STUDY ON DISCRETIZATION IN ROUGH SET BASED ON GENETIC ALGORITHM

CAI-YUN CHEN, ZHI-GUO LI, SHENG-YONG QIAO, SHUO-PIN WEN

Center for Combinatorics, LPMC, Nankai University, Tianjin, 300071, P. R. China
E-MAIL: sbickle@eyou.com

Abstract:
Discretization of attributes with real values in rough set is an important problem in data mining. It is different from the traditional discretization which has particular characteristic. Nguyen S. H has given a detailed description about discretization in rough set. This paper gives a genetic algorithm aimed at discretization proposed by Nguyen S. H. And at the same time we make an experiment on several datasets from UCI Machine Learning Repository by this method. During the experiment, we constantly optimized genetic algorithm by adopting some optimization strategies. The experiment has proved that using genetic algorithm to solve discretization of rough set is efficient whatever in time complexity or in accuracy.

Keywords:
Discretization; Rough Set; Genetic Algorithm

1. Introduction

The discretization of real value attributes is one of the important problems to be solved in data mining, especially in rough set. We know, when the value set of any attribute in a decision table is a continuous value or a real number, and then it is likely that there will be few objects that will have the same value of the corresponding attributes. In such a situation the number of equivalence classes based on that attribute will be large and there will be very few elements in each of such equivalence class, which will lead to the generation of a large number of rules in the classification of rough set, thereby making rough set theoretic classifiers inefficient. Discretization is a process of grouping the values of the attributes in intervals in such a way that the knowledge content or the discernibility is not lost. Many discretization approaches have been developed so far. Nguyen S. H had given some detailed description about discretization in rough set in reference [1]. He gave the complexity of discretization problem and proved that it is an NP-hard problem. And at the same time he also proposed a basic heuristic algorithm based on rough set and Boolean reasoning (for convenience of citation, we call this algorithm basic heuristic algorithm), which brought great improvement in dealing with discretization problem in rough set. Aimed at the particularity of discretization problem, this paper gives a genetic algorithm (for short GA) for discretization in rough set. Its main idea is to find possibly minimum number of discrete intervals by constantly adopting some optimization strategies during the experiment. The experiment shows that the GA is more efficient in both accuracy and time complexity than basic heuristic algorithm.

This paper is organized by 6 sections. Description of discretization in rough set is introduced in section 2. In section 3 we construct a genetic algorithm for discretization problem. In section 4, some optimization strategies for genetic algorithm are showed. Section 5 some experiments for discretization are done by two methods including genetic algorithm and basic heuristic algorithm using several datasets from UCI Machine Learning Repository and the decision table in reference [1], we give an analysis about genetic algorithm for discretization. Section 6 gives a summary of this paper.

2. Description of discretization in rough set

A decision table is composed of a 4-tuple as follows: 
S = < U, Q \cup \{d\}, V, f >, where 
U: a finite set of N objects \{x_1, x_2, \ldots, x_N\}; 
Q : a finite set of n condition attributes \{q_1, q_2, \ldots, q_n\} (a nonempty set), and d is decision attribute; 
V = \bigcup_{q \in Q} V_q, where V_q is a domain of the attribute q; 
f : U \times Q \cup d \rightarrow V is the total decision function called information function such that f(x, q) \in V_q for every q \in Q \cup d, x \in U. The decision table can be represented as a finite data table, in which the columns are labeled by
attributes, the rows by objects and the entry in column \( q_j \) and row \( x_i \) has the value \( f(x_i, q_j) \). Each row in the table describes the information about some objects in \( S \).

We assume \( V_q = \{ (l_q, r_q) \} \subset R \), where \( R \) is the set of real numbers, and we also assume that \( S \) is consistent decision table\(^{[4]} \). The following notion and description about discretization is referred to reference \([2]\). First let us give the definition of cut.

**Definition 1** Any pair \((q, c)\), where \( q \in Q \) and \( c \in R \), defines a partition of \( V_q \) into left-hand-side and right-hand-side interval. The pair \((q, c)\) is called a cut on \( V_q \).

For an attribute \( q \in Q \),

\[
D_q = \{ (q, c^1_q), (q, c^2_q), \ldots, (q, c^k_q) \} \]

is composed by all the cuts, where \( k, q \in N \), and \( l_q = c^0_q < c^1_q < c^2_q < \ldots < c^k_q < c^k_{q+1} = r_q \), defines a partition on \( V_q \) into subintervals i.e. \( V_q = [c^0_q, c^1_q] \cup [c^1_q, c^2_q] \cup \ldots \cup [c^k_q, c^k_{q+1}] \). So any set of cuts on condition attributes \( D = \bigcup D_n \) transforms the original decision table \( S \) into discrete decision table \( S' = < U, Q \cup \{ d \}, V', f' > \), where \( f'(x, q) = i \iff f(x, q) \in [c^i_q, c^{i+1}_q] \), and \( x \in U, i \in \{ 0, 1, \ldots, k \}, q \in Q \).

After discretization, the original decision table is replaced with the new one. And different sets of cuts will construct different new decision table. It is obvious that discretization process is associated with loss of information. Usually the task of discretization is to determine a minimal set of cuts from a given decision table and keep the discernibility. The selected cuts can be evaluated by the following criteria\(^{[2]}\):

- Consistency of \( D \). For any objects \( u, v \in U \), they are satisfying if \( u \), \( v \) are discerned by \( Q \), then \( u \), \( v \) are discerned by \( D \);
- Irreducibility. There is no \( D' \subset D \), satisfying the consistency;
- Optimality. For any \( D' \) satisfying consistency, it follows \( \text{card}(D') \leq \text{card}(D) \), then \( D \) is optimal cuts.

Nguyen S. H had proved that the optimal discretization problem is an NP-hard problem. As we know, genetic algorithm is a better method for solving this kind of optimization problem. In the next section, we will give a design of genetic algorithm for discretization problem.


Genetic algorithm(for short GA), introduced by Holland in \([3]\), have been used by many researchers as a tool for search and optimization. It is a search algorithm developed to explain and simulate the mechanisms of natural systems: genetics and natural selection. The "objective" of the solution is its performance in a survival competition. Genetic algorithm starts with a population of solutions. These solutions compete against each other, and the survivors of the competition reproduce to form new solutions by exchanging partial characteristics of their solutions. After generations, these survivors become similar to each other. With the random initial population, random crossover and mutation, different replications of a genetic algorithm may produce different solutions.

To beginning, we give some terms description about genetic algorithm. A population consists of a fixed number of chromosomes. Each chromosome is a fixed-length string of genes. More formally, the genetic algorithm is defined as an 8-tuple::

\[
GA = \left( C, E, P_0, M, \Phi, \Gamma, \Psi, \Upsilon \right)
\]

where:
- \( C \) - encoding of chromosome;
- \( E \) - the fitness function of every chromosome;
- \( P_0 \) - the initial population;
- \( M \) - the scale of population;
- \( \Phi \) - the selection operator;
- \( \Gamma \) - the crossover operator;
- \( \Psi \) - the mutation operator;
- \( \Upsilon \) - the terminating criterion of GA.

By this definition, we use a GA to find the minimal set of cuts for discretization in the rough set.

We have to do some preprocessing work before designing the genetic algorithm. First, we construct a new decision table \( S' = < U', Q' \cup \{ d \}, V', f' > \) from a given original decision table \( S = < U, Q \cup \{ d \}, V, f > \) as follows:

\[
U' = \{ (x_i, x_j) \in U \times U | d(x_i) = d(x_j) \};
\]

\[
Q' = \{ c^i_q | i \in \{ 0, 1, \ldots, k \}, q \in Q \}, \text{where } c^i_q \text{ is the cut of attribute } q;
\]

for \( u = (x_i, x_j) \in U' \), \( c \in Q' \) we have:
The solution space is the set of all possible combinations of the cut c can discern. By the definition of f', if f'(u,c) = 1 we say that the cut c can discern u, or we say u can be discerned by cut c, otherwise we say that cut c can't discern u, or u can not be discerned by c.

Now we begin to design the genetic algorithm.
1. Encoding the points in the solution space. (The encoding length is L)
   For the GA to be described in this paper, we use the following representation for a point in the solution space. The solution space is the set of all possible combinations of L prime implicants (hence the size of the solution space is $2^L - 1$, excluding the empty set). We use L-length binary encoding for every chromosome. An L-bit chromosome represents a particular choice of cuts. In this L-bit chromosome, a value 1 in the i-th bit (where $i \in \{1, \ldots, L\}$) implies that the i-th cut is chosen for discretization of rough set. A value 0 in the i-th bit means that the i-th cut is not chosen. For example, an L-bit string '01011110...' can be looked as a chromosome. If these cuts corresponding to these genes whose values are 1 in a chromosome can discern all the elements in $U^*$, we say the chromosome can discern all the elements in $U^*$, otherwise we say the chromosome can not discern all the elements in $U^*$.

2. Determining the scale M of the population
   By experience, we know, M is larger, the result is better, but it will cost more time, and vice versa. Here we let M=50 by experiment. The initial population denoted by $P_0$ is composed of M chromosomes which are created randomly. And we denote the initial generation by $\bar{X}(0) = (X_1(0), X_2(0), \ldots, X_M(0)) \in H_M^L$ where $H_M^L$ is population space, $X_i(0), (i = 1, 2, \ldots, M)$ is a initial chromosome.

3. Designing the crossover operation and the mutation operation
   The crossover operation leads to an increased diversity of the population of strings as a new individuals emerge out of this process. The intensity of crossover is characterized in terms of the probability at which the elements of the strings are affected. The higher the probability, the more individuals are affected by the crossover. The best range of this probability $P_c$ is between 0.4 and 0.7, and in this paper we choose $P_c=0.7$ by the experiment.

   The mutation operator is an example of an operator adding an extra diversity of a stochastic nature. In binary strings this mechanism is implemented by flipping the values of some randomly selected bits. Again, the mutation rate $P_m$ is related to the probability at which the individual bits become affected. And in our experiments, we set $P_m = 0.02$.

   Generally, the operation of crossover can be viewed as a special type of recombination operation which involves two parents (strings) and leads to two offspring.

4. Constructing the fitness function of GA
   In this paper, our aim is to find the minimal set of cuts which can discern all the pairs in $U^*$. Since the fitness function must be positive, we choose the fitness function as:
   $$ F(X) = e^{-\frac{1}{L}} $$
   where:
   $$ X = g_1, g_2, \ldots, g_L $$
   is a chromosome which is composed by L genes $g_i$ ($i = 1, 2, \ldots, L$), $g_i = 0$ or 1.

5. Setting the terminating criterion
   The simplest terminating criterion is a predetermined number of generations (or objective function evaluations). So we terminate GA after 1000 generations. And the optimal set of cuts is the one whose value of its fitness function is minimized.

4. Optimization Strategies

   To help GA finding the best solution faster, some optimization strategies have been adopted.

   (1) Elitist selection and father-offspring combined selection strategy
   To make GA get the best solution faster, we apply the Elitist selection and father-offspring combined selection strategy into our algorithm. The elitist selection and father-offspring combined selection can help GA to find the best solution.\textsuperscript{[16]} The algorithm is shown as follows:
   **Elitist Selection and Father-Offspring Combined Selection GA**
   Step 1 (starting) set $t \leftarrow 0$, selecting M chromosomes $X_1(0), X_2(0), \ldots, X_M(0)$ randomly, and evaluating
Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, 2-5 November 2003

been generated into the initial population and hope to further optimize the solution. This strategy is different from the strategy which onlv increases the times of generation.

Step 2 (Evolution)

2.1 Select M pairs of “parents” chromosomes from current generation $X(t)$.

2.2 M “offspring” chromosomes will be generated from these M pairs of chromosomes by “crossover” operation.

2.3 Next, by mutation operation for these M “offspring” chromosomes. M new chromosomes have been generated:

$$X(t+1) = X_1(t+1), X_2(t+1), ..., X_M(t+1)$$

2.4 Select M-1 chromosomes $X_1(t+1), X_2(t+1), ..., X_{M-1}(t+1)$ from $X(t) \cup X'(t+1)$, and at the same time we choose the optimal chromosome $X^*(t)$ from $X(t) \cup X'(t+1)$, and let $X_M(t+1) = X^*(t)$. So we get the (t+1) generation:

$$X(t+1) = X_1(t+1), X_2(t+1), ..., X_M(t+1)$$

Step 3 (Terminating criterion)

If the algorithm has reached the terminating criterion, we will output the optimal chromosome $X^*(t+1)$ from the (t+1)-th generation as the solution of this problem, otherwise we set $t = t + 1$ and go to step 2.

We use the second strategy: restart strategy.

2. Restart strategy

In our algorithm, we put the best solution which has been generated into the initial population and hope to further optimize the solution. This strategy is different from the strategy which only increases the times of generation. The former can further optimize the second best solution according to the diversity of the initial population. Because all the chromosomes are almost similar in the later period of GA, and it is difficult to further optimize the best solution by only increasing the times of generation, the former strategy is more effective than the later one.

This strategy can also avoid the “precocious” phenomena to some extent.

The third strategy: Penalty strategy.

3. Penalty strategy

Because the problem of discretization in rough set is a constrained optimization problem, i.e. not every L-bit string (a chromosome) which is created randomly is useful for finding the minimal set of cuts of the rough set. We call such a chromosome an incomplete solution for our problem. In order to penalize this kind of chromosome, for each of the newly added cut, a cost $\alpha$ is multiplied to fitness function of the chromosome, where $0 < \alpha < 1$. In our work, we set $\alpha = 0.8$ to eliminate the incomplete solutions. In fact, when the chosen chromosome can not discern all the elements of $U^*$, we resort to a sequential search starting at the first cut and including the cuts that can discern the objects which haven’t been discerned by any cuts. This process continues until all objects are discerned.

5. Experiments study

In our experiments, first, we compare the result obtained by GA and the basic heuristic algorithm on the decision table in reference [1]. The result of GA is \{ $P_2, P_4, P_5$ \} while the result of the basic heuristic algorithm is \{ $P_2, P_3, P_4, P_5$ \}, because there exist more than one columns whose number of occurrences of 1's is maximal in decision table $S^*$, it is difficult for the basic heuristic algorithm to choose the optimum result. And generally speaking the result obtained by GA is better than that of the basic heuristic algorithm. Next we proved that using GA is feasible dealing with the large datasets on several datasets of UCI Machine Learning Repository such as glass, iris, pima and wine (http://www.ics.uci.edu/~mlearn/ MLRepository.html). In the following table(table 1), we give the time complexity of GA.

<table>
<thead>
<tr>
<th>Table 1 the results of our experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
</tr>
<tr>
<td>Glass</td>
</tr>
<tr>
<td>Iris</td>
</tr>
<tr>
<td>Pima</td>
</tr>
<tr>
<td>Wine</td>
</tr>
</tbody>
</table>

The most complex part of GA is the part of checking and revising the incomplete solutions. In fact, the complexity of the procedure of checking is $O(|U^*|)$, while to revise the incomplete solution may reach $O(|U^*|^2)$. When $|U^*|$ becomes larger and larger, the runtime of GA will become longer. In our experiments we notice that when $|U^*|$ is large, GA is slow in the first several generations, but it becomes faster and faster in the following generation, so the
total time is not so long. We have found the reason of this phenomenon is that the first generation is created at random, i.e. the “father” chromosomes are not very similar, thus their “offspring” chromosomes have a higher probability to become incomplete. But in the late period of GA, all the chromosomes are similar, so the “children” chromosomes are very “like” their “parents”, so the probability of being an incomplete solution is lower, and the speed of GA becomes faster in the later period.

From above on, we can draw the conclusion that GA is effective to solve the problem of the discretization in rough set.

6. Conclusion

Discretization in rough set is different from the traditional discretization. Generally speaking, we should find possibly minimum number of cut, and at the same time it should not weaken the indiscernibility. Aimed at its particularity, we give a genetic algorithm for the discretization. We do the experiments by several datasets from the UCI Machine Learning Repository using GA. The experiment has proved that the GA is efficient no matter in accuracy or in time complexity.

Acknowledgments

This work was done under the auspices of the National “973” Project on Mathematical Mechanization and the National Science Foundation of China.

References