DISCRETIZATION OF CONTINUOUS ATTRIBUTES USING GENETIC AND ENTROPY BASED CONCEPT LEARNER

Suman Ahmmed[1], Shamim Ahmmed[2] and Chowdhury Mofizur Rahman[1]
Department of Computer Science and Engineering, United International University[1]
MF Asia Ltd[2]
E-mail: suman@uiu.ac.bd, brainaub@yahoo.com, cmr@uiu.ac.bd

Abstract: *Genetic Algorithm (GA) based concept learner is widely used in supervised learning system for attribute based spaces. The conventional GA based operators can not directly deal with continuous attributes. We have used two separate approaches one is pure Genetic Algorithm and another is Entropy based approach coupled with Genetic Algorithm for converting continuous attributes into discrete ones. Later on these converted discrete attributes have been used in traditional GA based concept learner to enhance its performance. In our experiments with benchmark data set it has been revealed that statistically significant improvement has been achieved with the proposed technique.*

Keywords: *Genetic Algorithm, Entropy, Concept Learner, Continuous Attribute.*

1. Introduction

Many algorithms developed in the machine learning community focus on learning in nominal feature spaces [1], [3]. However, many real world classification tasks exist that involve continuous features where such algorithms could not be applied unless the continuous features are first discretized. Continuous variable discretization has received significant attention in the machine learning community only recently. Often, uniform binning of the data is used to produce the necessary data transformations for a learning algorithm, and no careful study of how this discretization affects the learning process is performed [2], [9]. In decision tree methods, such as C4.5 [10], continuous values are discretized during the learning process. The advantages of discretizing during the learning process have not yet been shown.

There are three different axes by which discretization methods can be classified: global vs. local, supervised vs. unsupervised, and static vs. dynamic. Local methods, as exemplified by C4.5, produce partitions that are applied to localized regions of the instance space. Global methods [1], such as binning, produce a mesh over the entire n-dimensional continuous instance space, where each feature is partitioned into regions independent of the other attributes. The mesh contains $\prod_{i=1}^{n} k_i$ regions, where k_i is the number of partitions of the ith feature. Several discretization methods, such as equal width interval binning, do not make use of instance labels in the discretization process. In analogy to supervised versus unsupervised learning methods, we refer to these as unsupervised discretization methods. In contrast, discretization methods that utilize the class labels are referred to as supervised discretization methods. Discretization groups continuous numeric values into discrete intervals. Originally the motivation for discretization was the inability of handling continuous values exhibited by early propositional learning algorithms. Even through most propositional algorithms nowadays can handle continuous values; discretization can still be beneficial for the following reasons:

Efficiency: Handling (lots of) continuous values tends to slow down induction considerably.

Intelligibility: Especially when faced with a large number of noisy training examples, learned concepts involving continuous attributes tend to be rather complex.

Accuracy: In the presence of noise good discretizations can sometimes improve predictive accuracy.

2. Discretization

Informally, the discretization of a continuous variable specifies the set of cut points in the continuous range of the continuous variable that delimit the intervals to be mapped into the values of the discretized variable.

More formally, the discretization for a variable X_i, defined over the real interval [a, b], with a<b, is uniquely defined by the ordered set of cutpoints $T = \{t_0, t_1, ..., t_{r_i}, t_{r_i+1}\}$ that partition the value range of X_i into the set of subintervals $[t_0, t_1], [t_1, t_2], ..., [t_{r_i}, t_{r_i+1}]$, with $a=t_0 < t_1 < \ldots < t_{r_i+1}$. The partition defines a discrete variable taking values in the domain {0, 1, . . .r_i}with jth value corresponding to the interval $[t_j, t_{j+1}]$.

Given a database *D* of N_D cases defined over X, as candidate cutpoints we only consider the (at most) N_D-1 mid points between contiguous data points in *D*. Therefore, the number of values the discretized variable can take is supper-bounded by N_D . Even in the simpler univariate discretization problem, the search for the best discretization has combinatorial complexity in the number of data points, since the complete set of possible discretizations to be considered has cardinality $\mathbf{2}^{N_D}\mathbf{-1}$

3. Pure Genetic Algorithm Based Approach

Genetic algorithm is a widely used parallel algorithm which is successfully used in different optimization problem. We first use GA in solving the discretization problem. In this approach we divide the full domain randomly in some sub domains. Our target is to find the sub domains where the minimum or maximum class examples of a particular class exist. As a result we easily distinguish the particular class using this sub domain. The great benefit of this approach is that it can handle multiple classes. The algorithm for discretization using genetic algorithm based approach is as follows:

1. [Start]: Generate a population comprising n random chromosomes.
2. [Fitness] Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. [New population] Create a new population by repeating the following steps until the new population is complete
   a. [Selection]**:** Select parent chromosomes randomly from a population. Selection probability will depend on the fitness of chromosomes.
   b. [Crossover]: Find the crossover point between two chromosomes and interchange the value of the two points.
   c. [Mutation]: The mutation operates on a single chromosome. Select a point between the sub domains and randomly generate a new value between the upper and lower limit of the two consecutive sub domains.
   d. [Merge domain]: Randomly select two consecutive domain and merge them.
   e. [Split domain]: This operates on a single chromosome. Select a domain randomly and split the domain into two sub domains.
   f. [Accepting]: Place new offsprings in the new population
4. [Replace] Use newly generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

Chromosome Encoding

The chromosome should in some way contain information about the solution which it represents. We use the decimal representation for the chromosome. Here total domain of a particular attribute is represented as sub domains. The chromosome representation is depicted in the following Figure 1.

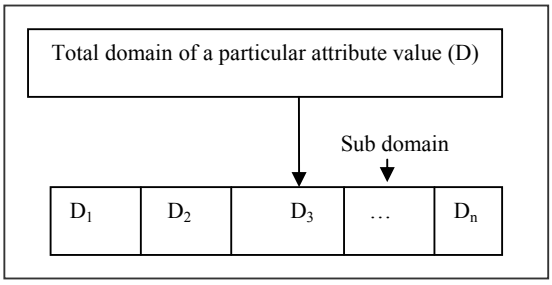

Fig. 1 Chromosome representation of a continuous attribute.

Operators

The various operators in use are discussed below.

Crossover operator

The crossover acts on two parent chromosomes. After randomly selecting two parent chromosomes it finds a crossover point between the two chromosomes and interchange this pint value.

Mutation

Mutation operates on a single chromosome. After selecting a random point on the chromosome it regenerates value between the up-

per and lower bound of the chromosome and replaces the previous value.

Domain Split

The domain split operates on a single chromosome. It randomly selects a domain and generates a number between upper and lower limit of this domain and split the domain by this value.

Domain Merge

The domain merge operation is opposite to domain split. It selects two consecutive domains and replaces them by a single domain.

Fitness Function

The performance of genetic algorithm depends on the fitness function. The more accurate the fitness function is the more chance to find an accurate solution. In this algorithm the fitness function is as follows:

$$\text{Fitness } \int(x) = 1 - \frac{\sum_{i=1}^{n} \frac{\min\_ event}{\max\_ event}}{total \_ domain}$$

Where i = 1, 2, 3 . . . n are the sub domains. min_event = total number of positive or negative examples whichever is minimum in a domain. max_event = total number of positive or negative examples whichever is maximum in a domain where $\max\_ event \neq 0$.

4. Genetic Algorithm Coupled With Entropy Based Approach

Genetic Algorithm coupled with Entropy based approach differs from the pure Genetic Algorithm based approach in fitness function. Here the operators are the same as previous algorithm. The Entropy measures the impurity of a set of examples. It is the lowest, if there is at most one class present, and it is the highest, if the proportions of all the present classes are equal. Entropy based methods use the class-information present in the data. The entropy (or the information content) is calculated on the basis of the class label. Entropy-based binning method finds the best split so that the bins are as pure as possible, i.e., the majority of the values in a bin correspond to having the same class label. Formally, it is characterized by finding the split with the maximal information gain.

We follow the notation presented in Dougherty et al [3]. Let *T* partitions the set *S* of examples into the subsets S_1 and S_2. Let there be k classed $C_1, \ldots, C_k$. Let $P(C_i,S_j)$ be the proportion of examples in S_j that have class C_i. The class entropy of a subset S_j, j=1,2 is defined as in Eq. (i)

$$Entropy \ \ (S_j) = -\sum_{i=1}^{k} P(C_i, S_j) \log(P(C_i, S_j)) \quad \text{(i)}$$

Suppose we have the following (attribute-value/class) pairs. Let S denotes the 9 pairs given here. *S* = (0, P), (4, P), (12, P), (16, N), (16, N), (18, P), (24, N), (26, N), (28, N). Let p = 4/9 be the fraction of pairs with class=P, and n = 5/9 be the fraction of pairs with class=N. The Entropy (or the information content) for S is defined for two classes

$$Entropy(S) = -p * \log(p) - n * \log(n)$$

If the entropy is small, then the set is relatively pure. The smallest possible value is 0. If the entropy is larger, then the set is mixed. The largest possible value is 1. Let v be a possible split, dividing the set S into two sets, S1 and S2. S_1: value <= v and S_2: value > v. Information of the split is defined by the following Eq. (ii):

$$I(S_1, S_2) = \frac{|S_1|}{|S|} * Entropy(S_1) + \frac{|S_2|}{|S|} * Entropy(S_2) \quad \text{(ii)}$$

and the information gain of the split is defined by the Eq. (iii)

$$Gain(v, S) = Entropy(S) - I(S_1, S_2) \quad \text{(iii)}$$

For example, if we want to split on the attribute-value, *v*=14. S_1= (0, P), (4, P), (12, P) and S_2= (16, N), (16, N), (18, P), (24, N), (26, N), (28, N)

I (S_1, S_2) = 0 + 6/9**Entropy (S2)* = 6/9 * 0.1963 = 0.1308.

*Gain (*14, *S) = Entropy(S)* - 0.1308.

The goal of this algorithm is to find the split with the maximum information gain. Maximal gain is obtained when *I (S1, S2)* is minimal. The best split(s) are found by examining all possible splits and then selecting the optimal split

5. Genetic Algorithm Based Concept Learner

There has been an increasing interest in applying GA methods to machine learning in the last few decades. One of the particular applications is supervised learning from examples of an attribute based space. The two most noticeable genetic algorithm approaches

to supervised concept learning in attribute-based spaces, in the Learning System framework, come from Koza [7, 8] and Spears with DeJong. Koza uses Lisp programs as a means of representing potential solutions, with some tree-based operators that are closed in the space of such representation. Spears and DeJong use a binary representation for multi-valued domains, and implement only the traditional operators of mutation and crossover in their GABIL system [8]. Another important LS system is SAMUEL [9], but it was designed and applied mostly to sequential decision problems. Classifier systems have also been used for concept learning [10].

Another remarkable work was done by Janikow [11] who devised a completely different method of using GA in concept learning. This method, called the "knowledge intensive genetic algorithm for supervised learning" is conceptually different from traditional GA. Inspired from his success, an attempt was made to use this system on a larger scale to solve more complex problems. However, study of this system revealed the need for some modifications. The limitations of this algorithm were inability to handle continuous attributes. This limitation prohibited the use of this system in broader problem solving tasks. We have improved this system by enabling it to handle continuous attributes. The discretization of continuous attributes has been done by the aforesaid two approaches. Description of the modified system is as follows:

A. Representation and Search Space

The multiple-valued logic VL_1 is adopted for representing the chromosome. Then the search space is the space of sets of rules, spanned by given features. This is the space of VL_1 concept descriptions. VL_1 is a widely used language, which is closely associated with rules. Variables (attributes) are the basic units having multi-valued domains. According to the relationship among different domain values, such domains may be of different types: nominal, for example {Yes, No} in Boolean attributes; linear with widely ordered values; or structured with partially ordered values.

B. Initial population

The population contains individuals, each of which is a potentially feasible solution (a set of rules of the VL_1 language). Its size remains fixed (as a parameter of the system). Initially the population must be filled with potential solutions. This process might be totally random (as is normally the case in genetic algorithms), or it might incorporate some task-specific knowledge. There is an obvious trade-off between the levels of knowledge used in such an intelligent initialization. On one side of the spectrum is the random choice, very cheap and simple. On the other side, there is an initialization that produces actual solution to the problem, differing possibly by some applied criteria. This later initialization is actually as hard as the problem to be solved. Therefore, it is inapplicable.

C. Evaluation Mechanism

The evaluation function must reflect the learning criteria. In supervised learning from examples, the criteria normally include completeness, consistency, and possibly complexity. In general, one may wish to accommodate some additional criteria, such as cost of attributes, length of descriptions, their generality etc., but any of them is not considered here.

The completeness and consistency of a rule or a rule set, measures its quality with respect to the set of training events. The formulas used are presented in *Table 1*. Here e^+/e^- is the number of positive/negative training events currently covered by a rule, $\in^+/\in^-$ is the number of such events covered by a rule set, and E^+/E^- is the total number of such events in the training set.

Table 1: Completeness and consistency measures.

Structure Type	Completeness	Consistency
A rule set	$\in^+/E^+$	$1-\in^+/E^+$
A rule	$e^+/\in^+$	$1-e^+/\in^+$

These two measures are meaningful only to rule sets and individual rules. For conditions,

the measures of the parent rule are used. The evaluation function:

$$correctness = (w_1 . Completeness + w_2 . Consistency) / (w_1 + w_2)$$

$$evaluation = correctness . (1 + w_3 . (1 - cost))^f$$

where w3 determines the influence of *cost* (which itself is normalized on [0,1]), and *f* grows very slowly on [0,1] as the population ages (a dynamic approach). The description's cost is measured by its complexity (*complexity = 2. #rules + #conditions*). The w1 and w2 coefficients bias the search to more complete or consistent descriptions. For most practical tests, a very low w3 weight (~ 0.01) was used. A dynamic approach is used to the use of cost (an approach that adjusts its effects as the population ages).

D. Operator

The operators transform chromosomes to new (possibly better) states in the search space. A rule set is represented as a disjunction of VL1 complexes. Each complex is a left-hand side of a rule corresponding to the same decision. Operators are applied at rule set, rule and

E. Rule set level

This is the level of sets of VL1 complexes. Here, the operators act on whole rule sets (one or two at a time):

Independent:

Rules exchange: This operator requires two parent rule sets, and it exchanges random rules between these two. It requires two parameters: probability of application to a rule set, and probability of rules selection.

Generalization:

Rules copy: Requires two parent rule sets, and it copies a random rule from each of the sets to each other. Parameter: Probability of application to a rule set.

New event: Acts on a single rule set. If there is a positive event not covered yet by the current rule set, this event's description is added to the set as a new rule. Parameter: Probability of application to a rule set.

Rules generalization: This operator acts on a single rule set. It selects two random rules and replaces them by their most specific generalization. It requires one parameter: Probability of application to a rule set.

Specialization:

Rules drop: This operator acts on a single rule set, and it drops a random rule from that set. It requires one parameter: Probability of application to a rule.

Rules specialization: This operator acts on a single rule set, and it replaces two random rules by their most general specialization. It requires one parameter: probability of application to a rule set.

F. Rule level

This is the level of VL1 complexes. Here, the operators act on one rule at a time.

Independent:

Rule split: Acts on a single rule, and it splits it into a number of rules. Parameters: Probability of application to a rule, and probabilities of a subset vs. all values split, separately for the linear and nominal data types.

Generalization:

Condition drop: This operator acts on a single rule, and it removes a present condition from that rule. Parameter: Probability of application to a rule.

*Turning conjunction into disjunction: A*cts on a single rule, and it splits the complex into a disjunction. Parameter: Probability of application to a rule.

Specialization:

Condition introduce: Acts on a single rule and it introduces a random condition associated with an unconditioned attribute. The new selector is a random choice from among all of its possible internal disjunction. Parameter: probability of application to a rule.

Rule directed split: Acts on a single rule. If this rule covers a negative event, it is split into a set of maximally general rules that are yet consistent with that event. Parameter: Probability of application to a rule

G. Condition Level

This is the level of VL1 selectors. The operators act on one condition at a time.

Independent:

Reference change: This operator acts on a single condition, and it randomly removes or adds a single domain value to this condition. It requires a single parameter: Probability of application to a condition.

Generalization:

Reference extension: This operator acts on a single condition and it extends the domain by allowing a number of additional values. Parameters: Probability of application to a con-

dition, and a number of selection probabilities determining the choice of an action.

H. Calculation of coverage vector

The coverage vector for the discrete conditions is calculated by OR-ing the feature coverage as in [11]. The concept of feature is still applicable in continuous attributes. What we have to do is to calculate which positive or negative training events fall within the range of the condition. For example, if a condition is A=1 which indicates a range 1-10, we will calculate the number of positive event and negative event within the range. Suppose, we have 15 positive examples for a particular attribute the positive examples 1, 3, 4, 7, 9 are within the range 1-10 of the attribute value (A=1) Therefore the positive coverage vector will be 101100101000000

Once the coverage vector of each condition of a rule is found, the coverage vector of the rule is simply calculated by AND-ing all the condition level vectors. To perform this operation, we need not distinguish between coverage vectors of discrete and continuous conditions. Then the coverage vectors of higher level structures (i.e., rule set level) are calculated as in [11].

6. Experiments And Results

To test the above ideas, a simple c prototype of the proposed approach is implemented. In this literature, learning systems are often evaluated and compared using a standard set of well-recognized artificial and real data. To evaluate our system, we use a number of data sets form the UCI repository. This also allows us to use published results of experimental sessions. The performance of this system is compared with another powerful learning algorithm, C5. The other system used in the experiment is C5 which is the latest version of C4.5 [12]. Table 2 reports the average error for each algorithm for the data sets. The results indicate a higher recognition rate in favor of our system. However, before drawing any conclusion, some relevant facts are independent. But this approach has some disadvantages. Firstly, since the learning sessions are sequential, the running time increases n fold. In addition, as each concept is learned independently, we are ignoring the recognition power of the combined concept. For instance, since every learned concept has should be considered. Firstly, due to lack of necessary data, significance tests on individual data sets were not performed. No other statistical methods were employed to evaluate the advantage of one algorithm over another. So the probability that the observed outcomes are obtained by chance could not be determined. Secondly, during cross-validation, it is assumed that the ratio of positive and negative events in each of the three subsets is the same of the full data set. This assumption is not impractical since any subset of a data set should reflect the characteristics of the whole data set. Otherwise, the learned concept would largely different form one subset to another, which is not desirable. It is not known whether the same assumption was made during the evaluation of the other algorithms.

Table 2: Mean Error for each algorithm.

Data Set	C5	Proposed Algorithm
Breast-cancer (Wisconsin)	0.09	0.08
Pima Diabetes	0.26	0.29
Echocardiogram	0.28	0.22
Housing	0.11	0.12
Hepatitis	0.20	0.15

7. Conclusion and Future Extension

Although the proposed algorithm can handle continuous attributes, still there exist some limitations: it can learn only single concept at a time. To broaden the applicability of the algorithm, it must be given the ability to learn multiple concepts. The simplest approach is to apply subsequent learning sessions of the same algorithm, one session per class. Then, during a learning session for class n, the examples of category n are considered as positive events, while the examples of all other categories are considered as the current negative events. This approach assumes that the different class descriptions some error in recognition, the same test event might be recognized as a positive event by more than one concept. If we could learn the concepts in parallel, this error could be eliminated.

8. References

[1] Chmielewski, M. R. & Grzymala-Busse, J. W., "*Global discretization of continuous attributes as preprocessing for machine learning",* in Third International Workshop on Rough Sets and Soft Computing, pp. 294—301, 1994.

[2] DeJong, K. A. & Spears, W. M., *"Using Genetic Algorithms for Supervised Concept Learning*", Proceedings of the IEEE AI Tools Conference, pp. 335-341, 1990.

[3] Dougherty. J, Kohavi, R. and Shahami, M., *"Supervised and Unsupervised Discretization on Continuous Features*", Machine Learning: Proceedings of the Twelfth International Conference, Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[4] Geoffery, I. Webb. Multiboosting, *"A technique for combining boosting and wagging. Machine Learning"*, 40(2): pp.159-196, 1999.

[5] Grefenstette, J. J., R. K. Belew, and L. B. Booker, *"Lamarckian learning in multi-agent environments*", Proceedings of the Fourth international Conference on Genetic Algorithms, Morgan Kaufmann, 1991.

[6] Janikow, C. Z., *"A knowledge Intensive Genetic Algorithm for Supervised Learning",* Machine Learning 13:189-228, 1993.

[7] Koza, J. R., *"Genetic programming: On the programming of Computers by Means of Natural Selection*", MIT Press, 1992.

[8] Koza, J. R., *"Genetic Programming II: Automatic Discovery of Reusable Programs"*, MIT Press, 1994.

[9] Michalski, R. S. & Stepp, R. E., *"Learning from observations: Conceptual clustering",* in T. M. M. R. S. Michalski, J. G. Carbonell, ed., "Machine Learning: An Artificial Intelligence Approach", Tioga, Palo Alto, 1983.

[10] Quinlan, J. R. , *"C4.5: Programs for Machine Learning"*, Morgan Kaufmann, Los Altos, California, 1993.

[11] Weiss, S. M. & Kulikowski, C. A., *"Computer Systems that Learn, Morgan Kaufmann",* San Mateo, CA. 1991.

[12] Wilson, S. W. *"Classifier systems and the animate problem".* Machine Learning, 2, pp. 199-228, 1987.