A Method for Handling Numerical Attributes in GA-Based Inductive Concept Learners

Federico Divina, Maarten Keijzer, and Elena Marchiori

Department of Computer Science
Vrije Universiteit
De Boelelaan 1081a, 1081 HV Amsterdam
The Netherlands
{divina,mkeijzer,elena}@cs.vu.nl

Abstract. This paper proposes a method for dealing with numerical attributes in inductive concept learning systems based on genetic algorithms. The method uses constraints for restricting the range of values of the attributes and novel stochastic operators for modifying the constraints. These operators exploit information on the distribution of the values of an attribute. The method is embedded into a GA based system for inductive logic programming. Results of experiments on various data sets indicate that the method provides an effective local discretization tool for GA based inductive concept learners.

1 Introduction

Inductive Concept Learning (ICL) [14] constitutes a central topic in Machine Learning. The problem can be formulated in the following manner: given a description language used to express possible hypotheses, a background knowledge, a set of positive examples, and a set of negative examples, one has to find a hypothesis which covers all positive examples and none of the negative ones (cf. [12, 15]). The so learned concept can be used to classify previously unseen examples. Concepts are induced because obtained from the observation of a limited set of training examples. When hypotheses are expressed in (a fragment of) first order logic, ICL is called Inductive Logic Programming (ILP).

Many learning problems use data containing numerical attributes. Numerical attributes affect the efficiency of learning and the accuracy of the learned theory. The standard approach for dealing with numerical attributes in inductive concept learning is to discretize them into intervals that will be used instead of the continuous values. The discretization can be done during the learning process (*local* discretization), or beforehand (*global* discretization). Discretization methods that employ the class information of the instances are called *supervised* methods, while if they do not use this information they are called *unsupervised* methods. The simplest way is to use an equal interval width method. In this way, the continuous values are simply divided into n equal sized bins, where n is a parameter. A better way for discretizing numerical attributes was proposed by Fayyad and Irani [9]. This method uses a recursive entropy minimization

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 898–908, 2003.

algorithm and employs the Minimum Description Length principle in the stopping criterion. In [16] a variant of the Fayyad and Irani's method is used for discretizing numerical attributes in an Inductive Logic Programming system. In [1,2] methods using adaptive discrete intervals are used within a GA based system for classification. Another approach for local discretization is proposed by Kwedlo and Kretowski. In their work, information on a subset of all thresholds of a numerical attribute is used as to determine thresholds in evolved decision rules [13].

The aim of this paper is to introduce an alternative method for dealing with numerical attributes in evolving classifiers where the actual discretization is determined at run time (i.e. local discretization). An unsupervised, global method is used to determine the density of data. Due to the use of unsupervised methods, unlabeled data or a priori knowledge about the distribution of data can be used to fine-tune the density estimation. The information gathered in the global pre-processing step is used to guide the genetic operators to make density controlled operations in the search process.

The method introduced here is general, in that guiding the genetic operators using estimated, unlabeled or a priori information about the distribution of data can be used in any evolutionary classifying system. In order to assess the benefits of the method we use one particular system: the evolutionary ILP system ECL [7,6]. We run experiments on different data sets and compare the results obtained using the original ECL system (where numerical attributes are treated as nominal), ECL with numerical attributes discretized using Fayyad and Irani's method, and ECL with numerical attributes treated using the novel method based on constraints. The results of the experiments indicate that the proposed method allows ECL to find better solutions which are comparable or better than those found using Fayyad and Irani's method.

The paper it structured in the following way. In section 2 we describe the method for dealing with numerical attributes. In section 3 we introduce the main features of the ECL system. In section 4 we perform experiments and discuss the results and finally in section 5 some conclusions and future work are given.

2 Handling Numerical Attributes Using Constraints

We propose to handle numerical attributes by using constraints of the form $a \leq X \leq b$, where X is a variable relative to a numerical attribute, and a, b are attribute values. During the execution of a GA based inductive concept learner, a constraint for a numerical attribute is generated when that attribute is selected. Constraints are then modified during the evolutionary process by using the novel operators defined in the following sections. These novel operators use information on the distribution of the values of attributes in order to update the interval boundaries of the constraints. Information about the distribution of the values of attributes is obtained by clustering the values of each attribute using a mixture of Gaussian distribution.

We will first briefly introduce the clustering algorithm that is used, and then we will describe how constraints are modified. In the sequel we will use Prolog syntax, where a variable starts with uppercase character while a constant starts with a lowercase character.

2.1 Clustering Attribute Values

We cluster the values of each attribute in order to get information about the distribution of the data, and use this information in the operators for modifying constraints. Clustering is performed using the Expectation-Maximization (EM) algorithm [5] (in the experiments we use WEKA implementation [17]). For each attribute the EM algorithm returns n clusters described by means μ_i and standard deviations σ_i, $1 \leq i \leq n$ of Gaussian distributions.

A begin (b_{cl_i}) and end (e_{cl_i}) of a cluster $cluster_i$ are generated by intersecting the distributions of $cluster_i$ with the one of $cluster_{i-1}$ and $cluster_{i+1}$, respectively. Special cases are $b_{cl_1} = -\infty$ and $e_{cl_n} = +\infty$.

The boundaries a, b of each constraint $a \leq X \leq b$ are contained in one cluster.

2.2 Operators

Within each cluster, we use constraints for restricting the range of values of an attribute variable.

A constraint can be modified either by enlarging its boundaries, or by shrinking, or by shifting the boundaries, or by changing the cluster of its boundaries, or by grounding the constraint (i.e., restricting its range to a single value).

Formally, consider the constraint $C : a \leq X \leq b$, and let b_{cl} and e_{cl} be the begin and the end of the cluster cl containing C.

Enlarge. This operator applied to C returns a constraint $C' = a' \leq X \leq b'$ where $a' \leq a$ and $b \leq b'$. The new bounds a', b' are computed in the following way:

1. let $min = minimum\,\{P(b_{cl} \leq X \leq a), P(b \leq X \leq e_{cl})\}$ the minimum of the probability that X is between b_{cl} and a and the probability that X is between b and e_{cl}.
2. generate randomly p with $0 \leq p \leq min$;
3. find two points a', b' such that $p = P(a' \leq X \leq a)$ and $p = P(b \leq X \leq b')$.

Bounds are enlarged by generating probabilities instead of random points inside the cluster because in this way we can exploit the information about the distribution of the data values in an interval.

Shrink. This operator applied to C returns $C' = a' \leq X \leq b'$ where $a' \geq a$ and $b' \leq b$. a' and b' are computed by randomly choosing $p \leq P(a, b)$ such that $p = P(a \leq X \leq a') = P(b' \leq X \leq b)$, and $a' \leq b'$.

Ground. This operator, applied to C returns $C' = a' \leq X \leq a'$, with a' in the cluster containing a, b.

Shift. This operator, applied to C returns $C' = a' \leq X \leq b'$ where a', b' are points in the cluster containing a, b such that $P(a' \leq X \leq b') = P(a \leq X \leq b)$.

Change Cluster. This operator, applied to $C = a \leq X \leq b$ returns $C' = a' \leq X \leq b'$ where a', b' belong to a different cluster. The new cluster is chosen at random. Once the new cluster has been chosen, a pair a', b' with $a' \leq b'$ is randomly generated. In general, $P(a' \leq X \leq b')$ is not equal to $P(a \leq X \leq b)$.

3 ECL: A GA Based Inductive Concept Learner

In order to test the effectiveness of our discretization method, we embed it in the ILP system ECL. Like many ILP systems, ECL treats numerical attributes as if they were nominal, therefore it can be used as a platform for testing and comparing our local discretization method with the global discretization method by Fayyad and Irani.

In figure 1 a scheme of ECL is given.

```
ALGORITHM ECL
Sel = positive_examples
repeat
    Select partial Background Knowledge
    Population = ∅
    while (not terminate) do
        Adjust examples weights
        Select n chromosomes using Sel
        for each selected chromosome chrm
            Mutate chrm
            Optimize chrm
            Insert chrm in Population
        end for
    end while
    Store Population in Final_Population
    Sel = Sel - { positive examples
        covered by clauses in Population }
until max_iter is reached
Extract final theory from Final_Population
```

Fig. 1. The overall learning algorithm ECL

The system takes as input a background knowledge (BK), and a set of positive and negative examples, and outputs a set of Horn clauses that covers many positive examples and few negative ones.

Recall that a Horn clause is of the form $p(X,Y) : -r(X,Z), q(Y,a).$ with head $p(X,Y)$ and body $r(X,Z), q(Y,a)$. A clause has a declarative interpretation: $\forall X,Y,Z(r(X,Z), q(X,a) \rightarrow p(X,Y))$ and a procedural one: *in order to solve* $p(X,Y)$ *solve* $r(X,Z)$ *and* $q(Y,a)$. Thus a set of clauses forms a logic program, which can directly (in a slightly different syntax) be executed in the programming language Prolog. The background knowledge used by ECL contains ground facts (i.e. clauses of the form $r(a,b) \leftarrow .$ with a, b constants). The training set contains facts which are true (positive examples) and false (negative examples) for the target predicate. A clause is said to *cover an example* if the theory formed by the clause and the background knowledge logically entails the example.

In the repeat statement of the algorithm a `Final_population` is iteratively built from the empty one. Each iteration performs the following actions: part of the background knowledge is randomly selected, an evolutionary algorithm that uses that part of BK is run and the resulting set of Horn clauses is joined to the actual `Final_population`.

The evolutionary algorithm evolves a `Population` of Horn clauses starting from an empty population, where an individual represents a clause, by the repeated application of selection, mutation (the system does not use any crossover operator) and optimization in the following way.

At each generation n individuals are selected using a variant of the US selection operator [10]. Roughly, the selection operator selects a positive example and performs a roulette wheel on the set of individuals in the `Population` that cover that example. If that example is not covered by any individual then a new clause is created using that example as seed.

Each selected individual undergoes mutation and optimization.

Mutation consists of the application of one of the following four generalization/specialization operators. A clause is generalized either by deleting an atom from its body or by turning a constant into a variable, and it is specialized by either adding an atom or turning a variable into a constant. Each operator has a degree of greediness. In order to make a mutation, a number of mutation possibilities is considered, and the one yielding the best improvement, in terms of fitness, is applied.

Optimization consists of the repeated application of one of the mutation operators, until the fitness of the individual increases, or a maximum number of optimization steps has been reached.

Individuals are then inserted in the population. If the population is not full then the individuals are simply inserted. If the population has reached its maximum size, then n tournaments are made among the individuals in the population and the resulting n worst individuals are substituted by the new individuals.

The fitness of an individual x is given by the inverse of its accuracy:

$$fitness(x) = \frac{1}{Acc(x)} = \frac{P+N}{p_x+(N-n_x)}$$

In the above formula P and N are respectively the total number of positive and negative examples, while p_x and n_x are the number of positive and negative examples covered by the individual x. We take the inverse of the accuracy, because ECL was originally designed to minimize a fitness function.

When the `Final_population` is large enough (after `max_iter` iterations) a subset of its clauses is extracted in such a way that it covers as many positive examples as possible, and as few negative ones as possible. To this aim an heuristic algorithm for the weighted set covering is used.

3.1 Clu-Con: ECL Plus Local Discretization

We consider the following variant of ECL, called Clu-Con (Clustering and Constrain), which incorporates our method for handling numerical values.

When a new clause is built using a positive example as a seed, or when a clause is specialized, atoms of the background knowledge are added to its body. Each time an atom describing the value of a numerical attribute is introduced in a clause, a constraint relative to that attribute is added to the clause as well. For example, consider the following clause for example $c23$:

$$Cl = p(c23) : -q(c23, a), t(c23, y).$$

Suppose now that we would like to add the atom $r(c23, 8)$ stating that in example $c23$ attribute r has value 8. Then we obtain the clause

$$p(c23) : -q(c23, a), t(c23, y), r(c23, X), 8 \leq X \leq 8.$$

The operators for handling constraints, introduced in section 2.2, are used as mutation operators. When an operator is chosen then it is applied to a constraint a number *n_choices* of times, where *n_choices* is a user supplied parameter. In this way *n_choices* new constraints are generated and the one yielding the best fitness improvement is selected.

Shrink and Ground. These two operators are applied when specializing a clause. More precisely, when the system is specializing a clause by turning a variable into a constant, if the selected variable occurs in a constraint then either Shrink or Ground are applied to that constraint.

Enlarge. This operator is applied when the system decides to generalize a clause. ECL has two generalization operators: *delete an atom* and *constant into variable* operators. When *delete an atom* is selected and the atom chosen for deletion describes the value of a numerical attribute, then both the atom and the constraint relative to the described attribute are deleted. If *delete an atom* is not selected and there are constraints in the body of the clause chosen for mutation, then the system randomly selects between the *constant into variable* and *enlarge* operators.

Change Cluster and Shift. The standard operators and the above described operators are applied inside a *mutate* procedure. Before calling this procedure, a test is performed to check if the selected individual has got constraints in the body of its clause. If this is the case, then either the *change cluster* or the *shift* operator is applied with a probability *pc* (typical value 0.2), otherwise the *mutate* procedure is called.

3.2 Ent_MDL: ECL Plus Global Discretization

The other variant of ECL we consider, called Ent_MDL (Entropy minimization plus Minimum Description Length principle), incorporates the popular Fayyad and Irani's method for discretizing numerical values [9]. In [8,11] a study of some discretization methods is conducted, and it emerged that Fayyad and Irani's method represents a good way for globally discretizing numerical attributes.

This supervised recursive algorithm uses the class information entropy of candidate intervals to select the boundaries of the bins for discretization. Given a set S of instances, an attribute p, and a partition bound t, the class information entropy of the partition induced by t is given by

$$E(p,t,S) = Entropy(S_1)\frac{|S_1|}{|S|} + Entropy(S_2)\frac{|S_2|}{|S|}$$

where S_1 is the set of instances whose values of p are in the first half of the partition and S_2 the set of instances whose values of p are in the second half of the partition. Moreover, $|S|$ denotes the number of elements of S and $Entropy(S) = -p_+ log_2(p_+) - p_- log_2(p_-)$ with p_+ the proportion of positive examples in S and p_- is the proportion of negative examples in S.

For a given attribute p the boundary t^* which minimizes $E(p,t,S)$ is selected as a binary discretization boundary. The method is then applied recursively to both the partitions induced by t^* until a stopping criterion is satisfied. The Minimum Description Length principle is used to define the stopping criterion. Recursive partitions within a set of instances stops if $Entropy(S) - E(p,t,S)$ is smaller than $log_2(N-1)/N + \Delta(p,t,S)/N$, where $\Delta(p,t,S) = log_2(3^k - 2) - [k \cdot Entropy(S) - k_1 \cdot Entropy(S_1) - k_2 \cdot Entropy(S_2)]$, and k_i is the number of class labels represented in S_i.

The method is used as a pre-processing step for ECL, where the domains of numerical attributes are split into a number of intervals, and each interval is considered as one value of a nominal attribute.

4 Experiments

In order to asses the goodness of the proposed method for handling numerical attributes, we conduct experiments on benchmark datasets using ECL and the two variants Clu_Con and Ent_MDL described previously. The characteristics of the datasets are shown in table 1. Datasets 1 to 6 are taken from the UCI repository [3], while dataset 7 originates from [4]. These datasets are chosen

Table 1. Features of the datasets. The first column shows the total number of training examples with, between brackets, the number of positive and negative examples. The second and third columns show the number of numerical and nominal attributes. The fourth column shows the number of elements of the background knowledge.

	Dataset	Instances (+,-)	Numerical	Nominal	BK
1	Australian	690 (307,383)	6	8	9660
2	German	1000 (700,300)	24	0	24000
3	Glass2	163 (87,76)	9	0	1467
4	Heart	270 (120,150)	13	0	3510
5	Ionosphere	351 (225,126)	34	0	11934
6	Pima-Indians	768 (500,268)	8	0	6144
7	Mutagenesis	188 (125,63)	6	4	13125

Table 2. Parameter settings: pop_size = maximum size of population, mut_rate = mutation rate, n = number of selected clauses, max_gen = maximum number of GA generations, max_iter = maximum number of iterations, N(1,2,3,4) = parameters of the genetic operators, p= probability of selecting a fact in the background knowledge, l = maximum length of a clause.

	Australian	German	Glass2	Heart	Ionosphere	Pima-Indians	Mutagenesis
pop_size	50	200	150	50	50	60	50
mut_rate	1	1	1	1	1	1	1
n	15	30	20	15	15	7	15
max_gen	1	2	3	1	6	5	2
max_iter	10	30	15	10	10	10	10
N(1,2,3,4)	(4,4,4,4)	(3,3,3,3)	(2,8,2,9)	(4,4,4,4)	(4,8,4,8)	(2,5,3,5)	(4,8,2,8)
nb	∞	200	8	20	20	8	8
p	0.4	0.3	0.8	1	0.2	0.2	0.8
l	6	9	5	6	5	4	3

because they contain mainly numerical attributes. The parameters used in the experiments are shown in table 2 and are obtained after a number of preliminary experiments. We use ten-fold cross validation. Each dataset is divided in ten disjoint sets of similar size; one of these sets is used as test set, and the union of the remaining nine forms the training set. Then ECL is run on the training set and it outputs a logic program, whose performance on new examples is assessed using the test set. Three runs with different random seed are performed on each dataset.

Table 3 contains the results of the experiments. The performance of the GA learner improves when numerical attributes are discretized. In particular, Clu-Con seems to perform best on these datasets, being outperformed by Ent-MDL only in two cases, namely on the Ionosphere and the Pima-Indians datasets.

Table 3. Results of experiments. Average accuracy, with standard deviation between brackets. In the first column the method adopting clusters and constraints described in the paper is employed. In the second column numerical attributes are treated as nominal. In the third column the Fayyad and Irani's discretization algorithm was used for globally discretizing numerical attributes.

	Dataset	Clu-Con	ECL	Ent-MDL
1	Australian	**0.79 (0.03)**	0.71 (0.05)	0.62 (0.06)
2	German	**0.73 (0.02)**	0.63 (0.07)	0.59 (0.08)
3	Glass2	**0.78 (0.07)**	0.51 (0.06)	0.69 (0.11)
4	Heart	**0.74 (0.09)**	0.67 (0.11)	0.68 (0.10)
5	Ionosphere	0.81 (0.07)	0.41 (0.03)	**0.85 (0.06)**
6	Pima-Indians	0.65 (0.09)	0.61 (0.05)	**0.69 (0.07)**
7	Mutagenesis	**0.90 (0.04)**	0.83 (0.05)	0.85 (0.07)

5 Conclusions and Future Work

In this paper we have proposed an alternative method for dealing with numerical attributes. The method uses standard density estimation to obtain a view of the distribution of data, and this information is used to guide the genetic operators to find local discretizations that are optimal for classification. By using an unsupervised method for density estimation, the method can make use of either unlabeled data or a priori knowledge on the density of the numeric attribute. In section 4 we have tested the method in the system ECL on some datasets containing numerical attributes. In a previous version of the system, no particular method for dealing with numerical attributes was implemented, so numerical attributes were treated as if they were nominal. We have shown that not only the proposed method improves the performance of the system, but also that the proposed method is in general more effective than a global discretization by means of Fayyad and Irani's algorithm. We believe that the proposed method could be profitably applied to other learning systems for dealing with numerical attributes.

Currently, the density estimation procedure only works with single attributes. However, when there is reason to believe that numeric attributes are covariant (for instance weight and height of a person), the entire method can be converted to use multiple attributes. For this, a multivariate variant of the expectation maximization algorithm can be used which will find the covariance matrix Σ instead of a single standard deviation σ. Because our genetic operators are defined to use only density information derived from the clusters, the operators can be easily adapted to mutate two or more numerical attributes at the same time in the context of the globally estimated covariance matrices. This is however left for future work.

We are currently investigating the possibility of changing the operators that act on constraints. In particular we would like to generate the probabilities p_i in the enlarge and shrink operator not at random, but in a way that could take

into consideration the actual coverage of the constraint. We are also considering the possibility of enlarging or shrinking a constraint asymmetrically.

References

1. J. Bacardit and J. M. Garrel, *Evolution of adaptive discretization intervals for a rule-based genetic learning system*, in GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, 9–13 July 2002, Morgan Kaufmann Publishers, p. 677.
2. —, *Evolution of multi-adaptive discretization intervals for a rule-based genetic learning system*, in Proceedings of the 7th Iberoamerican Conference on Artificial Intelligence (IBERAMIA2002), 2002, p. to appear.
3. C. Blake and C. Merz, *UCI repository of machine learning databases*, 1998.
4. A. Debnath, R. L. de Compadre, G. Debnath, A. Schusterman, and C. Hansch, *Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with molecular orbital energies and hydrophobicity*, Journal of Medical Chemistry, 34(2) (1991), pp. 786–797.
5. A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm*, Journal of the the Royal Statistical Society, 39 (1977), pp. 1–38.
6. F. Divina, M. Keijzer, and E. Marchiori, *Non-universal suffrage selection operators favor population diversity in genetic algorithms*, in Benelearn 2002: Proceedings of the 12th Belgian-Dutch Conference on Machine Learning (Technical report UU-CS-2002-046), Utrecht, The Netherlands, 4 December 2002, pp. 23–30.
7. F. Divina and E. Marchiori, *Evolutionary concept learning*, in GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, 9–13 July 2002, Morgan Kaufmann Publishers, pp. 343–350.
8. J. Dougherty, R. Kohavi, and M. Sahami, *Supervised and unsupervised discretization of continuous features*, in International Conference on Machine Learning, 1995, pp. 194–202.
9. U. Fayyad and K. Irani, *Multi-interval discretization of continuos attributes as pre-processing for classification learning*, in Proceedings of the 13th International Join Conference on Artificial Intelligence, Morgan Kaufmann Publishers, 1993, pp. 1022–1027.
10. A. Giordana and F. Neri, *Search-intensive concept induction*, Evolutionary Computation, 3 (1996), pp. 375–416.
11. R. Kohavi and M. Sahami, *Error-based and entropy-based discretization of continuous features*, in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996, pp. 114–119.
12. M. Kubat, I. Bratko, and R. Michalski, *A review of Machine Learning Methods*, in Machine Learning and Data Mining, R. Michalski, I. Bratko, and M. Kubat, eds., John Wiley and Sons Ltd., Chichester, 1998.
13. W. Kwedlo and M. Kretowski, *An evolutionary algorithm using multivariate discretization for decision rule induction*, in Principles of Data Mining and Knowledge Discovery, 1999, pp. 392–397.
14. T. Mitchell, *Generalization as search*, Artificial Intelligence, 18 (1982), pp. 203–226.
15. —, *Machine Learning*, Series in Computer Science, McGraw-Hill, 1997.

16. W. Van Laer, *From Propositional to First Order Logic in Machine Learning and Data Mining – Induction of first order rules with ICL*, PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, June 2002. 239+xviii pages.
17. I. H. Witten and E. Frank, *Weka machine learning algorithms in java*, in Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann Publishers, 2000, pp. 265–320.