ROSETTA Technical Reference Manual[*]

Aleksander Øhrn[†]

Contents

[*]This document is under perpetual development. Last updated May 25, 2001.

[†]Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: aleks@idi.ntnu.no.

Preface

This document is the technical reference manual for ROSETTA [26, 27, 28, 29], a software system I designed and implemented as part of my doctoral dissertation [26]. As with all manuals, it almost certainly contains errors and has plenty of room for improvements. Please report any errors, typos, inconsistencies, omissions and suggestions for improvements to `aleks@idi.ntnu.no`.

The RSES library [38], developed at the Group of Logic at the University of Warsaw [17], is embedded as an optional component of the ROSETTA computational kernel. The names of these algorithms are in this document all prefixed with the string "RSES".

1 Introduction

This document constitutes the technical reference manual for the ROSETTA software system. The main algorithms and their options are briefly described, and appropriate references are given to the literature where relevant.

This document should not be read in isolation. The reader is referred to Øhrn [26, chapters 4–7] and references therein for an overview of the background theory, as the same terminology and notation is employed in this manual. Furthermore, it is assumed that the reader has a basic familiarity with the knowledge discovery and data mining process, and how a typical empirical modelling project is carried out. A small example ROSETTA case study can be found in [26, chapter 9], outlining how ROSETTA can be used to, e.g., induce and validate classification rules from a database.

Miscellaneous tips and take-home points are throughout this document indented and marked with a special symbol. An example is the following:

TIP News, tips, software updates and other relevant information can be found at the ROSETTA website [35]. To obtain source code of portions of the ROSETTA computational kernel, consult the ROSETTA C++ Library website [36].

Information under the **Name**, **Keywords** and **Signature** labels in this document are mostly relevant in the context of command scripts. Command scripts are described in Section 4.10 and Appendix A.4.

In the ROSETTA GUI, many of the same menu entries appear for different types of objects. For example, all objects have an **Annotations...** entry in their pop-up menus. Such multiply occurring entries are only listed once in this document. Also, not all menu entries are documented here, only the less self-explanatory ones.

2 GUI Preliminaries

An overview of the ROSETTA GUI is given by Øhrn [26, chapter 8]. The following ROSETTA GUI details are worth noting:

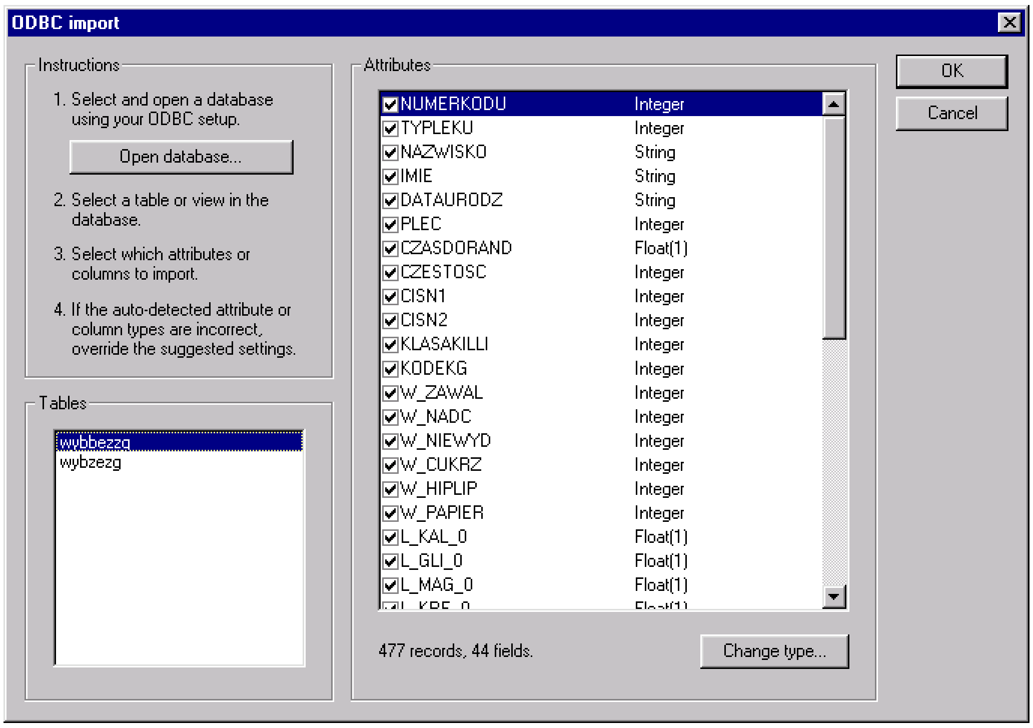
- A decision system can be read into a new ROSETTA project by selecting **Open...** from the main **File** menu, and will be placed immediately below the root of the **Structures** node in the project tree.
- Branches in the project tree can be expanded or collapsed by left-clicking on the "⊞" or "⊟" symbols next to the icons.
- Right-clicking on an icon in the project tree brings up a pop-up menu for that object. In the following, the symbol "▶" will be used to denote menu navigation.
- Left-clicking twice on an icon in the project tree can be used as a shortcut for viewing that object in detail.
- Grayed columns in views of decision systems indicate that the corresponding attributes are "masked away" and subsequently ignored by the ROSETTA kernel in any analysis steps. Missing values are indicated by the string **Undefined**.
- Rules can be sorted directly in their views by right-clicking the column to sort by.
- To rename an object, first left-click once on its icon to select it. Then left-click once more on the icon's label. The icon's label is edited directly in place.
- To view progress messages and warnings, select **Messages** from the main **View** menu. A document has to be present before the **View** menu appears.

3 Projects

P

Name: *Project*

Description: A project object is the top-level structural object in a project tree. A root project cannot be deleted from the GUI. It is possible for a project to have subprojects.



ODBC import

Instructions

1. Select and open a database using your ODBC setup.

Open database...

2. Select a table or view in the database.
3. Select which attributes or columns to import.
4. If the auto-detected attribute or column types are incorrect, override the suggested settings.

Tables

- wybbezzg
- wybzezg

Attributes

☑ NUMERKODU	Integer
☑ TYPLEKU	Integer
☑ NAZWISKO	String
☑ IMIE	String
☑ DATAURODZ	String
☑ PLEC	Integer
☑ CZASDORAND	Float(1)
☑ CZESTOSC	Integer
☑ CISN1	Integer
☑ CISN2	Integer
☑ KLASAKILLI	Integer
☑ KODEKG	Integer
☑ W_ZAWAL	Integer
☑ W_NADC	Integer
☑ W_NIEWYD	Integer
☑ W_CUKRZ	Integer
☑ W_HIPLIP	Integer
☑ W_PAPIER	Integer
☑ L_KAL_0	Float(1)
☑ L_GLI_0	Float(1)
☑ L_MAG_0	Float(1)

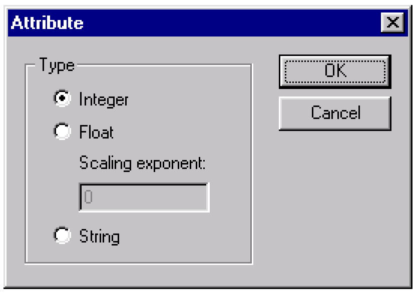
477 records, 44 fields.

Change type...

OK

Cancel

Figure 1: Dialog box for ODBC import of decision tables.



Attribute

Type

[✓ Integer]

Float

Scaling exponent: 0

String

OK

Cancel

Figure 2: Dialog box for specifying attribute types.

3.1 P▶ODBC...

Name: *ODBCDecisionTableImporter*

Description: Allows tabular data to be imported from a wide variety of data sources such as spreadsheets or relational databases, by means of ODBC.[1] Which data sources that are available depends on which ODBC drivers that are installed on your system.

TIP ODBC drivers for most popular database management systems are freely available on the Internet.

Dialogs: Figure 1, Figure 2.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* ⟶ *DecisionTable*

[1]Open Database Connectivity. ODBC is an open, vendor-neutral interface for database connectivity that provides access to a wide variety of computer systems. The ODBC programming interface enables applications to access data in database management systems (DBMSs) that use SQL as a data access standard. This enables developers to not have to target a specific DBMS. Instead, users can add modules called database drivers that link the application to their choice of DBMS.

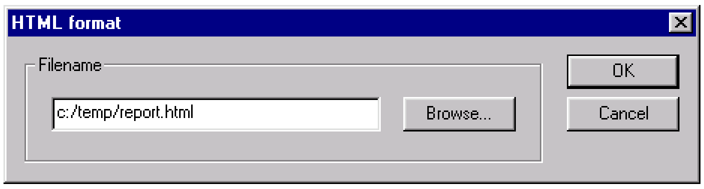


Figure 3: Dialog box for specifying filenames.

3.2 P▶Report

Name: *Reporter*

Description: Algorithms in this family export information and meta-information about the project as a whole. The algorithms function as simple identity pass-through routines, where the exporting is a side-effect of applying the algorithm.

3.2.1 P▶Report▶XML format...

Name: *XMLReporter*

Description: Creates a report in XML format that contains the annotations associated with the project, and basic information about all structures that are members of the project.

TIP XML greatly increases interoperability with other programs, and you can transform the XML into other formats using the XSLT language. Many current browsers also support viewing of XML documents.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Structure* $\longrightarrow$ *Structure*

3.2.2 P▶Report▶HTML format...

Name: *HTMLReporter*

Description: Creates a report in HTML format that contains the annotations associated with the project, and all structures that are members of this. A tree displaying the derivation interrelationships is also included in the report. The resulting report contains hyperlinks.

Dialogs: Figure 3.

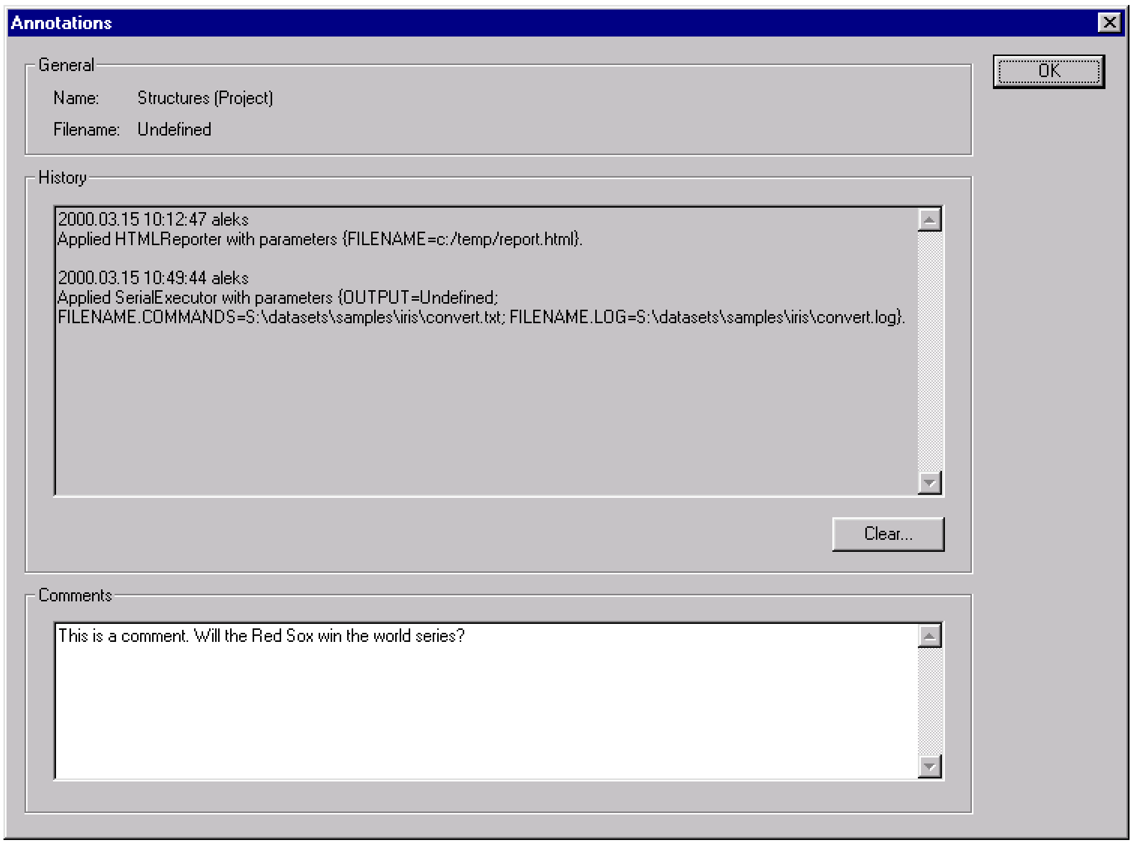
Keywords: FILENAME (*String*).

Signature: *Structure* $\longrightarrow$ *Structure*

3.3 P▶Annotations...

Description: An annotation includes a general comment field and a history list. Objects that are manipulated and created in the GUI automatically get their history list updated with a timestamp, a user name and an action description. This aids in generating automatic documentation of a ROSETTA session.

An annotation also holds a filename. The object is saved to this location if the **Save...** menu option is invoked.



Structures (Project)
Undefined

```
2000.03.15 10:12:47 aleks
Applied HTMLReporter with parameters {FILENAME=c:/temp/report.html}.

2000.03.15 10:49:44 aleks
Applied SerialExecutor with parameters {OUTPUT=Undefined;
FILENAME.COMMANDS=S:\datasets\samples\iris\convert.txt; FILENAME.LOG=S:\datasets\samples\iris\convert.log}.
```

This is a comment. Will the Red Sox win the world series?

Figure 4: Dialog box for annotations.

TIP In the history list, the set of employed parameters are displayed as a collection of keyword/value pairs. It is the same keywords that are used in command scripts. Hence, to regenerate a computation, it is possible to copy the parameter list into a command script.

Dialogs: Figure 4.

4 Decision Tables

D

Name: *DecisionTable*

Description: Information systems and decision systems are both represented by the same structure. It is up to the algorithms that operate on such objects to define how the table is to be interpreted.

Internally, all value sets for all attributes are represented as integers. Thus, an information system can essentially be perceived as a matrix with integer entries. The mapping between the internal integer representations and their meanings in the modelling domain is handled by a data dictionary, associated with each information system. Dictionaries are described in Section 4.1.

In addition to the attribute values, a decision table object also holds information per attribute of its type (condition or decision) and masking status (enabled or disabled). Attributes that are disabled are "invisible" to the algorithms that operate on the table.

4.1 Dictionaries

Name: *Dictionary*

Description: Each decision table has a dictionary associated with it, responsible for mapping between the integer values used internally and their meanings in the modelling domain.

A data dictionary is composed of several dictionary attributes, one per attribute in the table the dictionary is associated with. A dictionary attribute is responsible for mapping elements in V_a to a string that makes sense to the user and the modelling domain, and vice versa. How this mapping is performed depends on the type of attribute:

- *Integer:* Performs the mapping $v \mapsto$ "v", i.e., the integer v is mapped to its natural string representation.
- *Float:* Performs the mapping $v \mapsto$ "f" where $f = v/10^n$, i.e., the integer v is mapped to the string representation of the float f which corresponds to v with the decimal point moved n places to the left. The integer number n is called the scaling exponent of the attribute.
- *String:* Performs the mapping $v \mapsto \mathit{lookup}(v)$, i.e., the integer v is mapped to a string that is looked up in an associative map.

Inverse mappings are also done by dictionary attributes. An attribute in a dictionary also holds the name and unit of the attribute.

Example: Let $v \in V_a$ and let $v = 125$. If a is an integer attribute, v is mapped to the string "125" and vice versa. If a is a float attribute with $n = 2$, v is mapped to the string "1.25" and vice versa. If a is a string attribute, v might be mapped to, e.g., "One hundred and twenty five" and vice versa.

4.2 D▶View...

Description: Brings up a view of the decision table, enabling the data to be inspected in detail. Gray columns indicate that the attributes are "masked", i.e., made invisible to the algorithms that operate on the table. The decision attribute is indicated in bold face. Currently, the decision attribute is required to be the rightmost attribute in the table. Integer and float attributes are right-justified, string attributes are centered.

By right-clicking the grid origin or headers of columns or rows, context-sensitive pop-up menus appear.

The masking state of attributes can be altered by selecting the columns in question and invoking the **Masking...** menu option from one of the columns' pop-up menu.[2]

TIP Data can be viewed through the dictionary or not. Invoke the **Use dictionary** menu option from the grid origin's pop-up menu to change viewing mode.

TIP To swap the positions of two columns, select the columns in question and select the **Swap** menu option from one of the columns' pop-up menu.

TIP To join two or more attributes together to form a compound attribute, select the columns in question and invoke the **Join** menu option from one of the columns' pop-up menu.

TIP Internally, some data structures such as, e.g., reducts, are simply indices into a parent decision table. Thus, altering the table would invalidate such structures, unless additional housekeeping is performed. Such housekeeping is not currently implemented. Therefore, some of the table manipulation methods such as adding, deleting or moving columns or rows, get automatically disabled from the pop-up menus when the table has children structures derived from it. As a way around this restriction, you can duplicate the table and work on the duplicate instead.

[2]For multiple row or column selections, hold the CTRL key down when you left-click the rows or columns in question. Use the SHIFT key to select ranges.

4.3 D▶Dictionary

4.3.1 D▶Dictionary▶Export...

Name: *DictionaryExporter*

Description: Exports a data dictionary associated with a decision table to an ASCII file. Exported data dictionaries can be manually edited and then imported back into the system.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $\{Dictionary, DecisionTable\} \rightarrow \{Dictionary, DecisionTable\}$

4.3.2 D▶Dictionary▶Import...

Name: *DictionaryImporter*

Description: Allows hand-crafted data dictionaries to be imported into the system from ASCII files. The format of the dictionary file is documented in Section A.1.

TIP Importing a new dictionary for a decision table does not alter the internal integer representation of the decision table, but only replaces its data dictionary.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $\{Dictionary, DecisionTable\} \rightarrow \{Dictionary, DecisionTable\}$

4.4 D▶Export

Name: *DecisionTableExporter*

Description: Algorithms in this family export some aspect of a decision table to a file in some format. The algorithms function as simple identity pass-through routines, where the exporting is a side-effect of applying the algorithm.

4.4.1 D▶Export▶XML format...

Name: *XMLDecisionTableExporter*

Description: Exports a decision table to XML format. The exported XML contains both dictionary information and table entries.

TIP If an entry in the table is missing, then that descriptor is omitted in the output. Hence, the number of descriptors may vary across objects.

TIP XML greatly increases interoperability with other programs, and you can transform the XML into other formats using the XSLT language. Many current browsers also support viewing of XML documents.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $DecisionTable \rightarrow DecisionTable$

Example: The XML fragment below is an example of the dictionary meta-data contained in the XML output.

```
<attributes>
  <attribute id="0" name="Radius" type="Float" exponent="2" unit="cm"/>
  <attribute id="1" name="Color" type="String"/>
  <attribute id="2" name="Year" type="Integer" masked="true"/>
  <attribute id="3" name="Grade" type="Float" exponent="1"/>
  <attribute id="4" name="Sold" type="String" status="Decision"/>
</attributes>
```

The last two objects in the decision table shown in Appendix A.2.1 would be exported as the following XML fragment:

```
<object id="5">
  <descriptor attribute="Color" value="Yellow"/>
  <descriptor attribute="Year" value="1865"/>
  <descriptor attribute="Grade" value="2.5"/>
  <descriptor attribute="Sold" value="No"/>
</object>
<object id="6">
  <descriptor attribute="Radius" value="4925.60"/>
  <descriptor attribute="Year" value="1968"/>
  <descriptor attribute="Grade" value="6.0"/>
  <descriptor attribute="Sold" value="Yes"/>
</object>
```

4.4.2 D▶Export▶Prolog format...

Name: *PrologDecisionTableExporter*

Description: Exports a decision table as a set of Prolog facts. One fact is exported per observed entry per object in the table. The objects are named "o_n", where n is the index of the object in the table. Index counts start at 1.

TIP Missing entries in a table are not exported as facts.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

Example: The last two objects in the decision table shown in Appendix A.2.1 would be exported as the following set of facts:

```
color(o6, yellow).
year(o6, 1865).
grade(o6, 2.5).
sold(o6, no).

radius(o7, 4925.60).
year(o7, 1968).
grade(o7, 6.0).
sold(o7, yes).
```

4.4.3 D▶Export▶Matlab format...

Name: *MatlabDecisionTableExporter*

Description: Exports a decision table to an ASCII file as a matrix that the MATLAB software system [22] can import. Useful if we want to visualize or mathematically manipulate the contents of the table in ways not supported by ROSETTA.

TIP Integer and float attributes are exported as integer and float values. For string attributes, the integer codes used internally are exported.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* → *DecisionTable*

Example: The following MATLAB command sequence will load an exported table into MATLAB and generate a 2D plot of attribute number 1 against attribute number 5.

```
>> load table.txt
>> plot(table(:, 1), table(:, 5), '*')
```

Each data point in the plot will be marked with an asterisk. For a piecewise linear graph, omit the last argument to the `plot` command.

4.4.4 D▶Export▶Plain format...

Name: *MyDecisionTableExporter*

Description: Exports an information system to an ASCII file in a plain and simple, tabular format. The inverse of the algorithm described in Appendix A.2.1.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* → *DecisionTable*

4.4.5 D▶Export▶Indiscernibility graph...

Name: *IndiscernibilityGraphExporter*

Description: Exports the system's indiscernibility graph to a format recognized by the GraphViz suite of graph visualization programs [16]. The graph can be used for clustering and unsupervised learning.[3]

Support for IDGs[4] is provided. The syntax for specifying IDGs is described in Appendix A.5.

If there is a masked attribute in the information system that assigns unique names to objects, the name of this attribute can be specified to name the vertices in the exported graph.

If specified, the output file can also contain vertex degree data as well as all-pairs shortest path (APSP) data. Floyd's algorithm[5] is used to compute the APSP data. The APSP data can be used to define a distance metric for clustering [37].

TIP ROSETTA currently requires that the IDGs are reflexive and symmetric. It is the user's responsibility to see to this when defining the IDGs.

Dialogs: Figure 5.

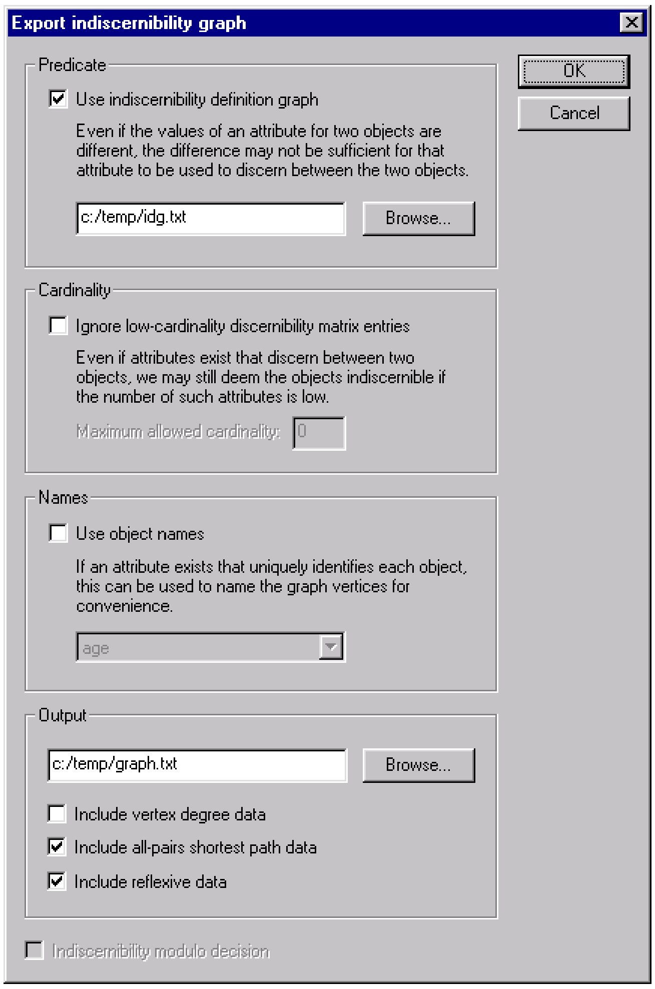
Keywords: IDG (*Boolean*), IDG.FILENAME (*String*), NAMES (*Boolean*), NAMES.ATTRIBUTE (*String*), MODULO.DECISION (*Boolean*), FILENAME (*String*), DATA.REFLEXIVE (*Boolean*), DATA.DEGREE (*Boolean*), DATA.APSP (*Boolean*), CARDINALITY (*Boolean*), CARDINALITY.THRESHOLD (*Integer*).

Signature: *DecisionTable* → *DecisionTable*

[3] Algorithms for generating visually pleasing graph layouts exist that operate by constructing a virtual physical model and running an iterative solver to find a low-energy configuration [14,21,15]. For clustering, the spatial layout of each vertex is really irrelevant, it is the distribution of edges between the vertices that is of interest. Visualization greatly aids understanding, though.

[4] Indiscernibility definition graphs. See Øhrn [26, page 43] for details.

[5] Note that this algorithm has a time complexity of $O(|U|^3)$.



Export indiscernibility graph

Predicate

[✓] Use indiscernibility definition graph

Even if the values of an attribute for two objects are different, the difference may not be sufficient for that attribute to be used to discern between the two objects.

c:/temp/idg.txt

Cardinality

[ ] Ignore low-cardinality discernibility matrix entries

Even if attributes exist that discern between two objects, we may still deem the objects indiscernible if the number of such attributes is low.

Maximum allowed cardinality: 0

Names

[ ] Use object names

If an attribute exists that uniquely identifies each object, this can be used to name the graph vertices for convenience.

age

Output

c:/temp/graph.txt

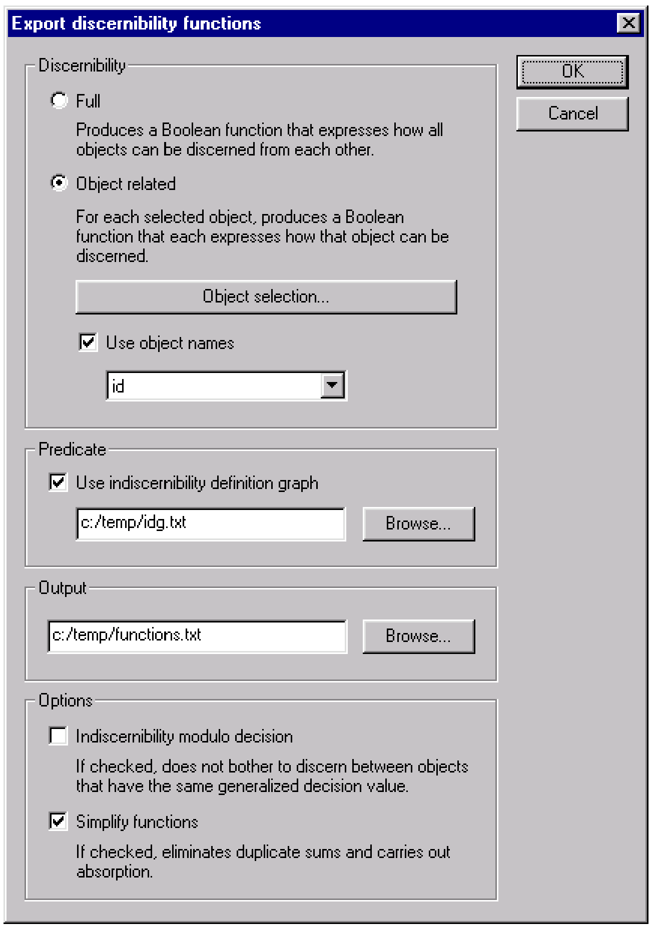
[ ] Include vertex degree data

[✓] Include all-pairs shortest path data

[✓] Include reflexive data

[ ] Indiscernibility modulo decision

Figure 5: Dialog box for exporting indiscernibility graphs.


Export discernibility functions

Discernibility

( ) Full

Produces a Boolean function that expresses how all objects can be discerned from each other.

(●) Object related

For each selected object, produces a Boolean function that each expresses how that object can be discerned.

Object selection...

☑ Use object names: id

Predicate

☑ Use indiscernibility definition graph: c:/temp/idg.txt

Output

c:/temp/functions.txt

Options

☐ Indiscernibility modulo decision

If checked, does not bother to discern between objects that have the same generalized decision value.

☑ Simplify functions

If checked, eliminates duplicate sums and carries out absorption.

Figure 6: Dialog box for exporting discernibility functions.

4.4.6 D▶Export▶Discernibility functions...

Name: *DiscernibilityFunctionExporter*

Description: Enables Boolean POS functions to be exported that express how objects can be discerned. The function that expresses how all objects can be discerned from each other can also be exported.

Support for IDGs is provided. The syntax for specifying IDGs is described in Appendix A.5.

If there is a masked attribute in the information system that assigns unique names to objects, the name of this attribute can be specified to name the exported functions.

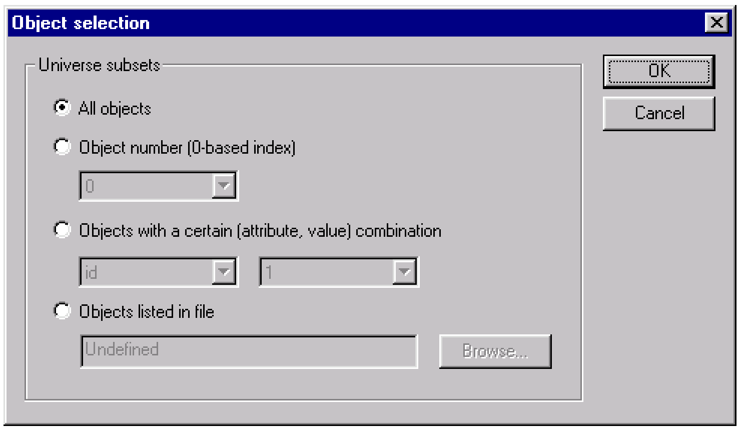
The discernibility functions can be exported in both unsimplified and simplified versions. Since unsimplified functions can be very large and unwieldy, simplification is recommended.

TIP ROSETTA currently requires that the IDGs are reflexive and symmetric. It is the user's responsibility to see to this when defining the IDGs.

Dialogs: Figure 6, Figure 7.

Keywords: DISCERNIBILITY ({All, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), SIMPLIFY (*Boolean*), IDG (*Boolean*), IDG.FILENAME (*String*), NAMES (*Boolean*), NAMES.ATTRIBUTE (*String*), MODULO.DECISION (*Boolean*), FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*


Object selection

Universe subsets

[✓ All objects]

Object number (0-based index): 0

Objects with a certain (attribute, value) combination: id, 1

Objects listed in file: Undefined

Figure 7: Dialog box for selecting subsets of U.

4.5 D▶Complete

Name: *Completer*

Description: Algorithms in this family take as input a decision table and return a new, "completed" decision table. The input table may have missing values, whereas the output table is a completed version of the input, i.e., it has no missing values.

TIP Completion is only necessary if the algorithms that are subsequently to be applied to the decision table require this. Also, the use of IDGs may make completion unnecessary. In some cases a missing entry might indicate "not applicable" instead of "not recorded". Completion of such attributes may not be desirable.

It is often desirable to condition the completion to the decision classes. Such conditioning is carried out by the following procedure:

1. Split the table into several subtables, each with their own decision class.
2. Apply an unconditioned *Completer* algorithm to each subtable.
3. Merge the completed subtables.

Saving of completion information to file is not yet implemented.

4.5.1 D▶Complete▶Remove incompletes

Name: *RemovalCompleter*

Description: Removes all objects that have one or more missing values. If U and $\widehat{U}$ denote the universes before and after removing incomplete objects, we have:

$$\widehat{U} = \{x \in U \mid \forall a \in A,\ a(x) \neq \top\} \quad (1)$$

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.5.2 D▶Complete▶Mean/mode fill

Name: *MeanCompleter*

Description: Substitutes missing values for numerical attributes with the mean value of all observed entries for that attribute. For string attributes, missing values are substituted by the "mode" value, i.e., the most frequently occurring value among the observed entries for that attribute. If a and $\widehat{a}$ denote an attribute before and after completion, we have:

$$O_a = \{x \in U \mid a(x) \neq \top\} \tag{2}$$

$$O_a^v = \{x \in O_a \mid a(x) = v\} \tag{3}$$

$$\widehat{a}(x) = \begin{cases} a(x) & \text{if } x \in O_a \\ \dfrac{1}{|O_a|} \sum\limits_{O_a} a(x) & \text{if } x \notin O_a \text{ and } a \text{ is numerical} \\ \arg\max\limits_{v} |O_a^v| & \text{if } x \notin O_a \text{ and } a \text{ is not numerical} \end{cases} \tag{4}$$

Ties for mode values are resolved arbitrarily.

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.5.3 D▶Complete▶Conditioned mean/mode fill

Name: *ConditionedMeanCompleter*

Description: Similar to the algorithm described in Section 4.5.2, but the computations of the mean and mode values are conditioned to the decision classes. No special provisions are made for objects with decision value **Undefined**, if any.

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.5.4 D▶Complete▶Combinatorial completion

Name: *CombinatorialCompleter*

Description: Expands each missing value for each object into the set of possible values. That is, an object is expanded into several objects covering all possible combinations of the object's missing values.

TIP This algorithm should be used with care, since the number of possible combinations for objects with multiple missing values grows very rapidly.

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

Example: Assume an object has missing values for condition attributes a and b, and let $|V_a| = 3$ and $|V_b| = 4$. The single incomplete object is then expanded into 12 complete objects, covering all possible combinations of values for a and b.

4.5.5 D▶Complete▶Conditioned combinatorial completion

Name: *ConditionedCombinatorialCompleter*

Description: Similar to the algorithm described in Section 4.5.4, but sets of expansion values are conditioned to the decision classes. No special provisions are made for objects with decision value **Undefined**, if any.

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6 D▶Discretize

Name: *Scaler*

Description: Algorithms that belong to this family discretize attributes in information systems.

Input to a discretization algorithm is a decision table, and a decision table is returned. Unless otherwise stated, the returned table is a discretized duplicate of the input table.

Discretization amounts to searching for "cuts" that determine intervals. All values that lie within each interval are then mapped to the same value, in effect converting numerical attributes to attributes that can be treated as being categorical. The search for cuts is performed on the internal integer representation of the input decision table. Automatic "grouping" as a symbolic counterpart to intervals is not currently implemented in ROSETTA, but can be done manually.

Only the unmasked attributes of the input table are visible to the discretization algorithms. If specified, condition string attributes can be temporarily masked away before the table is subsequently passed on to the actual discretization process. When done, the original masking states are reinstated. Masking can also be done manually in the GUI.

TIP By using the masking feature appropriately, it is possible to discretize a table using different discretization methods for different attributes, or to only discretize a selected subset of attributes.

TIP Discretized attributes are converted to string type.

In ROSETTA, algorithms for automatic discretization generally fall into one of three categories:

- Each attribute is considered in isolation, and no knowledge of any outcome or decision attribute is employed in the process. These algorithms are said to be univariate and unsupervised.
- Only one condition attribute is considered at a time, but is done so in conjunction with the decision attribute. These algorithms are said to be univariate and supervised.
- All condition attributes are considered simultaneously, and are done so in conjunction with the decision attribute. These algorithms are said to be multivariate and supervised.

Unsupervised, multivariate clustering algorithms would form a fourth discretization category, but none such are currently implemented in ROSETTA.

4.6.1 D▶Discretize▶Boolean reasoning algorithm...

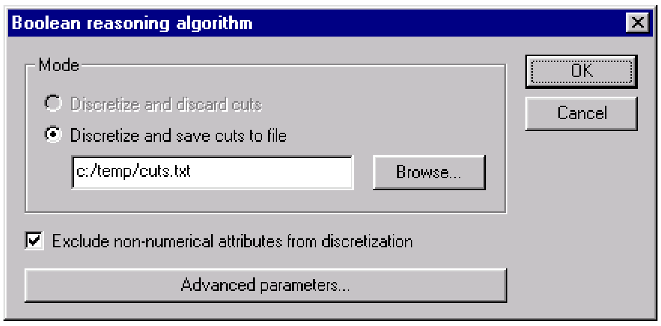
Name: *BROrthogonalScaler*

Description: A straightforward implementation of the algorithm outlined by Nguyen and Skowron [24], based on combining the cuts found by the algorithm from Section 4.6.5 with a Boolean reasoning procedure for discarding all but a small subset of these. The remaining subset is a minimal set of cuts that preserves the discernibility inherent in the decision system.

The algorithm operates by first creating a Boolean function f from the set of candidate cuts, and then computing a prime implicant of this function. (The set C_a below is defined by Equation 9.) The current implementation uses the greedy algorithm of Johnson [20] to compute the prime implicant, described in Section 4.7.2.

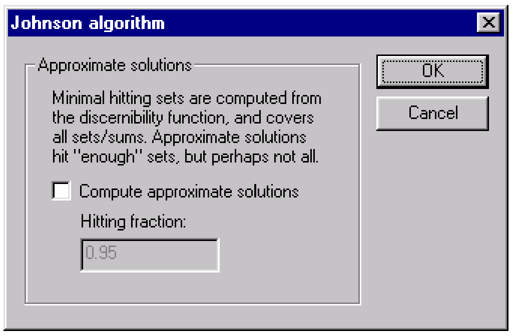
$$f = \prod_{(x,y)} \sum_{a} \left\{ \sum c^* \mid c \in C_a \text{ and } a(x) < c < a(y) \text{ and } \partial_A(x) \neq \partial_A(y) \right\} \tag{5}$$

Sometimes, the Boolean reasoning approach to discretization may result in no cuts being deemed necessary for some attributes. This means that these attributes are not really needed



Mode
( ) Discretize and discard cuts
(●) Discretize and save cuts to file
c:/temp/cuts.txt
☑ Exclude non-numerical attributes from discretization

Figure 8: Dialog box for discretization via "cuts".



Approximate solutions

Minimal hitting sets are computed from the discernibility function, and covers all sets/sums. Approximate solutions hit "enough" sets, but perhaps not all.

☐ Compute approximate solutions

Hitting fraction: 0.95

Figure 9: Dialog box for Johnson's greedy algorithm.

to preserve the discernibility, if the minimal set of cuts is employed. Rather than simply setting all values for these attributes to, e.g., 0, this algorithm leaves them untouched. The decision on how to deal with these attributes are left to the user. A common fallback is to revert to another discretization scheme for these undiscretized attributes. See Øhrn [26, page 108] for an example of this.

Note that the Boolean reasoning algorithm computes a reduct of the decision system.

TIP This straightforward implementation has a worst-case complexity of order $O(|A||U|^3)$, which may be prohibitively high for large tables. The algorithm described in Section 4.6.8 basically implements the same procedure, but uses an efficient counting technique that results in an implementation with a $O(|A||U| \log |U|)$ complexity.

TIP An approximate solution to the discretization problem can be obtained by computing an approximate prime implicant of the function f above. See Figure 9. This amounts to using fewer cuts, however on the cost of introducing inconsistencies into the discretized table.

Dialogs: Figure 8, Figure 9.

Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*), APPROXIMATE (*Boolean*), FRACTION (*Float*).

Signature: *DecisionTable* → *DecisionTable*

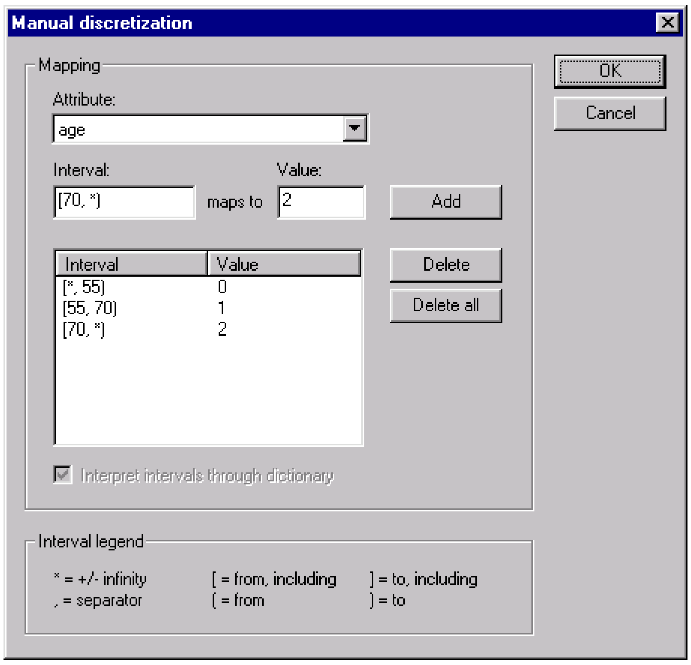
4.6.2 D▶Discretize▶Manual discretization...

Name: *ManualScaler*

Description: Enables the user to manually specify cuts to be used for discretizing a given attribute.

TIP The input table is modified directly, i.e., a duplicate table is not returned.

Dialogs: Figure 10.



Manual discretization

Mapping

Attribute: age

Interval: [70, *) maps to Value: 2

Interval	Value
[*, 55)	0
[55, 70)	1
[70, *)	2

[✓] Interpret intervals through dictionary

Interval legend

* = +/- infinity	[= from, including	] = to, including
, = separator	(= from	) = to

Figure 10: Dialog box for manual discretization.

Keywords: ATTRIBUTE (*Integer*), INTERVALS (*IntervalList*), DICTIONARY (*Boolean*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6.3 D▶Discretize▶Entropy/MDL algorithm...

Name: *EntropyScaler*

Description: Implements the algorithm described by Dougherty et al. [13], based on recursively partitioning the value set of each attribute so that a local measure of entropy is optimized. The minimum description length principle defines a stopping criterion for the partitioning process.

Missing values are ignored in the search for cuts. If no cuts are found for an attribute, the attribute is left unprocessed.

TIP You can discretize the attributes left unprocessed by means of some other "backup" discretization algorithm, e.g., equal frequency binning as described in Section 4.6.4.

Dialogs: Figure 8.

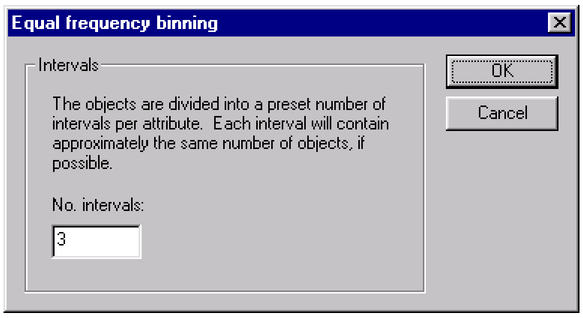
Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6.4 D▶Discretize▶Equal frequency binning...

Name: *EqualFrequencyScaler*

Description: Implements equal frequency binning, a simple unsupervised and univariate discretization algorithm. Fixing a number of intervals n and examining the histogram of each attribute, $n - 1$ cuts are determined so that approximately the same number of objects fall into each of the n intervals. This corresponds to assigning $n - 1$ cuts such that the area between two neighboring cuts in the normalized histogram is as close to $1/n$ as possible.


Equal frequency binning

Intervals

The objects are divided into a preset number of intervals per attribute. Each interval will contain approximately the same number of objects, if possible.

No. intervals: 3

Figure 11: Dialog box for equal frequency binning.

TIP If $n = 1$ is specified, this effectively amounts to masking away the attributes in a subsequent reduction process.

Dialogs: Figure 8, Figure 11.

Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*), INTERVALS (*Integer*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6.5 D▶Discretize▶Naive algorithm...

Name: *NaiveScaler*

Description: Implements a very straightforward and simple heuristic that may result in very many cuts, probably far more than are desired. In the worst case, each observed value is assigned its own interval. In some cases, however, a simplistic and naive scheme may suffice.

For the sake of simplifying the exposition, we will assume that all condition attributes A are numerical. For each condition attribute a we can sort its value set V_a to obtain the following ordering:

$$v_a^1 < \ldots < v_a^i < \ldots < v_a^{|V_a|} \tag{6}$$

Let C_a denote the set of all naively generated cuts for attribute a, defined as shown below. The set C_a simply consists of all cuts midway between two observed attribute values, except for the cuts that are clearly not needed if we do not bother to discern between objects with the same decision values.

$$X_a^i = \{x \in U \mid a(x) = v_a^i\} \tag{7}$$

$$\Delta_a^i = \{v \in V_d \mid \exists x \in X_a^i \text{ such that } d(x) = v\} \tag{8}$$

$$C_a = \left\{ \frac{v_a^i + v_a^{i+1}}{2} \mid |\Delta_a^i| > 1 \text{ or } |\Delta_a^{i+1}| > 1 \text{ or } \Delta_a^i \neq \Delta_a^{i+1} \right\} \tag{9}$$

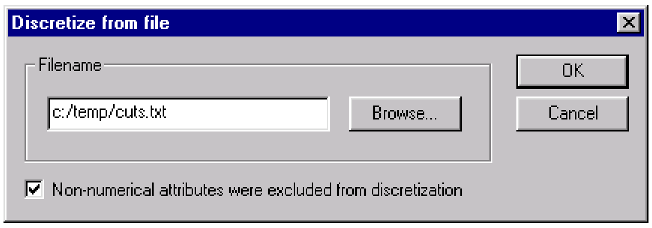
In essence, we place cuts midway between all v_a^i and v_a^{i+1}, except for in the situation when all objects that have these values also have equal generalized decision values wrt. a that are singletons.

If no cuts are found for an attribute, the attribute is left unprocessed. Missing values are ignored in the search for cuts.

Dialogs: Figure 8.

Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*



c:/temp/cuts.txt

☑ Non-numerical attributes were excluded from discretization

Figure 12: Dialog box for discretization from a file with cuts.

4.6.6 D▶Discretize▶Semi-naive algorithm...

Name: *SemiNaiveScaler*

Description: Functionally similar to the naive algorithm from Section 4.6.5, but has more logic to handle the case where value-neighboring objects belong to different decision classes. Typically results in fewer cuts than the simpler naive algorithm, but may still produce far more cuts than are desired.

The set of cuts found by the semi-naive algorithm is a subset of the cuts found by the naive algorithm from Section 4.6.5, computed as follows:

$$D_a^i = \{v \in V_d \mid v = \arg\max_{v'} |\{x \in X_a^i \mid d(x) = v'\}|\} \tag{10}$$

$$C_a = \left\{ \frac{v_a^i + v_a^{i+1}}{2} \mid D_a^i \not\subseteq D_a^{i+1} \text{ and } D_a^{i+1} \not\subseteq D_a^i \right\} \tag{11}$$

The set D_a^i simply collects the dominating decision values for the objects in X_a^i. If there are no ties, D_a^i is a singleton. The rationale for not adding a cut if the sets of dominating decisions define an inclusion is that we then hope, although the current implementation does not check, that a cut will be added for another attribute different from a such that the objects in X_a^i and X_a^{i+1} can be discerned.

Dialogs: Figure 8.

Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6.7 D▶Discretize▶From file with cuts...

Name: *OrthogonalFileScaler*

Description: Given a file with cuts, discretizes a table according to the file's contents. The format of the file is a set of tab-delimited (a, c) pairs, one per line. The attribute index a is an integer relative to a masked table, while the cut c is a value relative to the internal integer table representation. Attributes with no cuts are left unprocessed.

TIP If the cut file was generated with the auto-masking feature for string attributes set, this has to be specified since the attribute indices in the file are interpreted relative to a masked table.

Dialogs: Figure 12.

Keywords: MODE ({Load}), MASK (*Boolean*), FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *DecisionTable*

4.6.8 D▶Discretize▶Boolean reasoning algorithm (RSES)...

Name: *RSESOrthogonalScaler*

Description: An efficient implementation of the Boolean reasoning algorithm of Nguyen and Skowron [24], as described by Nguyen and Nguyen [23]. Functionally similar to the algorithm described in Section 4.6.1, but much faster. Approximate solutions are not supported.

If $a(x)$ is missing, object x is not excluded from consideration when processing attribute a, but is instead treated as an "infinitely large" positive value. If no cuts are found for an attribute, all entries for that attribute are set to 0.

TIP Since no support for missing values is provided, this algorithm is best applied to complete tables.

Dialogs: Figure 8.

Keywords: MODE ({Save, Discard}), MASK (*Boolean*), FILENAME (*String*).

Signature: $DecisionTable \longrightarrow DecisionTable$

4.6.9 D▶Discretize▶From file with cuts (RSES)...

Name: *RSESOrthogonalFileScaler*

Description: Functionally similar to the algorithm described in Section 4.6.7, but handles attributes with no cuts in a different manner. All entries for such attributes are set to 0, instead of being left unprocessed.

Dialogs: Figure 12.

Keywords: MODE ({Load}), MASK (*Boolean*), FILENAME (*String*).

Signature: $DecisionTable \longrightarrow DecisionTable$

4.7 D▶Reduce

Name: *Reducer*

Description: Algorithms for computing reducts or reduct approximations belong to this family. Note that any attribute subset is in this context considered to be an approximation to a reduct.

Input to a *Reducer* algorithm is a decision table, and a set of reducts is returned. The returned reduct set may possibly have a set of rules attached to it as a child. A reduct is a collection of attribute indices into the table the reduct belongs to.

Two main types of discernibility are currently supported by ROSETTA. In addition, reducts of both these types can be computed modulo the decision attribute or not. (More on this below.)

- *Full:* Computes reducts relative to the system as a whole, i.e., minimal attribute subsets that preserve our ability to discern all relevant objects from each other.
- *Object:* Computes reducts relative to a fixed object, i.e., minimal attribute subsets that preserve our ability to discern that object from the other relevant objects. Generally, instead of fixing a single object x, we select a subset X of U, and process each $x \in X$ sequentially. That is, we first compute the minimal attribute subsets that discern the first object in X from all other relevant objects in U, before proceeding to compute the minimal attribute subsets that discern the second object in X from all other relevant objects in U, etc.

TIP If the reducts are relative to an object, rules or patterns are computed on the fly as well for reasons of efficiency.

Option	Subset
All	$X = U$
Index	$X = \{x\}$
Value	$X = \{x \in U \mid a(x) = v\}$
File	$X = \{x \in U \mid x \text{ is listed in a file}\}$

Table 1: Options for selecting subsets of U. An object is specified by a 0-based index into U. For files, one 0-based object index should appear on each line in the file. Alternatively, a line can consist of a 0-based index set contained in curly braces. Lines in other formats than the ones described are ignored.

For reducts relative to an object, the set X can be selected in different ways, as shown in Table 1.

TIP The option to specify X by attribute values can be used if we want to only generate rules for a specified decision class.

A table can either be interpreted as a decision system or as a general Pawlak information system [31]. If the option to compute reducts modulo the decision attribute is checked, the table is interpreted as a decision system. If the decision system contains inconsistencies, boundary region thinning [44, 43] should be considered.

TIP For consistent systems, there are no boundary regions to thin, and boundary region thinning does hence not have any effect.

If the algorithm supports it, a set of IDGs can be supplied that enables the notion of discernibility to be overloaded on a per attribute basis. See Øhrn [26, pages 42–45] for details. The IDG file format is described in Appendix A.5. If no IDG file is specified, strict inequality is used.

TIP If IDGs are used, ROSETTA currently requires that the IDGs are reflexive and symmetric. It is the user's responsibility to see to this when defining the IDGs.

TIP Since a reduct is a prime implicant of a discernibility function, algorithms for computing reducts can be used for more general Boolean reasoning, too. See Øhrn [25] or Section A.2.2.

Dialogs: Figure 13, Figure 7.

4.7.1 D▶Reduce▶Genetic algorithm...

Name: *SAVGeneticReducer*

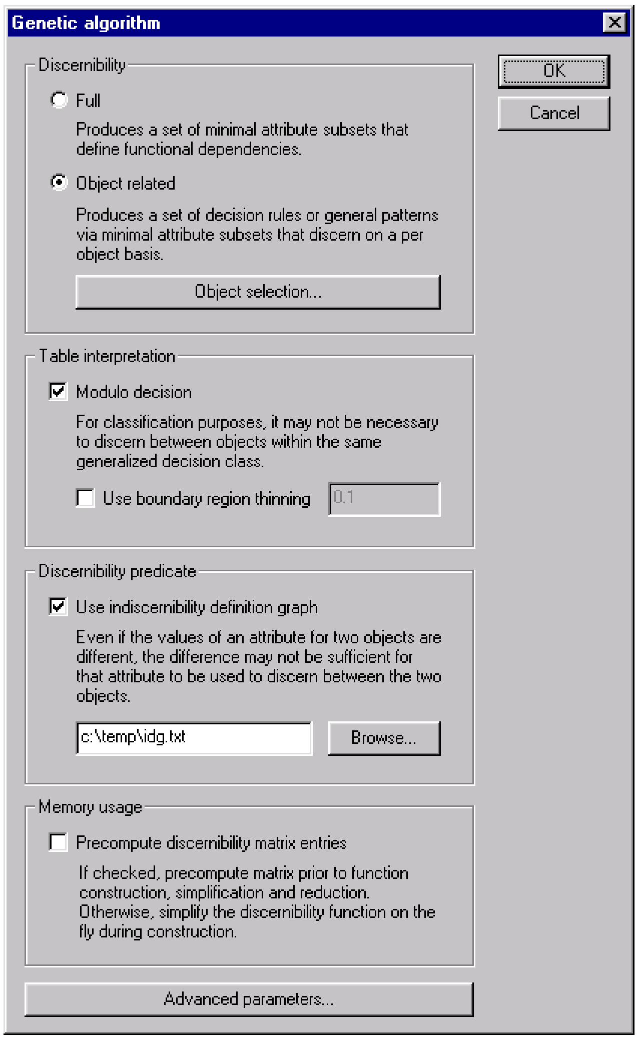
Description: Implements a genetic algorithm for computing minimal hitting sets, as described by Vinterbo and Øhrn [40, 41]. The algorithm has support for both cost information and approximate solutions.

The algorithm's fitness function f is defined below, where $\mathcal{S}$ is the set of sets corresponding to the discernibility function.[6] The parameter α defines a weighting between subset cost and hitting fraction, while ε is relevant in the case of approximate solutions. (More on this below.)

$$f(B) = (1-\alpha) \times \frac{cost(A) - cost(B)}{cost(A)} + \alpha \times \min\left\{\varepsilon, \frac{|[S \text{ in } \mathcal{S} \mid S \cap B \neq \emptyset]|}{|\mathcal{S}|}\right\} \tag{12}$$

The subsets B of A that are found through the evolutionary search driven by the fitness function and that are "good enough" hitting sets, i.e., have a hitting fraction of at least ε, are collected in a "keep list". The size of the keep list can be specified.

[6] See Vinterbo and Øhrn [40, 41] or Øhrn [26, pages 52–55] for details. The expression for the hitting fraction in the definition of f is here somewhat simplified. In reality, we associate a weight $w(S)$ with each S is $\mathcal{S}$.


Genetic algorithm

Discernibility

( ) Full

Produces a set of minimal attribute subsets that define functional dependencies.

(●) Object related

Produces a set of decision rules or general patterns via minimal attribute subsets that discern on a per object basis.

Table interpretation

[✓] Modulo decision

For classification purposes, it may not be necessary to discern between objects within the same generalized decision class.

[ ] Use boundary region thinning 0.1

Discernibility predicate

[✓] Use indiscernibility definition graph

Even if the values of an attribute for two objects are different, the difference may not be sufficient for that attribute to be used to discern between the two objects.

c:\temp\idg.txt

Memory usage

[ ] Precompute discernibility matrix entries

If checked, precompute matrix prior to function construction, simplification and reduction. Otherwise, simplify the discernibility function on the fly during construction.

Figure 13: Dialog box for computing reducts and rules.

The function *cost* specifies the cost of an attribute subset. The file format for specifying cost information is defined in Appendix A.6. If no cost information is used, then a default unit cost is assumed, effectively defining $cost(B) = |B|$.

Approximate solutions are controlled through two parameters, ε and k. The parameter ε signifies a minimal value for the hitting fraction, while k denotes the number of extra keep lists in use by the algorithm. If $k = 0$, then only minimal hitting sets with a hitting fraction of approximately ε are returned. If $k > 0$, then $k + 1$ groups of minimal hitting sets are returned, each group having an approximate (but not smaller) hitting fraction evenly spaced between ε and 1. Note that $\varepsilon = 1$ implies proper minimal hitting sets.

Each reduct in the returned reduct set has a support count associated with it. The support count is a measure of the "strength" of the reduct, and may interpreted differently according to which algorithm that produced the reduct. For reducts computed with this algorithm, the support count equals the reduct's hitting fraction, multiplied by 100.

Dialogs: Figure 13, Figure 14, Figure 7.

Keywords: DISCERNIBILITY ({Full, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), MODULO.DECISION (*Boolean*), BRT (*Boolean*), BRT.PRECISION (*Float*), IDG (*Boolean*), IDG.FILENAME (*String*), PRECOMPUTE (*Boolean*), ALGORITHM ({Simple, Modified, Variable}), LIFETIME.LOWER (*Integer*), LIFETIME.UPPER (*Integer*), SCALING (*Boolean*), SCALING.TYPE ({Boltzmann}), TEMPERATURE.UPPER (*Float*), TEMPERATURE.LOWER (*Float*), TEMPERATURE.DELTA (*Float*), REPORT (*Boolean*), REPLACE (*Boolean*), ELITISM (*Boolean*), SEED (*Integer*), SIZE.KEEP (*Integer*), SIZE.POPULATION (*Integer*), GAP.GENERATION (*Integer*), STOP.FITNESS (*Boolean*), STOP.KEEP (*Boolean*), PROBABILITY.CROSSOVER (*Float*), PROBABILITY.MUTATION (*Float*), PROBABILITY.INVERSION (*Float*), NUMBER.CROSSOVER (*Integer*), NUMBER.MUTATION (*Integer*), NUMBER.INVERSION (*Integer*), BIAS (*Float*), COST (*Boolean*), COST.FILENAME (*String*), APPROXIMATE (*Boolean*), FRACTION (*Float*), KEEP.LEVELS (*Integer*).

Signature: $\textit{DecisionTable} \longrightarrow \textit{Reducts}$

Example: Having specified that we want approximate solutions, consider the case where $\varepsilon = 0.7$ and $k = 3$. In all, 4 groups of minimal hitting sets will be computed and returned, with hitting fractions approximately equal to (but not below) 0.7, 0.8, 0.9 and 1.0.

4.7.2 D▶Reduce▶Johnson's algorithm...

Name: *JohnsonReducer*

Description: Invokes a variation of a simple greedy algorithm to compute a single reduct only, as described by Johnson [20]. The algorithm has a natural bias towards finding a single prime implicant of minimal length.

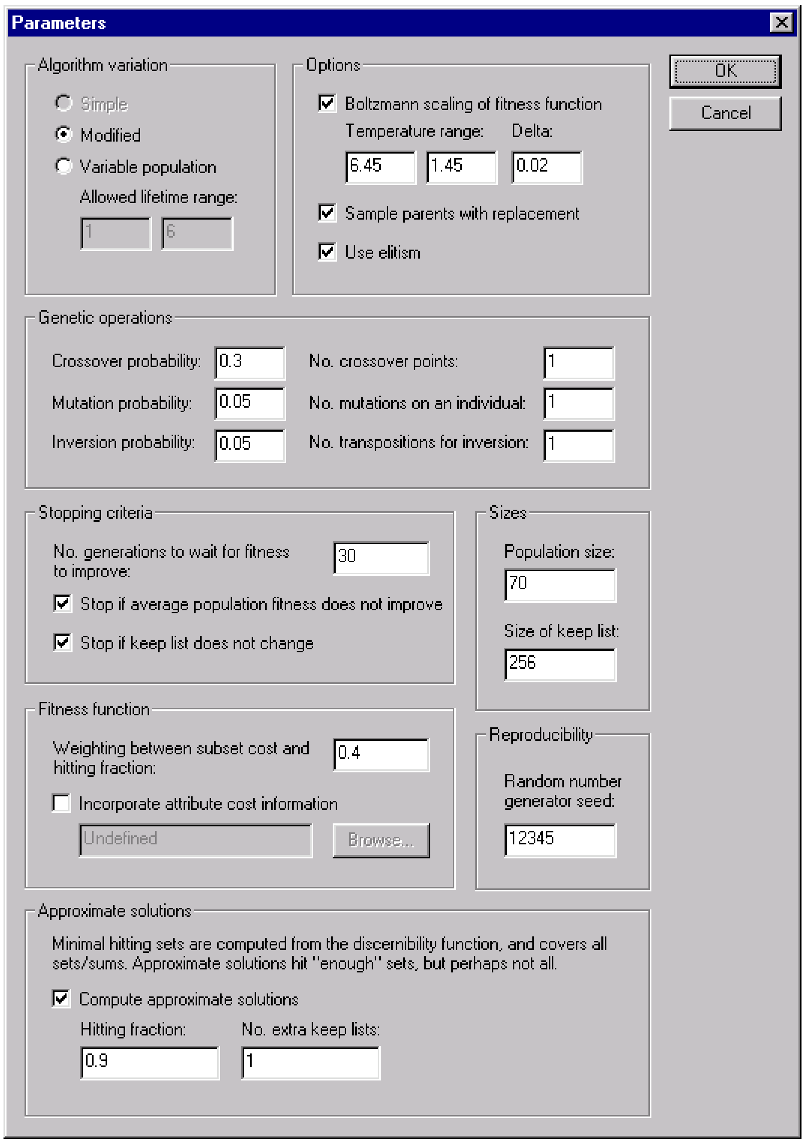
The reduct B is found by executing the algorithm outlined below, where $\mathcal{S}$ denotes the set of sets corresponding to the discernibility function, and $w(S)$ denotes a weight for set S in $\mathcal{S}$ that automagically gets computed from the data.[7]

1. Let $B = \emptyset$.
2. Let a denote the attribute that maximizes $\sum w(S)$, where the sum is taken over all sets S in $\mathcal{S}$ that contain a. Currently, ties are resolved arbitrarily.
3. Add a to B.
4. Remove all sets S from $\mathcal{S}$ that contain a.
5. If $\mathcal{S} = \emptyset$ return B. Otherwise, goto step 2.

Support for computing approximate solutions is provided by aborting the loop when "enough" sets have been removed from $\mathcal{S}$, instead of requiring that $\mathcal{S}$ has to be fully emptied.

The support count associated with the computed reduct equals the reduct's hitting fraction multiplied by 100, i.e., the percentage of sets in $\mathcal{S}$ that B has a non-empty intersection with.

[7]See Øhrn [26, pages 54–55] or Vinterbo and Øhrn [40,41] for details.


Parameters

Algorithm variation
- ○ Simple
- [✓ ● Modified]
- ○ Variable population

Allowed lifetime range: 1 6

Options
- [✓] Boltzmann scaling of fitness function

Temperature range: 6.45 1.45 Delta: 0.02
- [✓] Sample parents with replacement
- [✓] Use elitism

Genetic operations

Crossover probability:	0.3	No. crossover points:	1
Mutation probability:	0.05	No. mutations on an individual:	1
Inversion probability:	0.05	No. transpositions for inversion:	1

Stopping criteria

No. generations to wait for fitness to improve: 30
- [✓] Stop if average population fitness does not improve
- [✓] Stop if keep list does not change

Sizes

Population size: 70

Size of keep list: 256

Fitness function

Weighting between subset cost and hitting fraction: 0.4
- [ ] Incorporate attribute cost information

Undefined Browse...

Reproducibility

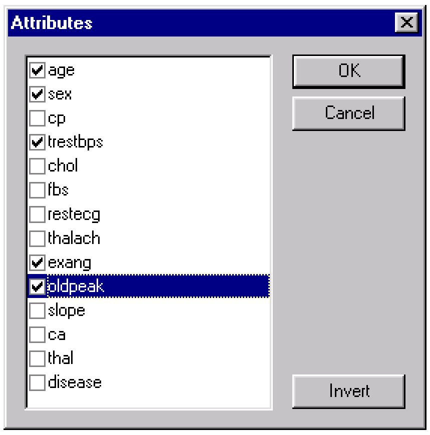
Random number generator seed: 12345

Approximate solutions

Minimal hitting sets are computed from the discernibility function, and covers all sets/sums. Approximate solutions hit "enough" sets, but perhaps not all.
- [✓] Compute approximate solutions

Hitting fraction: 0.9 No. extra keep lists: 1

Figure 14: Dialog box for providing parameters to the genetic algorithm.


Attributes

- [✓] age
- [✓] sex
- [ ] cp
- [✓] trestbps
- [ ] chol
- [ ] fbs
- [ ] restecg
- [ ] thalach
- [✓] exang
- [✓] oldpeak
- [ ] slope
- [ ] ca
- [ ] thal
- [ ] disease

Figure 15: Dialog box for selecting subsets of A.

Dialogs: Figure 13, Figure 9, Figure 7.

Keywords: DISCERNIBILITY ({Full, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), MODULO.DECISION (*Boolean*), BRT (*Boolean*), BRT.PRECISION (*Float*), IDG (*Boolean*), IDG.FILENAME (*String*), PRECOMPUTE (*Boolean*), APPROXIMATE (*Boolean*), FRACTION (*Float*).

Signature: *DecisionTable* $\longrightarrow$ *Reducts*

Example: Let $\mathcal{S} = \{\{cat, dog, fish\}, \{cat, man\}, \{dog, man\}, \{cat, fish\}\}$ and let for simplicity w be the constant function that assigns 1 to all sets S in $\mathcal{S}$. Step 2 in the algorithm then amounts to selecting the attribute that occurs in the most sets in $\mathcal{S}$.

Initially, $B = \emptyset$. Since *cat* is the most frequently occurring attribute in $\mathcal{S}$, we update B to include *cat*. We then remove all sets from $\mathcal{S}$ that contain *cat*, and obtain $\mathcal{S} = \{\{dog, man\}\}$. Repeating the process, we arrive at a tie in the occurrence counts of *dog* and *man*, and arbitrarily select *dog*. We add *dog* to B, and remove all sets from $\mathcal{S}$ that contain *dog*. Now, $\mathcal{S} = \emptyset$, so we're done. Our computed answer is thus $B = \{cat, dog\}$.

4.7.3 D▶Reduce▶Holte's 1R

Name: *Holte1RReducer*

Description: Returns all singleton attribute sets, inspired by the paper of Holte [19]. The set of all 1R rules, i.e., univariate decision rules, are indirectly returned as a child of the returned set of singleton reducts.

Signature: *DecisionTable* $\longrightarrow$ *Reducts*

4.7.4 D▶Reduce▶Manual reducer...

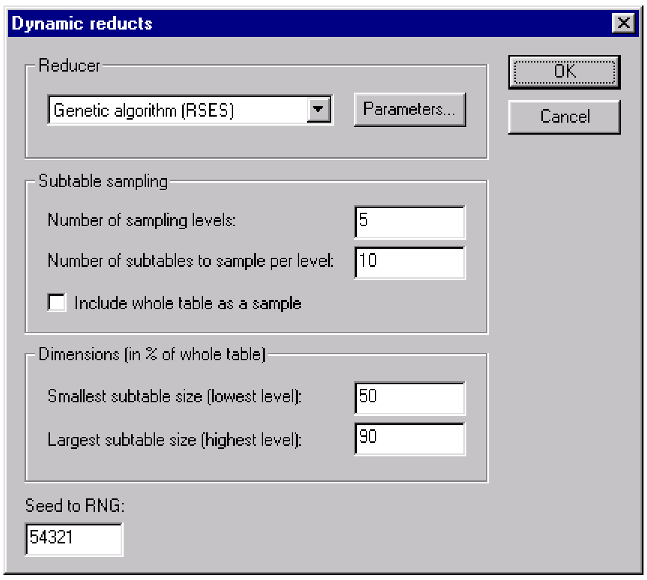
Name: *ManualReducer*

Description: Enables the user to manually specify an attribute subset that can be used as a reduct in subsequent computations.

Dialogs: Figure 15.

Keywords: ATTRIBUTES (*AttributeList*)

Signature: *DecisionTable* $\longrightarrow$ *Reducts*


Dynamic reducts

Reducer: Genetic algorithm (RSES)

Subtable sampling

Number of sampling levels: 5

Number of subtables to sample per level: 10

☐ Include whole table as a sample

Dimensions (in % of whole table)

Smallest subtable size (lowest level): 50

Largest subtable size (highest level): 90

Seed to RNG: 54321

Figure 16: Dialog box for computing dynamic reducts.

4.7.5 D▶Reduce▶Dynamic reducts (RSES)...

Name: *RSESDynamicReducer*

Description: Implements dynamic reducts as defined by Bazan et al. [7, 6]. A number of subtables are randomly sampled from the input table, and proper reducts are computed from each of these using some algorithm. The reducts that occur the most often across subtables are in some sense the most "stable".

TIP Computing dynamic reducts can be extremely computationally intensive, even for moderately sized tables. Be wary about initiating very lengthy reduction processes.

TIP Only other RSES reducers can be used as "workhorses" to compute proper reducts from the sampled subtables.

Dialogs: Figure 16, Figure 13, Figure 7.

Keywords: SEED (*Integer*), LEVELS (*Integer*), SAMPLES (*Integer*), PERCENTAGE.LOWER (*Integer*), PERCENTAGE.UPPER (*Integer*), INCLUDEWHOLE (*Boolean*), REDUCER (*Id*).

Signature: $RSESDecisionTable \rightarrow RSESReducts$

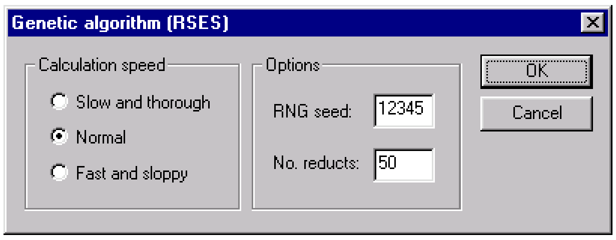
Example: Consider 5 sampling levels, with 10 subtables per level. Let the subtable sizes vary between 50% and 90% of the input table, which has $|U|$ objects. These sampling parameters would generate subtables of the sizes given below, 50 subtables in total.

10 different subtables with 50% $\times$ $|U|$ objects.
10 different subtables with 60% $\times$ $|U|$ objects.
10 different subtables with 70% $\times$ $|U|$ objects.
10 different subtables with 80% $\times$ $|U|$ objects.
10 different subtables with 90% $\times$ $|U|$ objects.

In addition, we can choose to include the input table itself as a sample.

4.7.6 D▶Reduce▶Exhaustive calculation (RSES)...

Name: *RSESExhaustiveReducer*


Genetic algorithm (RSES)

Calculation speed: Slow and thorough [✓ Normal] Fast and sloppy

Options: RNG seed: 12345; No. reducts: 50

Figure 17: Dialog box for providing parameters to the RSES genetic algorithm.

Description: Computes all reducts by brute force. No support is provided for IDGs, boundary region thinning or approximate solutions.

TIP This algorithm does not scale up well, and is only suitable for tables of moderate size. Computing all reducts is NP-hard.

Dialogs: Figure 13, Figure 7.

Keywords: DISCERNIBILITY ({Full, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), MODULO.DECISION (*Boolean*).

Signature: *RSESDecisionTable* → *RSESReducts*

4.7.7 D▶Reduce▶Johnson's algorithm (RSES)...

Name: *RSESJohnsonReducer*

Description: Invokes the RSES implementation of the greedy algorithm of Johnson [20]. See also Section 4.7.2.

No support is provided for IDGs, boundary region thinning or approximate solutions.

Dialogs: Figure 13, Figure 7.

Keywords: DISCERNIBILITY ({Full, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), MODULO.DECISION (*Boolean*).

Signature: *RSESDecisionTable* → *RSESReducts*

4.7.8 D▶Reduce▶Genetic algorithm (RSES)...

Name: *RSESGeneticReducer*

Description: Implements a variation of the algorithm described by Wróblewski [42]. Uses a genetic algorithm to search for reducts, either until the search space is exhausted or until a given maximum number of reducts have been found.

Three predefined parameter settings can be chosen among that control the thoroughness and speed of the genetic search procedure. No support is provided for IDGs, boundary region thinning or approximate solutions.

Dialogs: Figure 17, Figure 13, Figure 7.

Keywords: DISCERNIBILITY ({Full, Object}), SELECTION ({All, Value, File, Index}), SELECTION.ATTRIBUTE (*String*), SELECTION.VALUE (*String*), SELECTION.FILENAME (*String*), SELECTION.INDEX (*Integer*), MODULO.DECISION (*Boolean*), NUMBER (*Integer*), SEED (*Integer*), SPEED ({Fast, Normal, Slow}).

Signature: *RSESDecisionTable* → *RSESReducts*

4.8 D▶Classify...

Name: *BatchClassifier*

Description: Classifies all objects in a decision table using one of the classifiers described in Section 14. A confusion matrix is constructed, and, optionally, ROC information is returned. Several ASCII files may be produced as side-effects along the way, each giving a detailed breakdown of some aspect of the classification process.

If the selected classifier cannot come up with a decision, a fallback or default decision may be specified. A typical fallback choice is the most frequently occurring decision class. A certainty coefficient can be associated with the chosen fallback, and should typically be set to the a priori probability of the selected fallback decision. This only has practical relevance if a ROC curve is to be generated.

In some cases the selected classifier may flag that several alternative decision values are possible for an object, each one with a different certainty coefficient associated with it. The options in this category determine how such situations should be dealt with:

- *Best:* Classifies the object as belonging to the decision class that has the highest degree of certainty associated with it, as determined by the selected classifier.
- *Prioritize:* Classifies the object according to the motto "if the classifier says that the object could belong to a specific decision class (even though it is not very likely), then classify it as such". This option might be desirable to use if we deal with very rare events. In such cases, one might want to prioritize a specified decision class (the rare event or events that are more important to detect than others). This can be further generalized by requiring that the associated certainty coefficient must be above some threshold in order for prioritization to take effect.
- *Refrain:* Refrains from classifying objects for which the selected classifier indicates multiple possible decisions.

TIP Currently, all classifiers except the one described in Section 14.4 have support for multiple decisions and certainty coefficients.

Classification details for each object may be logged to a file. This file contains a detailed breakdown of the performance and output of the selected classifier.

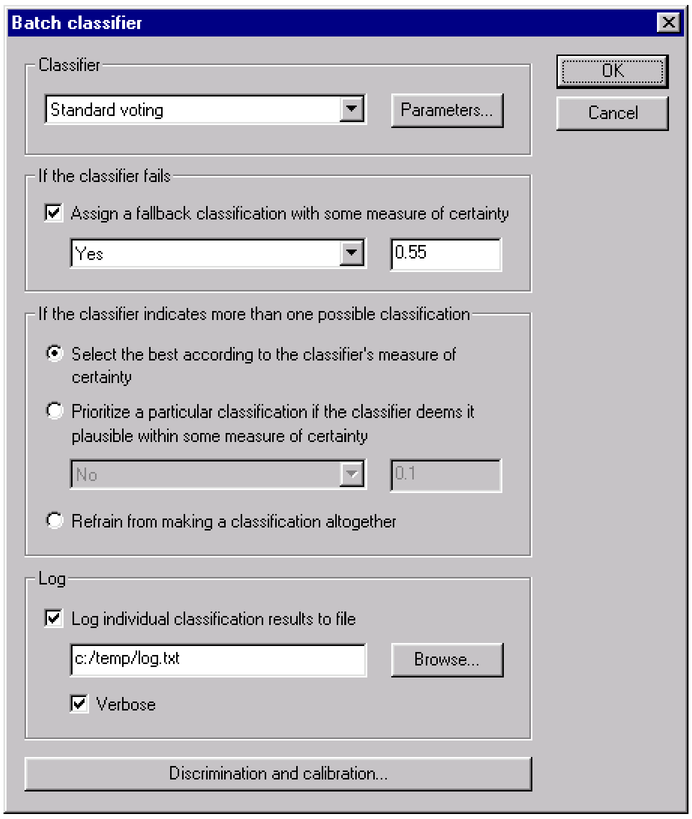
If the selected classifier has support for certainty coefficients, data points for ROC curves may be generated for a selected "focus" class. (For an introduction to ROC analysis, see Øhrn [26, chapter 7] and references therein.) If we have more than two decision classes, a virtual binarization implicitly takes place. All objects from decision classes different from the focus class are treated as the same class 0, while objects from the focus class are treated as class 1. Note that for ROC purposes then, we do not distinguish between misclassifications between objects that belong to different decision classes than the focus class. Such a misclassification is in this context in fact treated as a correct classification.

If the selected classifier has support for certainty coefficients, data points for calibration plots may be generated for a selected "focus" class. (For an introduction to calibration plots, see Øhrn [26, chapter 7] and references therein.) If there are more than two decision classes, a virtual binarization implicitly takes place, as discussed above. A calibration plot file also contains information about the Brier score [9,8,33] and its covariance decomposition [5], and the linear regression equation of the plot's data points.

TIP With a group size of 1, the calibration plot file will contain a list of the individual certainty coefficients. If the number of groups specified exceeds the number of objects in the table, a group size of 1 is used. See also Appendix A.8.

Dialogs: Figure 18, Figure 19.

Keywords: CLASSIFIER (*Id*), FALLBACK (*Boolean*), FALLBACK.CLASS (*String*), FALLBACK.CERTAINTY (*Float*), MULTIPLE ({Best, Prioritize, Refrain}), MULTIPLE.CLASS (*String*), MULTIPLE.THRESHOLD (*Float*), LOG (*Boolean*),


Batch classifier

Classifier: Standard voting | Parameters...

If the classifier fails

[✓] Assign a fallback classification with some measure of certainty

Yes | 0.55

If the classifier indicates more than one possible classification

(●) Select the best according to the classifier's measure of certainty

( ) Prioritize a particular classification if the classifier deems it plausible within some measure of certainty

No | 0.1

( ) Refrain from making a classification altogether

Log

[✓] Log individual classification results to file

c:/temp/log.txt | Browse...

[✓] Verbose

Figure 18: Dialog box for providing batch classification parameters.

LOG.FILENAME (*String*), LOG.VERBOSE (*Boolean*), CALIBRATION (*Boolean*), CALIBRATION.CLASS (*String*), CALIBRATION.FILENAME (*String*), CALIBRATION.GROUPS (*Integer*), ROC (*Boolean*), ROC.CLASS (*String*), ROC.FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *BatchClassification*

Example: If we want to apply a classifier to a set of objects for which no outcome is known, a dummy decision attribute has to be appended to the input decision table. Make sure the option to log the classification details to file is selected. The suggested decisions for each object will be logged to the specified file. The resulting confusion matrix will not make much sense in such a situation, and only serves to summarize the contents of the log file.

4.9 D▶Other

4.9.1 D▶Other▶Import reducts...

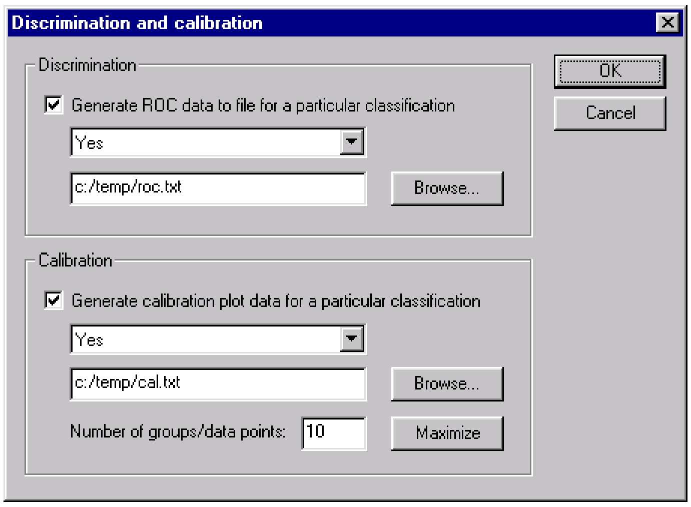
Name: *MyReductImporter*

Description: A collection of user-defined attribute subsets can be imported into ROSETTA from an ASCII file. The file format is documented in Appendix A.3.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* $\longrightarrow$ *Reducts*


Discrimination and calibration

Discrimination

☑ Generate ROC data to file for a particular classification

Yes

c:/temp/roc.txt

Calibration

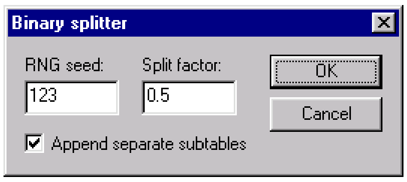
☑ Generate calibration plot data for a particular classification

Yes

c:/temp/cal.txt

Number of groups/data points: 10

Figure 19: Dialog box for providing ROC and calibration plot parameters.


Binary splitter

RNG seed: 123

Split factor: 0.5

☑ Append separate subtables

Figure 20: Dialog box for randomly splitting a decision table in two.

4.9.2 D▶Other▶Split in two...

Name: *BinarySplitter*

Description: Splits a decision table with a universe U of objects in two disjoint and randomly sampled subtables, with universes U_1 and U_2 respectively.

$$U = U_1 \cup U_2 \qquad U_1 \cap U_2 = \emptyset \tag{13}$$

The proportion $|U_1|/|U|$ can be specified, as well as the seed to the random number generator. The sampled subtables are normally appended to the input table's child list as separate entities.

Dialogs: Figure 20.

Keywords: SEED (*Integer*), FACTOR (*Float*), APPEND (*Boolean*).

Signature: *DecisionTable* → {*DecisionTable*, *DecisionTables*}

4.9.3 D▶Other▶Partition...

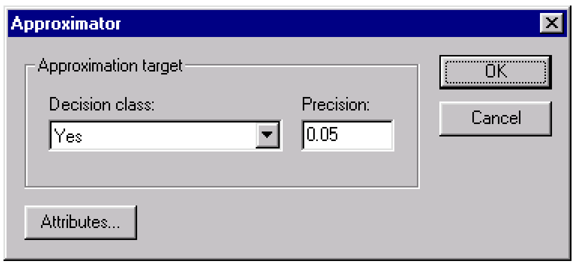
Name: *Partitioner*

Description: Takes as input a decision table and computes its equivalence classes of objects wrt. a given set of attributes.

Dialogs: Figure 15.

Keywords: ATTRIBUTES (*AttributeList*).

Signature: *DecisionTable* → *EquivalenceClasses*



Decision class: Yes

Precision: 0.05

Figure 21: Dialog box for computing rough set approximations.

4.9.4 D▶Other▶Approximate decision class...

Name: *Approximator*

Description: Computes a rough set approximation of a specified decision class, using the variable precision model [44, 43].

TIP In command scripts, the decision class is specified via the integer coding used internally. Integer codes can be looked up in, e.g., the data dictionary associated with the decision table.

Dialogs: Figure 21, Figure 15.

Keywords: DECISION (*Integer*), PRECISION (*Float*), ATTRIBUTES (*AttributeList*).

Signature: *DecisionTable* $\longrightarrow$ *Approximation*

4.10 D▶Execute

Name: *Executor*

Description: This family of algorithms enables us to execute simple command scripts, thus offering the possibility to automate lengthy and repetitive command sequences.

The commands to execute are assumed supplied in a script file, described in Appendix A.4. A command is a pair consisting of an algorithm name (or its description) and a parameter set, each residing on separate subsequent lines. How these commands are executed depends on the selected member of the *Executor* family.

The result of successfully executing a command script is a log file, and whatever files the commands may have produced as side-effects. Optionally, the command script may return the last produced structure of a specified type.

TIP If no structure return type is specified or is set to **Undefined**, then the script's input structure itself is returned. Otherwise, the last produced structure that matches the specified type is returned. Approximate type matches are supported.

Dialogs: Figure 22.

4.10.1 D▶Execute▶Pipeline script...

Name: *SerialExecutor*

Description: Implements pipeline execution, i.e., the command script is interpreted as defining a "flow" of algorithm applications. Thus, if our script describes a sequence of algorithms $A_1, \ldots, A_n$ and we apply it to the input structure S, we compute:

$$A_n(A_{n-1}(\cdots(A_2(A_1(S)))\cdots)) \tag{14}$$

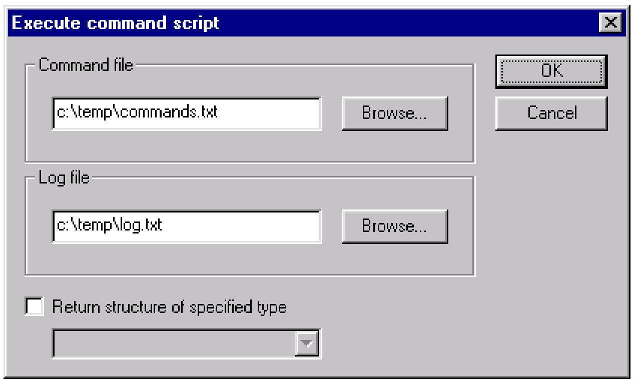


Figure 22: Dialog box for executing command scripts.

The format for command scripts is described in Appendix A.4.

TIP Make sure the algorithms' type signatures match up, i.e., that algorithm A_{i+1} is applicable to the output of algorithm A_i, and that A_1 is applicable to S.

Dialogs: Figure 22.

Keywords: FILENAME.COMMANDS (*String*), FILENAME.LOG (*String*), OUTPUT (*Id*).

Signature: *Structure* → *Structure*

Example: The command script file below, assumed applied to a decision table, defines a small pipeline. First, the table is discretized, before all reducts are computed. Rules are subsequently generated and exported to Prolog format.

```
RSESOrthogonalScaler
  {MODE = Save; FILENAME = c:/temp/cuts.txt}
RSESExhaustiveReducer
  {DISCERNIBILITY = Object}
RuleGenerator
  {}
PrologRuleExporter
  {FILTERING = c:/temp/rules.txt}
```

Note that an algorithm may take more parameters than we supply, as discussed in Appendix A.4.

4.10.2 D▶Execute▶Pipeline script with CV...

Name: *CVSerialExecutor*

Description: Implements *k*-fold cross-validation (CV), where the training and test pipelines are defined via command scripts. Input to the algorithm is a decision table. The format for command scripts is described in Appendix A.4.

The command script is read and split into two pipelines, one for training and one for testing. The user currently has to specify the length of the training pipeline, so that the split can be done at the correct location.

The following process is performed:

1. Sample two disjoint tables from the input decision table: A training table and a test table.
2. Feed the training table into the training pipeline, as described in Section 4.10.1. The training pipeline is assumed to produce a set of rules along the way.

3. Feed the test table into the test pipeline, as described in Section 4.10.1. The test pipeline is assumed to produce a batch classification structure along the way, typically by an algorithm from Section 4.8. The set of rules produced by the training pipeline is implicitly passed to that algorithm in the test pipeline.
4. Collect statistics from the batch classification structure produced by the test pipeline, and dump these to a log file.
5. Repeat the steps above k times in all, at each iteration systematically varying the sampling in step 1.
6. Compute summary statistics, and dump these to a log file.

With standard CV sampling, each object in the input decision table is guaranteed to be in the training table $k-1$ times and in the test table once. With inverted sampling, what would usually be the training table for an iteration becomes the test table, and vice versa. With inverted sampling, each object is thus guaranteed to be in the training table once and in the test table $k-1$ times.

TIP All occurrences of the string '`#ITERATION#`' in a parameter set are substituted by the index of the current CV iteration.

TIP It is possible to use this algorithm for classifiers that don't produce if-then rules, too. For such classifiers, simply produce a dummy rule set in the training pipeline.

Dialogs: Figure 23, Figure 22.

Keywords: FILENAME.COMMANDS (*String*), FILENAME.LOG (*String*), OUTPUT (*Id*), NUMBER (*Integer*), INVERT (*Boolean*), LENGTH (*Integer*), SEED (*Integer*).

Signature: *DecisionTable* → *Structure*

Example: The CV command script below specifies the following procedure to the performed k times: First, sample a training table and a testing table for the current iteration. (This is done implicitly.) A model is then built by discretizing the table before computing reducts and rules. To test the model, we discretize the test table, using the same cuts that were used to discretize the training table. The objects in the discretized test table are then classified, implicitly using the rules generated during training.

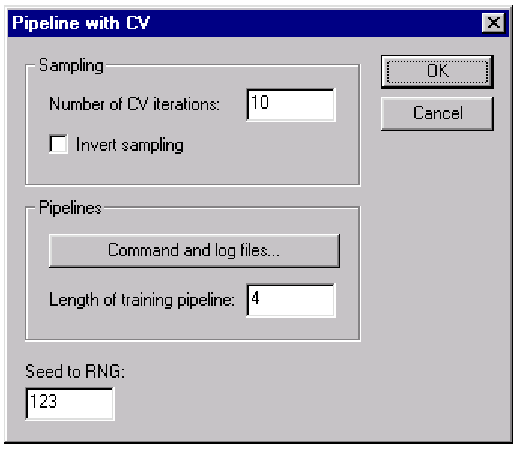
```
RSESOrthogonalScaler
  {MODE = Save; FILENAME = c:/temp/cuts.txt}
RSESExhaustiveReducer
  {DISCERNIBILITY = Object}
RuleGenerator
  {}

RSESOrthogonalFileScaler
  {MODE = Load; FILENAME = c:/temp/cuts.txt}
BatchClassifier
  {CLASSIFIER = StandardVoter}
```

The length of the training pipeline is 3, which has to be specified. After all k iterations are done, the log file contains the summarized performance statistics.

If we in the script above want to, e.g., store the cuts for each iteration, then all occurrences of '`cuts.txt`' in the script could be replaced with '`cuts#ITERATION#.txt`'. In the case of $k=3$, this would produce files named '`cuts0.txt`', '`cuts1.txt`' and '`cuts2.txt`'.

4.10.3 D▶Execute▶Batch script...

Name: *ParallelExecutor*



Number of CV iterations: 10
Invert sampling: [ ]
Length of training pipeline: 4
Seed to RNG: 123

Figure 23: Dialog box for executing CV command scripts.

Description: Implements batch execution of commands. If our script describes a set of algorithms $A_1, \ldots, A_n$ and we apply it to the input structure S, we compute:

$$A_1(S), A_2(S), \ldots, A_{n-1}(S), A_n(S) \tag{15}$$

The format for command scripts is described in Appendix A.4.

TIP Make sure the algorithms' type signatures match up, i.e., that S will be accepted as input to every algorithm A_i.

TIP Each algorithm A_i could, e.g., be a *SerialExecutor* algorithm as described in Section 4.10.1. Conceptually, this would have the effect of executing several "parallel" pipelines.

Dialogs: Figure 22.

Keywords: FILENAME.COMMANDS (*String*), FILENAME.LOG (*String*), OUTPUT (*Id*).

Signature: *Structure* $\longrightarrow$ *Structure*

Example: The command script below, assumed applied to a decision table, defines a small batch execution task. The script simply exports the table to three different formats.

```
PrologDecisionTableExporter
  {FILENAME = c:/temp/table1.pl}
MatlabDecisionTableExporter
  {FILENAME = c:/temp/table2.txt}
MyDecisionTableExporter
  {FILENAME = c:/temp/table3.txt}
```

4.11 D▶Statistics...

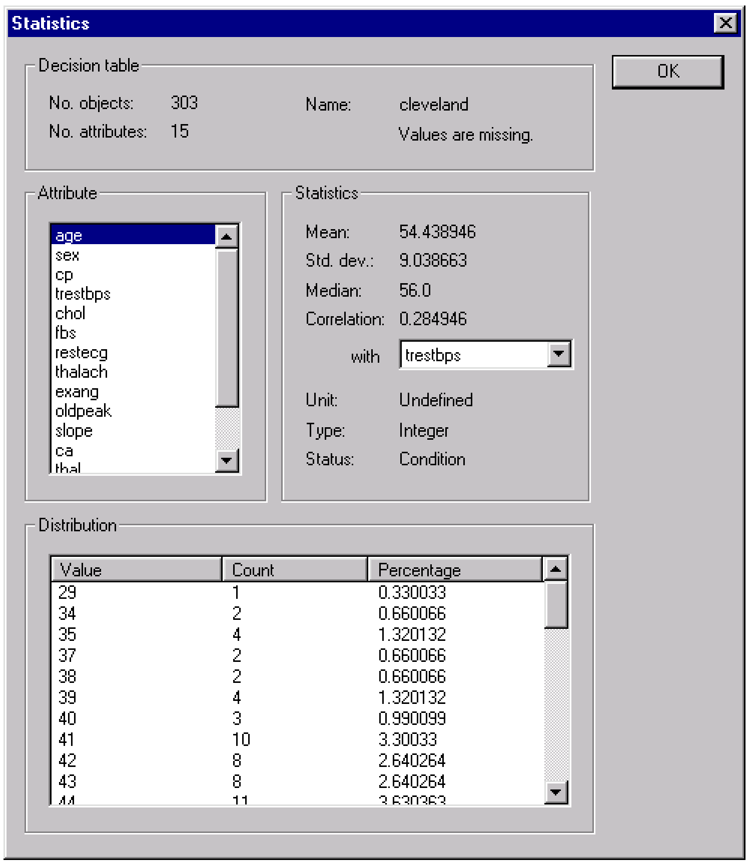
Description: Pops up a dialog where miscellaneous table statistics can be examined. Mean and median values, standard deviations and correlations are shown.

TIP For string attributes, the correlations are computed on the basis of the internal integer coding of the value set. See Section 4.1.

The value distribution of each attribute can also be viewed.

TIP Left-clicking on a header in a list control sorts the data. Right-clicking in a list control enables data to be copied to the Windows clipboard.

Dialogs: Figure 24.



Statistics

Decision table

No. objects: 303

No. attributes: 15

Name: cleveland

Values are missing.

Attribute

- age
- sex
- cp
- trestbps
- chol
- fbs
- restecg
- thalach
- exang
- oldpeak
- slope
- ca
- thal

Statistics

Mean: 54.438946

Std. dev.: 9.038663

Median: 56.0

Correlation: 0.284946

with trestbps

Unit: Undefined

Type: Integer

Status: Condition

Distribution

Value	Count	Percentage
29	1	0.330033
34	2	0.660066
35	4	1.320132
37	2	0.660066
38	2	0.660066
39	4	1.320132
40	3	0.990099
41	10	3.30033
42	8	2.640264
43	8	2.640264
44	11	3.630363

Figure 24: Dialog box for viewing table statistics.

5 Decision Tables D

Name: *DecisionTables*

Description: A compound structure holding a collection of decision tables.

5.1 D▶View...

Description: Brings up the views of all decision tables contained in the collection.

6 Reducts R

Name: *Reducts*

Description: Contains a collection of reducts. A reduct can be perceived as a vector of column indices into a decision table.

6.1 R▶View...

Description: Brings up a view of the reduct set, enabling the data to be inspected in detail.

TIP Right-click on a column header to sort the reducts.

6.2 R▶Export

Name: *ReductExporter*

Description: This family of algorithms comprises routines that export reduct sets to files in some format. The algorithms function as simple identity pass-through routines, where the exporting is a side-effect of applying the algorithm.

6.2.1 R▶Export▶XML format...

Name: *XMLReductExporter*

Description: Exports a set of reducts to XML format.

TIP XML greatly increases interoperability with other programs, and you can transform the XML into other formats using the XSLT language. Many current browsers also support viewing of XML documents.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $Reducts \longrightarrow Reducts$

Example: The reduct set $\{\{boats, planes, trains\}, \{cars, trains\}\}$ might be represented as the following XML fragment:

```
<reducts name="example">
  <reduct support="1" type="Full" modulo="false">
    <attribute name="boats"/>
    <attribute name="planes"/>
    <attribute name="trains"/>
  </reduct>
  <reduct support="1" type="Full" modulo="false">
    <attribute name="cars"/>
    <attribute name="trains"/>
  </reduct>
</reducts>
```

6.2.2 R▶Export▶Prolog format...

Name: *PrologReductExporter*

Description: Exports a set of reducts as a set of Prolog facts. The reducts are named "r_n", where n is the index of the reduct in the reduct set. Index counts start at 1.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $Reducts \longrightarrow Reducts$

Example: The reduct set $\{\{boats, planes, trains\}, \{cars, trains\}\}$ would be exported as the following set of facts:

```
reduct(r1, boats).
reduct(r1, planes).
reduct(r1, trains).

reduct(r2, cars).
reduct(r2, trains).
```

Option	Remove if
Length	$l \leq \lvert B \rvert \leq u$
Support	$l \leq support(B) \leq u$
Attribute	$a \in B$

Table 2: Options for removal of a reduct B from a reduct set. Here, l and u denote specified range parameters, while a is a fixed attribute identified by its index.

6.2.3 R▶Export▶Plain format...

Name: *MyReductExporter*

Description: Exports a set of reducts to an ASCII file in a plain and simple format. The inverse of the algorithm described in Appendix A.3.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Reducts* $\longrightarrow$ *Reducts*

6.3 R▶Filter

Name: *ReductFilter*

Description: Spans procedures that remove elements from reduct sets, according to different evaluation criteria. Unless explicitly stated, algorithms in this family modify their input directly.

6.3.1 R▶Filter▶Basic filtering...

Name: *MyReductFilter*

Description: Removes individual reducts from a reduct set. Possible removal criteria are listed in Table 2.

More than one removal criterion may be combined to define a compound criterion. The removal decision of the compound criterion may be negated, if specified.

Dialogs: Figure 25.

Keywords: FILTERING ({*Integer*, Length, Support, Attribute}), CONNECTIVE ({Or, And}), LENGTH.LOWER (*Integer*), LENGTH.UPPER (*Integer*), SUPPORT.LOWER (*Integer*), SUPPORT.UPPER (*Integer*), ATTRIBUTE (*Integer*), INVERT (*Boolean*).

Signature: *Reducts* $\longrightarrow$ *Reducts*

6.3.2 R▶Filter▶Cost filtering...

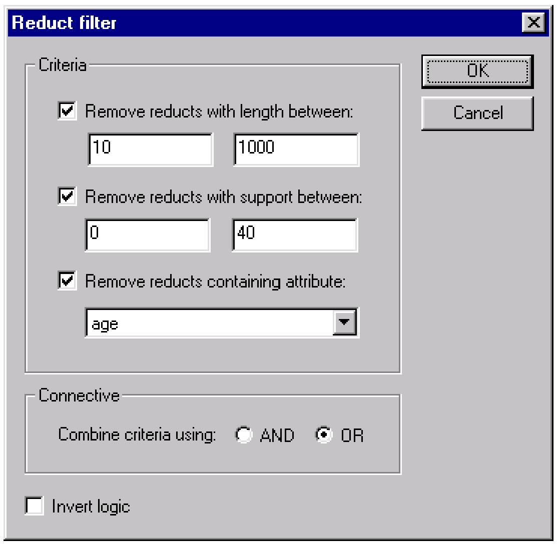
Name: *ReductCostFilter*

Description: Removes reducts from a reduct set according to their "cost". The function *cost* specifies the cost of attribute subset B. The file format for specifying cost information is defined in Appendix A.6.

$$cost(B) = \sum_{a \in B} cost(a) \tag{16}$$

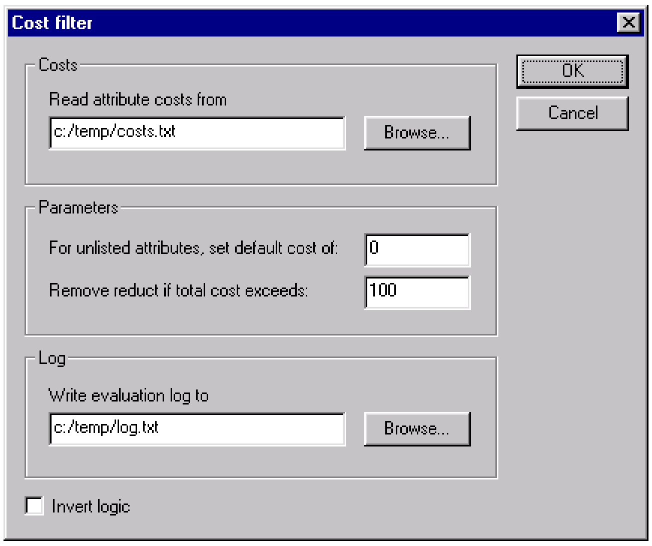
If a reduct's cost exceeds some specified threshold, the reduct is scheduled for removal. The removal decision may be negated, if specified.

The log file contains a detailed breakdown and ranking of the reducts.



☑ Remove reducts with length between: 10 1000
☑ Remove reducts with support between: 0 40
☑ Remove reducts containing attribute: age
Combine criteria using: AND [✓ OR]
☐ Invert logic

Figure 25: Dialog box for simple filtering of reducts.



Read attribute costs from: c:/temp/costs.txt
For unlisted attributes, set default cost of: 0
Remove reduct if total cost exceeds: 100
Write evaluation log to: c:/temp/log.txt
☐ Invert logic

Figure 26: Dialog box for cost filtering of reducts.

TIP Cost filtering may be of use if each attribute represents a test of some kind, and there is some cost associated with acquiring the attribute. In a medical setting, e.g., an invasive procedure might be more costly than a non-invasive one, and some tests may involve expensive drugs while others do not.

TIP By specifying an unreachable filtering threshold, this algorithm can be used for cost evaluation of reducts without filtering any of them away.

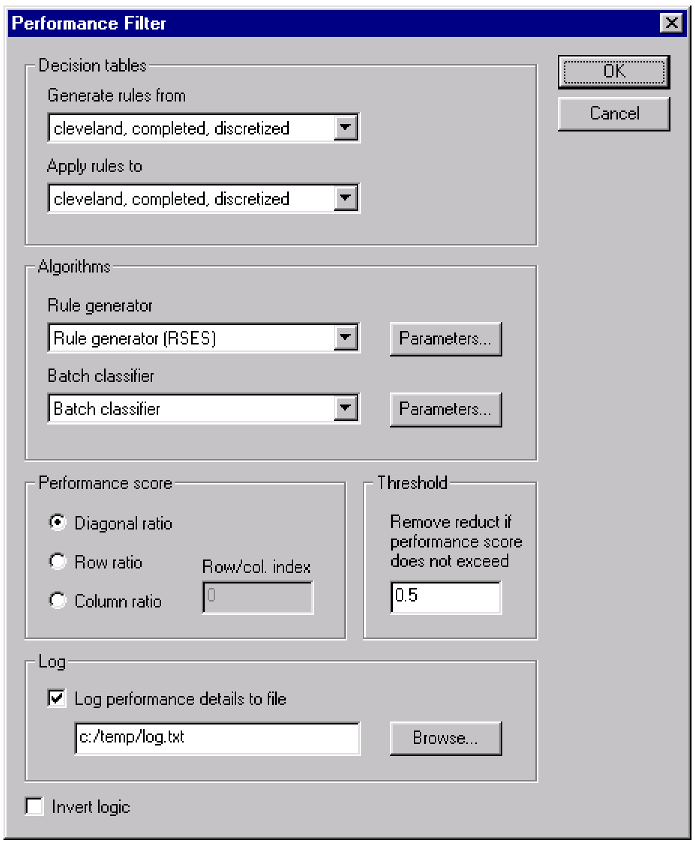
Dialogs: Figure 26.

Keywords: COST.FILENAME (*String*), DEFAULT (*Float*), THRESHOLD (*Float*), LOG.FILENAME (*String*), INVERT (*Boolean*).

Signature: *Reducts* $\longrightarrow$ *Reducts*

6.3.3 R▶Filter▶Performance filtering...

Name: *ReductPerformanceFilter*



Performance Filter

Decision tables

Generate rules from: cleveland, completed, discretized

Apply rules to: cleveland, completed, discretized

Algorithms

Rule generator: Rule generator (RSES)

Batch classifier: Batch classifier

Performance score: [✓ Diagonal ratio] Row ratio Column ratio

Row/col. index: 0

Threshold

Remove reduct if performance score does not exceed: 0.5

Log

[✓] Log performance details to file: c:/temp/log.txt

[ ] Invert logic

Figure 27: Dialog box for performance filtering of reducts.

Description: Each reduct in the reduct set is evaluated according to the classificatory performance of the rules generated from that reduct alone. The reduct is removed if the performance score does not exceed a specified threshold.

The following process is performed for each reduct:

1. Generate a set of rules from the reduct, using a given rule generator algorithm and a given decision table.
2. Classify all objects in another given decision table, using the generated set of rules and a specified classifier.
3. Harvest a performance score from the resulting confusion matrix. A performance score can be the classification accuracy, or the ratio between a diagonal element of the matrix and the corresponding row or column sum.

The log file contains a detailed breakdown of the performance of each reduct.

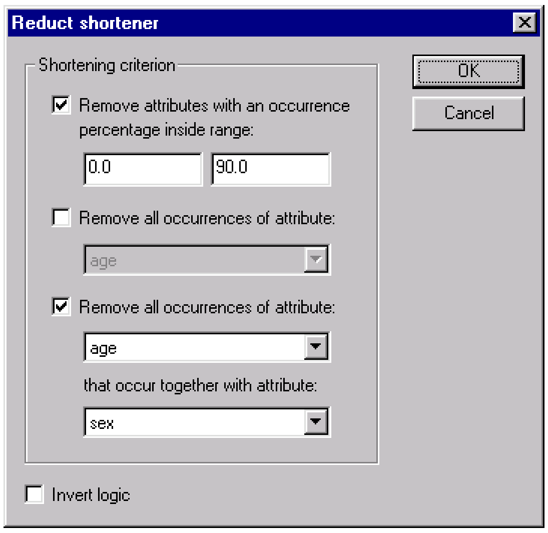
For an example of the algorithm's use, see Vinterbo et al. [39].

TIP By specifying an unreachable filtering threshold, this algorithm can be used for performance evaluation of reducts without filtering any of them away.

Dialogs: Figure 27, Figure 18, Figure 19, Figure 29.

Keywords: RULEGENERATOR.DECISIONTABLE (*String*), BATCHCLASSIFIER.DECISIONTABLE (*String*), RULEGENERATOR (*Id*), BATCHCLASSIFIER (*Id*), RATIO ({Diagonal, Row, Column}), INDEX (*Integer*), THRESHOLD (*Float*), FILENAME (*String*), INVERT (*Boolean*).

Signature: $Reducts \longrightarrow Reducts$


Reduct shortener

Shortening criterion

[✓] Remove attributes with an occurrence percentage inside range: 0.0 90.0

[ ] Remove all occurrences of attribute: age

[✓] Remove all occurrences of attribute: age that occur together with attribute: sex

[ ] Invert logic

Figure 28: Dialog box for simple shortening of reducts.

6.3.4 R▶Filter▶Remove reducts with no rules (RSES)

Name: *RSESRulelessReductFilter*

Description: Removes a reduct from the reduct set if there are no rules derived from that reduct.

Keywords: INVERT (*Boolean*).

Signature: *RSESReducts* $\longrightarrow$ *RSESReducts*

6.3.5 R▶Filter▶Basic shortening...

Name: *MyReductShortener*

Description: Removes attributes from reducts according to a set of selected criteria. An attribute is removed if one or more of the criteria apply. The removal decision may be negated, if specified.

TIP Elimination of duplicate reducts after shortening is not implemented.

Dialogs: Figure 28.

Keywords: SHORTENING ({*Integer*, Percentage, Occurrence, Combined}), PERCENTAGE.LOWER (*Float*), PERCENTAGE.UPPER (*Float*), ATTRIBUTE (*Integer*), PRIMARY (*Integer*), SECONDARY (*Integer*), INVERT (*Boolean*).

Signature: *Reducts* $\longrightarrow$ *Reducts*

6.4 R▶Generate rules...

Name: *RSESRuleGenerator*

Description: Generates rules from a set of reducts. Conceptually, this is done by overlaying each reduct in the reduct set over the reduct set's parent decision table, and reading off the values.

If the reduct set already has a rule set attached to it, this child rule set is returned. Otherwise, and if the reducts are of type "full discernibility", a new set of rules is generated.

TIP Since the RSES library does not provide support for IDGs, the rules' support counts are overly conservative if the reducts are generated using anything other than strict inequality to define the discernibility predicate.[8]

[8]Contact aleks@idi.ntnu.no if you need a workaround.

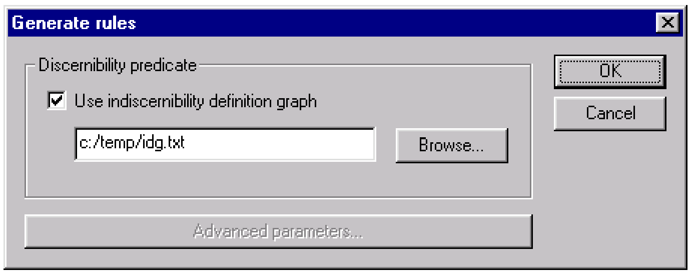
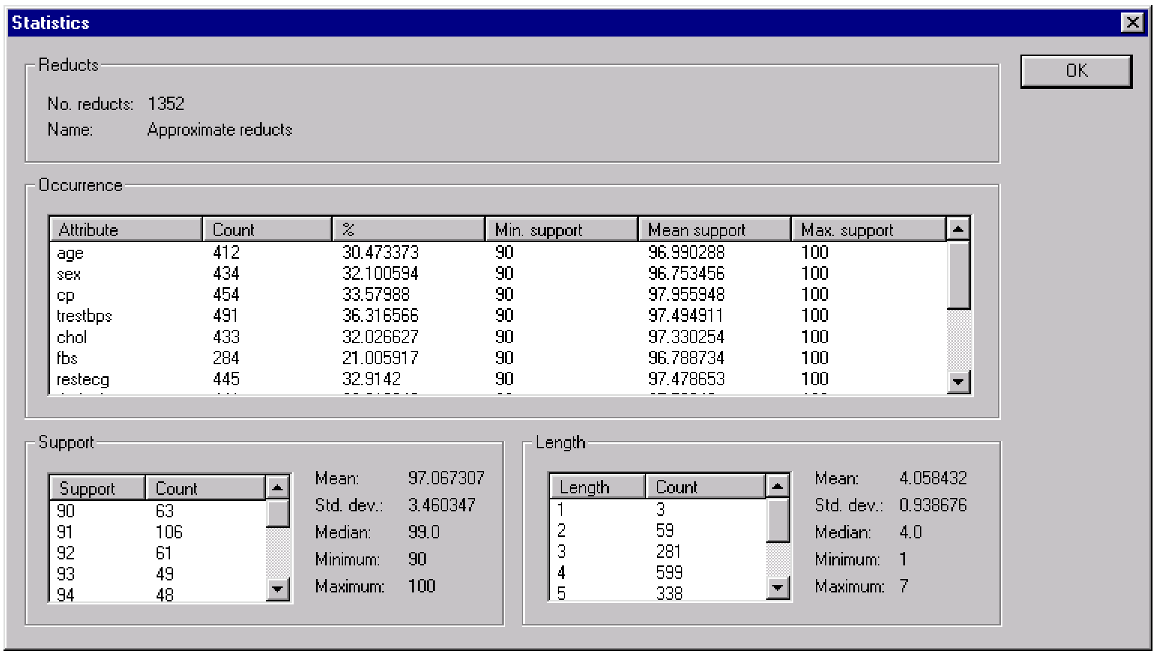


Figure 29: Dialog box for generating rules.



No. reducts: 1352
Name: Approximate reducts

Attribute	Count	%	Min. support	Mean support	Max. support
age	412	30.473373	90	96.990288	100
sex	434	32.100594	90	96.753456	100
cp	454	33.57988	90	97.955948	100
trestbps	491	36.316566	90	97.494911	100
chol	433	32.026627	90	97.330254	100
fbs	284	21.005917	90	96.788734	100
restecg	445	32.9142	90	97.478653	100

Support	Count
90	63
91	106
92	61
93	49
94	48

Mean: 97.067307
Std. dev.: 3.460347
Median: 99.0
Minimum: 90
Maximum: 100

Length	Count
1	3
2	59
3	281
4	599
5	338

Mean: 4.058432
Std. dev.: 0.938676
Median: 4.0
Minimum: 1
Maximum: 7

Figure 30: Dialog box for viewing reduct statistics.

Dialogs: Figure 29.

Keywords: IDG (*Boolean*).

Signature: *RSESReducts* → *RSESRules*

6.5 R▶Statistics...

Description: Pops up a dialog where miscellaneous reduct set statistics can be examined.

TIP Left-clicking on the column headers in a list control sorts the data. Right-clicking in a list control enables data to be copied to the Windows clipboard.

Dialogs: Figure 30.

7 Rules

R

Name: *Rules*

Description: Contains a collection of rules. A rule can be perceived as two vectors of descriptors, where each descriptor is an attribute index into a decision table and an element from that attribute's value set.

7.1 R▶View...

Description: Brings up a view of the rule set, enabling the data to be inspected in detail.

TIP Right-click on a column header to sort the rules.

7.2 R▶Export

Name: *RuleExporter*

Description: This family of algorithms comprises routines that export rule sets to files in some format. The algorithms function as simple identity pass-through routines, where the exporting is a side-effect of applying the algorithm.

7.2.1 R▶Export▶XML format...

Name: *XMLRuleExporter*

Description: Exports a set of rules to XML format.

TIP XML greatly increases interoperability with other programs, and you can transform the XML into other formats using the XSLT language. Many current browsers also support viewing of XML documents.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Rules* $\longrightarrow$ *Rules*

Example: The following XML fragment represents an example decision rule:

```
<rule>
  <if support="5" coverage="0.208333">
    <and>
      <descriptor attribute="Icon" value="Y"/>
      <descriptor attribute="Word" value="[1, 4)"/>
    </and>
  </if>
  <then>
    <or>
      <decision support="1" accuracy="0.2" coverage="0.0909091">
        <descriptor attribute="Type" value="C"/>
      </decision>
      <decision support="4" accuracy="0.8" coverage="0.307692">
        <descriptor attribute="Type" value="U"/>
      </decision>
    </or>
  </then>
</rule>
```

7.2.2 R▶Export▶Prolog format...

Name: *PrologRuleExporter*

Description: Exports a set of rules to an ASCII file as a set of Prolog clauses. The head of each exported clause contains various numerical information associated with the rule. An inconsistent rule is split into several individually consistent rules. Intervals are exported using a numerical syntax.

TIP This enables one to, e.g., directly employ induced rules as part of a larger expert system.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Rules* $\longrightarrow$ *Rules*

Example: An inconsistent rule indicating two possible decisions might be exported as the following Prolog clauses:

```
disease(X, no, 112, 0.848485, 0.000001) :-
  exang(X, no),
  ca(X, V1),
    V1 < 1.

disease(X, yes, 20, 0.151515, 0.0) :-
  exang(X, no),
  ca(X, V1),
    V1 < 1.
```

See the exported file for a legend to the clause heads.

7.2.3 R▸Export▸C++ format...

Name: *CPPRuleExporter*

Description: Exports a set of rules to C++ code that realizes a classifier. Conflict resolution among firing rules is done via standard voting.

TIP This enables an induced classifier to be embedded into your own C++ code.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Rules* $\longrightarrow$ *Rules*

Example: The generated code defines two classes, *ROSETTAObject* and *ROSETTAClassifier*. In a small user-written "driver program", these classes could be used together as follows:

```
ROSETTAObject object;

object.Age      = 63;
object.Sex      = object.LookupSex("Male");
object.Cp       = object.LookupCp("Typical angina");
object.Trestbps = 145;
object.Chol     = 233;
object.Fbs      = object.LookupFbs("True");
object.Restecg  = object.LookupRestecg("LV hypertrophy");
object.Thalach  = 150;
object.Exang    = object.LookupExang("No");
object.Oldpeak  = 2.3;
object.Slope    = object.LookupSlope("Downsloping");
object.Ca       = 0;
object.Thal     = object.LookupThal("Fixed defect");

ROSETTAClassifier classifier;

int votes[2];

int no_matches = classifier.Classify(object, votes);
```

Option	Remove if
RHS Support	$l \leq support(\alpha \cdot (d = v)) \leq u$
RHS Accuracy	$l \leq accuracy(\alpha \rightarrow (d = v)) \leq u$
RHS Coverage	$l \leq coverage(\alpha \rightarrow (d = v)) \leq u$
RHS Stability	$l \leq stability(\alpha \rightarrow (d = v)) \leq u$
Decision	$v = v'$
LHS Length	$l \leq length(\alpha) \leq u$
Condition	$(a = v')$ occurs in α

Table 3: Options for removal of a rule $\alpha \rightarrow (d = v)$ from a rule set. Here, l and u denote specified range parameters. See Øhrn [26, chapter 6] for a legend.

```
cout << "no_matches = " << no_matches << endl;
cout << "votes[0]   = " << votes[0]   << endl;
cout << "votes[1]   = " << votes[1]   << endl;
```

For details, see the comments in the generated C++ code.

7.2.4 R▶Export▶Plain format...

Name: *MyRuleExporter*

Description: Exports a set of rules to an ASCII file in a plain and simple format.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *Rules* $\rightarrow$ *Rules*

7.3 R▶Filter

Name: *RuleFilter*

Description: Spans procedures that remove rules from rule sets, according to different evaluation criteria. Unless explicitly stated, algorithms in this family modify their input directly.

7.3.1 R▶Filter▶Basic filtering...

Name: *MyRuleFilter*

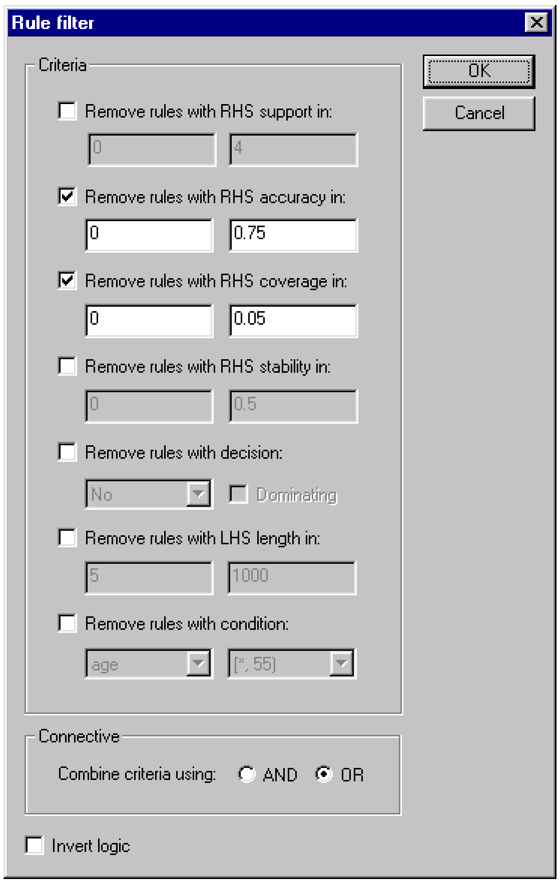
Description: Removes individual rules or patterns from a rule set. Possible removal criteria are listed in Table 3.

More than one removal criterion may be combined to define a compound criterion. The removal decision of the compound criterion may be negated, if specified.

Dialogs: Figure 31.

Keywords: FILTERING ({*Integer*, RHS Support, RHS Accuracy, Decision, LHS Length, Condition, RHS Stability, RHS Coverage}), CONNECTIVE ({And, Or}), SUPPORT.RHS.LOWER (*Integer*), SUPPORT.RHS.UPPER (*Integer*), ACCURACY.RHS.LOWER (*Float*), ACCURACY.RHS.UPPER (*Float*), COVERAGE.RHS.LOWER (*Float*), COVERAGE.RHS.UPPER (*Float*), STABILITY.RHS.LOWER (*Float*), STABILITY.RHS.UPPER (*Float*), DECISION (*Integer*), DOMINATE (*Boolean*), LENGTH.LHS.LOWER (*Integer*), LENGTH.LHS.UPPER (*Integer*), ATTRIBUTE (*Integer*), VALUE (*Integer*), INVERT (*Boolean*).

Signature: *Rules* $\rightarrow$ *Rules*


Rule filter

Criteria

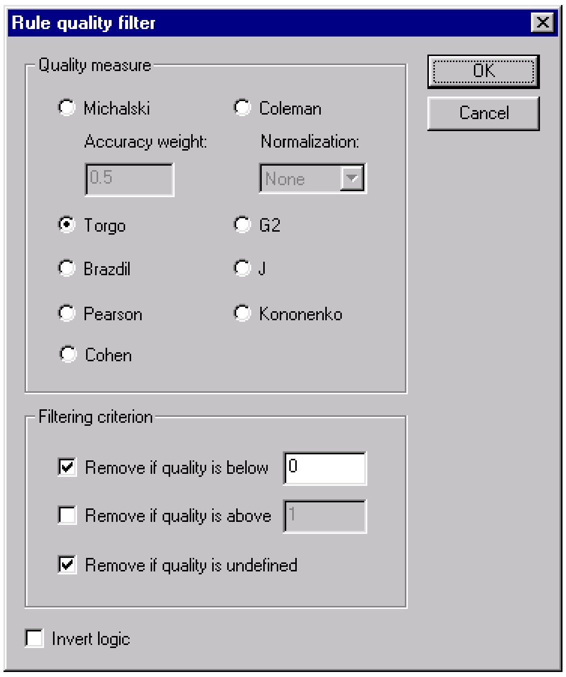
- [ ] Remove rules with RHS support in: 0 4
- [✓] Remove rules with RHS accuracy in: 0 0.75
- [✓] Remove rules with RHS coverage in: 0 0.05
- [ ] Remove rules with RHS stability in: 0 0.5
- [ ] Remove rules with decision: No [ ] Dominating
- [ ] Remove rules with LHS length in: 5 1000
- [ ] Remove rules with condition: age [*, 55)

Connective

Combine criteria using: AND [✓ OR]

- [ ] Invert logic

Figure 31: Dialog box for simple filtering of rules.


Rule quality filter

Quality measure

( ) Michalski
Accuracy weight: 0.5

( ) Coleman
Normalization: None

[✓ Torgo] ( ) G2
( ) Brazdil ( ) J
( ) Pearson ( ) Kononenko
( ) Cohen

Filtering criterion

☑ Remove if quality is below 0
☐ Remove if quality is above 1
☑ Remove if quality is undefined

☐ Invert logic

Figure 32: Dialog box for quality filtering of rules.

7.3.2 R▶Filter▶Quality filtering...

Name: *QualityRuleFilter*

Description: Filters away rules according to various measures of rule quality. For definitions of the implemented quality measures, see Bruha [10] or Ågotnes [1].

Dialogs: Figure 32.

Keywords: FILTERING ({Michalski, Torgo, Brazdil, Pearson, G2, J, Cohen, Coleman, Kononenko}), FILTERING.BIAS (*Float*), FILTERING.NORMALIZATION ({None, C1, C2}), UPPER.THRESHOLD (*Float*), LOWER.THRESHOLD (*Float*), REMOVE.UNDEFINED (*Boolean*), INVERT (*Boolean*).

Signature: *Rules* $\longrightarrow$ *Rules*

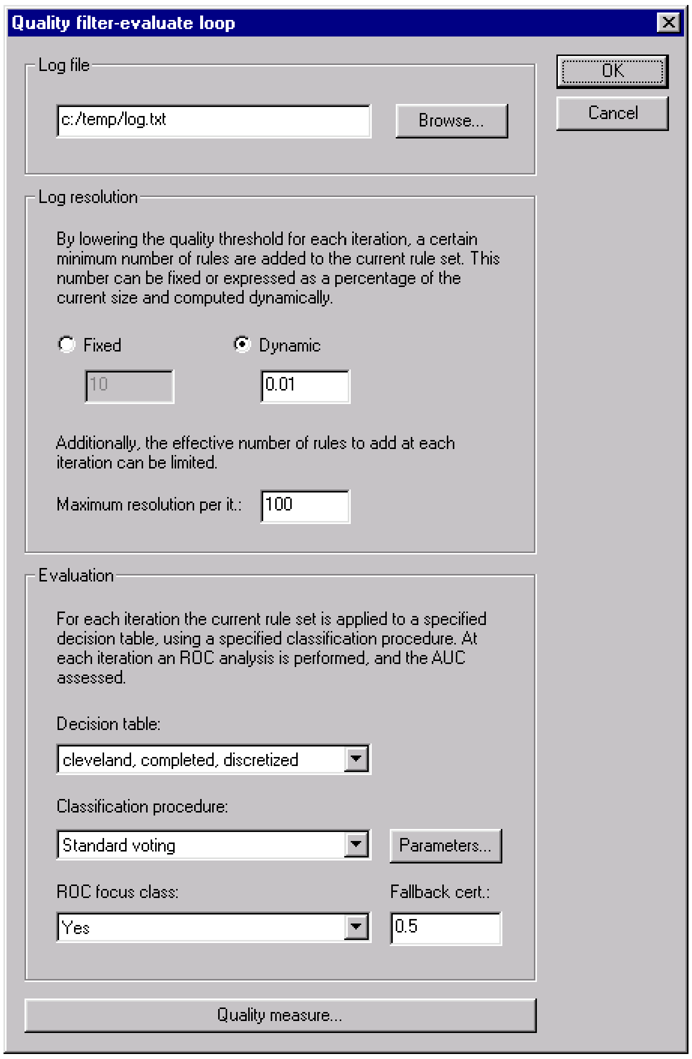
7.3.3 R▶Filter▶Quality filtering loop...

Name: *QualityRuleFilterLoop*

Description: Couples the filtering scheme from Section 7.3.2 together with ROC analysis. Enables the classificatory performance of a set of rules to be monitored as a function of the quality threshold.

The following process is performed:

1. Given a quality measure, compute the quality of each rule in the rule set. Dump the qualities to a log file.
2. Let $R = \emptyset$ and $t = \infty$.
3. Lower t so that a certain minimum number of new rules have qualities above t. Add these rules to R.
4. Use R to classify the objects in a given decision table, and note the resulting area under the ROC curve.
5. Dump t, $|R|$, and various ROC performance information to a log file.
6. If R contains all rules, exit. Otherwise, goto step 3.


Quality filter-evaluate loop

Log file

c:/temp/log.txt

Log resolution

By lowering the quality threshold for each iteration, a certain minimum number of rules are added to the current rule set. This number can be fixed or expressed as a percentage of the current size and computed dynamically.

Fixed [✓ Dynamic]

Fixed: 10

Dynamic: 0.01

Additionally, the effective number of rules to add at each iteration can be limited.

Maximum resolution per it.: 100

Evaluation

For each iteration the current rule set is applied to a specified decision table, using a specified classification procedure. At each iteration an ROC analysis is performed, and the AUC assessed.

Decision table: cleveland, completed, discretized

Classification procedure: Standard voting

ROC focus class: Yes

Fallback cert.: 0.5

Figure 33: Dialog box for looped quality filtering of rules.

For an example of the algorithm's use, see Øhrn et al. [30] or Ågotnes et al. [2].

TIP The algorithm returns the original, unfiltered input rule set.

Dialogs: Figure 33, Figure 32.

Keywords: FILTERING ({Michalski, Torgo, Brazdil, Pearson, G2, J, Cohen, Coleman, Kononenko}), FILTERING.BIAS (*Float*), FILTERING.NORMALIZATION ({None, C1, C2}), REMOVE.UNDEFINED (*Boolean*), RESOLUTION ({Fixed, Dynamic}), RESOLUTION.THRESHOLD (*Integer*), RESOLUTION.GAP (*Integer*), RESOLUTION.FRACTION (*Float*), FILENAME (*String*), DECISIONTABLE (*String*), CLASSIFIER (*Id*), ROC.CLASS (*String*), FALLBACK.CERTAINTY (*Float*).

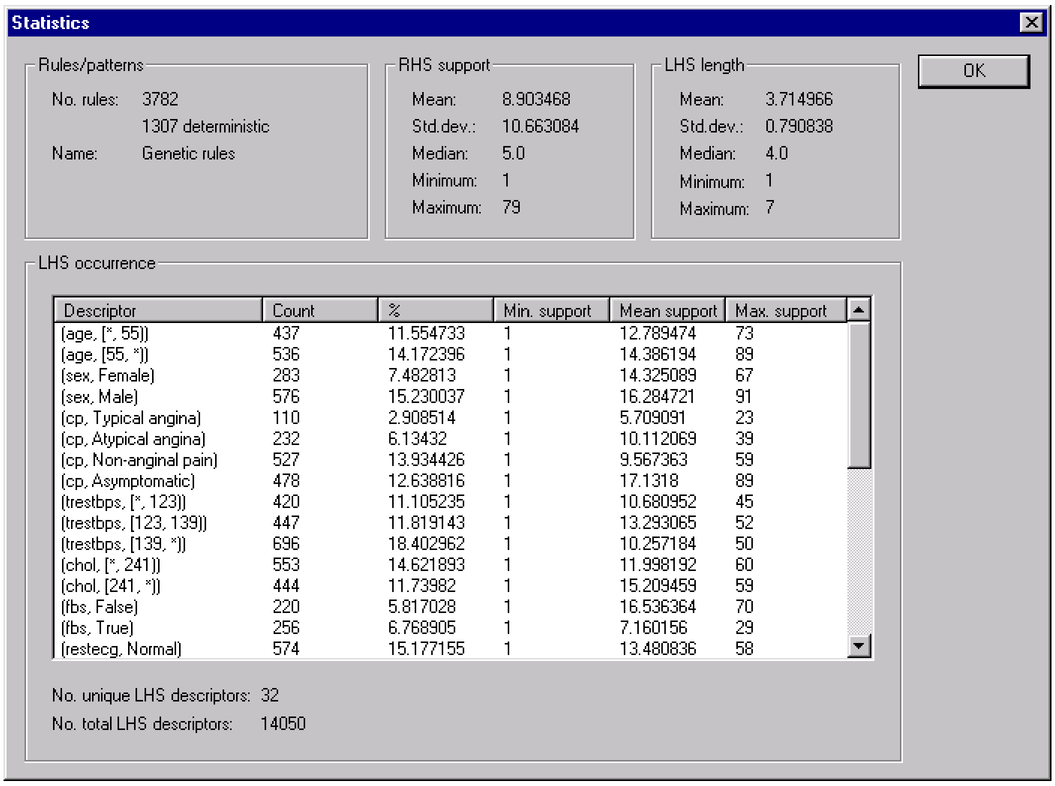
Signature: *Rules* → *Rules*

7.4 R ▶ Statistics...

Description: Pops up a dialog where miscellaneous rule set statistics can be examined.

TIP Left-clicking on a column header in a list control sorts the data. Right-clicking in a list control enables data to be copied to the Windows clipboard.

Dialogs: Figure 34.


Statistics

Rules/patterns

No. rules:	3782
	1307 deterministic
Name:	Genetic rules

RHS support

Mean:	8.903468
Std.dev.:	10.663084
Median:	5.0
Minimum:	1
Maximum:	79

LHS length

Mean:	3.714966
Std.dev.:	0.790838
Median:	4.0
Minimum:	1
Maximum:	7

OK

LHS occurrence

Descriptor	Count	%	Min. support	Mean support	Max. support
(age, [*, 55))	437	11.554733	1	12.789474	73
(age, [55, *))	536	14.172396	1	14.386194	89
(sex, Female)	283	7.482813	1	14.325089	67
(sex, Male)	576	15.230037	1	16.284721	91
(cp, Typical angina)	110	2.908514	1	5.709091	23
(cp, Atypical angina)	232	6.13432	1	10.112069	39
(cp, Non-anginal pain)	527	13.934426	1	9.567363	59
(cp, Asymptomatic)	478	12.638816	1	17.1318	89
(trestbps, [*, 123))	420	11.105235	1	10.680952	45
(trestbps, [123, 139))	447	11.819143	1	13.293065	52
(trestbps, [139, *))	696	18.402962	1	10.257184	50
(chol, [*, 241))	553	14.621893	1	11.998192	60
(chol, [241, *))	444	11.73982	1	15.209459	59
(fbs, False)	220	5.817028	1	16.536364	70
(fbs, True)	256	6.768905	1	7.160156	29
(restecg, Normal)	574	15.177155	1	13.480836	58

No. unique LHS descriptors: 32

No. total LHS descriptors: 14050

Figure 34: Dialog box for viewing rule statistics.

8 Patterns

Name: *Rules*

Description: A set of patterns is the same as a set of rules that have no consequents. Such structures may result from computing reducts and not taking the decision attribute into account.

8.1 P▶View...

Description: Brings up a view of the pattern set, enabling the data to be inspected in detail.

TIP Right-click on a column header to sort the patterns.

9 Batch Classifications

Name: *BatchClassification*

Description: Summarizes the results of a batch classification procedure. Holds a confusion matrix and some ROC-derived quantities, if relevant.

9.1 C▶View...

Description: Brings up a view of the batch classification structure, enabling the data to be inspected in detail.

10 Text Files F

Name: *TextFile*

Description: A wrapper around an ASCII file residing somewhere on the file system. Enables the file to be represented in the GUI.

TIP Objects of this type are simple wrappers and do not hold any actual data. Thus, if a text file icon is deleted from the project tree, the underlying file does not get deleted.

10.1 F▶View...

Description: Bring up a view of the underlying ASCII file. The file is read on demand.

11 Partitions P

Name: *EquivalenceClasses*

Description: Represents a set of equivalence classes. An equivalence class is a set of object indices into a parent decision table.

11.1 P▶View...

Description: Brings up a view of the partition, enabling the data to be inspected in detail.

TIP Object indices can be 0-based or 1-based. Select the **Use offset** menu option from the grid origin's pop-up menu to switch between viewing modes.

TIP Right-click on a column header to sort the data.

12 Approximations A

Name: *Approximation*

Description: Represents a rough set approximation. An approximation consists of several approximation regions, each region being a collection of equivalence classes. An equivalence class is a set of object indices into a parent decision table.

12.1 A▶View...

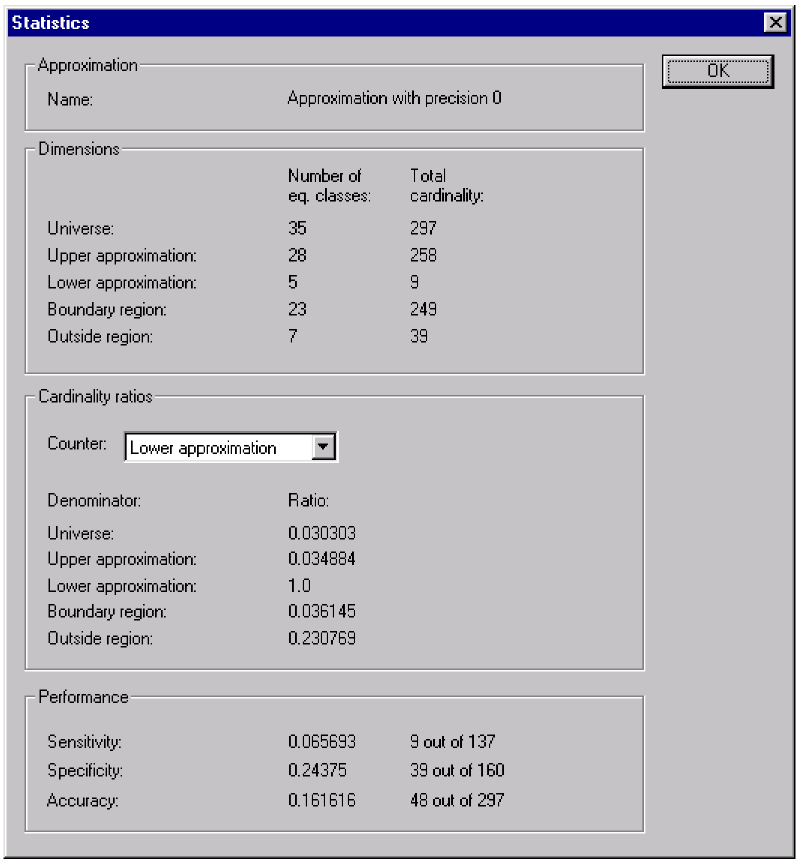
Description: Brings up a view of the approximation, enabling the data to be inspected in detail.

TIP Object indices can be 0-based or 1-based. Select the **Use offset** menu option from the grid origin's pop-up menu to switch between viewing modes.

12.2 A▶Statistics...

Description: Brings up a dialog enabling various statistics and numerical quantities of the rough set approximation to be viewed.

Dialogs: Figure 35.



Approximation

Name:	Approximation with precision 0

Dimensions

	Number of eq. classes:	Total cardinality:
Universe:	35	297
Upper approximation:	28	258
Lower approximation:	5	9
Boundary region:	23	249
Outside region:	7	39

Cardinality ratios

Counter: Lower approximation

Denominator:	Ratio:
Universe:	0.030303
Upper approximation:	0.034884
Lower approximation:	1.0
Boundary region:	0.036145
Outside region:	0.230769

Performance

Sensitivity:	0.065693	9 out of 137
Specificity:	0.24375	39 out of 160
Accuracy:	0.161616	48 out of 297

Figure 35: Dialog box for viewing approximation statistics.

13 Algorithms

Name: *Algorithm*

Description: Represents an installed algorithm. This branch of the project tree is seldom in practical use.

TIP The algorithm names and descriptions that appear in the project tree are the same ones that can be used in command scripts.

13.1 A ▶ Apply...

Description: Brings up a dialog box where an applicable structure in the current project can be selected. The selected structure is then used as input to the algorithm. If there is only one structure which the algorithm is applicable to, the dialog box is bypassed.

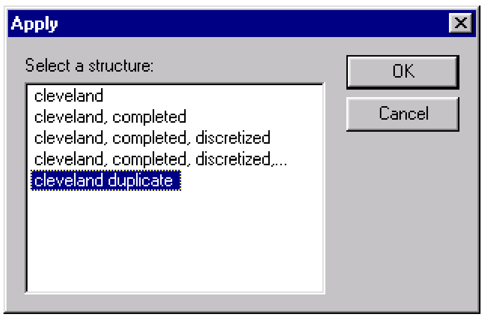
TIP Note that the following three actions are equivalent:

- From the pop-up menu of structure S, select an algorithm A to apply.
- From the pop-up menu of algorithm A, select **Apply...** and indicate S.
- Drag the icon of A and drop it onto the icon of S, or vice versa.

Dialogs: Figure 36.

14 Classifier Algorithms

Name: *Classifier*



Select a structure:

- cleveland
- cleveland, completed
- cleveland, completed, discretized
- cleveland, completed, discretized,...
- cleveland duplicate

Figure 36: Dialog box for selecting an input structure.

Name	Option	Description
Voting	*Support*	Cast a number of votes equal to the number of objects in the training set that match both α and $(d = v)$.
	Simple	Cast one vote.
Normalization	*Firing*	Divide the number of votes by the sum of all casted votes.
	All	Divide the number of votes by the sum of all votes for all rules.

Table 4: Options for specifying how many votes a firing rule $\alpha \rightarrow (d = v)$ gets to cast in favour of decision value v, and how the tallied votes are to be normalized.

Description: Algorithms in this family classify a single object in a decision table according to some classification strategy, and are typically used as components of batch classifiers as described in Section 4.8.

Classifier algorithms take as input an object's information vector, and return (unless otherwise stated) a list of possible decision classes ranked according to some measure of certainty.

14.1 Standard voting

Name: *StandardVoter*

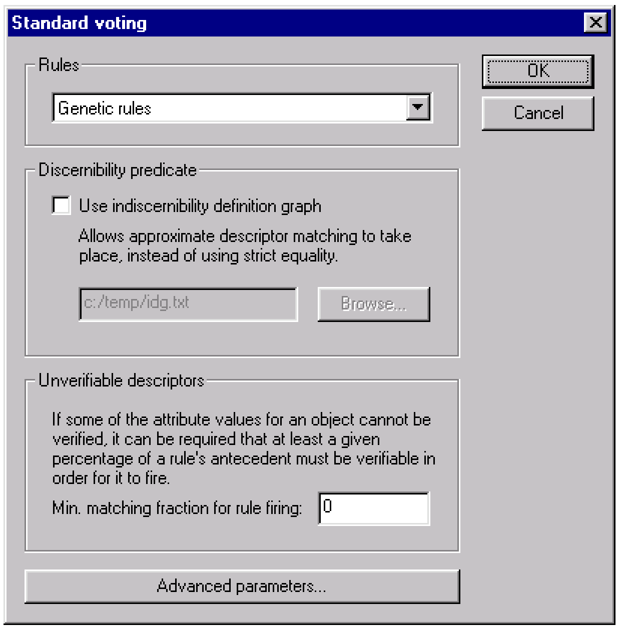
Description: Implements voting as described by Øhrn [26, pages 66-68], using a specified rule set.

In the firing step, a rule fires if its antecedent is not in conflict with the presented object, and if the percentage of verifiable terms in the antecedent is above a certain threshold t. IDGs can be used in the matching stage to allow for approximate matching of descriptors.

In some cases, and especially if the rules are generated as a result of dynamic reduct computation across subtables of varying sizes, it may happen that some rules are generalizations of other rules. If two or more of the rules in the rule set form a generalization hierarchy, the algorithm has an option to only let the most specific rule fire, i.e., exclude the generalizations from the set of firing rules.

TIP If a firing rule indicates more than one possible decision value, we may imagine this rule to be logically expanded to several rules, each with the same antecedent but with a single descriptor only as a consequent.

In the election process among the firing rules, each rule gets to cast a certain number of votes in favour of the decision value it indicates, according to a selected voting strategy. The certainty coefficient for each possible decision value is computed by dividing the total number of casted votes for each decision value by a normalization factor. The voting strategies and normalization options currently implemented are shown in Table 4.


Standard voting

Rules: Genetic rules

Discernibility predicate

☐ Use indiscernibility definition graph

Allows approximate descriptor matching to take place, instead of using strict equality.

c:/temp/idg.txt

Unverifiable descriptors

If some of the attribute values for an object cannot be verified, it can be required that at least a given percentage of a rule's antecedent must be verifiable in order for it to fire.

Min. matching fraction for rule firing: 0

Figure 37: Dialog box for setting voting parameters.

TIP If we normalize by the set of firing rules, the certainty coefficients over all possible decision values will sum to unity, and is equivalent to reporting the percentage of votes that each decision class has received, relative to the number of votes that have been cast.

Dialogs: Figure 37, Figure 38.

Keywords: RULES (*String*), FRACTION (*Float*), IDG (*Boolean*), IDG.FILENAME (*String*), SPECIFIC (*Boolean*), VOTING ({Support, Simple}), NORMALIZATION ({Firing, All}).

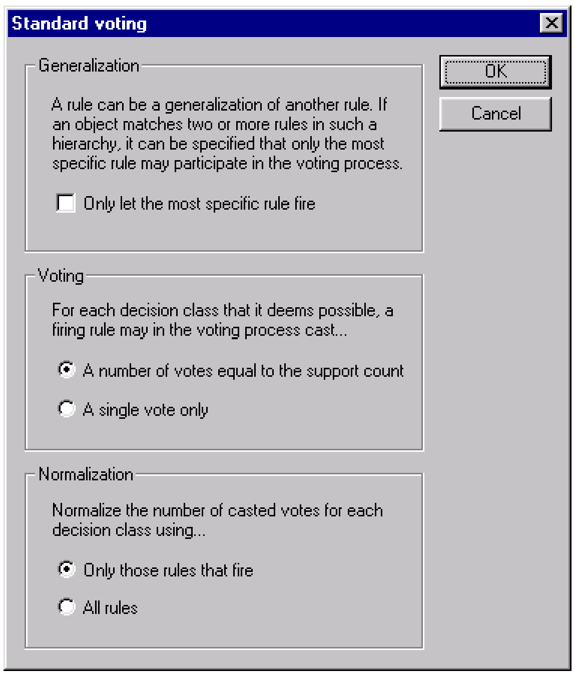
Signature: *InformationVector* → *RuleBasedClassification*

Example: The parameter t controls how unverifiable descriptors should be handled. A choice of $t = 1$ signifies intolerance to missing values, while $t = 0$ signifies complete tolerance. By varying the tolerance parameter t between these two extremes, one can control how conservatively the firing stage should behave.

Example: Let the specified rule set consist of the rules below, and assume that the tolerance level t has been set so that only rules 1, 2, 2′, 3 and 3′ fire when presented with an object x such that $a(x) = 0$ and $b(x) = 1$.

Rule	Antecedent		Consequent	Support
1	$(a = 0) \cdot (b = 1)$	$\rightarrow$	$(d = 0)$	3
2	$(b = 1) \cdot (c = 2)$	$\rightarrow$	$(d = 0)$	5
2′	$(b = 1) \cdot (c = 2)$	$\rightarrow$	$(d = 1)$	1
3	$(a = 0) \cdot (c = 2)$	$\rightarrow$	$(d = 0)$	7
3′	$(a = 0) \cdot (c = 2)$	$\rightarrow$	$(d = 1)$	2
4	$(c = 3)$	$\rightarrow$	$(d = 0)$	2
4′	$(c = 3)$	$\rightarrow$	$(d = 1)$	1
5	$(b = 2) \cdot (c = 2)$	$\rightarrow$	$(d = 1)$	8

There are eight rules in the total rule set, whereof the set of firing rules consists of the five rules mentioned above. The sum of the support counts in the total rule set is 29, while the sum of the support counts in the set of firing rules is 18. The returned certainty coefficients for each decision class would be computed as follows:


Standard voting

Generalization

A rule can be a generalization of another rule. If an object matches two or more rules in such a hierarchy, it can be specified that only the most specific rule may participate in the voting process.

☐ Only let the most specific rule fire

Voting

For each decision class that it deems possible, a firing rule may in the voting process cast...

[✓ A number of votes equal to the support count]

A single vote only

Normalization

Normalize the number of casted votes for each decision class using...

[✓ Only those rules that fire]

All rules

Figure 38: Dialog box for setting more voting parameters.

Options		Certainty factors		
Support	*Firing*	$certainty(x, (d = 0))$	15/18	0.83333
		$certainty(x, (d = 1))$	3/18	0.16667
Support	*All*	$certainty(x, (d = 0))$	15/29	0.51724
		$certainty(x, (d = 1))$	3/29	0.10354
Simple	*Firing*	$certainty(x, (d = 0))$	3/5	0.6
		$certainty(x, (d = 1))$	2/5	0.4
Simple	*All*	$certainty(x, (d = 0))$	3/8	0.375
		$certainty(x, (d = 1))$	2/8	0.25

Given the set of rules, the algorithm would thus suggest 0 to be the most likely decision value for object x. Whether this suggestion is followed is up to the calling batch classifier algorithm or the user to decide.

14.2 Voting with object tracking

Name: *ObjectTrackingVoter*

Description: Implements voting via object tracking as described by Øhrn [26, pages 68-69], using a specified rule set.

Dialogs: Figure 37.

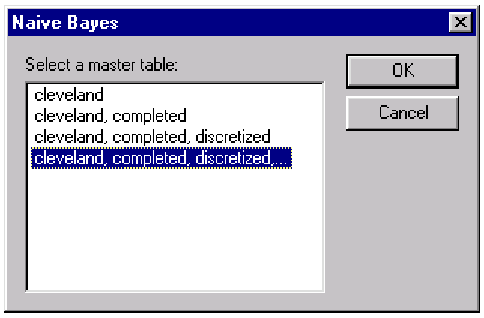
Keywords: RULES (*String*), FRACTION (*Float*), IDG (*Boolean*), IDG.FILENAME (*String*).

Signature: *InformationVector* $\longrightarrow$ *RuleBasedClassification*

14.3 Naive Bayes

Name: *NaiveBayesClassifier*

Description: Implements the naive Bayes classifier, see, e.g., Ripley [34] or Øhrn [26, pages 84–85]. For each decision class, computes the conditional probability that that decision class is the correct



cleveland
cleveland, completed
cleveland, completed, discretized
cleveland, completed, discretized...

Figure 39: Dialog box for the naive Bayes classifier.

one, given an object's information vector. Assumes that the object's attributes are independent. The probabilities involved in producing the final estimate are computed as frequency counts from a "master" decision table.

The naive Bayes classifier often works very well in practice, and excellent classification results may be obtained even when the probability estimates contain large errors [12].

TIP Even though this is not a rule-based classifier, it can be used in command scripts. See Section 4.10.2.

Dialogs: Figure 39.

Keywords: DECISIONTABLE (*String*).

Signature: *InformationVector* → *Classification*

14.4 Standard/tuned voting (RSES)

Name: *RSESClassifier*

Description: Offers rule-based classification based on voting, similar in spirit to the algorithm described in Section 14.1. Two main options are implemented:

- *Majority:* Similar to the algorithm from Section 14.1 with support-based voting, but with no tolerance for missing values. If any rules fire, the decision class that achieves the highest certainty factor is returned.
- *Tuned:* Allows the voting to incorporate user-defined "distance" values between decision classes. If x denotes the object to classify and there are different groups of decision rules R_i that recognize x and indicate decision value i, x will be assigned decision value k, where k satisfies the following condition:

$$w_j = \sum_i (|R_i| \times f(i, j)) \tag{17}$$

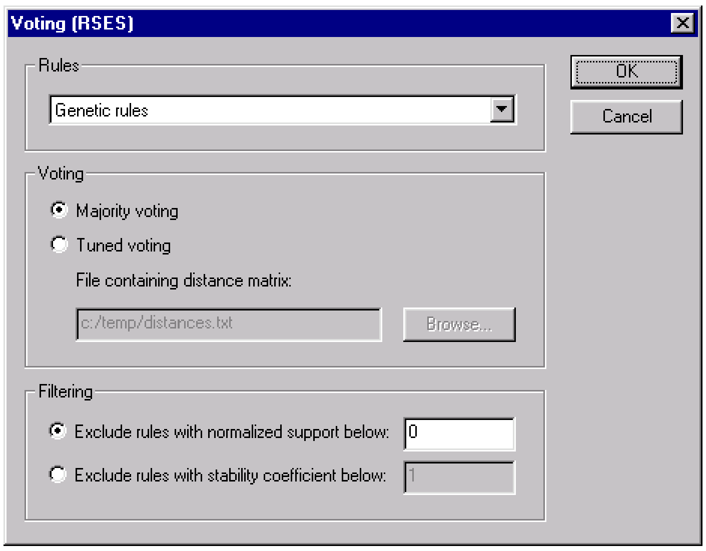
$$k = \arg\min_j w_j \tag{18}$$

The term $f(i, j)$ denotes the distance between decisions i and j, and is defined via an ASCII file described in Appendix A.7.

For both schemes, there is an option to exclude rules that fulfill certain filtering criteria to participate in the voting procedure.

TIP The RSES classifier does not have support for certainty coefficients, nor for returning multiple classifications. Some of the effects of this are discussed in Section 4.8.

Dialogs: Figure 40.


Voting (RSES)

Rules: Genetic rules

Voting:
- [✓ Majority voting]
- Tuned voting

File containing distance matrix: c:/temp/distances.txt

Filtering:
- [✓ Exclude rules with normalized support below:] 0
- Exclude rules with stability coefficient below: 1

Figure 40: Dialog box for setting RSES voting parameters.

Keywords: RULES (*String*), VOTING ({Tuned, Majority}), FILENAME (*String*), FILTERING ({Support, Stability}), THRESHOLD.SUPPORT (*Float*), THRESHOLD.STABILITY (*Float*).

Signature: *InformationVector* $\longrightarrow$ *Classification*

Example: To exemplify tuned voting as offered by the RSES classifier, consider the situation where $V_d = \{1, 2, 3\}$ and we have the following distances between the decision values:

$$f(1,2) = 1$$
$$f(2,3) = 2$$
$$f(1,3) = 3$$

Assume that there for a given object x are three groups of rules R_1, R_2 and R_3 that recognize it. Let the cardinalities of these be given by $|R_1| = 10$, $|R_2| = 5$ and $|R_3| = 2$. We then calculate the following quantities:

$$\begin{aligned} w_1 &= (10 \times 0) + (5 \times 1) + (2 \times 3) = 11 \\ w_2 &= (10 \times 1) + (5 \times 0) + (2 \times 2) = 14 \\ w_3 &= (10 \times 3) + (5 \times 2) + (2 \times 0) = 40 \end{aligned}$$

Hence, the would assign decision value 1 to object x.

15 Script Algorithms

Name: *ScriptAlgorithm*

Description: Algorithms in this family are only intended used as part of command scripts, as described in Section 4.10 and Appendix A.4.

15.1 *Loader*

Name: *Loader*

Description: Fills the input structure with data, before returning it. Enables files in internal ROSETTA format to be used in command scripts.

Keywords: FILENAME (*String*).

Signature: *Structure* $\longrightarrow$ *Structure*

15.2 *Saver*

Name: *Saver*

Description: Saves the input structure, before returning it. Enables files in internal ROSETTA format to be used in command scripts.

Keywords: FILENAME (*String*).

Signature: *Structure* → *Structure*

15.3 *StructureCreator*

Name: *StructureCreator*

Description: Creates and returns a new structure of the specified type. The input structure is ignored.

Keywords: OUTPUT (*Id*).

Signature: *Structure* → *Structure*

15.4 *Kidnapper*

Name: *Kidnapper*

Description: Returns the input structure's child structure number i.

Keywords: INDEX (*Integer*).

Signature: *Structure* → *Structure*

A Import File Formats

A.1 Dictionaries

Name: *DictionaryImporter*

Description: Hand-crafted dictionaries can be imported into ROSETTA from ASCII files. The format of the dictionary file is documented by the example below. See also Section 4.1 for an overview of attribute types, and the data associated with each type.

TIP Importing a new dictionary for a decision table does not alter the internal integer representation of the decision table, but only replaces its data dictionary.

TIP Blank lines and lines that start with the character '%' are ignored.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: $\{Dictionary, DecisionTable\} \rightarrow \{Dictionary, DecisionTable\}$

Example: The dictionary below states that the name of attribute 0 is "Radius", and that it is a float variable with a precision of 2 decimals, measured in "cm" units. The name of string attribute 1 is "Color", and the attribute maps the integer value 0 to "Red", 1 to "Green", 2 to "Blue" and 3 to "Yellow".

```
0.name       = Radius
0.unit       = cm
0.type       = Float
0.decimals   = 2

1.name       = Color
1.unit       = Undefined
1.type       = String
1.map.2      = Blue
1.map.1      = Green
1.map.0      = Red
1.map.3      = Yellow

2.name       = Year
2.unit       = Undefined
2.type       = Integer

3.name       = Grade
3.unit       = Undefined
3.type       = Float
3.decimals   = 1

4.name       = Sold
4.unit       = Undefined
4.type       = String
4.map.0      = No
4.map.1      = Yes
```

If we edit our dictionary entry for attribute 1 as shown below, we do not alter the colors of the objects, i.e., their internal coding remain the same even after the dictionary is imported. Rather, the fragment below simply redefines the names of the colors, and not the assignment of colors to objects.

```
1.name       = Color
1.unit       = Undefined
1.type       = String
1.map.2      = Green
1.map.1      = Blue
1.map.0      = Yellow
1.map.3      = Red
```

If we want to enforce a particular coding scheme, this can be achieved by importing the coded table with integer entries, and specifying that the attribute is of string type.[9] When the explicitly coded decision table is imported, the constructed data dictionary can be exported, edited to define "ordinary" color names, and then imported back into the system.

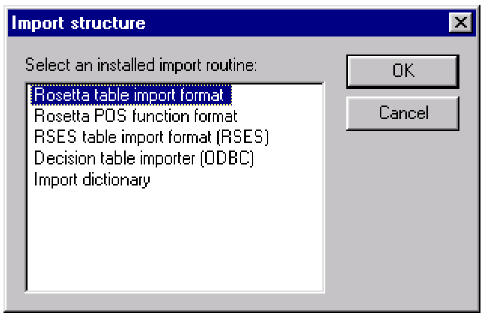
A.2 Decision Tables

Name: *DecisionTableImporter*

Description: Algorithms in this family fill a decision table with the contents specified in a file in alien format.

Dialogs: Figure 41, Figure 3.

[9]When dictionaries are automatically constructed during import of tables, string attributes try to assign strings their "natural" encoding, if possible. For example, the string "3" would be mapped to the number 3, if possible. Otherwise, strings are assigned integers on a first-come first-serve basis.


Import structure

Select an installed import routine:

Rosetta table import format
Rosetta POS function format
RSES table import format (RSES)
Decision table importer (ODBC)
Import dictionary

Figure 41: Dialog box for selecting a decision table import format.

A.2.1 Plain format

Name: *MyDecisionTableImporter*

Description: Tabular data can be imported into ROSETTA from ASCII files. The file should adhere to the following formatting rules:

- The first line contains attribute names.
- The second line contains attribute types, as described in Section 4.1. For floating point fields, the number of decimal points to use has to be specified, too.
- Missing values are indicated by the string **Undefined**.
- Table entries are separated by spaces, tabs or commas. Entries that contains any of these characters must be contained in double quotes.

TIP Blank lines and lines that start with the character '%' are ignored. The rightmost attribute is assigned decision status, all other attributes are assigned condition status.

TIP For numerical attributes, any string entry will be treated as a missing value.

TIP A data dictionary associated with the table is constructed automatically. (See Section 4.1.) For string attributes, integer values are generally assigned on a first-come first-serve basis. An example of how to enforce a user-defined coding scheme is given in Appendix A.1.

TIP If the first line consists of two integers, then these are interpreted by ROSETTA as table dimensions and are used as a hint to preallocate memory for the table, thus making the import routine slightly more efficient. Giving such a hint is optional. If the RSES library is present, the hint will have no effect.

Dialogs: Figure 41, Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* → *DecisionTable*

Example: The following decision table can be imported into ROSETTA:

```
Radius    Color     Year      Grade     Sold
Float(2)  String    Integer   Float(1)  String
3.14      Red       1970      1.0       No
2.71      Green     1492      1.5       Yes
10.666    Red       1814      2.0       Yes
0.99      Red       Undefined Undefined No
0.2       Blue      1776      3.5       No
Undefined Yellow    1865      2.5       No
4925.6    Undefined 1968      6.0       Yes
```

A.2.2 POS format

Name: *MyPOSDecisionTableImporter*

Description: Creates a decision table with 0/1 entries from a description of a Boolean POS function residing in an ASCII file, as described by Øhrn [26, page 50]. In addition, a decision attribute is added that specifies the object indices.

An informal grammar for specifying a POS function is given below. The function definition ends when the end of the file is reached, or when the string "`end`" is encountered on a separate line.

function	→	name '=' product
product	→	'(' sum ')'
product	→	product '*' product
sum	→	variable
sum	→	sum '+' sum

TIP Blank lines and lines that start with the character '%' are ignored.

TIP The current implementation does not handle extremely large function definitions very well.

TIP The ability to import function definitions as tables enables ROSETTA to be used for more general Boolean reasoning purposes. Note that the prime implicants of the imported function corresponds to the reducts relative to the first object in the resulting table.

Dialogs: Figure 41, Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* → *DecisionTable*

Example: The function definition below can be imported into ROSETTA. Note that the order the variables are listed in does not matter.

```
h =
  (boats + cars + planes) *
  (cars + boats) *
  (trains) *
  (planes + cars) *
  (trains + cars)
end
```

A.2.3 RSES format (RSES)

Name: *RSESDecisionTableImporter*

Description: Decision tables residing in legacy RSES ASCII files can be imported into ROSETTA. See Synak [38] for details on the file format.

TIP All attributes will be assigned string type.

Dialogs: Figure 41, Figure 3.

Keywords: FILENAME (*String*).

Signature: *RSESDecisionTable* → *RSESDecisionTable*

A.3 Reducts

Name: *MyReductImporter*

Description: A collection of user-defined attribute subsets can be imported into ROSETTA from an ASCII file. The following formatting rules apply:

- A reduct definition should span a single line, and must be enclosed in curly braces.
- Lines that do not define reducts are ignored, unless the line specifies a support count.
- The attribute names within each reduct are separated by commas.
- Tabs and spaces are ignored within each line.

By default, imported reducts are assigned a support count of 0. If another support count should be assigned to a reduct, a support count must be explicitly stated on the line following the reduct definition. Providing a support count is optional.

TIP The input decision table is required to convert attribute names to attribute indices. Make sure the reduct file is compatible with the decision table.

Dialogs: Figure 3.

Keywords: FILENAME (*String*).

Signature: *DecisionTable* → *Reducts*

Example: The reduct set specified below can be imported into ROSETTA. Note the format for explicitly specifying support counts.

```
{boats, planes, trains}
Support = [1 subtable(s)]
{cars, trains}
Support = [1 subtable(s)]
```

A.4 Command Scripts

Description: Command scripts can be executed by algorithms in the *Executor* family, described in Section 4.10. A command script is a sequence of commands. A command is a pair consisting of an algorithm name (or an algorithm description) and a parameter set, each residing on separate subsequent lines.

TIP Blank lines and lines that start with the character '%' are ignored.

TIP A command pair should occupy exactly two lines of text in the file.

An algorithm name or description in a command identifies one of the installed algorithms in the system. Algorithm naming is case-insensitive. Approximate matches on algorithm types are supported, i.e., "abstract superclasses" can be specified.

TIP The name of an algorithm is the name that is used in this document to refer to it. The description of an algorithm is the menu text that is displayed in the GUI when an algorithm in invoked from a pop-up menu. The names and descriptions of all installed algorithms can be read from the *Algorithms* branch in the GUI project tree.

A parameter set in a command is a list of zero or more keyword/value pairs, enclosed in curly braces. Each keyword/value pair is separated by the ';' character. A keyword is separated from its value by the '=' character. A keyword is case-insensitive, while a value is case-sensitive.

TIP The keywords relevant for an algorithm are listed in this document under the **Keywords** heading, together with the type its values may take on. To see which keyword/value pairs that were used by a computation, you can also inspect the resulting structure's annotation in the GUI.

TIP More keywords than the ones we care to supply may be relevant for an algorithm. If a relevant keyword is not supplied, then the last used value (or a default value) for that keyword is employed.

TIP Members of the *Executor* family are themselves algorithms. This means that a script can invoke other scripts.

Example: The lines below define two valid script file commands. Observe that algorithms that take no parameters are passed the empty parameter set.

```
BROrthogonalScaler
  {MODE = Save; FILENAME = c:\temp\cuts.txt; MASK = T}
Holte1RReducer
  {}
```

The parameter sets are here indented to increase readability. Leading or trailing whitespace is ignored.

A.5 IDG Information

Description: An IDG for an attribute a is a directed graph with the elements of V_a as nodes or vertices, and a set of edges $E_a \subseteq V_a^2$. ROSETTA can read a set of IDGs from an ASCII file. A collection of IDGs is simply several concatenated individual IDG definitions.

Each IDG is defined according to the format below. Attributes that are not listed in the IDG file are assumed to adhere to strict inequality, i.e., $E_a = \{(v, v) \mid v \in V_a\}$.

```
begin ⟨attribute-name⟩
   nodes ⟨vertex-set-specification⟩
   ⟨edge-specification_1⟩
   ⋮
   ⟨edge-specification_n⟩
end ⟨attribute-name⟩
```

TIP Blank lines and lines that start with the character '%' are ignored.

The user-defined components of an IDG specification for an attribute are:

- ⟨attribute-name⟩
  Identifies an attribute a by name. Attribute names are kept by ROSETTA in data dictionaries, associated with each information system.
- ⟨vertex-set-specification⟩
  A list of domain values separated by whitespace, whose union should define V_a.

  TIP The wildcard symbol '*' can be used as shorthand for all observed values for attribute V_a.

  TIP The missing value symbol $\top$ does not count as an observed value, but has to be specified explicitly.

  TIP The specification '$n\,..\,m$' can be used as shorthand for the range or set of integers $\{n, \ldots, m\}$.
- ⟨edge-specification$_i$⟩
  A command specifying some transformation of E_a according to the rules specified in Table 5.

$\langle$edge-specification$_i\rangle$	Semantics
`make-reflexive`	$E_a \leftarrow E_a \cup \{(v,v) \mid v \in V_a\}$
`make-symmetric`	$E_a \leftarrow E_a \cup \{(v_2,v_1) \mid v_1,v_2 \in V_a \text{ and } (v_1,v_2) \in E_a\}$
`make-transitive`	$E_a \leftarrow E_a^*$
`make-distance` r_a	$E_a \leftarrow E_a \cup \{(v_1,v_2) \mid v_1,v_2 \in V_a \text{ and } \lvert v_1 - v_2\rvert \leq r_a\}$
`make-complement`	$E_a \leftarrow V_a^2 - E_a$
v_1 `->` v_2	$E_a \leftarrow E_a \cup \{(v_1,v_2)\}$
v_1 `--` v_2	$E_a \leftarrow E_a \cup \{(v_1,v_2),(v_2,v_1)\}$

Table 5: A ROSETTA IDG specification for an attribute a consists of a sequence of edge specifications. This table gives an overview of how various edge specification commands alter E_a. Before executing $\langle$edge-specification$_1\rangle$, the graph has no edges, i.e., $E_a = \emptyset$. Here, E_a^* denotes the transitive closure of E_a, computed by Warshall's algorithm [3, 11]. Note that Warshall's algorithm has a time complexity of $O(|V_a|^3)$.

TIP When specifying individual edges, the wildcard symbol '`*`' can be used in place of any of $\{v_1, v_2\}$ as a shorthand notation for specifying all $v \in V_a$.

Example: The following definition describes an IDG for the attribute "Type". The symbol "`->`" can be read as "is a". Note the use and placement of the various commands.

```
begin Type
  nodes Vehicle Bike Car Ford
  Bike -> Vehicle
  Car -> Vehicle
  Ford -> Car
  make-reflexive
  make-symmetric
  make-transitive
end Type
```

Example: The IDG below states that the domain of the attribute "Ethnicity" is reflexive, and that a missing value matches everything (and vice versa). Note that the edges are undirected, thus making a call to `make-symmetric` superfluous.

```
begin Ethnicity
  nodes * Undefined
  make-reflexive
  Undefined -- *
end Ethnicity
```

Example: The IDG below is for an attribute "ca" with the value set $\{0,1,2,3\}$. Thus, this IDG effectively amounts to the same as discretizing the attribute to the "new" value set $\{\{0\},\{1,2,3\}\}$, in addition to letting missing values match everything (and vice versa).

```
begin ca
  nodes * Undefined
  make-reflexive
  Undefined -- *
  1 -- 2
  1 -- 3
  2 -- 3
end ca
```

Example: The IDG below states that all values for the attribute "disease" are indiscernible. Thus, this IDG amounts to the same as masking away or ignoring the attribute.

```
begin disease
  nodes * Undefined
  make-complement
end disease
```

A.6 Attribute Cost Information

Description: Several algorithms in ROSETTA can make use of attribute cost information. A cost file is a list of individual attribute costs, with data on one attribute per line. Attribute names and their costs are separated by the '=' character. Costs are allowed to take on both positive and negative values.

Currently, information about costs shared among attributes is not handled. Such information is relevant if the cost of an attribute depends on the context it is evaluated in.

TIP Blank lines and lines that start with the character '%' are ignored.

Example: The following file defines a valid cost file:

```
color   = 100
height  = 15.7
is_open = 5.1
```

A.7 RSES Distance Matrices

Description: The RSES classifier described in Section 14.4 makes use of a distance function f. This function is supplied in an ASCII file, whose format is documented by the example below.

Example: The file containing the distance function f from the example in Section 14.4 is shown below. The second line specifies the number of decision values, while the third line specifies the decision values themselves. The distance function f is defined by the matrix in the last three lines.

```
Information about decision values
3
1 2 3
;
0 1 3
1 0 2
3 2 0
```

Note that the first and fourth line are for all practical purposes ignored.

A.8 Pairs Files

Description: The HYPOCLASS utility and its command-line counterpart operate on "pairs files". Each line of data in the ASCII file describes a classifier's output for a single case or object x. Each line of data has the following format:[10]

$$\langle d(x)\rangle\ \langle \phi(x)\rangle\ [\mathsf{key}[\ldots]]$$

The fields are interpreted as follows:

- $\langle d(x)\rangle$ (*Integer* $\in \{0,1\}$)
  Denotes the actual outcome for object x.
- $\langle \phi(x)\rangle$ (*Float* $\in [0,1]$)
  Denotes the classifier's output when applied to object x. The value indicates the classifier's degree of certainty that object x has outcome 1.
- [key] (*Integer*)
  Typically denotes the index of object x. This field enables the data lines to be sorted so that the data from two classifiers can be automatically "aligned", if compared. If this field is missing, no sorting takes place and it is the user's responsibility that the data lines are correctly ordered.

[10] Arguments in angled brackets are required. Arguments in squared brackets are optional.

TIP Blank lines and lines that start with the character '%' are ignored.

TIP Note that a "pairs file" as described above can be produced by ROSETTA. Such a file is simply a calibration curve plot file with a single object per group, i.e., with the number of groups maximized. See Section 4.8.

B Command-Line Versions

Description: Both ROSETTA and the accompanying HYPOCLASS utility come in command-line versions, too, called CLROSETTA and CLHYPOCLASS. These versions are supplied so that they can be used as computational engines called from elsewhere, e.g., from some kind of script.

TIP An example of using Perl to programmatically prepare inputs and parse outputs can be found on the ROSETTA website [35].

It is possible to port CLROSETTA and CLHYPOCLASS to other platforms than Windows.

B.1 CLROSETTA

Description: CLROSETTA offers all the same core functionality as its GUI counterpart, and is invoked as shown below.

```
clrosetta ⟨algorithm⟩ ⟨parameters⟩ [filename]
```

The arguments are interpreted as follows:

- ⟨algorithm⟩ (*Id*)
  The algorithm that is to be executed. Most often, this will be either *SerialExecutor*, *CVSerialExecutor* or *ParallelExecutor*.
- ⟨parameters⟩ (*String*)
  The list of parameters that is to be passed to the above specified algorithm, given as a single argument.

  TIP Quote the parameter list to make the shell interpret it as a single argument.
- [filename] (*String*)
  The location of the input structure, if any, to the above specified algorithm. If not given, an empty *Project* structure is passed to the algorithm instead. The file is assumed to be in internal ROSETTA format.

  TIP If the input file is not in internal ROSETTA format, then don't supply the last argument. Use a *StructureCreator* and an import routine in the command script instead.

Example: Assume that we want to apply 10-fold CV to a decision table residing in the file 'iris.ros'. Let the command script reside in the file 'cmds.txt', and assume that the training pipeline contains five steps. If we want to save the log file as 'log.txt', we can enter the following at the command prompt:

```
clrosetta CVSerialExecutor "NUMBER = 10; FILENAME.COMMANDS = cmds.txt;
                            LENGTH = 5; FILENAME.LOG = log.txt" iris.ros
```

Note that the parameter list is quoted. (The text above is here, for presentational purposes, split across two lines. In reality, of course, it would have been entered at the command prompt as one line.)

B.2 CLHYPOCLASS

Description: CLHYPOCLASS implements Hanley-McNeil's test [18] and McNemar's test [34, 4], and is invoked as shown below.

$\texttt{clhypoclass}\ \langle\text{test}\rangle\ \langle\text{parameter}_1\rangle\ \langle\text{parameter}_2\rangle\ \langle\text{filename}_1\rangle\ \langle\text{filename}_2\rangle\ [\langle\text{swapped}_1\rangle\ \langle\text{swapped}_2\rangle]$

The arguments are interpreted as follows:

- $\langle\text{test}\rangle$ ({HanleyMcNeil, McNemar})
  Identifies the statistical test to perform.
- For Hanley-McNeil's test:
  1. $\langle\text{parameter}_1\rangle$ ({Pearson, Kendall})
     Specifies how correlations should be computed.
  2. $\langle\text{parameter}_2\rangle$ ({CIndex, Trapezoidal})
     Specifies how the area under the ROC curve should be computed.
- For McNemar's test:
  1. $\langle\text{parameter}_i\rangle$ (*Float*)
     Specifies the threshold τ to impose on outputs from classifier i.
- $\langle\text{filename}_i\rangle$ (*String*)
  Gives the location of a "pairs file", i.e., the output of classifier i. The format of a "pairs file" is described in Appendix A.8.

  TIP Make sure the two files hail from classifiers applied to the same set of objects.
- $[\langle\text{swapped}_i\rangle]$ (*Boolean*)
  Boolean flag set to true if pairs file i contains $(\phi(x), d(x))$ pairs instead of $(d(x), \phi(x))$ pairs. Assumed false if not given.

Example: Let 'pairs1.txt' and 'pairs1.txt' denote two pairs files, as specified in Appendix A.8. The two files stem from classifiers applied to the same set of cases. To perform Hanley-McNeil's test using "ordinary" Pearson correlation and trapezoidal integration, we can enter the following at the command prompt:

```
clhypoclass HanleyMcNeil Pearson Trapezoidal pairs1.txt pairs2.txt
```

References

[1] T. Ågotnes. Filtering large propositional rule sets while retaining classifier performance. MSc thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, Feb. 1999.

[2] T. Ågotnes, J. Komorowski, and A. Øhrn. Finding high performance subsets of induced rule sets: Extended summary. In H.-J. Zimmermann and K. Lieven, editors, *Proc. Seventh European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, Sept. 1999.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[4] D. G. Altman. *Practical Statistics for Medical Research*. Chapman & Hall, London, UK, 1991.

[5] H. R. Arkes, N. W. Dawson, T. Speroff, F. E. Harrel, Jr., C. Alzola, R. Phillips, N. Desbiens, R. K. Oye, W. Knaus, and A. F. Connors, Jr. The covariance decomposition of the probability score and its use in evaluating prognostic estimates. *Medical Decision Making*, 15:120–131, 1995.

[6] J. G. Bazan. A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables. In Polkowski and Skowron [32], chapter 17, pages 321–365.

[7] J. G. Bazan, A. Skowron, and P. Synak. Dynamic reducts as a tool for extracting laws from decision tables. In *Proc. International Symposium on Methodologies for Intelligent Systems*, volume 869 of *Lecture Notes in Artificial Intelligence*, pages 346–355. Springer-Verlag, 1994.

[8] D. A. Bloch. Evaluating predictions of events with binary outcomes: An appraisal of the Brier score and some of its close relatives. Technical Report 135, Stanford University, Division of Biostatistics, Stanford University, CA, May 1990.

[9] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[10] I. Bruha. Quality of decision rules: Definitions and classification schemes for multiple rules. In G. Nakhaeizadeh and C. C. Taylor, editors, *Machine Learning and Statistics: The Interface*, chapter 5, pages 107–131. John Wiley & Sons, 1997.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[12] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proc. Thirteenth International Conference on Machine Learning*, pages 105–112, Bari, Italy, 1996. Morgan Kaufmann.

[13] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proc. Twelfth International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, 1995.

[14] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[15] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed replacement. *Software – Practice and Experience*, 21(11):1129–1164, Nov. 1991.

[16] The GraphViz homepage. [`http://www.research.att.com/sw/tools/graphviz/`]. AT&T Research.

[17] The Group of Logic homepage. [`http://alfa.mimuw.edu.pl/logic/`]. University of Warsaw, Poland.

[18] J. A. Hanley and B. J. McNeil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148:839–843, Sept. 1983.

[19] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, Apr. 1993.

[20] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[21] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, Apr. 1989.

[22] The MATLAB homepage. [`http://www.mathworks.com/products/matlab/`]. The MathWorks, Inc.

[23] H. S. Nguyen and S. H. Nguyen. Some efficient algorithms for rough set methods. In *Proc. Fifth Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96)*, pages 1451–1456, Granada, Spain, July 1996.

[24] H. S. Nguyen and A. Skowron. Quantization of real-valued attributes. In *Proc. Second International Joint Conference on Information Sciences*, pages 34–37, Wrightsville Beach, NC, Sept. 1995.

[25] A. Øhrn. Cracking a logical puzzle with ROSETTA. Technical report, Knowledge Systems Group, Department of Computer and Information Science, NTNU, Trondheim, Norway, Dec. 1999.

[26] A. Øhrn. *Discernibility and Rough Sets in Medicine: Tools and Applications*. PhD thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, Dec. 1999. NTNU report 1999:133. [`http://www.idi.ntnu.no/~aleks/thesis/`].

[27] A. Øhrn and J. Komorowski. ROSETTA: A rough set toolkit for analysis of data. In P. P. Wang, editor, *Proc. Third International Joint Conference on Information Sciences*, volume 3, pages 403–407, Durham, NC, Mar. 1997.

[28] A. Øhrn, J. Komorowski, A. Skowron, and P. Synak. The design and implementation of a knowledge discovery toolkit based on rough sets: The ROSETTA system. In Polkowski and Skowron [32], chapter 19, pages 376–399.

[29] A. Øhrn, J. Komorowski, A. Skowron, and P. Synak. The ROSETTA software system. In L. Polkowski and A. Skowron, editors, *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, volume 19 of *Studies in Fuzziness and Soft Computing*, pages 572–576. Physica-Verlag, Heidelberg, Germany, 1998.

[30] A. Øhrn, L. Ohno-Machado, and T. Rowland. Building manageable rough set classifiers. In C. G. Chute, editor, *Proceedings AMIA 1998 Annual Symposium*, pages 543–547, Orlando, FL, Nov. 1998. Supplement to *Journal of the American Medical Informatics Association*, Hanley & Belfus, Inc.

[31] Z. Pawlak. *Rough Sets: Theoretical Aspects of Reasoning about Data*, volume 9 of *Series D: System Theory, Knowledge Engineering and Problem Solving*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[32] L. Polkowski and A. Skowron, editors. *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, volume 18 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, Heidelberg, Germany, 1998.

[33] D. A. Redelmeier, D. A. Bloch, and D. H. Hickam. Assessing predictive accuracy: How to compare Brier scores. *Journal of Clinical Epidemiology*, 44(11):1141–1146, 1991.

[34] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[35] The ROSETTA homepage. [`http://www.idi.ntnu.no/~aleks/rosetta/`]. Norwegian University of Science and Technology, Department of Computer and Information Science.

[36] The ROSETTA C++ library homepage. [`http://www.idi.ntnu.no/~aleks/thesis/source/`]. Norwegian University of Science and Technology, Department of Computer and Information Science.

[37] A. Skowron. Synthesis of adaptive decision systems from experimental data. In A. Aamodt and J. Komorowski, editors, *Proc. Fifth Scandinavian Conference on Artificial Intelligence*, number 28 in Frontiers in Artificial Intelligence and Applications, pages 220–238. IOS Press, May 1995.

[38] P. Synak. *Rough Set Expert System User's Guide*. Institute of Mathematics, Warsaw University, Poland, 1995. Version 1.0.

[39] S. Vinterbo, L. Ohno-Machado, and H. Fraser. A description of a strategy for building rough set classifiers using performance filtering of reducts. In H.-J. Zimmermann and K. Lieven, editors, *Proc. Sixth European Congress on Intelligent Techniques and Soft Computing (EUFIT'98)*, volume 2, pages 975–979, Aachen, Germany, Sept. 1998.

[40] S. Vinterbo and A. Øhrn. Minimal approximate hitting sets and rule templates. In *Predictive Models in Medicine: Some Methods for Construction and Adaptation*. Department of Computer and Information Science, Dec. 1999. NTNU report 1999:130. [`http://www.idi.ntnu.no/~staalv/dev/thesis.ps.gz`].

[41] S. Vinterbo and A. Øhrn. Minimal approximate hitting sets and rule templates. *International Journal of Approximate Reasoning*, 25(2):123–143, 2000.

[42] J. Wróblewski. Finding minimal reducts using genetic algorithms. In *Proc. Second International Joint Conference on Information Sciences*, pages 186–189, Sept. 1995.

[43] W. Ziarko. Analysis of uncertain information in the framework of variable precision rough sets. *Foundations of Computing and Decision Sciences*, 18(3–4):381–396, 1993.

[44] W. Ziarko. Variable precision rough set model. *Journal of Computer and System Sciences*, 46:39–59, 1993.

Index