Tree-based Algorithms for Action Rules Discovery

Zbigniew W. Raś[1,2], Li-Shiang Tsay[3], and Agnieszka Dardzińska[4]

[1] Univ. of North Carolina, Charlotte, Dept. of Computer Science, 9201 Univ. City Blvd., Charlotte, NC 28223, USA
[2] Polish-Japanese Institute of Information Technology, Koszykowa 86, 02-008 Warsaw, Poland
[3] North Carolina A&T State Univ., School of Technology, Greensboro, NC 27411, USA
[4] Bialystok Technical Univ., Dept. of Computer Science, 15-351 Bialystok, Poland

Abstract. One of the main goals in Knowledge Discovery is to find interesting associations between values of attributes, those that are meaningful in a domain of interest. The most effective way to reduce the amount of discovered patterns is to apply two interestingness measures, subjective and objective. Subjective measures are based on the subjectivity and understandability of users examining the patterns. They are divided into actionable, unexpected, and novel. Because classical knowledge discovery algorithms are unable to determine if a rule is truly actionable for a given user [1], we focus on a new class of rules [15], called E-action rules, that can be used not only for automatic analysis of discovered classification rules but also for hints of how to reclassify some objects in a data set from one state into another more desired one. Actionability is closely linked with the availability of flexible attributes [18] used to describe data and with the feasibility and cost [23] of desired re-classifications. Some of them are easy to achieve. Some, initially seen as impossible within constraints set up by a user, still can be successfully achieved if additional attributes are available. For instance, if a system is distributed and collaborating sites agree on the ontology [5], [6] of their common attributes, the availability of additional data from remote sites can help to achieve certain re-classifications of objects at a server site [23]. Action tree algorithm, presented in this paper, requires prior extraction of classification rules similarly as the algorithms proposed in [15] and [17] but it guarantees a faster and more effective process of E-action rules discovery. It was implemented as system $DEAR\_2.2$ and tested on several public domain databases. Support and confidence of E-action rules is introduced and used to prune a large number of generated candidates which are irrelevant, spurious, and insignificant.

1 Introduction

Finding useful rules is an important task of knowledge discovery in data. Most of the researchers focused on techniques for generating patterns, such as classification rules, association rules...etc, from a data set. They assume that it is users responsibility to analyze these patterns and infer actionable

solutions for specific problems within a given domain. The classical knowledge discovery algorithms have the potential to identify enormous number of significant patterns from data. Therefore, people are overwhelmed by a large number of uninteresting patterns which are very difficult to analyze and use to form timely solutions. So, there is a need to look for new techniques and tools with the ability to assist people in identifying rules with useful knowledge.

There are two types of interestingness measure: objective and subjective (see [10], [1], [19], [20]). Subjective interestingness measures include unexpectedness [19] and actionability [1]. When a rule contradicts the user's prior belief about the domain, uncovers new knowledge, or surprises him, it is classified as unexpected. A rule is deemed actionable, if the user can take action to gain an advantage based on this rule. Domain experts basically look at a rule and say that this rule can be converted into an appropriate action.

E-action rules mining is a technique that intelligently and automatically assists humans in acquiring useful information from data. This information can be turned into actions which can benefit users. The approach gives suggestions about how to change certain attribute values of a given set of objects in order to reclassify them according to a user wish.

There are two frameworks for mining actionable knowledge: loosely coupled and tightly coupled [9]. In the tightly coupled framework, action rules are extracted directly from a database [7], [8], [22]. In the loosely coupled framework, proposed in [15], the extraction of actionable knowledge is preceded by classification rules discovery. It is further subdivided into:

- strategies generating action rules from certain pairs of classification rules [18], [21], [23],
- strategies generating action rules from single classification rules [16], [24].

This paper relates to a loosely coupled framework. In most of the algorithms for action rules mining, there is no guarantee that the discovered patterns in the first step will lead to actionable knowledge that is capable of maximizing profits. One way to approach this problem is to assign a cost function to all changes of attribute values [24]. If changes of attribute values in the classification part of an action rule are too costly, then they can be replaced by composing this rule with other action rules, as proposed in [23]. Each composition of these rules uniquely defines a new action rule. Objects supporting each new action rule, let's say r, are the same as objects supporting the action rule replaced by r but the cost of reclassifying them is lower for the new rule.

E-action rule models the actionability concept in a better way than action rule [15] by introducing a notion of its supporting class of objects. E-action rules are constructed from certain pairs of classification rules. They can be used not only for evaluating discovered patterns but also for reclassifying

some objects in a dataset from one state into a new more desired state. For example, classification rules found from a bank's data can be very useful to describe who is a good client (whom to offer some additional services) and who is a bad client (whom to watch carefully to minimize the bank loses). However, if bank managers need to improve their understanding of customers and seek for specific actions to improve the services, mere classification rules are not sufficient. In this paper, we propose to use classification rules to build a new strategy of action based on their condition features in order to get a desired effect on their decision feature. Going back to the bank example, the strategy of action would consist of modifying some condition features in order to improve our understanding of customers behavior and then improve the services. E-action rules are useful in many other fields, including medical diagnosis. In medical diagnosis, classification rules can explain the relationships between symptoms and sickness and in predicting the diagnosis of a new patient. E-action rules are useful in providing a hint to a doctor what symptoms have to be modified in order to recover a certain group of patients from a given illness.

Action Tree algorithm is presented for generating E-action rules and it is implemented as System $DEAR\_2.2$. The algorithm follows a top-down strategy that searches for a solution in a part of the search space. It is seeking at each stage for a stable attribute that has a least number of values. Then, the set of rules is split recursively using that attribute. When all stable attributes are processed, the final subsets are split further based on a decision attribute. This strategy generates an action tree which is used to construct E-action rules from the leaf nodes of the same parent.

2 Information System and E-Action Rules

An information system is used for representing knowledge. Its definition, presented here, is due to Pawlak [12].

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. $a : U \longrightarrow V_a$ 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 paper, 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 consider a special case of information systems called decision tables [12]. In any decision table together with the set of attributes a partition of that set into conditions and decisions is given. Additionally, we assume that the set of conditions is partitioned into stable conditions and flexible

conditions. For simplicity reason, we assume that there is only one decision attribute. *Date of birth* is an example of a stable attribute. The *interest rate* on any customer account is an example of a flexible attribute as 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 the decision. The elements of A_{St} are called stable conditions, whereas the elements of A_{Fl} are called flexible conditions.

As an example of a decision table we take $S = (\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}\}, \{a, c\} \cup \{b\} \cup \{d\})$ represented by Table 1. The set $\{a, c\}$ lists stable attributes, b is a flexible attribute and d is a decision attribute. Also, we assume that H denotes a *high* profit and L denotes a *low* one.

Table 1. Decision System

	a	b	c	d
x_1	2	1	2	L
x_2	2	1	2	L
x_3	1	1	0	H
x_4	1	1	0	H
x_5	2	3	2	H
x_6	2	3	2	H
x_7	2	1	1	L
x_8	2	1	1	L
x_9	2	2	1	L
x_{10}	2	3	0	L
x_{11}	1	1	2	H
x_{12}	1	1	1	H

In order to induce rules in which the THEN part consists of the decision attribute d and the IF part consists of attributes belonging to $A_{St} \cup A_{Fl}$, for instance system *LERS* [4] can be used for rules extraction.

Alternatively, we can extract rules from sub-tables $(U, B \cup \{d\})$ of S, where B is a d-reduct (see [11]) of S, to improve efficiency of the algorithm when the number of attributes is large. The set B is called d-reduct in S if there is no proper subset C of B such that d depends on C. The concept of d-reduct in S was introduced to induce efficiently rules from S describing values of the attribute d depending on minimal subsets of $A_{St} \cup A_{Fl}$.

By $L(r)$ we mean all attributes listed in the IF part of a rule r. For example, if $r_1 = [(a_1, 2) \wedge (a_2, 1) \wedge (a_3, 4) \longrightarrow (d, 8)]$ is a rule then $L(r_1) = \{a_1, a_2, a_3\}$.

By $d(r_1)$ we denote the decision value of that rule. In our example $d(r_1) = 8$. If r_1, r_2 are rules and $B \subseteq A_{St} \cup A_{Fl}$ is a set of attributes, then $r_1/B = r_2/B$ means that the conditional parts of rules r_1, r_2 restricted to attributes B are the same. For example if $r_2 = [(a_2, 1) * (a_3, 4) \longrightarrow (d, 1)]$, then $r_1/\{a_2, a_3\} = r_2/\{a_2, a_3\}$.

In our example, we get the following certain rules with support greater or equal to 2:

$(b, 3) * (c, 2) \longrightarrow (d, H)$, $(a, 1) * (b, 1) \longrightarrow (d, L)$,
$(a, 1) * (c, 1) \longrightarrow (d, L)$, $(b, 1) * (c, 0) \longrightarrow (d, H)$,
$(a, 1) \longrightarrow (d, H)$

Now, let us assume that $(a, v \longrightarrow w)$ denotes the fact that the value of attribute a has been changed from v to w. Similarly, the term $(a, v \longrightarrow w)(x)$ means that $a(x) = v$ has been changed to $a(x) = w$. Saying another words, the property (a, v) of object x has been changed to property (a, w).

Let $S = (U, A_{St} \cup A_{Fl} \cup \{d\})$ be a decision table and rules r_1, r_2 are extracted from S. The notion of an extended action rule (E-action rule) was given in [21]. Its definition is given below. We assume here that:

- B_{St} is a maximal subset of A_{St} such that $r_1/B_{St} = r_2/B_{St}$,
- $d(r_1) = k_1$, $d(r_2) = k_2$ and $k_1 \leq k_2$,
- $(\forall a \in [A_{St} \cap L(r_1) \cap L(r_2)])[a(r_1) = a(r_2)]$,
- $(\forall i \leq q)(\forall e_i \in [A_{St} \cap [L(r_2) - L(r_1)]])[e_i(r_2) = u_i]$,
- $(\forall i \leq r)(\forall c_i \in [A_{Fl} \cap [L(r_2) - L(r_1)]])[c_i(r_2) = t_i]$,
- $(\forall i \in p)(\forall b_i \in [A_{Fl} \cap L(r_1) \cap L(r_2)])[[b_i(r_1) = v_i] \& [b_i(r_2) = w_i]]$.

Let $A_{St} \cap L(r_1) \cap L(r_2) = B$. By (r_1, r_2) -E-action rule on $x \in U$ we mean the expression r:

$[\prod\{a = a(r_1) : a \in B\} \wedge (e_1 = u_1) \wedge (e_2 = u_2) \wedge ... \wedge (e_q = u_q) \wedge (b_1, v_1 \longrightarrow w_1) \wedge (b_2, v_2 \longrightarrow w_2) \wedge ... \wedge (b_p, v_p \longrightarrow w_p) \wedge (c_1, \longrightarrow t_1) \wedge (c_2, \longrightarrow t_2) \wedge ... \wedge (c_r, \longrightarrow t_r)](x) \Longrightarrow [(d, k_1 \longrightarrow k_2)](x)$

Object $x \in U$ supports (r_1, r_2)-E-action rule r in $S = (U, A_{St} \cup A_{Fl} \cup \{d\})$, if the following conditions are satisfied:

- $(\forall i \leq p)[b_i \in L(r)][b_i(x) = v_i] \wedge d(x) = k_1$
- $(\forall i \leq p)[b_i \in L(r)][b_i(y) = w_i] \wedge d(y) = k_2$
- $(\forall j \leq p)[a_j \in (A_{St} \cap L(r_2))][a_j(x) = u_j]$
- $(\forall j \leq p)[a_j \in (A_{St} \cap L(r_2))][a_j(y) = u_j]$
- object x supports rule r_1
- object y supports rule r_2

By the support of E-action rule r in S, denoted by $Sup_S(r)$, we mean the set of all objects in S supporting r. Saying another words, this set is defined as:

$\{x : [a_1(x) = u_1] \wedge [a_2(x) = u_2] \wedge ... \wedge [a_q(x) = u_q] \wedge [b_1(x) = v_1] \wedge$
$[b_2(x) = v_2] \wedge ... \wedge [b_p(x) = v_p] \wedge [d(x) = k_1]\}$.

By the confidence of r in S, denoted by $Conf_S(r)$, we mean

$[Sup_S(r)/Sup_S(L(r))] \times [Conf(r_2)]$

To find the confidence of (r_1, r_2)-E-action rule in S, we divide the number of objects supporting (r_1, r_2)-action rule in S by the number of objects supporting left hand side of (r_1, r_2)-E-action rule times the confidence of the classification rule r_2 in S.

3 Discovering E-Action Rules

In this section we present a new algorithm, called Action-Tree algorithm, for discovering E-action rules. Basically, we partition the set of classification rules R discovered from a decision system $S = (U, A_{St} \cup A_{Fl} \cup \{d\})$, where A_{St} is the set of stable attributes, A_{Fl} is the set of flexible attributes and, $V_d = \{d_1, d_2, ..., d_k\}$ is the set of decision values, into subsets of rules having the same values of stable attributes in their classification parts and defining the same value of the decision attribute. Classification rules can be extracted from S using, for instance, discovery system $LERS$ [2].

Action-tree algorithm for extracting E-Action rules from decision system S is as follows:

i. Build Action-Tree
- **a.** Partition the set of classification rules R in a way that two rules are in the same class if their stable attributes are the same
  1. Find the cardinality of the domain V_{v_i} for each stable attribute v_i in S.
  2. Take v_i, which $card(V_{v_i})$ is the smallest, as the splitting attribute and divide R into subsets each of which contains rules having the same value of the stable attribute v_i.
  3. For each subset, obtained in step 2, determine if it contains rules of different decision values and different values of flexible attributes. If it does, go to step 2. If it doesn't, there is no need to split the subset further and we place a mark.
- **b.** Partition each resulting subset into new subsets each of which contains only rules having the same decision value.
- **c.** Each leaf of the resulting tree represents a set of rules which do not contradict on stable attributes and also it uniquely defines decision value d_i. The path from the root to that leaf gives the description of objects supported by these rules.

ii. Generate E-action rules
- **a.** Form E-action rules by comparing all unmarked leaf nodes of the same parent.

b. Calculate support and confidence of each new-formed E-action rule. If support and confidence meet the thresholds set up by user, print the rule.

The algorithm starts at the root node of the tree, called E-action tree, representing all classification rules extracted from S. A stable attribute is selected to partition these rules. For each value of that attribute an outgoing edge from the root node is created, and the corresponding subset of rules that have the attribute value assigned to that edge is moved to the newly created child node. This process is repeated recursively for each child node. When we are done with stable attributes, the last split is based on a decision attribute for each current leaf of E-action tree. If at any time all classification rules representing a node have the same decision value, then we stop constructing that part of the tree. We still have to explain which stable attributes are chosen to split classification rules representing a node of E-action tree. The algorithm selects any stable attribute which has the smallest number of possible values among all the remaining stable attributes. This step is justified by the need to apply a heuristic strategy (see [14]) which will minimize the number of edges in the resulting tree and the same make the time-complexity of the algorithm lower.

We have two types of nodes: a leaf node and a non-leaf node. At a non-leaf node, the set of rules is partitioned along the branches and each child node gets its corresponding subset of rules. Every path to the decision attribute node, one level above the leaf node, represents a subset of the extracted classification rules when the stable attributers have the same value. Each leaf node represents a set of rules, which do not contradict on stable attributes and also define decision value d_i. The path from the root to that leaf gives the description of objects supported by these rules.

Instead of splitting the set of rules R by stable attributes and next by the decision attribute, we can also start the partitioning algorithm from a decision attribute. For instance, if a decision attribute has 3 values, we get 3 initial sub-trees. In the next step of the algorithm, we start splitting these sub-trees by stable attributes following the same strategy as the one presented for E-action trees. This new algorithm is called action-forest algorithm.

Now, let us take Table 1 as an example of a decision system S. Attributes a, c are stable and b, d flexible. Assume now that our plan is to re-classify some objects from the class $d^{-1}(\{d_i\})$ into the class $d^{-1}(\{d_j\})$. In our example, we also assume that $d_i = (d, L)$ and $d_j = (d, H)$.

First, we represent the set R of certain rules extracted from S as a table (see Table 2). The first column of this table shows objects in S supporting the rules from R (each row represents a rule). For instance, the second row represents the rule $[[(a, 2) \wedge (c, 1)] \Rightarrow (d, L)]$. The construction of an action

Table 2. Set of rules R with supporting objects

Objects	a	b	c	d
$\{x_3, x_4, x_{11}, x_{12}\}$	1			H
$\{x_1, x_2, x_7, x_8\}$	2		1	L
$\{x_7, x_8, x_9\}$	2		0	L
$\{x_3, x_4\}$		1	0	H
$\{x_5, x_6\}$		3	2	H

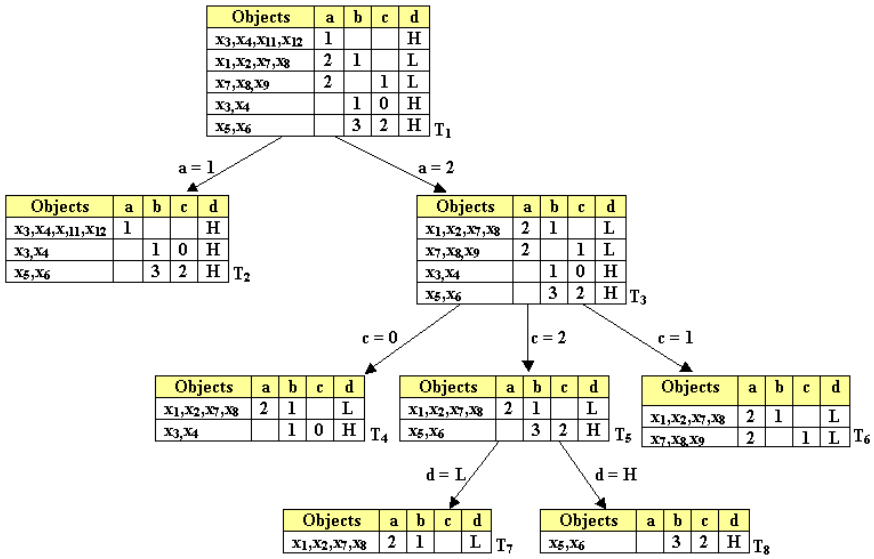

Fig. 1. Action tree

tree starts with the set R as a table (see Table 2) representing the root of the tree (T_1 in Fig. 1). The root node selection is based on a stable attribute with the smallest number of values among all stable attributes. The same strategy is used for a child node selection. After labelling the nodes of the tree by all stable attributes, the tree is split based on the value of the decision attribute. Referring back to the example in Table 1, we use stable attribute a to split that table into two sub-tables defined by values $\{1, 2\}$ of attribute a. The domain of attribute a is $\{1, 2\}$ and the domain of attribute c is $\{0, 1, 2\}$. Clearly, $card[V_a]$ is less than $card[V_c]$ so we partition the table into two: one table with rules containing $a = 0$ and another with rules containing $a = 2$. Corresponding edges are labelled by values of attribute a. All rules in the sub-table T_2 have the same decision value. So, we can not construct E-action rule from that sub-table which means it is not divided any further. Because rules in the sub-table T_3 contain different decision values and a stable attribute

c, T_3 is partitioned into three sub-tables, one with rules containing c=0, one with rules containing c=1, and one with rules containing c=2. Now, rules in each of the sub-tables do not contain any stable attributes. Sub-table T_6 is not split any further for the same reason as sub-table T_2. All objects in sub-table T_4 have the same value of flexible attribute b. There is no way to form a workable strategy from this sub-table so it is not partitioned any further. Sub-table T_5 is divided into two new sub-tables. Each leaf represents a set of rules, which do not contradict on stable attributes and also define decision value d_i.

The path from the root of the tree to that leaf gives the description of objects supported by these rules. Following the path labelled by value $[a = 2]$, $[c = 2]$, and $[d = L]$, we get table T_7. Following the path labelled by value $[a = 2]$, $[c = 2]$, and $[d = H]$, we get table T_8. Because T_7 and T_8 are sibling nodes, we can directly compare pairs of rules belonging to these two tables and construct one E-action rule such as:

$[[(a, 2) \wedge (b, 1 \rightarrow 3)] \Rightarrow (d, L \rightarrow H)]$.

After the rule is formed, we evaluate it by checking its support and its confidence ($sup = 4$, $conf = 100\%$).

This new algorithm (called DEAR_2.2) was implemented and tested on several data sets from UCI Machine Learning Repository. In all cases, the action tree algorithm was more efficient then the action forest algorithm. The generated E-action rules by both algorithms are the same. The confidence of E-action rules is higher than the confidence action rules.

4 Conclusion

E-action rules are structures that represent actionability in an objective way. The strategy used to generate them is data driven and domain independent because it does not depend on domain knowledge. Although the definition of E-action rules is purely objective, we still can not get rid of some degree of subjectivity in determining how actions can be implemented. To build E-action rules, we divide all attributes into two subsets, stable and flexible. Obviously, this partition has to be done by users who decide which attributes are stable and which are flexible. This is a purely subjective decision. A stable attribute has no influence on change, but any flexible attribute may influence changes. Users have to be careful judging which attributes are stable and which are flexible. If we apply E-action rule on objects then it shows how values of their flexible features should be changed in order to achieve their desired re-classification. Stable features always will remain the same. Basically, any E-action rule identifies a class of objects that can be reclassified from an undesired state to a desired state by properly changing some of the values of their flexible features. How to implement these changes often depends on the user. If the attribute is an interest rate on the banking account

then banks can take appropriate action as the rule states (i.e., change lower interest rate to 4.75%). In this case, it is a purely objective action. However, if the attribute is a fever then doctors may lower the temperature by following a number of different actions. So, this is a purely subjective concept. Basically, we cannot eliminate some amount of subjectivity in the process of E-action rules construction and implementation.

5 Acknowledgement

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

References

1. Adomavicius G, Tuzhilin A (1997) Discovery of actionable patterns in databases: the action hierarchy approach. In: Proceedings of KDD97 Conference. Newport Beach, CA. AAAI Press
2. Chmielewski M R, Grzymala-Busse J W, Peterson N W, Than S (1993) The rule induction system LERS - a version for personal computers. In: Foundations of Computing and Decision Sciences. Vol. 18, No. 3-4, Institute of Computing Science, Technical University of Poznan, Poland: 181–212
3. Geffner H, Wainer J (1998) Modeling action, knowledge and control. In: ECAI 98, Proceedings of the 13th European Conference on AI, (Ed. H. Prade). John Wiley & Sons, 532–536
4. Grzymala-Busse J (1997) A new version of the rule induction system LERS. In: Fundamenta Informaticae, Vol. 31, No. 1, 27–39
5. Guarino N (1998) Formal Ontology in Information Systems, IOS Press, Amsterdam
6. Guarino N, Giaretta P (1995) Ontologies and knowledge bases, towards a terminological clarification, in Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, IOS Press, Amsterdam
7. He, Z., Xu, X., Deng, S., Ma, R. (2005) Mining action rules from scratch, Expert Systems with Applications, Vol. 29, No. 3, 691-699
8. Im, S., Ras, Z.W. (2008) Action rule extraction from a decision table: ARED, in Foundations of Intelligent Systems, Proceedings of ISMIS′08, A. An et al. (Eds.), Toronto, Canada, LNAI, Springer, 2008, will appear
9. Kaur, H. (2005) Actionable rules: issues and new directions, in Transactions on Engineering, Computing and Technology, World Informatica Society, April, 61–64
10. Liu B, Hsu W, Chen S (1997) Using general impressions to analyze discovered classification rules. In: Proceedings of KDD97 Conference, Newport Beach, CA, AAAI Press
11. Pawlak Z (1991) Rough sets-theoretical aspects of reasoning about data. Kluwer, Dordrecht
12. Pawlak Z (1981) Information systems - theoretical foundations. In: Information Systems Journal, Vol. 6, 205–218

13. Polkowski L, Skowron A (1998) Rough sets in knowledge discovery. In: Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer
14. Raś Z (1999) Discovering rules in information trees. In: Principles of Data Mining and Knowledge Discovery, (Eds. J. Zytkow, J. Rauch), Proceedings of PKDD'99, Prague, Czech Republic, LNAI, No. 1704, Springer, 518–523
15. Raś Z, Wieczorkowska A (2000) Action rules: how to increase profit of a company. In: Principles of Data Mining and Knowledge Discovery, (Eds. D.A. Zighed, J. Komorowski, J. Zytkow), Proceedings of PKDD'00, Lyon, France, LNCS/LNAI, No. 1910, Springer-Verlag, 587–592
16. Raś, Z.W., Wyrzykowska, E., Wasyluk, H. (2008) ARAS: Action rules discovery based on Agglomerative Strategy, in Mining Complex Data, Post-Proceedings of 2007 ECML/PKDD Third International Workshop (MCD 2007), LNAI, Vol. 4944, Springer, 196–208
17. Raś Z W, Tsay L.-S. (2003) Discovering extended action-rules (System DEAR). In: Intelligent Information Systems 2003, Proceedings of the IIS'2003 Symposium, Zakopane, Poland, Advances in Soft Computing, Springer-Verlag, 293–300
18. Raś Z W, Tzacheva A, Tsay, L.-S. (2005) Action rules. In: Encyclopedia of Data Warehousing and Mining, (Ed. J. Wang), Idea Group Inc., 1–5
19. Silberschatz A, Tuzhilin A (1995) On subjective measures of interestingness in knowledge discovery. In: Proceedings of KDD́95 Conference, AAAI Press
20. Silberschatz A, Tuzhilin A (1996) What makes patterns interesting in knowledge discovery systems. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 6
21. Tsay L-S, Raś Z W (2005) Action rules discovery: System DEAR2, method and experiments. In:the Special Issue on Knowledge Discovery, Journal of Experimental and Theoretical Artificial Intelligence, Taylor & Francis, Vol. 17, No. 1-2, 119–128
22. Tsay, L.-S., Raś, Z.W. (2008) Discovering the concise set of actionable patterns, in Foundations of Intelligent Systems, Proceedings of ISMIS′08, A. An et al. (Eds.), Toronto, Canada, LNAI, Springer, 2008, will appear
23. Tzacheva A, Raś Z W (2005) Action rules mining. In:the Special Issue on Knowledge Discovery, International Journal of Intelligent Systems, Wiley, Vol. 20, No. 7, 719–736
24. Tzacheva A, Raś Z W (2007) Constraint based action rule discovery with single classification rules, in Proceedings of the Joint Rough Sets Symposium (JRS07), May 14-16, 2007, in Toronto, Canada, LNAI, Vol. 4482, Springer, 322–329