The Rough Set Exploration System

Jan G. Bazan[1] and Marcin Szczuka[2]

[1] Institute of Mathematics, University of Rzeszów,
Rejtana 16A, 35-310 Rzeszów, Poland
bazan@univ.rzeszow.pl

[2] Institute of Mathematics, Warsaw University,
Banacha 2, 02-097, Warsaw, Poland
szczuka@mimuw.edu.pl

Abstract. This article gives an overview of the Rough Set Exploration System (RSES). RSES is a freely available software system toolset for data exploration, classification support and knowledge discovery. The main functionalities of this software system are presented along with a brief explanation of the algorithmic methods used by RSES. Many of the RSES methods have originated from rough set theory introduced by Zdzisław Pawlak during the early 1980s.

1 Introduction

This paper introduces the latest edition of the *Rough Set Exploration System* (RSES) software system toolset that makes it possible to analyze tabular datasets utilizing various methods. In particular, many RSES methods are based on rough set theory (see, e.g., [20–28, 30–32]). The first version of RSES and its companion RSESlib became available over a decade ago. After a number of modifications, improvements, and removal of detected bugs, RSES has been used in many applications (see, e.g., [5, 11, 14, 15, 29, 34, 41]).

The RSESlib library of tools for rough set computations was successfully used for data analysis with encouraging results. Comparison with other classification systems (see, e.g., [2, 19, 29]) proves its value. The early version of RSESlib was also used in construction of the computational kernel of ROSETTA, an advanced system for data analysis (see,e.g, [12, 42]).

At the moment of this writing RSES version 2.2 is the most current. This version was prepared by the research team supervised by Professor Andrzej Skowron. Currently, the RSES R&D team consists of: Jan Bazan, Rafał Latkowski, Michał Mikołajczyk, Nguyen Hung Son, Nguyen Sinh Hoa, Dominik Ślęzak, Piotr Synak, Marcin Szczuka, Arkadiusz Wojna, Marcin Wojnarski, and Jakub Wróblewski.

The RSES ver. 2.2 software and its computational kernel – the RSESlib 3.0 library – maintains all advantages of previous versions. The algorithms have been redesigned to provide better flexibility, extended functionality and the ability to process massive data sets. New algorithms added to the library reflect the current state of our research in classification methods originating in rough sets theory. Improved construction of library allows further extensions and supports augmentation of RSESlib into different data analysis tools.

Today RSES is freely distributed for non-commercial purposes. Anybody can download it from the Web site [40].

The RSES software and underlying computational methods have been successfully applied in many studies and applications. A system based on LTF-C (see Subsection 8.3) won the first prize in the EUNITE 2002 World Competition “Modeling the Bank’s Client behavior using Intelligent Technologies” (see [41]). Approaches based on rule calculation have been successfully used in areas such as gene expression discovery (see, e.g., [34]), survival analysis in oncology (see, e.g., [5]), software change classification (see, e.g., [29]), classification of biomedical signal data (see, e.g., [15]), and classification of musical works and processing musical data (see, e.g, [11, 14]).

In this paper we attempt to provide quite a general description of capabilities of our software system. We also present a handful of basic facts about the underlying computational methods. The paper starts with introduction of basic notions (Section 2) that introduces the vocabulary for the rest of this article. Next, we describe, in general terms, main functionalities of RSES (Section 3) and architecture of the system (Section 4). Rest of the paper presents computational methods in the order introduced in Figure 1. Starting from input management (Section 5) we go through data preprocessing (Section 6) and data reduction (Section 7), to conclude with description of methods for classifier construction and evaluation (Section 8).

2 Basic Notions

In order to provide clear description further in the paper and avoid any misunderstandings we bring here some essential definitions from Rough Set theory. We will frequently refer to the set of notions introduced in this section. Quite comprehensive description of notions and concepts related to classical rough sets may be found in [13].

The structure of data that is central point of our work is represented in the form of *information system* [28] or, more precisely, the special case of information system called *decision table*.

Information system is a pair of the form $\mathbf{A} = (U, A)$ where U is a finite *universe* of *objects* and $A = \{a_1, ..., a_m\}$ is a set of *attributes*, i.e., mappings of the form $a_i : U \to V_a$, where V_a is called *value set* of the attribute a_i. The decision table is also a triple of the form $\mathbf{A} = (U, A, d)$ where the major feature that is different from the information system is the distinguished attribute d. In case of decision table the attributes belonging to A are called *conditional attributes* or simply *conditions* while d is called *decision* (sometimes *decision attribute*). The cardinality of the image $d(U) = \{k : d(s) = k \text{ for some } s \in U\}$ is called the *rank of* d and is denoted by $rank(d)$.

One can interpret the decision attribute as a kind of classifier on the universe of objects given by an expert, a decision-maker, an operator, a physician, etc. This way of looking at data classification task is directly connected to the general idea of *supervised learning* (learning with a “teacher”).

Decision tables are usually called training sets or training samples in machine learning.

The $i-$th *decision class* is a set of objects $C_i = \{o \in U : d(o) = d_i\}$, where d_i is the $i-$th decision value taken from decision value set $V_d = \{d_1, ..., d_{rank(d)}\}$.

For any subset of attributes $B \subset A$ *indiscernibility relation* $IND(B)$ is defined as follows:

$$xIND(B)y \Leftrightarrow \forall_{a \in B} a(x) = a(y) \tag{1}$$

where $x, y \in U$. If a pair $(x, y) \in U \times U$ belongs to $IND(B)$ then we say that x and y are indiscernible by attributes from B.

Having indiscernibility relation we may define the notion of reduct. $B \subset A$ is a (global) *reduct* of information system if $IND(B) = IND(A)$ and no proper subset of B has this property. There may be many reducts for a given information system (decision table). We are usually interested only in some of them, in particular those leading to good classification models.

As the discernibility relation and reducts (reduction) are the key notions in classical rough sets we want to dwell a little on these notions and their variants used in RSES.

A *relative* (local) reduct for an information system $\mathbf{A} = (U, A)$ and an object $o \in U$ is an irreducible subset of attributes $B \subset A$ that suffices to discern o from all other objects in U. Such reduct is only concerned with discernibility relative to the preset object.

In case of decision tables the notion of *decision reduct* is handy. The decision reduct is a set $B \subset A$ of attributes such that it cannot be further reduced and $IND(B) \subset IND(d)$. In other words, decision reduct is a reduct that only cares about discernibility of objects that belong to different decision classes. This works under assumption that the table is consistent. If consistency is violated, i.e., there exist at least two objects with identical attribute values and different decision, the notion of decision reduct can still be utilized, but the notion *generalized decision* has to be used.

As in general case, there exists a notion of a decision reduct relative to an object. For an object $o \in U$ the relative decision reduct is a subset $B \subset A$ of attributes such that it cannot be further reduced and suffices to make o discernible from all objects that have decision value different from $d(o)$.

Dynamic reducts are reducts that are calculated in a special way. First, from the original information system a family of subsystems is selected. Then, for every member of this family (every subsystem) reducts are calculated. Reducts (subsets of attributes) that appear in results for many subtables are chosen. They are believed to carry essential information, as they are reducts for many parts of original data. For more information on dynamic reducts please refer to [2].

The set $\underline{B}X$ is the set of all elements of U which can be classified with certainty as elements of X, having the knowledge about them represented by attributes from B; the set $BN_B(X)$ is the set of elements which one can classify neither to X nor to $-X$ having knowledge about objects represented by B.

If $C_1, \ldots, C_{r(d)}$ are decision classes of $\mathbf{A}$ then the set $\underline{B}C_1 \cup \cdots \cup \underline{B}C_{rank(d)}$ is called *the B-positive region of* $\mathbf{A}$ and is denoted by $POS_B(d)$. The relative size of positive region, i.e., the ratio $\frac{|POS_B(d)|}{|U|}$, is an important indicator used in RSES.

Decision rule r is a formula of the form

$$(a_{i_1} = v_1) \wedge ... \wedge (a_{i_k} = v_k) \Rightarrow d = v_d \tag{2}$$

where $1 \leq i_1 < ... < i_k \leq m$, $v_i \in V_{a_i}$. Atomic subformulae $(a_{i_1} = v_1)$ are called *descriptors* or *conditions.* We say that rule r is *applicable* to object, or alternatively, the object *matches* rule, if its attribute values satisfy the premise of a rule.

With the rule we can associate some characteristics. $Supp_{\mathbf{A}}(r)$ denotes *Support*, and is equal to the number of objects from $\mathbf{A}$ for which rule r applies correctly, i.e., the premise of rule is satisfied and the decision given by rule is similar to the one preset in decision table. $Match_{\mathbf{A}}(r)$ is the number of objects in $\mathbf{A}$ for which rule r applies in general. Analogously the notion of matching set for a rule or collection of rules may be introduced (see [2–4]).

The notions of matching and supporting set are common to all classifiers, not only decision rules. For a classifier C we will denote by $Supp_{\mathbf{A}}(C)$ the set of objects that support classifier, i.e., the set of objects for which classifier gives the answer (decision) identical to that we already have. Similarly, $Match_{\mathbf{A}}(C)$ is a subset of objects in $\mathbf{A}$ that are recognized by C. Support and matching make it possible to introduce two measures that are used in RSES for classifier scoring. These are Accuracy and Coverage, defined as follows:

$$Accuracy_{\mathbf{A}}(C) = \frac{|Supp_{\mathbf{A}}(C)|}{|Match_{\mathbf{A}}(C)|}$$

$$Coverage_{\mathbf{A}}(C) = \frac{|Match_{\mathbf{A}}(C)|}{|\mathbf{A}|}$$

By *cut* for an attribute $a_i \in A$, such that V_{a_i} is an ordered set we will denote a value $c \in V_{a_i}$. Cuts mostly appear in the context of discretization of real-valued attributes. In such situation, the cut is a a value for an attribute such that it determines a split of attribute domain (interval) into two disjoint subintervals.

With the use of cut we may replace original attribute a_i by a new, binary attribute which tells as whether actual attribute value for an object is greater or lower than c (more in [17]).

Template of $\mathbf{A}$ is a propositional formula $\bigwedge(a_i = v_i)$ where $a_i \in A$ and $v_i \in V_{a_i}$. A generalized template is the formula of the form $\bigwedge(a_i \in T_i)$ where $T_i \subset V_{a_i}$. An object *satisfies* (matches) a template if for every attribute a_i occurring in the template the value of this attribute on considered object is equal to v_i (belongs to T_i in case of generalized template). The template induces in natural way the split of original information system into two distinct subtables. One of those subtables contains objects that satisfy the template, the other those that do not.

Decomposition tree is a binary tree, whose every internal node is labeled by some template and external node (leaf) is associated with a set of objects matching all templates in a path from the root to a given leaf (see [16] for more details).

3 Main Functionalities

RSES is a computer software system developed for the purpose of analyzing data. The data is assumed to be in the form of information system or decision table. The main step in the process of data analysis with RSES is the construction and evaluation of classifiers.

Classification algorithms, or classifiers, are algorithms that permit us to repeatedly make a forecast in new situation on the basis of accumulated knowledge. In our case the knowledge is embedded in the structure of classifier which itself is constructed (learned) from data (see [19]). RSES utilizes classification algorithms using elements of rough set theory, instance based learning, artificial neural networks and others. Types of classifiers that are available in RSES are discussed in Section 8.

The construction of classifier is usually preceded by several initial steps. First, the data for analysis has to be loaded/imported into RSES. RSES can accept several input formats as described in Section 5. Once the data is loaded, the user can examine it using provided visualization and statistics tools (see Figures 3F3 and 3F4).

In order to have a better chance for constructing (learning) a proper classifier, it is frequently advisable to transform the initial data set. Such transformation, usually referred to as *preprocessing* may consist of several steps. RSES supports preprocessing methods which make it possible to manage missing parts in data, discretize numeric attributes, and create new attributes. Preprocessing methods are further described in Section 6.

Once the data is preprocessed we may be interested in learning about its internal structure. By using classical rough set concepts such as reducts, dynamic reducts and positive region one may pinpoint dependencies that occur in our data set. Knowledge of reducts may lead to reduction of data by removing some of the redundant attributes. Reducts can also provide essential hints for the parameter setting during classifier construction. Calculation of reducts and their usage is discussed in Section 7.

The general scheme of data analysis process with use of RSES functionalities in presented in Figure 1.

4 The Architecture of RSES

To simplify the use of RSES algorithms and make it more intuitive the RSES graphical user interface was constructed. It is directed towards ease of use and visual representation of workflow. Project interface window consists of two parts. The visible part is the project workspace with icons representing objects created during blue computation. Behind the project window there is the history

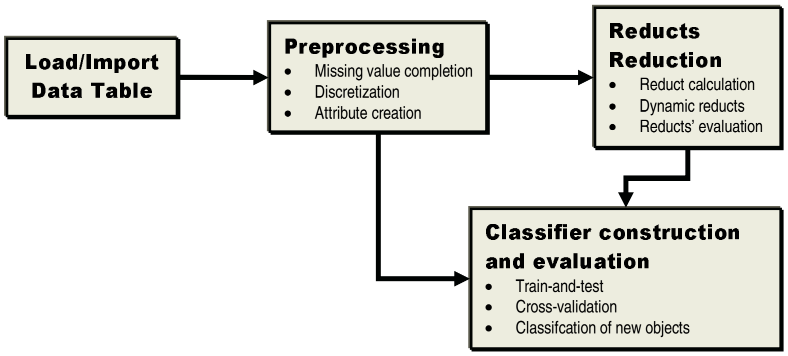


Fig. 1. RSES data analysis process.

window, reachable via tab, and dedicated to messages, status reports, errors and warnings. While working with multiple projects, each of them occupies a separate workspace accessible via tab at the top of workplace window.

It was designers' intention to simplify the operations on data within project. Therefore, the entities appearing in the process of computation are represented in the form of icons placed in the upper part of workplace. Such an icon is created every time the data (table, reducts, rules,...) is loaded from the file. Users can also place an empty object in the workplace and further fill it with results of operation performed on other objects. Every object appearing in the project has a set of actions associated with it. By right-clicking on the object the user invokes a context menu for that object. It is also possible to invoke an action from the general pull-down program menu in the main window. Menu choices make it possible to view and edit objects as well as include them in new computations. In

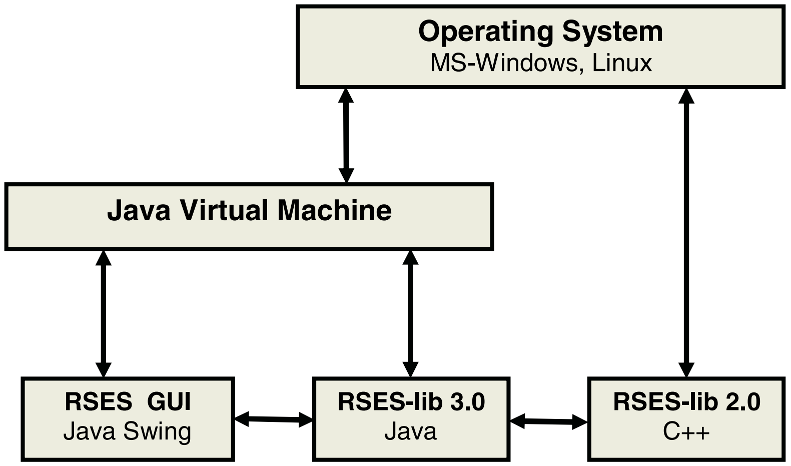


Fig. 2. The architecture of RSES.

many cases a command from context menu causes a new dialog box to open. In this dialog box the user can set values of parameters used in desired calculation. If the operation performed on the object leads to creation of a new object or modification of existing one then such a new object is connected with edge originating in object(s) which contributed to its current state. Placement of arrows connecting icons in the workspace changes dynamically as new operations are being performed. The user has the ability to align objects in workspace automatically, according to his/her preferences (eg. left, horizontal, bottom).

An important, recently added GUI feature is the possibility to display some statistical information about tables, rules and reducts in a graphical form. The look of various components of RSES interface is shown in Figure 3.

Behind the front-end that is visible to the user, there is RSES computational kernel. This most essential part of the system is built around the library of methods known as RSESlib ver. 3.0. The library is mostly written in Java but, it also uses a part that was implemented using C++. The C++ part is the legacy of previous RSESlib versions and contains those algorithms that could only lose optimality if re-implemented in Java. The layout of RSES components is presented in Figure 2. Currently, it is possible to install RSES in Microsoft Windows 95/98/2000/XP and in Linux/i386. The computer on which the RSES is installed has to be equipped with Java Runtime Environment.

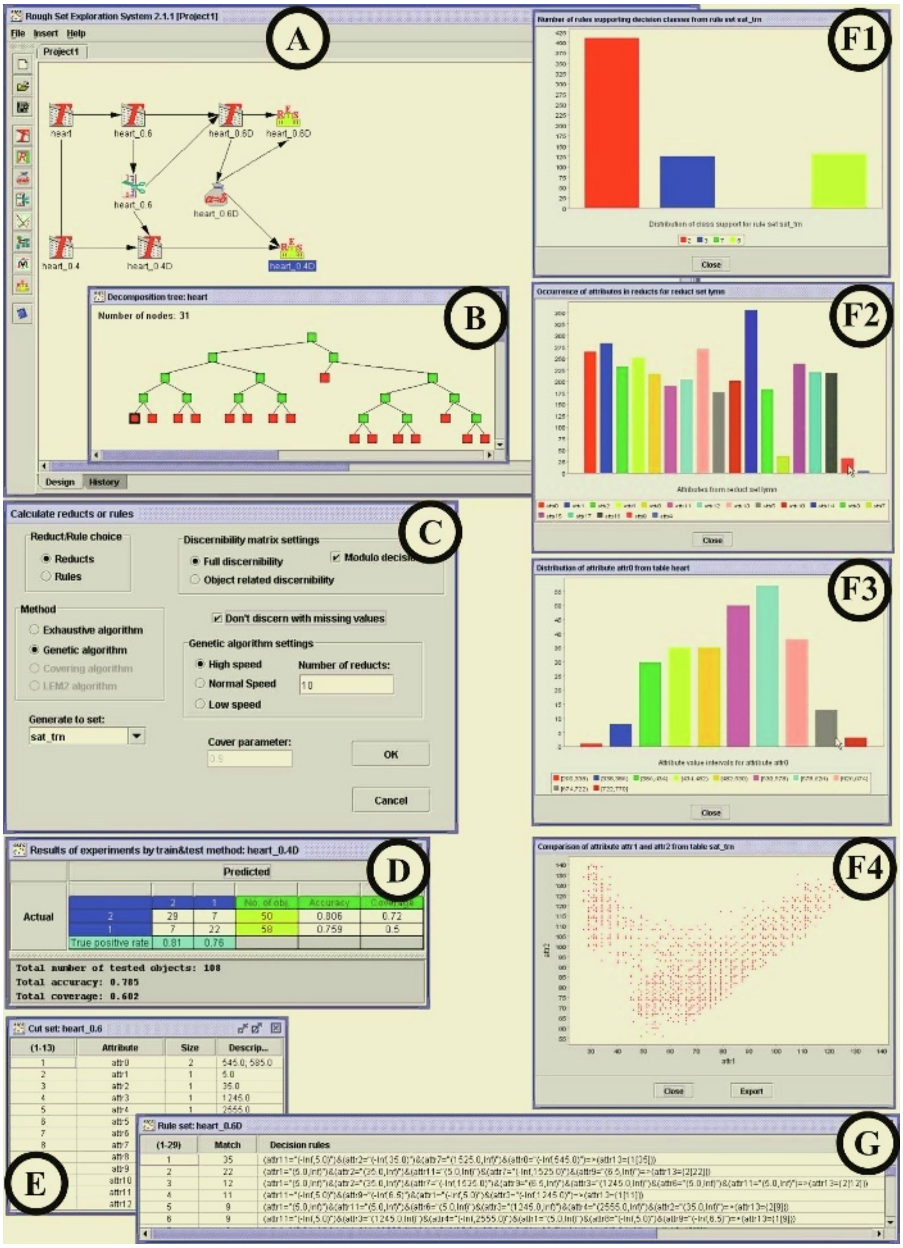
5 Managing Input and Output

During operation certain functions in RSES may read and write information to/from files. Most of the files that can be read or written are regular ASCII text files. A particular sub-types can be distinguished by reviewing the contents or identifying file extensions (if used). Description of RSES formats can be found in [40].

As the whole system is about analyzing tabular data, it is equipped with abilities to read several tabular data formats. At the time of this writing the system can import text files formatted for old version of RSES (RSES1 format), Rosetta, and Weka systems. Naturally, there exists native RSES2 file format used to store data tables.

The old RSES1 format is just a text file that stores data table row by row. The only alternation is in the first row, which defines data dimension – number of rows and columns (attributes and objects). All the other input formats are more complex. The file in these formats consist of the header part and the data part. The header part defines structure of data, number and format of attributes, attribute names etc. Additional information from the header proved to be very useful during analysis, especially in interpretation of results. For detailed definition of these formats please refer to [42] for Rosetta, [43] for Weka, and [40] for RSES.

The RSES user can save and retrieve data entities created during experiment, such as rule sets, reduct sets etc. The option of saving the whole workspace (project) in a single file is also provided. The project layout together with underlying data structures is stored using dedicated, optimized binary file format.


A
B
C
D
E
F1
F2
F3
F4
G
Rough Set Exploration System 2.1.1 [Project1]
File Insert Help
Project1
heart
heart_0.6
heart_0.6D
heart_0.6D
heart_0.6
heart_0.6D
heart_0.4
heart_0.4D
heart_0.4D
Design
History
Decomposition tree: heart
Number of nodes: 31
Number of rules supporting decision classes from rule set sat_trn
Distribution of class support for rule set sat_trn
Close
Occurrence of attributes in reducts for reduct set lymn
Attributes from reduct set lymn
Close
Distribution of attribute attr0 from table heart
Attribute value intervals for attribute attr0
Close
Comparison of attribute attr1 and attr2 from table sat_trn
attr2
attr1
Close
Export

Calculate reducts or rules

Reduct/Rule choice: [✓ Reducts] Rules

Method: Exhaustive algorithm [✓ Genetic algorithm] Covering algorithm LEM2 algorithm

Generate to set: sat_trn

Discernibility matrix settings: [✓ Full discernibility] Object related discernibility; [✓ Modulo decision]

[✓ Don't discern with missing values]

Genetic algorithm settings: [✓ High speed] Normal Speed Low speed; Number of reducts: 10

Cover parameter: 0.9

OK Cancel

Results of experiments by train&test method: heart_0.4D

		Predicted				
		2	1	No. of obj.	Accuracy	Coverage
Actual	2	29	7	50	0.806	0.72
	1	7	22	58	0.759	0.5
	True positive rate	0.81	0.76			

```
Total number of tested objects: 108
Total accuracy: 0.785
Total coverage: 0.602
```

Cut set: heart_0.6

(1-13)	Attribute	Size	Descrip...
1	attr0	2	545.0; 585.0
2	attr1	1	5.0
3	attr2	1	35.0
4	attr3	1	1245.0
5	attr4	1	2555.0
6	attr5		
7	attr6		
8	attr7		
	attr8		
	attr9		
	attr10		
	attr11		
	attr12		

Rule set: heart_0.6D

(1-29)	Match	Decision rules
1	35	(attr11="(-Inf,5.0)")&(attr2="(-Inf,35.0)")&(attr7="(1525.0,Inf)")&(attr0="(-Inf,545.0)")=>(attr13={1[35]})
2	22	(attr1="(5.0,Inf)")&(attr2="(35.0,Inf)")&(attr11="(5.0,Inf)")&(attr7="(-Inf,1525.0)")&(attr9="(6.5,Inf)")=>(attr13={2[22]})
3	12	(attr1="(5.0,Inf)")&(attr2="(35.0,Inf)")&(attr7="(-Inf,1525.0)")&(attr9="(6.5,Inf)")&(attr3="(1245.0,Inf)")&(attr6="(5.0,Inf)")&(attr11="(5.0,Inf)")=>(attr13={2[12]}
4	11	(attr11="(-Inf,5.0)")&(attr9="(-Inf,6.5)")&(attr1="(-Inf,5.0)")&(attr3="(-Inf,1245.0)")=>(attr13={1[11]})
5	9	(attr1="(5.0,Inf)")&(attr11="(5.0,Inf)")&(attr6="(5.0,Inf)")&(attr3="(1245.0,Inf)")&(attr4="(2555.0,Inf)")&(attr2="(35.0,Inf)")=>(attr13={2[9]})
6	9	(attr11="(-Inf,5.0)")&(attr3="(1245.0,Inf)")&(attr4="(-Inf,2555.0)")&(attr1="(5.0,Inf)")&(attr6="(-Inf,5.0)")&(attr9="(-Inf,6.5)")=>(attr13={1[9]})

Fig. 3. RSES interface windows: A. Project window; B. Decomposition tree; C. Reduct/rule calculation; D. Classification results; E. Set of cuts; F1–F4. Graphical statistics for rules, reducts and attributes; G. Decision rules.

6 Data Preprocessing

The data that is an input to analysis with use of RSES can display several features that negatively influence quality and generality of classification model we want to construct. There may be several such problems with data, depending on classification paradigm we want to apply. Some of the RSES classifier construction methods require data (attributes) to be represented with use of specific value sets, e.g., only symbolic attributes are allowed. There may be other problems, such as missing information (missing values) or inconsistencies in the table. Finally, we may want to alter the original data by introducing new attributes that are better suited for classification purposes.

RSES provides users with several functionalities that make it possible to preprocess original data and prepare a training sample which more likely to lead to a good classifier.

6.1 Examining Data Statistics

Not a preprocessing method *per se*, RSES functions for producing data statistics may serve as a handy tool. By using them the user can get familiar with the data as well as compare some crucial data characteristics before and after data modification.

In RSES the user may examine distribution of a single attribute, in particular the decision distribution. The RSES system is capable of presenting numerical measurements for distribution of values of a single attribute as well as displaying the corresponding histogram (see Figure 3 F3). The information gathered for a single attribute includes range, mean value, and standard deviation. It is also possible to compare distribution of two attributes on a single plot as shown in Figure 3 F4.

6.2 Missing Value Completion

The missing elements in data table, so called `NULL` values, may pose a problem when constructing classifier. The lack of some information may be due to incomplete information, error or the constraints embedded in data collection process.

RSES offers four approaches to the issue of missing values. These are as follows:

– removal of objects with missing values,
– filling the missing part of data in one of two ways (see [10]):
  - filling the empty (missing) places with most common value in case of nominal attributes and filling with mean over all attribute values in data set in case of numerical attribute.
  - filling the empty (missing) places with most common value for the decision class in case of nominal attributes and filling with mean over all attribute values in the decision class in case of numerical attribute.

– analysis of data without taking into account those objects that have incomplete description (contain missing values). Objects with missing values (and their indiscernibility thereof) are disregarded during rule/reduct calculation. This result is achieved by activating corresponding options in the RSES dialog windows for reduct/rule calculation.
– treating the missing data as information (`NULL` is treated as yet another regular value for attribute).

6.3 Discretization

Suppose we have a decision table $\mathbf{A} = (U, A \cup \{d\})$ where $card(V_a)$ is high for some $a \in A$. Then there is a very low chance that a new object is recognized by rules generated directly from this table, because the attribute value vector of a new object will not match any of these rules. Therefore for decision tables with real (numerical) value attributes some discretization strategies are built into RSES in order to obtain a higher quality of classification.

Discretization in RSES is a two-fold process. First, the algorithm generates a set of cuts (see Figure 3 E). These cuts can be then used for transforming a decision table. As a result we obtain the decision table with the same set of attributes, but the attributes have different values. Instead of $a(x) = v$ for an attribute $a \in A$ and object $x \in U$ we rather get $a(x) \in [c_1, c_2]$, where c_1 and c_2 are cuts generated for attribute a by discretization algorithm. The cuts are generated in a way that the resulting intervals contain possibly most uniform sets of objects w.r.t decision.

The discretization method available in RSES has two versions, code-named *global* and *local*. Both methods are using a bottom-up approach which adds cuts for a given attribute one-by-one in subsequent iterations of algorithm. The difference between these two is in the way the candidate for new cut is scored. In the global method we consider in scoring all the objects in data table at every step. In the local method we only take part of objects that are concerned with this candidate cut, i.e., which have value of the currently considered attribute in the same range as the cut candidate. Naturally, the second (local) method is faster as less objects have to be examined at every step. In general, the local method is producing more cuts. The local method is also capable of dealing with nominal (symbolic) attributes. The grouping (quantization) of nominal attribute domain with use of local method always results in two subsets of attribute values.

6.4 Creation of New Attributes

RSES makes it possible to add an attribute to decision table. This new attribute is created as a weighted sum of selected existing (numerical) attributes. We may have several such attributes for different weight settings and different attributes participating in weighted sum. These attributes are carrying agglomerated information and are intended as a way of simplifying classifier construction for a given data set.

Linear combinations are created on the basis of collection of attribute sets consisting of k elements. These k-element attribute sets as well as parameters of combination (weights) are generated automatically by adaptive optimization algorithm implemented in RSES. As a measure for optimization we may use one of three possibilities. The measures take into account potential quality of decision rules constructed on the basis of newly created linear combination attribute. For details on these measures please turn to [33]. The user may specify the number (k) of new attributes to be constructed and the number of original attributes to be used in each linear combination.

7 Reduction and Reducts

As mentioned before (Section 2) there are several types of reducts that may be calculated in RSES for a given information system (decision table). The purposes for performing reduct calculation (reduction) may vary. One of advantages that reducts offer is better insight into data. By calculating reducts we identify this part of data (features) which carries most essential information. Reducts are also canvas for building classifiers based on decision rules.

Inasmuch as calculation of interesting, meaningful and useful reducts may be a complicated and computationally costly task, there is a necessity for larger flexibility when setting up an experiment involving reducts. For that reason there are several different methods for discovering reducts implemented in RSES.

7.1 Calculating Reducts

In Section 2 we mentioned four general types of reducts. All four of them can be derived in RSES by selecting appropriate settings for algorithms implemented (see Figure 3 C). It is important to mention that in most cases selecting object-related indiscernibility relation (local reducts) leads to creation of much larger set of reducts. On the other hand, selection of decision-dependant indiscernibility usually reduces computational cost.

There are two algorithms for reducts calculation available in RSES. First of them is an exhaustive algorithm. It examines subsets of attributes incrementally and returns those that are reducts of required type. This algorithm, although optimized and carefully implemented, may lead to very extensive calculations in case of large and complicated information systems (decision tables). Therefore, should be used with consideration.

To address problems with sometimes unacceptable cost of exhaustive, deterministic algorithm for reduct calculation, an alternative evolutionary method is implemented in RSES. This method is based on an order-based genetic algorithm coupled with heuristic. Theoretical foundations and practical construction of this algorithm are presented in [37] and [3].

The user, when invoking this method, has some control over its behavior since the population size and convergence speed may be set from the RSES interface (see Figure 3 C).

7.2 Calculating Dynamic Reducts

Dynamic reducts are reducts that remain to be such for many subtables of the original decision table (see [2], [3]). The process of finding the dynamic reduct is computationally more costly, as it requires several subtables to be examined in order to find the frequently repeating minimal subsets of attributes (dynamic reducts). Such dynamic reducts may be calculated for general or decision-related indiscernibility relation.

The purpose of creating dynamic reducts is that in many real-life experiments they proved to be more general and better suited for creation of meaningful and applicable decision rules. Examples of solutions and results obtained with use of dynamic reduct approach may be found in [2, 3].

The dynamic reduct calculation process involves sampling several subtables from original table and is controlled by number of options such as: number of sampling levels, number of subtables per sampling level, smallest and largest permissible subtable size, and so on. We also decide right on the start if we are interested in general or decision-related reducts.

As mentioned before, calculation of reducts may be computationally expensive. To avoid overly exhaustive calculation it is advisable to carefully select parameters for dynamic reduct calculation, taking into account size of data, number of samples, and size of attribute value sets.

7.3 From Reducts to Rules

Reducts in RSES are, above all, used to create decision rule sets. Equipped with collection of reducts (reduct set) calculated beforehand, the user may convert them into a set of decision rules. That may be achieved in two ways, as there are two methods for creating rules from reducts implemented in RSES.

First option is to calculate so called *global rules*. The algorithm scans the training sample object by object and produces rules by matching object against reduct. The resulting rule has attributes from reducts in conditional part with values of currently considered object, and points at decision that corresponds to the decision for this training object. Note, that for large tables and large reduct set the resulting set of rules may be quite large as well.

Another alternative is to generate all *local rules*. For each reduct a subtable, containing only the attributes present in this reduct, is selected. For this subtable algorithm calculates a set of minimal rules (rules with minimal number of descriptors in conditional part – see, e.g., [3]) w.r.t decision. Finally, the rule sets for all reducts are summed up to form result.

Sets of reducts obtained in RSES may be examined with use of included graphical module. This module makes it possible to find out how the attributes are distributed among reducts and how reducts overlap (see Figure 3 F2). In case of decision reduct it is also possible to verify the size of positive region.

Sets of decision rules obtained as a result may be quite large and some of the rules may be of marginal importance. This can be, among other things, the result of using reducts that are of mediocre quality.

The quality of reducts may be improved after they are calculated. One way to do that is by shortening. The user may specify the shortening threshold (between 0 and 1) and the algorithm will attempt to remove some attributes from reducts. The shortening is performed as long as the relative size of positive region for shortened reduct exceeds the threshold. Note, that in the process of shortening the resulting subset of attributes may no longer be a proper reduct but it is shorter and results in more general rules.

In case of dynamic reduct there is even more room for improvement here. We may perform reduct filtering, eliminating reducts of lower expected quality. To filter dynamic reducts in RSES the user has to set a value of *stability coefficient* (between 0 and 1). This coefficient is calculated for each dynamic reduct during the process of its creation. We will not bring here the entire theory behind the stability coefficient. Interested reader may find detailed explanation in [2] and [3]. Important thing to know is that stability coefficient keeps the record of appearances of a reduct for subtables sampled in reduct calculation process. The more frequent occurrence of the reduct (the greater stability) the higher stability coefficient. High stability (coefficient) of a dynamic reducts strongly suggest that it contains vital piece of information. Naturally, there is no point in considering stability coefficient filtering in case on non-dynamic (regular) reducts, as there in no sampling involved in their creation, and their stability coefficients always equal 1.

8 Construction and Utilization of Classifiers

Several types of classifiers are represented in RSES, and we present them in some detail in subsequent parts of this section. All of them follow the scheme of construction, evaluation and usage.

The classifier in RSES is constructed on the basis of training set consisting of labeled examples (objects with decisions). Such a classifier may be further used for evaluation with use of test/validation set or applied to new, unseen and unlabeled cases in order to establish the value of decision (classification) for them.

The evaluation of the classifier's performance in RSES may be conducted in two ways. We can either apply a *train-and-test* (also known as hold-out) or *cross-validation* procedure.

In train-and-test scenario the (labeled) data is split into two parts of which first becomes the training, second the testing/validation set. The classifier is build on the basis of the training set and then evaluated with use of testing one. The choice of method for splitting data into training and testing set depends on the task at hand. For some tasks this split is imposed by the task, the nature of data or the limitations of the methods to be applied. If there are no constraints on the data split, the training and testing sample is chosen by random. The responses given by the classifier for test table are compared with desired answers (known for our data set) and the classification errors are calculated. The results of such procedure are stored in dedicated object in RSES project interface. The set of results, when displayed (see Figure 3D), provide the user with values of accuracy and coverage of classifier for the entire testing data as well as for particular decision classes. The distribution of errors made by classifier on test data set is shown in detail using the typical confusion matrix (see, e.g., [19]).

In the k-fold cross-validation approach the data is split into k possibly equal parts (folds). Then the train-and-test procedure described above is applied repeatedly k times in such a way that each of k parts becomes the test set while the sum of remaining $k-1$ parts is used as a training set to construct classifier. In each run classification errors are calculated for the current test set. As a result the RSES returns the same set of results as in train-and-test approach but, the values of errors are averages over k iterations. The cross-validation approach to classifier evaluation is commonly used and has a good theoretical background (see [19]), especially for data sets with no more than 1000 objects. In RSES the application of cross-validation scheme is controlled with use of dedicated window which makes it possible to select number of folds and all important parameters for a classifier to be constructed and evaluated.

When using a previously constructed classifier for establishing decision for previously unseen, unlabeled objects, the user have to take care of the proper format of the examples. If during construction of classifier the training set was preprocessed (e.g., discretized) then the same procedure has to be repeated for the new data table. If the format of data and the classifier match, the result is created as a new column in the data table. This column contains the value of decision predicted by classifier for each object.

8.1 Decision Rules

Classifiers based on a set of decision rules are the most established methods in RSES. Several methods for calculation of the decision rule sets are implemented. Also, various methods for transforming and utilizing rule sets are available (see parts C, F1 and G of Figure 3).

The methods for retrieving rules, given a set of reducts, have been already described in Subsection 7.3. These methods produce set of rules by matching training objects against selected set of reducts. In RSES it is possible to calculate such rules instantly, without outputting the set of reducts. But, it has to be stated that the reduct calculation is performed in background anyway.

The two methods for rule calculation that use reducts, i.e., the exhaustive and GA algorithms, are accompanied with another two that are based on slightly different approach. These two are applying a *covering approach*. First of the two utilizes subsets of attributes that are likely to be local (relative) reducts. The details of this method are described in [38]. Second of the covering algorithm is a customized implementation of the LEM2 concept introduced by Jerzy Grzymała-Busse in [9]. In LEM2 a separate-and-conquer technique is paired with rough set notions such as upper and lower approximation. Both covering-based methods for rule calculation tend to produce less rules than algorithms based on explicit reduct calculation. They are also (on average) slightly faster. On the downside, the covering methods sometimes return too few valuable and meaningful rules.

In general, the methods used by RSES to generate rules may produce quite a bunch of them. Naturally, some of the rules may be marginal, erroneous or redundant. In order to provide better control over the rule-based classifiers some simple techniques for transforming rule sets are available in RSES. Note, that before any transformation of rule set it is advisable to examine the statistics produced in RSES for such an entity (see Figure 3 F1). The simplest way to alter the set of decision rules is by filtering them. It is possible to eliminate from rule set these rules that have insufficient support on training sample, or those that point at decision class other than desired.

More advanced operations on rule sets are *shortening* and *generalization.* Rule shortening is a method that attempts to eliminate descriptors from the premise of the rule. The resulting rule is shorter, more general (apply to more training objects) but, it may lose some of its precision. The shortened rule may be less precise, i.e., may give wrong answers (decision) for some of the matching training objects. The level to which we accept decrease of quality in favor of improved generality of rules is known as *shortening ratio* and may be set by the user of RSES. Generalization is the process which attempts to replace single descriptors (conditions) in the rule with more general ones. Instead of a unary condition of the form $a(x) = v$, where $a \in A$, $v \in V_a$, $x \in U$, the algorithm tries to use *generalized descriptors* of the form $a(x) \in V_c$, where $V_c \subset V_a$. Note, that in generalization process the implicit assumption about manageable size of V_a for each $a \in A$ is crucial for the algorithm to be computationally viable. A descriptor (condition) in a rule is replaced by its generalized version if such a change do not decrease size of positive region by the ratio higher than a threshold set by the user.

When we attempt to classify an object from test sample with use of generated rule set it may happen that various rules suggest different decision values. In such conflict situations we need a strategy to resolve controversy and reach a final result (decision). RSES provides a conflict resolution strategy based on voting among rules. In this method each rule that matches the object under consideration casts a vote in favor of the decision value it points at. Votes are summed up and the decision that got majority of votes is chosen. This simple method (present in RSES) may be extended by assigning weights to rules. Each rule then votes with its weight and the decision that has the highest total of weighted votes is the final one. In RSES this method (also known as *Standard voting*) assigns each rule the weight that is equal to the number of training objects supporting this rule.

8.2 Instance Based Method

As an instance based method we implemented the special, extended version of the k nearest neighbors (k-nn) classifier [7]. First the algorithm induces a distance measure from a training set. Then for each test object it assigns a decision based on the k nearest neighbors of this object according to the induced distance measure.

The distance measure ρ for the k-nn classifier is defined as the weighted sum of the distance measures ρ_a for particular attributes $a \in A$:

$$\rho(x,y) = \sum_{a \in A} w_a \cdot \rho_a(a(x), a(y)).$$

Two types of a distance measure are available to the user. The City-SVD metric [6] combines the city-block Manhattan metric for numerical attributes with the Simple Value Difference (SVD) metric for symbolic attributes.

The distance between two numerical values $\rho_a(a(x), a(y))$ is the difference $|a(x) - a(y)|$ taken either as an absolute value or normalized with the range $a_{\max} - a_{\min}$ or with the doubled standard deviation of the attribute a on the training set. The SVD distance $\rho_a(a(x), a(y))$ for a symbolic attribute a is the difference between the decision distributions for the values $a(x)$ and $a(y)$ in the whole training set. Another metric type is the SVD metric. For symbolic attributes it is defined as in the City-SVD metric and for a numerical attribute a the difference between a pair of values $a(x)$ and $a(y)$ is defined as the difference between the decision distributions in the neighborhoods of these values. The neighborhood of a numerical value is defined as the set of objects with similar values of the corresponding attribute. The number of objects considered as the neighborhood size is the parameter to be set by a user.

A user may optionally apply one of two attribute weighting methods to improve the properties of an induced metric. The distance-based method is an iterative procedure focused on optimizing the distance between the training objects correctly classified with the nearest neighbor in a training set. The detailed description of the distance-based method is described in [35]. The accuracy-based method is also an iterative procedure. At each iteration it increases the weights of attributes with high accuracy of the 1-nn classification.

As in the typical k-nn approach a user may define the number of nearest neighbors k taken into consideration while computing a decision for a test object. However, a user may use a system procedure to estimate the optimal number of neighbors on the basis of a training set. For each value k in a given range the procedure applies the leave-one-out k-nn test and selects the value k with the optimal accuracy. The system uses an efficient leave-one-out test for many values of k as described in [8].

When the nearest neighbors of a given test object are found in a training set they vote for a decision to be assigned to the test object. Two methods of nearest neighbors voting are available. In the simple voting all k nearest neighbors are equally important and for each test object the system assigns the most frequent decision in the set of the nearest neighbors. In the distance-weighted voting each nearest neighbor vote is weighted inversely proportional to the distance between a test object and the neighbor. If the option of filtering neighbors with rules is checked by a user, the system excludes from voting all the nearest neighbors that produce a local rule inconsistent with another nearest neighbor (see [8] for details).

The k-nn classification approach is known to be computationally expensive. The crucial time-consuming task is searching for k nearest neighbors in a training set. The basic approach is to scan the whole training set for each test object. To make it more efficient an advanced indexing method is used [35]. It accelerates searching up to several thousand times and allows to test datasets of a size up to several hundred thousand of objects.

8.3 Local Transfer Function Classifier

Local Transfer Function Classifier (LTF-C) is a neural network solving classification problems [36]. LTF-C was recently added to RSES as yet another classification paradigm. Its architecture is very similar to this of Radial Basis Function neural network (RBF) or Support Vector Machines (SVM) – the network has a hidden layer with gaussian neurons connected to an output layer of linear units. The number of inputs corresponds to the number of attributes while the number of linear neurons in output layers equals the number of decision classes. There are some additional restrictions on values of output weights that enable to use an entirely different training algorithm and to obtain very high accuracy in real-world problems.

The training algorithm of LTF-C comprises four types of modifications of the network, performed after every presentation of a training object:

1. changing positions (means) of gaussians,
2. changing widths (deviations) of gaussians, separately for each hidden neuron and attribute,
3. insertion of new hidden neurons,
4. removal of unnecessary or harmful hidden neurons.

As one can see, the network structure is dynamical. The training process starts with an empty hidden layer, adding new hidden neurons when the accuracy is insufficient and removing the units which do not positively contribute to the calculation of correct network decisions. This feature of LTF-C enables automatic choice of the best network size, which is much easier than setting the number of hidden neurons manually. Moreover, this helps to avoid getting stuck in local minima during training, which is a serious problem in neural networks trained with gradient-descend. The user is given some control over the process of network construction/trainig. In particular, it is for user to decide how rigid are the criteria for creating and discarding neurons in the hidden layer. Also, the user may decide whether to perform data (attribute) normalization or not.

8.4 Decomposition Trees

Decomposition trees are used to split data set into fragments not larger than a predefined size. These fragments, after decomposition represented as leafs in decomposition tree, are supposed to be more uniform and easier to cope with decision-wise.

The process of constructing a decomposition tree is fully automated, the user only has to decide about the maximal size of subtable corresponding to the leaf. The algorithm generates conditions one by one on subsequent levels of the tree. The conditions are formulated as a constraints for value of particular attribute. In this way, each node in the tree have an associated template as well as subset of training sample that corresponds to this template. It is possible to generate decomposition trees for data with numerical attributes. In this case discretization is performed during selection of conditions in tree nodes. A dedicated display method for presenting decomposition trees is implemented in RSES GUI, so that the user can examine interactively the resulting decomposition (see Figure 3B). For more information on underlying methods please turn to [16] and [18].

The decomposition tree is mostly used in RSES as a special form of classifier. Usually the subsets of data in the leafs of decomposition tree are used for calculation of decision rules (cf. [3]). The set of data in the leaf is selected by the algorithm in such a way, that (almost) all objects it contains belong to the same decision class. If such a set of objects is used to generate rules, there is a good chance of obtaining some significant decision rules for the class corresponding to the leaf.

The tree and the rules calculated for training sample can be used in classification of unseen cases. The rules originating in decomposition tree may be managed in the same manner as all other decision rules in RSES. It is possible to generalize and shorten them, although such modified rule sets may not be reinserted into original tree.

9 Conclusion

We have presented main features of the Rough Set Exploration system (RSES) hoping that this paper will attract more attention to our software. Interested reader, who wants to learn more about RSES, may download the software and documentation form the Internet (cf. [40]).

RSES will further grow, as we intend to enrich it by adding newly developed methods and algorithms. We hope that many researchers will find RSES an useful tool for experimenting with data, in particular using methods related to rough sets.

Acknowledgement

We would like to express our gratitude to all the current and previous members and supporters of RSES development team, in particular the creators of Rosetta [12, 42] – Aleksander Øhrn and Jan Komorowski. Special tribute should be paid to Professor Andrzej Skowron, who has for many years supervised this project.

Over the years development of our software was significantly supported by several national and international research grants. Currently RSES is supported by the grant 3T11C00226 from Polish Ministry of Scientific Research and Information Technology.

References

1. Alpigini J. J.,Peters J. F.,Skowron A.,Zhong N.(Eds.),Proceedings of 3rd Int. Conf. on Rough Sets and Current Trends in Computing (RSCTC2002), Malvern, PA, 14–16 Oct. 2002. LNAI 2475, Springer-Verlag, Berlin, 2002
2. Bazan J., A Comparison of Dynamic and non-Dynamic Rough Set Methods for Extracting Laws from Decision Tables. In: Skowron A., Polkowski L.(ed.), Rough Sets in Knowledge Discovery 1, Physica-Verlag, Heidelberg, 1998, pp. 321–365
3. Bazan J., Nguyen H. S., Nguyen S. H., Synak P., and Wróblewski J., Rough set algorithms in classification problem. In: Polkowski L., Tsumoto S., Lin T.Y. (eds.), Rough Set Methods and Applications, Physica-Verlag, Heidelberg, 2000, pp. 49–88.

4. Bazan J., Nguyen H. S., Nguyen T. T., Skowron A., Stepaniuk J., Decision rules synthesis for object classification. In: E. Orłowska (ed.), Incomplete Information: Rough Set Analysis, Physica - Verlag, Heidelberg, 1998, pp. 23-57.
5. Bazan J., Skowron A., Ślęzak D. Wróblewski J., Searching for the Complex Decision Reducts: The Case Study of the Survival Analysis. In: Zhong N. et al. (eds.), Proceedings of ISMIS 2003, LNAI 2871, Springer-Verlag, Berlin, 2003, pp. 160—168,
6. Domingos P., Unifying Instance-Based and Rule-Based Induction. Machine Learning, Vol. 24(2), 1996, pp. 141–168.
7. Duda R.O., Hart P.E., Pattern Classification and Scene Analysis. Wiley, New York, 1973.
8. Góra G., Wojna A.G., RIONA: a New Classification System Combining Rule Induction and Instance-Based Learning. Fundamenta Informaticae, Vol. 51(4), 2002, pp. 369–390.
9. Grzymała-Busse, J., A New Version of the Rule Induction System LERS. Fundamenta Informaticae, Vol. 31(1), 1997, pp. 27–39
10. Grzymała-Busse J., Hu M., A comparison of several approaches to missing attribute values in data mining. Proceedings of the Second International Conference on Rough Sets and Current Trends in Computing RSCTC'2000, LNAI 2005, Springer-Verlag, Berlin, 2000, pp. 378–385.
11. Hippe, M.: Towards the classification of musical works: A rough set approach. In: [1], 546-553
12. Komorowski J., Øhrn A., Skowron A., ROSETTA Rough Sets. In: Kloesgen W., Zytkow J. (eds.), Handbook of KDD, Oxford University Press, Oxford, 2002, pp. 554–559.
13. Komorowski J., Pawlak Z., Polkowski L., Skowron A., Rough Sets: A tutorial. In: Pal S.K., Skowron A., Rough Fuzzy Hybridization, Springer Verlag, Singapore, 1999, pp. 3–98
14. Kostek B., Szczuko P., Zwan P., Processing of musical data employing rough sets and artificial neural networks. In: Tsumoto, S., Slowinski, R., Komorowski, J., Grzymala-Busse, J.W.(Eds.), Rough Sets and Current Trends in Computing Lecture Notes in Artificial Intelligence, Vol. 3066, Springer Verlag, Berlin, 2004, pp. 539-548
15. Lazareck L., Ramanna S., Classification of swallowing sound signals: A rough set approach. In: Tsumoto, S., Slowinski, R., Komorowski, J., Grzymala-Busse, J.W.(Eds.), Rough Sets and Current Trends in Computing *Lecture Notes in Artificial Intelligence*, **3066** (2004) 679-684
16. Nguyen Sinh Hoa, Data regularity analysis and applications in data mining. Ph. D. Thesis, Department of Math., Comp. Sci. and Mechanics, Warsaw University, Warsaw, 1999
17. Nguyen S. H., Nguyen H. S., Discretization Methods in Data Mining. In: Skowron A., Polkowski L.(ed.), Rough Sets in Knowledge Discovery 1, Physica Verlag, Heidelberg, 1998, pp. 451–482
18. Nguyen S. H., Skowron A., Synak P., Discovery of data patterns with applications to decomposition and classfification problems. In: L. Polkowski and A. Skowron (eds.), Rough Sets in Knowledge Discovery 2, Physica-Verlag, Heidelberg, 1998, pp. 55–97.
19. Michie D., Spiegelhalter D. J., Taylor C. C., Machine Learning, Neural and Statistical Classification. Ellis Horwood, London, 1994
20. Pawlak Z., Rough sets. International J. Comp. Inform. Science, Vol. 11, 1982, pp. 341–356
21. Pawlak Z., Rough sets and decision tables. Lecture Notes in Computer Science, Vol. 208, Springer Verlag, Berlin, 1985, pp. 186–196

22. Pawlak Z., On rough dependency of attributes in information systems. Bulletin of the Polish Academy of Sciences, Vol. 33, 1985, pp. 551–599
23. Pawlak Z., On decision tables. Bulletin of the Polish Academy of Sciences, Vol. 34, 1986, pp. 553–572
24. Pawlak Z., Decision tables – a rough set approach. Bulletin of EATCS, Vol. 33, 1987, pp. 85–96
25. Pawlak Z., Skowron A., Rough membership functions. In: Yager R., et al.(Eds.), Advances in Dempster Shafer Theory of Evidence, Wiley, N.Y., 1994, pp. 251–271
26. Pawlak Z., In pursuit of patterns in data reasoning from data – the rough set way. In: [1], 2002, pp. 1–9
27. Pawlak, Z.: Rough sets and decision algorithms. In: [39], 2001, pp. 30–45
28. Pawlak Z., Rough Sets: Theoretical Aspects of Reasoning about Data. Kluwer, Dordrecht, 1991
29. Peters J.F., Ramanna S., Towards a software change classification system. Software Quality Journal, Vol. 11(2), 2003, pp. 121–148
30. Skowron A., Rauszer C., The discernibility matrices and functions in information systems. In: Slowinski R., (Ed.),Intelligent Decision Support: Handbook of Applications and Advances in Rough Set Theory, Kluwer Academic Publishers, Dordrecht, 1992, pp. 259–300
31. Skowron A., Polkowski L., Synthesis of decision systems from data tables. In: Lin T.Y, Cercone N. (Eds.), Rough Sets and Data Mining: Analysis for Imprecise Data, Kluwer Academic Publishers, Dordrecht, 1997, pp.331–362
32. Skowron A., Rough Sets in KDD (plenary talk). 16-th World Computer Congress (IFIP'2000). In: Zhongzhi Shi, Boi Faltings, Mark Musen (eds.) Proceedings of Conference on Intelligent Information Processing (IIP2000), Publishing House of Electronic Industry, Beijing, 2000, pp. 1–17.
33. Ślęzak D., Wróblewski J., Classification Algorithms Based on Linear Combinations of Features. Proceedings of PKDD'99, LNAI 1704, Springer-Verlag, Berlin, 1999, pp. 548–553.
34. Valdés J.J, Barton A.J, Gene Discovery in Leukemia Revisited: A Computational Intelligence Perspective. In: R. Orchard et al. (eds.), Proceedings of IEA/AIE 2004, LNAI 3029, Springer-Verlag, Berlin, 2004, pp. 118–127
35. Wojna A.G., Center-Based Indexing in Vector and Metric Spaces. Fundamenta Informaticae, Vol. 56(3), 2003, pp. 285-310.
36. Wojnarski M., LTF-C: Architecture, Training Algorithm and Applications of New Neural Classifier. Fundamenta Informaticae, Vol. 54(1), 2003, pp. 89–105
37. Wróblewski J., Genetic algorithms in decomposition and classification problem. In: Skowron A., Polkowski L.(ed.), Rough Sets in Knowledge Discovery 1, Physica Verlag, Heidelberg, 1998, pp. 471–487
38. Wróblewski J., Covering with Reducts - A Fast Algorithm for Rule Generation. Proceeding of RSCTC'98, LNAI 1424, Springer Verlag, Berlin, 1998, pp. 402-407
39. Ziarko, W., Yao, Y.,Eds.:Proceedings of 2nd Int. Conf. on Rough Sets and Current Trends in Computing (RSCTC2000), Banff, Canada, 16-19 Oct. 2000. *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin (2001)
40. The RSES Homepage, `http://logic.mimuw.edu.pl/∼rses`
41. Report from EUNITE World competition in domain of Intelligent Technologies, `http://www.eunite.org/eunite/events/eunite2002/competitionreport2002.htm`
42. The ROSETTA Homepage, `http://rosetta.lcb.uu.se/general/`
43. The WEKA Homepage, `http://www.cs.waikato.ac.nz/∼ml`