

RSES 2.2 User's Guide

Warsaw University
<http://logic.mimuw.edu.pl/~rses>

January 19, 2005

Contents

1	Introduction to RSES	5
1.1	The history of RSES creation	5
1.2	Aims and capabilities of RSES	7
1.3	Technical requirements and installation of RSES	7
1.3.1	Installation in MS Windows	9
1.3.2	Instalation in Linux	9
2	Using RSES	11
2.1	Managing projects	11
2.2	Objects	14
2.3	Main system menu	16
2.4	Toolbar	18
2.5	Context menu	19
2.6	Status and progress of calculation	21
3	Objects in project - quick reference	23
3.1	Tables	23
3.2	Reduct sets	32
3.3	Rule sets	34
3.4	Cut sets	37
3.5	Linear combinations	39
3.6	Decomposition trees	41
3.7	LTF Classifiers	42
3.8	Results	46
4	Main data analysis methods in RSES	49
4.1	Missing value completion	49
4.2	Cuts, discretization and grouping	50
4.3	Linear combinations	51
4.4	Reducts and decision rules	52
4.5	Data decomposition	58

4.6	k-NN classifiers	60
4.7	LTF Classifier	63
4.8	Cross-validation method	65
5	RSES scenario examples	69
5.1	Train-and-test scenarios	69
5.1.1	Rule based classifier	69
5.1.2	Rule based classifier with discretization	70
5.1.3	Decomposition tree	72
5.1.4	k-NN classifier	74
5.1.5	LTF Classifier	75
5.2	Testing with use of cross-validation	77
5.3	Scenario for decision generation	78
A	Selected RSES 2.2 file formats	81
A.1	Data sets	81
A.2	Reduct sets	83
A.3	Rule sets	84
A.4	Set of cuts	86
A.5	Linear combinations	89
A.6	LTF-C	90
A.7	Classification results	91
	Bibliography	93
	Index	98

Chapter 1

Introduction to RSES

RSES 2.2 - *Rough Set Exploration System* 2.2 is a software tool that provides the means for analysis of tabular data sets with use of various methods, in particular those based on Rough Set Theory (see [22]).

The RSES system was created by the research team supervised by Professor Andrzej Skowron. Currently, the RSES R&D team consists of: Jan Bazan (University of Rzeszów), Rafał Latkowski (Warsaw University), Michał Mikołajczyk (Warsaw University), Nguyen Hung Son (Warsaw University), Nguyen Sinh Hoa (Polish-Japanese Institute of Information Technology), Andrzej Skowron (Warsaw University), Dominik Ślęzak (University of Regina and Polish-Japanese Institute of Information Technology), Piotr Synak (Polish-Japanese Institute of Information Technology), Marcin Szczuka (Warsaw University), Arkadiusz Wojna (Warsaw University), Marcin Wojnarski (Warsaw University), Jakub Wróblewski (Polish-Japanese Institute of Information Technology).

The RSES system is freely available (for non commercial use) on the Internet. The software and information about it can be downloaded from:

<http://logic.mimuw.edu.pl/~rses>

1.1 The history of RSES creation

Back in 1993, as a project accompanying master theses of Krzysztof Przyłucki and Joanna Słupek (supervised by Andrzej Skowron at Warsaw University) the piece of software named *decision table analyzer* has been created. This early system was written in C++ for Windows 3.11 platform. The creation of this software was also supported by: Jan Bazan, Tadeusz Gąsior, and Piotr Synak.

In the next year (1994) first version of RSES (version 1.0) was created. Written in C++, for HP-UX (a Unix flavor) was only available for Apollo workstations by Hewlett Packard. Version 1.0 of the RSES system was developed by: Jan Bazan, Agnieszka Chądzyńska, Nguyen Hung Son, Nguyen Sinh Hoa, Adam Cykier, Andrzej Skowron, Piotr Synak, Marcin Szczuka, and Jakub Wróblewski.

In 1996 the RSES-lib 1.0 library of computational methods was put together. Written in C++ it was running on both Unix and Microsoft Windows platforms. The RSES-lib 1.0 library was used as a part of computational kernel of ROSETTA (*Rough Set Toolkit for Analysis of Data*). The ROSETTA system was developed between 1996 and 1998, as a result of cooperation between Warsaw University and Norwegian University of Science and Technology (NTNU) in Trondheim. The ROSETTA system was initially developed for Microsoft Windows 9x/NT (see, e.g. [21]). The creators of RSES-lib 1.0 include: Jan Bazan, Nguyen Hung Son, Nguyen Sinh Hoa, Adam Cykier, Andrzej Skowron, Piotr Synak, Marcin Szczuka, and Jakub Wróblewski. The ROSETTA system owes its concept and initial development to Jan Komorowski and Alexander Ørn (both of Knowledge Systems Group/NTNU at that time). The ROSETTA system is still in use today, for details refer to [26].

The next version of RSES-lib, i.e., RSES-lib 2.0 was created in 1998 and 1999, mostly to satisfy the demand for newer and more versatile tool to be used in computations done for the research project ESPRIT-CRIT2 (funded by European Commission). One of CRIT2 sub-projects devoted to data analysis and knowledge discovery was realized at the Group of Logic, Warsaw University under the supervision of Professor Andrzej Skowron. The work on creation of RSES-lib 2.0 was done by: Jan Bazan, Nguyen Hung Son, Nguyen Sinh Hoa, Andrzej Skowron, Piotr Synak, Marcin Szczuka, and Jakub Wróblewski.

In 2000 a new version of RSES emerged. This time it was equipped with Graphical User Interface (GUI) for Microsoft Windows 9x/NT/2000/Me. The system was written in C++ and used RSES-lib 2.0 as its computational backbone. As this version of RSES was the first to be equipped with Microsoft Windows GUI, it was named version 1.0. The developers of this version include: Jan Bazan, Nguyen Hung Son, Nguyen Sinh Hoa, Andrzej Skowron, Piotr Synak, Marcin Szczuka, and Jakub Wróblewski.

The year of 2002 brought the next major version of RSES (version 2.0), significantly different from the previous ones. This time the system was written in Java, although some parts of the computational kernel still use elements of RSES-lib 2.0 (written in C++). The C++ part of computational kernel has been partly re-written in order to comply to the standards of GCC

compiler (GNU C++). In this way, by using Java and GCC, the portability of the system was achieved. Starting with this version (2.0) the RSES system is distributed for both Microsoft Windows 9x/NT/2000/Me/XP and Linux/i386.

One year after RSES 2.0 the next version – RSES 2.1 was put together. It features newer, improved, more versatile and more user friendly GUI as well as several new computational methods.

1.2 Aims and capabilities of RSES

The main aim of RSES is to provide a tool for performing experiments on tabular data sets.

In general, the RSES system offers the following capabilities:

- import of data from text files,
- visualization and pre-processing of data including, among others, methods for discretization and missing value completion,
- construction and application of classifiers for both smaller and vast data sets, together with methods for classifier evaluation.

The RSES system is a software tool with an easy-to-use interface, at the same time featuring a bunch of methods that make it possible to perform compound, non-trivial experiments in data exploration with use of Rough Set methods.

1.3 Technical requirements and installation of RSES

In order to run RSES we recommend at least:

- CPU - Pentium 200 MHz;
- 128 MB RAM;
- Java Virtual Machine version 1.4.1;
- Operating system MS Windows 9x/NT/2000/Me/XP or Linux/i386 (kernel 2.2 or newer).

Majority of RSES is written in Java. Therefore, in order to make it running, an appropriate version of JVM (Java Virtual Machine) is required. Moreover, as part of computation is done by methods from RSES-lib 2.0 (written in C++, see 1.1), the compiled version of this code is distributed with the installation bundle as an executable file named `RSES.kernel`. Please note that this part of RSES is platform-dependant. The figure 1.3 illustrates the general architecture of RSES. Notice, that Java part of RSES comprises of two logical sub-parts, the RSES 2.2 GUI and the computational kernel powered by the RSES-lib 3.0 library. The GUI part is responsible for interaction with user while the RSES-lib 3.0 serves as computational engine and a wrapper for RSES-lib 2.0 computational routines.

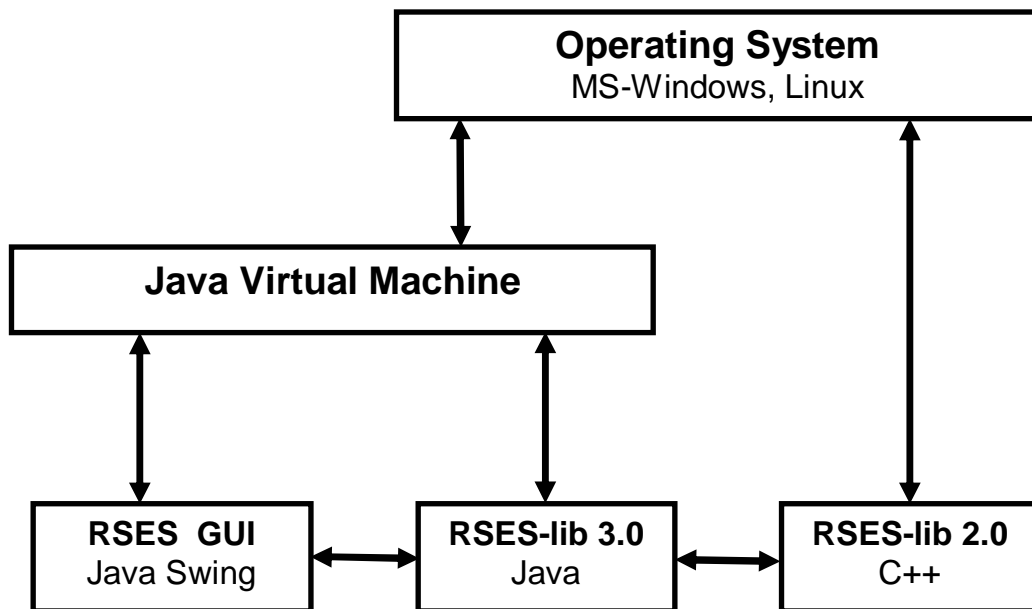


Figure 1.1: RSES internal architecture.

On the RSES Homepage <http://logic.mimuw.edu.pl/~rses> one will find the installation bundles for both MS Windows and Linux/i386. Those bundles contain the RSES executables as well as several demonstration data sets. These data sets are provided in order to help user in starting to work with RSES without necessity of preparing the data in advance. We do hope that the example data sets provided in installation bundles will make the first steps in RSES easier for the user and help him/her in preparing his/her own data for experiments.

1.3.1 Installation in MS Windows

MS Windows who already have the Java Virtual Machine installed can directly download and run the RSES installation bundle. If JVM is not present or is in version older than 1.4, we recommend that it is installed prior to RSES installation. The current version of Java (either SDK or JRE) may be obtained at no cost from SUN Microsystems website <http://java.sun.com>.

After downloading the RSES installation bundle, which is a single executable file, it is enough to double-click on it in order to launch the installer program. The installation is performed according to standard MS Windows procedure. During the installation process the user may choose the installation directory and some other basic installation parameters. In case of confusion during the installation process we recommend to accept the default values proposed by installer.

1.3.2 Instalation in Linux

Please note that the RSES installation bundle for Linux contains, as an executable binary file, the part of computational kernel that is written in C++ and statically build. This binary file was created with GCC ver. 2.95 for machines running Linux kernel 2.2 or newer for i386 architecture.¹

In order to run RSES on Linux the Java Virtual Machine version 1.4.1 or newer has to be installed. If JVM is not present or is in version older than 1.4, we recommend that it is installed prior to RSES installation. The current version of Java for Linux (either SDK or JRE) may be obtained at no cost from SUN Microsystems website <http://java.sun.com>. The RSES installation bundle for Linux is provided as a single `.tgz` file containing tarred and compressed files. In order to install it the user has to unpack the files into the directory of choice, preserving the structure and names of directories from the `.tgz` file.

The easiest way to unpack RSES properly, inside the directory of choice, is by executing from command line:

```
tar -xvzf rses_22.tgz
```

It is important to check after unpacking if the files `startRSES` and `rses.kernel` have the attribute `Executable`. In this way we are ready to launch RSES (refer to the beginning of next chapter).

¹Tested with vanilla and customized kernels up to 2.6.7.

Chapter 2

Using RSES

To start RSES 2.2 on Microsoft Windows platform one have to open the menu Start/Programs/Rses2 and select Rough Set Exploration System ver. 2.2.

To start RSES 2.2 in Linux one have to open terminal window in the directory where the `Rses.jar` is located and execute the command:

```
java -jar Rses.jar
```

or use provided simple script to start RSES from command line by:

```
./startRSES
```

To stop the application one have to select from menu /File/Exit (Alt+fe) and confirm the decision to quit. Thanks to obligatory confirmation is quite impossible to close program by accident without saving the results of current work.

Once the program is launched the main RSES 2.2 window, consisting of main menu, toolbar and project workspace appears.

Main menu contains those most general options offered by RSES. Some of its functionality is also accessible via toolbar and context menus.

2.1 Managing projects

The user may work several experiments at the same time. The RSES system is capable of working with several projects at the same time . However, only one of the experiments may be active (perform computation) at any given moment.

At this point it is worth mentioning, that it is possible to run several RSES experiments at the same time in distributed environment. For this purpose

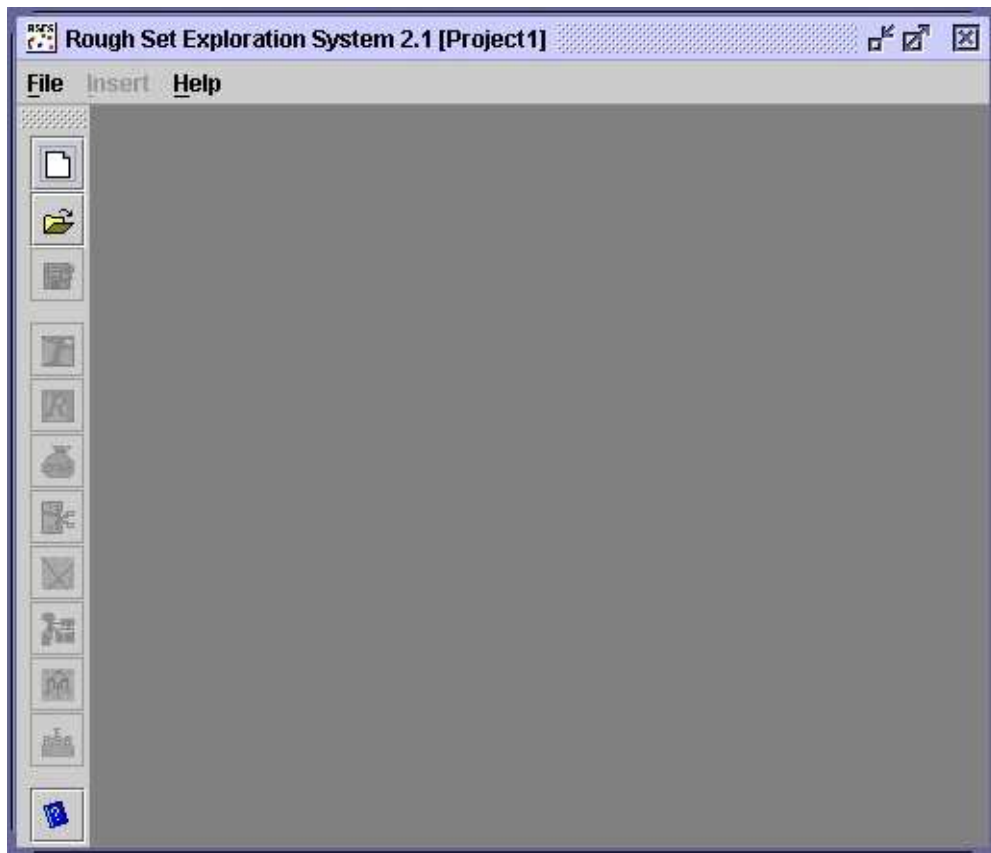


Figure 2.1: RSES 2.2 - main window just after start



Figure 2.2: Main Menu



Figure 2.3: Toolbar

a special, additional module named *Dixer - DIstributed eXEcutoR* has been developed. For the sake of convenience and versatility the Dixer subsystem is equipped with separate graphical user interface. The simplistic interface for Dixer was designed in order to provide an easy way of designing multiple

distributed experiments. (see <http://logic.mimuw.edu.pl/~rses>).

A new project in RSES may be created in one of three possible ways:

- by choosing from main menu /File/New project,
- using keyboard shortcut: Alt+fn,
- clicking on the first (from the top) toolbar button.

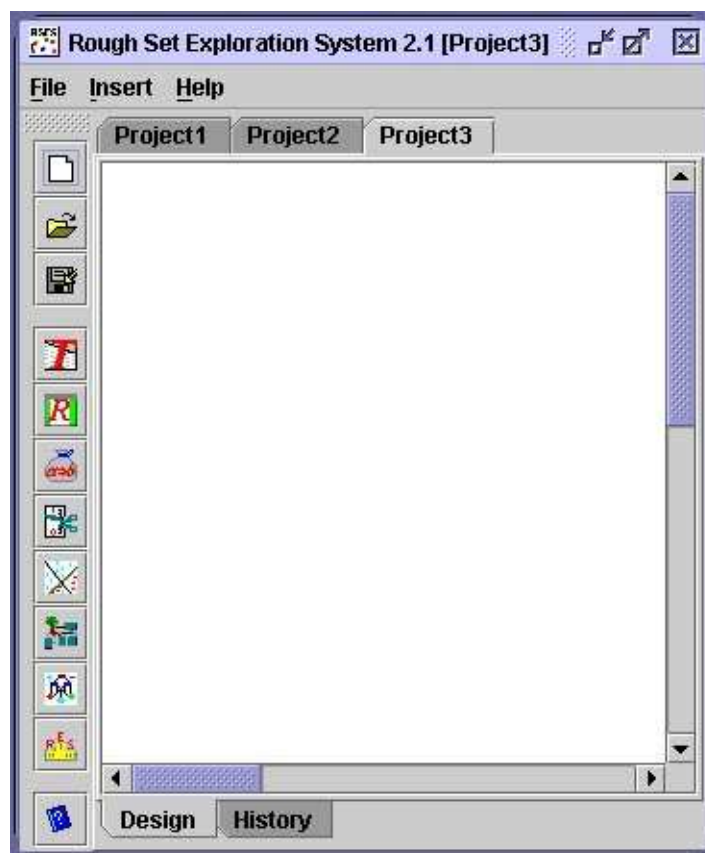


Figure 2.4: New project in RSES

Once new (empty) project has been created, we may place in it various objects such as: data tables, decision rule sets, result summaries etc. To find out more about possible objects in projects refer to section 2.2.

In the upper part of the window user can see tabs with names of currently active (lighter tab) and other (darker tabs) existing projects. By clicking on

tab the user can access his/her projects. This feature simplifies work when several projects are being developed.

In the lower part of each project workspace two tabs are present. These tabs are used to switch between two types of project views:

- Design view – standard view for working with project,
- History view – this view presents all registered events that happened during the operations on project.

At the bottom and on the right side of project window (while in Design view) the scrollbars are placed. They simplify work with large projects which are impossible to fit into a small window.

In /File menu user will find options for saving and restoring project to/from file. Detailed description of main menu options is given in section 2.3.

By right-clicking within project's workspace area (white area) the user invokes a context menu. With use of this context menu user can insert new objects (entities) into the project (see subsection 2.5).

2.2 Objects

Objects that can be placed in projects fell into following categories:

- Data Tables
- Reduct Sets
- Rule Sets
- Cut Sets
- Linear Combinations
- Decomposition Trees
- LTF-C (Local Transfer Function Classifiers)
- Classification Results (Experiments' effects)

To create an object we may (besides using the context menu) choose a corresponding option from menu (by using mouse or keyboard shortcut) or click the corresponding button on the toolbar. If we introduce new object

into project using context menu then the object is placed at the position of mouse cursor.¹, . In the case of object introduced with use of /Insert menu or toolbar the objects emerge in the central part of visible workspace. Each new object is shifted few pixels from the central point, so that occlusions does not occur when we introduce several objects at the same time.

Dependencies between objects within project are marked by connecting such object with arrows. For example, if we calculate decision rules for some data table then the arrow originating in table and pointing at set of rules appears.

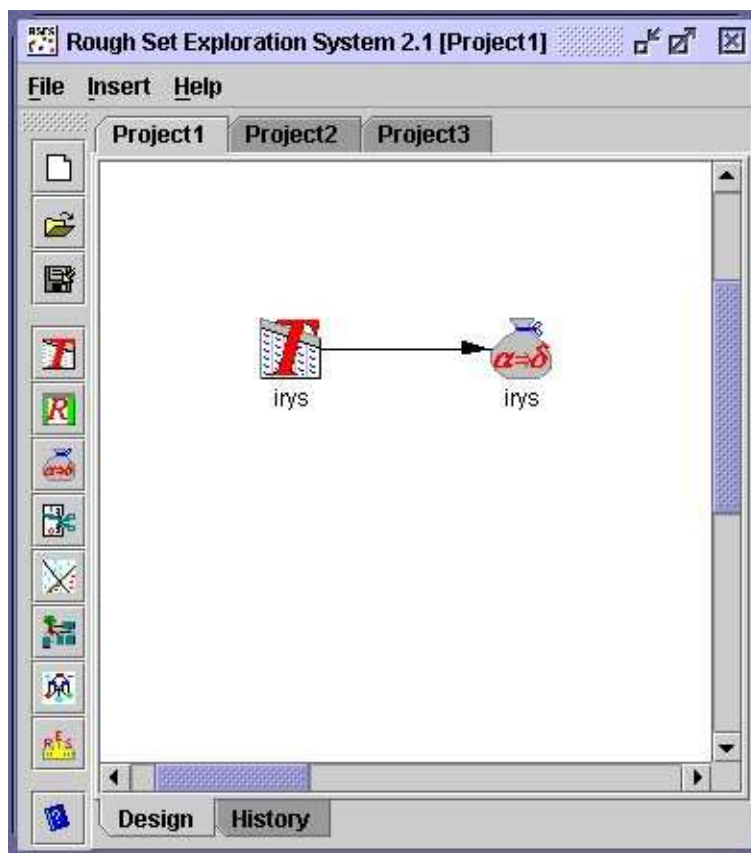


Figure 2.5: Two objects bound by dependence (table and rules).

To move object within project user has to select it with mouse click and then, holding the left mouse button, move mouse to desired position. Releasing mouse button causes the object to be placed at new position.

¹The context menu for main project window will be further referred to as *general menu*

User can also move several objects at the same time. To do so one has to mark several objects (using mouse) and then click and hold the left mouse button on one of selected objects. The group of objects is then moved and once the mouse button is released, the objects are placed at new position.

To select several objects in project it is enough to push left mouse button while mouse cursor is at the position within project space and then, still holding the mouse button, mark a rectangular area within project. Objects that are inside this rectangular area are instantly selected.

Another way of selecting/reselecting objects in groups is by clicking on an object while holding the **Ctrl** key. The object marked is this way becomes selected or deselected (if was previously selected) without change in the selection status of other project's objects.

Each object as well as group of objects have a corresponding context menu attached. With this menu it is possible to change name, duplicate, save, restore, remove, show components, and perform other operations (specific to object's type) for an object. Context menu is accessible by right-clicking on selected object. Detailed description of object's context menus is provided in section 3. Description of context menu for groups of objects is given in subsection 2.5.

Selected objects and object groups may be removed from the project using key. Each deletion, in order to avoid loss of data, has to be confirmed by clicking "OK" in corresponding dialog box.

2.3 Main system menu

Option from RSES 2.2 main menu may be applied by selecting it from drop-down menu with the use of mouse, or by using keyboard shortcut. Keyboard shortcuts are constructed according to the simple principle. The combination to be used is **Alt**+<letter>, where <letter> is the (only) underlined letter in the name of option we want to chose. By holding the **Alt** key and selecting subsequent option we may navigate through the entire menu structure. For instance by pressing **Alt**+<f>+<n> we create a new project (/File/New project).

Some of the menu options are accompanied with small icons. These icons directly correspond to buttons on the toolbar that have the same functionality. Such icons are also present in general (context) menu.

Short description of options in main menu is presented below.

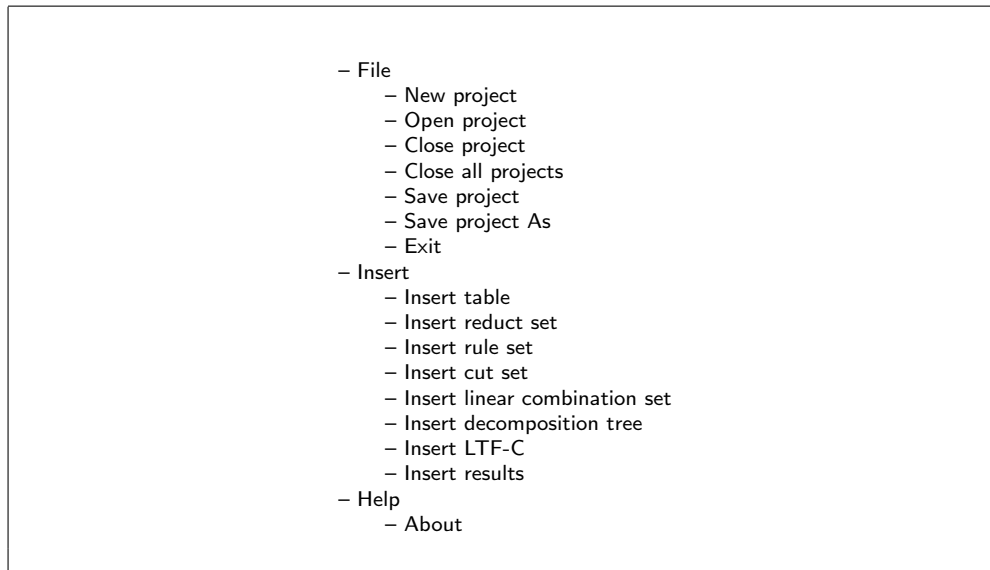


Figure 2.6: Layout of the main menu

- File – projects' management
 - New project – creates new project
 - Open project – restores previously saved project from the disk
 - Close project – closes active project
 - Close all projects – closes all currently open projects
 - Save project – saves active project to a file on disk
 - Save project As – saves active project to the specified file on disk
 - Exit – terminates RSES
- Insert – inserting new objects into active project
 - Insert table – inserts data table
 - Insert reduct set – inserts reduct set
 - Insert rule set – inserts rule set
 - Insert cut set – inserts cut and/or attribute partition set
 - Insert linear combination set – inserts a set of linear combinations
 - Insert decomposition tree – inserts decomposition tree

- Insert LTF-C – inserts LTF-C (Local Transfer Function Classifier)
- Insert results – inserts an object for viewing experiment results
- Help – help and information
 - About – basic information about RSES

Notice! All objects added to the project with use of menu or toolbar appear in the central part of project workspace, at the position that is randomly shifted by few pixels from the actual center. This is to avoid occlusion (when inserting several objects at the same time) and make selection of new object easier while working with mouse.

2.4 Toolbar

The toolbar contains buttons corresponding to selected options from main and general menus. In this way the RSES user have instant access to most common actions.

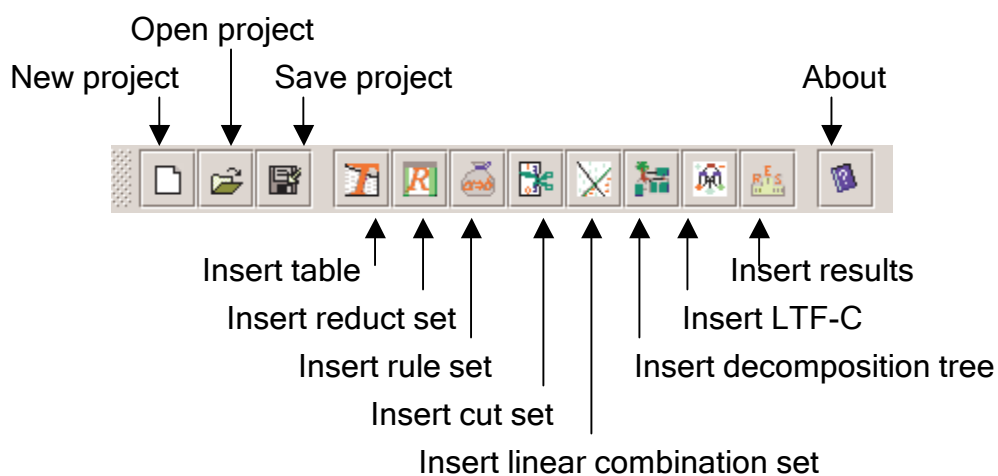


Figure 2.7: Toolbar and corresponding menu options

- New project – creates new project
- Open project – restores previously saved project from the disk
- Save project – saves active project to a file on disk

- Exit – terminates RSES
- Insert table – inserts data table
- Insert reduct set – inserts reduct set
- Insert rule set – inserts rule set
- Insert cut set – inserts cut and/or attribute partition set
- Insert linear combination set – inserts a set of linear combinations
- Insert decomposition tree – inserts decomposition tree
- Insert LTF-C – inserts LTF-C (Local Transfer Function Classifier)
- Insert results – inserts an object for viewing experiment results
- About – basic information about RSES

Notice! All objects added to the project with use of menu or toolbar appear in the central part of project workspace, at the position that is randomly shifted by few pixels from the actual center. This is to avoid occlusion (when inserting several objects at the same time) and make selection of new object easier while working with mouse.

2.5 Context menu

A context menu is associated with every objects (or group of objects) in project workspace. It can be accessed by right-clicking on the object.

The contents of context menu depend on the kind of object (objects). We briefly present the contents of context menus below, and discuss their options in more detail further in this manual.

Along with the context menu for particular object(s) there exists the context menu for the entire project space. We call it *general menu*. To access general menu user have to right-click on empty area within the project workplace (white area).

Options from general menu are also accessible from main program menu and the toolbar.

List of options in general menu:

- Insert table – inserts data table
- Insert reduct set – inserts reduct set

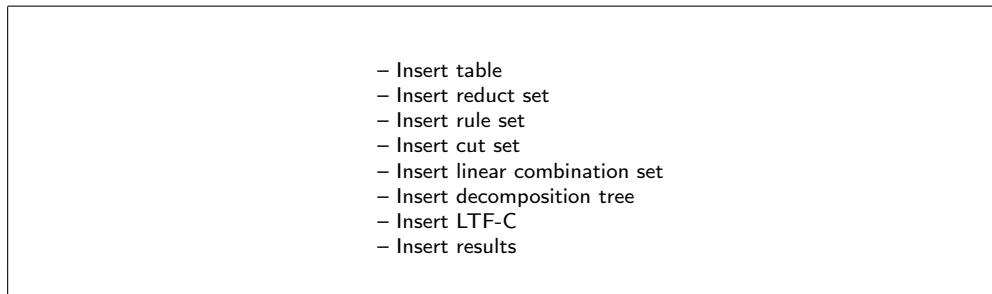


Figure 2.8: Layout of general menu (context menu for the project)

- Insert rule set – inserts rule set
- Insert cut set – inserts cut and/or attribute partition set
- Insert linear combination set – inserts a set of linear combinations
- Insert decomposition tree – inserts decomposition tree
- Insert LTF-C – inserts LTF-C (Local Transfer Function Classifier)
- Insert results – inserts an object for viewing experiment results

Context menu for a group of (selected) objects can be accessed by clicking on any of selected objects with right mouse button.

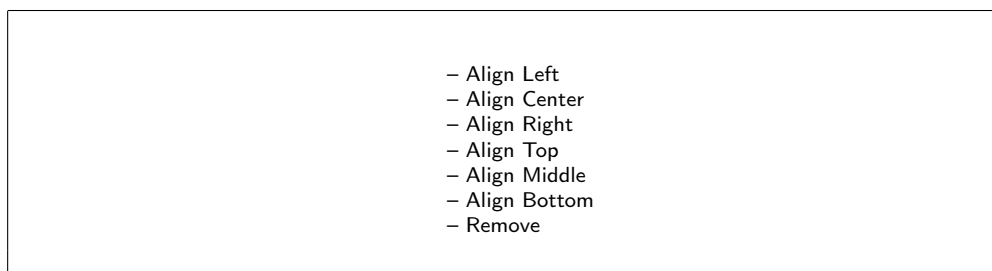


Figure 2.9: Layout of context menu for a group of objects

Options in context menu for a group of objects:

- Align Left – align selected objects horizontally to the leftmost object
- Align Center – center selected objects horizontally

- **Align Right** – align selected objects horizontally to the rightmost object
- **Align Top** – align selected objects vertically to the one on the top
- **Align Middle** – center selected objects vertically
- **Align Bottom** – align selected objects vertically to the one at the bottom
- **Remove** – removes all selected objects

Note, that using **Align Left**, **Align Center**, **Align Right**, **Align Top**, **Align Middle** and **Align Bottom** causes selected objects to appear in one (horizontal or vertical) line. As a result some objects may be occluded by others.

2.6 Status and progress of calculation

The user has an access to information about the track of operations performed in the project. This information is collected in history view that is reachable via **History** tab at the bottom of project workspace. The history view stores information about operations on objects, performed calculations, errors, and calculation terminations.

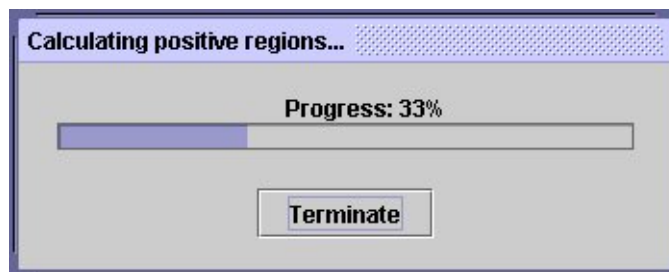


Figure 2.10: Computation progress control

During each calculation new control window appears. In this window the user can see the advancement of current calculation shown with use of progress bar. The user can instantly terminate currently running computation by clicking **Terminate** button. After such termination a dialog box with information appears, and the fact of termination is logged in project's history.

Chapter 3

Objects in project - quick reference

In this chapter we present a quick review of objects appearing in RSES projects together with short description of operations that may be performed on them. More detailed description of algorithms and their options are presented in chapter 4.

3.1 Tables

Tables are the most important entities in project. They represent data tables (tabular data sets) and allow for their examination, edition, and launching computations on data.



Figure 3.1: Icon representing decision table

The user can view the data contained in the table by double clicking on it or by selecting **View** from table object's context menu.

The context menu for table contains the following options:

- **Load** – load data from file into table object. File is in one of formats: RSES, RSES 1.0, Rosetta, and Weka.
- **Save As** – save data to file in RSES format.
- **View** – view the contents of table (see figure 3.3). The user may scroll, and rearrange the view window.

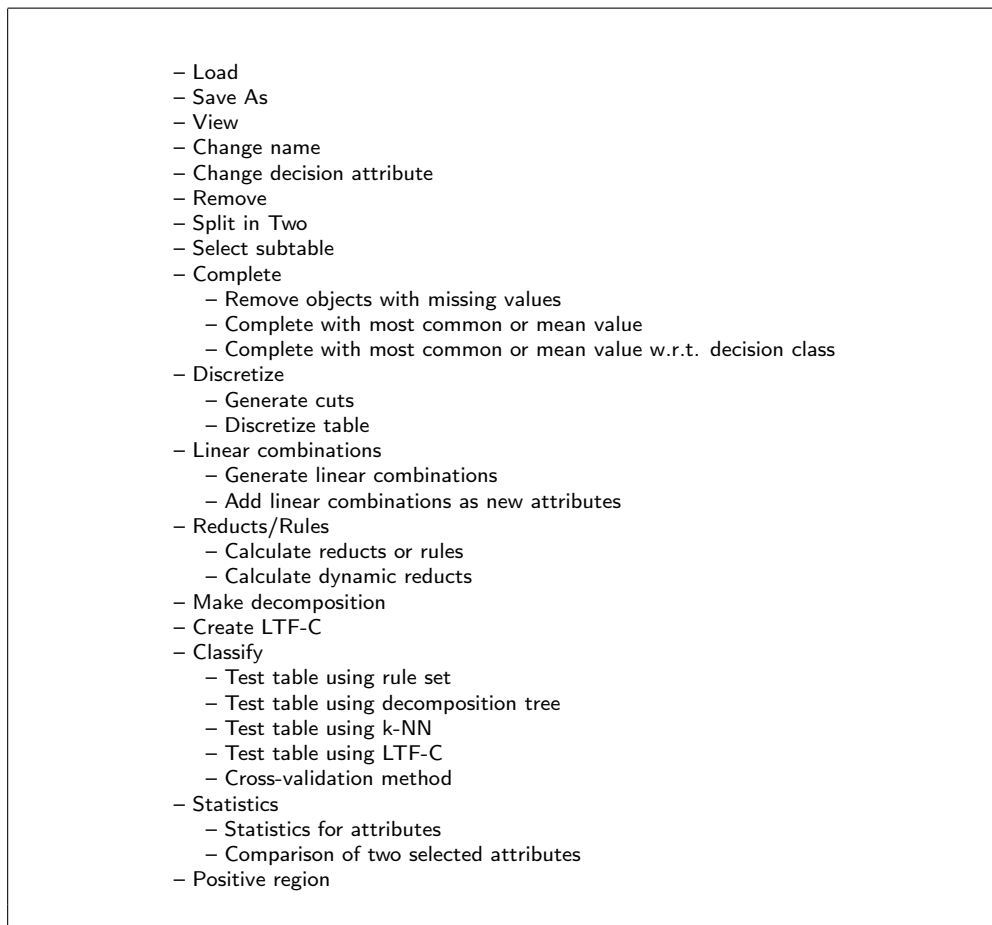
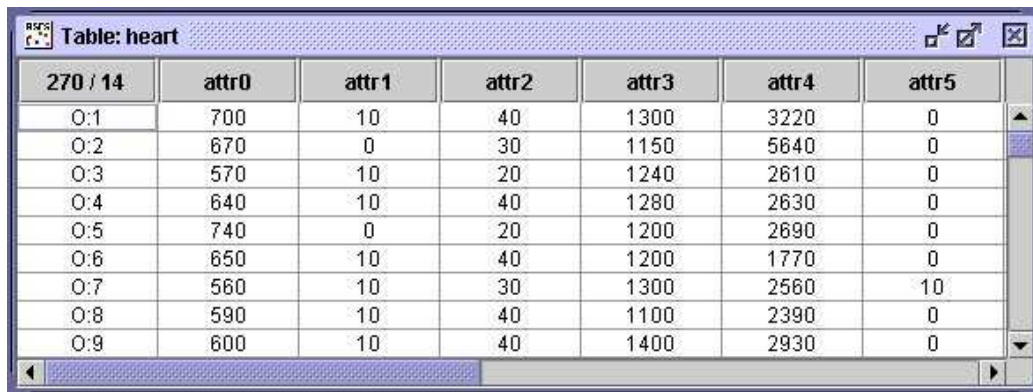


Figure 3.2: Layout of context menu for data table object

- **Change name** – change the table name (see figure 3.4). This name is saved together with data. Table name does not have to be identical with the name of file used to store the table on disk. Table name can also be altered by double-clicking on table name appearing below the icon.
- **Change decision attribute** – selecting the decision attribute. Selected attribute is moved to the end of table (becomes the last attribute).
- **Remove** – removes table (after separate confirmation).
- **Split in Two** – randomly splits table into two disjoint subtables (see figure 3.5).



270 / 14	attr0	attr1	attr2	attr3	attr4	attr5
O:1	700	10	40	1300	3220	0
O:2	670	0	30	1150	5640	0
O:3	570	10	20	1240	2610	0
O:4	640	10	40	1280	2630	0
O:5	740	0	20	1200	2690	0
O:6	650	10	40	1200	1770	0
O:7	560	10	30	1300	2560	10
O:8	590	10	40	1100	2390	0
O:9	600	10	40	1400	2930	0

Figure 3.3: View of data table contents



Change name

New name:

heart

OK Cancel

Figure 3.4: Changing table name

The **Split factor** parameter between 0.0 and 1.0 – specified by user – determines the size of first subtable. The other subtable is a complement of the first. For instance, setting **Split factor** to be 0.45 means that the first subtable will contain 45% of objects from original table, while the other will contain the remaining 55%. The new table objects in project, resulting from split operation, are automatically assigned names composed of original table's name and the value of **Split factor** (and it's complement to 1 - see figure 3.5).

- **Select subtable** – creation of subtable (see figure 3.6) by selection of attribute subset. In case of all attributes being selected, a copy of original table is created.
- **Complete** – completion of missing data (see subsection 4.1). The user

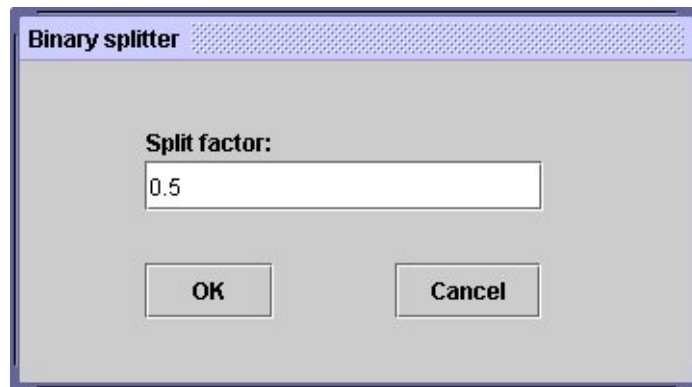


Figure 3.5: Splitting table in two

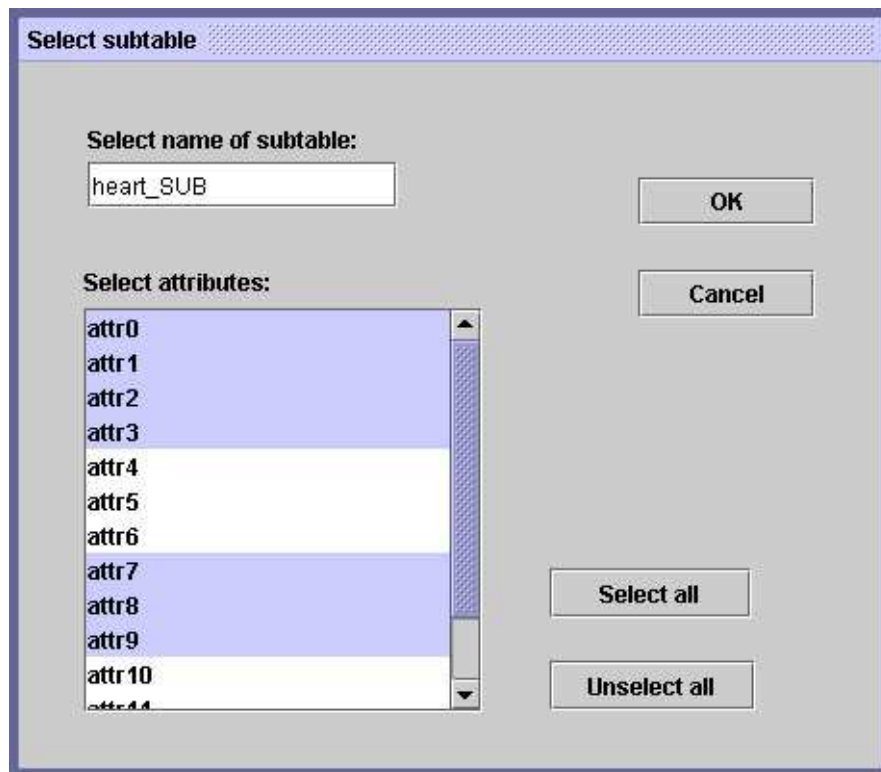


Figure 3.6: Selecting subtable

provides the name for new table which is created by selected algorithm on the basis of original one. This new table contains no missing values.

- **Discretize** – data discretization (cf. [5]) and generation of cuts for attributes (refer to subsection 4.2).
- **Linear combinations** – generation of new attributes in the table. New attributes are generated as linear combinations of existing ones (see subsection 4.3)
- **Reducts/Rules** – reducts, dynamic reduct and decision rules from the data (refer to subsection 4.4).
- **Make decomposition** – this option initiates calculation of the decomposition tree for data table (refer to subsection 4.5).
- **Create LTF-C** – creates LTF-C (*Local Transfer Function Classifier*) based on artificial neural network (see subsection 4.7).
- **Classify** – launches the classification with use of selected classifier. The classifier have to be constructed and trained prior to it's use. Regardless of the classifier chosen, the user can use one of two classification modes:
 - **Generate confusion matrix** – calculates a matrix that store summarized classification error;
 - **Classify new cases** – classifies new cases (with no known decision) and stores the result as a new (decision) column in the table.

To select classification mode user has to select appropriate option in **General test mode** field. This choice is available for all classification methods.

The user have choice of the following classification methods:

- **Test table using rule set** – classification with use of decision rules (see subsection 4.4);
- **Test table using decomposition tree** – classification with use of decomposition tree (see subsection 4.5);
- **Test table using k-NN** – classification of selected table with use of Nearest Neighbors method (see subsection 4.6);
- **Test table using LTF-C** – classification with use of LTF-C (see subsection 4.7);
- **Cross-validation method** – classification with use of cross-validation method applied to any of the classifiers mentioned above (see subsection 4.8).

- Statistics – displays basic information about table and attributes.
 - Statistics for attributes opens the window as shown in figure 3.7. Clicking **Show chart** button results in graphical display of selected attribute value distribution in the form of bar chart (see figure 3.8). For symbolic attributes occurrences of all attribute values are counted and presented as histogram. In case of numerical (continuous) attributes the attribute value space is divided into several equal intervals and the histogram for these intervals is displayed. The number of intervals is selected by the user by using **Number of intervals** option. As the user may be interested only in some part of attribute value space, it is possible to alter the attribute range by clicking on **Change limits** button and inputting desired values (limits). The graphs with attribute distributions may be exported to an external file. Currently, RSES 2.2 allows to save them in PNG, JPG and HTML format. To export a graph user has to click on **Export** button in graph window (see figure 3.8).

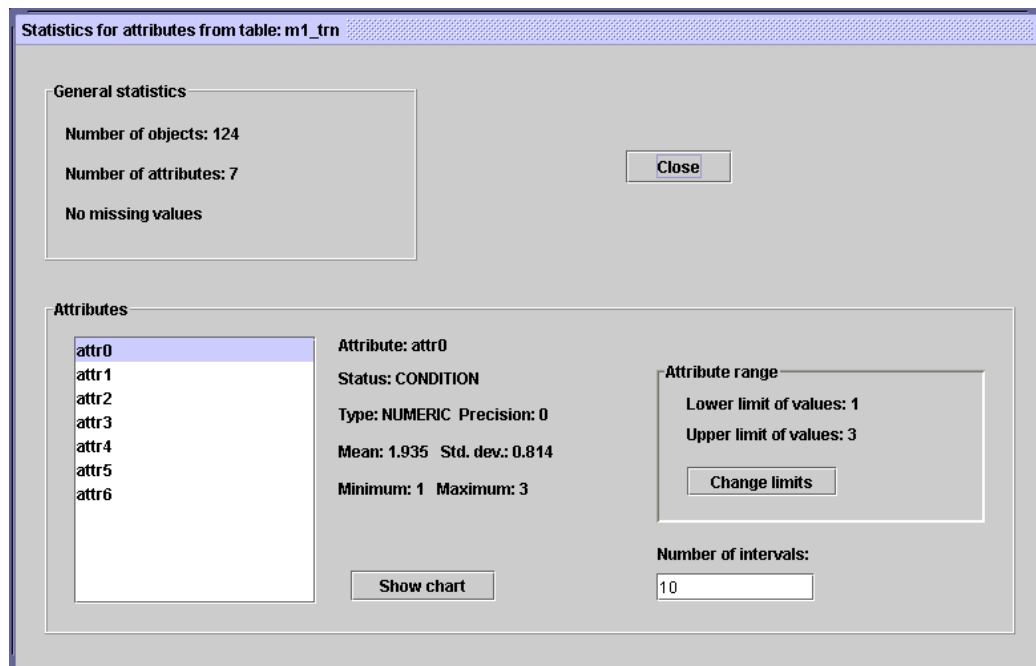


Figure 3.7: Information on data table

- Comparison of two selected attributes opens a dialog window that permits the user to choose two attributes to be compared (see

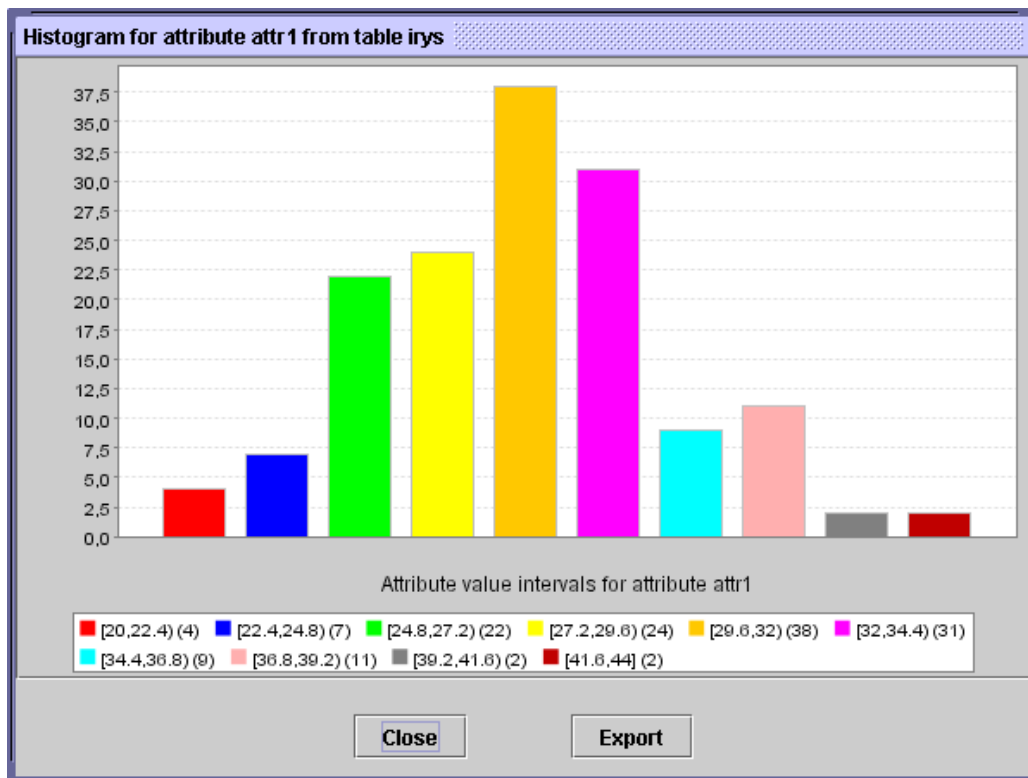


Figure 3.8: Attribute distribution presented as graph

figure 3.9). First, the user selects the types of attributes to be compared. This can be done by marking one of: **Numeric and numeric**, **Symbolic and symbolic** or **Symbolic and numeric**. Once types are chosen, the system filters attributes with respect to their type and places in two columns (see figure 3.9). The user has to select one attribute in each column and hit the **Show chart** button. Depending of the types of attributes chosen (symbolic/numeric), different plots may be produced as a result. In case of two numeric attributes being compared the result is a scatter plot as shown in figure 3.10. Each of axes on the plot correspond to one of compared attributes. Comparison of two symbolic attributes results in bar chart as shown in figure 3.11 whereas comparison of symbolic and numeric attribute is a scatter plot as in figure 3.12.

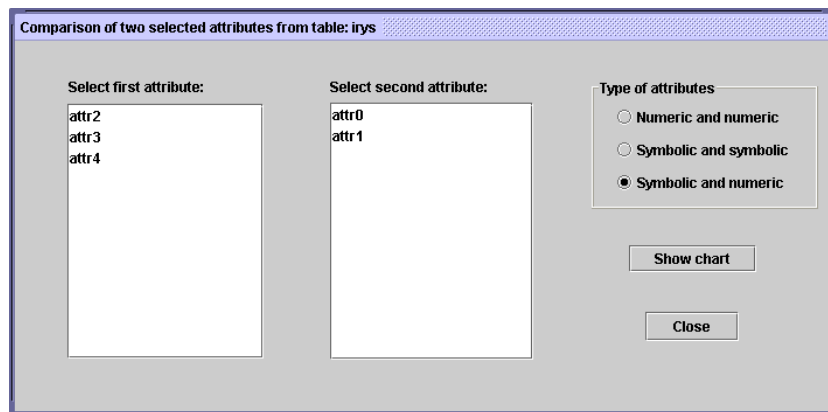


Figure 3.9: Settings for comparison of two attributes

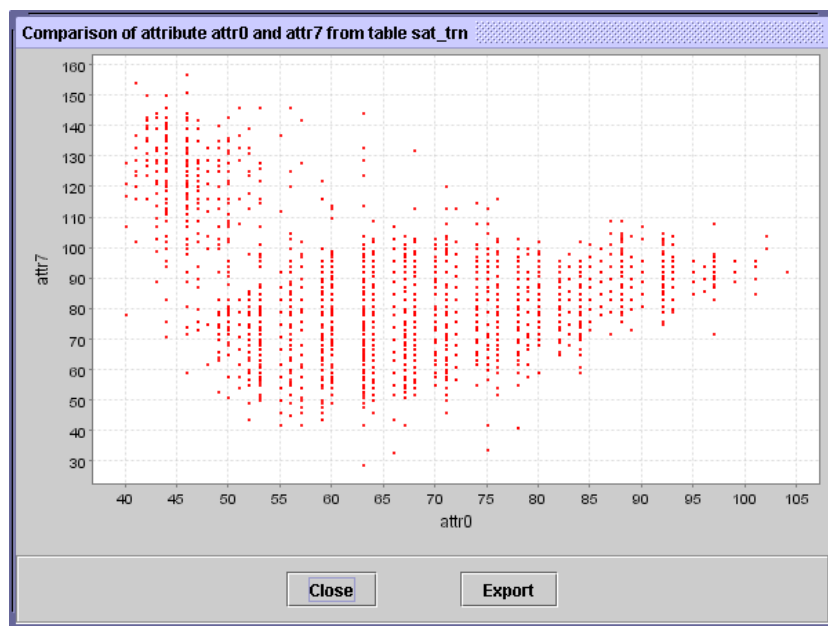


Figure 3.10: Distribution of two numeric attributes presented as graph.

- Positive region – calculates positive region for the table (*Notice: last column in the table is assumed to be decision attribute. The precedence of columns cannot be changed in the table view. It is, however, possible to move a column to the last position using the option Change decision attribute from table's context menu.*)

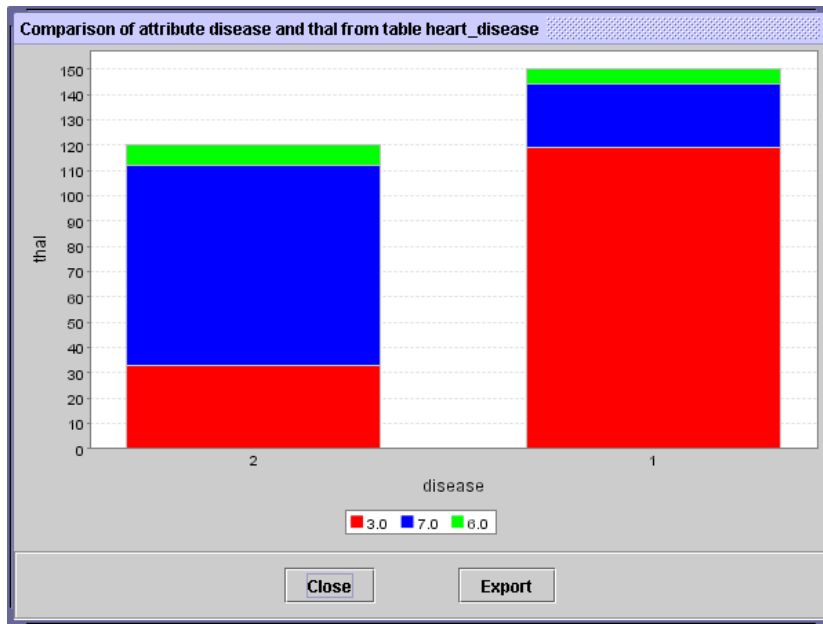


Figure 3.11: Distribution of two symbolic attributes presented as graph.

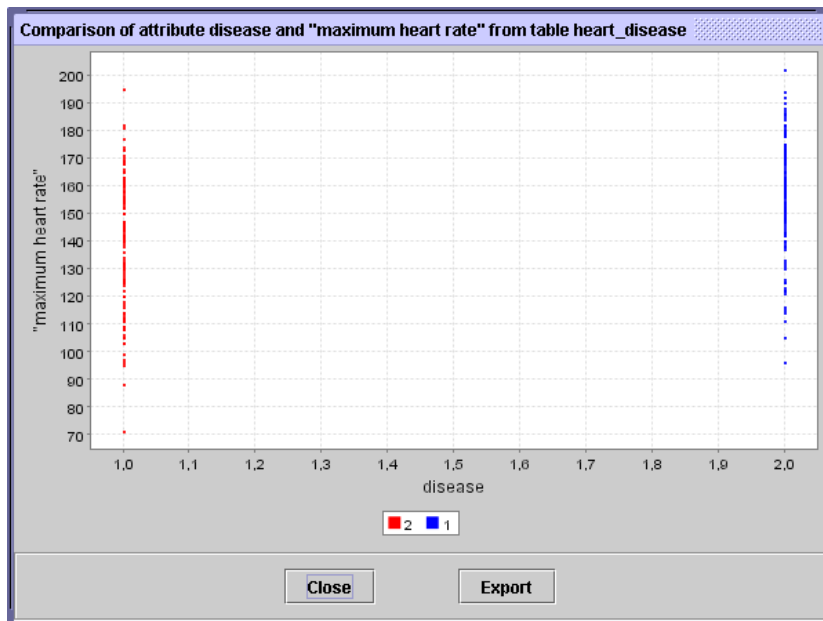


Figure 3.12: Comparison of two attributes: symbolic and numeric.

3.2 Reduct sets

Reduct for an information system is a subset of attributes which preserves all discernibility information from the information system, and none of its proper subsets has this ability (see [22]).



Figure 3.13: Icon representing reduct set

Double clicking on the icon representing reduct set corresponds to the **View** option in the context menu, and results in the contents of this object being displayed.

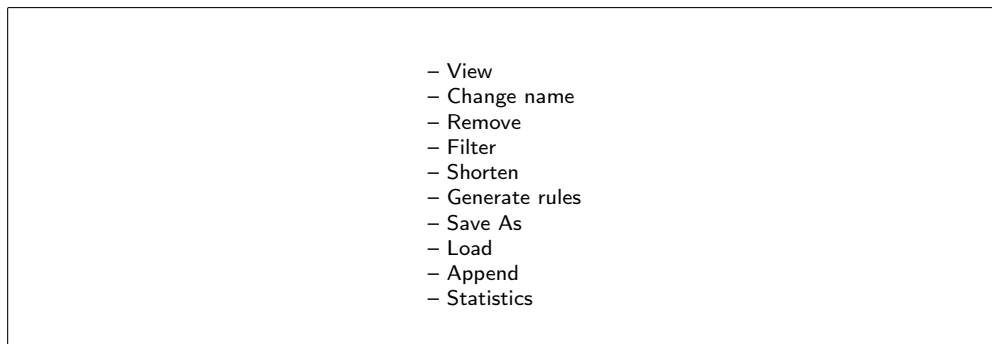


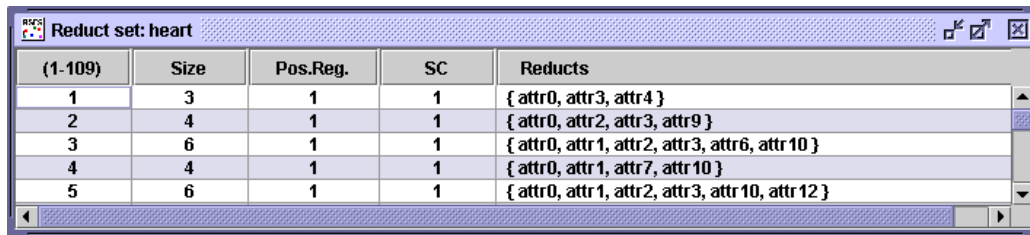
Figure 3.14: Layout of context menu for reduct set

Options in the context menu for reduct set:

- **View** – displays contents of the reduct set (see figure 3.15). The user can scroll and resize this window according to requirements.

The reduct set view window consists of five columns. First of these columns stores the identification number, the others have the following meaning (for a single row):

- **Size** – size of the reduct, number of participating attributes.
- **Pos.Reg.** – the positive region for the table after reduction, i.e. after removing attributes from outside the reduct.



(1-109)	Size	Pos.Reg.	SC	Reducts
1	3	1	1	{ attr0, attr3, attr4 }
2	4	1	1	{ attr0, attr2, attr3, attr9 }
3	6	1	1	{ attr0, attr1, attr2, attr3, attr6, attr10 }
4	4	1	1	{ attr0, attr1, attr7, attr10 }
5	6	1	1	{ attr0, attr1, attr2, attr3, attr10, attr12 }

Figure 3.15: Viewing contents of reduct set

- SC – value of the Stability Coefficient (SC) for the reduct. this value is used to determine the stability of reduct in dynamic case (see [2] and subsection 4.4).
 - Reducts – reduct presented as a list of attributes.
- Change name – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
 - Remove – removes reduct set (additional confirmation required).
 - Filter – filters the reduct set. The user can remove reducts on the basis of stability coefficient (SC). Before using this option it is recommended to examine statistics for the set of reducts to be filtered.
 - Shorten – shortening of reducts. The user provides a coefficient between 0 and 1, which determines how “aggressive” the shortening procedure should be. The coefficient equal to 1.0 means that no shortening occurs. If Shortening ratio is near zero, the algorithm attempts to maximally shorten reducts. This shortening ratio is in fact a threshold imposed on the relative size of positive region after shortening.
 - Generate rules – generates a set of decision rules on the basis of the reduct set and selected data table (see also subsection 4.4).
 - Save As – saves the set of reducts to a file.
 - Load – loads previously stored reduct set from a file.
 - Append – appends the current reduct set with reducts from a file. Repeating entries only appear once.

- **Statistics** – present basic statistics on the reduct set (see figure 3.16). It also provides the ability for displaying the core (intersection of all reducts).

The user may also review the graphical information on distribution of reduct lengths as well as on frequency and role of particular attributes in reducts' construction.

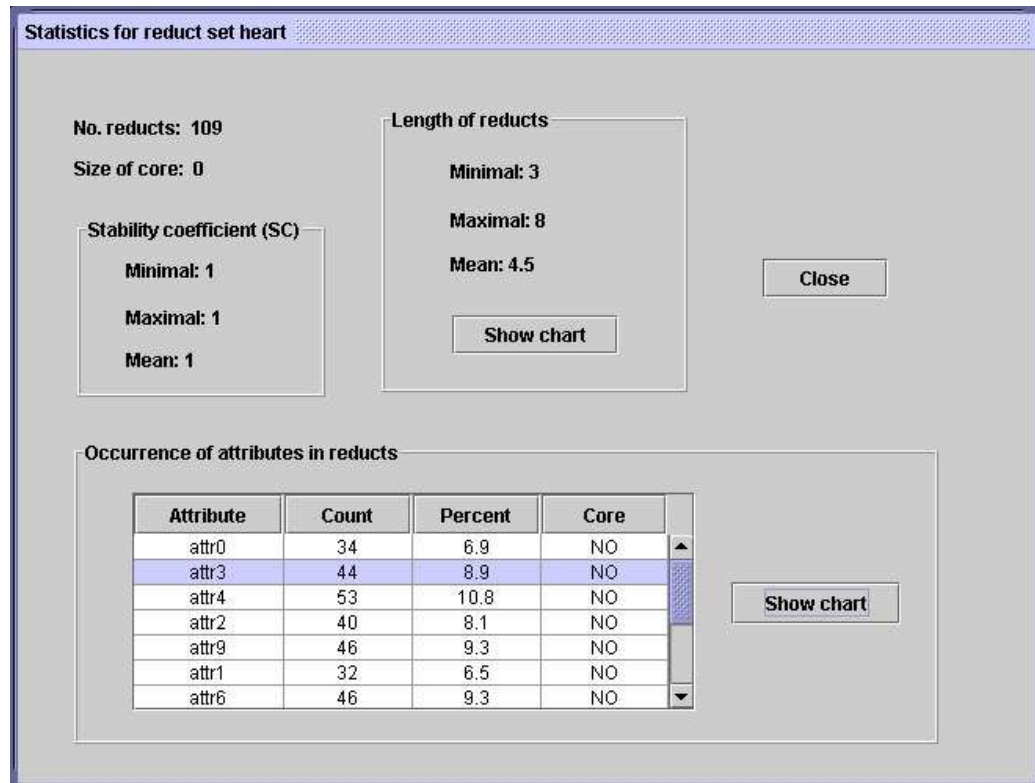


Figure 3.16: Information on reduct set

3.3 Rule sets

Decision rules make it possible to classify objects, i.e. assign the value of decision attribute. Having a collection of rules pointing at different decision we may perform a voting obtaining in this way a simple rule-based decision support system.

Double clicking on the icon representing rule set corresponds to the **View** option in the context menu, and results in the contents of this object being displayed.



Figure 3.17: Icon representing rule set

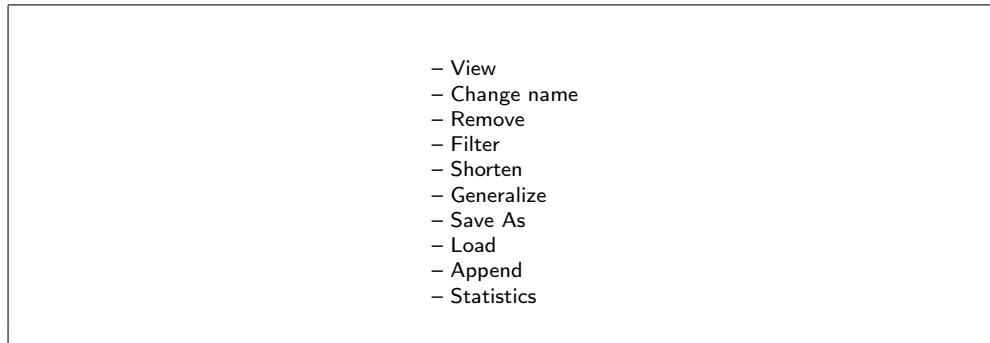


Figure 3.18: Layout of context menu for rule set.

List of options in context menu for rule set:

- View – displays contents of the rule set (see figure 3.19). The user can scroll and resize this window according to requirements.

(1-161)	Match	Decision rules
1	29	(attr4=1)=>(attr6={1[29]})
2	17	(attr0=3)&(attr1=3)=>(attr6={1[17]})
3	15	(attr0=2)&(attr1=2)=>(attr6={1[15]})
4	9	(attr0=1)&(attr1=1)=>(attr6={1[9]})
5	8	(attr0=1)&(attr1=3)&(attr4=4)=>(attr6={0[8]})
6	8	(attr0=3)&(attr3=1)&(attr5=1)=>(attr6={1[8]})
7	7	(attr0=1)&(attr2=1)&(attr4=4)=>(attr6={0[7]})
8	7	(attr0=1)&(attr4=4)&(attr5=2)=>(attr6={0[7]})
9	7	(attr0=1)&(attr1=2)&(attr2=2)=>(attr6={0[7]})
10	7	(attr0=2)&(attr1=1)&(attr2=2)=>(attr6={0[7]})

Figure 3.19: Viewing contents of rule set

The rule set view window consists of three columns. First of these columns stores the identification number, the others have the following meaning (for a single row):

- **Match** – number of objects from training table matching the conditional part of the rule (support of the rule).
- **Decision rules** – the rule itself, presented as a logical formula.
- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
- **Remove** – removes reduct set (additional confirmation required).
- **Filter** – filters the rule set. The user can remove rules on the basis of support or the rules pointing at particular decision class. Before using this option it is advisable to examine statistics for the set of rules to be filtered.
- **Shorten** – shortening of rules. The user provides a coefficient between 0 and 1, which determines how “aggressive” the shortening procedure should be. The coefficient equal to 1.0 means that no shortening occurs. If **Shortening ratio** is near zero, the algorithm attempts to maximally shorten rules (cf. [2], [5]).
- **Generalize** – make rules more general. The user provides a coefficient (a ratio) between 0 and 1, which determines how “aggressive” the generalization procedure should be. The coefficient equal to 1.0 means that the generalization must preserve the precision level of original rules. For coefficients closer to zero, the generalization may cause rule to loose precision for the sake of greater applicability.
- **Save As** – saves the set of rules to a file.
- **Load** – loads previously stored rule set from a file.
- **Append** – appends the current rule set with rules from a file. Repeating entries only appear once.
- **Statistics** – present basic statistics on the rule set (see figure 3.20).

The user may also review the graphical information on distribution of rule lengths as well as on distribution of rules between decision classes.

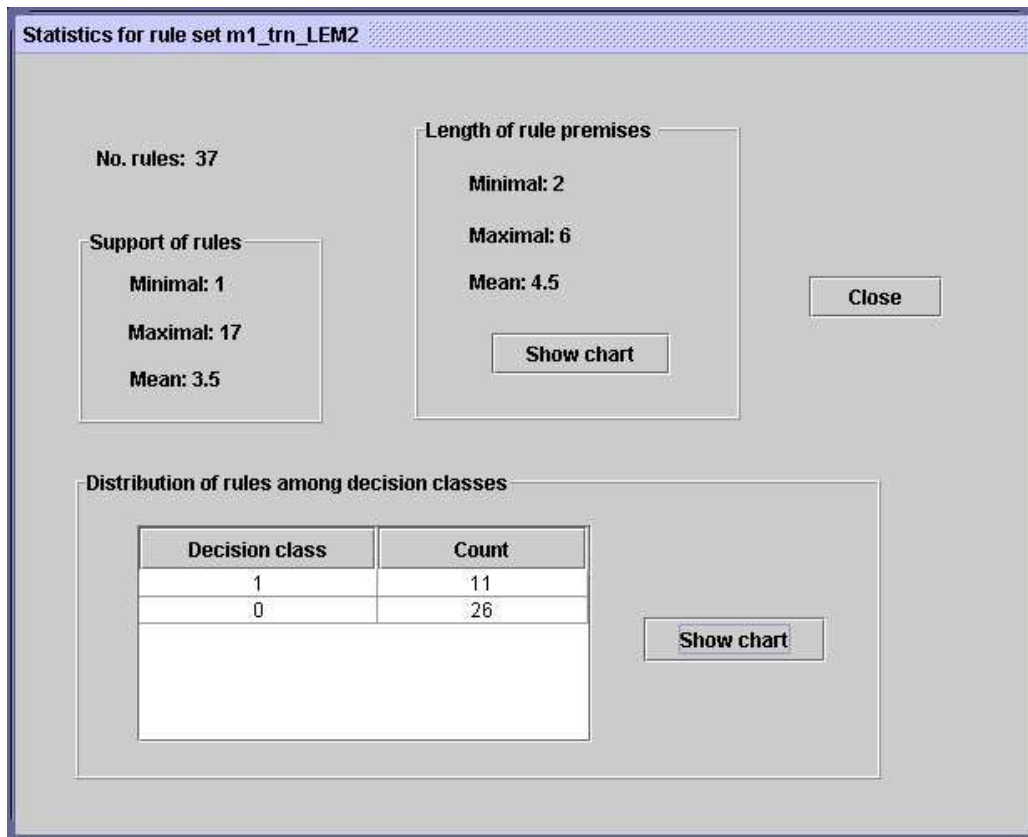


Figure 3.20: Information about rule set

3.4 Cut sets

By cuts we understand the definition for decomposition of attribute value sets. In case of numerical attributes being discretized in order to produce a collection of intervals, the cuts are thresholds defining these intervals. In case of symbolic attributes being grouped (quantized), cuts define disjoint subsets of original attribute values.



Figure 3.21: Icon representing cut set

Double clicking on the icon representing rule set corresponds to the View option in the context menu, and results in the contents of this object being

displayed.

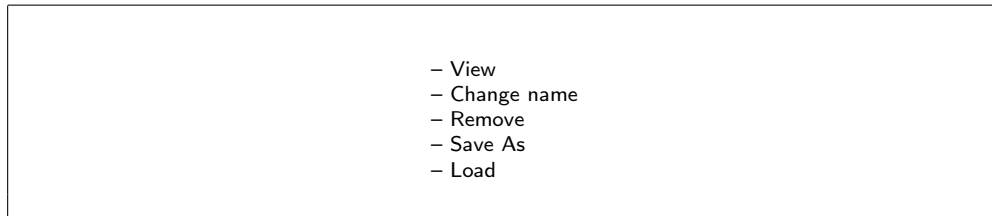


Figure 3.22: Layout of context menu for cut set

List of options in context menu for cut set:

- **View** – displays contents of the rule set (see figure 3.23). The user can scroll and resize this window according to requirements.

(1-6)	Attribute	Size	Descrip...
1	attr0	2	1.5; 2.5
2	attr1	2	1.5; 2.5
3	attr2	1	1.5
4	attr3	1	1.5
5	attr4	2	1.5; 2.5
6	attr5	1	1.5

Figure 3.23: Viewing contents of cut set

The cut set view window consists of four columns. First of these columns stores the identification number, the others have the following meaning (for a single row):

- **Attribute** – name of of the attribute for which the cuts have been calculated.
 - **Size** – number of cuts used for this attribute.
 - **Description** – list of values representing cuts, * represents absence of cuts for attribute.
- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does

not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.

- Remove – removes cut set (additional confirmation required).
- Save As – saves the set of cuts to a file.
- Load – loads previously stored cut set from a file.

3.5 Linear combinations

Linear combination is an attribute (newly) created as a weighted sum of selected existing attributes. We may have several such attributes for different weight settings and different attributes participating in weighted sum.

Linear combinations are created on the basis of collection of attribute sets consisting of k elements. Those 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 [23]. The user may specify, by inputting the collection of numbers in **Pattern of linear combinations** field, the number of new attributes to be constructed and the number of original attributes to be used in linear combination. In this field user states how many attributes should appear in particular combination. For instance, by entering the sequence “223344” the user orders the algorithm to generate 6 (as there are 6 numbers in the sequence) linear combinations using two (first couple), three (third and fourth) and four (last two) original attributes, respectively.

Notice! *The algorithm sometimes returns less combinations that it was ordered, or returns combinations with fewer components. This may happen if there are no combinations that comply to specification and, at the same time, make any sense for a given data.*



Figure 3.24: Icon representing a set of linear combinations

Double clicking on the icon representing the set of linear combinations corresponds to the **View** option in the context menu, and results in the contents of this object being displayed.

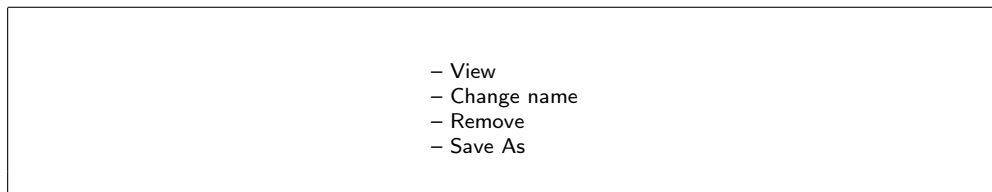


Figure 3.25: Layout of context menu for a set of linear combinations

List of options in context menu for a set of linear combinations:

- **View** – displays contents of the set (see figure 3.26). The user can scroll and resize this window according to requirements.

(1-4)	Linear combinations
1	$\text{attr0} \cdot 0.707 + \text{attr2} \cdot 0.707$
2	$\text{attr1} \cdot 0.0 + \text{attr3} \cdot 0.0 + \text{attr4} \cdot (-1.0)$
3	$\text{attr0} \cdot 0.447 + \text{attr5} \cdot (-0.894)$
4	$\text{attr0} \cdot 0.707 + \text{attr2} \cdot 0.707 + \text{attr4} \cdot 0.0$

Figure 3.26: Viewing the contents of linear combination set

The linear combination view window consists of just two columns. First of these columns stores the identification number, the other stores the linear combination itself written as an arithmetic formula.

- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
- **Remove** – removes set of linear combinations (additional confirmation required).
- **Save As** – saves the set of linear combinations to a file.

3.6 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. For more information on underlying methods please turn to [20] and [4]. Usually the subsets of data in the leafs of decomposition tree are used for calculation of decision rules (cf. [5]).



Figure 3.27: Icon representing decomposition tree

Double clicking on the icon representing the set of linear combinations corresponds to the **View** option in the context menu, and results in the opening of a window with a graph of the tree.

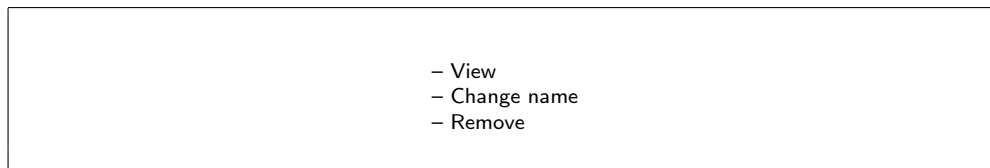


Figure 3.28: Layout of context menu for decomposition tree

List of options in context menu for a decision tree:

- **View** – opens new window with the display of decision tree (see figure 3.29). The user can scroll and resize this window according to requirements.

The decomposition tree view window displays the information on the number of tree nodes and the tree itself. By placing mouse cursor over any of tree nodes we may get (after a second a tag appears) an information about the template (pattern) which is matched by subset of objects corresponding to this node. Each internal (green) tree node is associated with a simple context menu consisting of just one option **View node info**. This context menu allows for inspection of template corresponding to each node. The same information may be accessed by double-clicking on the selected internal node. In case of leaf (red) nodes the context menu contains two more options **View rules** and **Save**

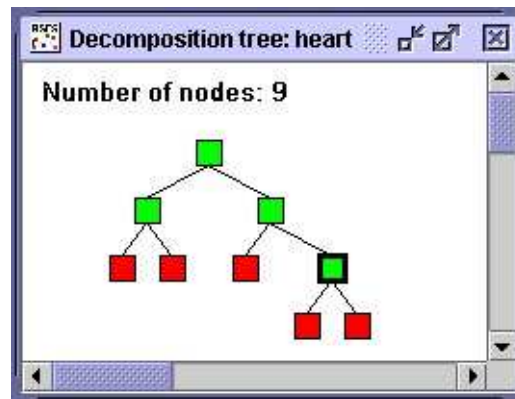


Figure 3.29: Decomposition tree view

rules. The View rules option opens a new window that lists the rules corresponding to the leaf. The rules are displayed in the standard RSES way (see figure 3.19 in subsection 3.3). Save rules option lets user store the rules in a file on disk.

- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in project file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
- **Remove** – removes decomposition tree from project (additional confirmation required).

3.7 LTF Classifiers

LTF-C (*Local Transfer Function Classifier*) (see [24]) is a classification-oriented artificial neural network model similar to that of Radial Basis Function network (RBF). It consists of two layers of computational neurons (plus an input layer). The first computational layer (hidden layer) consists of neurons that correspond to clusters of objects from the same decision class. Each of these neurons have a decision class assigned and tries to construct a cluster of objects from this class. The second computational layer (output layer) consists of neurons which gather the information from hidden (cluster-related) neurons, sum it up and produce final network's output.

Double clicking on the icon representing the LTF-C corresponds to the



Figure 3.30: Icon representing LTF-C

View option in the context menu, and results in the opening of a window with the network description.

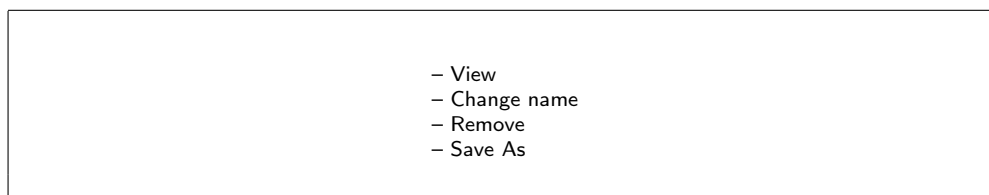


Figure 3.31: Layout of context menu for LTF-C

List of options in context menu for LTF-C:

- **View** – opens new window with the display of network parameters (see figure 3.32). The user can scroll and resize this window according to requirements.

The window with information on LTF-C displays contents of the file that is being created during the network's learning process. This file is split into two parts. First part stores the parameters used for learning, the other describes network architecture (settings for both the whole structure and single neurons).

The value of each parameter in this description is trailed by **@** and parameter's name. Among some teen parameters in file (starting with **@MaxNeur** up to **@DeltaTr**) only **@EtaUse** and **@UseTr** are important for the user. they are set to be equal to the value which is input by the user in the field **Threshold for neuron removal** in the window used to create LTF-C. The value of this threshold controls the process of removing redundant hidden neurons during network training. The larger the threshold, the easier neurons are being dropped. For large threshold the resulting neural network is more compact. However, the threshold should not be too high in order to obtain relevant and accurate classifier.

At the line **@Inputs** the network description starts:

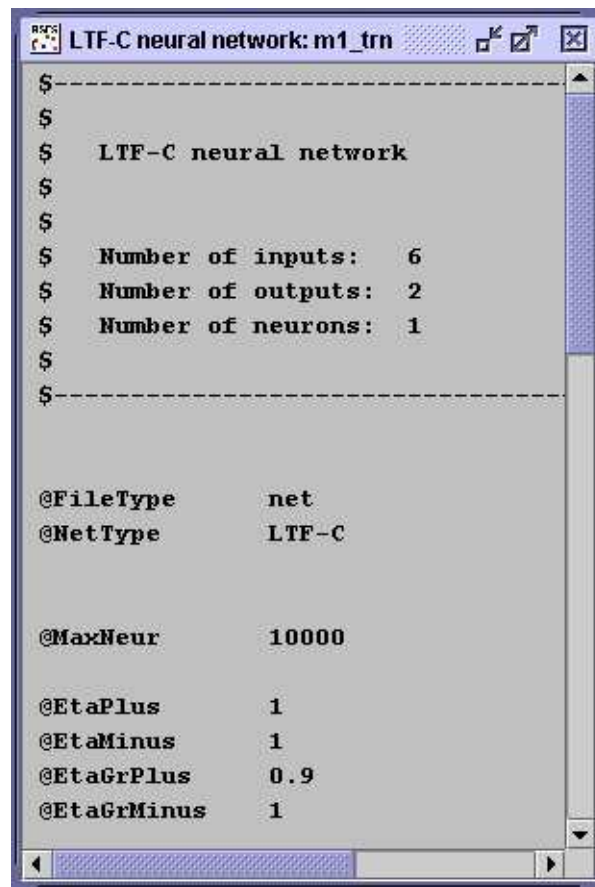


Figure 3.32: LTF-C view

- @Inputs – number of attributes, i.e. size of input
- @Outputs – number of outputs, i.e. number of decision classes
- @Neurons – number of neurons
- @SumCycles – number of number of training cycles (each cycle is a single presentation of training object). This number is equal to value input by the user in Number of training cycles field in network creation window.

In the following lines neurons are described (see fig. 3.33).

Important parameters describing neuron:

- @Class – number of decision class to which this neuron corresponds – number between 0 and (@Outputs-1)

```

@<Neuron0
  @Class      1
  @Life       2999
  @AvgUsefuln 0.04410261
  @AvgActiv   0.00400000
  @EtaEta     0.000
  @Out        0.000
  @Height     1.0000

  @Weights
    0.8408  0.3130  0.3921  0.3070

  @RecipOfRadii
    0.3955  0.7920  1.9009  1.5605
@>

```

Figure 3.33: Example of the neuron description (view) in LTF-C

- **@Life** – neuron’s lifespan in number of training cycles. For most of neurons this number is smaller than **@SumCycles** as there is only one neuron in initial networks, and others are being automatically added if necessary.
- **@Weights** – neuron’s weights, i.e. coordinates of the center of cluster which correspond to the neuron. These coordinates describe input vector for which the neuron produces maximal output value.
*Notice! If during network training the **Normalize each numeric attribute** was active then weights are normalized as well as corresponding attributes. The attributes are normalized to have zero as expected value and variance equal to 1.*
- **@RecipOfRadii** – reciprocal of neuron’s radii, i.e. width of its reception field (in all dimensions). Smaller value corresponds to wider reception field (w.r.t a given dimension) which is an indicator of decreased importance of this dimension for the neurons final output.
*As with weights, the values of radii depend on normalization (**Normalize each numeric attribute** option set by user).*

Notice! neurons which were removed during training are not present in network description. only neurons which survived to the end of training process are there.

- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
- **Remove** – removes LTF-C (additional confirmation required).
- **Save As** – saves the LTF-C to a file.

3.8 Results

The “Results” object is used to store the outcome of classification experiment.



Figure 3.34: Icon representing results

Double clicking on the icon representing the set of results corresponds to the **View** option in the context menu, and results in the contents of this object being displayed.

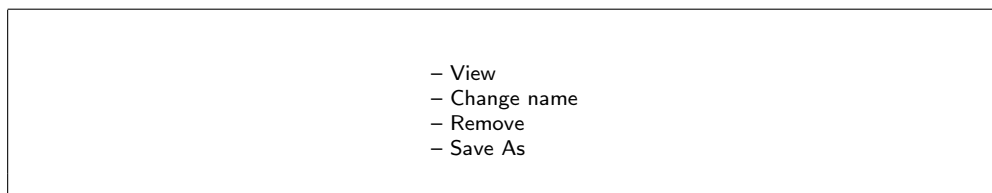


Figure 3.35: Layout of context menu for a set of results

List of options in the context menu for results:

- **View** – opens new window with the display of result summary (see figure 3.36). The user can scroll and resize this window according to requirements.

		Predicted				
Actual		1	0	No. of obj.	Accuracy	Coverage
	1	210	6	216	0.972	1
	0	52	164	216	0.759	1
True positive rate		0.8	0.96			

Total number of tested objects: 432
Total accuracy: 0.866
Total coverage: 1

Figure 3.36: View of classification results

The window presenting classification results provides various information. The central part is occupied by (*confusion matrix*), which in our example is of the form:

	Predicted	
	1	0
Actual	210	6
	52	164

Rows in this matrix correspond to actual decision classes (all possible values of decision) while columns represent decision values as returned by classifier in discourse. In our example (above) we may see that the constructed classifier sometimes mistakes objects from class 1 for those from class 0 (6 such cases). We may also see that the classifier mistakes class 0 for class 1 several times more frequently than the other way round (52 cases as compared to 6). The values on diagonal represent correctly classified cases. If all non-zero values in confusion matrix appear on the diagonal, we conclude that classifier makes no mistakes for a given set of data.

On the right side of the matrix there are additional columns with information such as:

- No. of obj. – number of objects in data set that belong to the decision class corresponding to current row.

- **Acuracy** – ratio of correctly classified objects from the class to the number of all objects assigned to the class by the classifier.
- **Coverage** – ratio of classified (recognized by classifier) objects from the class to the number of all objects in the class.

Last row in the table contains **True positive rate** for each decision class.

Below the table number of additional values is presented:

- **Total number of tested objects:** – number of test objects used to obtain this result.
- **Total accuracy** – ratio of number of correctly classified cases (sum of values on diagonal in confusion matrix) to the number of all tested cases (as in previous point).
- **Total coverage** – percentage of test objects that were recognized by classifier.

In our example **Total coverage** equals 1 which means that all objects have been recognized (classified). Such total coverage is not always the case, as the constructed classifier may not be able to recognize previously unseen object. If some test objects remain unclassified, the **Total coverage** value is less than 1.

- **Change name** – changes object name (see figure 3.4), the name is stored together with the contents of object in file. The name of object does not need to be identical with the name of file that is used to store it. The name of object can also be changed by double clicking on name tag below the icon representing object.
- **Remove** – removes results (additional confirmation required).
- **Save As** – saves results to a file.

Chapter 4

Main data analysis methods in RSES

This part presents a very brief description of main data analysis methods (algorithms) available in RSES 2.2. It contains brief listing of options used to control the implemented algorithms as well as short introduction to underlying theories with reference to appropriate literature.

Details concerning the meaning and construction of objects appearing in RSES projects were already introduced in section 3.

4.1 Missing value completion

The missing elements in data table may be denoted as `MISSING`, `NULL` or `'?'` (not case sensitive).

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

- removal of objects with missing values by using `Complete/Remove objects with missing values` option from data table context menu,
- filling the missing part of data in one of two ways (see [11]):
 - 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. This procedure is invoked by selecting `Complete/Complete with most common or mean value` from data table context menu.
 - 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. This procedure is invoked by selecting **Complete/Complete with most common or mean value w.r.t. decision class** from data table context menu.

- 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 dialog windows for reduct/rule calculation.
- treating the missing data as information (NULL is treated as yet another regular value for attribute).

4.2 Cuts, discretization and grouping

With use of **Discretize/Generate cuts** from data table context menu we may generate decompositions of attribute value sets. With these descriptions, further referred to as *cuts* we may perform next step, i.e. discretization of numerical attributes or grouping (quantization) of nominal attributes.

The user may set several parameters that control discretization/grouping procedure:

- **Method choice** – choice of discretization method from:
 - **Global method** – global method (see, e.g. [5]),
 - **Local method** – local method, slightly faster than the global one but, generating much more cuts in some cases (see, e.g. [5]).
- **Include symbolic attributes** – option available only for local method. Causes algorithms to perform grouping for nominal (symbolic) attributes in parallel to discretization of numerical ones.
- **Cuts to** – determines the name of object in project used to store cuts.

Discretize/Discretize table option (in data table context menu) makes it possible to discretize (group) attributes in the data table with use of previously calculated cuts. The user sets the set of cuts to be used and the name of object to store resulting discretized table. The set of cuts must be present in project prior to its use which means it has to be calculated in the previous step or loaded from a file.

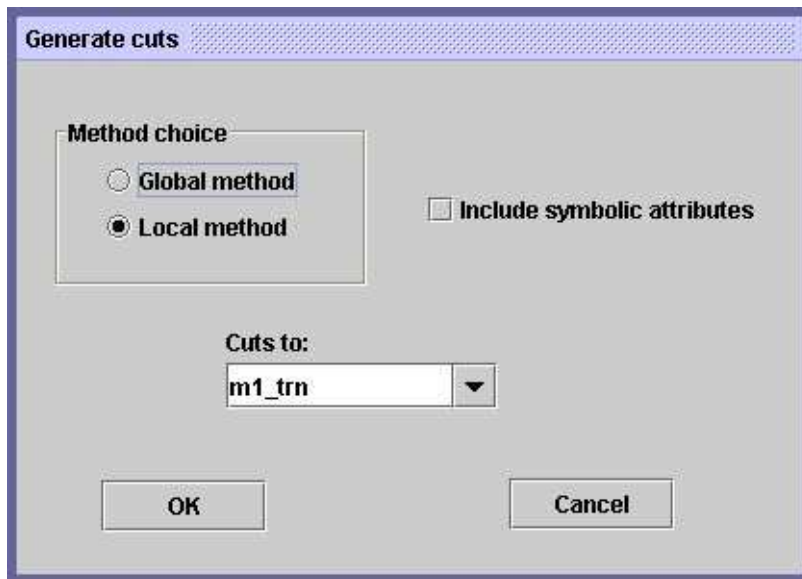


Figure 4.1: Options in generation of cuts.

4.3 Linear combinations

RSES 2.2 sports an algorithm for construction of new attributes based on linear combinations of existing ones (see [23]). To perform search for appropriate linear combination the user has to use **Linear combinations/Generate linear combinations** option from the context menu for data table.

The dialog window associated with creation of linear combinations provides the user with several options (see figure 4.2):

- **Pattern of linear combinations** – a scheme for new attributes to be constructed. In this field user states how many attributes should appear in particular combination. For instance, by entering the sequence “22344” the user orders the algorithm to generate 5 linear combinations using two (first couple), three (third) and four (last two) original attributes, respectively.
- **Measure choice** – selection of one of measures for quality of combination (see [23]).
- **Linear combinations to** – selection of object in project which will be used to store the result (linear combinations).

In order to make use of derived linear combinations the user has to invoke **Linear combinations/Add linear combinations as new attributes** from context

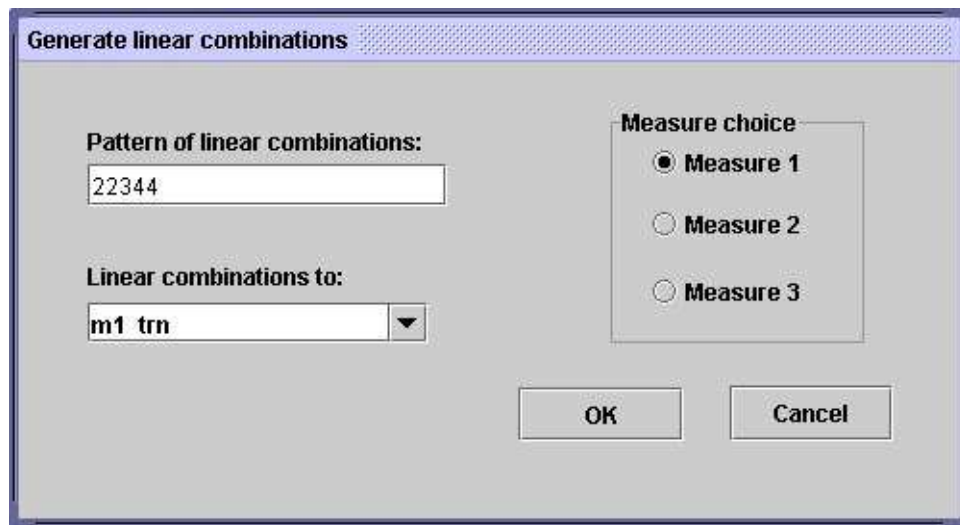


Figure 4.2: Options for generation of linear combinations

menu for data table. Use of this option results in new attributes, based on linear combinations, being added to the decision table. The linear combinations to used for extending data table have to be calculated beforehand.

Please note, that by using **Select subtable** option from the decision table's context menu it is possible to manually select the attributes (columns). In this way the user may create table that contains, for instance, only newly created attributes and the decision.

4.4 Reducts and decision rules

Given a data table we may want to derive reducts and/or decision rules. For this purpose we use **Reducts/Rules/Calculate reducts or rules** option from the decision table's context menu.

The user should determine, whether the rules or reducts should be calculated and what options should be active during calculation (see figure 4.3). The setting of options controls the behavior of algorithms for reduct/rule calculation. The options are:

- **Reduct/Rule choice** – choice of the operation mode:
 - **Reducts** – calculation of reducts,
 - **Rules** – induction of decision rules.

- **Discernibility matrix settings** – properties of discernibility matrix (*option available only for reduct calculation*):
 - Full discernibility – full (global discernibility),
 - Object related discernibility – discernibility w.r.t single object in table is used (relative discernibility),
 - Modulo decision – discernibility w.r.t decision only is used (decision/class discernibility),
- **Method** – choice of calculation method (algorithm):
 - Exhaustive algorithm – an exhaustive, deterministic algorithm constructing all reducts or all minimal decision rules, depending on the setting of Reduct/Rule choice option. The calculated rules are those with minimal number of descriptors in conditional part (for more information see [5]).
 - Genetic algorithm – genetic algorithm for calculation of some reducts/rules (see, e.g. [5]).
 - Covering algorithm – covering algorithm (*rule calculation only* – see [5]),
 - LEM2 algorithm – LEM2 algorithm (*rule calculation only* – (see [10])),
- **Genetic algorithm settings** – parameters for calculation of reducts/rules with use of genetic algorithm (*options only active if genetic algorithm is chosen as calculation method*):
 - High speed – quick mode – fast but may result in less accurate output.
 - Medium speed – regular mode.
 - Low speed – thorough mode (slow).
 - Number of reducts – maximum number of reducts to be calculated by algorithm (size of population for genetic algorithm).
- **Cover parameter** – expected degree of coverage of the training set by derived rules (*option available only for covering and LEM2 methods*),
- **Don't discern with missing values** – do not consider two objects to be discernible on some attribute in case when at least one of them have NULL as a value of this attribute. See also subsection 4.1.

- Generate to set – name of object in project that is used to store results of calculation (reducts or rules). It may be either existing or completely new object. In the latter case an appropriate object is created at the end of calculation.

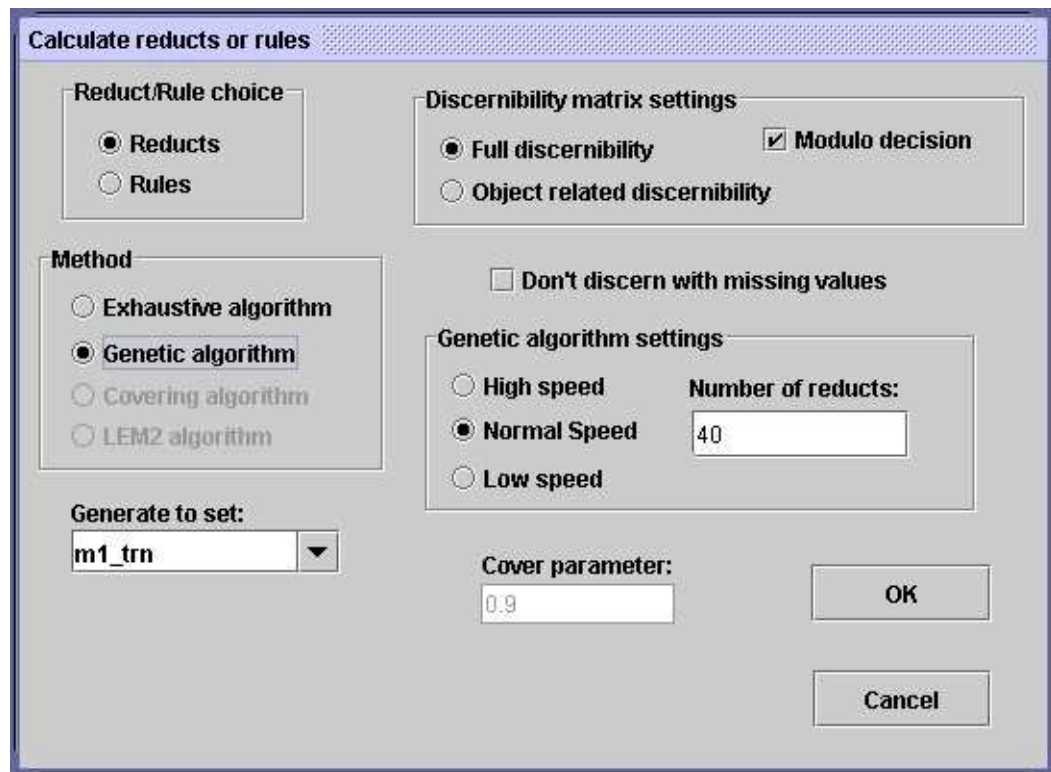


Figure 4.3: Options for calculating reducts/rules

It is also possible to generate dynamic reducts. Dynamic reducts are reducts that remain to be such for many subtables of the original decision table (see [2], [5]). 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). To invoke dynamic reduct calculation the user have to select Reducts/Rules/Calculate dynamic reducts option from context menu associated with data table.

The dynamic reduct calculation process is controlled by following options (see Fig. 4.4):

- Number of sampling levels – number of sampling levels for selection of subtables (samples).

- Number of subtables to sample per level – number of subtables (samples) chosen by random on each level.
- Smallest subtable size (in % of whole table) – size of the smallest allowable subtable used in calculation, expressed as a percentage of the whole, original decision table.
- Largest subtable size (in % of whole table) – size of the largest allowable subtable used in calculation, expressed as a percentage of the whole, original decision table.
- Include whole table – permission for the whole table to be used as sample. In such case the calculated dynamic reducts have to be proper reduct for the entire decision table.
- Modulo decision – only discernibility based on decision (class) is considered.
- Don't discern with missing values – ignore missing values. Do not consider two objects to be discernible on some attribute in case when at least one of them have NULL as a value of this attribute. See also subsection 4.1.
- Dynamic reducts to – name of object in project that is used to store results of calculation (dynamic reducts). It may be either existing or completely new object. In the latter case an appropriate object is created at the end of calculation.

Once the reducts are calculated, one may use them to generate decision rules. To do that the user has to use **Generate rules** option from the context menu associated with reduct set.

The following options are available while creating rules from reducts (see figure 4.5):

- Train table from – data table to be used as training sample (*this table must already exist in the project!*).
- Decision rules to – name of object used to store derived rules.
- Rules generation setting – choice of rule generating method:
 - Global rules – calculation of 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

Figure 4.4: Dynamic reducts – options

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.

- **All local rules** – calculation of rules with use of local approach. 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. [5]) w.r.t decision. Finally, the rule sets for all reducts are summed up to form result.

In order to use rule set for classification of objects the user should invoke **Classify/Test table using rule set** option from the context menu of the table used to store the objects to be classified. The set of rules to be used must already exist in the project.

The parameters that may be set by the user when starting classification (see figure 4.6):

- **General test mode** – test mode (decision on expected outcome). The possibilities are:

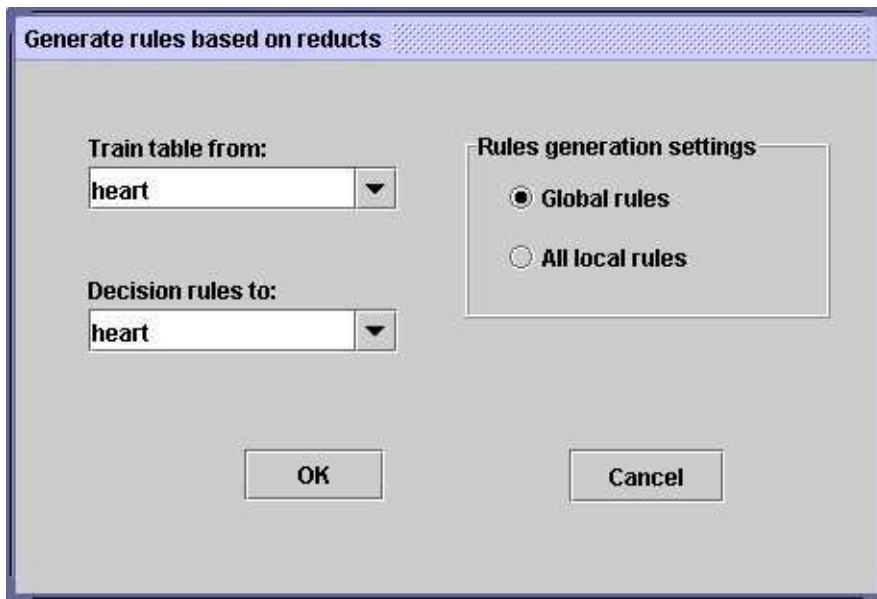


Figure 4.5: Options for building rules from reduct set

- **Generate confusion matrix** – calculates confusion matrix (see subsection 3.8). Applicable only for the tables that already contain decision column.
- **Classify new cases** – classifies objects and adds derived decisions (as last column) to the original table.
- **Conflict resolve by** – method for resolving conflicts when different rules give contradictory results:
 - **Simple voting** – decision is chosen by counting votes casted in favor of each possibility (one matching rule - one vote).
 - **Standard voting** – standard voting solution (each rule has as many votes as supporting objects).
- **Rules from set** – rule set to be used (must exist in project).
- **Results to** – object used to store results.

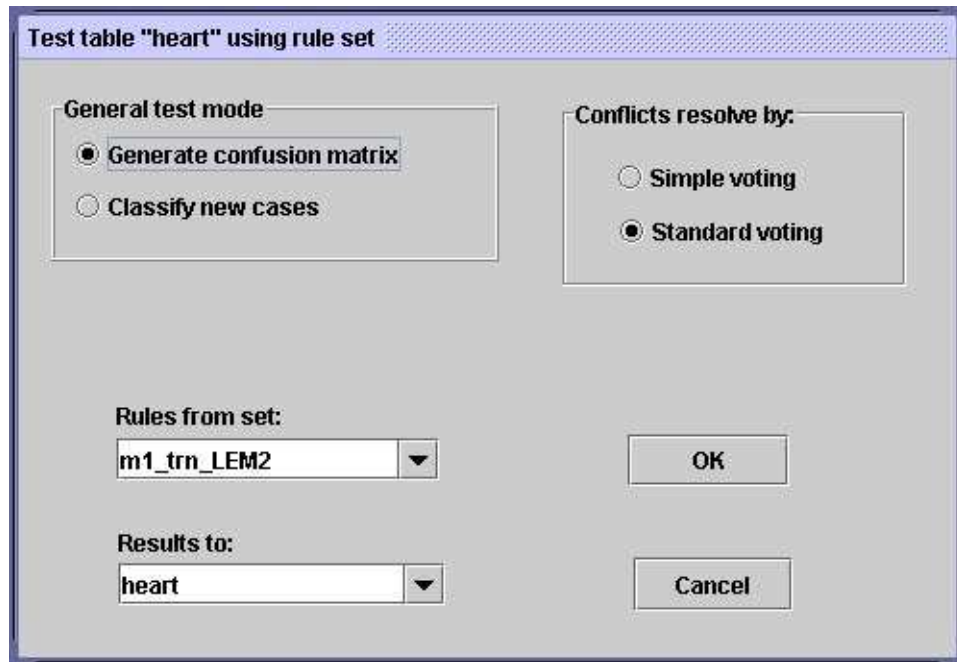


Figure 4.6: Options for classification with decision rules or decomposition tree

4.5 Data decomposition

In order to perform data decomposition, i.e. to construct a decomposition tree that can be further used as a classifier (see [20], [4]) the user should invoke **Make decomposition** option from the table's context menu.

The parameters that may be set by the user when starting decomposition algorithm (see figure 4.7):

- **Maximal size of leaf** – maximal number of objects (rows) in the subtable corresponding to the leaf in decomposition tree. If this number is not exceeded in any of tree nodes, the algorithm ceases to work.
- **Discretization in leafs** – by activating this option the user orders the algorithm to perform on-the-fly discretization of attributes during decomposition. The decision on discretization of particular attribute is taken automatically by the algorithm but, the user has control over some of discretization parameters (see subsection 4.2).
- **Shortening ratio** – rule shortening ratio. The ratio of shortening applied to rules that are calculated by algorithm for subtables corresponding

to decomposition tree nodes. For the value of 1.0 no shortening occurs. The smaller this ratio the more “aggressive” shortening (see subsections 3.3 and 3.6).

- Decomposition tree to – name of object in project used to store the resulting decomposition tree.

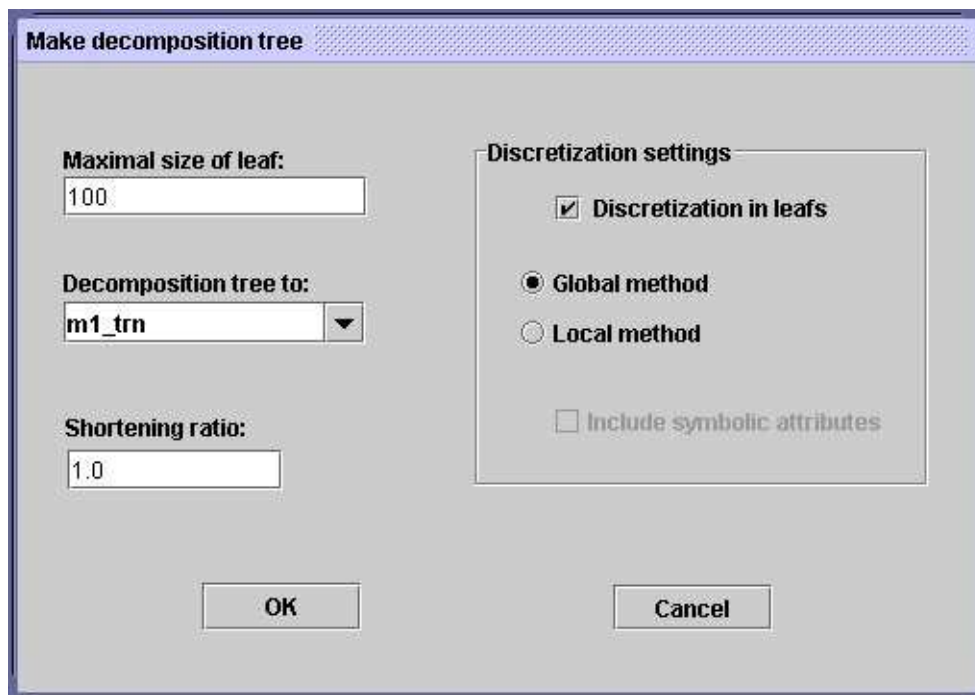


Figure 4.7: Data table decomposition – options

In order to use a decomposition tree as a classifier the user has to call **Classify/Test table using decomposition tree** option from the context menu for decision table. There has to exist at least one decomposition tree object in project, which means that it has to be calculated in prior to classification.

Both the the dialog window (see figure 4.6) and the available classification options are similar to those for rule based classification:

- **General test mode** – test mode (decision on expected outcome). The possibilities are:
 - **Generate confusion matrix** – calculates confusion matrix (see subsection 3.8). Applicable only for the tables that already contain decision column.

- **Classify new cases** – classifies objects and adds derived decisions (as last column) to the original table.
- **Conflict resolve by** – method for resolving conflicts when different rules give contradictory results:
 - **Simple voting** – decision is chosen by counting votes casted in favor of each possibility (one matching rule - one vote).
 - **Standard voting** – standard voting solution (each rule has as many votes as supporting objects).
- **Decomposition tree from** – the name of decision tree object to be used (must exist in project).
- **Results to** – object used to store results.

4.6 k-NN classifiers

The k-NN (k Nearest Neighbor) classification method, as implemented in RSES, does not require a separate training step. It is an instance-based classification technique. All the necessary steps, including initialization and parameter setting, are done at the beginning of classification (test) run. Therefore, to classify objects with use of k-NN the user only has to invoke **Classify/Test table using k-NN** from table's context menu. The only prerequisite for this method is the existence (within project) of nonempty table object that will be used as training sample.

The algorithm constructs a distance measure on the basis of training sample (table). Then, for each object in tested sample, chooses the decision on the basis of decisions for k training objects that are nearest to the tested one with respect to the calculated distance (see [8, 9]).

Parameters for the k-NN method (see figure 4.8):

- **General test mode** – test mode (decision on expected outcome). The possibilities are:
 - **Generate confusion matrix** – calculates confusion matrix (see subsection 3.8). Applicable only for the tables that already contain decision column.
 - **Classify new cases** – classifies objects and adds derived decisions (as last column) to the original table.

Figure 4.8: Options for k-NN classification

- Train table from – determine the table to be used as training sample.
 - Results to – object used to store results.
 - Metric type – choice of metric (parameterized distance measure) to be used:
 - SVD – *Simple Value Difference metric*. SVD - Simple Value Difference Metric. Each attribute value is assigned with the decision distribution vector determined from the training set. The distance between two objects is defined by weighted linear sum of distances for all the attributes. The distance between the two objects w.r.t a single attribute is defined by the difference between the decision distribution vectors corresponding to the attribute values of these objects.
- SVD metric is controlled by Value vicinity parameter that is used to define the distance for numerical attributes. For each numerical value w value vicinity defines how many attribute values close to w in the training set are used for determining the decision

distribution for w . For nominal values the decision distribution is determined as described in [?, ?].

- **City-SVD** – a metric that combines city metric¹ for numerical attributes with *Simple Value Difference* metric for symbolic attributes. For numeric attributes the absolute value of difference between attribute values is taken, for symbolic attributes the difference between corresponding decision distribution vectors from training sample is used.

The user have to set **Normalization** parameter for this metric. This parameter determines the value to be used for normalization of the absolute value of difference for numerical attributes. The choices are:

None – absolute value of difference for numerical attributes is taken as-is (no normalization).

Std.dev. – absolute value of difference for numerical attribute is divided by standard deviation of this attribute calculated from the training sample.

Range – absolute value of difference for numerical attribute is divided by standard the length of the range of this attribute in training set. This length is derived as a difference largest and smallest observed value of the attribute in training sample.

- **Attribute weighting** – choice of attribute weighting (scaling) method. In order to avoid dominance of some attributes over the whole distance, one may use the following options:
 - **None** – no scaling/weighting.
 - **Distance-based** – iterative method for choosing the weights so that they optimize the distance to correctly recognized training objects.
 - **Accuracy-based** – iterative method for choosing the weights so that they optimize the accuracy of decision prediction for training objects.
 - **Number of iterations** – number of iterations during weight optimization.
- **Number of neighbours** – number of neighbours used to classify the object (k in the name of k-NN method):

¹Also known as Manhattan metric.

- Search optimal between 1 and ... – if this option is active, the algorithm determines the optimal size of neighborhood automatically.
- Voting – choice of voting method. The methods that can be used to choose decision as a result of voting among objects in the are:
 - Simple – the object assigned the decision value that is most frequent in the neighborhood.
 - Distance-weighted – each neighbor casts the vote in favor of its decision value. The votes are weighted with use of distance between the neighbor and the object to be classified. Final decision is the one with the largest total of weighted votes.
- Filter neighbors using rules – if this option is selected, the algorithm excludes from voting these neighbor objects which generate a local rule incoherent with other members of neighborhood (see [8, 9]).

4.7 LTF Classifier

In order to build LTF-C (*Local Transfer Function Classifier*) the user have to use Create LTF-C option from the context menu associated with decision table.

The LTF-C is based on dedicated artificial neural network architecture. LTF-C training process is based on slight alternations of network's structure and parameters made after presentation of each training object. There are four basic kinds of alternations:

1. coordinates for the center of gaussian functions;
2. correcting the width of gaussian functions for each hidden neuron and each component of the input independently;
3. adding new neuron to hidden layer;
4. removing redundant neurons from hidden layer.

Before the start of learning procedure the network contains no hidden neurons. Hidden neurons are automatically added during learning if it is necessary. Therefore, the user does not have to determine the network architecture as input and output layers are determined by data dimensions.

Followin options may be used to control LTF-C creation (see figure 4.9):

- LTF-C to – name of object that will store the resulting LTF-C.
- Use default training parameters – choice between default and custom setting of learning parameters. The customizable parameters are:
 - Number of training cycles – the number of LTF-C's training cycles.
 - Threshold for neurons removal – level of error that, if exceeded by neuron, results in removal of a hidden neuron.

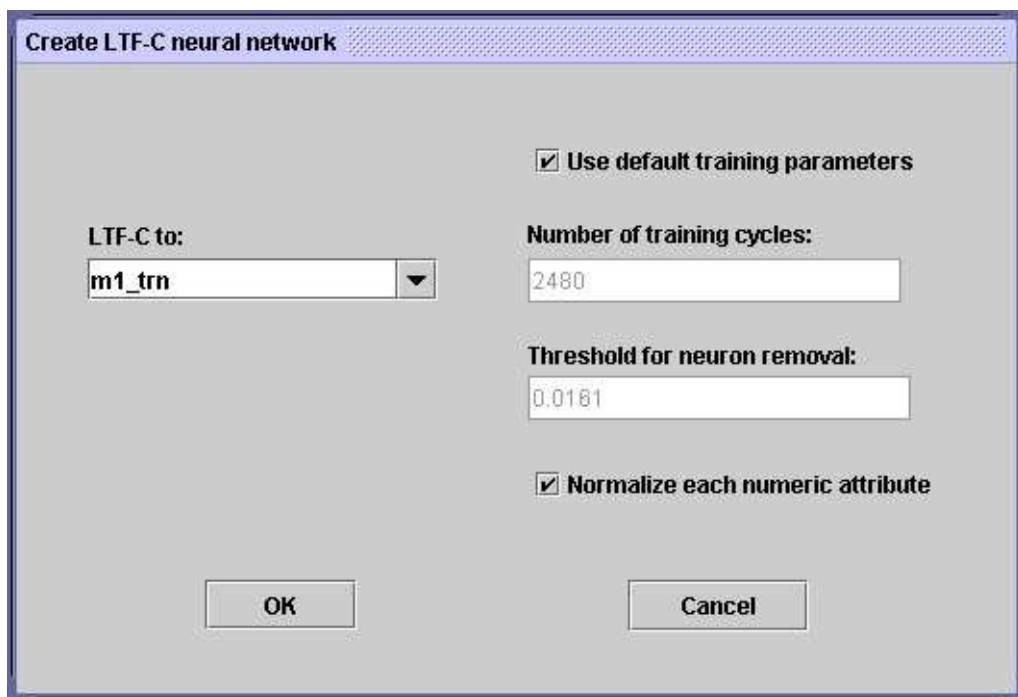


Figure 4.9: LTF-C building options

In order to use LTF-C for classification of new objects (from some table) the user have to call **Classify/Test** table using LTF-C option from the context menu associated with the decision table object. The project should already contain nonempty LTF-C object.

During classification the user may use the following options (see figure 4.10):

- **General test mode** – test mode (decision on expected outcome). The possibilities are:

- **Generate confusion matrix** – calculates confusion matrix (see subsection 3.8). Applicable only for the tables that already contain decision column.
 - **Classify new cases** – classifies objects and adds derived decisions (as last column) to the original table.
- **LTF-C from** – name of LTF-C object to be used.
 - **Results to** – name of object used to store results (if **Generate confusion matrix** is checked).

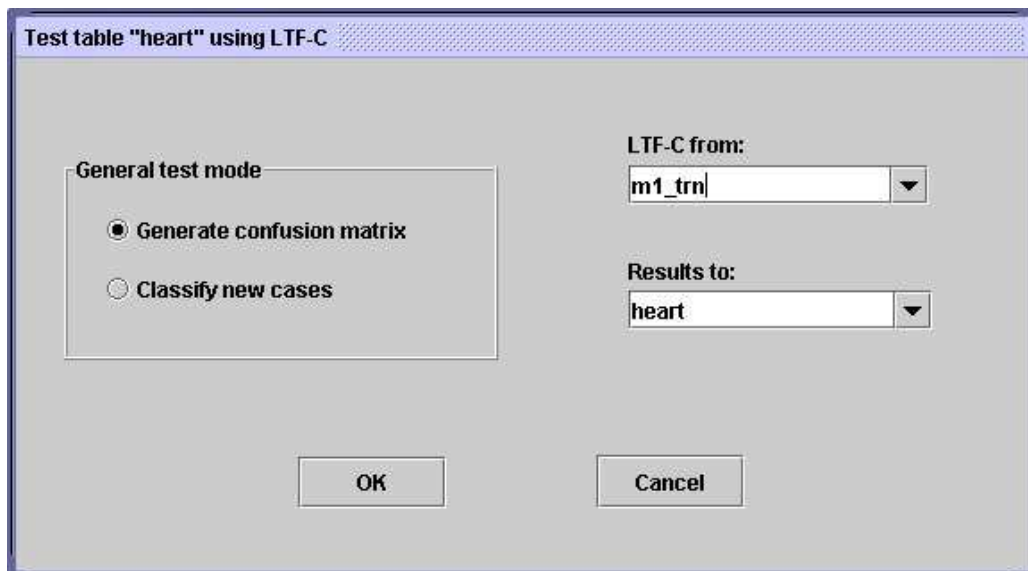


Figure 4.10: Options for classification with LTF-C

4.8 Cross-validation method

Cross-validation (see, e.g. [12]) is frequently used as a method for evaluation of classification models. It comprises of several training and testing runs. The data set is first split into several, possibly equal in size, disjoint parts. Then, one of the parts is taken as training sample and the remainder (sum of all other parts) becomes the test sample. The classifier is constructed with use of training sample and its performance is checked on test sample. These

steps are repeated as many times as there are data parts, so that each of parts is used as training set once. The final result of cross-validation procedure is the average of scores from subsequent steps.

In order to use cross-validation method the user has to select **Classify/Cross-validation method** option from the context menu associated with data table.

The following options are available to user (see figure 4.11):

- **Type of classifier** – the classifier to be used:
 - **Decision rules** – classifier based on decision rules.
 - **Decomposition tree** – decomposition tree based classifier.
 - **k-NN** – classification with k-NN method.
 - **LTF-C** – classification with use of LTF-C.
- **Number of Folds** – number of runs, i.e. number of equal parts into which the original data will be split.
- **Discretization** – when this option is chosen, the discretization of data is performed automatically if such need arises (*option not available in case of LTF-C*).
- **Change discretization method** – choice of discretization method. The discretization options are identical with **Generate cuts** dialog described in subsection 4.2 (*available only when Discretization option is active*).
- **Change parameters of ...** – changing the parameters of the classifier to be used in cross-validation. The parameters are the same as those described before for each of the classifier types.
- **Change shortening ratio** – change of rule shortening ratio for rule based classifiers and decomposition trees (*option available only when rule set or decomposition tree is the current classification method*).
- **Change method of resolving conflicts** – choice of conflict resolution method (voting method) for rule based classifiers and decomposition trees (*option available only when rule set or decomposition tree is the current classification method*).
- **Results to** – name of object used to store the cumulative cross-validation results.

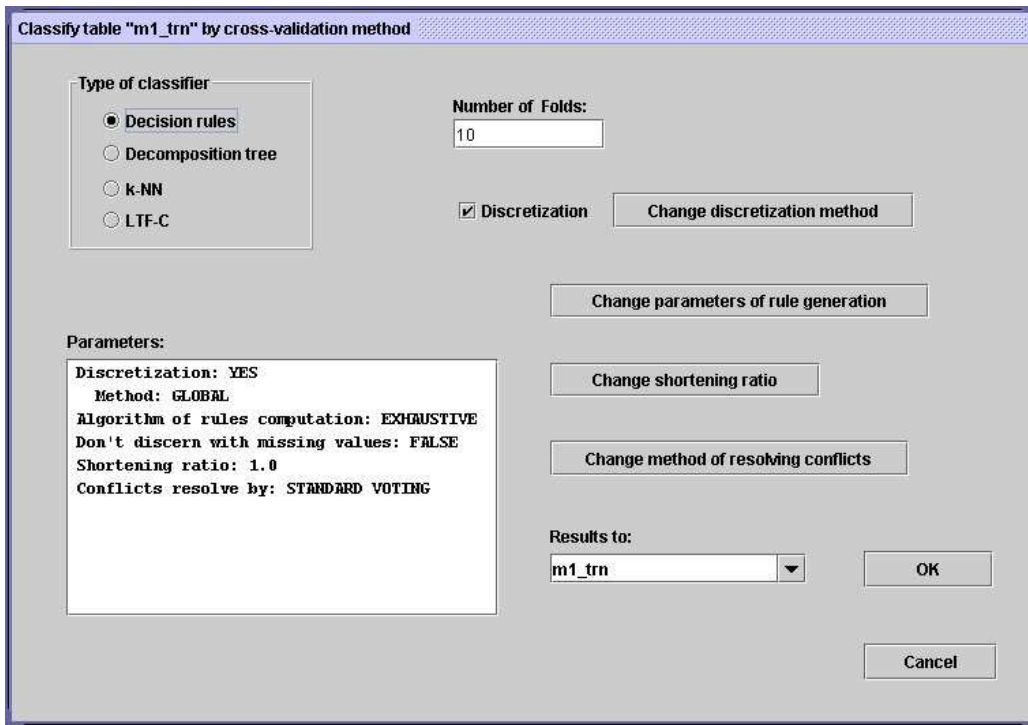


Figure 4.11: Cross-validation options

Results of experiments by cross-validation method: m1_trn						
		Predicted				
		1	0	No. of obj.	Accuracy	Coverage
Actual	1	5.8	0.1	5.9	0.975	1
	0	0.5	5.6	6.1	0.93	1
	True positive rate	0.91	0.99			
Total number of tested objects: 12						
Total accuracy: 0.95						
Total coverage: 1						

Figure 4.12: View of cross-validation results

For greater ease of use the Parameters field in main cross-validation control window lists current parameter settings (e.g. discretization type, shortening ratio etc.).

The summarized results of cross-validation are stored in project with use of typical result set object. It contains the confusion matrix and some additional information, as described in subsection 3.8. The only difference is that the values presented are the averages over all iterations of cross-validation. Therefore, non-integer values may appear in the confusion matrix (see figure 4.12).

Chapter 5

RSES scenario examples

In this part we present several example scenarios for data analysis with use of methods implemented in RSES.

5.1 Train-and-test scenarios

One of the simplest scenarios in data analysis with use of rough set methods is the *train-and-test*. In the event of having a data table for which we want construct and evaluate a classifier we may start with splitting this data into two disjoint parts. One part of data becomes the training sample, the other is used as a testing set for classifier constructed from the training data. As a result of such procedure we obtain classification results for test subtable and the structure of classifier we have learned from training data.

The train-and-test process, as described above, may be parameterized by choice of preprocessing method (discretization, completing missing values etc.) and choice of classifier to be constructed (decision rules, decomposition tree etc.).

5.1.1 Rule based classifier

In this scenario we use train-and-test procedure to construct and evaluate rule based classifier. We do not perform any preprocessing. The scenario comprises of following steps:

1. Launch RSES and create new project.
2. Insert new table to project.

3. Load data into table object (Load option from context menu); in our example we have chosen `irys` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. We split data into two parts containing 60% and 40% of original table, respectively. To do that we select `Split in Two` from context menu and set 0.6 as the value of `Split factor`. As a result we obtain two new table objects in project, named `irys_0.6` and `irys_0.4`. First of them (`irys_0.6`) is used as training set, the other (`irys_0.4`) is used as test set.
5. We calculate decision rules for table `irys_0.6` (from the context menu for this table we select `Reducts/Rules / Calculate reducts or rules` option). As the data set is small, we can afford calculation of all decision rules. We select `Exhaustive algorithm` from rule calculation control window. A new object is being created – a set of rules with the same name as originating table, i.e. `irys_0.6`.
6. We test the rules we just calculated using the test table `irys_0.4`. We open context menu for `irys_0.4` and select `Classify/Test table using rule set` option. In the window that opens we set `Rules from set` value to be `irys_0.6` and `Results to` to be `irys_0.4` (default). For the `Conflict resolve by` option we choose `Standard voting`. Additionally, the `General test mode` option value is `Generate confusion matrix`. A new result object named `irys_0.4` emerges after we hit OK button.
7. We can now examine test results and rules we have calculated by double clicking on result object named `irys_0.4` and rule set object named `irys_0.6`, respectively.

Figure 5.1 shows RSES windows after the end of train-and-test scenario with use of rule based classifier.

5.1.2 Rule based classifier with discretization

The previous scenario may be extended and enriched by application of discretization procedure to training and testing sample. This is particularly important in case of decision tables with numerical attributes. In order to perform the previous scenario with added discretization we need to:

1. Launch RSES and create new project.
2. Insert new table to project.

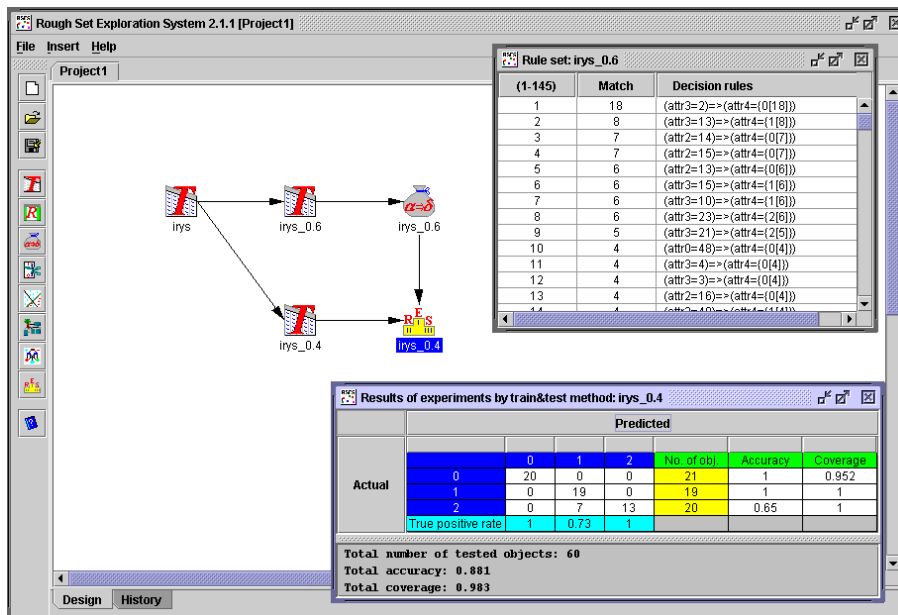


Figure 5.1: RSES after train-and-test scenario with use of rule based classifier

3. Load data into table object (Load option from context menu); in our example we have chosen *irys* data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. We split data into two parts containing 60% and 40% of original table, respectively. To do that we select *Split in Two* from context menu and set 0.6 as the value of *Split factor*. As a result we obtain two new table objects in project, named *irys_0.6* and *irys_0.4*. First of them (*irys_0.6*) is used as training set, the other (*irys_0.4*) is used as test set.
5. We calculate cuts for table *irys_0.6*. For that we select *Discretize/Generate cuts* option from context menu and in dialog that pops up we select method (e.g. *global*) and choose the object that will store resulting cuts (we choose default *irys_0.6* in the *Cuts to* field). A new set of cuts named *irys_0.6* is created within project.
6. We discretize the *irys_0.6* table. This is done by selecting *Discretize/Discretize table* option from context menu. We accept default values in the next window (cuts from *irys_0.6* and resulting table to *irys_0.6D*). As a result a new table object *irys_0.6D* is created in project. This object contains discretized training table.

7. We discretize the `irys_0.4` table. This is done by selecting `Discretize/Discretize table` option from its context menu. We accept default values in the next window (cuts from `irys_0.6` and resulting table to `irys_0.4D`). As a result new table object `irys_0.4D` is created in project. This object contains discretized test table.
8. We calculate decision rules for table `irys_0.6D` (from the context menu for this table we select `Reducts/Rules / Calculate reducts or rules` option). We select `LEM2 algorithm` from rule calculation control window. A new object is being created – a set of rules with the same name as originating table, i.e. `irys_0.6D`.
9. We test the rules we just calculated using the test table `irys_0.4D`. We open context menu for `irys_0.4D` and select `Classify/Test table using rule set` option. In the window that opens we set `Rules from set` value to be `irys_0.6D` and `Results to` to be `irys_0.4D` (default). For the `Conflict resolve by` option we choose `Standard voting`. Additionally, the `General test mode` option value is `Generate confusion matrix`. A new result object named `irys_0.4D` emerges after we hit `OK` button.
10. We can now examine test results and cuts we have calculated by double clicking on result object named `irys_0.4D` and cut set object named `irys_0.6`, respectively.

Figure 5.2 shows RSES windows after the end of train-and-test scenario with use of rule based classifier and discretization.

5.1.3 Decomposition tree

In case of larger data sets (more than 1000 objects) we may find out that traditional methods for classifier construction are not working too well. This is due to complexity issues associated with, e.g. calculation of rules. To address this problem a decomposition method has been implemented in RSES. This method makes it possible to effectively process even very large data table (hundred thousand objects and more).

Here we would like to present typical brief scenario for creating and using a classifier based on decomposition tree.

1. Launch RSES and create new project.
2. Insert new table to project.

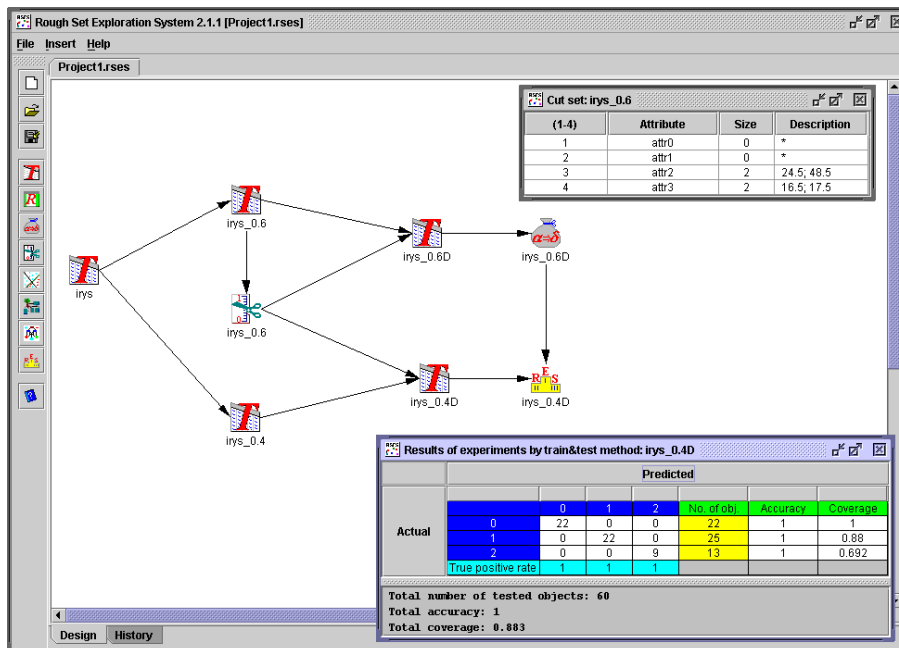


Figure 5.2: RSES after train-and-test scenario with use of rule based classifier and discretization

3. Load data into table object (Load option from context menu); in our example we have chosen `sat_trn` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. We create a decomposition tree for table `sat_trn`. This is done by selecting Make decomposition option from its context menu. We set control values in the next window: data from `sat_trn`; resulting tree to new tree object named `sat_trn`; discretization is on (checkbox is selected); 500 as maximum size of leaf; Shortening ratio set to 1.0 (no shortening). As a result new tree object `sat_trn` is created in project.
5. We insert new table to project. This table will contain test sample.
6. We load data into table object (Load option from context menu); in our example we have chosen `sat_tst` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
7. Now we can test the constructed tree. To do that we choose Classify/Test table using decomposition tree from the context menu associated with `sat_tst` table. In control window that pops up we set value of Decomposition tree from set to `sat_trn`, value of Results to `sat_tst`

and we choose Standard voting as conflict resolution method. Additionally, we select Generate confusion matrix in General test mode field. New result object `sat_tst` is created.

8. We can now examine classification results and decomposition tree in separate windows. For that we double click on `sat_tst` result set and tree object `sat_trn`, respectively.

Figure 5.3 shows RSES after this scenario is finished.

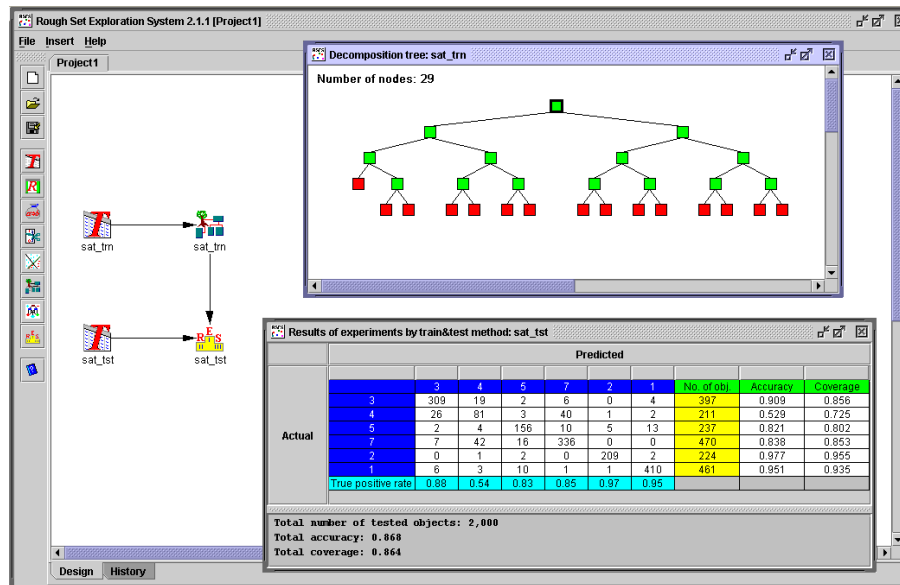


Figure 5.3: RSES after train-and-test scenario with use of decomposition tree

5.1.4 k-NN classifier

For many data sets the k Nearest Neighbor classifier produces very reasonable results. RSES contains the algorithms for construction of k-NN classifiers. These algorithms utilize a version of k-NN that is enriched with several sophisticated methods for selecting the right distance measure and weighting scheme (see subsection 4.6). Moreover, thanks to implementation of some clever algorithms for data decomposition and representation, the version of k-NN that RSES delivers is capable of dealing with relatively large data sets.

Here we would like to present typical brief scenario for using a classifier based on k-NN approach.

1. Launch RSES and create new project.

2. Insert new table to project. This will be our training table.
3. Load data into table object (Load option from context menu); in our example we have chosen `sat_trn` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. Now we insert new table into project. This will be our test table. We load data to this table using Load option from context menu and selecting `sat_tst` file from DATA folder in main RSES installation directory.
5. We test the table `sat_tst` using the training sample `sat_trn`. Note, that there is no separate classifier learning phase in case of constructing k-NN classifier. To perform test we open context menu for the table `sat_tst` and select Classify/Test table using k-NN option. In the window that appears we set Train table from to `sat_trn` and Confusion matrix to to `sat_tst`. We also have to set few control parameters for the k-NN algorithm. In our example we set Metric type->City SVD and Normalization->Range. To speed up calculation we select in this window Attribute weighting->None, Voting->Simple, Number of neighbours->1 and deactivate Search optimal between 1 and ... option. After we click OK a new result object named `sat_tst` emerges in the project.
6. Now we can examine classification results by double-clicking on `sat_tst` result object.

Figure 5.4 shows the outlook of RSES after this scenario is finished.

5.1.5 LTF Classifier

The empirical study has shown that LTF-C (*Local Transfer Function Classifier*) based on artificial neural network produces encouraging results for many data sets, especially in case of predominantly numerical attributes (see [24]). The LTF-C's in RSES may be treated as an alternative to classical approaches based on rough sets, especially in situations where those latter are only partially applicable.

Here we would like to present typical brief scenario for constructing and using a classifier based on LTF neural network.

1. Launch RSES and create new project.
2. Insert new table to project. This will be our training table.

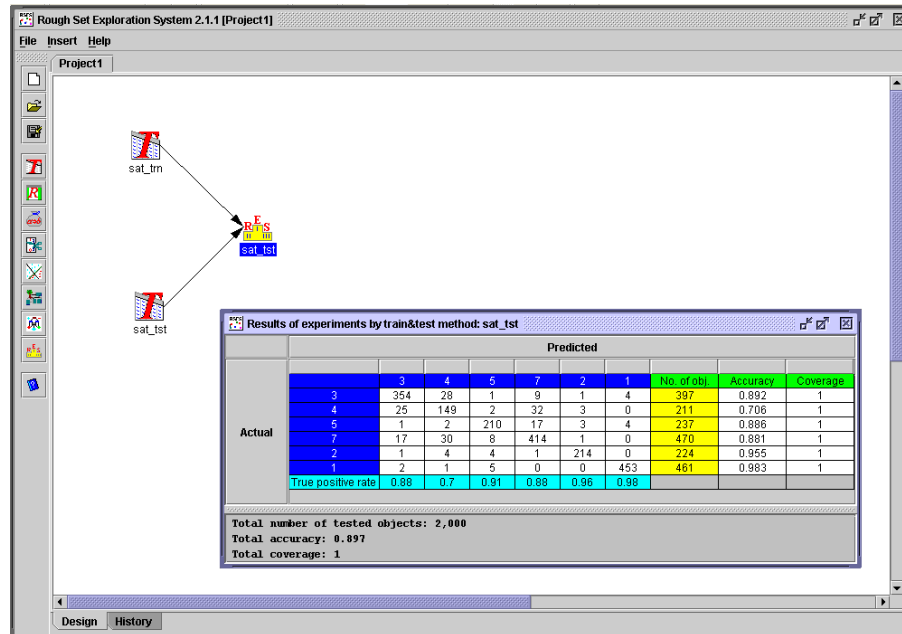


Figure 5.4: RSES after scenario with use of k-NN classification method

3. Load data into table object (Load option from context menu); in our example we have chosen `sat_trn` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. We create a LTF-C for table `sat_trn`. This is done by selecting Create LTF-C option from it's context menu. We set (default) control values in the next window: data from `sat_trn`; resulting LTF-C to new object named `sat_trn`; Use default training parameters is on; Normalize each numeric attribute in active. As a result (after a while) new LTF-C object `sat_trn` is created in project.
5. We insert new table to project. This table will contain test sample.
6. We load data into table object (Load option from context menu); in our example we have chosen `sat_tst` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
7. Now we can test our LTF-C. To do that we choose Classify/Test table using LTF-C from the context menu associated with `sat_tst` table. In control window that pops up we set value of LTF-C from to `sat_trn` and value of Results to to `sat_tst`. Additionally, we select Generate confusion matrix in General test mode field. New result object `sat_tst` is created.

8. We can now examine classification results by double clicking on `sat_tst` result set.

Figure 5.5 presents the outlook of RSES after the finish of scenario with use of LTF-C.

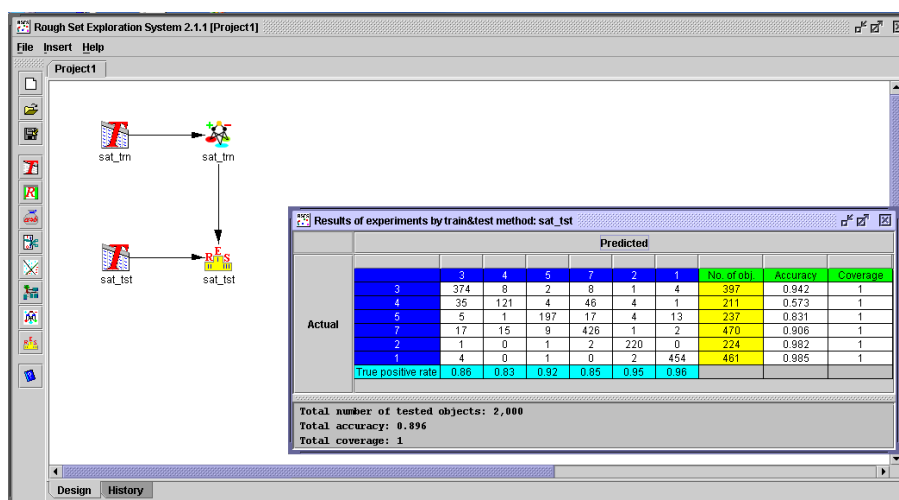


Figure 5.5: RSES after scenario with use of LTF-C

5.2 Testing with use of cross-validation

For many data sets, especially if there are no more than 1000 objects in the table, we may use cross-validation to evaluate quality of learned model (see subsection 4.8) effectively.

Cross-validation in RSES comprises of several training and testing runs. The data set is first split into several, possibly equal in size, disjoint parts. Then, one of the parts is taken as training sample and the remainder (sum of all other parts) becomes the test sample. The classifier is constructed with use of training sample and its performance is checked on test sample. These steps are repeated as many times as there are data parts, so that each of parts is used as training set once. The final result of cross-validation procedure is the average of scores from subsequent steps.

Here we present a brief scenario for evaluating a classifier based on a set of decision rules. In general, each classification method implemented in RSES can be used in this place.

1. Launch RSES and create new project.

2. Insert new table to project.
3. Load data into table object (Load option from context menu); in our example we have chosen `heart` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. Now we perform cross-validation test. For that we select **Classify/Cross validation method** from the context menu of the `heart` table object. In the window that pops up we select **Decision rules** as value of **Type of classifier** and set **Number of folds** to 10. As we have chosen to use decision rules, we need to set some parameters of rule-based classifier to be tested. Using the checkbox we activate discretization and select global discretization method by clicking on **Change discretization method** button and selecting global method in the dialog box that appears. With use of **Change parameters of rule generation** button we may alter the way the rules are calculated, with use of **Change shortening ratio** – the level of rule shortening, and with use of **Change method of resolving conflicts** – the scheme for voting among conflicting rules. In our example all the values are set to default values of **Exhaustive algorithm**, 1.0, and **Standard voting**, respectively. Finally, we enter `heart` in **Results to field**. The **Parameters** field in the bottom left part of the window keeps us informed about the current setting. After clicking **OK** button the cross-validation procedure commences and as a result we see new result object `heart`. See figure 5.6.
5. We can now examine the results by double clicking on `heart` result set.

5.3 Scenario for decision generation

To this end all the described scenarios used data tables with already existing decision column. In practical problems we are frequently facing a different problem. We may be given some set of labeled objects – the training sample, but what we want to do is to classify a set of objects for which the decision value is not known. In other words, we have to generate the decision column for the table that does not have one. RSES may be used for this purpose. It may assign objects to decision classes (produce decision column for a table) using a classifier created beforehand. The decision value generated by RSES is appended to an objects at the last position. In this way the decision generated by system is always stored in newly created last column in resulting data table.

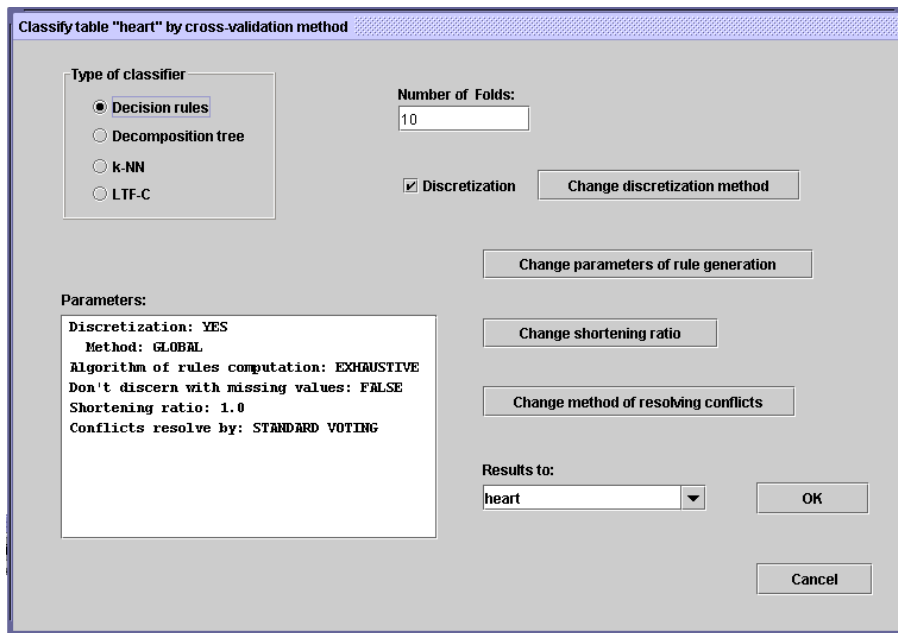


Figure 5.6: Example of parameter setting for cross-validation method

In order to present the capability of generating decision column in RSES we will need two tables. First of them will be the training table (training sample), for the second we will be generating decision attribute. We will use the RSES' capability of subtable selection in order to perform the following scenario. This scenario is very simplistic and meant only as an example of using selected RSES capabilities.

1. Launch RSES and create new project.
2. Insert new table to project.
3. Load data into table object (Load option from context menu); in our example we have chosen `heart` data file from the data sets distributed with RSES (DATA folder in main RSES installation directory).
4. From `heart` table we extract subtable that contains no decision column. To do that we select **Select subtable** option from it's context menu. In the window which appears we select all but last attributes (columns), i.e. only `attr13` is not selected. Once we confirm our choice of attributes, a new table object named `heart_SUB` appears in the project.
5. Now we need to create the classifier. For that we calculate decision rules for table `heart` (from the context menu for this table we select

Reducts/Rules / Calculate reducts or rules option). We select **Exhaustive algorithm** from rule calculation control window. A new object is being created – a set of rules with the same name as originating table, i.e. **heart**.

6. Now we apply calculated rules to **heart_SUB** in order to generate decision column. To do that we perform testing without generating confusion matrix. Namely, we select **Classify/Test table using rule set** option from the context menu of **heart_SUB** table object. In the window that appears we have to select **Classify new cases** as the value of **General test mode**, **Standard voting** as method of conflict resolution, and set **Results to field** to **heart_SUB**. As a result (after confirming with **OK**) the new, decision column is appended at the end of **heart_SUB** table.

In this simple example we have calculated all rules for indiscrepant data table. Therefore, the decision columns in **heart_SUB** and **heart** tables should be identical after this simple experiment, as both tables contain the very same set of objects.

Appendix A

Selected RSES 2.2 file formats

This appendix contains a short description of the essential file formats used by RSES 2.2 to store information.

In order to improve readability of listings extra spacing have been introduced. Extra spaces have no impact on the way RSES 2.2 reads and interprets contents of it's files.

A.1 Data sets

Data sets to be user in analysis with use of RSES 2.2 should follow the format presented in figure A.1.

Table and attribute names as well as attribute values may contain spaces. But, if such string contains space, it must be placed inside quotes or double quotes. For instance, name of attribute *serum cholestoral* should be entered as "*serum cholestoral*" or '*serum cholestoral*'. The RSES can accept characters that are defined in ISO-8859-1 (Western) and ISO-8859-2 (CEE-Latin2) standards.

An example of data set is presented in figure A.1. First row that shows:

```
TABLE therapy
```

contains information about the name of the table. Unique name assigned to data set simplifies management of entities in RSES project.

Second row stating:

```
ATTRIBUTES 5
```

provides us with information on the number of attributes (columns) in our data, including decision attribute. In the following rows we have definition of attributes:

```

TABLE therapy
ATTRIBUTES 5
  temperature numeric 1
  headache numeric 0
  cough symbolic
  catarrh symbolic
  disease symbolic
OBJECTS 4
38.7   7      no no  angina
38.3   MISSING yes yes influenza
MISSING 3      no no  cold
36.7   1      no no  healthy

```

Figure A.1: Example of RSES 2.1 data set

```

temperature numeric 1
headache numeric 0
cough symbolic
catarrh symbolic
disease symbolic

```

Each row describes a single attribute (column). The description for each attribute consists of attribute's name and type. There are two possible types of attributes – **symbolic** and **numeric**. Obviously, **symbolic** is used to mark the attributes that have limited set of symbolic (e.g. string) values. For numerical attributes type declaration is followed by number which determines the number of digits in decimal expansion of attribute values. For instance, **numeric 1** describes numerical attribute that has one digit after dot while **numeric 0** corresponds to attribute with integer values.

The line below attribute descriptions contains information on the number of objects (rows) in data table .

```
OBJECTS 4
```

After that objects (rows) follow. The values in rows correspond to attribute types described before. ¹.

¹In this example extra spaces are added to increase readability. These spaces have no influence on the way RSES interprets input data set.

38.7	7	no	no	angina
38.3	MISSING	yes	yes	influenza
MISSING	3	no	no	cold
36.7	1	no	no	healthy

The `MISSING` symbol denotes missing attribute value i.e., the place in data table where attribute value is not known, Such a missing value may also be denoted using `NULL` or `'?'` symbols.

The default extension for RSES file containing data is `.tab`.

A.2 Reduct sets

RSES 2.2 can store sets of reducts in text files. Example contents of such file are shown in figure A.2.

```

REDUCTS (5)
{ temperature, headache } 1.0
{ headache, cough } 1.0
{ headache, catarrh } 1.0
{ cough } 0.25
{ catarrh } 0.25

```

Figure A.2: Set of reducts in RSES 2.1 format

In this example the first row:

```
REDUCTS (5)
```

contains the number of reducts stored in this file. In our case there are five reducts.

Subsequent rows contain reducts (one per row).

```

{ temperature, headache } 1.0
{ headache, cough } 1.0
{ headache, catarrh } 1.0
{ cough } 0.25
{ catarrh } 0.25

```

Description of each reduct (each row) consists of list of attributes placed between { and } and the number. The number shows the positive region for the table reduced to only those listed attributes.

In our example first reduct contains two attributes: `temperature` and `headache`. The value of positive region for this reduct is 1.0 which means that the reduced table – with original decision attribute added – is consistent (contain no discrepancies).

The default extension for RSES file containing reducts is `.red`.

A.3 Rule sets

Figure A.3 presents example contents of file that is used in RSES 2.2 to store decision rules.

First row,

```
RULE_SET Demo
```

contains a unique name of rule set. This name is used to identify rule set within project.

Subsequent rows contain attribute information. This information is essential if rules are to be applied to some data set. Before checking the data against rules, the system have to verify if attributes in the data are identical (in terms of format) to those appearing in rules. The description of attributes follow the format already described for data files (see subsection A.1).

```
ATTRIBUTES 5
temperature numeric 1
headache numeric 0
cough symbolic
catarrh symbolic
disease symbolic
```

Right below rows with (conditional) attribute description appears information pertaining the decision attribute, i.e. the last attribute (column) in decision table.

```
DECISION_VALUES 4
angina
influenza
cold
healthy
```

```

RULE_SET Demo
ATTRIBUTES 5
  temperature numeric 1
  headache numeric 0
  cough symbolic
  catarrh symbolic
  disease symbolic
DECISION_VALUES 4
angina
influenza
cold
healthy
RULES 16
(temperature=38.7)&(headache=7)=>(disease=angina[1]) 1
(temperature=38.3)&(headache=MISSING)=>
(disease=influenza[1]) 1
(temperature=MISSING)&(headache=3)=>(disease=cold[1]) 1
(temperature=36.7)&(headache=1)=>(disease=healthy[1]) 1
(headache=7)&(cough=no)=>(disease=angina[1]) 1
(headache=MISSING)&(cough=yes)=>(disease=influenza[1]) 1
(headache=3)&(cough=no)=>(disease=cold[1]) 1
(headache=1)&(cough=no)=>(disease=healthy[1]) 1
(headache=7)&(catarrh=no)=>(disease=angina[1]) 1
(headache=MISSING)&(catarrh=yes)=>(disease=influenza[1]) 1
(headache=3)&(catarrh=no)=>(disease=cold[1]) 1
(headache=1)&(catarrh=no)=>(disease=healthy[1]) 1
(cough=no)=>(disease={angina[1],cold[1],healthy[1]}) 3
(cough=yes)=>(disease=influenza[1]) 1
(catarrh=no)=>(disease={angina[1],cold[1],healthy[1]}) 3
(catarrh=yes)=>(disease=influenza[1]) 1

```

Figure A.3: Example set of rules in RSES 2.1 format

First, the number of possible decision values is declared. Then, admissible decision values are listed, one in each row.

The remainder of rule set file stores the actual rules. In this part the number of stored rules is declared first.

RULES 16

In our example 16 rules are present. They are listed in next rows (one per row).

```
(temperature=38.7)&(headache=7)=>(disease=angina[1]) 1
(temperature=38.3)&(headache=MISSING)=>
(disease=influenza[1]) 1
(temperature=MISSING)&(headache=3)=>(disease=cold[1]) 1
(temperature=36.7)&(headache=1)=>(disease=healthy[1]) 1
...
(catarrh=no)=>(disease={angina[1],cold[1],healthy[1]}) 3
...
```

In our example the first rule states that if a patient has a mild fever of 38.7C and serious headache (value of `headache` attribute is 7) then his sickness is recognized as `angina` and there exist one case that support this rule. By supporting we understand the objects in training data that match at the conditional part of rule and have the decision identical with that suggested by rule. The number of supporting objects is written at the end of each rule.

In case of generalized rules there may be several decision values listed after the => sign, as in:

```
...
(catarrh=no)=>(disease={angina[1],cold[2],healthy[1]}) 4
...
```

In this case the supports for each decision value is given in square brackets after this value, and total (summarized) support appears at the end of row. In our example `angina[1]` means that there is one case (out of four that match the rule) which supports this decision.

The default extension for RSES file containing reducts is `.rul`.

A.4 Set of cuts

Figure A.4 presents example contents of file that is used in RSES 2.2 to store cuts used in discretization/grouping.

The first row:

```
CUT_SET demo_global
```

```

CUT_SET demo_global
ATTRIBUTES 4
INCLUDED_SYMBOLIC false
temperature numeric 1
[ 38.5 ]
headache numeric 0
[ 5.0 ]
cough symbolic
[ ]
catarrh symbolic
[ ]

```

Figure A.4: Example of file used to store cuts in RSES 2.1

declares a unique name for this cut set.

Second row:

```
ATTRIBUTES 4
```

informs us about the number of attributes to be discretized.

Third row:

```
INCLUDED_SYMBOLIC false
```

states whether the symbolic attribute values are grouped. This option (corresponding to `true`) is only available in case of local discretization method.

Next rows contain information about attribute name, type and about cuts that have been generated for this attribute. For each attribute there are two rows assigned. The first contains attribute description, the second - cuts.

```

temperature numeric 1
[ 38.5 ]
headache numeric 0
[ 5.0 ]
cough symbolic
[ ]
catarrh symbolic
[ ]

```

Empty brackets [] denote lack of cuts for corresponding attribute, i.e. this attribute will not undergo discretization.

In case of numerical attribute a cut is an attribute value (threshold) that splits the attribute domain into two sets. Using several such threshold makes it possible to decompose the set of values of numerical attribute into several disjoint parts.

In our example the attribute `temperature` is discretized with use of a single cut, i.e. the attribute value space is decomposed into two smaller intervals. First interval contains all values that are greater than 38.5, the other those below 38.5.

In case of symbolic attributes we can find cuts (decompositions) only if we use local discretization method with active `Include symbolic attributes` option. The local discretization method can decompose set of values of a symbolic attribute into two disjoint subsets. Therefore, to fully describe such decomposition it suffices to know only one of resulting subsets, as the other is simply a complement. In figure A.5 we present a set of cuts generated for the same decision table as in figure A.4. The difference is in the method chosen for cut generation (global vs. local) and inclusion of symbolic attributes.

```
CUT_SET demo_local
ATTRIBUTES 4
INCLUDED_SYMBOLIC true
temperature numeric 1
[ 37.5 ]
headache numeric 0
[ 2.0 5.0 ]
cough symbolic
[ { MISSING,no } ]
catarrh symbolic
[ ]
```

Figure A.5: Example of file for a set of cuts obtained by local method with inclusion of symbolic attributes

Note, that attribute `cough` is decomposed into two subsets. The first subset gathers the values `MISSING` and `no`, the other – all the remaining values for this attribute. Also, compared with figure A.4, the cut for `temperature` has changed and attribute `headache` is now decomposed into three subsets.

This shows the difference between global and local method for cut generation.

The default extension for RSES file containing cuts is `.cut`.

A.5 Linear combinations

Figure A.6 presents example contents of file that is used in RSES 2.2 to store linear combinations used in creation of new attributes.

```
DIRECTIONS (4)
temperature*0.707+headache*0.707
temperature*0.705+headache*0.705+disease*(-0.062)
temperature*0.707+headache*0.707
temperature*0.696+headache*0.696+disease*0.174
```

Figure A.6: Example of file containing linear combinations

First row:

```
DIRECTIONS (4)
```

informs us that this file is used to store linear combinations (directions) and how many such combinations appear in this file (in our case 4).

Subsequent rows list these linear combinations, one per row.

```
temperature*0.707+headache*0.707
temperature*0.705+headache*0.705+disease*(-0.062)
temperature*0.707+headache*0.707
temperature*0.696+headache*0.696+disease*0.174
```

Description of each combination is presented as a formula and may used to create a new attribute.

The default extension for RSES file containing linear combinations (directions) is `.dir`.

A.6 LTF-C

The file used to store a LTF classifier (LTF-C) is quite large and complicated, as it contains a lot of information. Therefore, we will describe various parts of it separately. Linear Transfer Function (LTF) classifier is based on special kind of Artificial Neural Network. Some description of LTF-C and its parameters is given in section 3.7.

The description of subdivides into: header, network parameters, description of neurons, and extended description of attributes from analyzed table.

The header contains exactly two rows. The first contains classifier's name, in our case `demo`. Second row in header give the number of lines (rows in file) used to describe the classifier, in our case 62.

```
LTF_CLASSIFIER demo
62
```

Next part provides the description of network architecture input as a comment. They are put here in order to provide the user with easy accessible description right from the beginning. The information contained in this comment is repeated further in the file, but in less readable form. Note, that `$` sign denotes the beginning of the comment line.

```
$-----
$ LTF-C neural network
$ Number of inputs: 2
$ Number of outputs: 4
$ Number of neurons: 1
$-----
```

The value of each parameter is preceded by `@` and parameter's name. Of all the parameters (from `@FileType` to `@SumCycles`) those really important for the user are `@EtaUse` and `@UseTr`.

```
@FileType net
...
@EtaUse 0.013
@UseTr 0.013
...
@SumCycles 120
```

Next part gives the description of neurons. The details are given in section 3.7.

```
@<Neuron0
  @Class    1
  @Life     0
  ...
@>
```

The last part of LTF-C file contains information about training data that was used to construct it. This include description of decision attribute (name and type) and its values (number of values and their names). Also, for numerical conditional attributes the file stores information whether these attributes were normalized or not (true/false). For all numerical attributes the mean values and standard deviations on training data set are stored. These values form two blocks. Each block start with number of attributes, in our example 4.

```
disease symbolic
DECISION_VALUES 4
angina
influenza
cold
healthy
true
4
37.73333333333333
3.0
0.0
0.0
4
0.8617811013631386
2.1908902300206643
0.0
0.0
```

The default extension for RSES file containing LTF classifier is `.ltf`.

A.7 Classification results

Figure A.7 presents an example contents of a file used to store results of classification in RSES 2.1. This file stores almost exactly the same information as the result view window (see figure 3.36)

```

TEST RESULTS:
  Global coverage=0.8333333333333334
  Global accuracy=0.8333333333333334
Decision classes:
  angina influenza cold healthy
Confusion matrix:
  0.0 0.0 0.0 0.0
  0.0 0.6666666666666666 0.0 0.0
  0.0 0.3333333333333333 0.0 0.0
  0.0 0.0 0.0 0.6666666666666666
True positive rates for decision classes:
  0.0 0.5 0.0 0.6666666666666666
Accuracy for decision classes:
  0.0 0.6666666666666666 0.0 0.6666666666666666
Coverage for decision classes:
  0.0 0.6666666666666666 0.3333333333333333 0.6666666666666666

```

Figure A.7: RSES 2.1 classification result file contents

The very first row (TEST RESULTS:) states the purpose of this file.

Next two rows present values of coverage and accuracy for entire testing data set.

```

Global coverage=0.8333333333333334
Global accuracy=0.8333333333333334

```

Then, all admissible values of decision (decision class names) are listed.

```

Decision classes:
  angina influenza cold healthy

```

Next part of the file presents complete confusion matrix.

```

Confusion matrix:
  0.0 0.0 0.0 0.0
  0.0 0.6666666666666666 0.0 0.0
  0.0 0.3333333333333333 0.0 0.0
  0.0 0.0 0.0 0.6666666666666666

```

Finally, some statistics for decision classes are presented. For each collection of values a couple of rows is used. First row in each couple names the kind of measurement, the second brings the values. The order of values corresponds to the order of decision classes defined earlier in the file.

True positive rates for decision classes:

0.0 0.5 0.0 0.6666666666666666

Accuracy for decision classes:

0.0 0.6666666666666666 0.0 0.6666666666666666

Coverage for decision classes:

0.0 0.6666666666666666 0.3333333333333333 0.6666666666666666

The default extension for RSES file containing classification results is `.res`.

Bibliography

- [1] J. Bazan, Son H. Nguyen, Trung T. Nguyen, A. Skowron and J. Stepaniuk (1998): Decision rules synthesis for object classification. In: E. Orłowska (ed.), *Incomplete Information: Rough Set Analysis*, Physica – Verlag, Heidelberg, pp. 23–57.
- [2] J. Bazan (1998): A Comparison of Dynamic and non-Dynamic Rough Set Methods for Extracting Laws from Decision Table. In: L. Polkowski, A. Skowron (eds.), *Rough Sets in Knowledge Discovery*, Physica – Verlag, Heidelberg, pp. 321–365.
- [3] J. Bazan (1998): Metody wnioskowań aproksymacyjnych dla syntezy algorytmów decyzyjnych. Ph. D. thesis, supervisor A. Skowron, Warsaw University, pp. 1–179. (In Polish only)
- [4] J. Bazan, M. Szczuka (2000): RSES and RSESlib – A Collection of Tools for Rough Set Computations. *Lecture Notes in Artificial Intelligence* 3066, 592–601, Berlin, Heidelberg: Springer-Verlag.
- [5] J. Bazan, H. S. Nguyen, S. H. Nguyen, P. Synak, and J. Wróblewski (2000): Rough set algorithms in classification problem. In L. Polkowski, S. Tsumoto, and T. Lin, editors, *Rough Set Methods and Applications*, Physica-Verlag, Heidelberg New York, pp. 49–88.
- [6] J. Bazan, M. Szczuka, J. Wróblewski (2002): A New Version of Rough Set Exploration System, *Lecture Notes in Artificial Intelligence* 2005, 106–113, Berlin, Heidelberg: Springer-Verlag.
- [7] J. Bazan, M. Szczuka, A. Wojna, M. Wojnarski (2004): On Evolution of Rough Set Exploration System, *Lecture Notes in Artificial Intelligence* 3066, 592–601, Berlin, Heidelberg: Springer-Verlag.
- [8] G. Gora, A. Wojna (2002): RIONA: A Classifier Combining Rule Induction and k-NN Method with Automated Selection of Optimal Neighbourhood, *Proceedings of the Thirteenth European Conference on Machine*

Learning, ECML 2002, Helsinki, Finland, Lecture Notes in Artificial Intelligence, 2430, Springer-Verlag, pp. 111–123

- [9] G. Gora, A. Wojna (2002): RIONA: A New Classification System Combining Rule Induction and Instance-Based Learning, *Fundamenta Informaticae*, 51(4), pp. 369–390
- [10] J. Grzymala-Busse (1997): A New Version of the Rule Induction System LERS *Fundamenta Informaticae*, Vol. 31(1), pp. 27–39
- [11] J. Grzymala-Busse and M. Hu (2000): 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, October 16–19, 2000, Banff, Canada, 340–347.
- [12] D. Michie, D. J. Spiegelhalter, C. C. Taylor (1994): *Machine learning, neural and statistical classification*. Ellis Horwood, New York.
- [13] Son H. Nguyen and A. Skowron (1997). Quantization of real value attributes: Rough set and boolean reasoning approach. *Bulletin of International Rough Set Society* 1/1, pp. 5–16.
- [14] Son H. Nguyen (1997). Discretization of real value attributes. Boolean reasoning approach. Ph. D. thesis, supervisor A. Skowron, Warsaw University
- [15] Son H. Nguyen, Hoa S. Nguyen (1998). Discretization Methods in Data Mining. In: L. Polkowski, A. Skowron (eds.): *Rough Sets in Knowledge Discovery*. Physica-Verlag, Heidelberg, pp. 451–482.
- [16] Hoa S. Nguyen, H. Son Nguyen (1998). Pattern extraction from data. *Fundamenta Informaticae* 34/1-2, pp. 129–144.
- [17] Son H. Nguyen (1998). From Optimal Hyperplanes to Optimal Decision Trees. *Fundamenta Informaticae* 34/1–2, pp. 145–174.
- [18] Hoa S. Nguyen, A. Skowron and P. Synak (1998). Discovery of data patterns with applications to decomposition and classification problems. In: L. Polkowski and A. Skowron (eds.), *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, Physica-Verlag, Heidelberg, pp. 55–97.

- [19] Hoa S. Nguyen (1999). Discovery of generalized patterns. Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems, Foundations of Intelligent Systems (ISMIS'99), June 8–11, Warsaw, Lecture Notes in Artificial Intelligence 1609, Springer-Verlag, Berlin, pp. 574–582.
- [20] Hoa S. Nguyen (1999). Data regularity analysis and applications in data mining. Ph. D. thesis, supervisor B. Chlebus, Warsaw University.
- [21] A. Øhrn, J. Komorowski (1997): ROSETTA – A rough set tool kit for analysis of data, *Proceedings of the Fifth International Workshop on Rough Sets and Soft Computing (RSSC'97) at the Third Joint Conference on Information Sciences (JCIS'97)*, Research Triangle Park, NC, March 2–5 (1997) 403–407.
- [22] Z. Pawlak (1991): *Rough sets: Theoretical aspects of reasoning about data*. Dordrecht: Kluwer.
- [23] D. Ślęzak, J. Wróblewski (1999). Classification Algorithms Based on Linear Combinations of Features. Proc. of PKDD'99, Prague, Czech Republic, Springer-Verlag (LNAI 1704), Berlin Heidelberg 1999, pp. 548–553.
- [24] M. Wojnarski (2003): LTF-C: Architecture, Training Algorithm and Applications of New Neural Classifier. *Fundamenta Informaticae*, Vol. 54(1), pp. 89–105. IOS Press, 2003
- [25] J. Wróblewski (1998). Genetic algorithms in decomposition and classification problem. In: L. Polkowski and A. Skowron (eds.), *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, Physica-Verlag, Heidelberg, pp. 471–487
- [26] ROSETTA Homepage <http://rosetta.lcb.uu.se/general/>
- [27] RSES Homepage <http://logic.mimuw.edu.pl/~rses>

Index

- attribute
 - cuts, 27, 50
 - decision, 84
 - description, 82
 - new, 39
 - numeric, 82
 - statistics, 28
 - symbolic, 82
 - type, 82
- classification
 - storage format, 91
- computations, 23
- confusion matrix, 47
 - storage format, 92
- cross-validation, 27, 65
- cuts, 37
 - global method, 89
 - local method, 88, 89
 - numerical attributes, 88
 - storage format, 86
 - symbolic attributes, 88
- data, 23
 - examples, 8
 - format, 82
 - MISSING, 49, 83
 - missing values, 49
 - NULL, 49, 83
 - statistics, 28
- data format, 82
- decomposition, 27, 41, 58
- discretization, 27, 37, 50
- distributed environment, 13
- Dixer, 13
- dynamic reducts, 54
- generating cuts, 50
- grouping, 50
- icons, 16
- k-NN, 27, 60
- linear combinations, 27, 39, 51
 - attributes, 89
 - storage format, 89
- LTF-C, 27, 42, 63
 - storage format, 90
- menu
 - context, 16, 19
 - context for object group, 16, 20
 - general, 15, 19
 - icons, 16
 - layout, 16
 - main, 11, 16
 - project's context, 15
- methods
 - description, 49
- MISSING, 49, 83
- missing values, 49
- neuron, 44
- NULL, 49, 83
 - completion, 26
 - in data, 49
 - various approaches, 49
- objects, 14

- moving, 15
- selecting, 16
- project, 11
 - create, 13
 - history, 14, 21
 - project view, 14
 - saving and restoring, 14
- reducts, 27, 32, 52
 - core, 34
 - statistics, 34
 - storage format, 83
- results, 46
 - storage format, 91
- RSES, 5
- RSES-lib, 6
- rules, 27, 34, 52, 86
 - statistics, 36
 - storage format, 84
 - support, 86
- starting RSES, 11
- statistics
 - attribute, 28
 - data, 28
 - reducts, 34
 - rules, 36
- SVD, 61
 - City-SVD, 62
- table, 23
 - attribute description, 82
 - attribute's type, 82
 - format, 82
 - name, 24, 81
 - number of objects, 82
 - statistics, 28
- toolbar, 18
- trees
 - decomposition, 27, 41, 58