H.: and Wechsler H. Using learning to facilitate the evolution of features for recognizing visual concepts. Evolutionary Computation 4(3) - Special Issue on Evolution, Learning, and Instinct: 100 years of the Baldwin Effect. (1997)
3. Bandyopadhyay, S., and Muthy, C.A.: Pattern Classification Using Genetic Algorithms, Pattern Recognition Letters, Vol. 16, (1995) 801–808
4. De Jong K.A., Spears W.M. and Gordon D.F.: Using genetic algorithms for concept learning. Machine Learning 13, (1993) 161–188
5. Duda, R.O., Hart, P.E., and Stork, D.G.: Pattern Classification. 2nd Edition, John Wiley & Sons, Inc., New York NY. (2001)
6. Falkenauer E.: Genetic Algorithms and Grouping Problems. John Wiley & Sons, (1998)
7. Freitas, A.A.: A survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery,See: www.pgia.pucpr.br/~alex/papers. A chapter of: A. Ghosh and S. Tsutsui. (Eds.) Advances in Evolutionary Computation. Springer-Verlag, (2002)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, MA, Addison-Wesley, (1989)
9. Guerra-Salcedo C. and Whitley D.: Feature Selection mechanisms for ensemble creation: a genetic search perspective. In: Freitas AA (Ed.) Data Mining with Evolutionary Algorithms: Research Directions, Technical Report WS-99-06. AAAI Press, (1999)
10. Jain, A. K.; Zongker, D.: Feature Selection: Evaluation, Application, and Small Sample Performance, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 19, No. 2, February (1997)
11. Kuncheva , L.I., and Jain, L.C.: Designing Classifier Fusion Systems by Genetic Algorithms, IEEE Transaction on Evolutionary Computation, Vol. 33 (2000) 351–373
12. Martin-Bautista MJ and Vila MA. A survey of genetic feature selection in mining issues. Proceeding Congress on Evolutionary Computation (CEC-99), Washington D.C., July (1999) 1314–1321
13. Michalewicz Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd Ed. Springer-Verlag, (1996)
14. Muhlenbein and Schlierkamp-Voosen D.: Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization, Evolutionary Computation, Vol. 1, No. 1, (1993) 25–49
15. Park Y and Song M.: A genetic algorithm for clustering problems. Genetic Programming 1998: Proceeding of 3rd Annual Conference, Morgan Kaufmann, (1998), 568–575.
16. Pei, M., Goodman, E.D. and Punch, W.F.: Pattern Discovery from Data Using Genetic Algorithms, Proceeding of 1st Pacific-Asia Conference Knowledge Discovery & Data Mining (PAKDD-97) Feb. (1997)
17. Pei, M., Punch, W.F., and Goodman, E.D.: Feature Extraction Using Genetic Algorithms, Proceeding of International Symposium on Intelligent Data Engineering and Learning'98 (IDEAL'98), Hong Kong, Oct. (1998)
18. Punch, W.F., Pei, M., Chia-Shun, L., Goodman, E.D.: Hovland, P., and Enbody R. Further research on Feature Selection and Classification Using Genetic Algorithms, In 5th International Conference on Genetic Algorithm, Champaign IL, (1993) 557–564
19. Siedlecki, W., Sklansky J., A note on genetic algorithms for large-scale feature selection, Pattern Recognition Letters, Vol. 10, (1989) 335–347
20. Skalak D. B.: Using a Genetic Algorithm to Learn Prototypes for Case Retrieval an Classification. Proceeding of the AAAI-93 Case-Based Reasoning Workshop, Washigton, D.C., American Association for Artificial Intelligence, Menlo Park, CA, (1994) 64–69