Discovering the Concise Set of Actionable Patterns

Li-Shiang Tsay[1] and Zbigniew W. Raś[2,3]

[1] North Carolina A&T State Univ., School of Tech., Greensboro, NC 27411
[2] Univ. of North Carolina, Dept. of Comp. Science, Charlotte, NC 28223
[3] Polish-Japanese Inst. of Inf. Tech., 02-008 Warsaw, Poland
ltsay@ncat.edu, ras@uncc.edu

Abstract. It is highly expected that knowledge discovery and data mining (KDD) methods can extract useful and understandable knowledge from large amount of data. Action rule mining presents an approach to automatically construct relevantly useful and understandable strategies by comparing the profiles of two sets of targeted objects – those that are desirable and those that are undesirable. The discovered knowledge provides an insight of how relationships should be managed so that objects of low performance can be improved. Traditionally, it was constructed from one or two classification rules. The quality and quantity of such *Action Rules* depend on adopted classification methods. In this paper, we present *StrategyGenerator*, a new algorithm for constructing a complete set of *Action Rules* which satisfies specified constraints. This algorithm does not require prior extraction of classification rules. Action rules are generated directly from a database.

Keywords: Action Rules, Interestingness Measure, Reclassification Models.

1 Introduction

Knowledge Discovery and Data mining (KDD) is the process which identifies and exploits useful and understandable knowledge buried in large volumes of data. The products of KDD have been proven very effective in many fields, such as business, science, government, etc. While most of the KDD algorithms generate predictions and describe behaviors, a focus on understanding changes in object behaviors normally improves the quality of the decision making process. *Action Rule* mining constructs relatively interesting and useful strategies by comparing the profiles of two groups of targeted objects – those that are desirable and those that are undesirable. It is formed as a term $[(\omega)\wedge(\alpha\rightarrow\beta)]\Rightarrow(\phi\rightarrow\psi)$, where ω is a conjunction of fixed condition features shared by both groups, $(\alpha\rightarrow\beta)$ represents proposed changes in values of flexible features, and $(\phi\rightarrow\psi)$ is a desired effect of the action. The discovered knowledge provides an insight of how relationships should be managed so the undesirable objects can be changed to desirable objects. For example, in society, one would like to find a way to improve his or her salary from a low-income to a high-income. Another example in business area is when an owner would like to improve his or her company's profits by going from a high-cost, low-income business to a low-cost, high-income business.

The goal of this research is twofold: (1) making the discovered patterns actionable by providing specific action plans; (2) facilitate the decision-making process in an efficient and easy way by giving users the information they need. Making a decision implies that there are several choices to be considered in a short time frame. It is important not only to identify as many of these choices as possible but also to avoid a redundancy among them.

Action Rules algorithms exam the data in an objective way and represent the discovered information in a short and clear statement. The discovered rules can be served as choices to help a decision maker to produce better decisions. The rules presented to a decision maker should only consist of simple, understandable, and complete strategies that allow a reasonably easy identification of preferable rules. The support and confidence are used to determine which candidate pattern passes the criteria and becomes a desired *action rule*.

Conventional actionable patterns [6-15], and [3] are built on the basis of previously discovered classification rules, so the quality and quantity of the action rules strictly depend on the adopted classification methods. Because these methods may fail to discover some useful action strategies, there is a strong need to develop an algorithm which can derive a set of actionable patterns directly from a given data set. Paper [4] is probably the first attempt towards formally introducing the problem of mining action rules from the scratch. Authors explicitly formulated it as a search problem in a support-confidence-cost framework and next they presented an Apriori-like [1] algorithm for mining action rules. Their definition of an action rule is an object-oriented one and it allows changes on stable attributes. Changing the value of an attribute, either stable or flexible, is linked with a cost [15]. In order to rule out action rules with undesired changes on stable attributes, authors have assigned very high cost to such changes. However, that way, the cost of action rules discovery is getting unnecessarily increased. Also, they did not take into account the dependencies between attribute values which are naturally linked with the cost of rules used either to accept or reject a rule. In this paper, we investigate properties of action rules and present a new efficient algorithm, *StrategyGenerator*, generating a simple and complete set of action rules without using classification rules. This type of action rules is called *Object-Based Action Rules* (OBAC). Three thresholds, *Right Support*, *Left Support*, and *Confidence* of OBAC, are defined and used to identify which action rules are interesting.

2 Mining Action Rules

An information system is used for representing knowledge. Its definition, presented here, was proposed in [5]. By an information system we mean a pair $S = (U, A)$, where:

- U is a nonempty, finite set of objects,
- A is a nonempty, finite set of attributes, i.e. $a : U \rightarrow V_a$ is a function for any $a \in A$, where V_a is called the domain of a.

Elements of U are called objects. In this section, for the purpose of clarity, objects are interpreted as customers. Attributes are interpreted as features such as, offers made by a bank, characteristic conditions etc. We only consider a special type of information systems called decision tables.

A decision table consists of a set of objects where each object is described by a set of attributes. Attributes are partitioned into conditions and decisions. Additionally, we assume that the set of conditions is partitioned into stable conditions and flexible conditions. In our example, we take "profit ranking" as the decision attribute. Its domain is defined as a set of integers. The decision attribute classifies objects (customers) with respect to the profit gained by a bank. Date of birth is an example of a stable attribute. The interest rate on any customer account is an example of a flexible attribute because the bank can adjust rates. We adopt the following definition of a decision table:

By a decision table we mean any information system $S = (U, A_{St} \cup A_{Fl} \cup \{d\})$, where $d \notin (A_{St} \cup A_{Fl})$ is a distinguished attribute called a decision. The set of attributes A in S is partitioned into stable conditions A_{St} and flexible conditions A_{Fl}.

The number of elements in $d(U) = \{ k: (\exists x \in U)[d(x)=k]\}$ is called the rank of d and it is denoted by $r(d)$. Let us observe that the decision d determines the partition $Part_S(d) = \{X_1, X_2,\ldots, X_{r(d)}\}$ of the universe U, where $X_k = d^{-1}(\{k\})$ for $1 \leq k \leq r(d)$. $Part_S(d)$ is called the classification of objects in S with respect to the decision d.

As we have mentioned before, objects in U are interpreted as bank customers. Additionally, we assume that customers in $d^{-1}(\{k_2\})$ are more profitable than customers in $d^{-1}(\{k_1\})$, for any $k_1 < k_2$. The set $d^{-1}(\{r(d)\})$ contains the most profitable customers. Clearly one of the main goals of any bank is to increase its profit. One way to do that is to shift some customers from a group $d^{-1}(\{k_1\})$ to $d^{-1}(\{k_2\})$, for any $k_1 < k_2$. Action rules can be used for that purpose since they provide hints about what type of special offers can me made by a bank to guarantee that values of targeted flexible attributes will be changed in a way that a desired group of customers should move from a group of a lower profit ranking to a group of a higher profit.

The basic principle of reclassification is a process of learning a function that maps one class of objects into another class by changing values of some conditional attributes describing them. The conditional attributes are divided into stable and flexible. The goal of the learning process is to create a reclassification model, for objects in a decision system, which suggests possible changes that can be made within values of some flexible attributes to reclassify these objects the way user wants. In other words, reclassification is the process of showing what changes in values of some of the flexible attributes for a given class of objects are needed in order to shift them from one decision class into another more desired one.

A decision system S classifies a set of objects so that for each object there exists a class label assigned to it.

By *action rule* in S we mean an expression $r = [[(a_1 = \omega_1) \wedge (a_2 = \omega_2) \wedge \ldots \wedge$

$$(a_q = \omega_q)] \wedge (b_1, \alpha_1 \rightarrow \beta_1) \wedge (b_2, \alpha_2 \rightarrow \beta_2) \wedge \ldots \wedge (b_p, \alpha_p \rightarrow \beta_p)] \Rightarrow [(d, k_1 \rightarrow k_2)],$$

where $\{b_1, b_2, \ldots, b_p\}$ are flexible attributes and $\{a_1, a_2,\ldots, a_q\}$ are stable in S. Additionally, we assume that $\omega_i \in Dom(a_i)$, $i=1,2,\ldots,q$ and $\alpha_i, \beta_i \in Dom(b_i)$, $i=1,2,\ldots,p$. The term $(a_i = \omega_i)$ states that the value of the attribute a_i is equal to ω_i, and $(b_j, \alpha_j \rightarrow \beta_j)$ means that value of the attribute b_j has been changed from α_j to β_j.

We say that object $x \in U$ supports an action rule r in S, if there is an object $y \in U$ such that: $(\forall i \leq p)[\, [b_i(x) = \alpha_i] \wedge [b_i(y) = \beta_i]],\ (\forall i \leq q)\ [a_i(x) = a_i(y) = \omega_i]$, $d(x) = k_1$ and $d(y) = k_2$.

An action rule is meaningful only if it contains at least one flexible attribute. If we apply the left hand side of an action rule to object x, then the rule basically says: the values ω_i of stable attributes a_i (i=1,2,…,q) have to remain unchanged in x and then if we change the value of attribute b_i in x from α_i to β_i, for i=1,2,…,p, then the object x which is in the class k_1 is expected to move to class k_2.

From the point of reclassification, we are not targeting all possible cases on the decisional part of reclassification. Since some states are more preferable than other states, we should basically ask users to specify in what direction they prefer to see the changes. On the conditional part of action rules, we have no information to verify if the rule is applicable. If the domain expert can supply prior knowledge of a given domain then some of the rules cannot be applied. For example, the size of a tumor's growth can not increase when the status of a patient is changing from sick to becoming cured. Therefore, some combinations can be ruled out automatically just by having an expert who is involved in the application domain.

Since action plans are constructed by comparing the profiles of two sets of targeted customers, we can assume that there are two patterns associated with each object-based action rule, a left hand side pattern P_L and a right hand side pattern P_R. There are three objective measures of rule interestingness including *Left Support*, *Right Support*, and *confidence*.

The *Left Support* defines the domain of an action rule which identifies objects in U on which the rule can be applied. The larger its value is, the more interesting the rule will be for a user. The left hand side pattern of action rule

$$r = [[(a_1 = \omega_1) \wedge (a_2 = \omega_2) \wedge \ldots \wedge (a_q = \omega_q)] \wedge (b_1, \alpha_1 \rightarrow \beta_1) \wedge$$
$$(b_2, \alpha_2 \rightarrow \beta_2) \wedge \ldots \wedge (b_p, \alpha_p \rightarrow \beta_p)] \Rightarrow [(d, k_1 \rightarrow k_2)]$$

is defined as the set $P_L = V_L \cup \{k_1\}$, where $V_L = \{\omega_1, \omega_2, \ldots, \omega_q, \alpha_1, \alpha_2, \ldots, \alpha_p\}$. The domain $Dom_S(V_L)$ of the left pattern P_L is a set of objects in S that exactly match V_L. Card[$Dom_S(V_L)$] is the number of objects in that domain. Card[$Dom_S(P_L)$] is the number of objects in S that exactly match P_L and Card[U] is the total number of objects in the decision system S. By the left support *supL* of an action rule r, we mean $supL(r)$ = Card[$Dom_S(P_L)$] /Card[U].

The *Right Support* shows how well is the rule supported by objects in S from the preferable decision class. The higher its value is, the stronger case of the reclassification effect will be. The pattern P_R of an action rule r is defined as $P_R = V_R \cup \{k_2\}$, where $V_R = \{\omega_1, \omega_2, \ldots, \omega_q, \beta_1, \beta_2, \ldots, \beta_p\}$.

By domain $Dom_S(V_R)$ we mean a set of objects matching V_R. Card[$Dom_S(P_R)$] is the number of objects that exactly match P_R. By the right support *supR* of action rule r, we mean $supR(r)$ = Card[$Dom_S(P_R)$] /Card[U].

The *confidence* of rule r shows the success measure in transforming objects from a lower preference decision class to a higher one. The *support* of action rule r in S, denoted by Sup_S(r), is the same as the left support $supL(r)$ of action rule r. This is the percentage of objects that need to be reclassified into more preferable class. By the *confidence* of the action rule r in S, denoted by $Conf_S(r)$, we mean

$$Conf_S(r)=(\mathrm{Card}[Dom_S(P_L)]/\mathrm{Card}[Dom_S(V_L)])*(\mathrm{Card}[Dom_S(P_R)]\mathrm{Card}[Dom_S(V_R)]).$$

3 Algorithm and Example: *StrategyGenerator*

The Brute Force method used in [10] to construct all action rules, directly from the decision table, is expensive and inefficient because it considers all possible pair combinations of flexible attributes. Hence, we propose the *StrategyGenerator* algorithm to find the set of most concise action rules. It considers each change of value within a single flexible attribute and each value of a stable attribute as an atomic expression from which more complex expressions are built. The algorithm operates similarly to LERS [2] and the same it guarantees that all discovered action rules are the shortest. This is an agglomerative type of a strategy used for instance in [9] to construct action rules. However, the new method does not require prior extraction of classification rules.

There are two basic steps in the proposed approach. (1) Partition the decision table and select target sub-tables: The original decision table S is first partitioned into a number of sub-tables S_1, S_2, …, S_p according to the decision attribute in the decision table. Two relevant sub-tables are selected based on the reclassification goal for forming workable strategies. (2) Form actionable plans: The workable strategies are formed by comparing the domains of these two chosen sub-tables. In this case, we can avoid generating unqualified candidate terms similarly to *LERS* algorithm [2]. First, single-element candidate terms are computed and checked for its relations with the reclassification goal. If the relation holds, a positive mark is placed on it and the rule is generated. By doing this, we guarantee that the discovered action rules are the most concise ones. The anti-monotonic property is applied to filter candidate terms. When one of the support values is below the threshold, a negative mark is placed on the candidate term. The algorithm recursively takes unmarked candidate terms and extends them by one new unmarked atomic term till no new candidates are found.

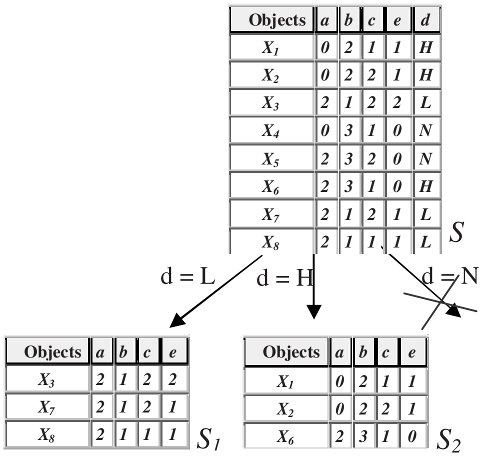
Now, we present a decision table used to illustrate the *StrategyGenerator* for construction of action rules step by step.

Assume that $S = (\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}, \{a\}\cup\{b, \mathrm{c}, \mathrm{e}\}\cup\{d\})$ is a decision table represented by Table 1. Attributes in $\{b, c, e\}$ are flexible, attribute a is stable, and d is the decision attribute. We assume that H denotes customers of a high profit ranking and L denotes customers of a low profit ranking. The direction of reclassification is from L to H. The minimum support for both *supR* and *supL* is 12.5%, and the minimum confidence for rules is 75%.

Partition the decision table. In this example, the domain of the decision attribute is *L, N,* and *H* and the reclassification direction is from L to H. That means the customers with decision value N are not the focus point in this case. Therefore, the decision table S can be divided into S_1 *and* S_2 according to the decision value L and H as represented in Figure 1. Actionable strategies will be constructed based on sub-tables S_1 and S_2 only.

Table 1. Decision table *S*

Objects	*a*	*b*	*c*	*e*	*d*
X_1	*0*	*2*	*1*	*1*	*H*
X_2	*0*	*2*	*2*	*1*	*H*
X_3	*2*	*1*	*2*	*2*	*L*
X_4	*0*	*3*	*1*	*0*	*N*
X_5	*2*	*3*	*2*	*0*	*N*
X_6	*2*	*3*	*1*	*0*	*H*
X_7	*2*	*1*	*2*	*1*	*L*
X_8	*2*	*1*	*1*	*1*	*L*



S

Objects	*a*	*b*	*c*	*e*	*d*
X_1	*0*	*2*	*1*	*1*	*H*
X_2	*0*	*2*	*2*	*1*	*H*
X_3	*2*	*1*	*2*	*2*	*L*
X_4	*0*	*3*	*1*	*0*	*N*
X_5	*2*	*3*	*2*	*0*	*N*
X_6	*2*	*3*	*1*	*0*	*H*
X_7	*2*	*1*	*2*	*1*	*L*
X_8	*2*	*1*	*1*	*1*	*L*

S_1

Objects	*a*	*b*	*c*	*e*
X_3	*2*	*1*	*2*	*2*
X_7	*2*	*1*	*2*	*1*
X_8	*2*	*1*	*1*	*1*

S_2

Objects	*a*	*b*	*c*	*e*
X_1	*0*	*2*	*1*	*1*
X_2	*0*	*2*	*2*	*1*
X_6	*2*	*3*	*1*	*0*

Fig. 1. Partition objects

Forming actionable strategies. The main idea of the reclassification goal is to move objects from an undesirable group into a more desirable one. Objects in *S* having property L are denoted by L_S* and objects in *S* having property R are denoted by R_S*. These two sets are also called granules. *StrategyGenerator* algorithm starts with atomic terms for *S* generated in its first loop. These terms are classified into two groups: premise-type and decision-type. Premise-type atomic terms are split into stable and flexible. As we mentioned before, an action rule without at least one flexible premise-type atomic term is meaningless. Stable atomic terms can not be solely used to construct action rules but they are important in boosting their confidence [10], [12]. In this example, one valid candidate term which is a stable atom (*a*, *2*) is generated. In order to create atomic terms for a flexible attribute we check its domain in both sub-tables. Referring back to Example 1, the values of attribute *b* are "*1*" in sub-table S_1 and "*2*" and "*3*" in sub-table S_2. It means that the action recommendations for attribute *b* say that its value should be changed from *1* to *2* or from *1* to *3*. The corresponding atomic terms are presented as (*b*, $1 \rightarrow 2$) and (*b*, $1 \rightarrow 3$). Following the same procedure for attributes *c* and *e*, their corresponding atomic terms can be formed and they are listed below.

One-element term loop:

```
   // Granules corresponding to values of a decision attribute
   Decision-type atomic term: (d, L→H),
   Granules:  L* = {x3, x7, x8},  R*={x1, x2, x6}
   // Granules corresponding to values of condition attributes
   Premise-type stable atomic expressions:
   (a,0),   L*= ∅   Marked   "-"
   (a,2),   L* = {x3,x7,x8},   R* = {x6}
```

Premise-type flexible atomic expressions:

(b, $1\rightarrow2$) , $L^* = \{x_3,x_7,x_8\}$, supL(r) = 3/8; $R^* = \{x_1,x_2\}$, supR(r) = 2/8; Conf(r) = (3/3) ×(2/2) = 100% Marked "+"
(b, $1\rightarrow3$), $L^* = \{x_3,x_7,x_8\}$, supL(r) = 3/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (3/3) ×(1/3) = 33%
(c, $2\rightarrow1$) , $L^* = \{x_3,x_7\}$, supL(r) = 2/8; $R^* = \{x_1,x_6\}$, supR(r) = 2/8; Conf(r) = (2/4)×(2/4) = 25%
(c, $1\rightarrow2$) , $L^* = \{x_8\}$, supL(r) = 1/8; $R^* = \{x_2\}$, supR(r) = 1/8; Conf(r) = (1/4)×(1/4) = 6.25%
(e, $2\rightarrow1$), $L^* = \{x_3\}$, supL(r) = 1/8; $R^* = \{x_1,x_2\}$, supR(r) = 2/8; Conf(r) = (1/1)×(2/4) = 50%
(e, $2\rightarrow0$), $L^* = \{x_3\}$, supL(r) = 1/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (1/1)×(1/3) = 33.3%
(e, $1\rightarrow0$), $L^* = \{x_7,x_8\}$, supL(r) = 1/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (2/4)×(1/3) = 16.7%

The action rule *r* linking each premise-type term and the decision-type term is acceptable when the values of the corresponding *supL(r)*, *supR(r)*, and *Conf(r)* meet the user specified thresholds. The primary idea of the StrategyGenerator algorithm lies in the property of anti-monotonic property of the support. It is used to prune unqualified candidates. This is achieved by placing a "-" mark when a term does not have sufficient support. Going back to the example, the support of the atomic term (*a*,*0)* does not satisfy the minimum support requirement, so it is marked with "-" symbol and it is not considered in later steps of the algorithm. The goal of this algorithm is to find the shortest action rules. It means when a premise-type term t_1 jointly with a decision-type term form an acceptable action rule, then t_1 is not investigated any further. In this example, the term (*b, 1→2*) jointly with (*d, L→H*) meet all three thresholds, so the action rule (*b, 1→2*) ⇒ (*d, L→H*) is discovered and the term (*b, 1→2*) is marked as "+".

Build two-element premise-type terms by concatenating any two unmarked premise-type terms that have different attributes. Below is the list of two-element terms. There is no action rule generated in this step, since none of the terms jointly with *(d, L→H)* satisfy all three thresholds.

Two-elements term loop:

(a,2) ∧ (b, $1\rightarrow3$), $L^* = \{x_3,x_7,x_8\}$, supL(r) = 3/8; $R^* =\{x_6\}$, supR(r) = 1/8; Conf(r) = (3/3)×(1/2) = 50%
(a,2) ∧ (c, $2\rightarrow1$), $L^* = \{x_3,x_7\}$, supL(r) = 2/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (2/3)×(1/2) = 33.3%
(a,2) ∧ (c, $1\rightarrow2$), $L^* = \{x_8\}$, supL(r) = 1/8; $R^* = \emptyset$; Marked "-"
(a,2) ∧ (e, $2\rightarrow1$), $L^* = \{x_3\}$, supL(r) = 1/8; $R^* = \emptyset$; Marked "-"
(a,2) ∧ (e, $2\rightarrow0$), $L^* = \{x_3\}$, supL(r) = 1/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (1/1)×(1/2) = 50%
(a,2) ∧ (e, $1\rightarrow0$), $L^* = \{x_7,x_8\}$, supL(r) = 2/8; $R^* = \{x_6\}$, supR(r) = 1/8;Conf(r) = (2/2)×(1/2) = 50%
(b, $1\rightarrow3$) ∧ (c, $2\rightarrow1$), $L^* = \{x_3,x_7\}$, supL(r) = 2/8; $R^* = \{x_6\}$, supR(r) = 1/8; Conf(r) = (2/2)×(1/2) = 50%
(b, $1\rightarrow3$) ∧ (c, $1\rightarrow2$), $L^* = \{x_8\}$, supL(r) = 1/8; $R^* = \emptyset$ Marked "-"
(b, $1\rightarrow3$) ∧ (e, $2\rightarrow1$), $L^* = \{x_8\}$, supL(r) = 1/8; $R^* = \emptyset$ Marked "-"

(b, 1→3) ∧ (e, 2→0), L* = {x_3}, supL(r) = 1/8; R* = {x_6},supR(r) = 1/8; Conf(r) = (1/1)×(1/3) = 33.3%
(b, 1→3) ∧ (e, 1→0), L* = {x_7, x_8}, supL(r) = 2/8; R* = {x_6}, supR(r) = 1/8; Conf(r) = (2/2)×(1/3) = 33.3%
(c, 2→1) ∧ (e, 2→1), L* = {x_3}, supL(r) = 1/8; R* = {x_1}, supR(r) = 1/8; Conf(r) = (1/1)×(1/2) = 50%
(c, 2→1) ∧ (e, 2→0), L* = {x_3}, supL(r) = 1/8; R* = {x_6}, supR(r) = 1/8; Conf(r) = (1/1)×(1/2) = 50%
(c, 2→1) ∧ (e, 1→0), L* = {x_7}, supL(r) = 1/8; R* = {x_6}, supR(r) = 1/8; Conf(r) = (1/2)×(1/2) = 25%
(c, 1→2) ∧ (e, 2→1), L* = ∅; Marked "-"
(c, 1→2) ∧ (e, 2→0), L* = ∅; Marked "-"
(c, 1→2) ∧ (e, 1→0), L* = {x_8}, supL(r) = 1/8; R* = ∅; Marked "-"

Build three-element terms by concatenating any two unmarked terms that have different attributes. Below is the list of three-element terms. There are three action rules discovered.

Three-elements term loop:

(a,2)∧(b, 1→3)∧(c, 2→1), L* = {x_3,x_7}, supL(r)=2/8; R*= {x_6}, supR(r) = 1/8; Conf(r) = (2/2)×(1/1) = 100%; Marked "+"
(a,2)∧(b, 1→3)∧(e, 2→0), L* = {x_3}, supL(r) = 1/8; R*={x_6}, supR(r) = 1/8; Conf(r) = (1/1)×(1/2) = 50%
(a,2)∧(b, 1→3)∧(e, 1→0), L* = {x_7, x_8}, supL(r)=2/8; R*= {x_6}, supR(r) = 1/8; Conf(r)=(2/2)×(1/2)=50%
(a,2)∧(c, 2→1)∧(e, 2→0), L* = {x_3}, supL(r)=1/8; R*= {x_6}, supR(r) = 1/8; Conf(r)=(1/1)×(1/1)=100%; Marked "+"
(a,2)∧(c, 2→1)∧(e, 1→0), L* = {x_7}, supL(r)= 1/8; R*={x_6}, supR(r)=1/8; Conf(r)= 1/1)×(1/1) = 100%; Marked "+"
(b, 1→3)∧(c, 2→1)∧(e, 2→1), L* = {x_3}, supL(r)=1/8; R*= ∅ Marked "-"
(b, 1→3)∧(c, 2→1)∧(e, 2→0), L* = {x_3}, supL(r)=1/8; R*={x_6}, supR(r)=1/8; Conf(r)=(1/1)×(1/2) = 50%
(b, 1→3)∧(c, 2→1)∧(e, 1→0), L* = {x_7}, supL(r)=1/8; R*={x_6}, supR(r)=1/8; Conf(r)=(1/1)×(1/2)=50%

In Example 1, we have the following four action rules:

(b, 1→2)⇒(d, L→H), supL(r)=3/8, supR(r)=2/8, Conf(r)=100%
((a,2)∧(b, 1→3)∧(c, 2→1))⇒(d, L→H), supL(r)=2/8, supR(r)=1/8, Conf(r)=100%
((a,2)∧(c, 2→1)∧(e, 2→0))⇒(d, L→H), supL(r)=1/8, supR(r)=1/8, Conf(r)=100%
((a,2) ∧(c, 2→1)∧(e, 1→0))⇒(d, L→H), supL(r)=1/8, supR(r)=1/8, Conf(r)=100%

We claim that the new method guarantees that the actionable patterns are concise, general, and reliable. As we can see the discovered action rules contain relatively few attribute-value pairs on the classification side and the number of these rules is also relatively small. Such rules are more readable, easier to understand and apply later on.

The algorithm, StrategyGenerator, was implemented under Windows XP. It was tested on several public domain databases and on the medical database HEPAR prepared in the Medical Center of Postgraduate Education (Warsaw, Poland) by Dr. med. Hanna Wasyluk. In all cases the recall of the new algorithm was higher than DEAR [11][12].

Finally, let us compare the action rules generated by StrategyGenerator with action rules constructed by the tree-based algorithms DEAR [7], [14], [10], [11], [6], [12], [13], [9]. For the same Example 1, thirteen classification rules have been generated by LERS algorithm and they are listed below:

$(b,2)\rightarrow(d,H)$	$(b,1)\rightarrow(d,L)$	$(e,2)\rightarrow(d,L)$
$(a,0)\wedge(b,3)\rightarrow(d,N)$	$(a,0)\wedge(c,2)\rightarrow(d,H)$	$(a,0)\wedge(e,1)\rightarrow(d,H)$
$(a,0)\wedge(e,0)\rightarrow(d,N)$	$(a,2)\wedge(e,1)\rightarrow(d,L)$	$(b,3)\wedge(c,2)\rightarrow(d,N)$
$(a,2)\wedge(b,3)\wedge(c,1)\rightarrow(d,H)$	$(a,2)\wedge(b,3)\wedge(c,2)\rightarrow(d,N)$	
$(a,2)\wedge(c,1)\wedge(e,0)\rightarrow(d,H)$	$(a,2)\wedge(c,2)\wedge(e,0)\rightarrow(d,N)$.	

Five classification rules have been generated by C4.5 algorithm and they are listed below:

$(b,2)\rightarrow(d,H)$	$(b,1)\rightarrow(d,L)$	$(a,0)\wedge(b,3)\rightarrow(d,N)$
$(a,2)\wedge(b,3)\wedge(c,1)\rightarrow(d,H)$	$(a,2)\wedge(b,3)\wedge(c,2)\rightarrow(d,N)$.	

DEAR algorithms generated from them only one action rule: $(b,1\rightarrow2)\Rightarrow(d,L\rightarrow H)$. The new method generates more action rules than DEAR as we have seen in the above example.

4 Conclusion

The ability to discover useful knowledge hidden in large volumes of data and to act on that knowledge is becoming increasingly important in today's competitive world. The knowledge extracted from data can provide a competitive advantage in support of decision-making. In this paper, we focus on analyzing a complete information system and obtaining a set of concise workable strategies. Any action rule provides a brief and clear hint to a user about required changes within flexible attributes that are needed to re-classify some objects from a lower ranked class to a higher one. This knowledge can be turned into action and this action may help to achieve user's goal. StrategyGenerator is a novel method of a reclassification strategy which extracts higher level actionable knowledge from large volumes of data.

Acknowledgements

This work was supported by the National Science Foundation under grant IIS-0414815.

References

1. Agrawal, R., Srikant, R.: Fast algorithm for mining association rules. In: Proceeding of the Twentieth International Conference on VLDB, pp. 487–499 (1994)
2. Chmielewski, M.R., Grzymala-Busse, J.W., Peterson, N.W., Than, S.: The rule induction system LERS - a version for personal computers. Foundations of Computing and Decision Sciences 18(3-4), 181–212 (1993)
3. Greco, S., Matarazzo, B., Pappalardo, N., Slowinski, R.: Measuring expected effects of interventions based on decision rules. Journal of Experimental and Theoretical Artificial Intelligence 17(1-2), 103–118 (2005)
4. He, Z., Xu, X., Deng, S., Ma, R.: Mining action rules from scratch. Expert Systems with Applications 29(3), 691–699 (2005)
5. Pawlak, Z.: Information systems - theoretical foundations. Information Systems Journal 6, 205–218 (1981)
6. Raś, Z.W., Tzacheva, A., Tsay, L.-S., Gurdal, O.: Mining for interesting action rules. In: Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2005), Compiegne University of Technology, France, pp. 187–193 (2005)
7. Raś, Z.W., Tsay, L.-S.: Discovering extended action-rules (System DEAR), in Intelligent Information Systems. In: Proceedings of the IIS 2003 Symposium, Advances in Soft Computing, pp. 293–300. Springer, Heidelberg (2003)
8. Raś, Z., Wieczorkowska, A.: Action rules: how to increase profit of a company. In: Zighed, A.D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 587–592. Springer, Heidelberg (2000)
9. Raś, Z.W., Wyrzykowska, E.: ARAS: Action rules discovery based on agglomerative strategy. In: Mining Complex Data, Post-Proceedings of the ECML/PKDD 2007 Third International Workshop, MCD 2007. LNCS (LNAI), vol. 4944, pp. 196–208. Springer, Heidelberg (2008)
10. Tsay, L.-S.: Discovery of extended action rules, Ph.D. Dissertation, Department of Computer Science, University of North Carolina, Charlotte (2005)
11. Tsay, L.-S., Raś, Z.W.: Action rules discovery: system DEAR2, method and experiments. Journal of Experimental and Theoretical Artificial Intelligence 17(1-2), 119–128 (2005)
12. Tsay, L.-S., Raś, Z.W.: Action rules discovery systems DEAR3. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 483–492. Springer, Heidelberg (2006)
13. Tsay, L.-S., Raś, Z.W.: E-Action Rules. In: Foundations of Data Mining, Studies in Computational Intelligence, Springer, Heidelberg (will appear, 2007)
14. Tsay, L.-S., Raś, Z., Wieczorkowska, A.: Tree-based algorithm for discovering extended action-rules (System DEAR2). In: Intelligent Information Processing and Web Mining, Advances in Soft Computing, Proceedings of the IIS 2004 Symposium, pp. 459–464. Springer, Heidelberg (2004)
15. Tzacheva, A., Raś, Z.W.: Action rules mining. International Journal of Intelligent Systems 20(7), 719–736 (2005)