CHAPTER 4

DISCOVERING CLASSIFICATION RULE IN DATA MINING

4.1 PREFACE

The data mining classification using wide range of human activities when some decision or forecast is made on the basis of currently available information and a classification procedure (or classification rule) is then some formal method for repeatedly making such judgments in new situations. In machine learning classification is performed as inductive learning, or induction that is generalization from the known to the unknown, so that appropriate responses to the unknown can be formulated when it appears. The standard data mining classification algorithm such as decision tree induction algorithm, rule based classifier, nearest-neighbour classifiers, Bayesian Classification, support-vector machines, and neural networks.

4.2 Decision Tree Induction Algorithm

The C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan 1993 [196]. The decision tree is a classifier that is depicted in a flowchart-like tree structure, which has been widely used to represent classification models, due to its comprehensible nature that resembles the human reasoning. Decision-tree induction algorithms present several advantages over other learning algorithms, such as robustness to noise, low computational cost for the generation of the model, and ability to deal with redundant attributes.

An example decision tree is shown fig 4.1. The first (decision node) is made based on the wage: if the attribute example’s Wage value is less than or equal to 30k then it is sent to
the second level left hand node, if it is greater than 30k is sent down the right hand edge and arrives at the leaf Credit=Good. This means that in the example decision tree any example having Wage > 30k will be assigned to the class good Credit, whereas if an example has Wage ≤ 30K its class will be Bad or Good Credit, depending on whether or not the example’s Mortgage value is yes or no, respectively. Decision trees and rule sets are interchangeable (as stated previously). Converting a decision tree to a rule set simply requires generating one rule per possible path from the root node to each leaf. For example the decision tree could be converted to the following (un-pruned and unordered) rule set:

IF (Salary > 30k) THEN (Good Credit)

IF (Salary ≤ 30k) AND (Mortgage = No) THEN (Good Credit)

IF (Salary ≤ 30k) AND (Mortgage = Yes) THEN (Bad Credit)

Fig 4.1 Example for Decision Tree Induction
4.2.1 Decision Tree Pseudo Code

**Step 1**: Check for base cases.

**Step 2**: For each attribute $a$.

Find the normalized information gain from splitting on $a$.

**Step 3**: Let $a_{best}$ be the attribute with the highest normalized information gain.

**Step 4**: Create a decision node that splits on $a_{best}$.

**Step 5**: Recurse on the sublists obtained by splitting on $a_{best}$, and add those nodes as children of node.

4.2.2 Rule Based Classifier

A best rule is found best from a data set and then all records covered (or explained) by the rule is removed from the data set. This procedure repeated until there is no record left in the data set. The way to find the best rule is usually by some heuristics technique. Rule induction algorithms generate rules representing patterns in the training set. These rules are then applied to a test set with examples of unseen (by the classification algorithm) or unknown classes. In classification the knowledge or patterns discovered in a data set can be represented in terms of a set of rules. A rule consists of an antecedent (a set of attribute-values and logical operators) and a consequent (class):

$$\text{IF } <\text{attrib operator value}> \text{ AND...}$$

$$\text{AND } <\text{attrib operator value}> \text{ THEN } <\text{class}>$$

The consequent of the rule is the class that is predicted by that rule. The antecedent consists of a set of terms, where each term is essentially an attribute-value pair.

More precisely, a term is defined by a triple $<\text{attribute}, \text{operator}, \text{value}>$, where value is a value belonging to the domain of attribute. The operator typically is “=” in the case of
categorical attributes, or “<“ and “≥” in the case of continuous attributes. The logical operation that is performed on these terms is AND in this example, however, it could be any other operation, i.e., NAND, XOR etc. The knowledge representation in the form of rules has the advantage of being intuitively comprehensible to the user. An example rule might be:

IF (Salary > 30k) AND (Mortgage = No) THEN (Good Credit) Sequential covering approach [183] [258] used to build a set of classification rules covering; it is one of the rule induction algorithms.

RS = [] /* initially, Rule Set is empty */

TS = {all training examples}

FOR EACH Class C

WHILE (number of uncovered training examples of class C > MaxUncovExampPerClass)

    Discover the best rule predicting class C, called

    Best Rule

    RS = RS U {Best Rule}

    TS = TS – {training examples correctly covered by Best Rule}

END WHILE

END FOR

The algorithm starts by initialising the rule set (RS) with the empty set and initialising the current training set with all the training examples. Then, for each class the algorithm performs a WHILE loop.
Each iteration of this loop performs one run of the rule discovery algorithm, returning the best discovered rule predicting examples of the current class (C). This rule is added to the rule set, and the examples correctly covered by that rule are removed from the training set (TS). An example is said to be correctly covered by a rule if that example satisfies all the terms (attribute-value pairs) in the rule antecedent (“IF part”) and it has the class predicted by the rule (“THEN part”). This WHILE loop is performed as long as the number of uncovered examples of the class C is greater than a user-defined threshold, the maximum number of uncovered examples per class (MaxUncovExampPerClass). After discovering rules for all classes, the algorithm returns RS, the discovered rule set.

4.2.3 Bayesian Classification

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. This Classification is named after Thomas Bayes (1702-1761), who proposed the Bayes Theorem [12]. Bayesian classification provides practical learning algorithms and prior knowledge an observed data can be combined. Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data. Bayesian reasoning is applied to decision making and inferential statistics that deals with probability inference. It is used in many applications probabilistic learning method, Spam filtering, unseen information and can predict whether a user would like a given resource and online application.

The Bayesian theorem represented by

\[
P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}
\]
The basic idea of Bayes's rule is that the outcome of a hypothesis or an event (H) can be predicted based on some evidences (E) that can be observed. From Bayes's rule, we have

(1) \textit{A priori} probability of H or P (H): This is the probability of an event before the evidence is observed.

(2) \textit{A posterior} probability of H or P (H \mid E): This is the probability of an event after the evidence is observed.

### 4.2.4 Support Vector Machine (SVM)

It was first heard in 1992, introduced by Boser, Guyon, and Vapnik in COLT-92. SVMs are a set of related supervised learning methods used for classification and regression [92]. They belong to a family of generalized linear classifiers. In another terms, SVM is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data.

SVM can be defined as systems which use hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. Support vector machine was initially popular with the Neural Information Processing System (NIPS) community and now is an active part of the machine learning research around the world. It is also being used for many applications, such as hand writing analysis, face analysis and so forth, especially for pattern classification and regression based applications.

The foundations of SVM have been developed by Vapnik [246] and gained popularity due to many promising features such as better empirical performance. A classification task usually involves with training and testing data which consist of some data instances [58]. Each instance in the training set contains one target values and several attributes. The goal of
SVM is to produce a model which predicts target value of data instances in the testing set which are given only the attributes [169].

Classification in SVM is an example of Supervised Learning. Known labels help indicate whether the system is performing in a right way or not. This information points to a desired response, validating the accuracy of the system, or be used to help the system learn to act correctly. A step in SVM classification involves identification as which are intimately connected to the known classes. This is called feature selection or feature extraction. Feature selection and SVM classification together have a use even when prediction of unknown samples is not necessary. They can be used to identify key sets which are involved in whatever processes distinguish the classes [169].

4.2.5 K-Nearest Neighbour Classifier

Nearest neighbor classifiers are based on learning by analogy. The training samples are described by n dimensional numeric attributes. Each sample represents a point in an n-dimensional space. In this way, all of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a k-nearest neighbor classifier searches the pattern space for the k training samples that are closest to the unknown sample. "Closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points,

\[ X = (x_1, x_2, x_3, \ldots, x_n) \] and \[ Y = (y_1, y_2, y_3, \ldots, y_n) \]

\[ d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \]
The unknown sample is assigned the most common class among its k nearest neighbors. When k=1, the unknown sample is assigned the class of the training sample that is closest to it in pattern space. Nearest neighbor classifiers are instance-based or lazy learners in that they store all of the training samples and do not build a classifier until a new (unlabeled) sample needs to be classified. This contrasts with eager learning methods, such as decision tree induction and back propagation, which construct a generalization model before receiving new samples to classify.

Lazy learners can incur expensive computational costs when the number of potential neighbors (i.e., stored training samples) with which to compare a given unlabeled sample is great. Therefore, they require efficient indexing techniques. Expected lazy learning methods are faster data training than eager methods, but slower at classification since all computation is delayed to that time. Unlike decision tree induction and back propagation, nearest neighbor classifiers assign equal weight to each attribute. This may cause confusion when there are many irrelevant attributes in the data.

Nearest neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown sample. In this case, the classifier returns the average value of the real-valued associated with the k nearest neighbors of the unknown sample.

The K-NN is amongst the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If k = 1, then the object is simply assigned to the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes.
The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its \( k \) nearest neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidian distance, though other distance measures, such as the Manhattan distance could in principle be used instead. The \( k \)-nearest neighbor algorithm is sensitive to the local structure of the data.

### 4.2.6 Discovering Classification Rules in Data Mining

The discovering classification rule in data mining using enhanced PSO, enhanced ACO and GA.

#### 4.2.6.1 Particle Swarm Optimization

PSO is a technique used to explore the search space of a given problem to find the settings or parameters required to maximize a particular objective. This technique, first described by James Kennedy and Russell C. Eberhart in 1995 [59], originates from two separate concepts: the idea of swarm intelligence based off the observation of swarming habits by certain kinds of animals (such as birds and fish); and the field of evolutionary computation.
The PSO algorithm works by simultaneously maintaining several candidate solutions in the search space. During each iteration of the algorithm, each candidate solution is evaluated by the objective function being optimized, determining the fitness of that solution. Each candidate solution can be thought of as a particle “flying” through the fitness landscape finding the maximum or minimum of the objective function.

In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called \( p_{best} \). Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbours of the particle. This location is called \( l_{best} \). When a particle takes all the population as its topological neighbours, the best value is a global best and is called \( g_{best} \). The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its \( p_{best} \) and \( l_{best} \) locations (local version of PSO).

Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward \( p_{best} \) and \( l_{best} \) locations. PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called \( p_{best} \). Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population.

This best value is a global best and called \( g_{best} \). When a particle takes part of the population as its topological neighbors, the best value is a local best and is called \( l_{best} \). After
finding the two best values, the particle updates its velocity and positions with following equation (1) and (2).

\[
V[i] = V[i] + c_1 \times \text{rand}(\cdot)(pbest[i] - \text{present}[i]) + c_2 \times \text{rand}(\cdot)(gbest[i] - \text{present}[i])
\]

\[
\text{present}[i] = \text{present}[i] + V[i]
\]

The \(V[i]\) particle velocity \(\text{present}[i]\) is the current particle (solution). The \(pbest[i]\) and \(gbest[i]\) are defined as stated before. The \(\text{rand}(\cdot)\) is a random number between \((0, 1)\). \(c_1, c_2\) are learning factors. Usually \(c_1 = c_2 = 2\).

**Summarization of Particle Swarm Optimization**

**Step1:** Initialize: Initialize parameters and population with random position and velocities.

**Step2:** Evaluation: Evaluate the fitness value for each particle.

**Step3:** Find the pbest: If the Fitness value of particle \(i\) is better than its best fitness value (pbest) in history, then set current fitness value as the new pbest to particle.

**Step4:** Find the gbest: If any pbest is updated and it is better than the current gbest, then set gbest to the current value.

**Step5:** Update velocity and position: Update velocity and move to next position according to equation (1) and (2).

**Step6:** Stopping Criterion: If the number of iteration or CPU time are met, then stop; otherwise go back to step 2.
PSO Algorithm Variants

The PSO has various variants such as Inertia Weight, Constriction Factor, Dynamic Inertia with Maximum Velocity Reduction and Discrete Optimization.

Inertia Weight

Inertia weight is a very important parameter in standard PSO algorithm which can be used to control the exploitation and exploration ability of algorithm. Its value determines how much it succeeds current rate: the bigger the inertia weight of algorithm is, the greater the speed of particle gets and thus particle has stronger exploration ability; the smaller the inertia weight is, the stronger the exploitation ability of particle is [169]. Inertia weight plays a key role in this process and can direct PSO to optimize algorithm. Indeed, the inertia weight determines the contribution rate of a particle’s previous velocity to its velocity at the current time step.

Constriction Factor

The convergence of PSO have introduced a constriction factor indicates that the use of a constriction factor may be necessary to insure convergence of the particle swarm algorithm. He had established some mathematical foundation to explain the behaviour of a simplified PSO model in its search for an optimal solution.

\[
v_{id}^{t+1} = k \left[ v_{id}^{t} + c_{1} r_{1} (P_{id} - x_{id}^{t}) + c_{2} r_{2} (P_{gd} - x_{id}^{t}) \right]
\]

\[
x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}
\]

Where the construction factor

\[
k = \frac{2}{\sqrt{2 - \phi - \sqrt{\phi^2 - 4\phi}}} - 4\phi
\]

Where \(\phi = c_{1} + c_{2}\)
**Discrete PSO**

A Discrete Particle Swarm Optimization (DPSO) proposed in does not consider any velocity since, from the lack of continuity of the movement in a discrete space, the notion of velocity loses sense; however they kept the attraction of the best positions. They interpret the weights of the updating equation as probabilities that, at each iteration, each particle has a random behaviour, or acts in a way guided by the effect of an attraction. The moves in a discrete or combinatorial space are jumps from one solution to another.

### 4.2.6.2 Genetic Algorithm

The major steps involved are the generation of a population of solutions, finding the objective function and fitness function and the application of genetic operators.

**Summarization of Genetic Algorithm**

- **Step 1**: Formulate initial population
- **Step 2**: Randomly initialize population
- **Step 3**: Repeat
- **Step 4**: Evaluate objective function
- **Step 5**: Find fitness function
- **Step 6**: Apply genetic operators
- **Step 7**: Reproduction
- **Step 8**: Crossover
- **Step 9**: Mutation

Until stopping criteria
Crossover

The idea behind a crossover operation is as follows: it takes as input 2 expressions, selects a random point, and exchanges the sub expressions behind this point. In general, however, not all attributes will be involved in an expression. This may have some undesired effects for a crossover. First, a crossover may produce individuals in which an attribute is involved more than once. Second, a crossover may result in an offspring that is exactly the same as the parents with probability 1.0.

To prevent the above-mentioned effects, we apply the following technique to perform crossover. If two individual $p_1$ and $p_2$ have been selected for crossover, the crossover operator is considered as follow:

1. If there are some same attributes between individual $p_1$ and $p_2$, then select one randomly among these attributes and exchange the corresponding term between the two individuals.
2. If there aren’t same attributes between individual $p_1$ and $p_2$, then select one attribute randomly from individual $p_1$ and $p_2$ respectively and exchange the corresponding term between the two individuals.

Mutation

Mutation is an important operator that acts a single individual at a time. This operator maintains the diversity of gene in the population and guarantees that the search is done in the whole solution space. Through mutation operator cannot always produce a better result, it play an important role for the global optimization. Considering an individual $p$ with a length of $n$, the mutation operator is defined as:
(1) When \( n=L \), select one attribute randomly from individual \( p \) and delete the corresponding term.

(2) When \( L>n>1 \), select one attribute randomly from individual \( p \) and calculate the fitness of individual \( p \). If \( F(p) \neq 0 \), then delete the corresponding term or replace the attribute value by another on the mutation probability of \( pm \). There the two operators are select irrespectively on the probability of 50%. If \( F(p) = 0 \), then delete the corresponding term.

(3) When \( n=1 \), replace present attribute value by another on the probability of \( pm \).

**Selection**

For simplifying the implementation of a genetic algorithm, the mechanism to select individuals for a new generation is based on the technique of elitist recombination. According to this technique, the individuals in a population are randomly shuffled. Then, the cross-over operation is applied on each mating pair, resulting into two offspring. The parent and the offspring with the highest fitness value are selected to be mutated with a probability. Then they are added to the new generation. In this way, there is a direct competition between the offspring and their own parents and the offspring provides the possibility to explore different parts of a search space.

**4.2.6.3 Ant Colony Optimization**

ACO was inspired by the observation of real ant colonies, in particular, by their foraging behaviour. Real ants are capable of finding shortest or near-shortest paths between a food source and their nest. While walking from food sources to the nest and vice versa, ants deposit on the ground a substance called pheromone, forming in this path a pheromone trail.
When they arrive at a decision point, like the intersection between the shorter and the longer branches, they make a probabilistic choice biased by the amount of pheromone they smell on the two branches.

It is illustrated in the following figure. What happens when an ant colony following a shortest path between a food source and the nest, it is illustrated in the figure. The ant gets interrupted when the ants reach the obstacle they will randomly choose some way around it (right, left, over or under). If we assume that the only way around the obstacle is either right or left, we can safely assume that approximately half of the ants will go right and the other half left, as illustrated below. The ants that happen to pick the shorter path will obviously create a strong trail of pheromone a lot faster than the ones choosing a longer path. This will cause more and more ants to choose the shorter path until eventually all ants have found the shortest path.

Fig 4.2(a) Ants Moving in Straight Line
Ant Colony Algorithms attempt somehow to apply similar techniques in order to solve real life problems. The main idea is to use repeated and often recurrent simulations of artificial ants (mobile agents inspired by real ant behaviour) to generate new solutions to the problem at hand. The ants use information collected during past simulations to direct their search and this information is available and modified through the environment. Many different artificial ant algorithms have been implemented and no universal definition of an artificial ant fits them all.
Pseudo code for ACO algorithm

TrainingSet = \{all training cases\};
DiscoveredRuleList = [];

WHILE (TrainingSet >= Max_Uncovered_Cases)

i = 1; /* ant index */
No_Ants_Converg = 1;

Initialize all trails with the same amount of pheromone;

REPEAT

Anti starts with an empty rule and incrementally constructs a
classification rule Ri, by adding one term at a time to the current
rule;

Prune rule Ri;

Update the pheromone of all trails, by increasing pheromone in the trail followed by
Ant i and decreasing pheromone in the other trails;

IF (Ri is equal to Ri – 1) /* update convergence test */

THEN No_Ants_Converge = No_Ants_Converge + 1;
ELSE No_Ants_Converge = 1;

END IF

i = i + 1;

UNTIL (i >= No_of_Ants) OR
(No_Ants_Converg >= No_Rules_Converg)

Choose the best rule Rbest among all rules Ri constructed by all the ants;

Add rule Rbest to DiscoveredRuleList;
TrainingSet = \{\text{set of cases correctly covered by } R_{\text{best}}\}

\text{END WHILE}

Result

4.2.7 Hybrid Enhanced ACO, Enhanced PSO, GA for Discovering Classification Rule

4.2.7.1 Preprocessing

In the breast cancer data set nominal attribute breast taking values “left-up”, “left-low”, “right-up”, “right-low” and “central”, it is converted in to numerical values. For instance left-up=1, left-low=2, right-up=3, right-low=4, central=5. Age attribute taking values “10-19”, “20-29”, “30-39”, “40-49”, “50-59”, “60-69”, “70-79”, “80-89” and “90-99” assign the numerical value like 10-19=1, 20-29=2, 30-39=3, 40-49=4, 50-59=5, 60-69=6, 70-79=7, 80-89=8, 90-99=9. All breast cancer data set attribute values are converted in to numerical values for discovering rule classification.

4.2.7.2 Classification rule

The calcification rules are formed Classification rule consists of an antecedent (a set of attribute-values) and a consequent (class). For the purposes of this chapter, a term is defined by a triple <attribute, operator, value>, where value is a value belonging to the domain of attribute. The operator used in this chapter is “=” in the case of categorical/nominal attributes, or “≤” and “>” in the case of continuous attributes. Knowledge representation in the form of rules has the advantage of being intuitively comprehensible to the user. [59][164].
Pseudo Code for Hybrid Meta Heuristic Algorithm

\[
RS = [ \] /* initially, Rule Set is empty */
\]

FOR EACH class \( C \)

\( TS = \{ \text{All training examples belonging to any class} \} \)

WHILE (Number of uncovered training examples belonging to class \( C \) > MaxUncovExampPerClass)

Run the PSO/ACO/GA algorithm to discover the best nominal rule predicting class \( C \), called \( Rule \)

Run the standard PSO algorithm to add continuous terms to \( Rule \), and return the best discovered rule \( BestRule \)

Prune \( BestRule \)

\[
RS = RS \cup \{ BestRule \}
\]

\( TS = TS – \{ \text{training examples correctly covered by discovered rule} \} \)

END WHILE

END FOR

Order rules in \( RS \) by descending Quality

Prune \( RS \) removing unnecessary terms or rules

The sequential covering approach starts by initialising the rule set \( (RS) \) with the empty set. Then, for each class the algorithm performs a WHILE loop, where \( TS \) is used to store the set of training examples the rules will be created from. Each iteration of this loop performs
one run of the hybrid enhanced ACO and enhanced PSO and GA algorithm which firstly
discovers a rule returning the best discovered rule (Rule) predicting examples of the C.

After this rule has been pruned the training examples correctly covered by that rule –
i.e., the examples whose attribute-values satisfy the rule antecedent and have the class
predicted by the rule consequent – are removed from the training set. This process is repeated
as long as necessary to discover rules covering all training examples of the current class.

The hybrid algorithm aimed at mixing the components from enhanced ACO and
enhanced PSO and GA algorithm to easily solve the problem. The enhanced PSO algorithms
make use of particle moving in an n-dimensional space to search for solution for an n-
variable function optimization problem. A particle decides where to move next considering
its own experience so that enhanced PSO find the global best position in the neighborhood
are maintained by using enhanced ACO , update the global position velocity associated with
each dimension which is an increment to be made in each iteration , to the dimension
associated equation(3). Produce a new solution \( V_{ij} \) in the neighborhood of \( X_{ij} \) for the
employee bee using the following equation

\[
V_{ij} = x_{i,j} + \phi_{ij} (x_{ij} - x_{kj})
\]

Where \( k \) is a solution in the neighborhood of \( i \), \( \phi \) is a random number in the range of (-1, 1)
and evaluate them apply the greedy solution process between process. the operator in genetic
provides a smooth way to find the best rule.
4.2.7.3 Rule Pruning

Immediately after the ant completes the construction of a rule, rule pruning is undertaken to increase the comprehensibility and accuracy of the rule. After the pruning step, the rule may be assigned a different predicted class based on the majority class in the cases covered by the rule antecedent. The rule pruning procedure iteratively removes the term whose removal will cause a maximum increase in the quality of the rule. During the rule discovery, pruning method are applied to irrelevant rules are removed.

4.3 SUMMARY

This chapter describes three traditional meta heuristic algorithms used to discover classification rule in data mining such as PSO, ACO and GA, and the best rule can be identified easily if the PSO, ACO and GA are hybridized.