Orange Visual Programming Documentation

Release 3

Orange Data Mining

Sep 27, 2017

Contents

1 Getting Started 1

2 Widgets 13

CHAPTER 1

Getting Started

Here we need to copy the getting started guide.

Loading your Data

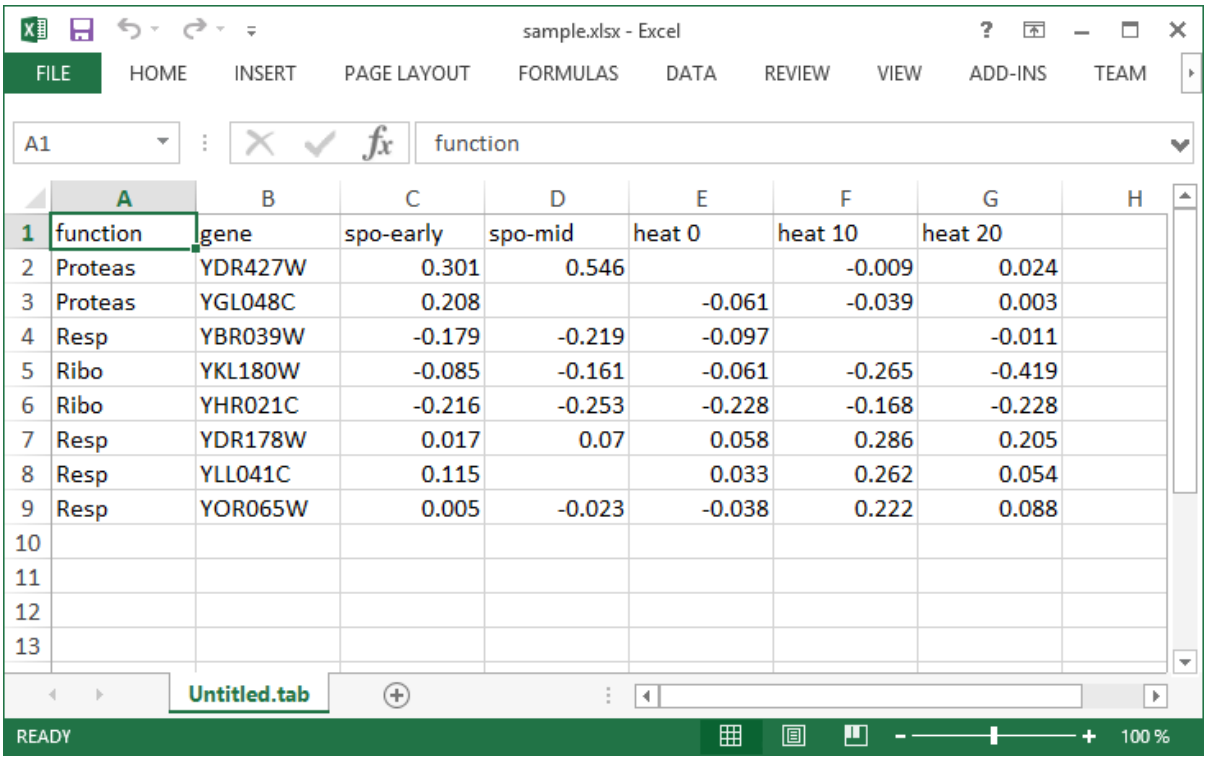
Orange comes with its own data format, but can also handle native Excel (.xlsx or .xls), comma- or tab-delimited data files. The input data set is usually a table, with data instances (samples) in rows and data attributes in columns. Attributes can be of different *types* (continuous, discrete, time, and strings) and have assigned *roles* (input features, meta attributes, and class). Data attribute type and role can be provided in the data table header. They can also be subsequently changed in the *File* widget, while data role can also be modified with *Select Columns* widget.

In a Nutshell

- Orange can import any comma- or tab-delimited data file, or Excel's native files or Google Sheets document. Use *File* widget to load the data and, if needed, define the class and meta attributes.
- Attribute names in the column header can be preceded with a label followed by a hash. Use c for class and m for meta attribute, i to ignore a column, w for weights column, and C, D, T, S for continuous, discrete, time, and string attribute types. Examples: C#mph, mS#name, i#dummy.
- An alternative to the hash notation is Orange's native format with three header rows: the first with attribute names, the second specifying the type (**continuous**, **discrete**, **time**, or **string**), and the third proving information on the attribute role (**class**, **meta**, **weight** or **ignore**).

Example: Data from Excel

Here is an example data set (download it from `sample.xlsx`) as entered in Excel:

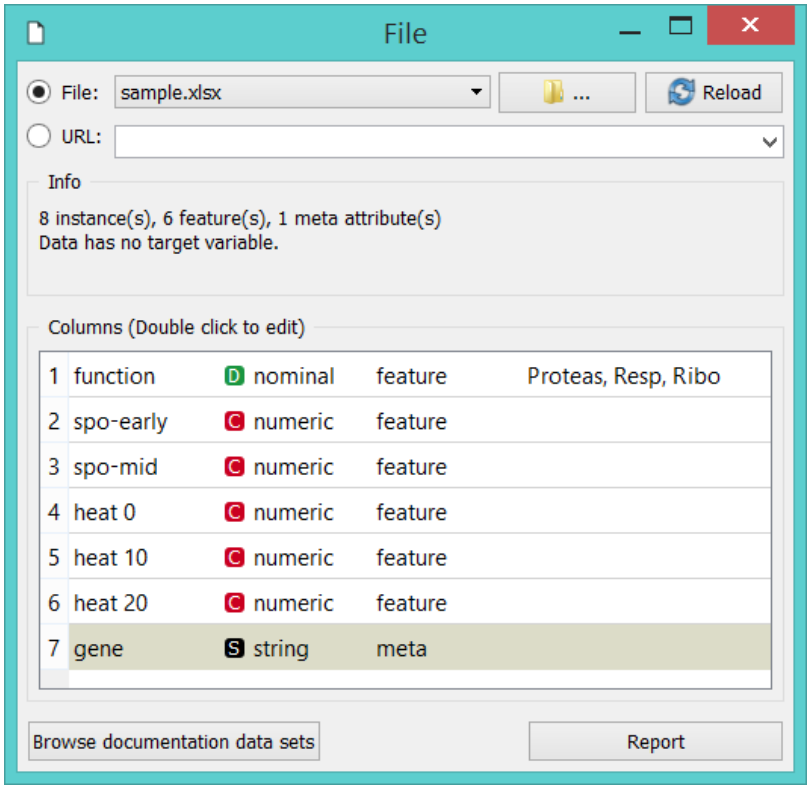


	A	B	C	D	E	F	G	H
1	function	gene	spo-early	spo-mid	heat 0	heat 10	heat 20	
2	Proteas	YDR427W	0.301	0.546		-0.009	0.024	
3	Proteas	YGL048C	0.208		-0.061	-0.039	0.003	
4	Resp	YBR039W	-0.179	-0.219	-0.097		-0.011	
5	Ribo	YKL180W	-0.085	-0.161	-0.061	-0.265	-0.419	
6	Ribo	YHR021C	-0.216	-0.253	-0.228	-0.168	-0.228	
7	Resp	YDR178W	0.017	0.07	0.058	0.286	0.205	
8	Resp	YLL041C	0.115		0.033	0.262	0.054	
9	Resp	YOR065W	0.005	-0.023	-0.038	0.222	0.088	
10								
11								
12								
13								

The file contains a header row, eight data instances (rows) and seven data attributes (columns). Empty cells in the table denote missing data entries. Rows represent genes; their function (class) is provided in the first column and their name in the second. The remaining columns store measurements that characterize each gene. With this data, we could, say, develop a classifier that would predict gene function from its characteristic measurements.

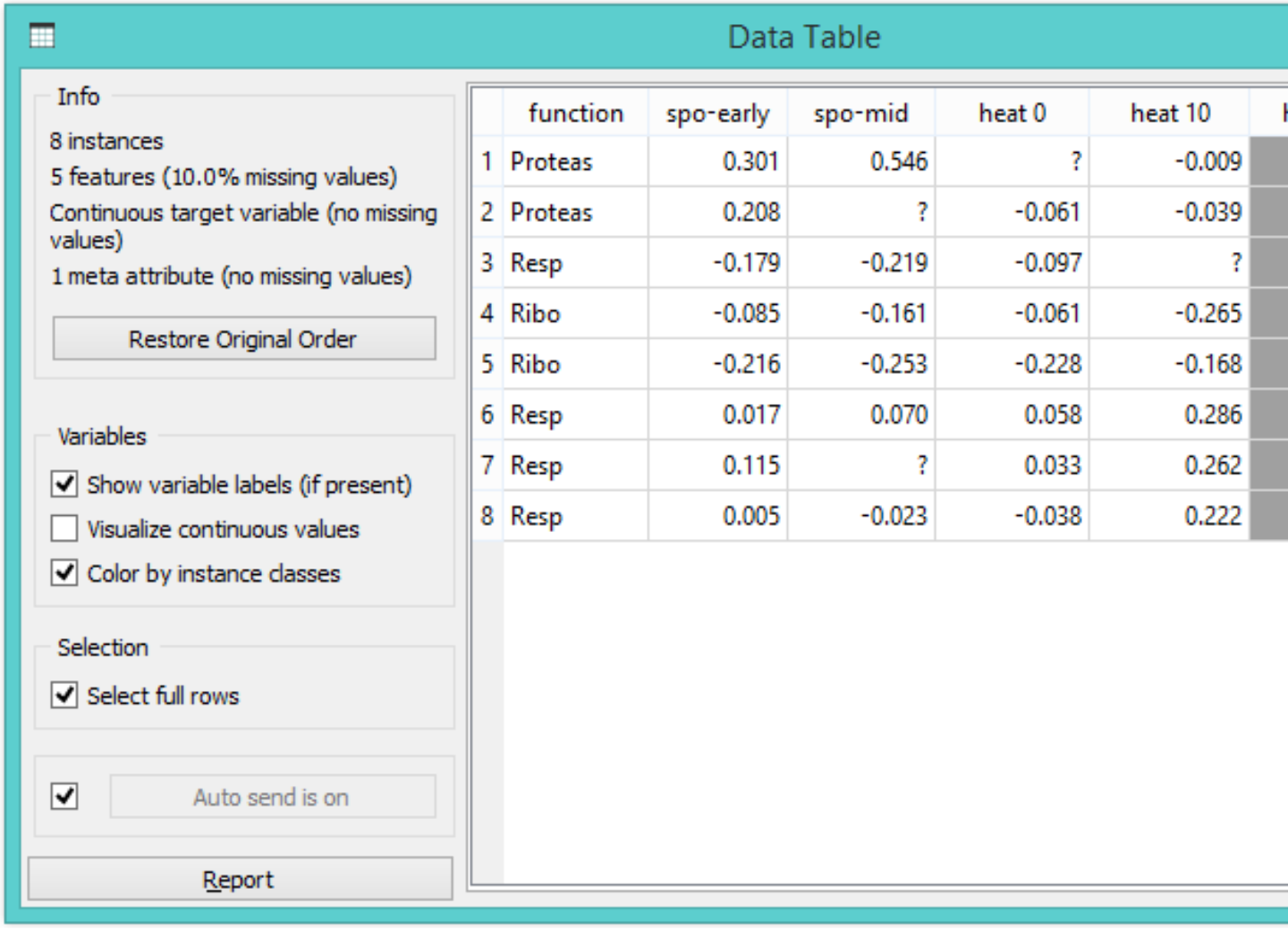
Let us start with a simple workflow that reads the data and displays it in a table:

To load the data, open the File widget (double click on the icon of the widget), click on the file browser icon ("...") and locate the downloaded file (from `sample.xlsx`) on your disk:

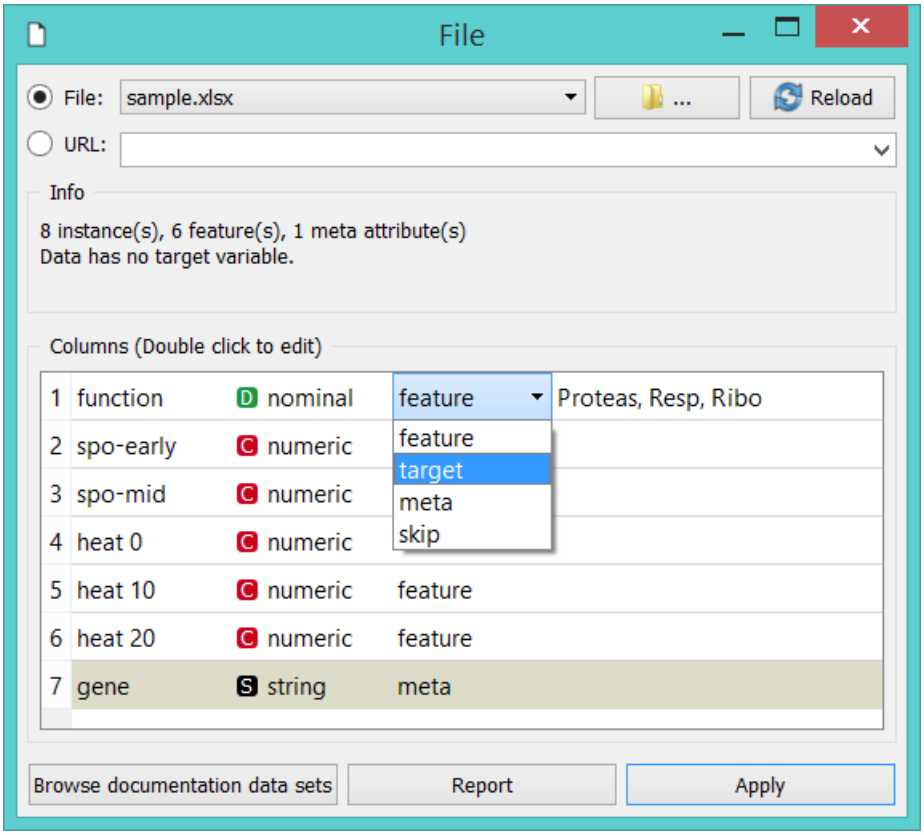


File Widget: Setting the Attribute Type and Role

The **File** widget sends the data to the **Data Table**. Double click the **Data Table** to see its contents:



Orange correctly assumed that a column with gene names is meta information, which is displayed in the **Data Table** in columns shaded with light-brown. It has not guessed that *function*, the first non-meta column in our data file, is a class column. To correct this in Orange, we can adjust attribute role in the column display of File widget (below). Double-click the *feature* label in the *function* row and select *target* instead. This will set *function* attribute as our target (class) variable.



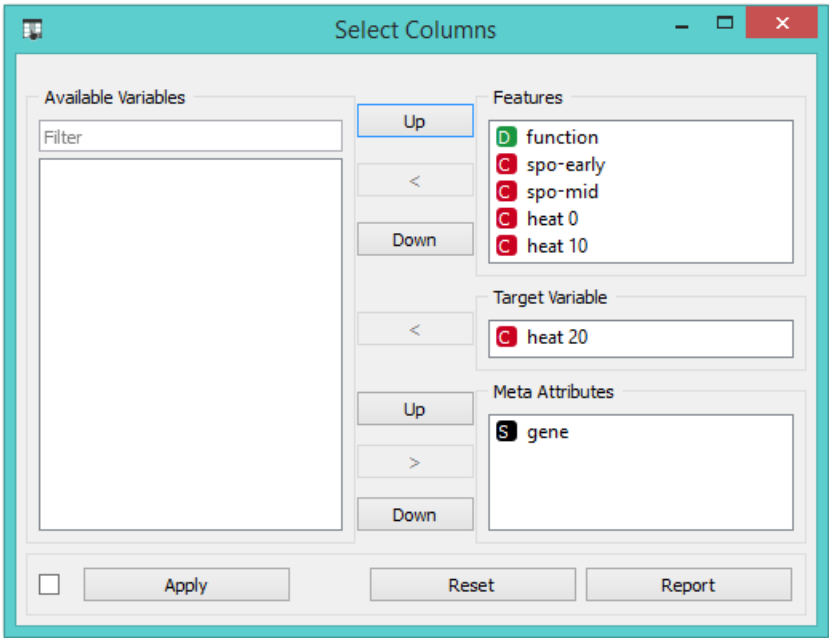
You can also change attribute type from nominal to numeric, from string to datetime, and so on. Naturally, data values have to suit the specified attribute type. Datetime accepts only values in ISO 8601 format, e.g. 2016-01-01 16:16:01. Orange would also assume the attribute is numeric if it has several different values, else it would be considered nominal. All other types are considered strings and are as such automatically categorized as meta attributes.

Change of attribute roles and types should be confirmed by clicking the **Apply** button.

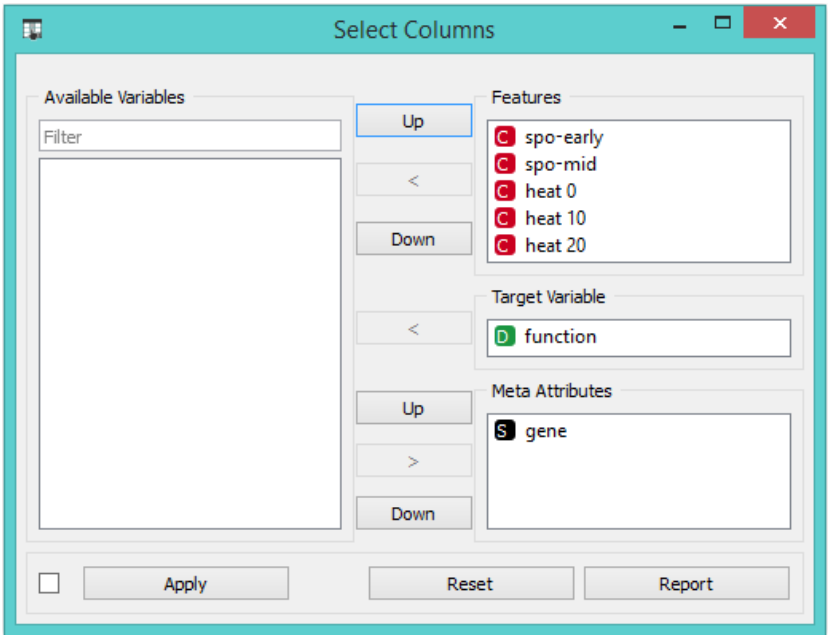
Select Columns: Setting the Attribute Role

Another way to set the data role is to feed the data to the *Select Columns* widget:

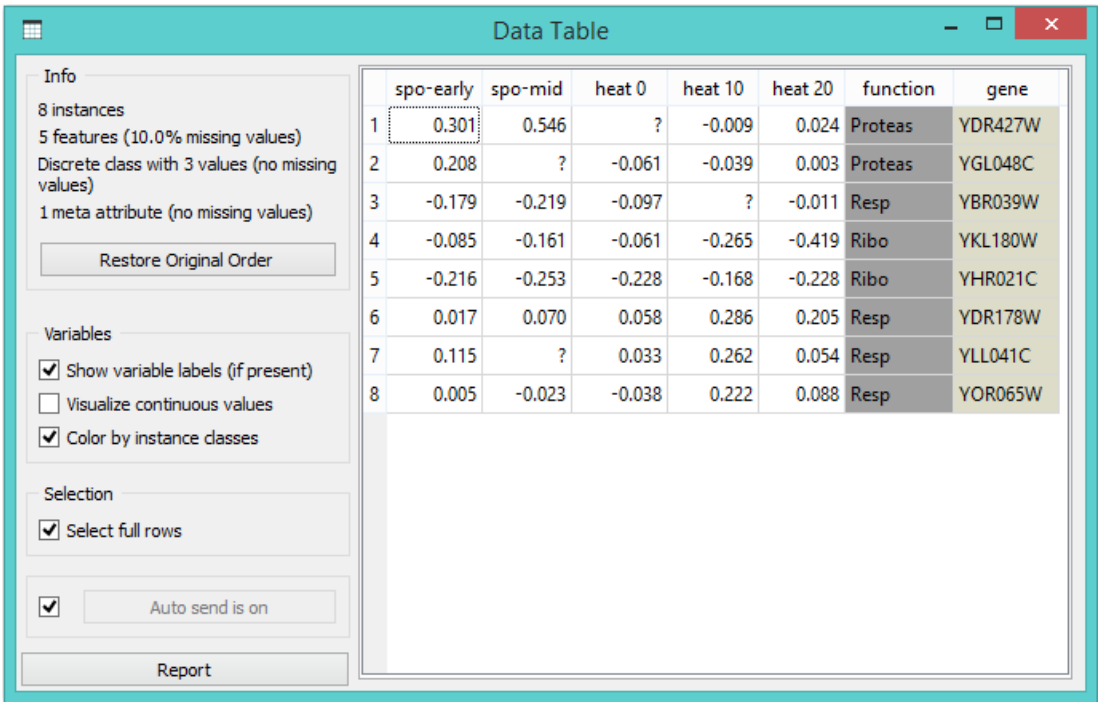
Opening *Select Columns* reveals Orange's classification of attributes. We would like all of our continuous attributes to be data features, gene function to be our target variable and gene names considered as meta attributes. We can obtain this by dragging the attribute names around the boxes in **Select Columns**:



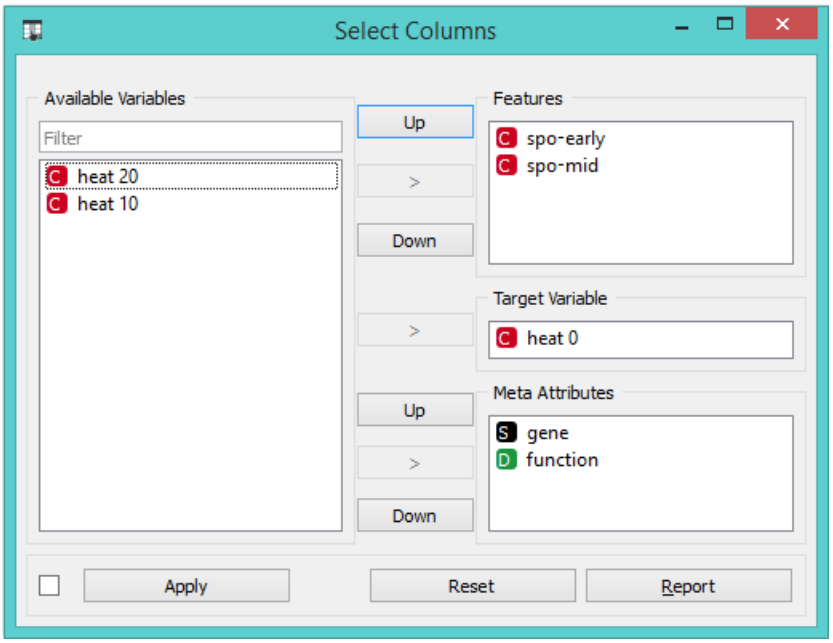
To correctly reassign attribute types, drag attribute named *function* to a **Class** box, and attribute named *gene* to a **Meta Attribute** box. The *Select Columns* widget should now look like this:



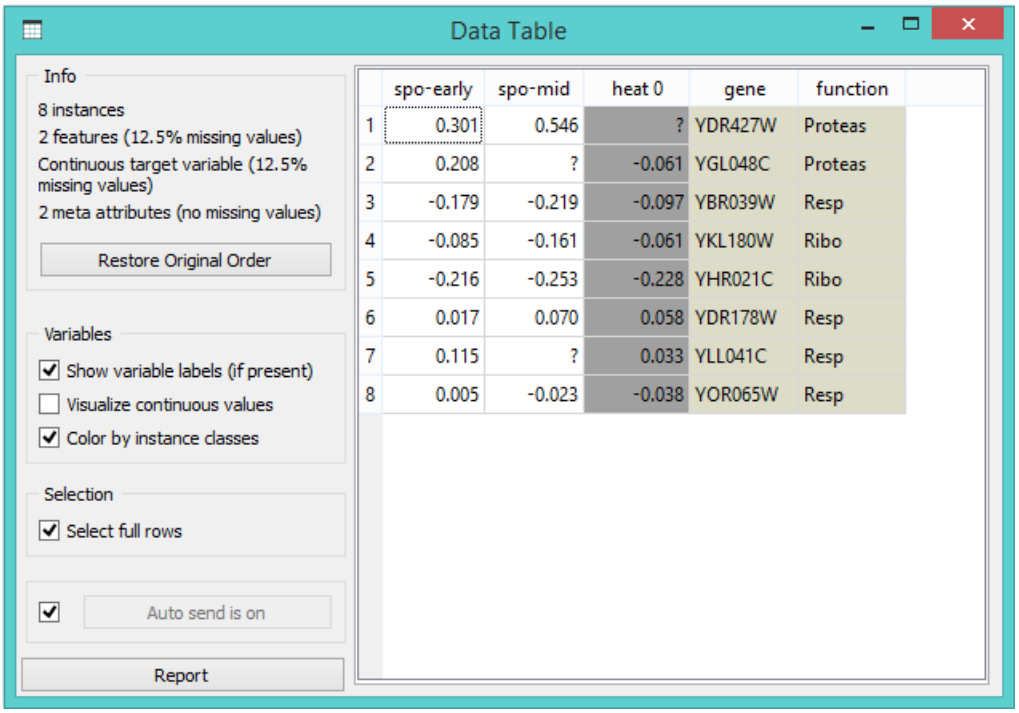
Change of attribute types in *Select Columns* widget should be confirmed by clicking the **Apply** button. The data from this widget is fed into *Data Table* that now renders the data just the way we intended:



We could also define the domain for this data set in a different way. Say, we could make the data set ready for regression, and use *heat 0* as a continuous class variable, keep gene function and name as meta variables, and remove *heat 10* and *heat 20* from the data set:



By setting the attributes as above, the rendering of the data in the Data Table widget gives the following output:



Header with Attribute Type Information

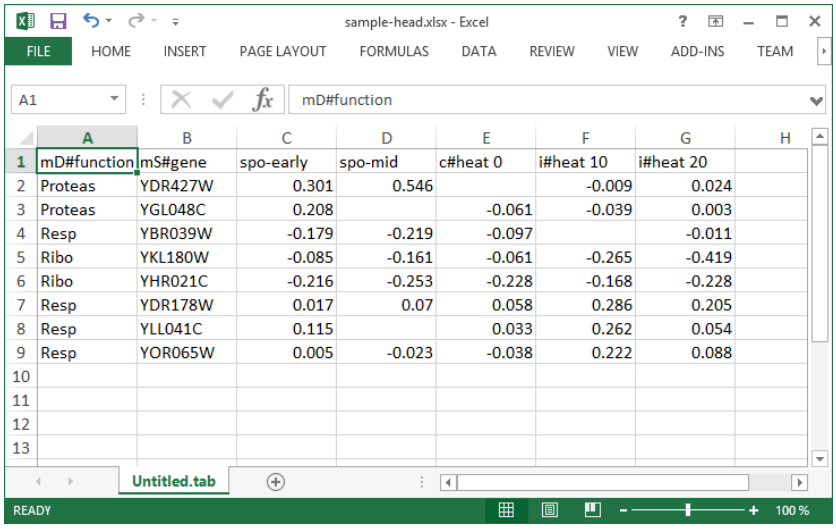
Consider again the `sample.xlsx` data set. This time we will augment the names of the attributes with prefixes that define attribute type (continuous, discrete, time, string) and role (class or meta attribute) Prefixes are separated from the attribute name with a hash sign ("#"). Prefixes for attribute roles are:

- c: class attribute
- m: meta attribute
- i: ignore the attribute
- w: instance weights

and for the type:

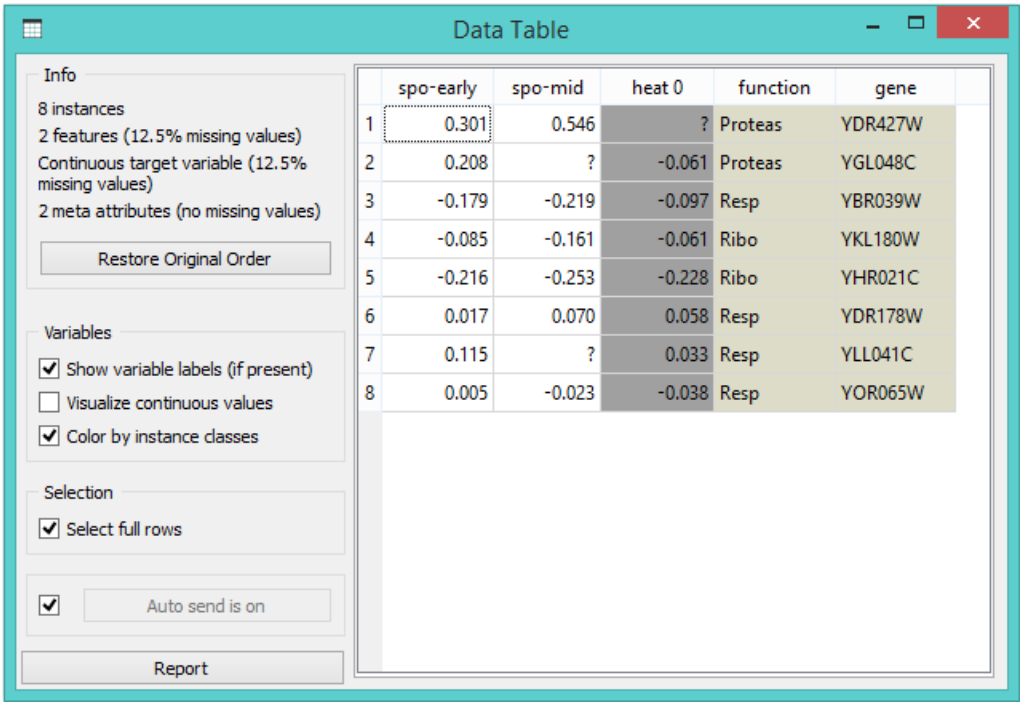
- C: Continuous
- D: Discrete
- T: Time
- S: String

This is how the header with augmented attribute names looks like in Excel (`sample-head.xlsx`):



	A	B	C	D	E	F	G	H
1	mD#function	mS#gene	spo-early	spo-mid	c#heat 0	i#heat 10	i#heat 20	
2	Proteas	YDR427W	0.301	0.546		-0.009	0.024	
3	Proteas	YGL048C	0.208		-0.061	-0.039	0.003	
4	Resp	YBR039W	-0.179	-0.219	-0.097		-0.011	
5	Ribo	YKL180W	-0.085	-0.161	-0.061	-0.265	-0.419	
6	Ribo	YHR021C	-0.216	-0.253	-0.228	-0.168	-0.228	
7	Resp	YDR178W	0.017	0.07	0.058	0.286	0.205	
8	Resp	YLL041C	0.115		0.033	0.262	0.054	
9	Resp	YOR065W	0.005	-0.023	-0.038	0.222	0.088	
10								
11								
12								
13								

We can again use a **File** widget to load this data set and then render it in the **Data Table**:

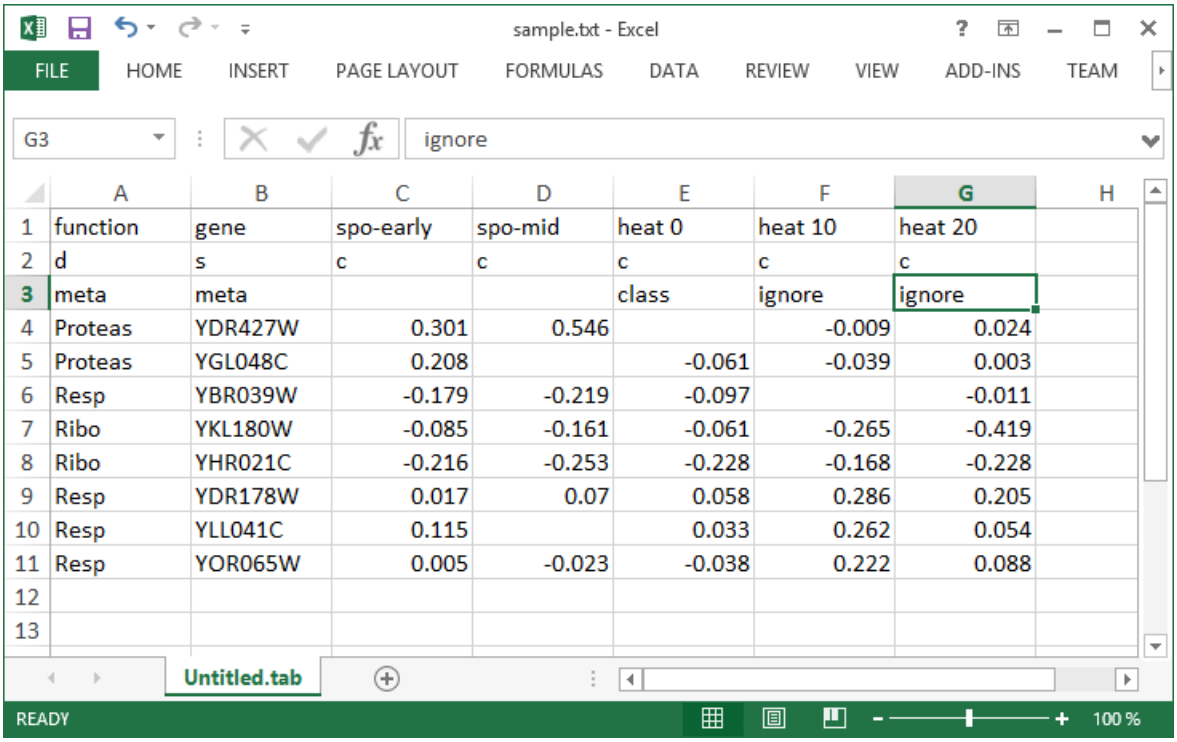


	spo-early	spo-mid	heat 0	function	gene
1	0.301	0.546	?	Proteas	YDR427W
2	0.208	?	-0.061	Proteas	YGL048C
3	-0.179	-0.219	-0.097	Resp	YBR039W
4	-0.085	-0.161	-0.061	Ribo	YKL180W
5	-0.216	-0.253	-0.228	Ribo	YHR021C
6	0.017	0.070	0.058	Resp	YDR178W
7	0.115	?	0.033	Resp	YLL041C
8	0.005	-0.023	-0.038	Resp	YOR065W

Notice that the attributes we have ignored (label "i" in the attribute name) are not present in the data set.

Three-Row Header Format

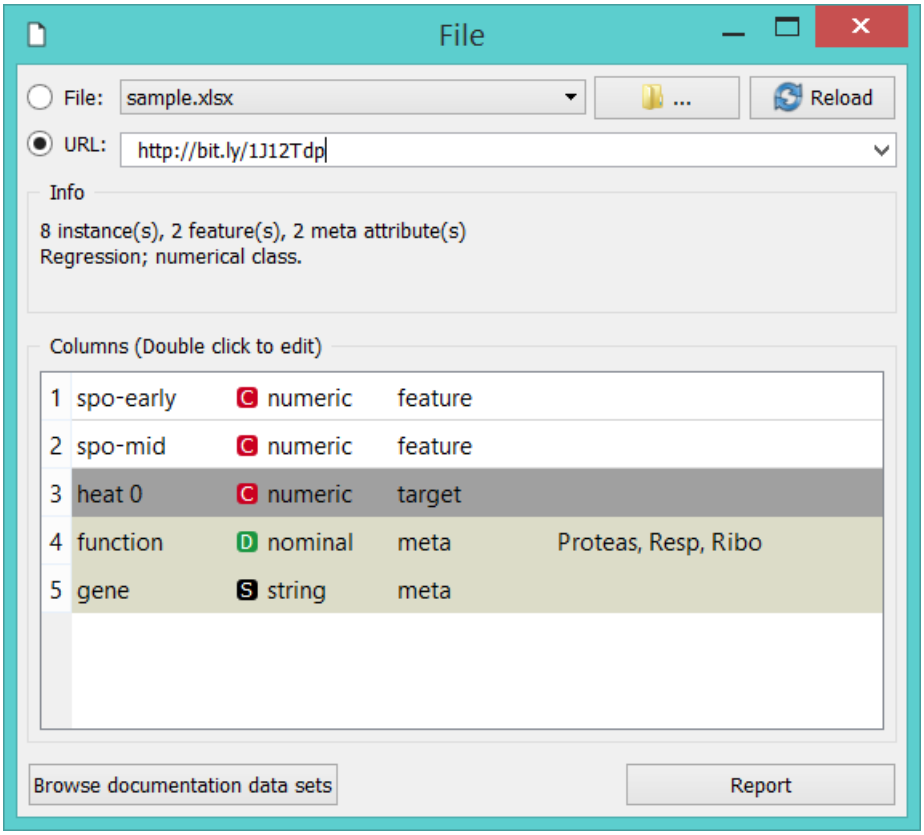
Orange's legacy native data format is a tab-delimited text file with three header rows. The first row lists the attribute names, the second row defines their type (continuous, discrete, time and string, or abbreviated c, d, t, and s), and the third row an optional role (class, meta, weight, or ignore). Here is an example:



	A	B	C	D	E	F	G	H
1	function	gene	spo-early	spo-mid	heat 0	heat 10	heat 20	
2	d	s	c	c	c	c	c	
3	meta	meta			class	ignore	ignore	
4	Proteas	YDR427W	0.301	0.546		-0.009	0.024	
5	Proteas	YGL048C	0.208		-0.061	-0.039	0.003	
6	Resp	YBR039W	-0.179	-0.219	-0.097		-0.011	
7	Ribo	YKL180W	-0.085	-0.161	-0.061	-0.265	-0.419	
8	Ribo	YHR021C	-0.216	-0.253	-0.228	-0.168	-0.228	
9	Resp	YDR178W	0.017	0.07	0.058	0.286	0.205	
10	Resp	YLL041C	0.115		0.033	0.262	0.054	
11	Resp	YOR065W	0.005	-0.023	-0.038	0.222	0.088	
12								
13								

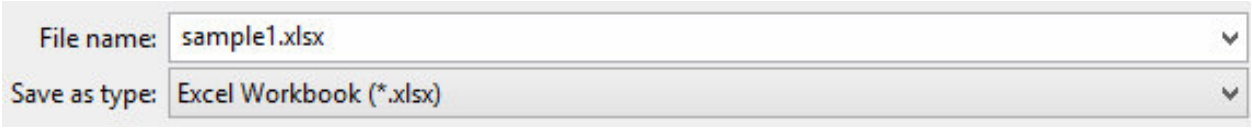
Data from Google Sheets

Orange can read data from Google Sheets, as long as it conforms to the data presentation rules we have presented above. In Google Sheets, copy the shareable link (Share button, then Get shareable link) and paste it in the *Data File / URL* box of the File widget. For a taste, here's one such link you can use: http://bit.ly/1J12Tdp, and the way we have entered it in the **File** widget:



Data from LibreOffice

If you are using LibreOffice, simply save your files in Excel (.xlsx or .xls) format (available from the drop-down menu under *Save As Type*).



Datetime Format

To avoid ambiguity, Orange supports date and/or time formatted in one of ISO 8601 formats. E.g., the following values are all valid:

```
2016
2016-12-27
2016-12-27 14:20:51+02:00
16:20
```

CHAPTER 2

Widgets

Data

File

Reads attribute-value data from an input file.

Signals

Inputs:

- (None)

Outputs:

- **Data**

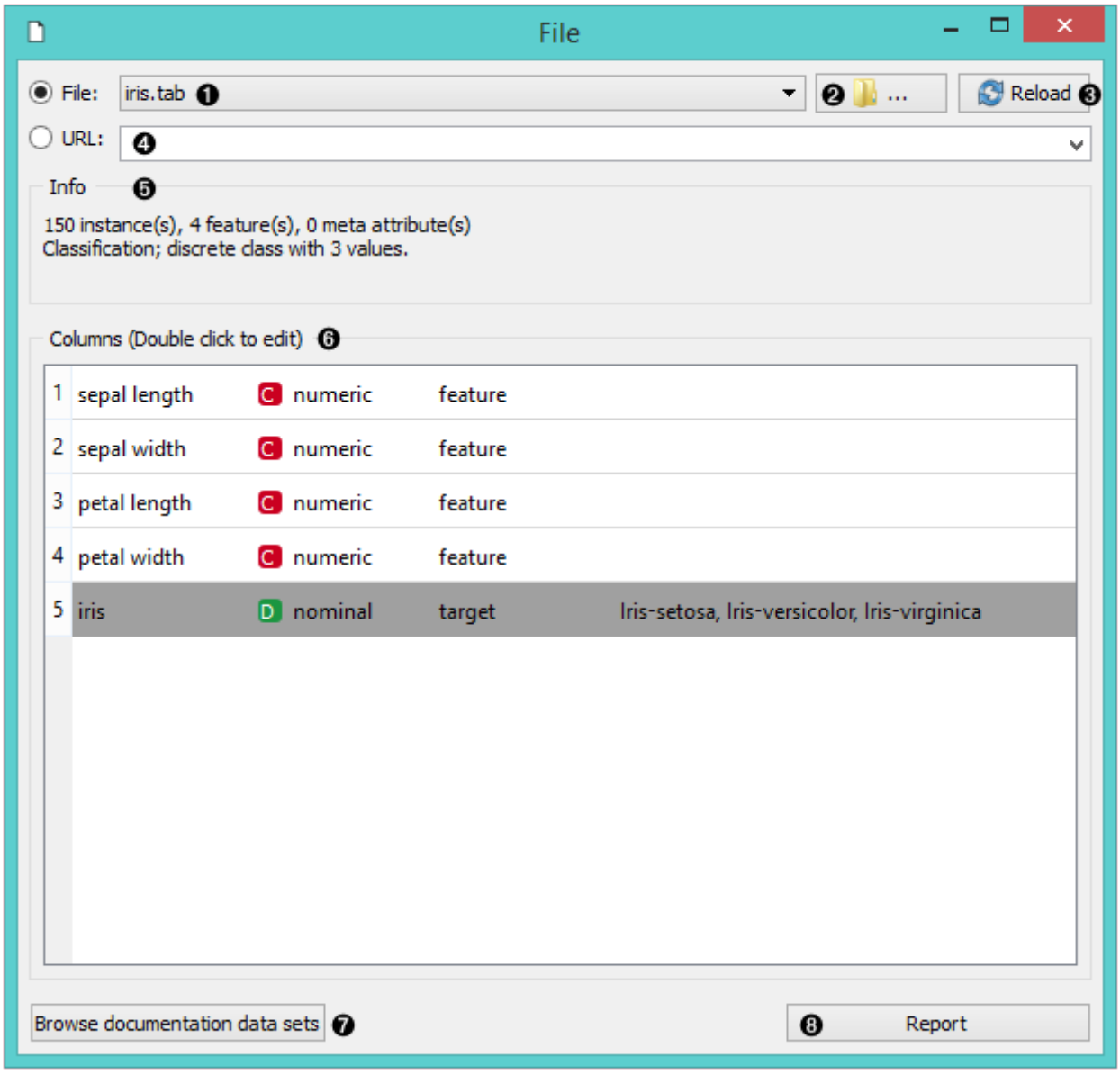
  Attribute-valued data from the input file

Description

The **File** widget *reads the input data file* (data table with data instances) and sends the data set to its output channel. The history of most recently opened files is maintained in the widget. The widget also includes a directory with sample data sets that come pre-installed with Orange.

The widget reads data from Excel (**.xlsx**), simple tab-delimited (**.txt**), comma-separated files (**.csv**) or URLs.

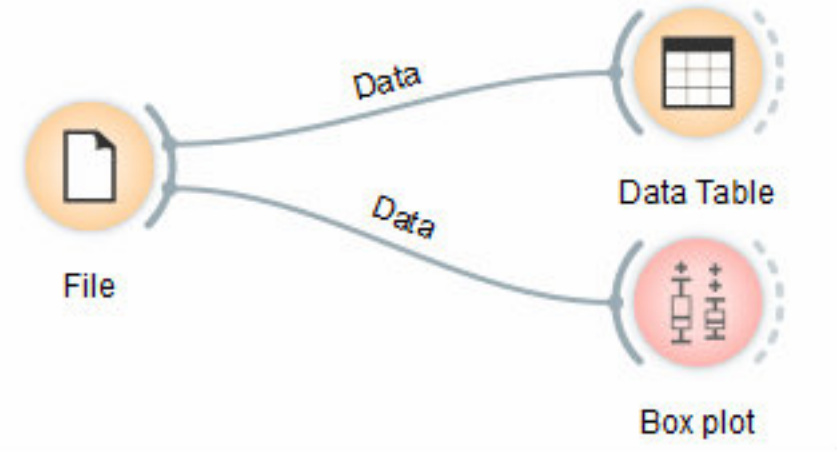
1. Browse through previously opened data files, or load any of the sample ones.
2. Browse for a data file.


File
File:
iris.tab
1
2
...
Reload
3
URL:
4
Info
5
150 instance(s), 4 feature(s), 0 meta attribute(s)
Classification; discrete class with 3 values.
Columns (Double click to edit)
6
1 sepal length C numeric feature
2 sepal width C numeric feature
3 petal length C numeric feature
4 petal width C numeric feature
5 iris D nominal target Iris-setosa, Iris-versicolor, Iris-virginica
Browse documentation data sets
7
8
Report

3. Reloads currently selected data file.
4. Insert data from URL adresses, including data from Google Sheets.
5. Information on the loaded data set: data set size, number and types of data features.
6. Additional information on the features in the data set. Features can be edited by double-clicking on them. The user can change the attribute names, select the type of variable per each attribute (*Continuous*, *Nominal*, *String*, *Datetime*), and choose how to further define the attributes (as *Features*, *Targets* or *Meta*). The user can also decide to ignore an attribute.
7. Browse documentation data sets.
8. Produce a report.

Example

Most Orange workflows would probably start with the **File** widget. In the schema below, the widget is used to read the data that is sent to both the *Data Table* and the *Box Plot* widget.



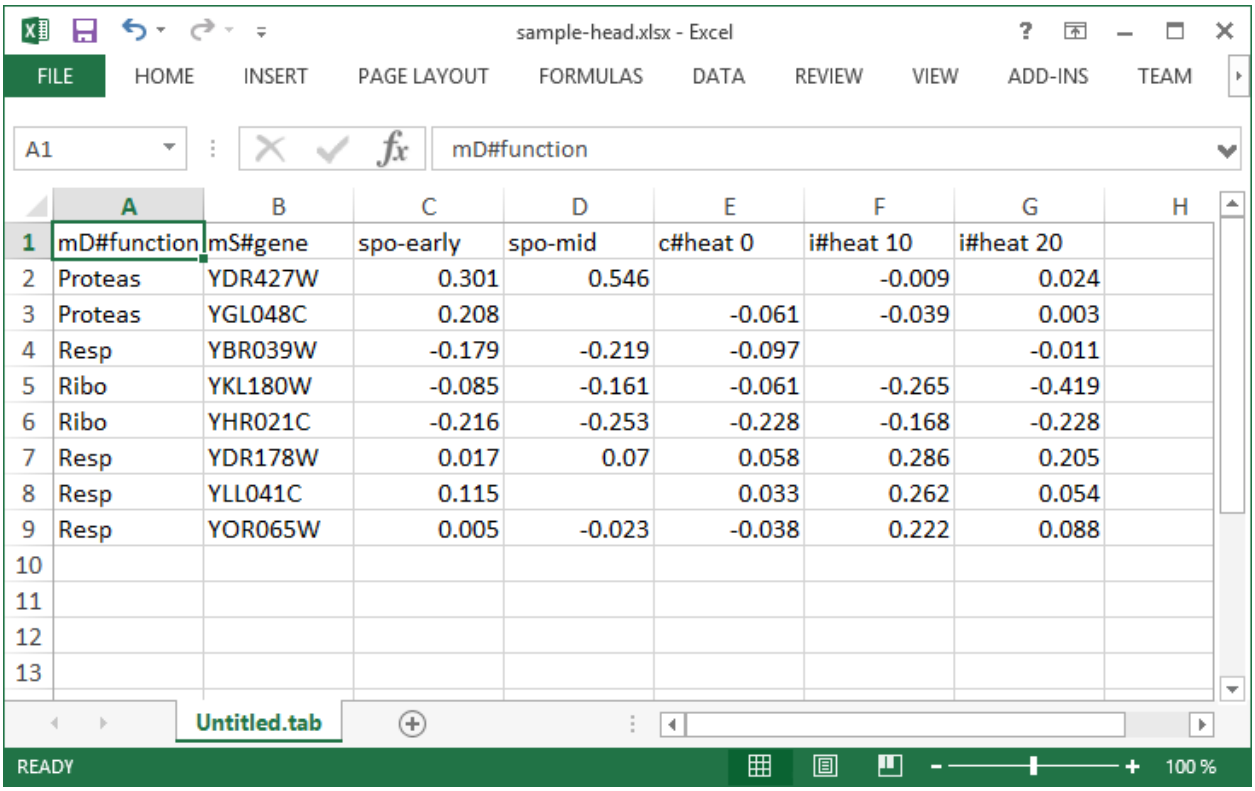
Loading your data

- Orange can import any comma, .xlsx or tab-delimited data file or URL. Use the *File widget* and then, if needed, select class and meta attributes.
- To specify the domain and the type of the attribute, attribute names can be preceded with a label followed by a hash. Use c for class and m for meta attribute, i to ignore a column, and C, D, S for continuous, discrete and string attribute types. Examples: C#mpg, mS#name, i#dummy. Make sure to set **Import Options** in *File* widget and set the header to **Orange simplified header**.
- Orange's native format is a tab-delimited text file with three header rows. The first row contains attribute names, the second the type (**continuous**, **discrete** or **string**), and the third the optional element (**class**, **meta** or **string**).

Read more on loading your data *here*.

SQL Table

Reads data from an SQL database.



	A	B	C	D	E	F	G	H
1	mD#function	mS#gene	spo-early	spo-mid	c#heat 0	i#heat 10	i#heat 20	
2	Proteas	YDR427W	0.301	0.546		-0.009	0.024	
3	Proteas	YGL048C	0.208		-0.061	-0.039	0.003	
4	Resp	YBR039W	-0.179	-0.219	-0.097		-0.011	
5	Ribo	YKL180W	-0.085	-0.161	-0.061	-0.265	-0.419	
6	Ribo	YHR021C	-0.216	-0.253	-0.228	-0.168	-0.228	
7	Resp	YDR178W	0.017	0.07	0.058	0.286	0.205	
8	Resp	YLL041C	0.115		0.033	0.262	0.054	
9	Resp	YOR065W	0.005	-0.023	-0.038	0.222	0.088	
10								
11								
12								
13								

Signals

Inputs:

- (None)

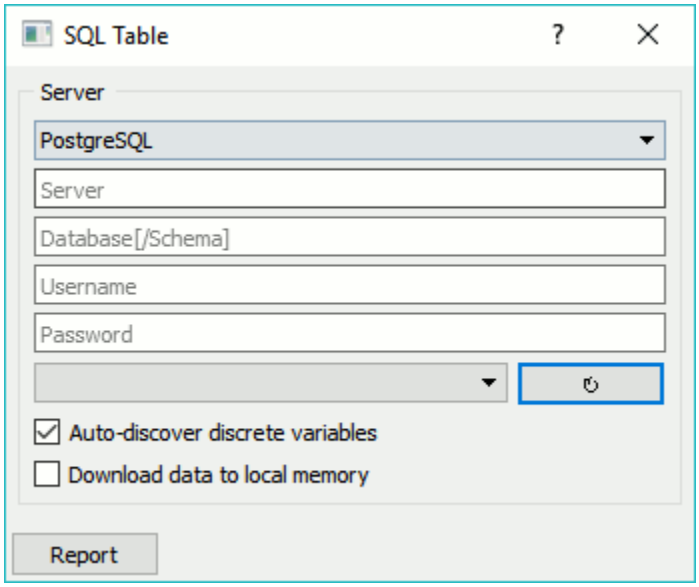
Outputs:

- **Data**

  Attribute-valued data from the database

Description

The **SQL** widget accesses data stored in an SQL database. It can connect to PostgreSQL (requires psycopg2 module) or SQL Server (requires pymssql module).



Save Data

Saves data to a file.

Signals

Inputs:

- **Data**

  A data set.

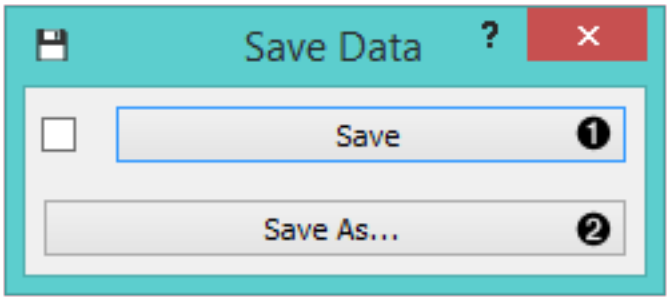
Outputs:

- (None)

Description

The **Save Data** widget considers a data set provided in the input channel and saves it to a data file with a specified name. It can save the data as a tab-delimited or a comma-separated file.

The widget does not save the data every time it receives a new signal in the input as this would constantly (and, mostly, inadvertently) overwrite the file. Instead, the data is saved only after a new file name is set or the user pushes the *Save* button.



1. Save by overwriting the existing file.
2. *Save as* to create a new file.

Example

In the workflow below, we used the *Zoo* data set. We loaded the data into the *Scatter Plot* widget, with which we selected a subset of data instances and pushed them to the **Save Data** widget to store them in a file.

Data Info

Displays information on a selected data set.

Signals

Inputs:

- **Data**

  A data set.

- **Selected Data**

  A data subset.

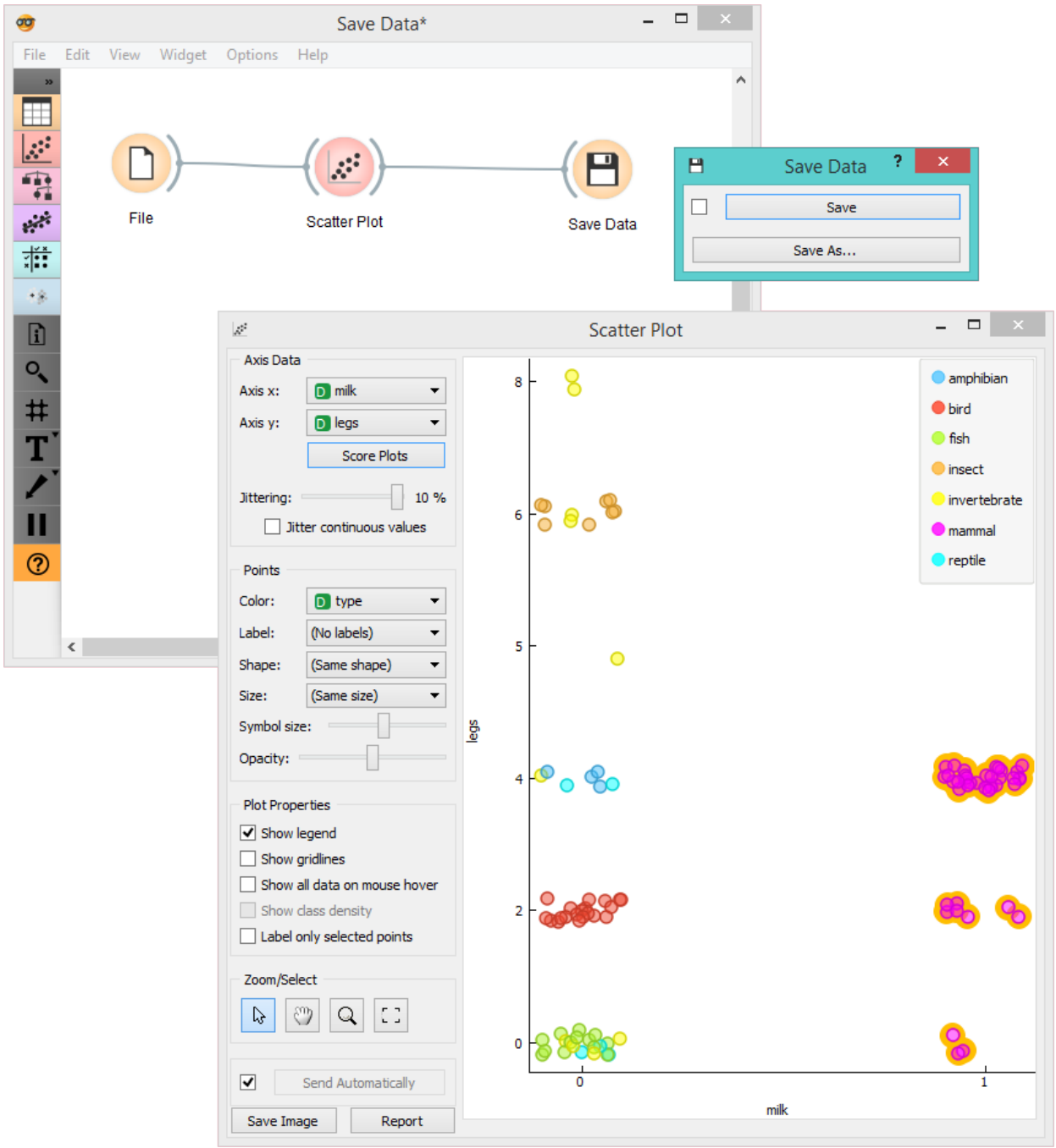
Outputs:

- (None)

Description

A simple widget that presents information on data set size, features, targets, meta attributes, and location.

1. Information on data set size
2. Information on discrete and continuous features

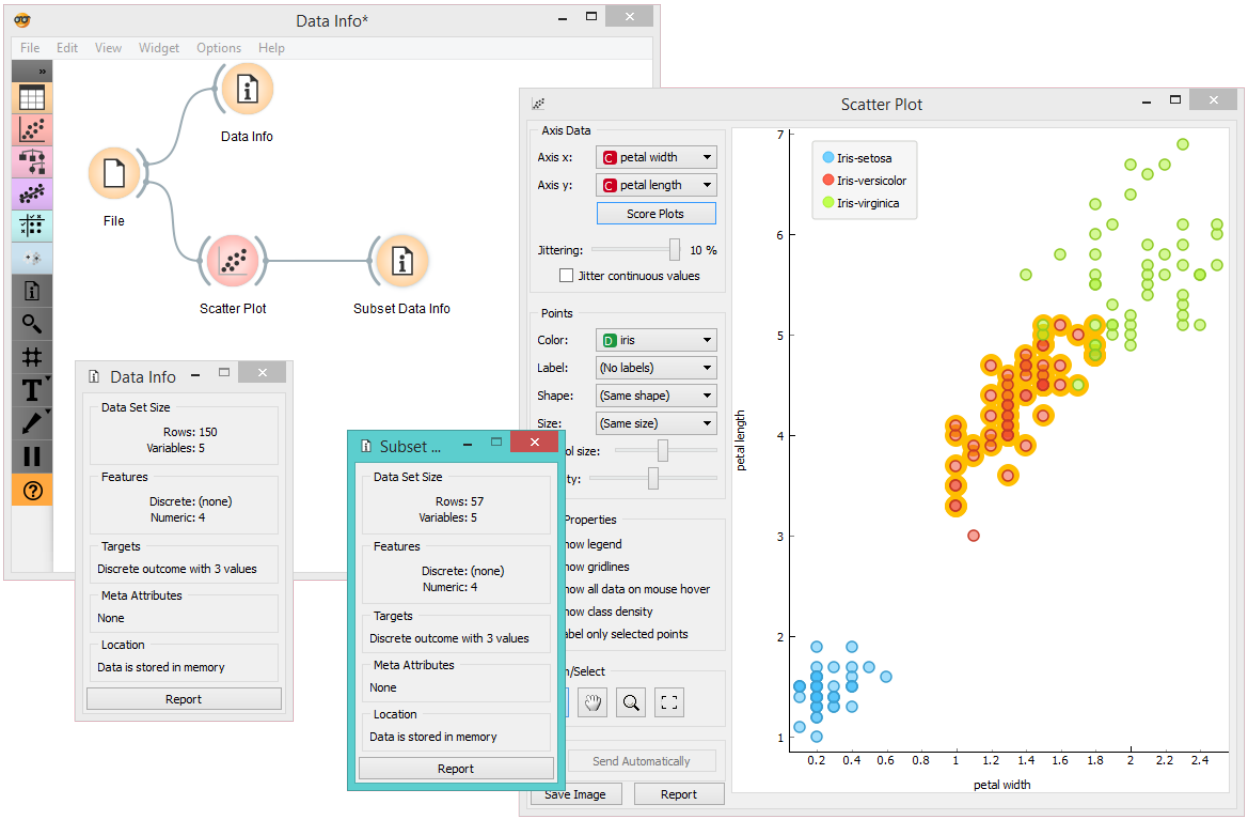

Save Data*
File Edit View Widget Options Help
File
Scatter Plot
Save Data
Save Data
Save
Save As...
Scatter Plot
Axis Data
Axis x: milk
Axis y: legs
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: type
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
amphibian
bird
fish
insect
invertebrate
mammal
reptile
legs
milk
0
1
2
4
5
6
8

Data Info
Data Set Size
1
Rows: 150
Variables: 5
Features
2
Discrete: (none)
Numeric: 4
Targets
3
Discrete outcome with 3 values
Meta Attributes
4
None
Location
5
Data is stored in memory
Report
6

3. Information on targets
4. Information on meta attributes
5. Information on where the data is stored
6. Produce a report.

Example

Below, we compare the basic statistics of two **Data Info** widgets - one with information on the entire data set and the other with information on the (manually) selected subset from the *Scatterplot* widget. We used the *Iris* data set.



Data Table

Displays attribute-value data in a spreadsheet.

Signals

Inputs:

- **Data**

  Attribute-valued data set.

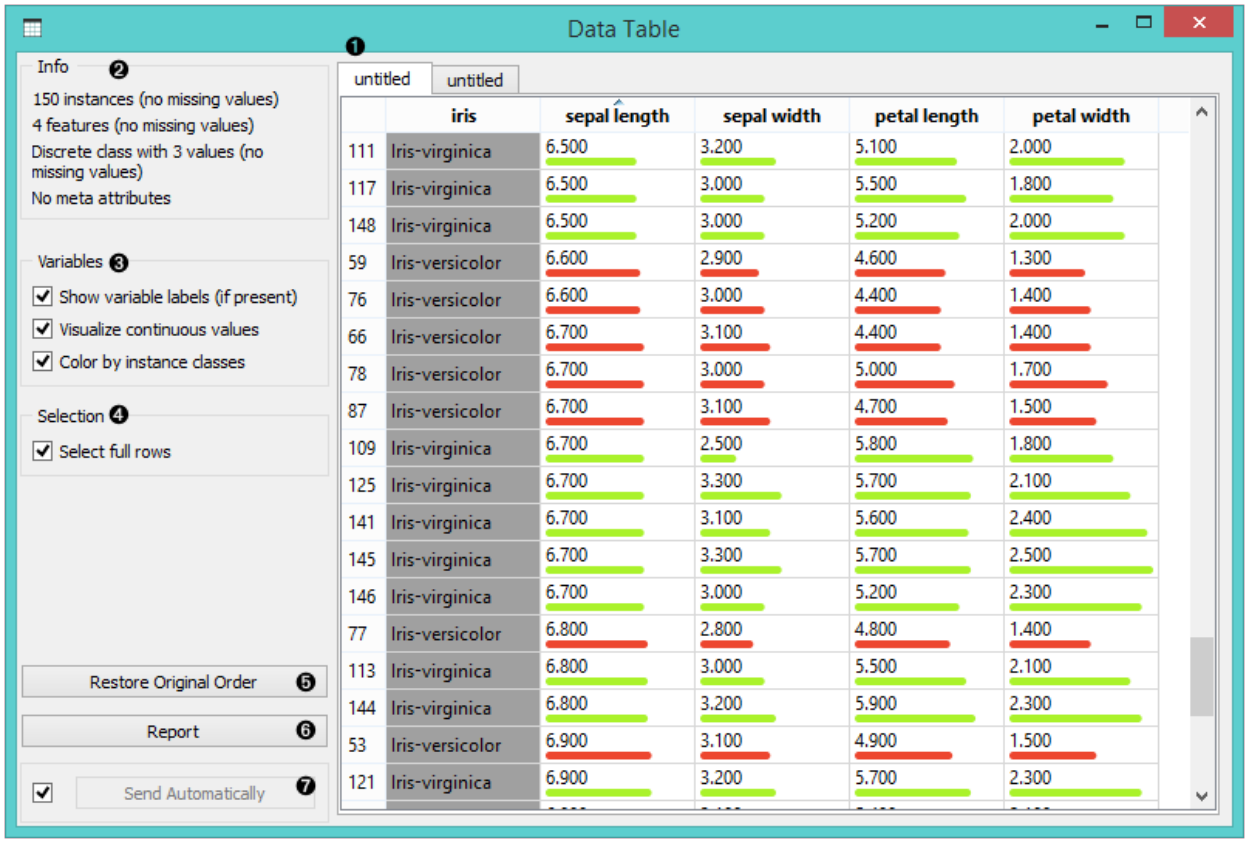
Outputs:

- **Selected Data**

  Selected data instances.

Description

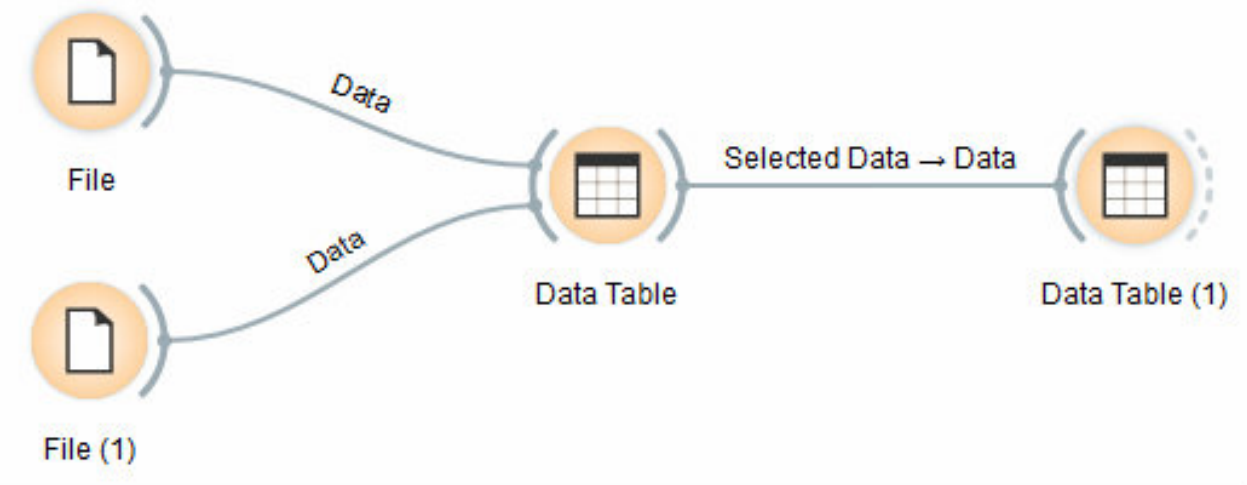
The **Data Table** widget receives one or more data sets in its input and presents them as a spreadsheet. Data instances may be sorted by attribute values. The widget also supports manual selection of data instances.



1. The name of the data set (usually the input data file). Data instances are in rows and their attribute values in columns. In this example, the data set is sorted by the attribute "sepal length".
2. Info on current data set size and number and types of attributes
3. Values of continuous attributes can be visualized with bars; colors can be attributed to different classes.
4. Data instances (rows) can be selected and sent to the widget's output channel.
5. Use the *Restore Original Order* button to reorder data instances after attribute-based sorting.
6. Produce a report.
7. While auto-send is on, all changes will be automatically communicated to other widgets. Otherwise, press *Send Selected Rows*.

Example

We used two *File* widgets to read the *Iris* and *Glass* data set (provided in Orange distribution), and send them to the **Data Table** widget.



Selected data instances in the first **Data Table** are passed to the second **Data Table**. Notice that we can select which data set to view (iris or glass). Changing from one data set to another alters the communicated selection of data instances if *Commit on any change* is selected.

Select Columns

Manual selection of data attributes and composition of data domain.

Signals

Inputs:

- **Data**

  Attribute-valued data set.

Outputs:

- **Data**

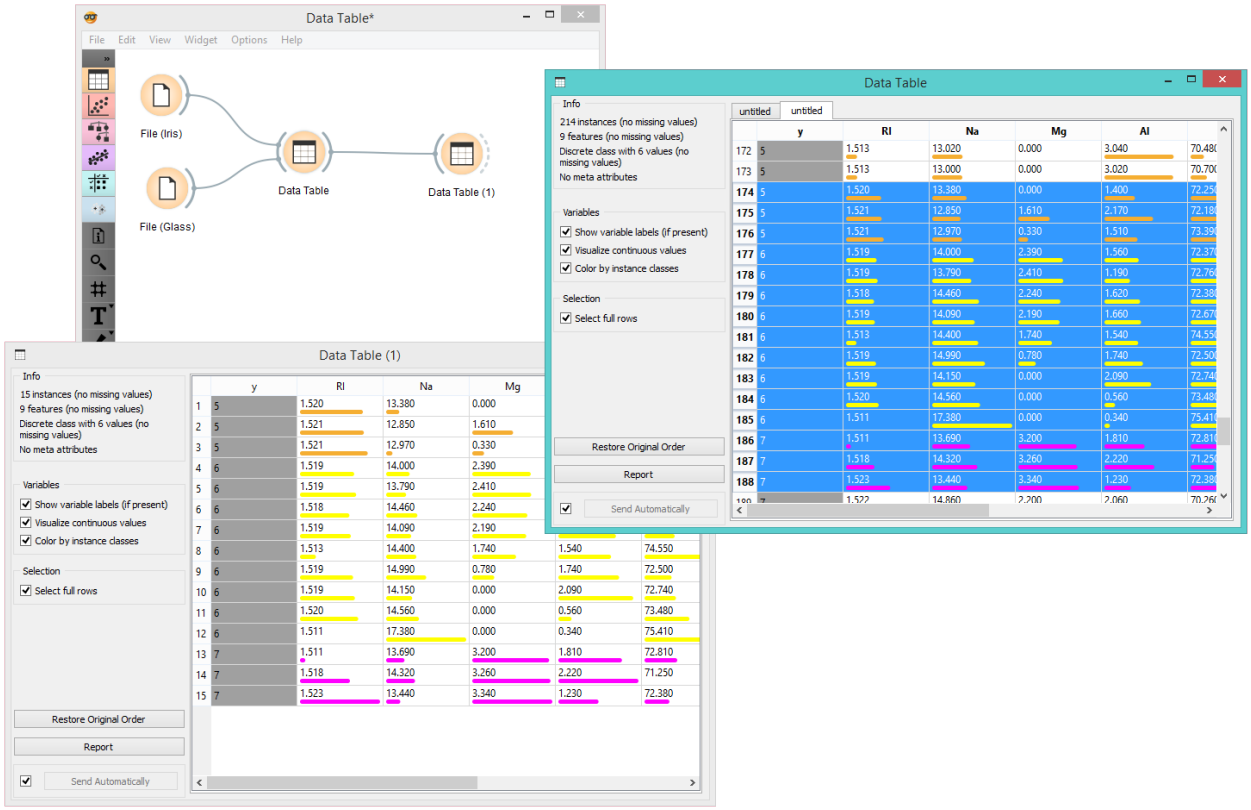
  Attribute-valued data set composed using the domain specification from the widget.

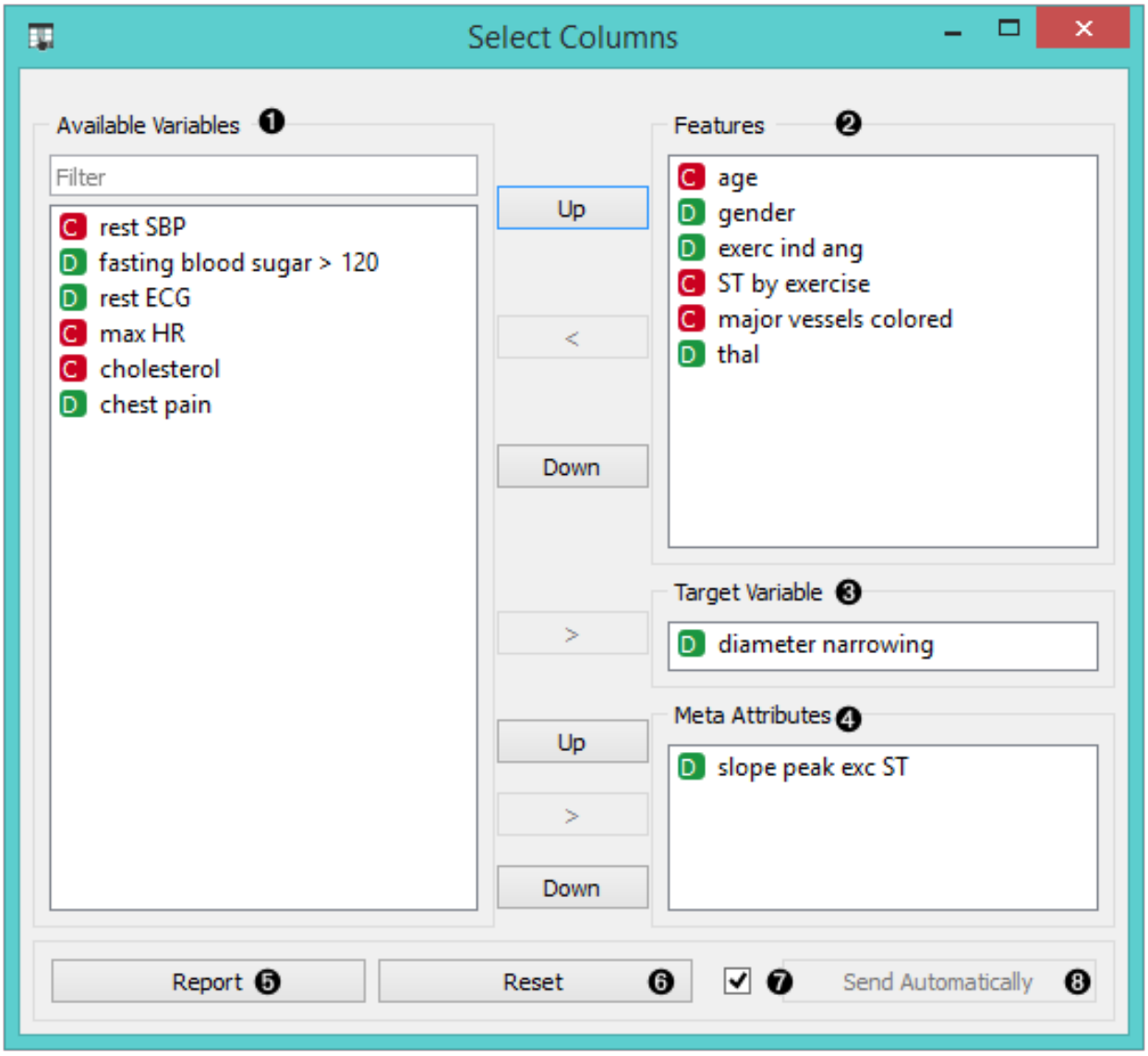
Description

The **Select Columns** widget is used to manually compose your data domain. The user can decide which attributes will be used and how. Orange distinguishes between ordinary attributes, (optional) class attributes and meta attributes. For instance, for building a classification model, the domain would be composed of a set of attributes and a discrete class attribute. Meta attributes are not used in modelling, but several widgets can use them as instance labels.

Orange attributes have a type and are either discrete, continuous or a character string. The attribute type is marked with a symbol appearing before the name of the attribute (D, C, S, respectively).

1. Left-out data attributes that will not be in the output data file

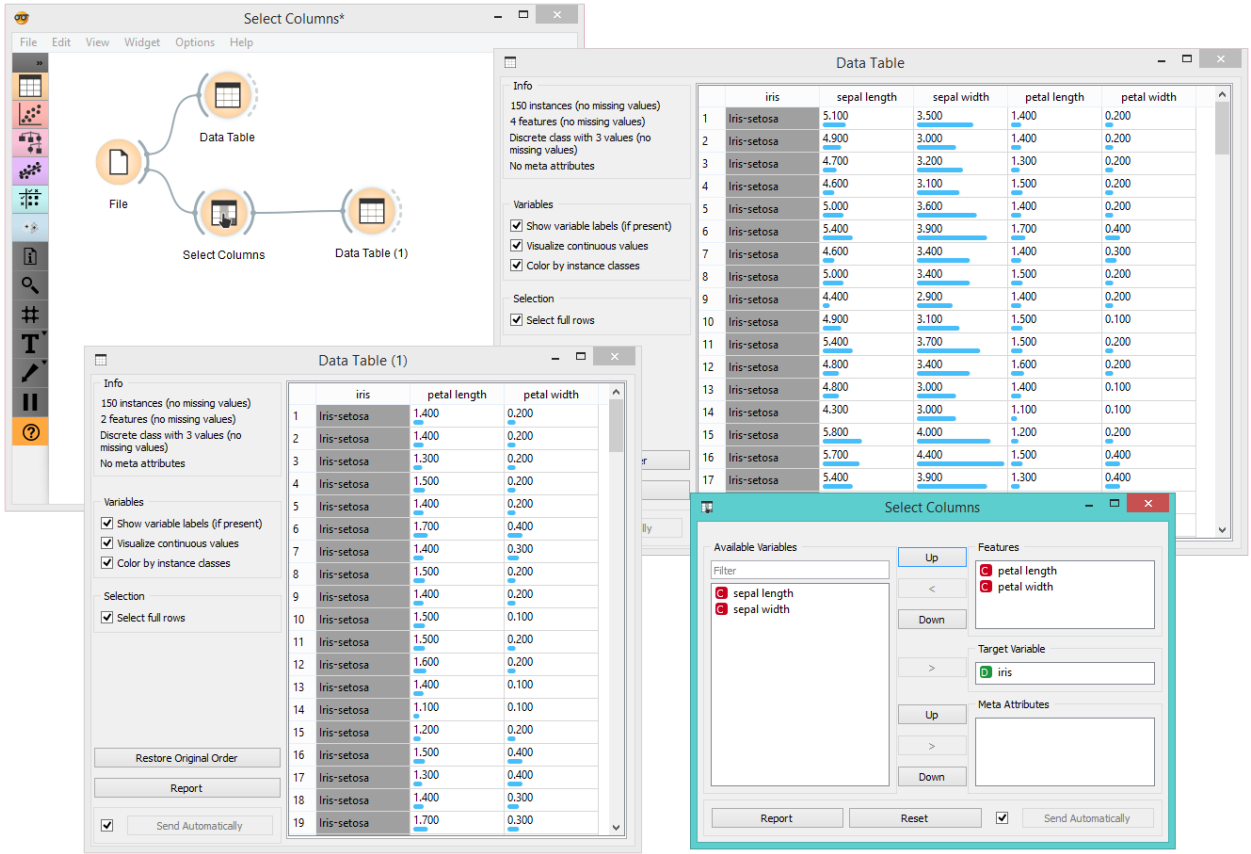

Data Table*
File Edit View Widget Options Help
File (Iris)
File (Glass)
Data Table
Data Table (1)
Data Table
Info
214 instances (no missing values)
9 features (no missing values)
Discrete class with 6 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
untitled
untitled
Data Table (1)
Info
15 instances (no missing values)
9 features (no missing values)
Discrete class with 6 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically


Select Columns
Available Variables ❶
Filter
rest SBP
fasting blood sugar > 120
rest ECG
max HR
cholesterol
chest pain
Up
<
Down
>
Up
>
Down
Features ❷
age
gender
exerc ind ang
ST by exercise
major vessels colored
thal
Target Variable ❸
diameter narrowing
Meta Attributes ❹
slope peak exc ST
Report ❺
Reset ❻
❼
Send Automatically ❽

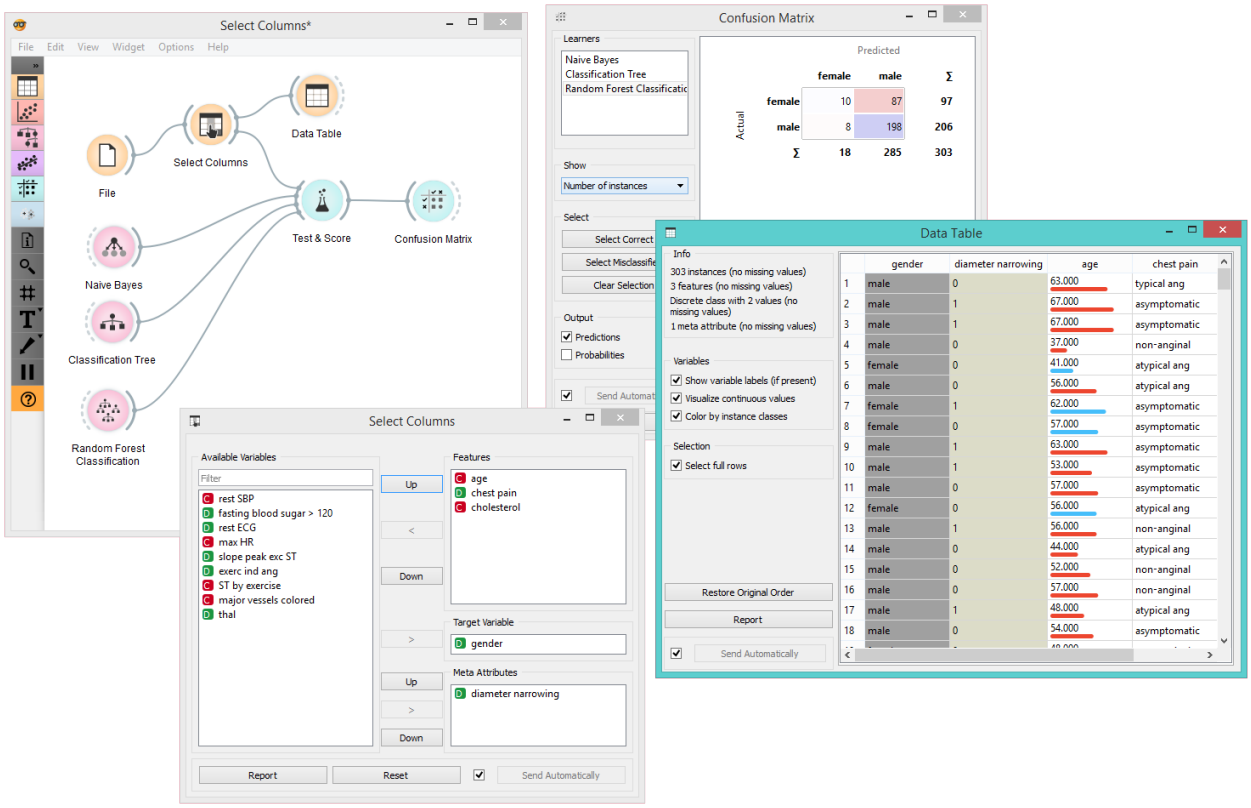
2. Data attributes in the new data file
3. Target variable. If none, the new data set will be without a target variable.
4. Meta attributes of the new data file. These attributes are included in the data set but are, for most methods, not considered in the analysis.
5. Produce a report.
6. Reset the domain composition to that of the input data file.
7. Tick if you wish to auto-apply changes of the data domain.
8. Apply changes of the data domain and send the new data file to the output channel of the widget.

Examples

In the workflow below, the *Iris* data from the *File* widget is fed into the **Select Columns** widget, where we select to output only two attributes (namely petal width and petal length). We view both the original data set and the data set with selected columns in the *Data Table* widget.



For a more complex use of the widget, we composed a workflow to redefine the classification problem in the *heart-disease* data set. Originally, the task was to predict if the patient has a coronary artery diameter narrowing. We changed the problem to that of gender classification, based on age, chest pain and cholesterol level, and informatively kept the diameter narrowing as a meta attribute.


Select Columns*
File Edit View Widget Options Help
Data Table
Select Columns
File
Test & Score
Confusion Matrix
Naive Bayes
Classification Tree
Random Forest Classification
Confusion Matrix
Learners
Naive Bayes
Classification Tree
Random Forest Classificatic
Show
Number of instances
Select
Select Correct
Select Misclassifie
Clear Selection
Output
Predictions
Probabilities
Predicted
female male Σ
female 10 87 97
male 8 198 206
Σ 18 285 303
Actual
Select Columns
Available Variables
Filter
rest SBP
fasting blood sugar > 120
rest ECG
max HR
slope peak exc ST
exerc ind ang
ST by exercise
major vessels colored
thal
Up
Down
Features
age
chest pain
cholesterol
Target Variable
gender
Meta Attributes
diameter narrowing
Report
Reset
Send Automatically
Data Table
Info
303 instances (no missing values)
3 features (no missing values)
Discrete class with 2 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
gender diameter narrowing age chest pain
1 male 0 63.000 typical ang
2 male 1 67.000 asymptomatic
3 male 1 67.000 asymptomatic
4 male 0 37.000 non-anginal
5 female 0 41.000 atypical ang
6 male 0 56.000 atypical ang
7 female 1 62.000 asymptomatic
8 female 0 57.000 asymptomatic
9 male 1 63.000 asymptomatic
10 male 1 53.000 asymptomatic
11 male 0 57.000 asymptomatic
12 female 0 56.000 atypical ang
13 male 1 56.000 non-anginal
14 male 0 44.000 atypical ang
15 male 0 52.000 non-anginal
16 male 0 57.000 non-anginal
17 male 1 48.000 atypical ang
18 male 0 54.000 asymptomatic

Select Rows

Selects data instances based on conditions over data features.

Signals

Inputs:

- **Data**

  Data set.

Outputs:

- **Matching Data**

  Instances that match the conditions.

- **Non-Matching Data**

  Instances that do not match the conditions.

Description

This widget selects a subset from an input data set, based on user-defined conditions. Instances that match the selection rule are placed in the output *Matching Data* channel.

Criteria for data selection are presented as a collection of conjuncted terms (i.e. selected items are those matching all the terms in '*Conditions*').

Condition terms are defined through selecting an attribute, selecting an operator from a list of operators, and, if needed, defining the value to be used in the condition term. Operators are different for discrete, continuous and string attributes.

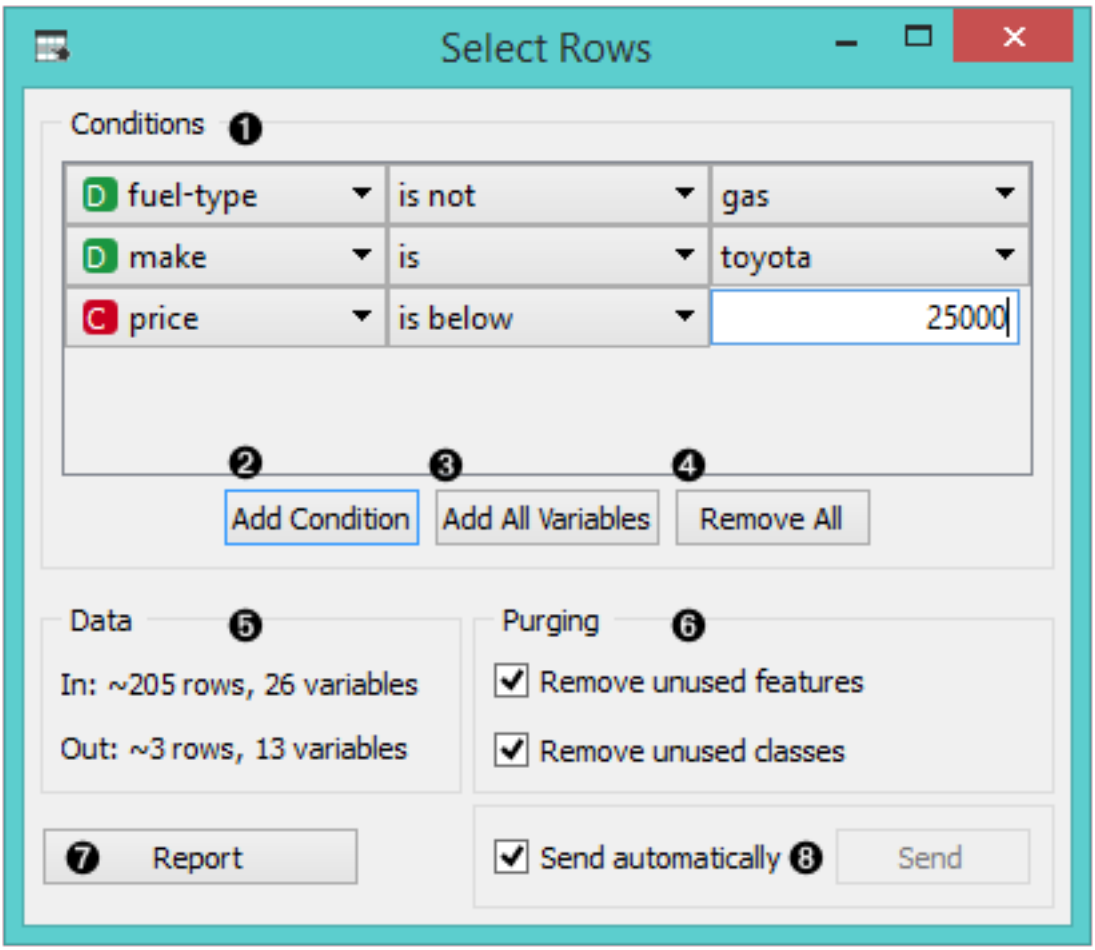
1. Conditions you want to apply, their operators and related values
2. Add a new condition to the list of conditions.
3. Add all the possible variables at once.
4. Remove all the listed variables at once.
5. Information on the input data set and information on instances that match the condition(s)
6. Purge the output data.
7. When the *Send automatically* box is ticked, all changes will be automatically communicated to other widgets.
8. Produce a report.

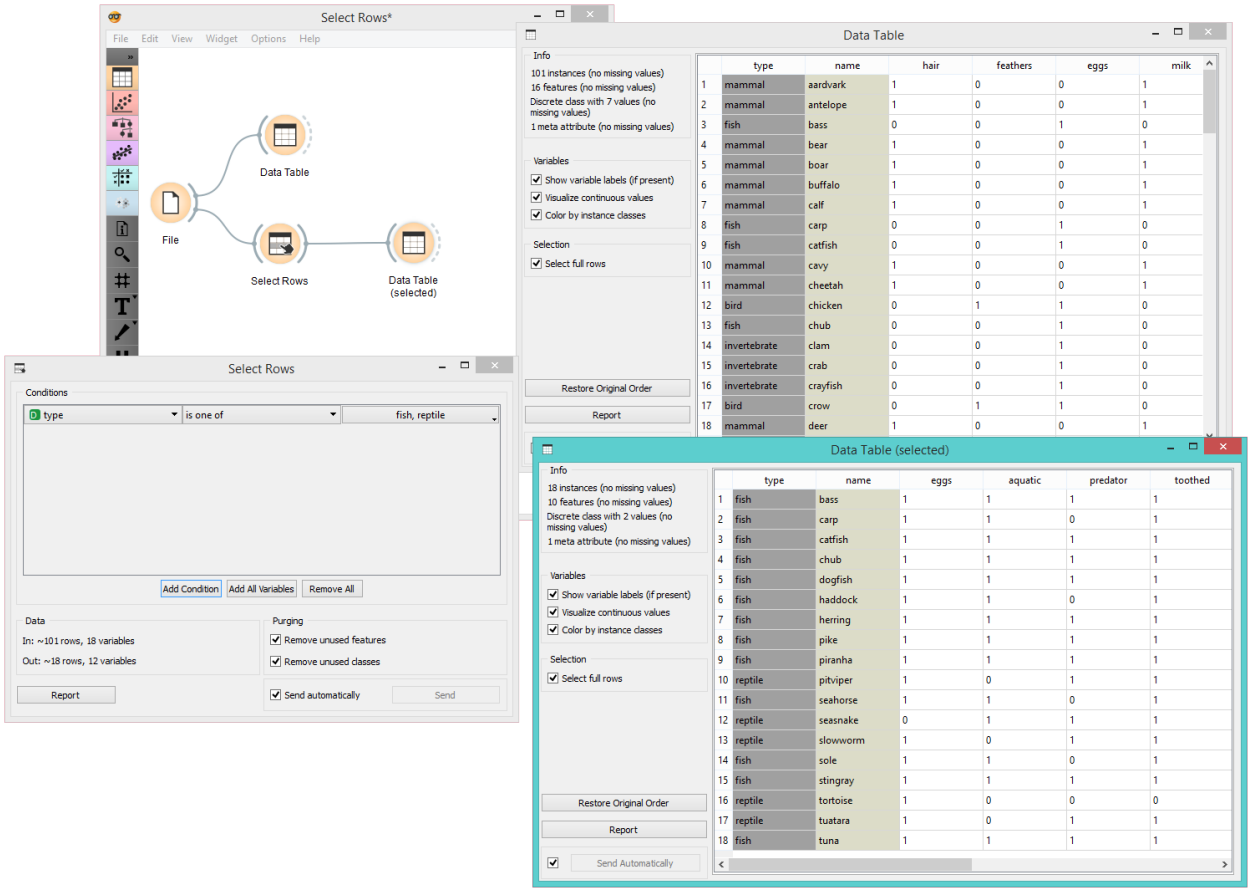
Any change in the composition of the condition will update the information pane (*Data Out*).

If *Send automatically* is selected, then the output is updated on any change in the composition of the condition or any of its terms.

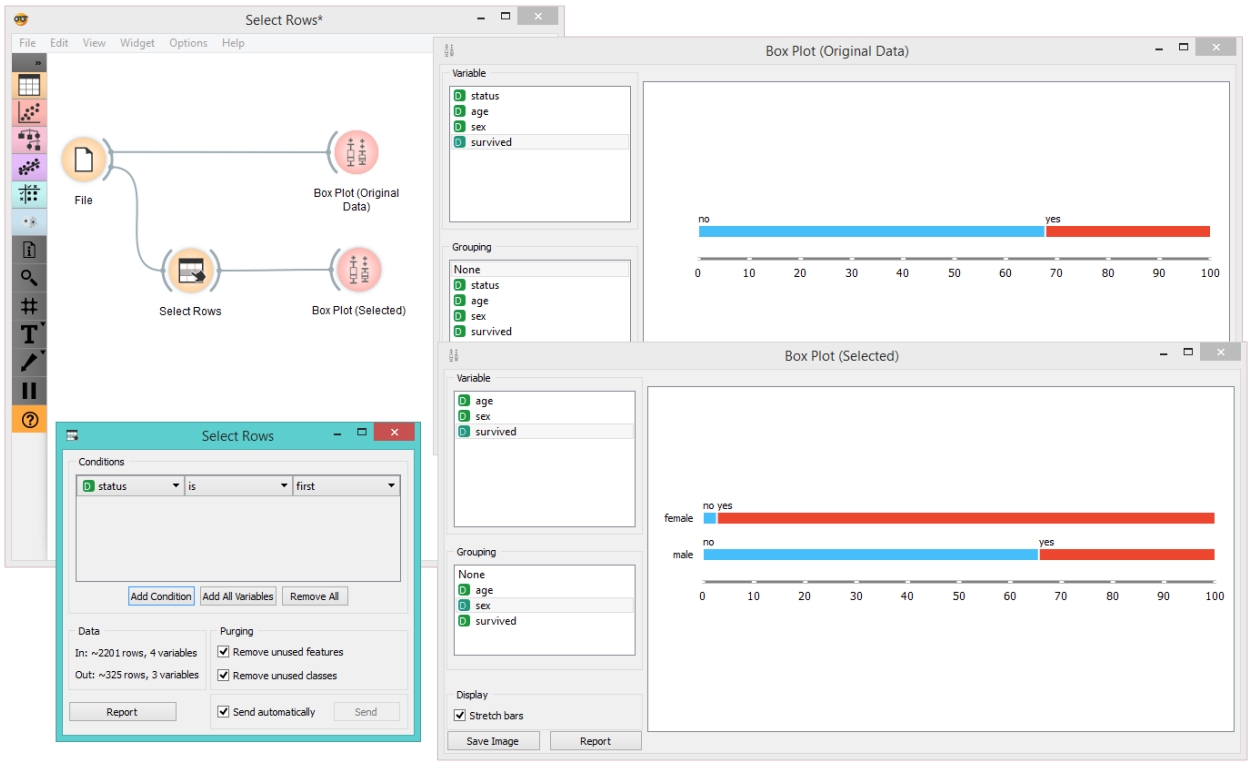
Example

In the workflow below, we used the *Zoo* data from the *File* widget and fed it into the **Select Rows** widget. In the widget, we chose to output only two animal types, namely fish and reptiles. We can inspect both the original data set and the data set with selected rows in the *Data Table* widget.


Select Rows
Conditions
1
fuel-type
is not
gas
make
is
toyota
price
is below
25000
2
3
4
Add Condition
Add All Variables
Remove All
Data
5
In: ~205 rows, 26 variables
Out: ~3 rows, 13 variables
Purging
6
Remove unused features
Remove unused classes
7
Report
Send automatically
8
Send


Select Rows*
File Edit View Widget Options Help
Data Table
File
Select Rows
Data Table (selected)
Select Rows
Conditions
type
is one of
fish, reptile
Add Condition
Add All Variables
Remove All
Data
In: ~101 rows, 18 variables
Out: ~18 rows, 12 variables
Report
Purging
Remove unused features
Remove unused classes
Send automatically
Send
Data Table
Info
101 instances (no missing values)
16 features (no missing values)
Discrete class with 7 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
type name hair feathers eggs milk
1 mammal aardvark 1 0 0 1
2 mammal antelope 1 0 0 1
3 fish bass 0 0 1 0
4 mammal bear 1 0 0 1
5 mammal boar 1 0 0 1
6 mammal buffalo 1 0 0 1
7 mammal calf 1 0 0 1
8 fish carp 0 0 1 0
9 fish catfish 0 0 1 0
10 mammal cavy 1 0 0 1
11 mammal cheetah 1 0 0 1
12 bird chicken 0 1 1 0
13 fish chub 0 0 1 0
14 invertebrate clam 0 0 1 0
15 invertebrate crab 0 0 1 0
16 invertebrate crayfish 0 0 1 0
17 bird crow 0 1 1 0
18 mammal deer 1 0 0 1
Data Table (selected)
Info
18 instances (no missing values)
10 features (no missing values)
Discrete class with 2 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
type name eggs aquatic predator toothed
1 fish bass 1 1 1 1
2 fish carp 1 1 0 1
3 fish catfish 1 1 1 1
4 fish chub 1 1 1 1
5 fish dogfish 1 1 1 1
6 fish haddock 1 1 0 1
7 fish herring 1 1 1 1
8 fish pike 1 1 1 1
9 fish piranha 1 1 1 1
10 reptile pitviper 1 0 1 1
11 fish seahorse 1 1 0 1
12 reptile seasnake 0 1 1 1
13 reptile slowworm 1 0 1 1
14 fish sole 1 1 0 1
15 fish stingray 1 1 1 1
16 reptile tortoise 1 0 0 0
17 reptile tuatara 1 0 1 1
18 fish tuna 1 1 1 1

In the next example, we used the data from the *Titanic* data set and similarly fed it into the *Box Plot* widget. We first observed the entire data set based on survival. Then we selected only first class passengers in the **Select Rows** widget and fed it again into the *Box Plot*. There we could see all the first class passengers listed by their survival rate and grouped by gender.



Data Sampler

Selects a subset of data instances from an input data set.

Signals

Inputs:

- **Data**

  Input data set to be sampled.

Outputs:

- **Data Sample**

  A set of sampled data instances.

- **Remaining Data**

  All other data instances from the input data set, which are not included in the sample.

Description

The **Data Sampler** widget implements several means of sampling data from an input channel. It outputs a sampled and a complementary data set (with instances from the input set that are not included in the sampled data set). The output is processed after the input data set is provided and *Sample Data* is pressed.

1. Information on the input and output data set
2. The desired sampling method:
   - **Fixed proportion of data** returns a selected percentage of the entire data (e.g. 70% of all the data)
   - **Fixed sample size** returns a selected number of data instances with a chance to set *Sample with replacement*, which always samples from the entire data set (does not subtract instances already in the subset)
   - Cross Validation partitions data instances into complementary subsets, where you can select the number of folds (subsets) and which fold you want to use as a sample.
3. *Replicable sampling* maintains sampling patterns that can be carried across users, while *stratification* mimics the composition of the input data set.
4. Produce a report.
5. Press *Sample data* to output the data sample.

Examples

First, let's see how the **Data Sampler** works. Let's look at the information on the original data set in the *Data Info* widget. We see there are 24 instances in the data (we used *lenses.tab*). We sampled the data with the **Data Sampler** widget and we chose to go with a fixed sample size of 5 instances for simplicity. We can observe the sampled data in the *Data Table* widget. The second *Data Table* shows the remaining 19 instances that weren't in the sample.

In the workflow below, we have sampled 10 data instances from the *Iris* data set and sent the original data and the sample to *Scatter Plot* widget for exploratory data analysis. The sampled data instances are plotted with filled circles, while the original data set is represented with empty circles.

Transpose

Transposes a data table.

Signals

Inputs:

- **Data**

  A data set.

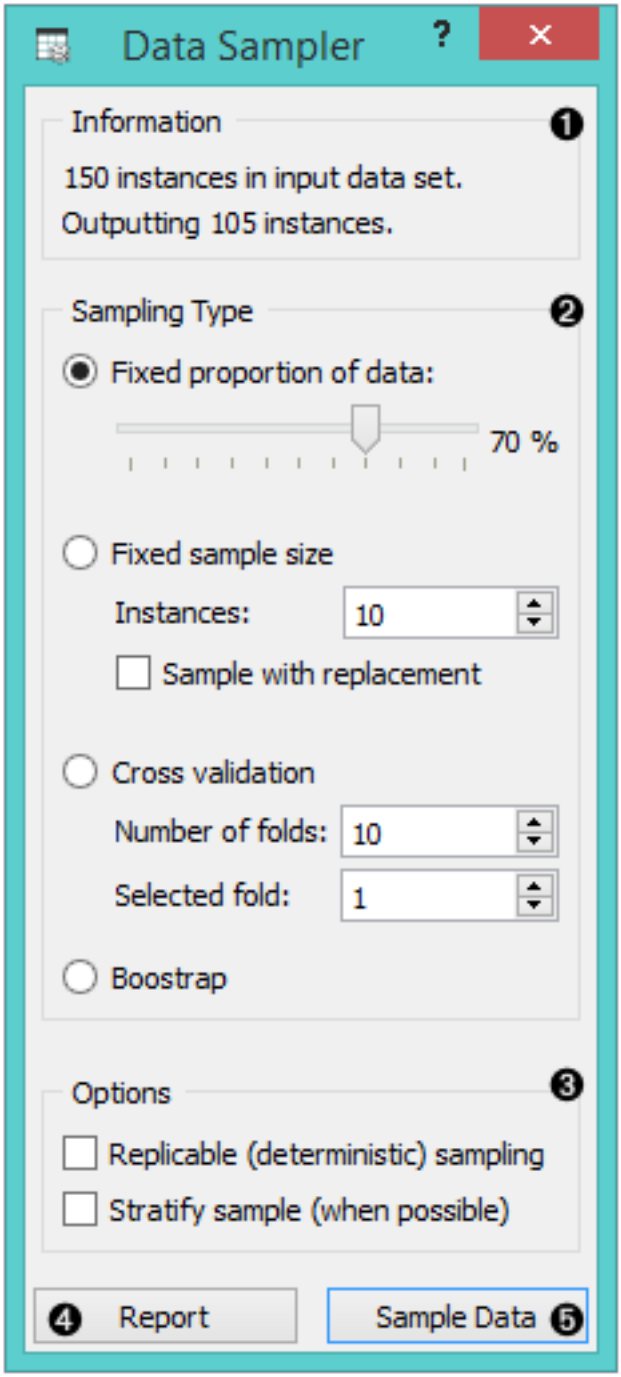
Outputs:

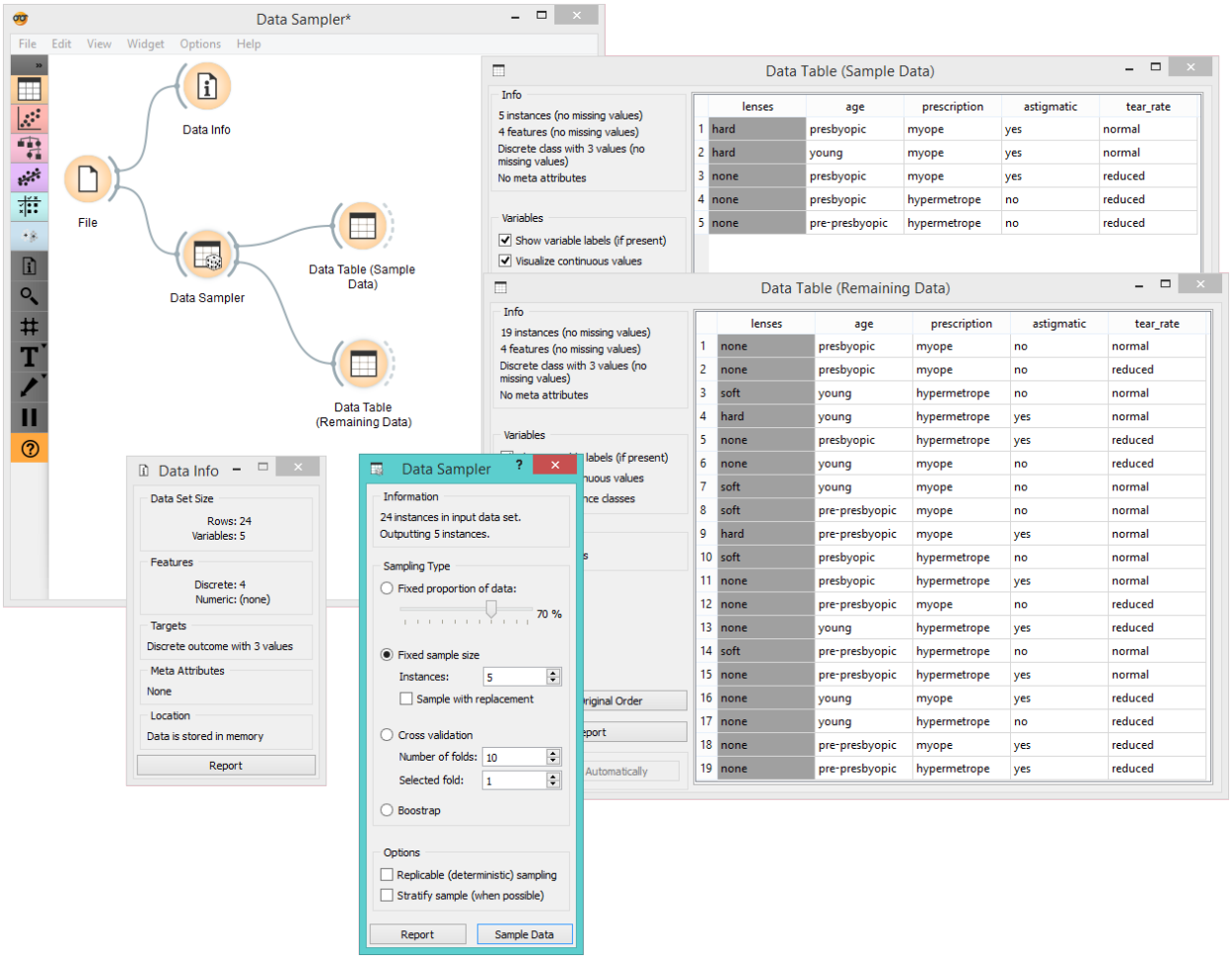
- **Data**

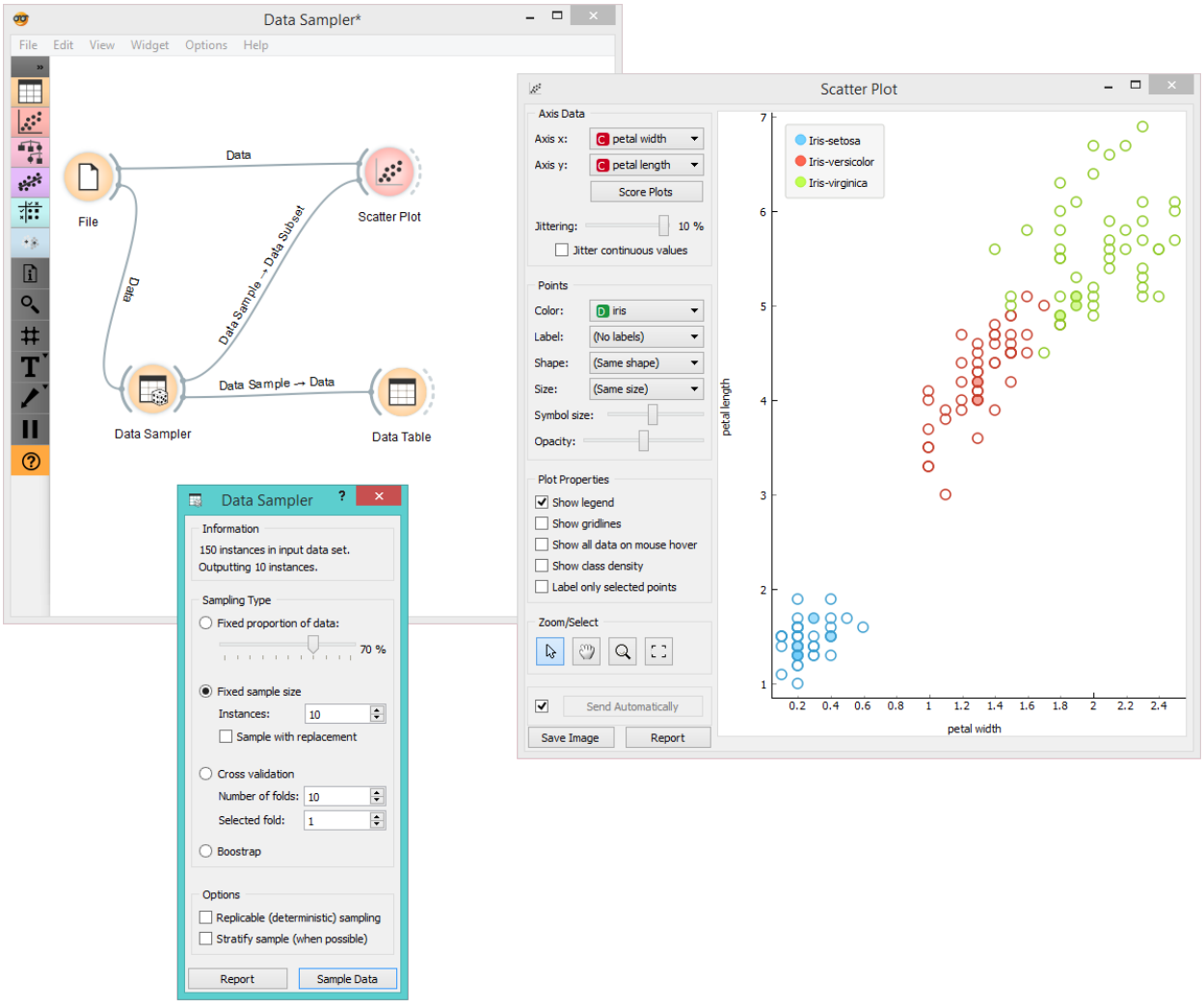
  Transposed data set

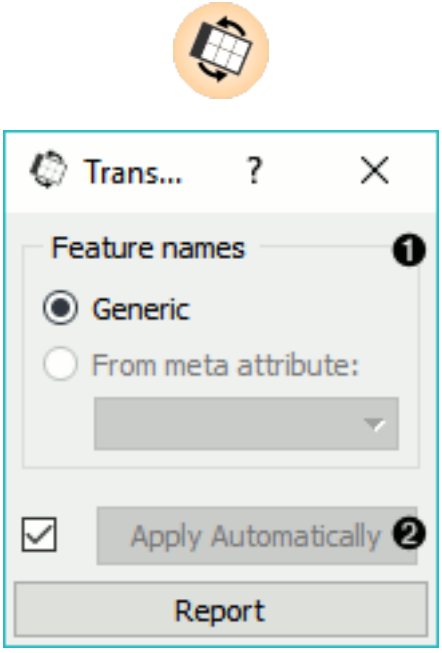
Description

Transpose widget transposes data table.


Data Sampler
Information
1
150 instances in input data set.
Outputting 105 instances.
Sampling Type
2
Fixed proportion of data:
70 %
Fixed sample size
Instances:
10
Sample with replacement
Cross validation
Number of folds:
10
Selected fold:
1
Boostrap
Options
3
Replicable (deterministic) sampling
Stratify sample (when possible)
4
Report
Sample Data
5

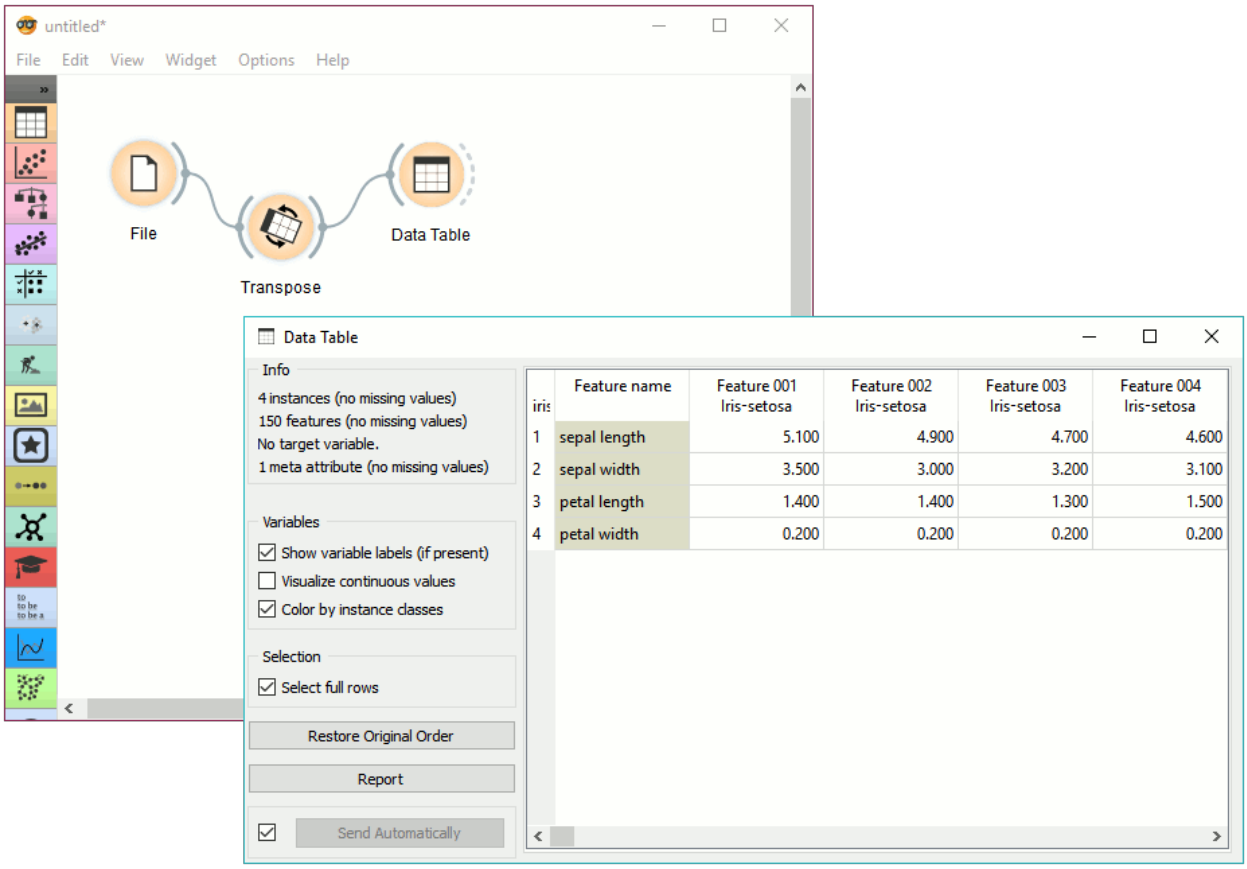

Data Sampler*
File Edit View Widget Options Help
Data Info
File
Data Sampler
Data Table (Sample Data)
Data Table (Remaining Data)
Data Table (Sample Data)
Info
5 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
lenses age prescription astigmatic tear_rate
1 hard presbyopic myope yes normal
2 hard young myope yes normal
3 none presbyopic myope yes reduced
4 none presbyopic hypermetrope no reduced
5 none pre-presbyopic hypermetrope no reduced
Data Table (Remaining Data)
Info
19 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
lenses age prescription astigmatic tear_rate
1 none presbyopic myope no normal
2 none presbyopic myope no reduced
3 soft young hypermetrope no normal
4 hard young hypermetrope yes normal
5 none presbyopic hypermetrope yes reduced
6 none young myope no reduced
7 soft young myope no normal
8 soft pre-presbyopic myope no normal
9 hard pre-presbyopic myope yes normal
10 soft presbyopic hypermetrope no normal
11 none presbyopic hypermetrope yes normal
12 none pre-presbyopic myope no reduced
13 none young hypermetrope yes reduced
14 soft pre-presbyopic hypermetrope no normal
15 none pre-presbyopic hypermetrope yes normal
16 none young myope yes reduced
17 none young hypermetrope no reduced
18 none pre-presbyopic myope yes reduced
19 none pre-presbyopic hypermetrope yes reduced
Data Info
Data Set Size
Rows: 24
Variables: 5
Features
Discrete: 4
Numeric: (none)
Targets
Discrete outcome with 3 values
Meta Attributes
None
Location
Data is stored in memory
Report
Data Sampler
Information
24 instances in input data set.
Outputting 5 instances.
Sampling Type
Fixed proportion of data:
70 %
Fixed sample size
Instances: 5
Sample with replacement
Cross validation
Number of folds: 10
Selected fold: 1
Boostrap
Options
Replicable (deterministic) sampling
Stratify sample (when possible)
Report
Sample Data


Data Sampler*
File Edit View Widget Options Help
File
Data
Scatter Plot
Data
Data Sample → Data Subset
Data Sample → Data
Data Sampler
Data Table
Scatter Plot
Axis Data
Axis x: petal width
Axis y: petal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal length
petal width
Data Sampler
Information
150 instances in input data set.
Outputting 10 instances.
Sampling Type
Fixed proportion of data:
70 %
Fixed sample size
Instances: 10
Sample with replacement
Cross validation
Number of folds: 10
Selected fold: 1
Boostrap
Options
Replicable (deterministic) sampling
Stratify sample (when possible)
Report
Sample Data


Trans...
Feature names
1
Generic
From meta attribute:
Apply Automatically
2
Report

Example

This is a simple workflow showing how to use **Transpose**. Connect the widget to *File* widget. The output of **Transpose** is a transposed data table with rows as columns and columns as rows. You can observe the result in a *Data Table*.



Discretize

Discretizes continuous attributes from an input data set.

Signals

Inputs:

- **Data**

  Attribute-valued data set.

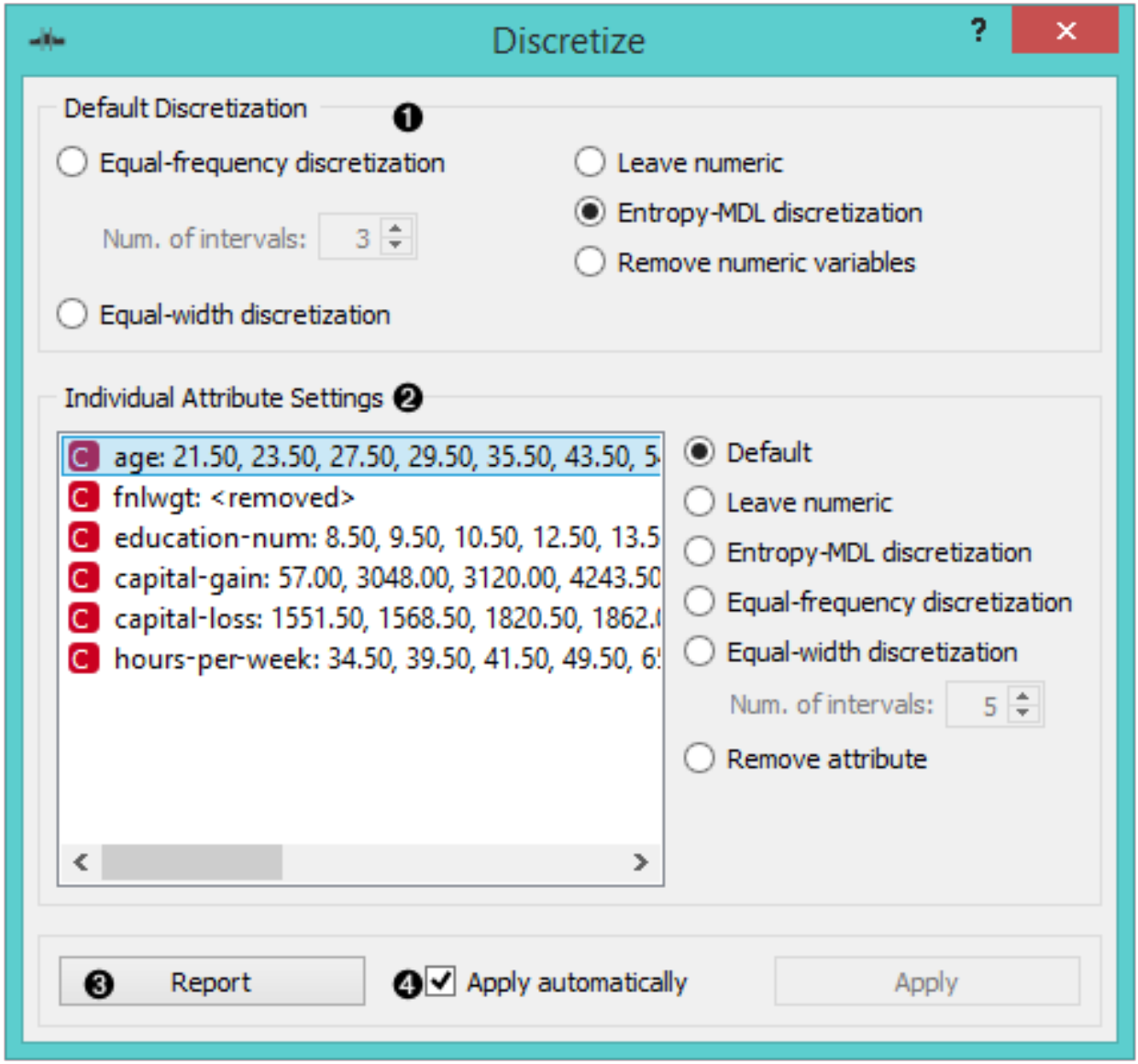
Outputs:

- **Data**

  A data set with discretized values.

Description

The **Discretize** widget discretizes continuous attributes with a selected method.



1. The basic version of the widget is rather simple. It allows choosing between three different discretizations.
   - Entropy-MDL, invented by Fayyad and Irani is a top-down discretization, which recursively splits the attribute at a cut maximizing information gain, until the gain is lower than the minimal description length of the cut. This discretization can result in an arbitrary number of intervals, including a single interval, in which case the attribute is discarded as useless (removed).
   - Equal-frequency splits the attribute into a given number of intervals, so that they each contain approximately the same number of instances.
   - Equal-width evenly splits the range between the smallest and the largest observed value. The *Number of intervals* can be set manually.
   - The widget can also be set to leave the attributes continuous or to remove them.

2. To treat attributes individually, go to **Individual Attribute Settings**. They show a specific discretization of each attribute and allow changes. First, the top left list shows the cut-off points for each attribute. In the snapshot, we used the entropy-MDL discretization, which determines the optimal number of intervals automatically; we can see it discretized the age into seven intervals with cut-offs at 21.50, 23.50, 27.50, 35.50, 43.50, 54.50 and 61.50, respectively, while the capital-gain got split into many intervals with several cut-offs. The final weight (fnlwgt), for instance, was left with a single interval and thus removed.

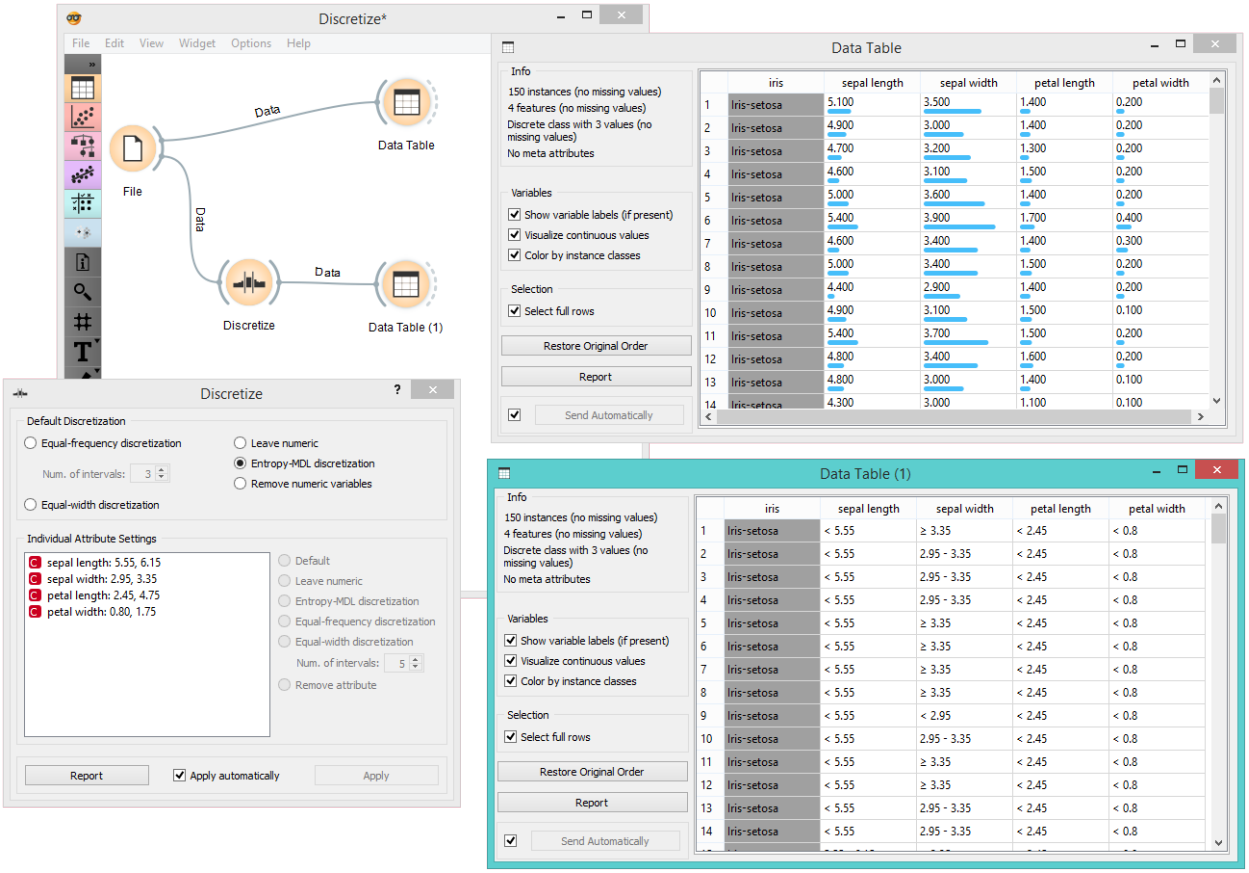
   On the right, we can select a specific discretization method for each attribute. Attribute *"fnlwgt"* would be removed by the MDL-based discretization, so to prevent its removal, we select the attribute and choose, for instance, **Equal-frequency discretization**. We could also choose to leave the attribute continuous.

3. Produce a report.

4. Tick *Apply automatically* for the widget to automatically commit changes. Alternatively, press *Apply*.

Example

In the schema below, we show the *Iris* data set with continuous attributes (as in the original data file) and with discretized attributes.



Continuize

Turns discrete attributes into continuous dummy variables.

Signals

Inputs:

- **Data**

  Input data set

Outputs:

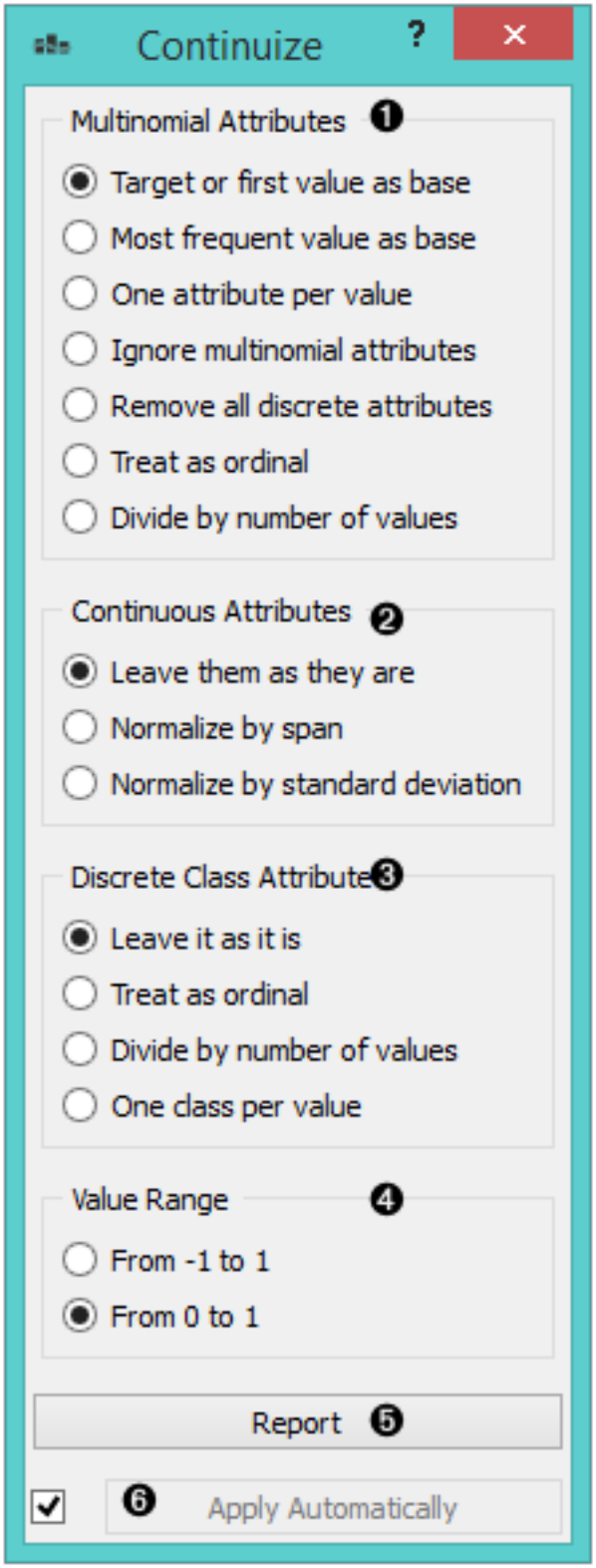
- **Data**

  Output data set

Description

The **Continuize** widget receives a data set in the input and outputs the same data set in which the discrete attributes (including binary attributes) are replaced with continuous ones.

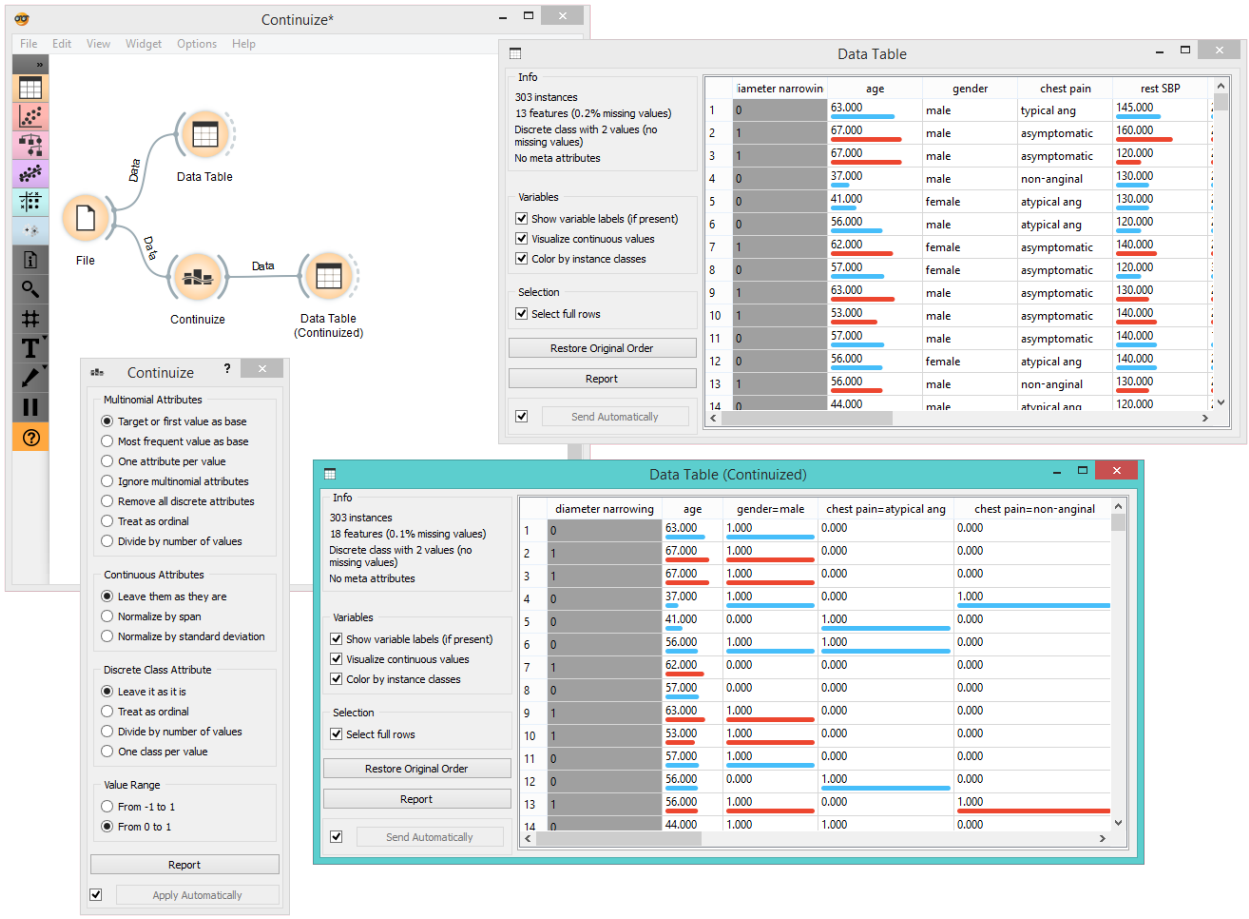
1. Continuization methods, which define the treatment of multivalued discrete attributes. Say that we have a discrete attribute status with the values low, middle and high, listed in that order. Options for their transformation are:
   - **Target or First value as base**: the attribute will be transformed into two continuous attributes, status=middle with values 0 or 1 signifying whether the original attribute had value middle on a particular example, and similarly, status=high. Hence, a three-valued attribute is transformed into two continuous attributes, corresponding to all except the first value of the attribute.
   - **Most frequent value as base**: similar to the above, except that the data is analyzed and the most frequent value is used as a base. So, if most examples have the value middle, the two newly constructed continuous attributes will be status=low and status=high.
   - **One attribute per value**: this would construct three continuous attributes out of a three-valued discrete one.
   - **Ignore multinominal attributes**: removes the multinominal attributes from the data.
   - **Treat as ordinal**: converts the attribute into a continuous attribute with values 0, 1, and 2.
   - **Divide by number of values**: same as above, except that the values are normalized into range 0-1. So, our case would give values 0, 0.5 and 1.
2. Define the treatment of continuous attributes. You will usually prefer the *Leave them as they are* option. The alternative is *Normalize by span*, which will subtract the lowest value found in the data and divide by the span, so all values will fit into [0, 1]. Finally, *Normalize by standard deviation* subtracts the average and divides by the deviation.
3. Define the treatment of class attributes. Besides leaving it as it is, there are also a couple of options available for multinominal attributes, except for those options which split the attribute into more than one attribute - this obviously cannot be supported since you cannot have more than one class attribute.
4. With *value range*, you can define the values of new attributes. In the above text, we supposed the range *from 0 to 1*. You can change it to *from -1 to 1*.
5. Produce a report.


Continuize
Multinomial Attributes
1
Target or first value as base
Most frequent value as base
One attribute per value
Ignore multinomial attributes
Remove all discrete attributes
Treat as ordinal
Divide by number of values
Continuous Attributes
2
Leave them as they are
Normalize by span
Normalize by standard deviation
Discrete Class Attribute
3
Leave it as it is
Treat as ordinal
Divide by number of values
One class per value
Value Range
4
From -1 to 1
From 0 to 1
Report
5
6
Apply Automatically

6. If *Apply automatically* is ticked, changes are committed automatically. Otherwise, you have to press *Apply* after each change.

Examples

First, let's see what is the output of the **Continuize** widget. We feed the original data (the *Heart disease* data set) into the *Data Table* and see how they look like. Then we continuize the discrete values and observe them in another *Data Table*.



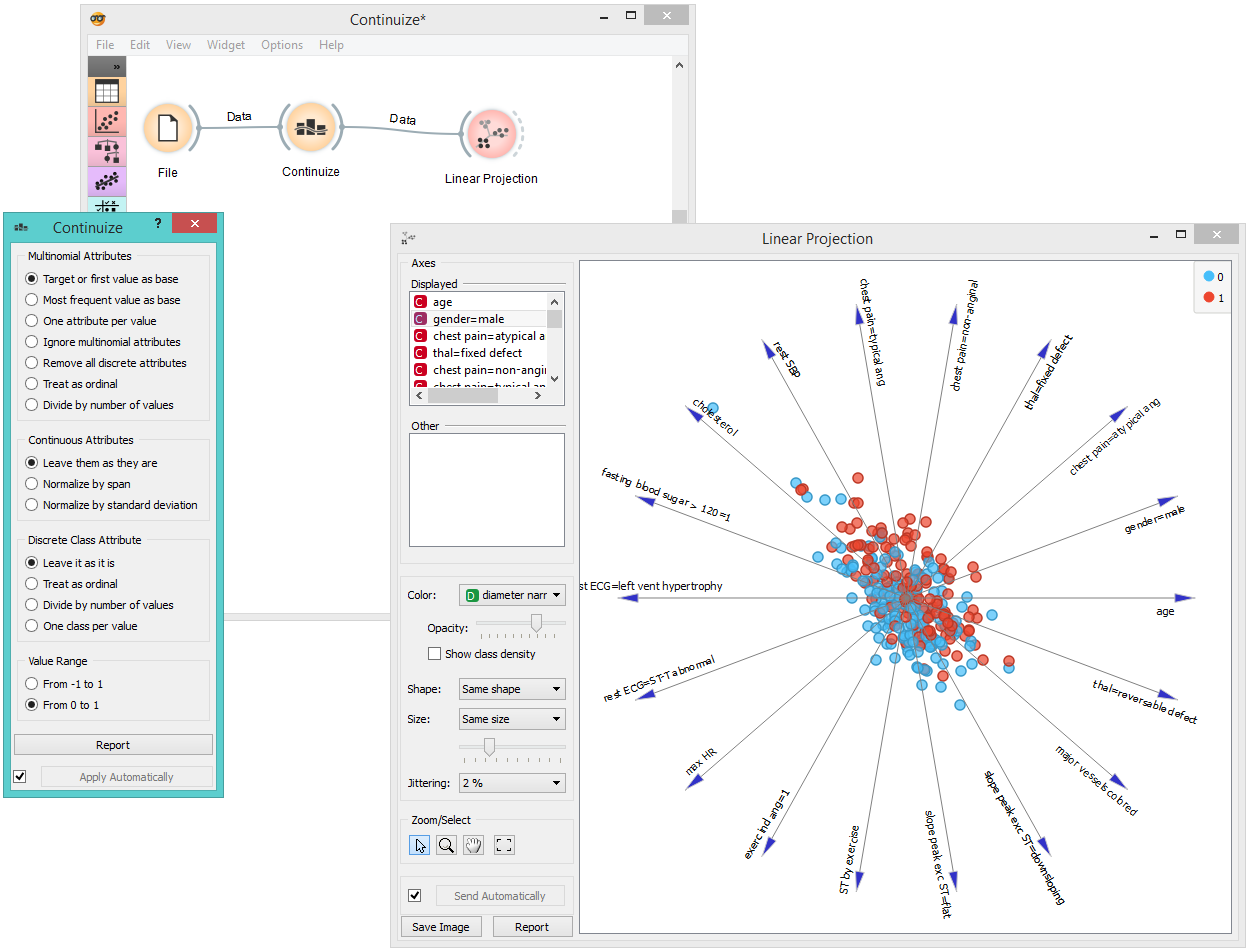
In the second example, we show a typical use of this widget - in order to properly plot the linear projection of the data, discrete attributes need to be converted to continuous ones and that is why we put the data through the **Continuize** widget before drawing it. The attribute "*chest pain*" originally had four values and was transformed into three continuous attributes; similar happened to gender, which was transformed into a single attribute "*gender=female*".

Create Class

Create class attribute from a string attribute.

Signals

Inputs:


Continuize*
File Edit View Widget Options Help
Data
Data
File
Continuize
Linear Projection
Continuize
Multinomial Attributes
Target or first value as base
Most frequent value as base
One attribute per value
Ignore multinomial attributes
Remove all discrete attributes
Treat as ordinal
Divide by number of values
Continuous Attributes
Leave them as they are
Normalize by span
Normalize by standard deviation
Discrete Class Attribute
Leave it as it is
Treat as ordinal
Divide by number of values
One class per value
Value Range
From -1 to 1
From 0 to 1
Report
Apply Automatically
Linear Projection
Axes
Displayed
age
gender=male
chest pain=atypical a
thal=fixed defect
chest pain=non-angi
Other
Color:
diameter narr
Opacity:
Show class density
Shape:
Same shape
Size:
Same size
Jittering:
2 %
Zoom/Select
Send Automatically
Save Image
Report
chest pain=typical ang
chest pain=non-anginal
thal=fixed defect
chest pain=typ ical ang
gender=male
age
thal=reversable defect
major vessels colored
slope peak exc ST=downsloping
slope peak exc ST=flat
ST by exercise
exercise ind ang=1
max HR
rest ECG=ST-T abnormal
st ECG=left vent hypertrophy
fasting blood sugar > 120=1
cholesterol
rest sbp
0
1

- **Data**

  Attribute-valued data set.

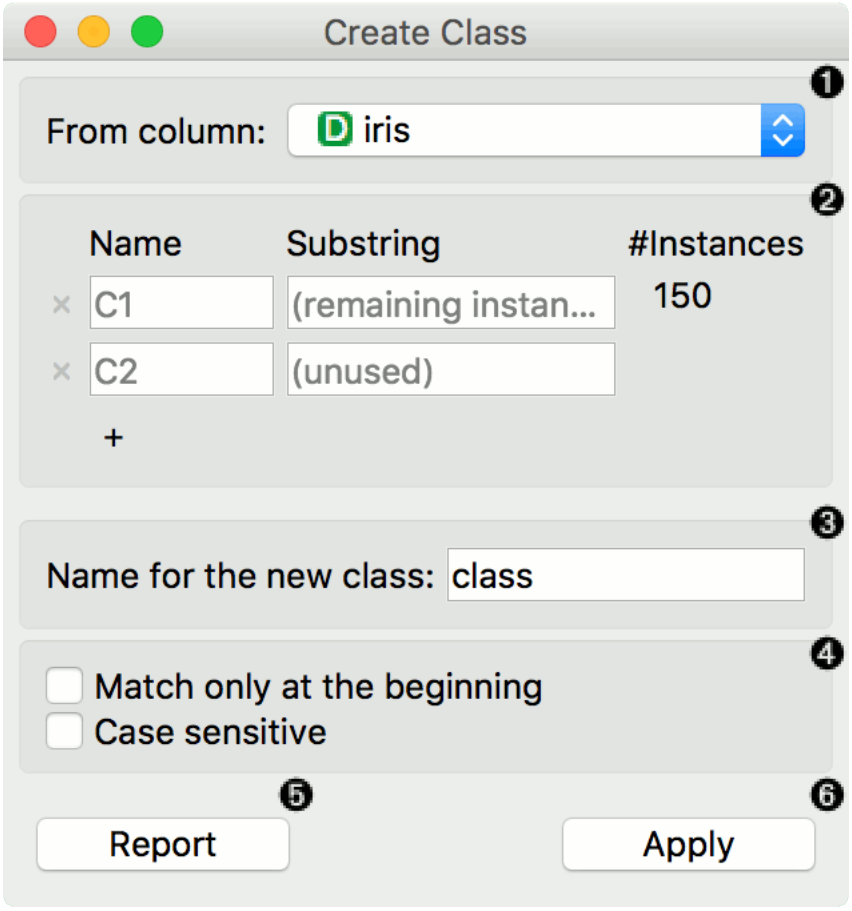
Outputs:

- **Data**

  Attribute-valued data set.

Description

Create Class creates a new class attribute from an existing discrete or string attribute. The widget matches the string value of the selected attribute and constructs a new user-defined value for matching instances.



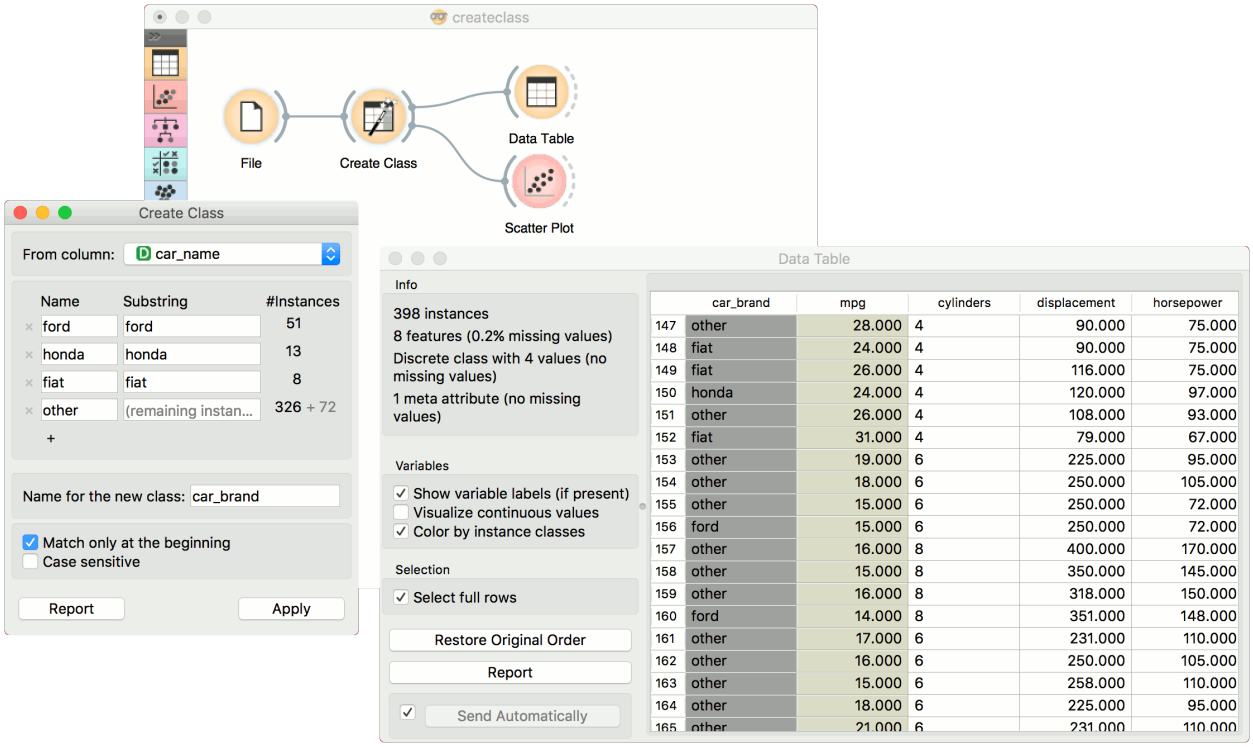
1. The attribute the new class is constructed from.
2. Matching: - Name: the name of the new class value - Substring: regex-defined substring that will match the values from the above-defined attribute - Instances: the number of instances matching the substring - Press '+' to add a new class value
3. Name of the new class column.
4. Match only at the beginning will begin matching from the beginning of the string. Case sensitive will match by case, too.
5. Produce a report.

6. Press *Apply* to commit the results.

Example

Here is a simple example with the *auto-mpg* data set. Pass the data to **Create Class**. Select *car_name* as a column to create the new class from. Here, we wish to create new values that match the car brand. First, we type *ford* as the new value for the matching strings. Then we define the substring that will match the data instances. This means that all instances containing *ford* in their *car_name*, will now have a value *ford* in the new class column. Next, we define the same for *honda* and *fiat*. The widget will tell us how many instance are yet unmatched (remaining instances). We will name them *other*, but you can continue creating new values by adding a condition with '+'.

We named our new class column *car_brand* and we matched at the beginning of the string.



Finally, we can observe the new column in a *Data Table* or use the value as color in the *Scatterplot*.

Randomize

Shuffles classes, attributes and/or metas of an input data set.

Signals

Inputs:

- **Data**

  Data set.

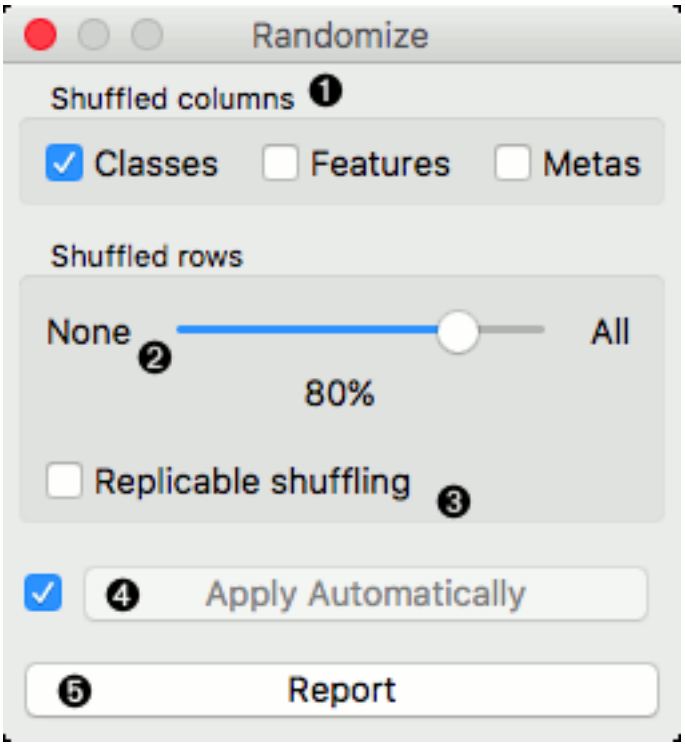
Outputs:

- **Data**

  Randomized data set.

Description

The **Randomize** widget receives a data set in the input and outputs the same data set in which the classes, attributes or/and metas are shuffled.

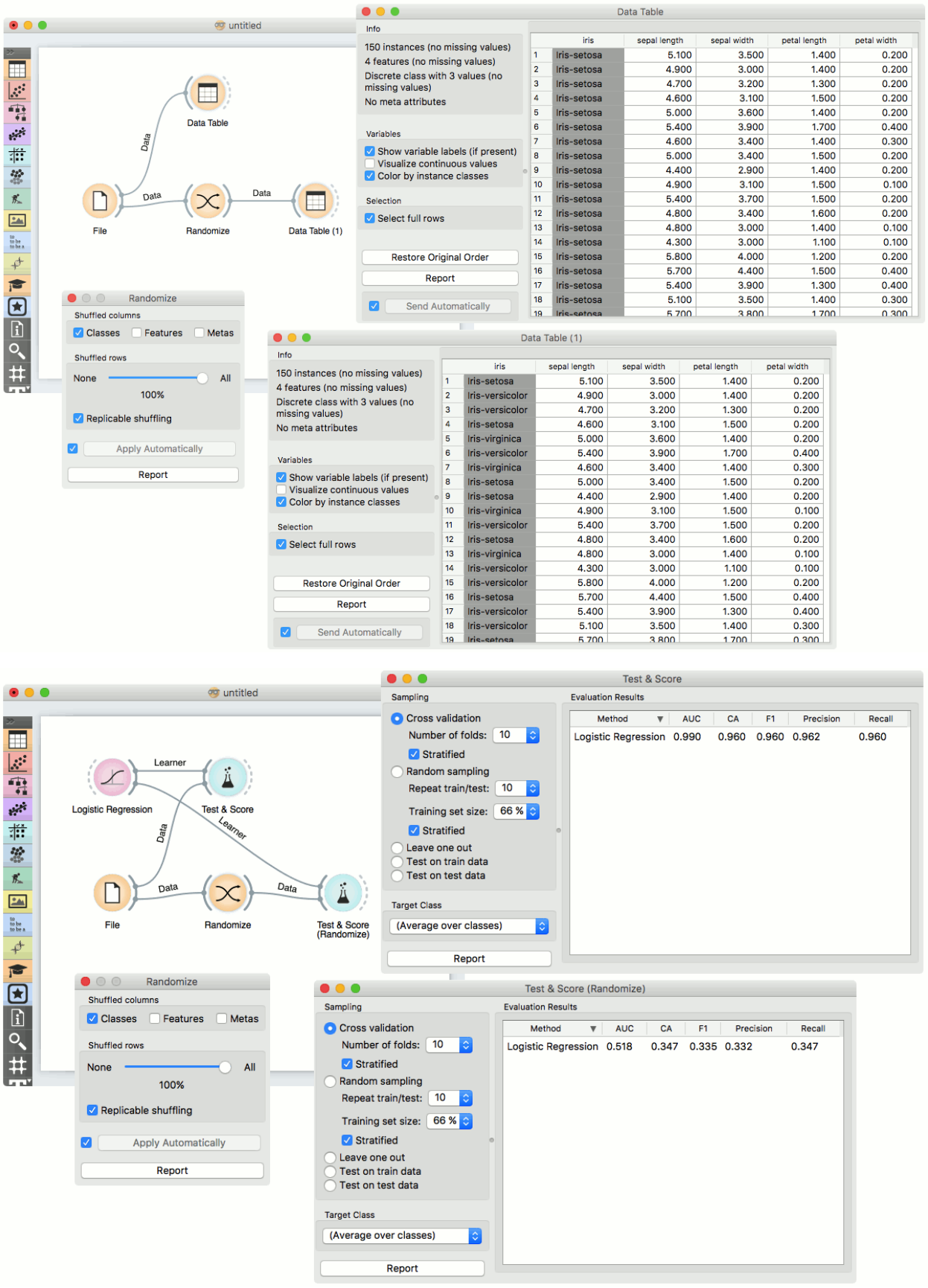


1. Select group of columns of the data set you want to shuffle.
2. Select proportion of the data set you want to shuffle.
3. Produce replicable output.
4. If *Apply automatically* is ticked, changes are committed automatically. Otherwise, you have to press *Apply* after each change.
5. Produce a report.

Example

The **Randomize** widget is usually placed right after (e.g. *File* widget. The basic usage is shown in the following workflow, where values of class variable of Iris data set are randomly shuffled.

In the next example we show how shuffling class values influences model performance on the same data set as above.


untitled
Data Table
Data
File
Randomize
Data Table (1)
Info
150 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
Randomize
Shuffled columns
Classes
Features
Metas
Shuffled rows
None
All
100%
Replicable shuffling
Apply Automatically
Learner
Logistic Regression
Test & Score
Test & Score (Randomize)
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Evaluation Results
Method AUC CA F1 Precision Recall
Logistic Regression 0.990 0.960 0.960 0.962 0.960
Logistic Regression 0.518 0.347 0.335 0.332 0.347

Concatenate

Concatenates data from multiple sources.

Signals

Inputs:

- **Primary Data**

  A data set that defines the attribute set.

- **Additional Data**

  An additional data set.

Outputs:

- **Data**

Description

The widget concatenates multiple sets of instances (data sets). The merge is "vertical", in a sense that two sets of 10 and 5 instances yield a new set of 15 instances.

1. Set the attribute merging method.
2. Add the identification of source data sets to the output data set.
3. Produce a report.
4. If *Apply automatically* is ticked, changes are communicated automatically. Otherwise, click *Apply*.

If one of the tables is connected to the widget as the primary table, the resulting table will contain its own attributes. If there is no primary table, the attributes can be either a union of all attributes that appear in the tables specified as *Additional Tables*, or their intersection, that is, a list of attributes common to all the connected tables.

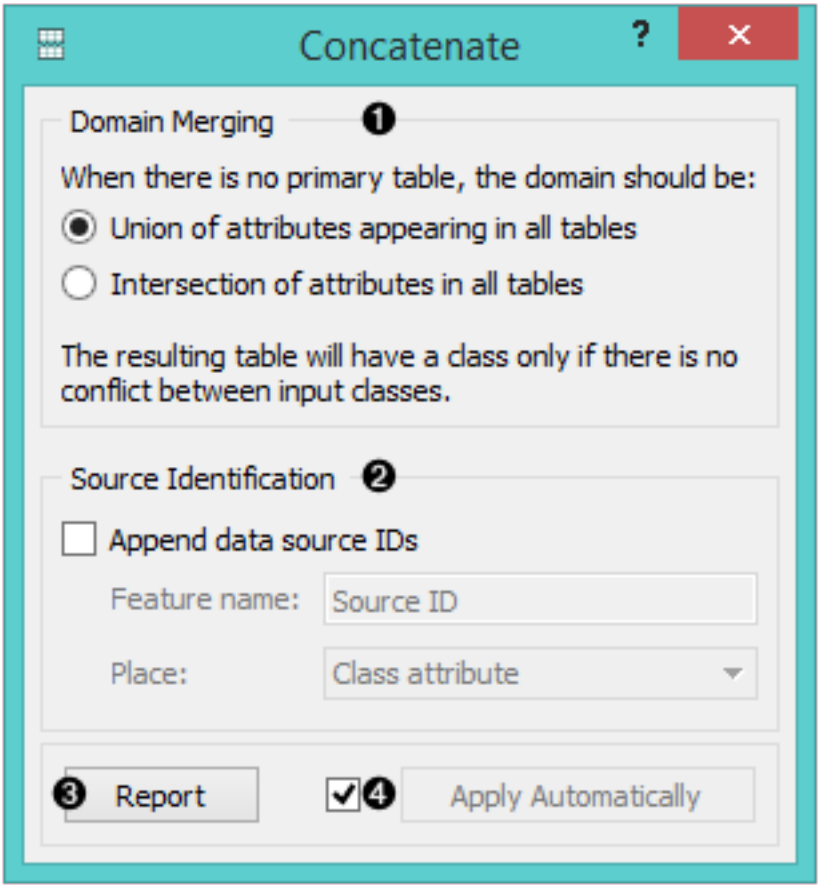
Example

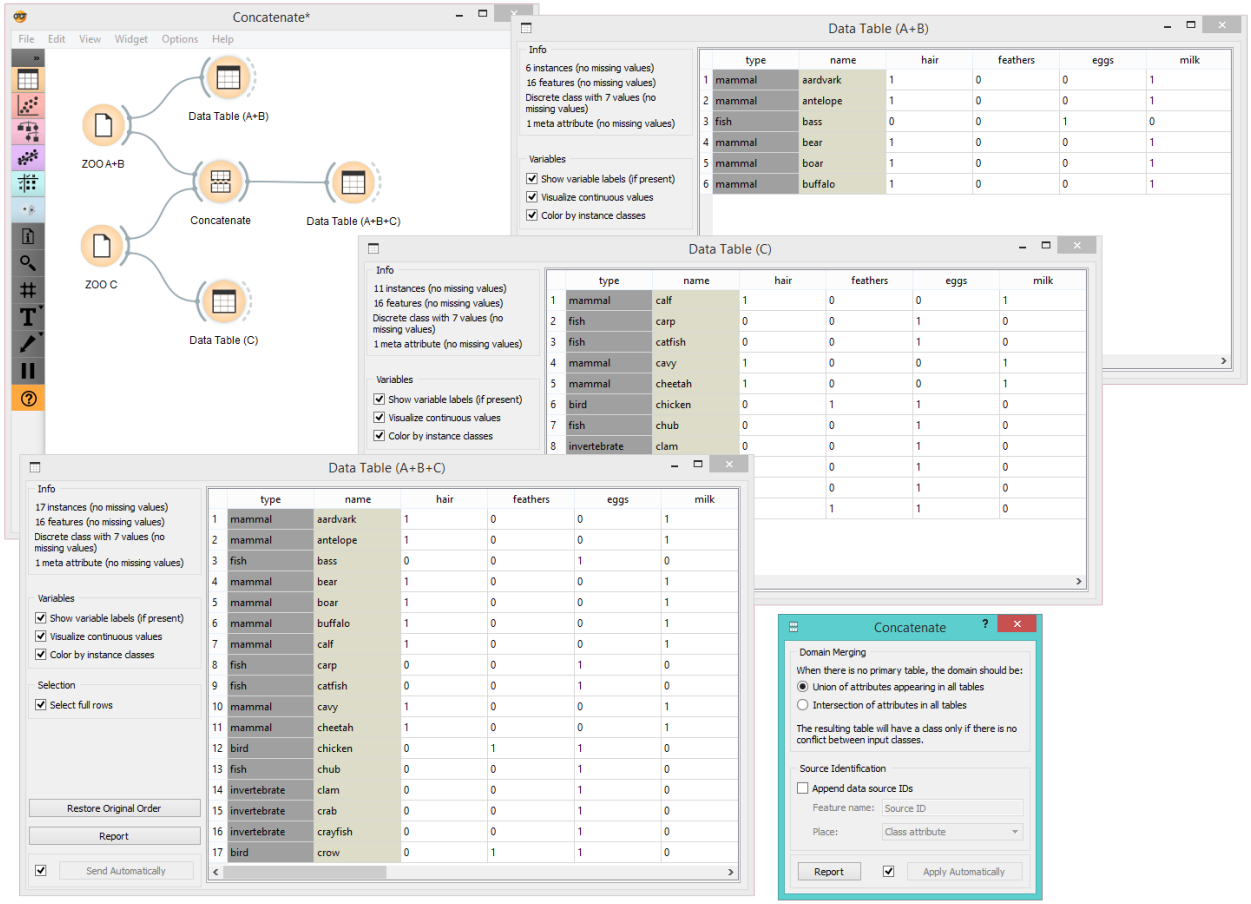
As shown below, the widget can be used for merging data from two separate files. Let's say we have two data sets with the same attributes, one containing instances from the first experiment and the other instances from the second experiment and we wish to join the two data tables together. We use the **Concatenate** widget to merge the data sets by attributes (appending new rows under existing attributes).

Below, we used a modified *Zoo* data set. In the `first` *File* widget, we loaded only the animals beginning with the letters A and B and in the `second` one only the animals beginning with the letter C. Upon concatenation, we observe the new data in the *Data Table* widget, where we see the complete table with animals from A to C.

Paint Data

Paints data on a 2D plane. You can place individual data points or use a brush to paint larger data sets.


Concatenate
Domain Merging
1
When there is no primary table, the domain should be:
Union of attributes appearing in all tables
Intersection of attributes in all tables
The resulting table will have a class only if there is no conflict between input classes.
Source Identification
2
Append data source IDs
Feature name:
Source ID
Place:
Class attribute
3
Report
4
Apply Automatically


Concatenate*
File Edit View Widget Options Help
Data Table (A+B)
ZOO A+B
Concatenate
Data Table (A+B+C)
ZOO C
Data Table (C)
Data Table (A+B)
Info
6 instances (no missing values)
16 features (no missing values)
Discrete class with 7 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
type name hair feathers eggs milk
1 mammal aardvark 1 0 0 1
2 mammal antelope 1 0 0 1
3 fish bass 0 0 1 0
4 mammal bear 1 0 0 1
5 mammal boar 1 0 0 1
6 mammal buffalo 1 0 0 1
Data Table (C)
Info
11 instances (no missing values)
16 features (no missing values)
Discrete class with 7 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
type name hair feathers eggs milk
1 mammal calf 1 0 0 1
2 fish carp 0 0 1 0
3 fish catfish 0 0 1 0
4 mammal cavy 1 0 0 1
5 mammal cheetah 1 0 0 1
6 bird chicken 0 1 1 0
7 fish chub 0 0 1 0
8 invertebrate clam 0 0 1 0
Data Table (A+B+C)
Info
17 instances (no missing values)
16 features (no missing values)
Discrete class with 7 values (no missing values)
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
type name hair feathers eggs milk
1 mammal aardvark 1 0 0 1
2 mammal antelope 1 0 0 1
3 fish bass 0 0 1 0
4 mammal bear 1 0 0 1
5 mammal boar 1 0 0 1
6 mammal buffalo 1 0 0 1
7 mammal calf 1 0 0 1
8 fish carp 0 0 1 0
9 fish catfish 0 0 1 0
10 mammal cavy 1 0 0 1
11 mammal cheetah 1 0 0 1
12 bird chicken 0 1 1 0
13 fish chub 0 0 1 0
14 invertebrate clam 0 0 1 0
15 invertebrate crab 0 0 1 0
16 invertebrate crayfish 0 0 1 0
17 bird crow 0 1 1 0
Concatenate
Domain Merging
When there is no primary table, the domain should be:
Union of attributes appearing in all tables
Intersection of attributes in all tables
The resulting table will have a class only if there is no conflict between input classes.
Source Identification
Append data source IDs
Feature name: Source ID
Place: Class attribute
Report
Apply Automatically

Signals

Inputs

- (None)

Outputs

- **Data**

  Attribute-valued data set created in the widget

Description

The widget supports the creation of a new data set by visually placing data points on a two-dimension plane. Data points can be placed on the plane individually (*Put*) or in a larger number by brushing (*Brush*). Data points can belong to classes if the data is intended to be used in supervised learning.

1. Name the axes and select a class to paint data instances. You can add or remove classes. Use only one class to create classless, unsupervised data sets.
2. Drawing tools. Paint data points with *Brush* (multiple data instances) or *Put* (individual data instance). Select data points with *Select* and remove them with the Delete/Backspace key. Reposition data points with Jitter (spread) and *Magnet* (focus). Use *Zoom* and scroll to zoom in or out. Below, set the radius and intensity for Brush, Put, Jitter and Magnet tools.
3. Reset to Input Data.
4. *Save Image* saves the image to your computer in a .svg or .png format.
5. Produce a report.
6. Tick the box on the left to automatically commit changes to other widgets. Alternatively, press *Send* to apply them.

Example

In the example below, we have painted a data set with 4 classes. Such data set is great for demonstrating k-means and hierarchical clustering methods. In the screenshot, we see that *k-means*, overall, recognizes clusters better than *hierarchical clustering*. It returns a score rank, where the best score (the one with the highest value) means the most likely number of clusters. Hierarchical clustering, however, doesn't group the right classes together. This is a great tool for learning and exploring statistical concepts.

Python Script

Extends functionalities through Python scripting.

Signals

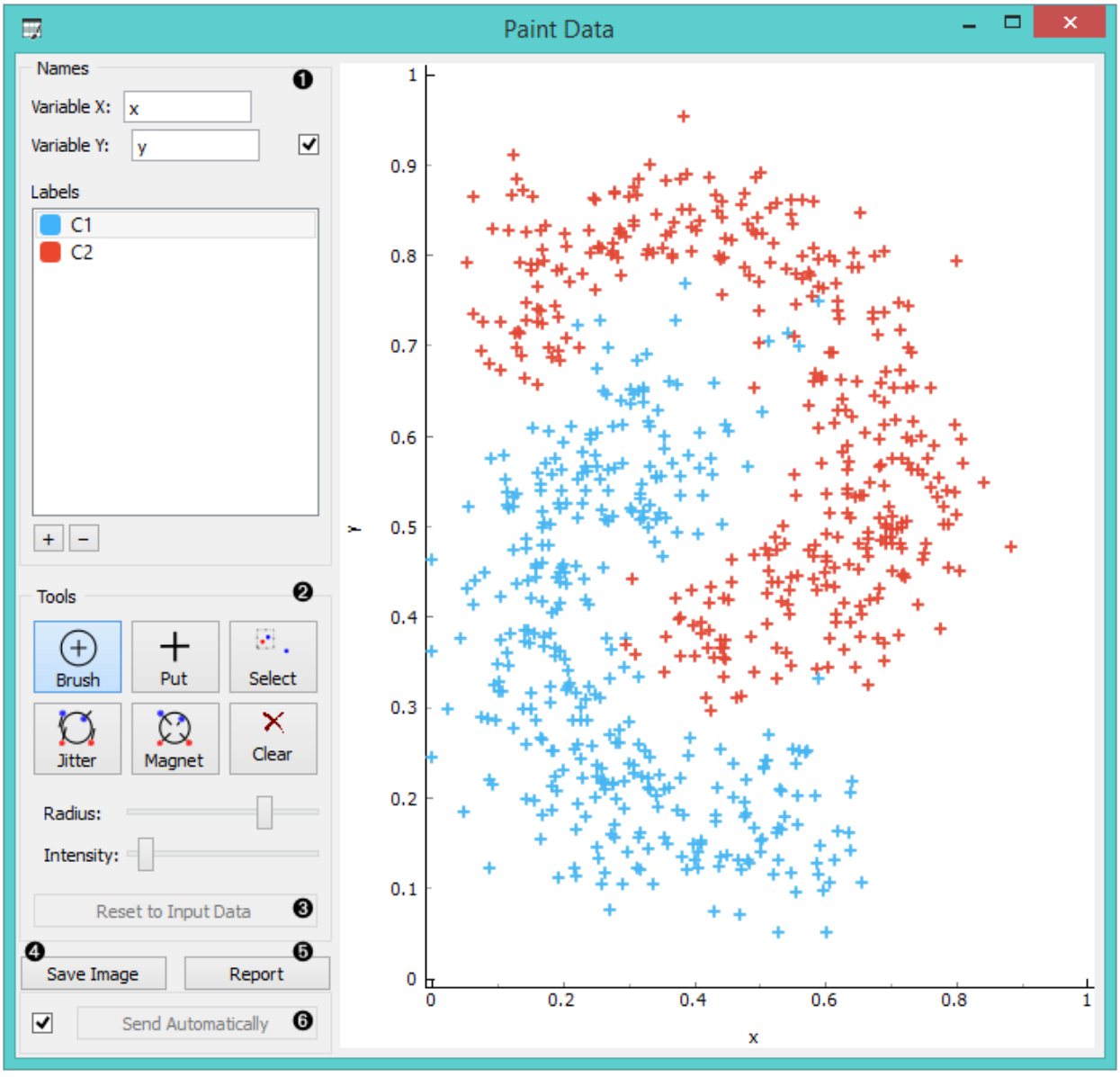
Inputs:

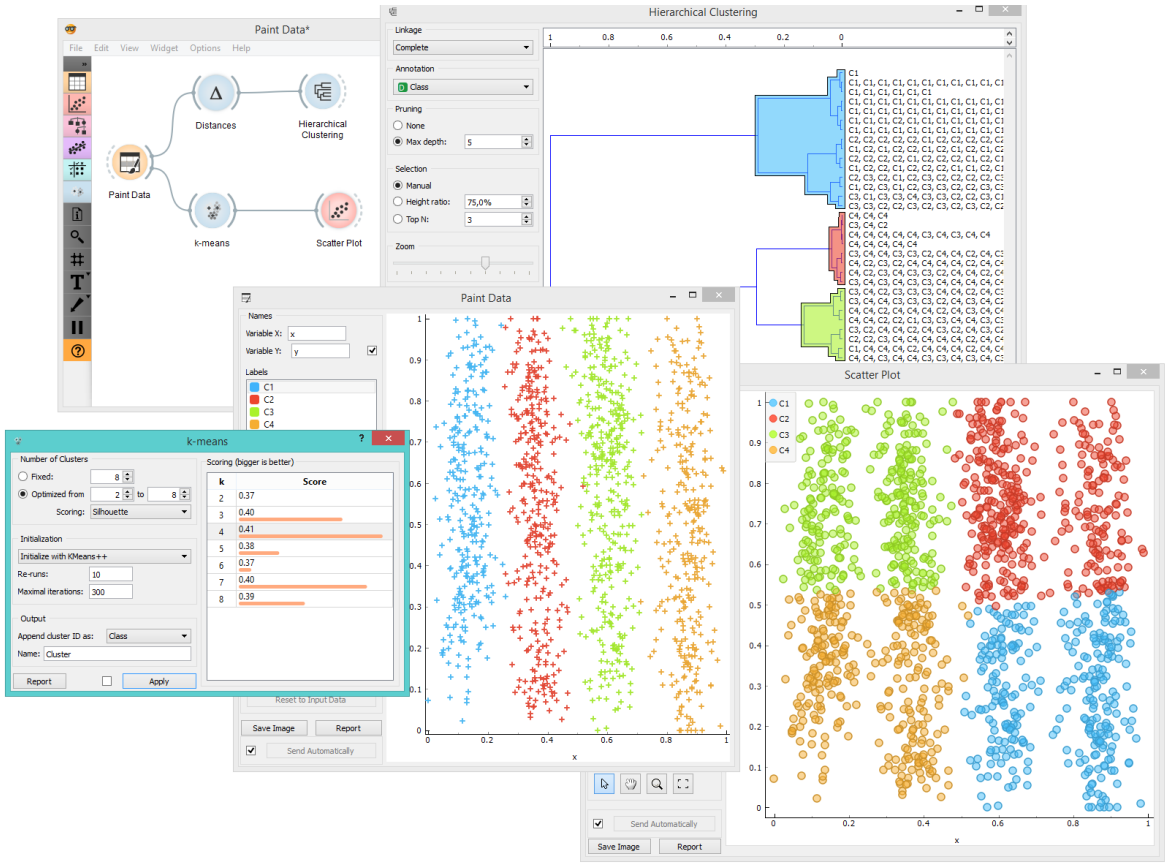
- **in_data (Orange.data.Table)**

  Input data set bound to `in_data` variable in the script's local namespace.

- **in_distance (Orange.core.SymMatrix)**

  Input symmetric matrix bound to `in_distance` variable in the script's local namespace.


Paint Data
Names
Variable X:
x
Variable Y:
y
Labels
C1
C2
Tools
Brush
Put
Select
Jitter
Magnet
Clear
Radius:
Intensity:
Reset to Input Data
Save Image
Report
Send Automatically
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y
0
0.2
0.4
0.6
0.8
1
x


Paint Data*
File Edit View Widget Options Help
Distances
Hierarchical Clustering
Paint Data
k-means
Scatter Plot
Hierarchical Clustering
Linkage
Complete
Annotation
Class
Pruning
None
Max depth:
Selection
Manual
Height ratio:
Top N:
Zoom
Paint Data
Names
Variable X:
Variable Y:
Labels
C1
C2
C3
C4
Reset to Input Data
Save Image
Report
Send Automatically
k-means
Number of Clusters
Fixed:
Optimized from
Scoring:
Silhouette
Initialization
Initialize with KMeans++
Re-runs:
Maximal iterations:
Output
Append cluster ID as:
Class
Name:
Cluster
Report
Apply
Scoring (bigger is better)
k
Score
Scatter Plot
Send Automatically
Save Image
Report

- **in_learner (Orange.classification.Learner)**

  Input learner bound to `in_learner` variable in the script's local namespace.

- **in_classifier (Orange.classification.Learner)**

  Input classifier bound to `in_classifier` variable in the script's local namespace.

- **in_object (object)**

  Input python object bound to `in_object` variable in the script's local namespace.

Outputs:

- **out_data (Orange.data.Table)**

  Data set retrieved from `out_data` variable in the script's local namespace after execution.

- **out_distance (Orange.core.SymMatrix)**

  Symmetric matrix retrieved from `out_distance` variable in the script's local namespace after execution.

- **out_learner (Orange.classification.Learner)**

  Learner retrieved from `out_learner` variable in the script's local namespace.

- **out_classifier (Orange.classification.Learner)**

  Classifier retrieved from `out_classifier` variable in the script's local namespace after execution.

- **out_object (object)**

  Python object retrieved from `out_object` variable in the script's local namespace after execution.

Description

Python Script widget can be used to run a python script in the input, when a suitable functionality is not implemented in an existing widget. The script has `in_data`, `in_distance`, `in_learner`, `in_classifier` and `in_object` variables (from input signals) in its local namespace. If a signal is not connected or it did not yet receive any data, those variables contain `None`.

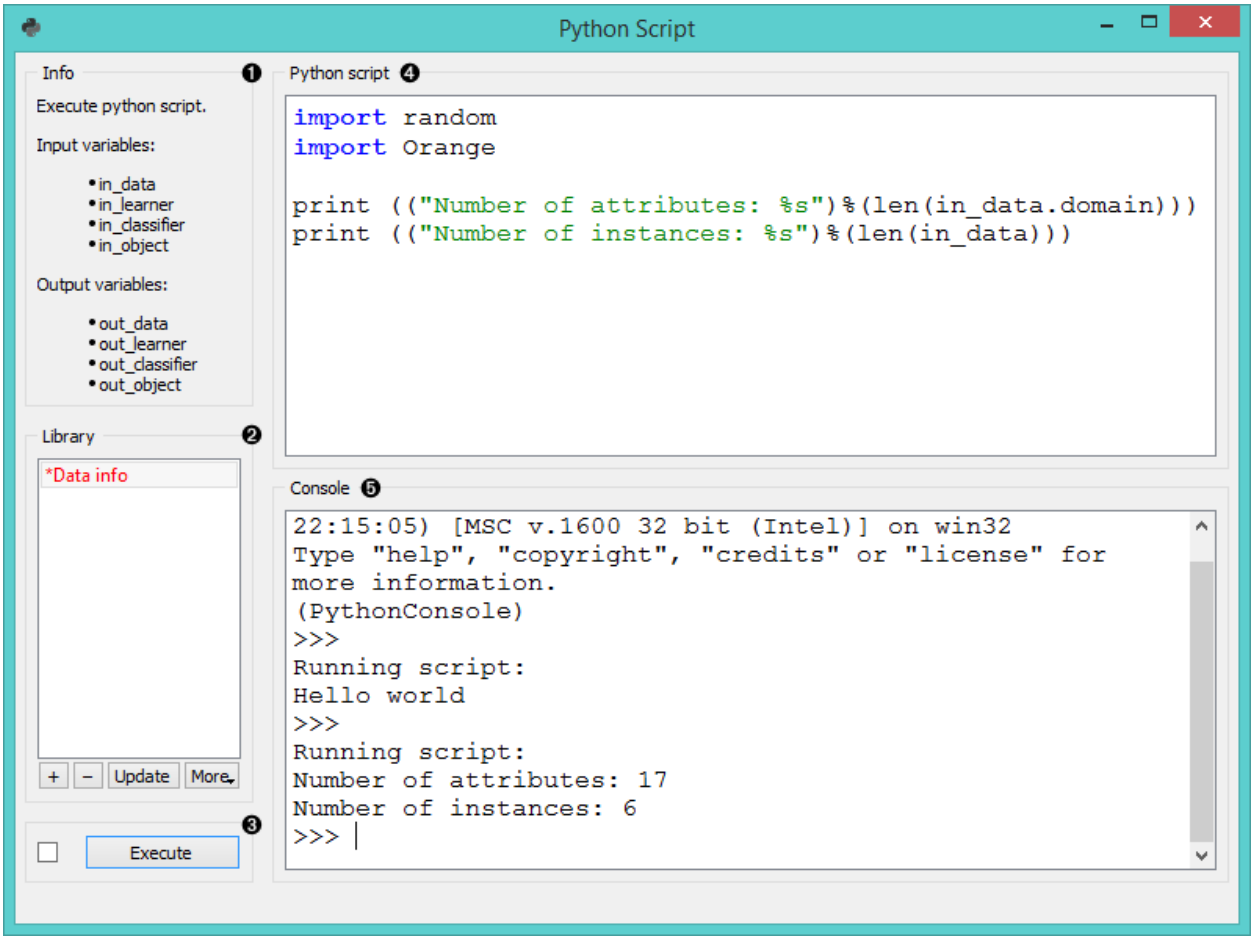
After the script is executed, `out_data`, `out_distance`, ... variables from the script's local namespace are extracted and used as outputs of the widget. The widget can be further connected to other widgets for visualizing the output.

For instance the following script would simply pass on all signals it receives:

```
out_data = in_data
out_distance = in_distance
out_learner = in_learner
out_classifier = in_classifier
out_object = in_object
```

Note: You should not modify the input objects in place.

1. Info box contains names of basic operators for Orange Python script.
2. The *Library* control can be used to manage multiple scripts. Pressing "+" will add a new entry and open it in the *Python script* editor. When the script is modified, its entry in the *Library* will change to indicate it has unsaved changes. Pressing *Update* will save the script (keyboard shortcut ctrl + s). A script can be removed by selecting it and pressing the "-" button.


Python Script
Info
Execute python script.
Input variables:
in_data
in_learner
in_classifier
in_object
Output variables:
out_data
out_learner
out_classifier
out_object
Library
*Data info
+
−
Update
More
Execute
Python script
import random
import Orange
print (("Number of attributes: %s")%(len(in_data.domain)))
print (("Number of instances: %s")%(len(in_data)))
Console
22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for
more information.
(PythonConsole)
>>>
Running script:
Hello world
>>>
Running script:
Number of attributes: 17
Number of instances: 6
>>>

3. Pressing *Execute* in the *Run* box executes the script (using `exec`). Any script output (from `print`) is captured and displayed in the *Console* below the script. If *Auto execute* is checked, the script is run any time inputs to the widget change.
4. The *Python script* editor on the left can be used to edit a script (it supports some rudimentary syntax highlighting).
5. Console displays the output of the script.

Examples

Python Script widget is intended to extend functionalities for advanced users.

One can, for example, do batch filtering by attributes. We used zoo.tab for the example and we filtered out all the attributes that have more than 5 discrete values. This in our case removed only 'leg' attribute, but imagine an example where one would have many such attributes.

```
from Orange.data import Domain, Table
domain = Domain([attr for attr in in_data.domain.attributes
                 if attr.is_continuous or len(attr.values) <= 5],
                in_data.domain.class_vars)
out_data = Table(domain, in_data)
```

The second example shows how to round all the values in a few lines of code. This time we used wine.tab and rounded all the values to whole numbers.

```
import numpy as np
out_data = in_data.copy()
#copy, otherwise input data will be overwritten
np.round(out_data.X, 0, out_data.X)
```

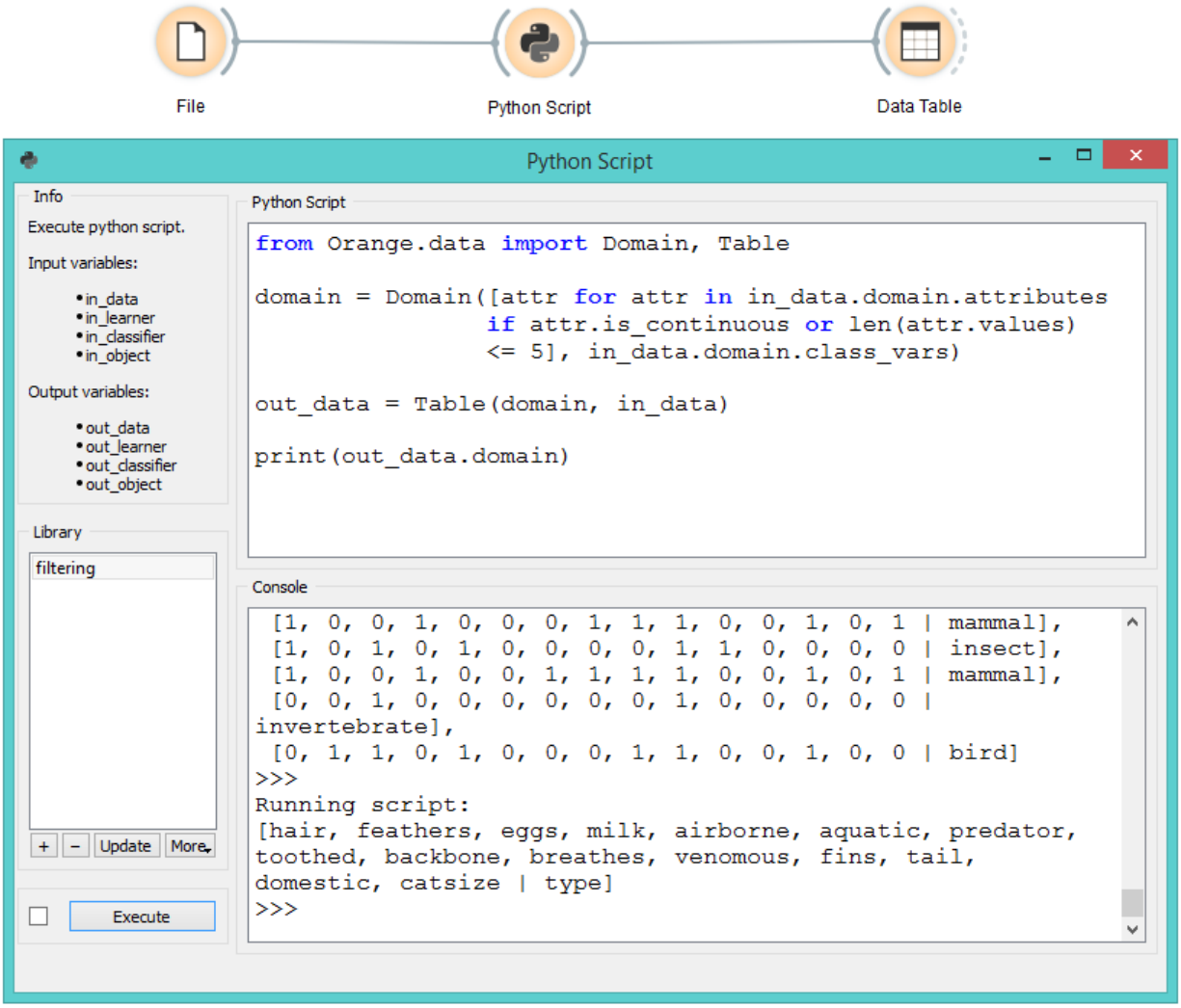
The third example introduces some gaussian noise to the data. Again we make a copy of the input data, then walk through all the values with a double for loop and add random noise.

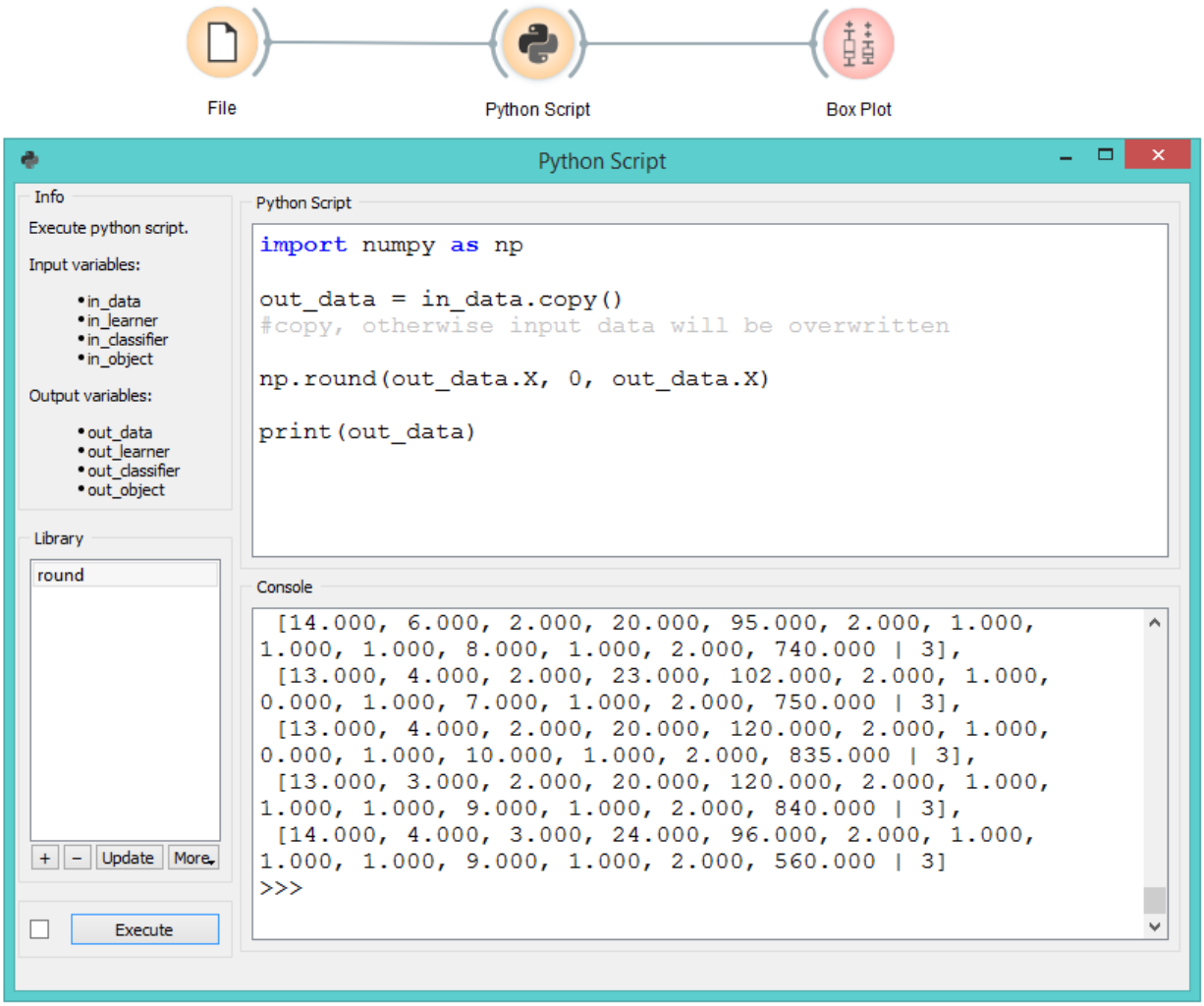
```
import random
from Orange.data import Domain, Table
new_data = in_data.copy()
for inst in new_data:
  for f in inst.domain.attributes:
    inst[f] += random.gauss(0, 0.02)
out_data = new_data
```

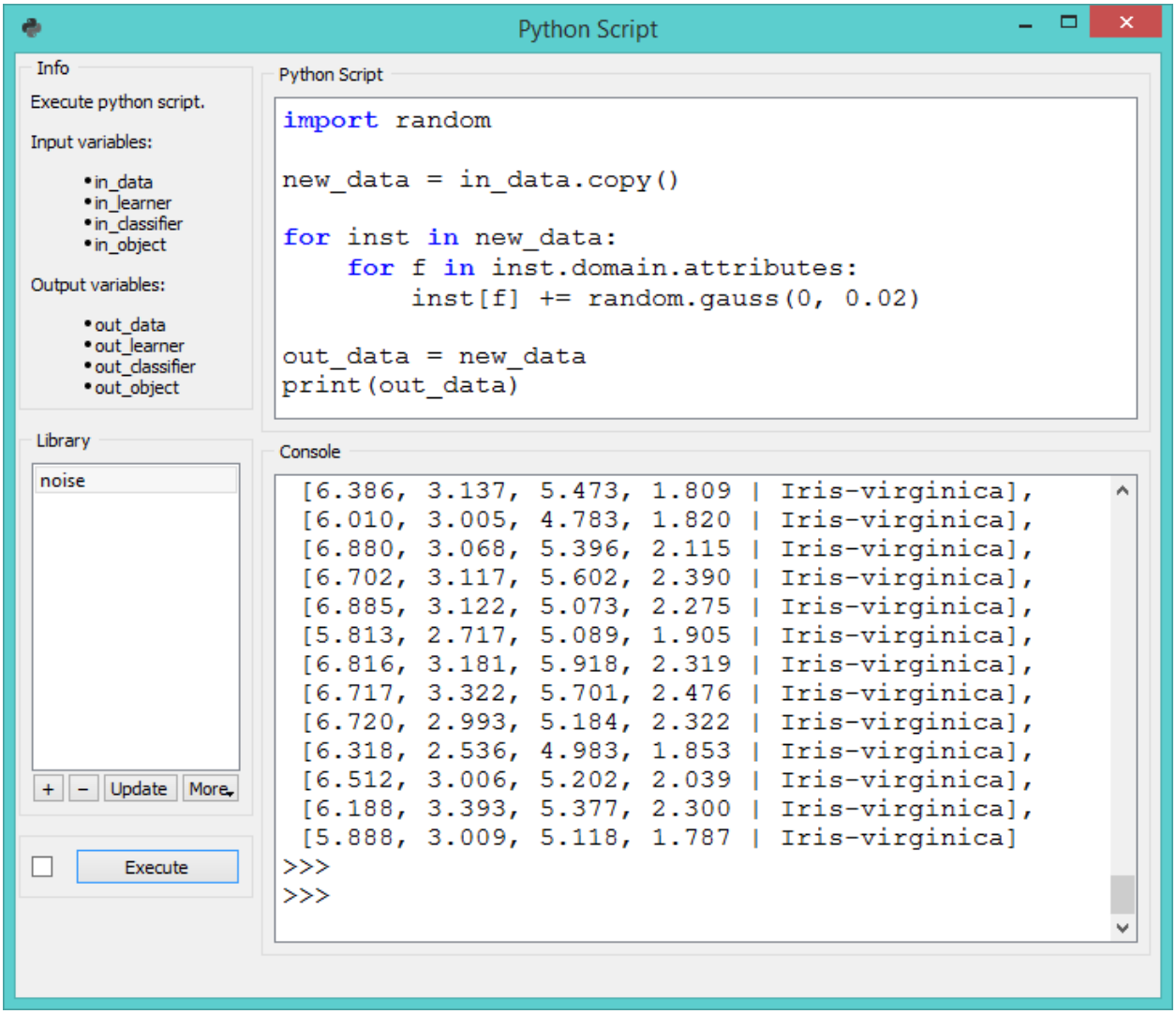
The final example uses Orange3-Text add-on. **Python Script** is very useful for custom preprocessing in text mining, extracting new features from strings, or utilizing advanced nltk or gensim functions. Below, we simply tokenized our input data from deerwester.tab by splitting them by whitespace.

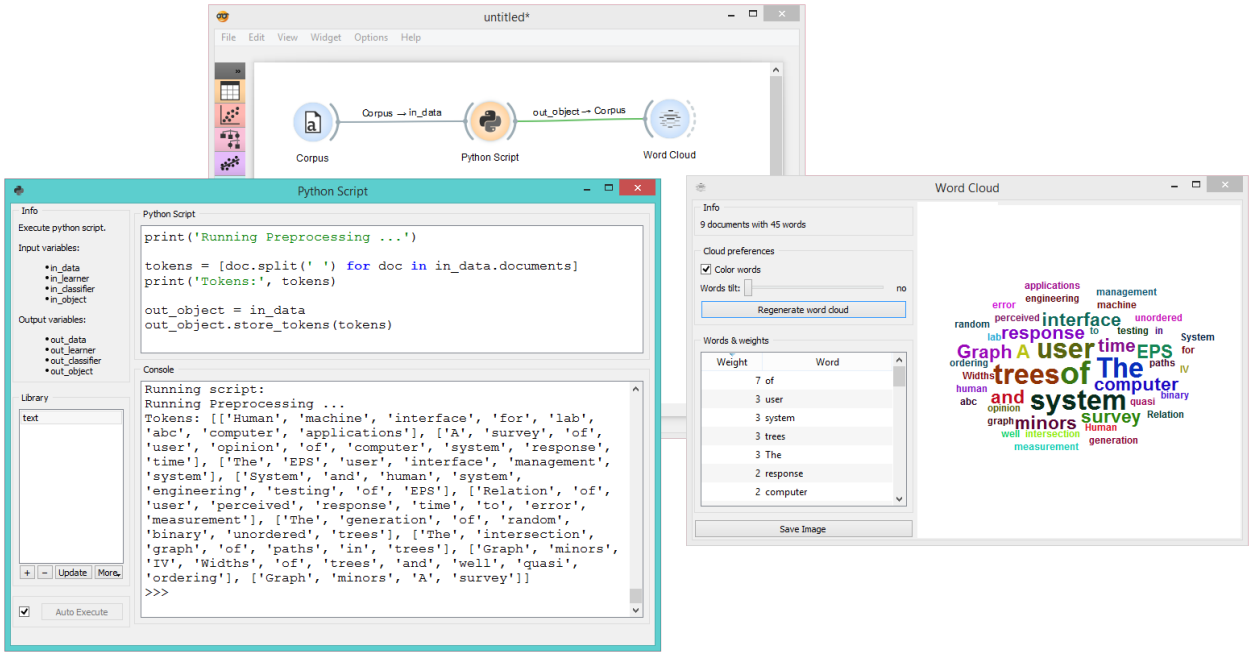
```
print('Running Preprocessing ...')
tokens = [doc.split(' ') for doc in in_data.documents]
print('Tokens:', tokens)
out_object = in_data
out_object.store_tokens(tokens)
```

You can add a lot of other preprocessing steps to further adjust the output. The output of **Python Script** can be used with any widget that accepts the type of output your script produces. In this case, connection is green, which signalizes the right type of input for Word Cloud widget.


File
Python Script
Data Table
Python Script
Info
Execute python script.
Input variables:
• in_data
• in_learner
• in_classifier
• in_object
Output variables:
• out_data
• out_learner
• out_classifier
• out_object
Library
filtering
+
–
Update
More
Execute
Python Script
from Orange.data import Domain, Table
domain = Domain([attr for attr in in_data.domain.attributes
if attr.is_continuous or len(attr.values)
<= 5], in_data.domain.class_vars)
out_data = Table(domain, in_data)
print(out_data.domain)
Console
[1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1 | mammal],
[1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0 | insect],
[1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1 | mammal],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 |
invertebrate],
[0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0 | bird]
>>>
Running script:
[hair, feathers, eggs, milk, airborne, aquatic, predator,
toothed, backbone, breathes, venomous, fins, tail,
domestic, catsize | type]
>>>


File
Python Script
Box Plot
Python Script
Info
Execute python script.
Input variables:
in_data
in_learner
in_classifier
in_object
Output variables:
out_data
out_learner
out_classifier
out_object
Library
round
+
−
Update
More
Execute
Python Script
import numpy as np
out_data = in_data.copy()
#copy, otherwise input data will be overwritten
np.round(out_data.X, 0, out_data.X)
print(out_data)
Console
[14.000, 6.000, 2.000, 20.000, 95.000, 2.000, 1.000,
1.000, 1.000, 8.000, 1.000, 2.000, 740.000 | 3],
[13.000, 4.000, 2.000, 23.000, 102.000, 2.000, 1.000,
0.000, 1.000, 7.000, 1.000, 2.000, 750.000 | 3],
[13.000, 4.000, 2.000, 20.000, 120.000, 2.000, 1.000,
0.000, 1.000, 10.000, 1.000, 2.000, 835.000 | 3],
[13.000, 3.000, 2.000, 20.000, 120.000, 2.000, 1.000,
1.000, 1.000, 9.000, 1.000, 2.000, 840.000 | 3],
[14.000, 4.000, 3.000, 24.000, 96.000, 2.000, 1.000,
1.000, 1.000, 9.000, 1.000, 2.000, 560.000 | 3]
>>>


Python Script
Info
Execute python script.
Input variables:
• in_data
• in_learner
• in_classifier
• in_object
Output variables:
• out_data
• out_learner
• out_classifier
• out_object
Library
noise
+
–
Update
More
Execute
Python Script
import random
new_data = in_data.copy()
for inst in new_data:
for f in inst.domain.attributes:
inst[f] += random.gauss(0, 0.02)
out_data = new_data
print(out_data)
Console
[6.386, 3.137, 5.473, 1.809 | Iris-virginica],
[6.010, 3.005, 4.783, 1.820 | Iris-virginica],
[6.880, 3.068, 5.396, 2.115 | Iris-virginica],
[6.702, 3.117, 5.602, 2.390 | Iris-virginica],
[6.885, 3.122, 5.073, 2.275 | Iris-virginica],
[5.813, 2.717, 5.089, 1.905 | Iris-virginica],
[6.816, 3.181, 5.918, 2.319 | Iris-virginica],
[6.717, 3.322, 5.701, 2.476 | Iris-virginica],
[6.720, 2.993, 5.184, 2.322 | Iris-virginica],
[6.318, 2.536, 4.983, 1.853 | Iris-virginica],
[6.512, 3.006, 5.202, 2.039 | Iris-virginica],
[6.188, 3.393, 5.377, 2.300 | Iris-virginica],
[5.888, 3.009, 5.118, 1.787 | Iris-virginica]
>>>
>>>



Feature Constructor

Add new features to your data set.

Signals

Inputs:

- **Data**

A data set

Outputs:

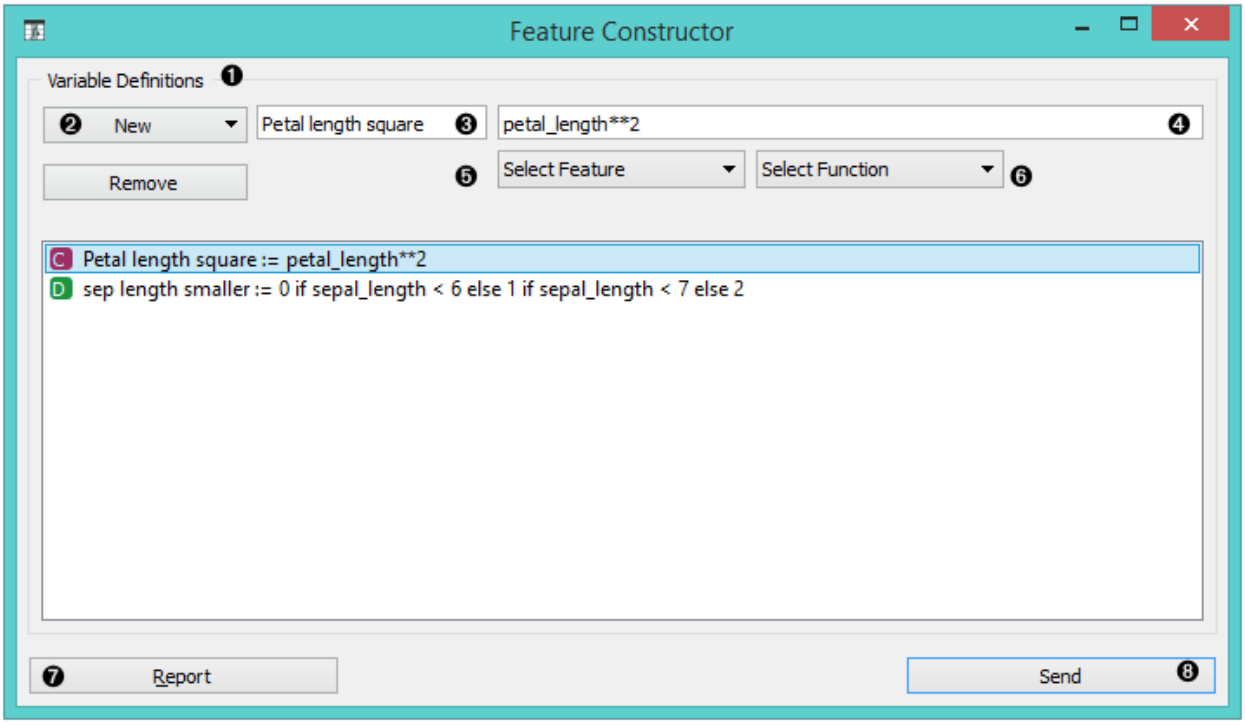
- **Data**

A modified data set

Description

The **Feature Constructor** allows you to manually add features (columns) into your data set. The new feature can be a computation of an existing one or a combination of several (addition, subtraction, etc.). You can choose what type of feature it will be (discrete, continuous or string) and what its parameters are (name, value, expression). For continuous variables you only have to construct an expression in Python.

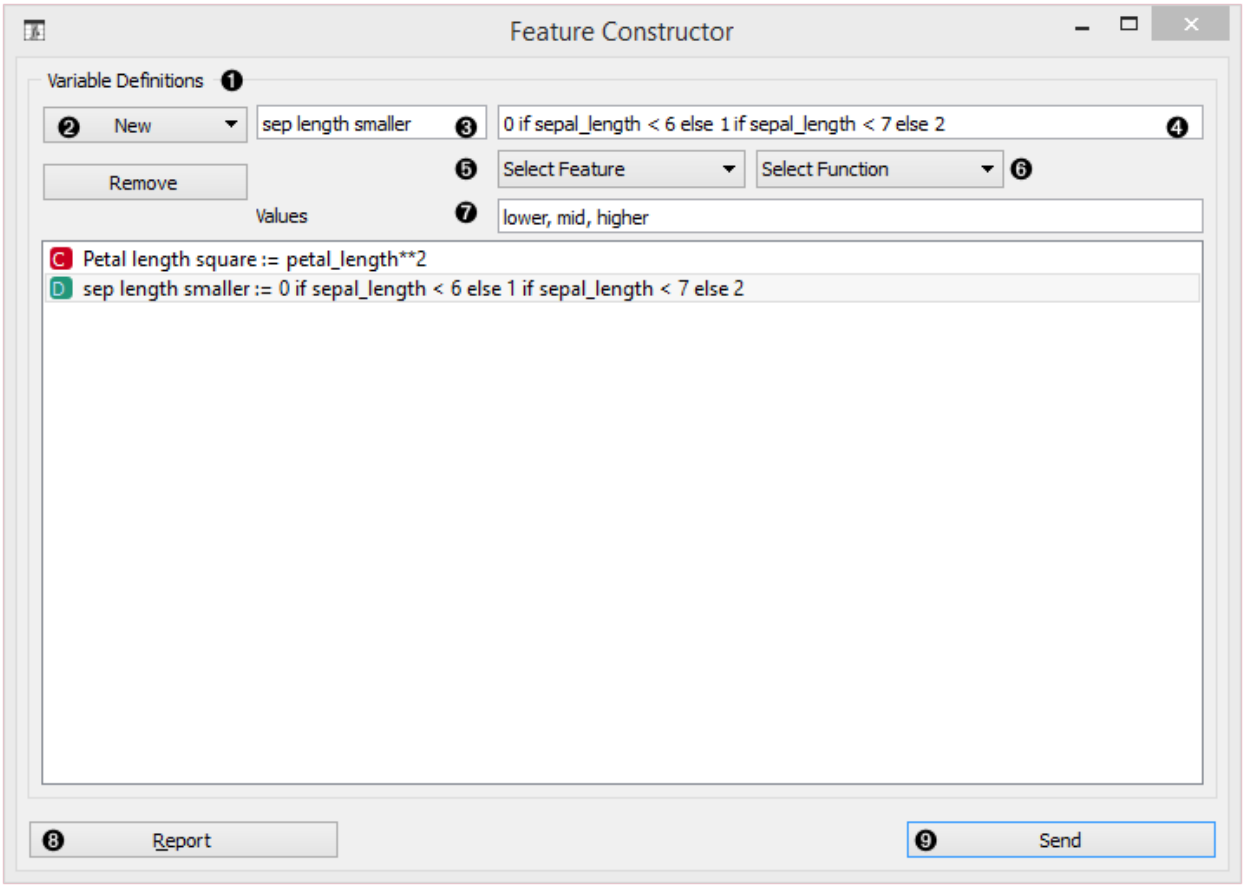
1. List of constructed variables
2. Add or remove variables.
3. New feature name



4. Expression in Python
5. Select a feature.
6. Select a function.
7. Produce a report.
8. Press *Send* to communicate changes.

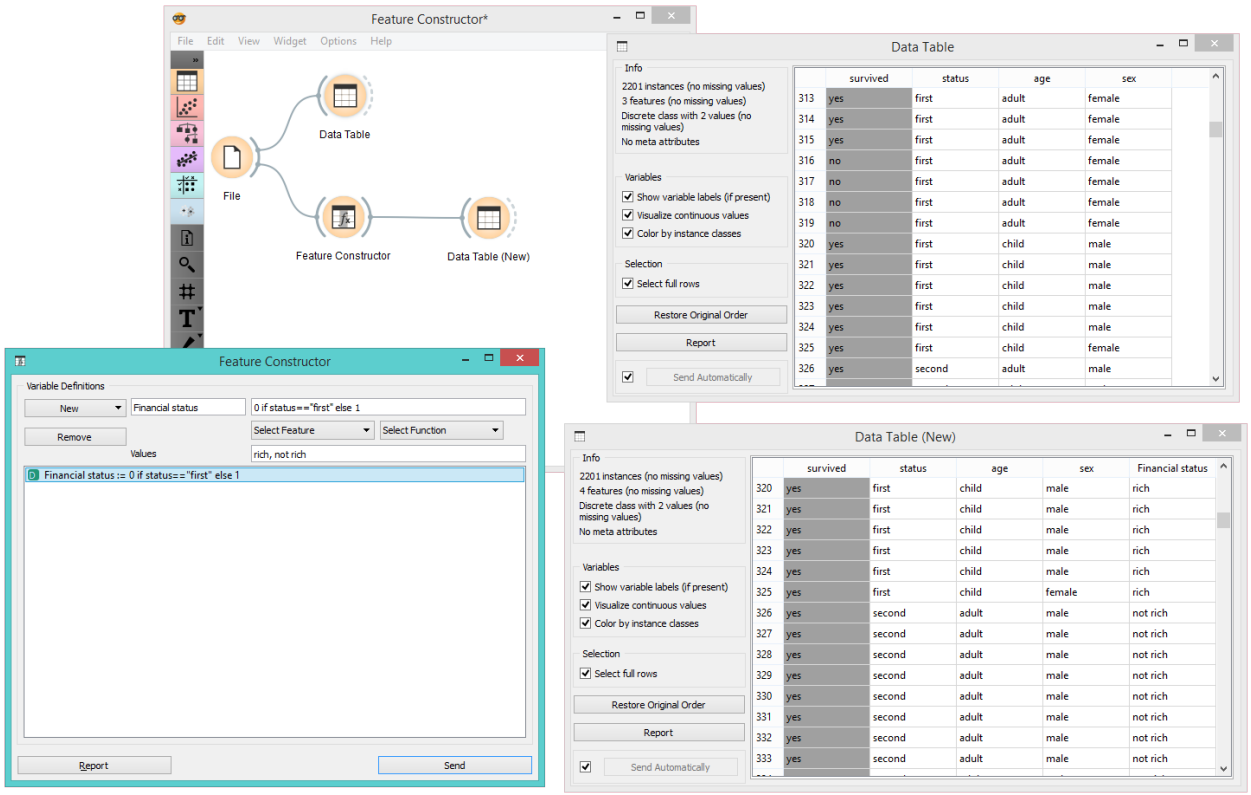
For discrete variables, however, there's a bit more work. First add or remove the values you want for the new feature. Then select the base value and the expression. In the example below, we have constructed an expression with 'if lower than' and defined three conditions; the program ascribes 0 (which we renamed to lower) if the original value is lower than 6, 1 (mid) if it is lower than 7 and 2 (higher) for all the other values. Notice that we use an underscore for the feature name (e.g. petal_length).

1. List of variable definitions
2. Add or remove variables
3. New feature name
4. Expression in Python
5. Select a feature.
6. Select a function.
7. Assign values.
8. Produce a report.
9. Press *Send* to communicate changes.


Feature Constructor
Variable Definitions
1
2
New
sep length smaller
3
0 if sepal_length < 6 else 1 if sepal_length < 7 else 2
4
Remove
5
Select Feature
Select Function
6
Values
7
lower, mid, higher
C
Petal length square := petal_length**2
D
sep length smaller := 0 if sepal_length < 6 else 1 if sepal_length < 7 else 2
8
Report
9
Send

Example

With the **Feature Constructor** you can easily adjust or combine existing features into new ones. Below, we added one new discrete feature to the *Titanic* data set. We created a new attribute called *Financial status* and set the values to be *rich* if the person belongs to the first class (status = first) and *not rich* for everybody else. We can see the new data set with *Data Table* widget.



Hints

If you are unfamiliar with Python math language, here's a quick introduction.

- +, - to add, subtract
- * to multiply
- / to divide
- % to divide and return the remainder
- ** for exponent (for square root square by 0.5)
- // for floor division
- <, >, <=, >= less than, greater than, less or equal, greater or equal
- == for equal
- != for not equal

As in the example: (*value*) if (*feature name*) < (*value*), else (*value*) if (*feature name*) < (*value*), else (*value*)

[Use value 1 if feature is less than specified value, else use value 2 if feature is less than specified value 2, else use value 3.]

See more here.

Edit Domain

Signals

Inputs:

- **Data**

  An input data set

Outputs:

- **Data**

  An edited output data set

Description

This widget can be used to edit/change a data set's domain.

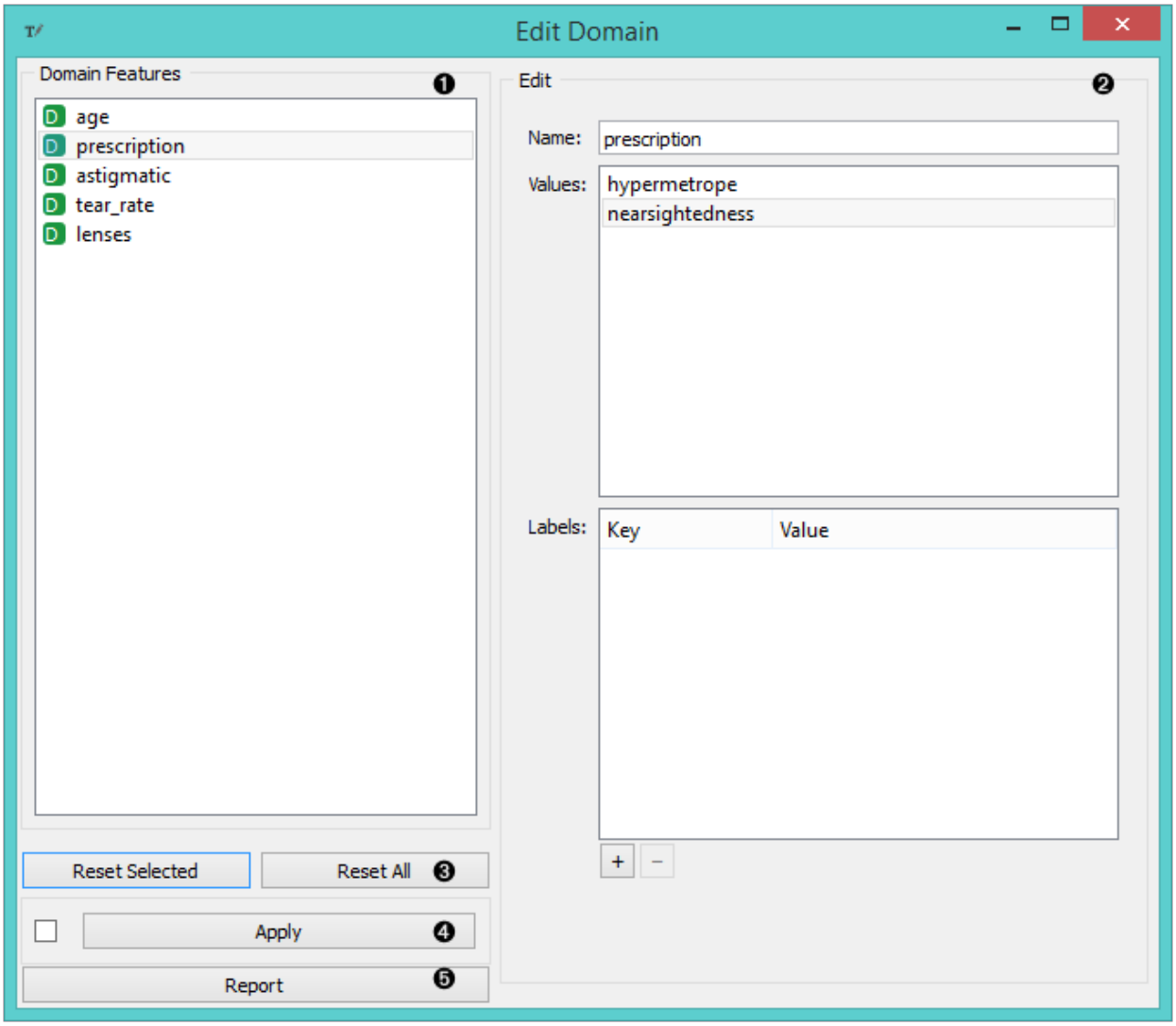
1. All features (including meta attributes) from the input data set are listed in the *Domain Features* list in the box on the left. Selecting one feature displays an editor on the right.
2. The name of the feature can be changed in the *Name* line edit. For *Discrete* features, value names can also be changed in the *Values* list box. Additonal feature annotations can be added/removed/edited in the *Labels* box. To add a new label, click the "+" button and add the *Key* and *Value* columns for the new entry. Selecting an existing label and pressing "-" will remove the annotation.
3. To revert the changes made to the feature, press the *Reset Selected* button in the *Reset* box while the feature is selected in the *Domain Features* list. Pressing *Reset All* will reset all features in the domain at the same time.
4. Pressing the *Apply* button will send the changed domain data set to the output channel.

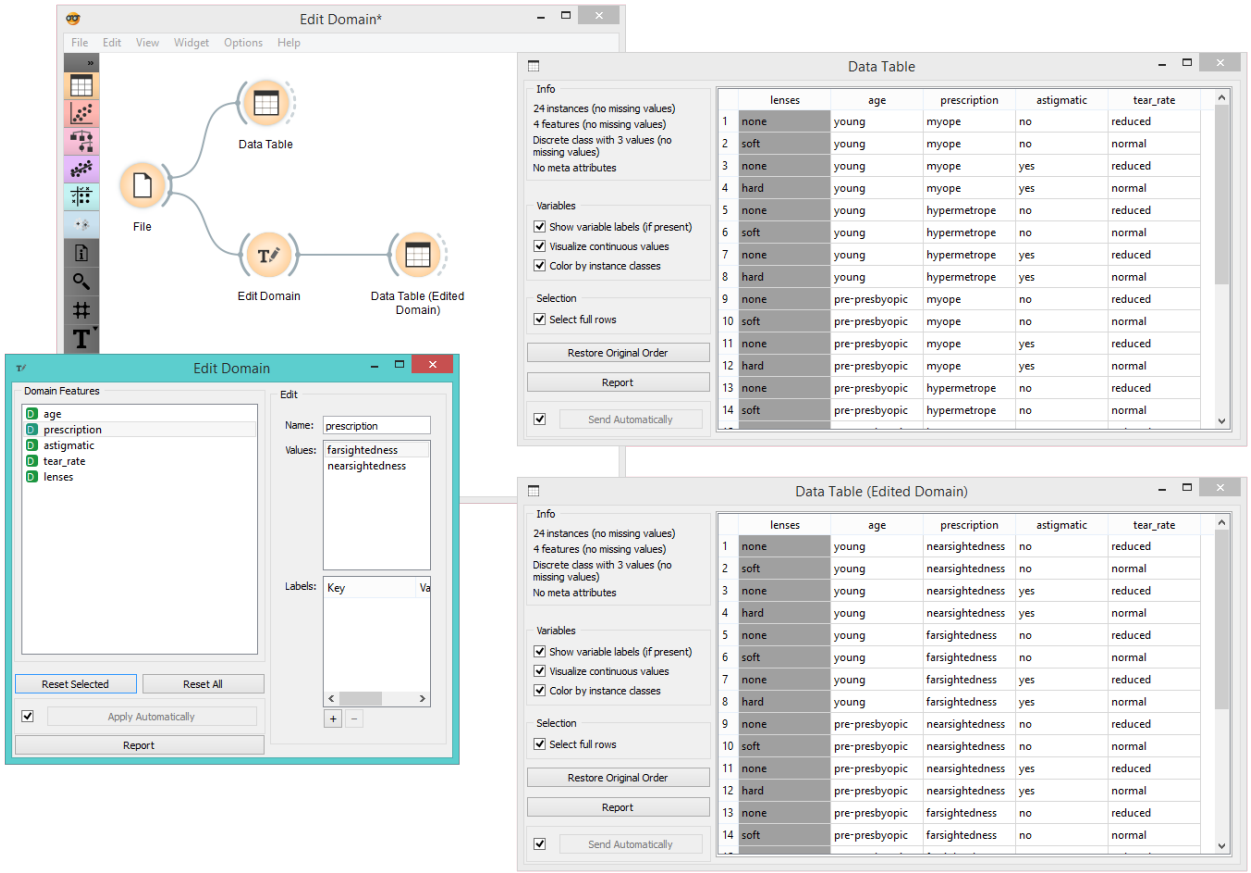
Example

Below, we demonstrate how to simply edit an existing domain. We selected the *lenses.tab* data set and edited the *perscription* attribute. Where in the original we had the values *myope* and *hypermetrope*, we changed it into *nearsightedness* and *farsightedness* instead. For an easier comparison, we fed both the original and edited data into the *Data Table* widget.

Image Viewer

Displays images that come with a data set.


Edit Domain
Domain Features
1
age
prescription
astigmatic
tear_rate
lenses
Edit
2
Name:
prescription
Values:
hypermetrope
nearsightedness
Labels:
Key
Value
+
−
Reset Selected
Reset All
3
Apply
4
Report
5


Edit Domain*
File Edit View Widget Options Help
Data Table
File
Edit Domain
Data Table (Edited Domain)
Edit Domain
Domain Features
age
prescription
astigmatic
tear_rate
lenses
Edit
Name: prescription
Values: farsightedness nearsightedness
Labels: Key
Reset Selected
Reset All
Apply Automatically
Report
Data Table
Info
24 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
lenses age prescription astigmatic tear_rate
1 none young myope no reduced
2 soft young myope no normal
3 none young myope yes reduced
4 hard young myope yes normal
5 none young hypermetrope no reduced
6 soft young hypermetrope no normal
7 none young hypermetrope yes reduced
8 hard young hypermetrope yes normal
9 none pre-presbyopic myope no reduced
10 soft pre-presbyopic myope no normal
11 none pre-presbyopic myope yes reduced
12 hard pre-presbyopic myope yes normal
13 none pre-presbyopic hypermetrope no reduced
14 soft pre-presbyopic hypermetrope no normal
Data Table (Edited Domain)
Info
24 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
lenses age prescription astigmatic tear_rate
1 none young nearsightedness no reduced
2 soft young nearsightedness no normal
3 none young nearsightedness yes reduced
4 hard young nearsightedness yes normal
5 none young farsightedness no reduced
6 soft young farsightedness no normal
7 none young farsightedness yes reduced
8 hard young farsightedness yes normal
9 none pre-presbyopic nearsightedness no reduced
10 soft pre-presbyopic nearsightedness no normal
11 none pre-presbyopic nearsightedness yes reduced
12 hard pre-presbyopic nearsightedness yes normal
13 none pre-presbyopic farsightedness no reduced
14 soft pre-presbyopic farsightedness no normal

Signals

Inputs:

- **Data**

  A data set with images.

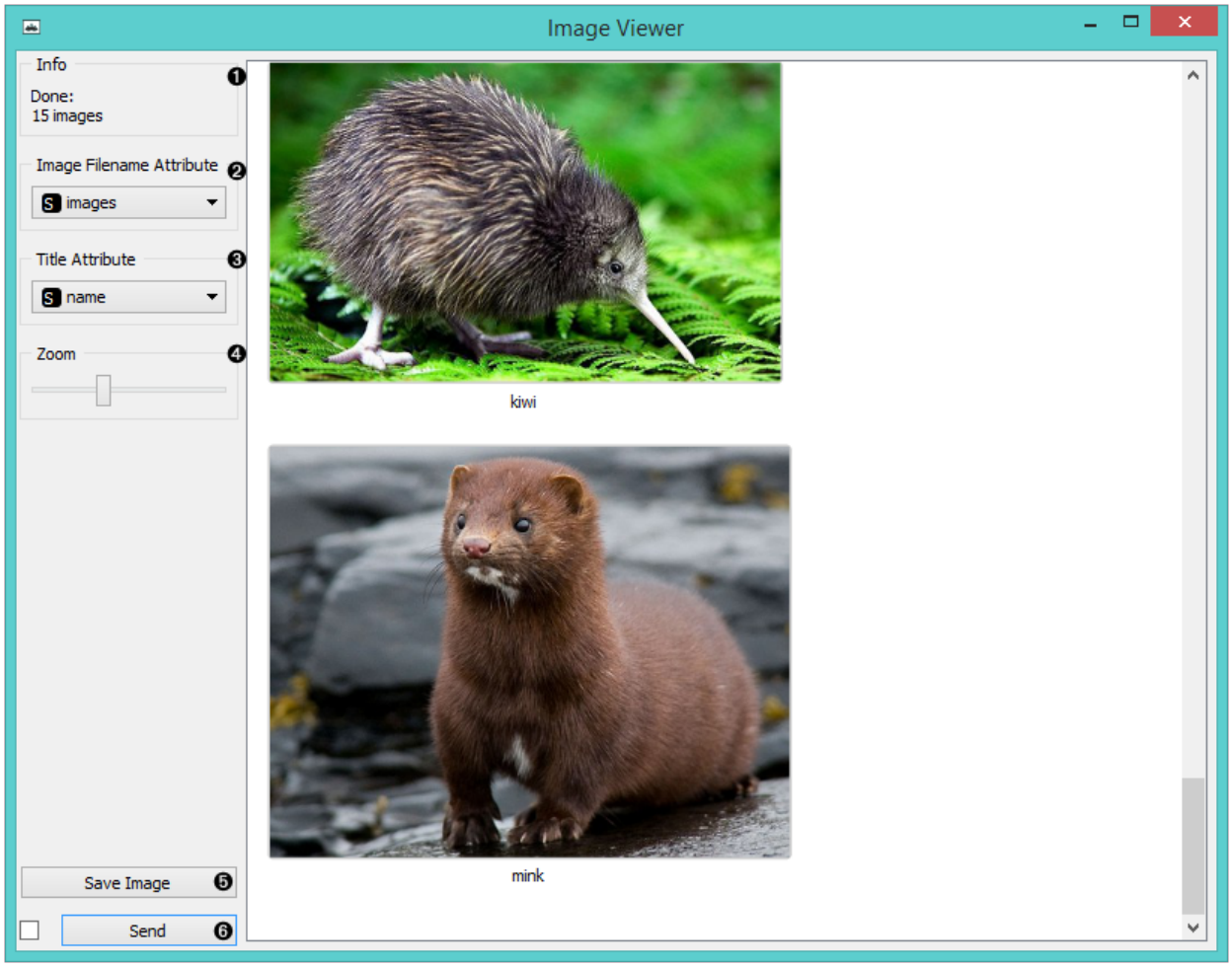
Outputs:

- **Data**

  Images that come with the data.

Description

The **Image Viewer** widget can display images from a data set, which are stored locally or on the internet. It can be used for image comparison, while looking for similarities or discrepancies between selected data instances (e.g. bacterial growth or bitmap representations of handwriting).

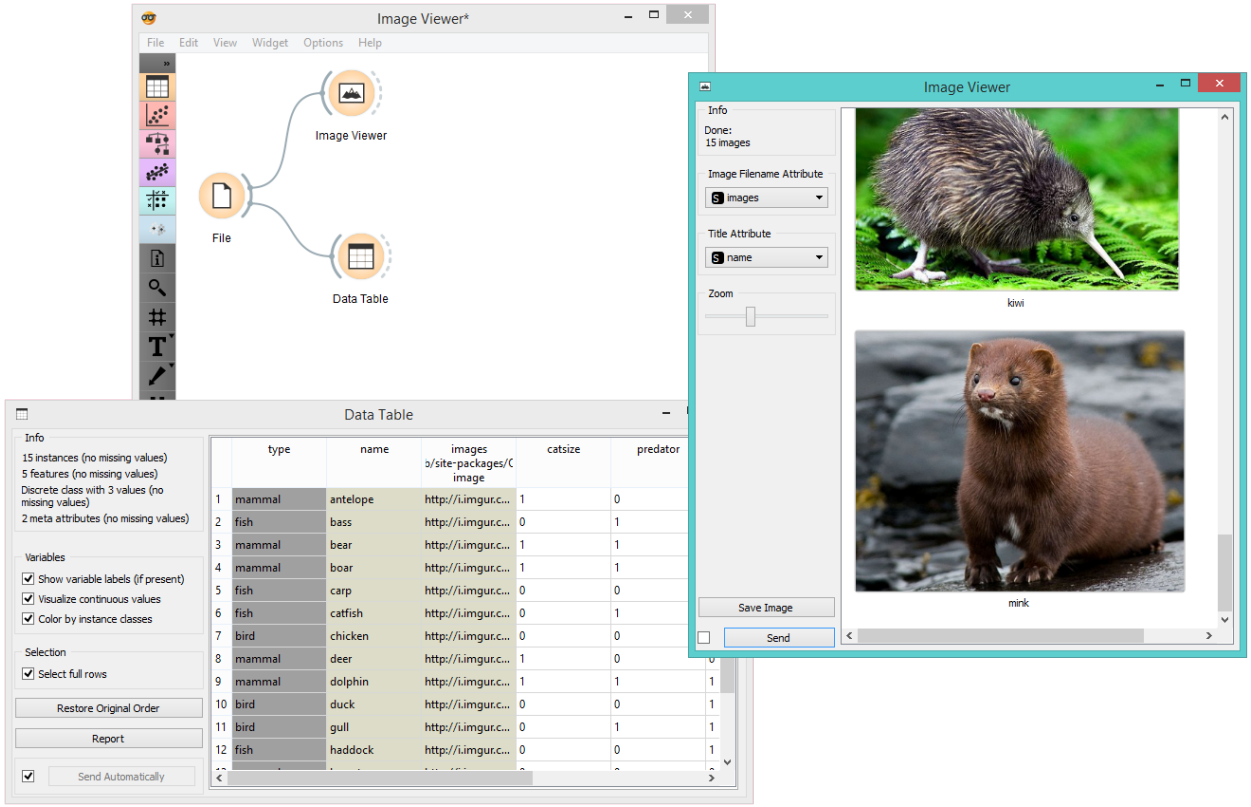


1. Information on the data set
2. Select the column with image data (links).
3. Select the column with image titles.

4. Zoom in or out.
5. Saves the visualization in a file.
6. Tick the box on the left to commit changes automatically. Alternatively, click *Send*.

Examples

A very simple way to use this widget is to connect the *File* widget with **Image Viewer** and see all the images that come with your data set.



Alternatively, you can visualize only selected instances, as shown in the example below.

Impute

Replaces unknown values in the data.

Signals

Inputs

- **Data**

  A data set.

- **Learner for Imputation**

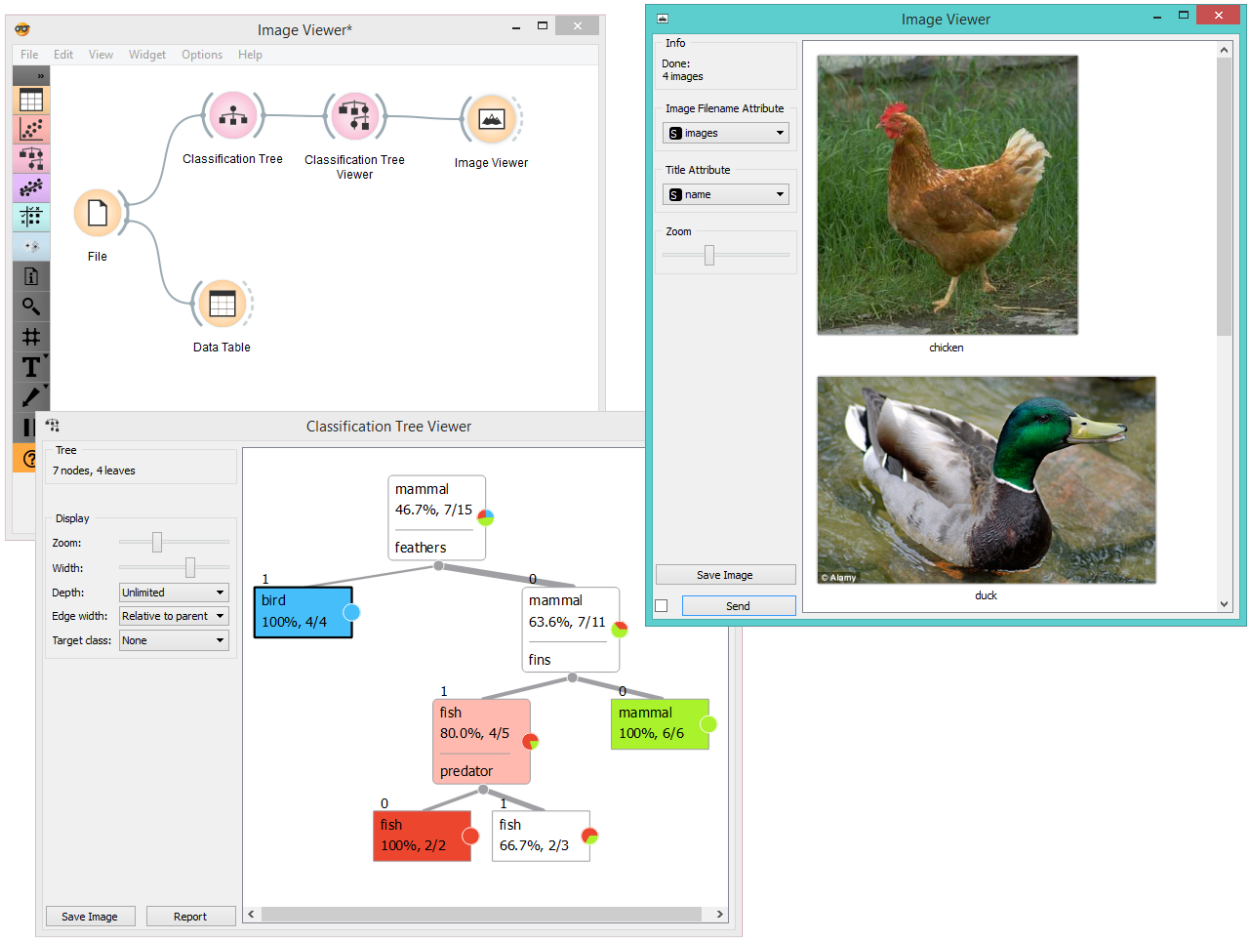


Image Viewer*
File Edit View Widget Options Help
Classification Tree
Classification Tree Viewer
Image Viewer
File
Data Table
Classification Tree Viewer
Tree
7 nodes, 4 leaves
Display
Zoom:
Width:
Depth: Unlimited
Edge width: Relative to parent
Target class: None
Save Image
Report
mammal
46.7%, 7/15
feathers
1
bird
100%, 4/4
0
mammal
63.6%, 7/11
fins
1
fish
80.0%, 4/5
predator
0
mammal
100%, 6/6
0
fish
100%, 2/2
1
fish
66.7%, 2/3
Image Viewer
Info
Done:
4 images
Image Filename Attribute
images
Title Attribute
name
Zoom
Save Image
Send
chicken
duck

A learning algorithm to be used when values are imputed with a predictive model. This algorithm, if given, substitutes the default (1-NN).

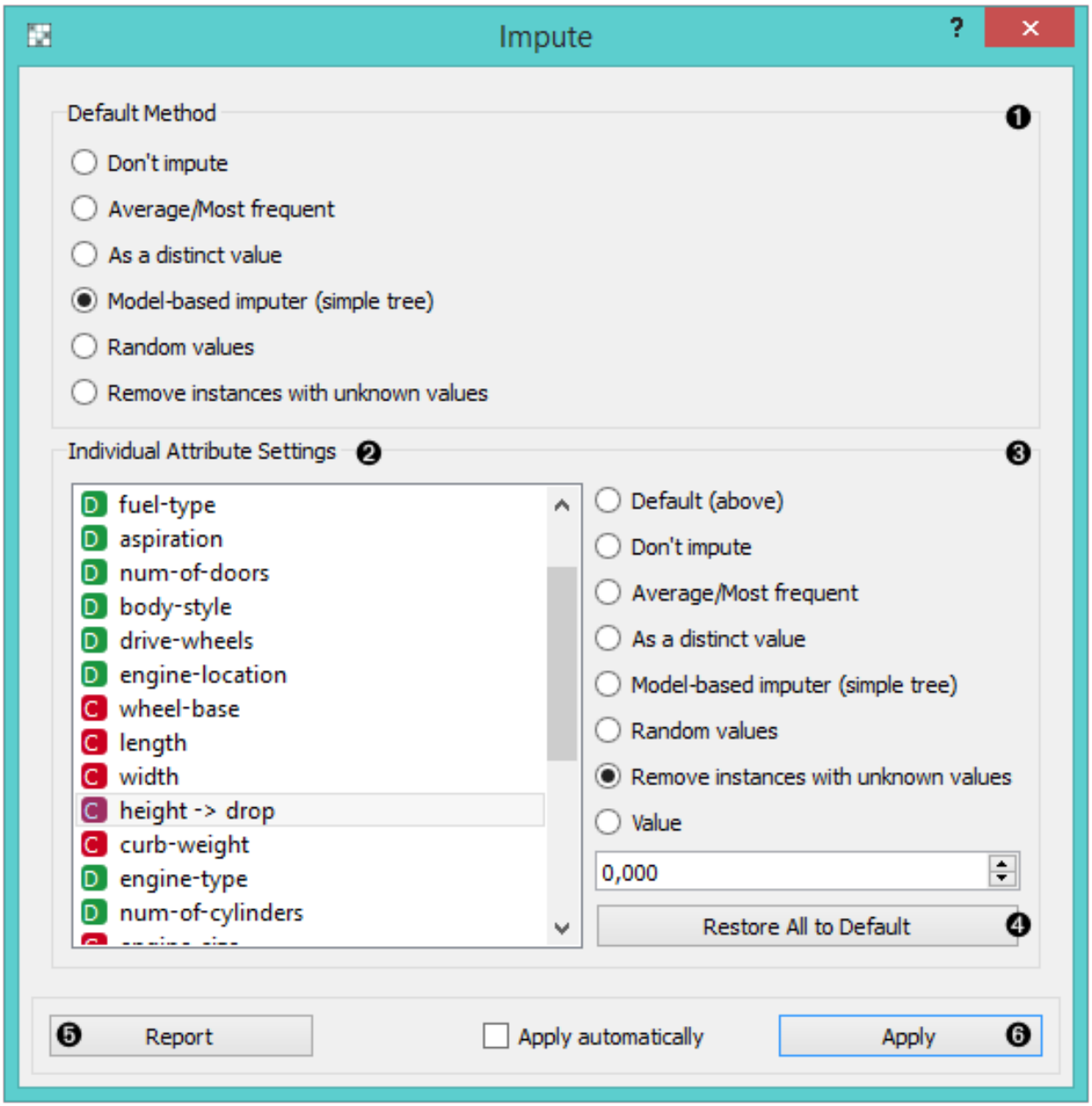
Outputs

- **Data**

  The same data set as in the input, but with the missing values imputed.

Description

Some Orange's algorithms and visualizations cannot handle unknown values in the data. This widget does what statisticians call imputation: it substitutes missing values by values either computed from the data or set by the user.



1. In the top-most box, *Default method*, the user can specify a general imputation technique for all attributes.
   - **Don't Impute** does nothing with the missing values.
   - **Average/Most-frequent** uses the average value (for continuous attributes) or the most common value (for discrete attributes).
   - **As a distinct value** creates new values to substitute the missing ones.
   - **Model-based imputer** constructs a model for predicting the missing value, based on values of other attributes; a separate model is constructed for each attribute. The default model is 1-NN learner, which takes the value from the most similar example (this is sometimes referred to as hot deck imputation). This algorithm can be substituted by one that the user connects to the input signal Learner for Imputation. Note, however, that if there are discrete and continuous attributes in the data, the algorithm needs to be capable of handling them both; at the moment only 1-NN learner can do that. (In the future, when Orange has more regressors, the Impute widget may have separate input signals for discrete and continuous models.)
   - **Random values** computes the distributions of values for each attribute and then imputes by picking random values from them.
   - **Remove examples with missing values** removes the example containing missing values. This check also applies to the class attribute if *Impute class values* is checked.
2. It is possible to specify individual treatment for each attribute, which overrides the default treatment set. One can also specify a manually defined value used for imputation. In the screenshot, we decided not to impute the values of "*normalized-losses*" and "*make*", the missing values of "*aspiration*" will be replaced by random values, while the missing values of "*body-style*" and "*drive-wheels*" are replaced by "*hatchback*" and "*fwd*", respectively. If the values of "*length*", "*width*" or "*height*" are missing, the example is discarded. Values of all other attributes use the default method set above (model-based imputer, in our case).
3. The imputation methods for individual attributes are the same as default. methods.
4. *Restore All to Default* resets the individual attribute treatments to default.
5. Produce a report.
6. All changes are committed immediately if *Apply automatically* is checked. Otherwise, *Apply* needs to be ticked to apply any new settings.

Example

To demonstrate how the **Impute** widget works, we played around with the *Iris* data set and deleted some of the data. We used the **Impute** widget and selected the *Model-based imputer* to impute the missing values. In another *Data Table*, we see how the question marks turned into distinct values ("Iris-setosa, "Iris-versicolor").

Merge Data

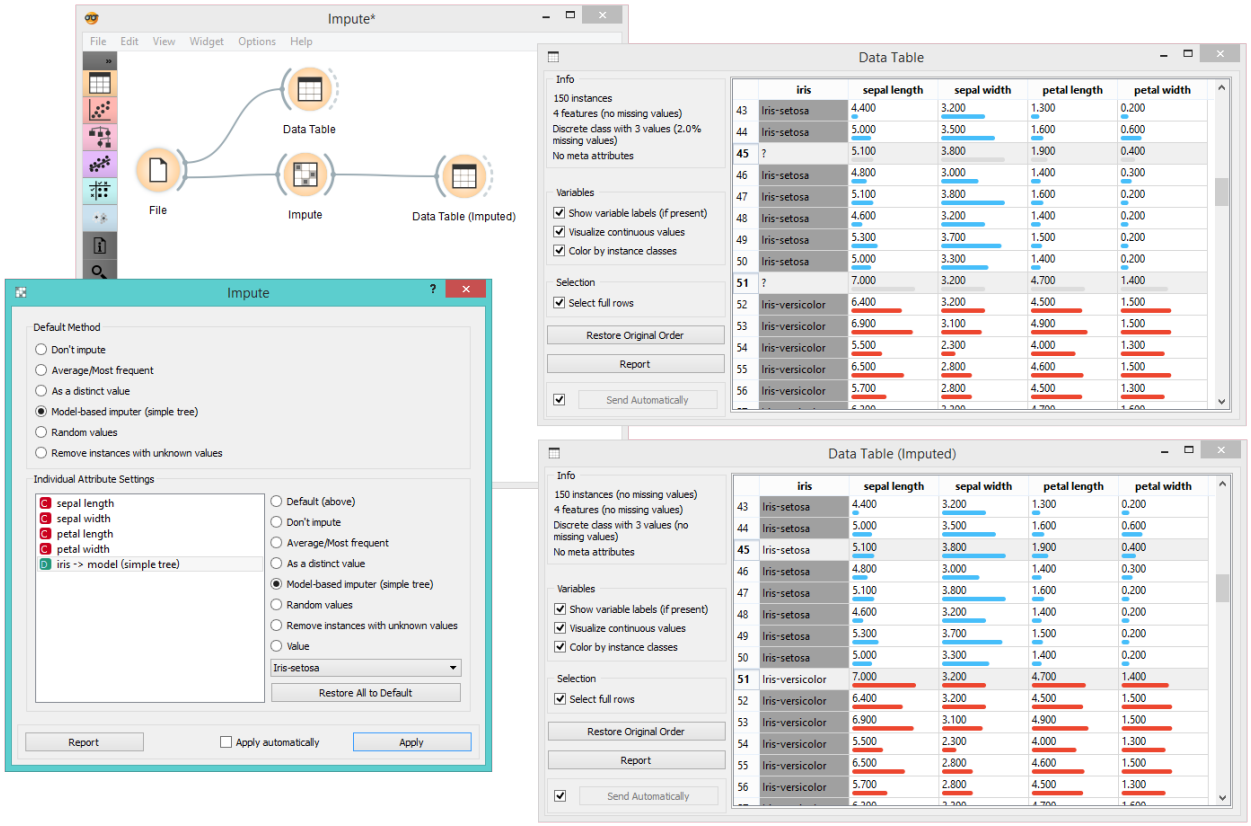
Merges two data sets, based on values of selected attributes.

Signals

Inputs:

- **Data**

  Attribute-valued data set.


Impute*
File Edit View Widget Options Help
Data Table
File
Impute
Data Table (Imputed)
Impute
Default Method
Don't impute
Average/Most frequent
As a distinct value
Model-based imputer (simple tree)
Random values
Remove instances with unknown values
Individual Attribute Settings
sepal length
sepal width
petal length
petal width
iris -> model (simple tree)
Default (above)
Don't impute
Average/Most frequent
As a distinct value
Model-based imputer (simple tree)
Random values
Remove instances with unknown values
Value
Iris-setosa
Restore All to Default
Report
Apply automatically
Apply
Data Table
Info
150 instances
4 features (no missing values)
Discrete class with 3 values (2.0% missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
iris sepal length sepal width petal length petal width
43 Iris-setosa 4.400 3.200 1.300 0.200
44 Iris-setosa 5.000 3.500 1.600 0.600
45 ? 5.100 3.800 1.900 0.400
46 Iris-setosa 4.800 3.000 1.400 0.300
47 Iris-setosa 5.100 3.800 1.600 0.200
48 Iris-setosa 4.600 3.200 1.400 0.200
49 Iris-setosa 5.300 3.700 1.500 0.200
50 Iris-setosa 5.000 3.300 1.400 0.200
51 ? 7.000 3.200 4.700 1.400
52 Iris-versicolor 6.400 3.200 4.500 1.500
53 Iris-versicolor 6.900 3.100 4.900 1.500
54 Iris-versicolor 5.500 2.300 4.000 1.300
55 Iris-versicolor 6.500 2.800 4.600 1.500
56 Iris-versicolor 5.700 2.800 4.500 1.300
Data Table (Imputed)
Info
150 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
iris sepal length sepal width petal length petal width
43 Iris-setosa 4.400 3.200 1.300 0.200
44 Iris-setosa 5.000 3.500 1.600 0.600
45 Iris-setosa 5.100 3.800 1.900 0.400
46 Iris-setosa 4.800 3.000 1.400 0.300
47 Iris-setosa 5.100 3.800 1.600 0.200
48 Iris-setosa 4.600 3.200 1.400 0.200
49 Iris-setosa 5.300 3.700 1.500 0.200
50 Iris-setosa 5.000 3.300 1.400 0.200
51 Iris-versicolor 7.000 3.200 4.700 1.400
52 Iris-versicolor 6.400 3.200 4.500 1.500
53 Iris-versicolor 6.900 3.100 4.900 1.500
54 Iris-versicolor 5.500 2.300 4.000 1.300
55 Iris-versicolor 6.500 2.800 4.600 1.500
56 Iris-versicolor 5.700 2.800 4.500 1.300

- **Extra Data**

  Attribute-valued data set.

Outputs:

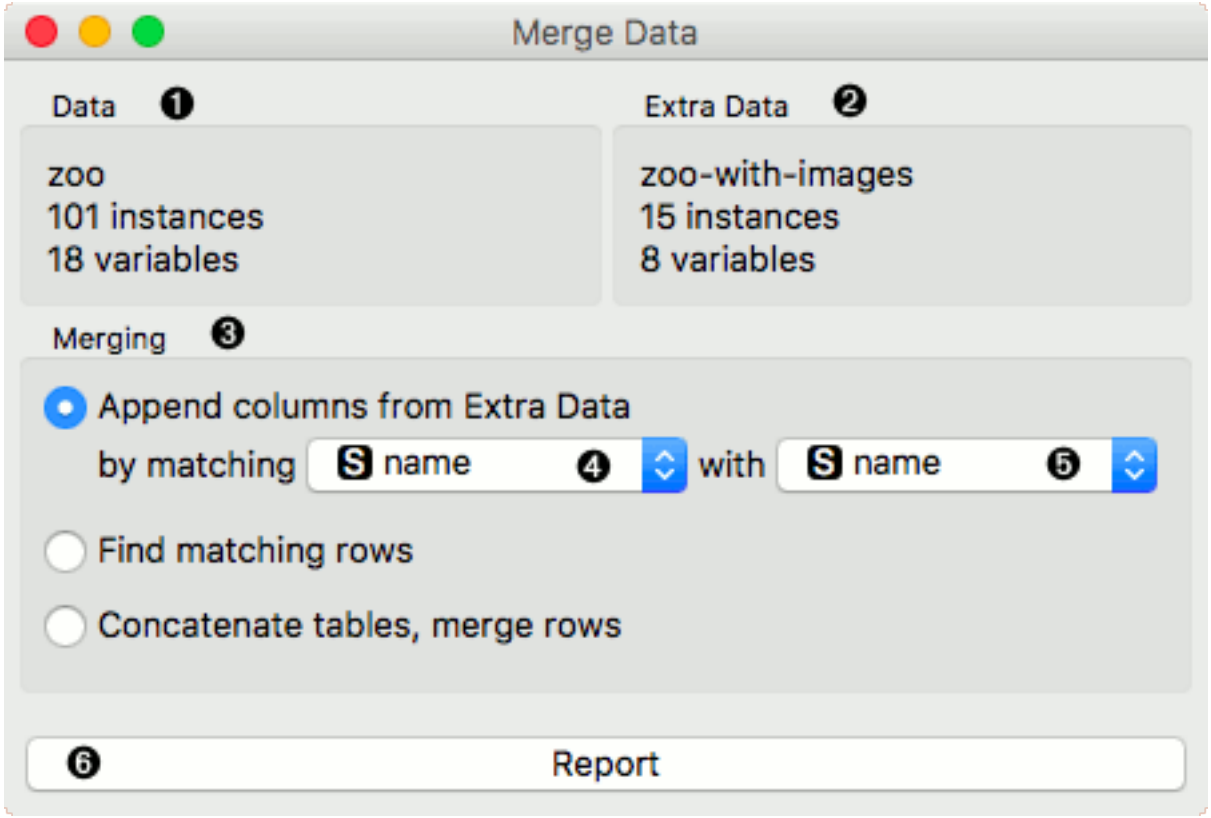
- **Data**

  Instances from input data to which attributes from input extra data are added.

Description

The **Merge Data** widget is used to horizontally merge two data sets, based on values of selected attributes. In the input, two data sets are required, data and extra data. The widget allows selection of an attribute from each domain, which will be used to perform the merging. The widget produces one output. It corresponds to instances from the input data to which attributes from input extra data are appended.

Merging is done by values of selected (merging) attributes. First, the value of the merging attribute from Data is taken and instances from Extra Data are searched for matching values. If more than a single instance from Extra Data was to be found, the attribute is removed from available merging attributes.

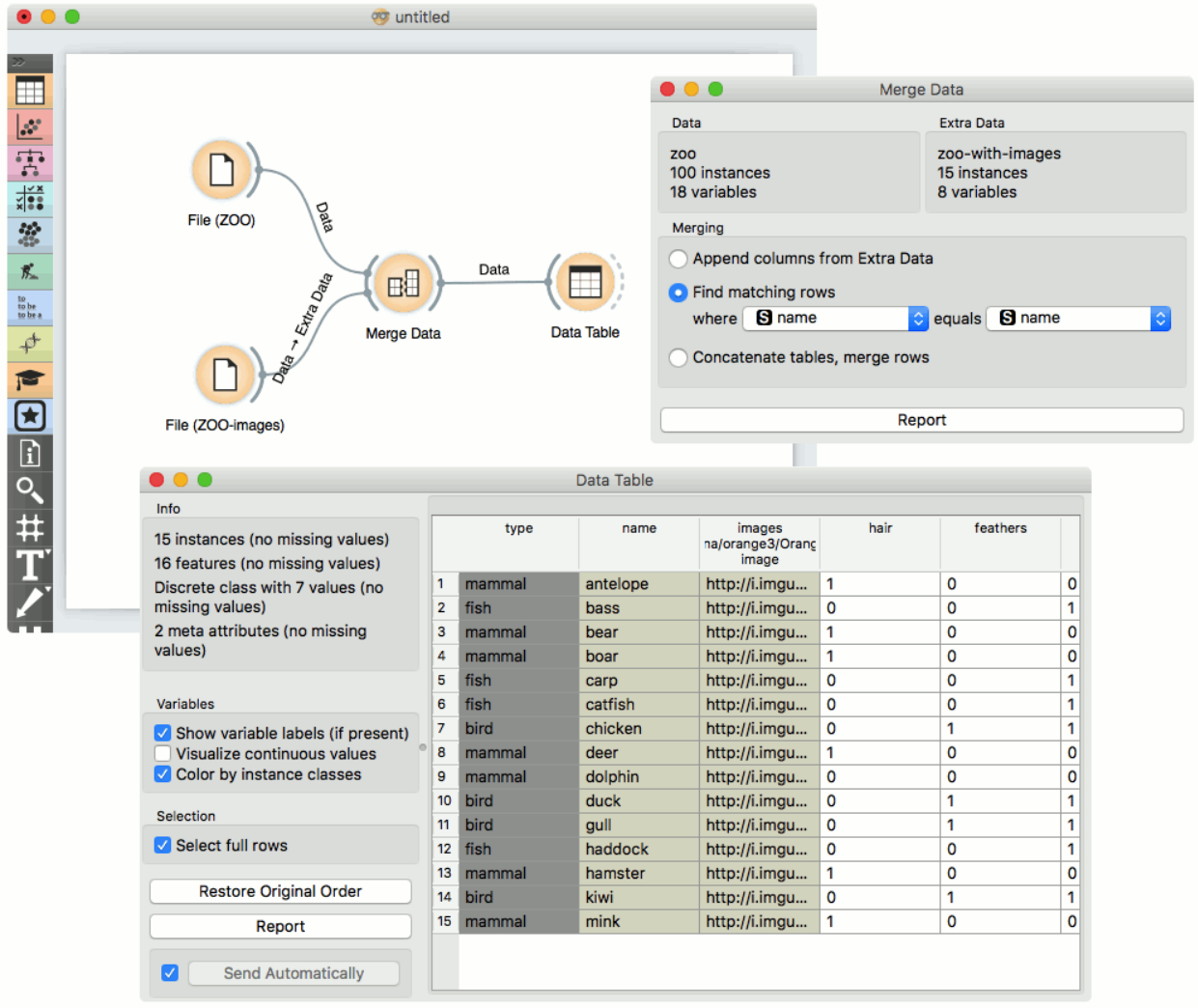


1. Information on Data
2. Information on Extra Data
3. Merging type. **Append columns from Extra Data** outputs all instances from Data appended by matching instances from Extra Data. When no match is found, unknown values are appended. **Find matching rows** outputs similar as above, except hen no match is found, instances are excluded. **Concatenate tables, merge rows** outputs all instances from both inputs, even though the match may not be found. In that case unknown values are assigned.

4. List of comparable attributes from Data
5. List of comparable attributes from Extra Data
6. Produce a report.

Example

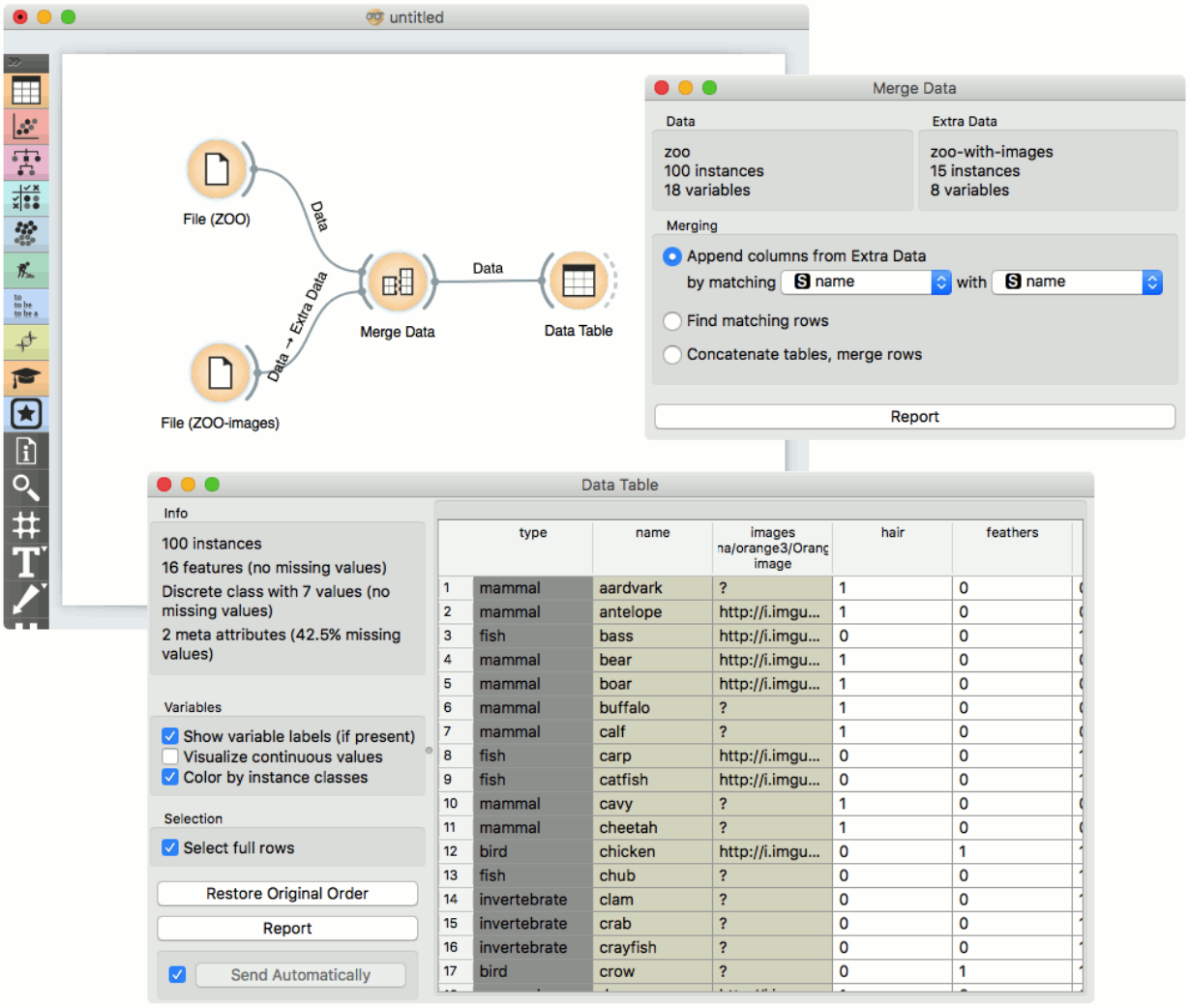
Merging two data sets results in appending new attributes to the original file, based on a selected common attribute. In the example below, we wanted to merge the **zoo.tab** file containing only factual data with `zoo-with-images.tab` containing images. Both files share a common string attribute *names*. Now, we create a workflow connecting the two files. The *zoo.tab* data is connected to **Data** input of the **Merge Data** widget, and the *zoo-with-images.tab* data to the **Extra Data** input. Outputs of the **Merge Data** widget is then connected to the *Data Table* widget. In the latter, the **Merged Data** channels are shown, where image attributes are added to the original data.

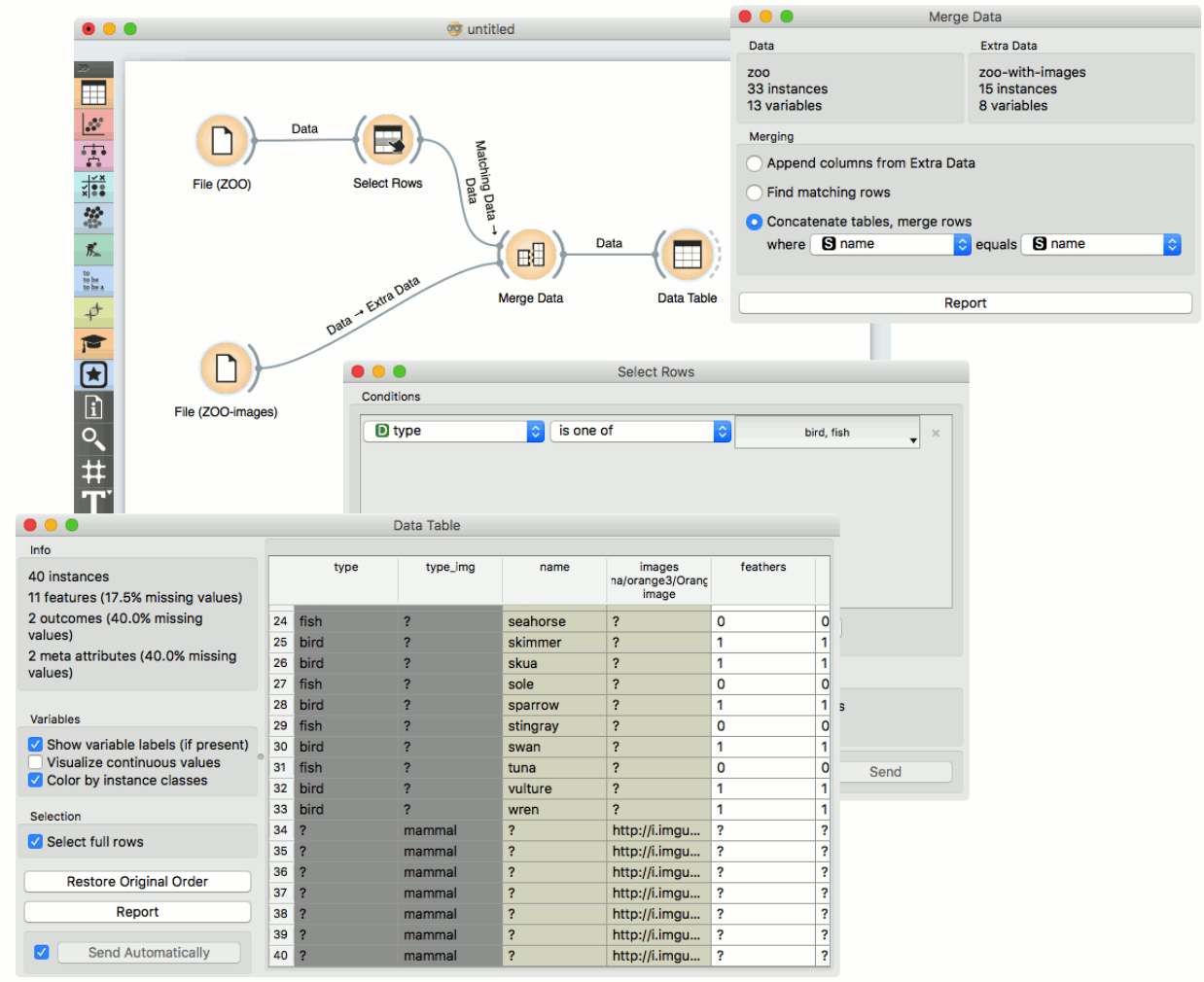


	type	name	images	hair	feathers	
1	mammal	antelope	http://i.imgu...	1	0	0
2	fish	bass	http://i.imgu...	0	0	1
3	mammal	bear	http://i.imgu...	1	0	0
4	mammal	boar	http://i.imgu...	1	0	0
5	fish	carp	http://i.imgu...	0	0	1
6	fish	catfish	http://i.imgu...	0	0	1
7	bird	chicken	http://i.imgu...	0	1	1
8	mammal	deer	http://i.imgu...	1	0	0
9	mammal	dolphin	http://i.imgu...	0	0	0
10	bird	duck	http://i.imgu...	0	1	1
11	bird	gull	http://i.imgu...	0	1	1
12	fish	haddock	http://i.imgu...	0	0	1
13	mammal	hamster	http://i.imgu...	1	0	0
14	bird	kiwi	http://i.imgu...	0	1	1
15	mammal	mink	http://i.imgu...	1	0	0

The case where we want to include all instances in the output, even those where no match by attribute *names* was found, is shown in the following workflow.

The third type of merging is shown in the next workflow. The output consist of both inputs, with unknown values assigned where no match was found.

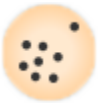

untitled
File (ZOO)
File (ZOO-images)
Data
Data → Extra Data
Merge Data
Data
Data Table
Merge Data
Data
zoo
100 instances
18 variables
Extra Data
zoo-with-images
15 instances
8 variables
Merging
Append columns from Extra Data
by matching name with name
Find matching rows
Concatenate tables, merge rows
Report
Data Table
Info
100 instances
16 features (no missing values)
Discrete class with 7 values (no missing values)
2 meta attributes (42.5% missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
type
name
images
hair
feathers
1 mammal aardvark ? 1 0
2 mammal antelope http://i.imgu... 1 0
3 fish bass http://i.imgu... 0 0
4 mammal bear http://i.imgu... 1 0
5 mammal boar http://i.imgu... 1 0
6 mammal buffalo ? 1 0
7 mammal calf ? 1 0
8 fish carp http://i.imgu... 0 0
9 fish catfish http://i.imgu... 0 0
10 mammal cavy ? 1 0
11 mammal cheetah ? 1 0
12 bird chicken http://i.imgu... 0 1
13 fish chub ? 0 0
14 invertebrate clam ? 0 0
15 invertebrate crab ? 0 0
16 invertebrate crayfish ? 0 0
17 bird crow ? 0 1


untitled
File (ZOO)
Data
Select Rows
Matching Data
Merge Data
Data
Data Table
File (ZOO-images)
Data → Extra Data
Merge Data
Data
Extra Data
zoo
33 instances
13 variables
zoo-with-images
15 instances
8 variables
Merging
Append columns from Extra Data
Find matching rows
Concatenate tables, merge rows
where name equals name
Report
Select Rows
Conditions
type
is one of
bird, fish
Send
Data Table
Info
40 instances
11 features (17.5% missing values)
2 outcomes (40.0% missing values)
2 meta attributes (40.0% missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
type
type_img
name
images
feathers
24 fish ? seahorse ? 0 0
25 bird ? skimmer ? 1 1
26 bird ? skua ? 1 1
27 fish ? sole ? 0 0
28 bird ? sparrow ? 1 1
29 fish ? stingray ? 0 0
30 bird ? swan ? 1 1
31 fish ? tuna ? 0 0
32 bird ? vulture ? 1 1
33 bird ? wren ? 1 1
34 ? mammal ? http://i.imgu... ? ?
35 ? mammal ? http://i.imgu... ? ?
36 ? mammal ? http://i.imgu... ? ?
37 ? mammal ? http://i.imgu... ? ?
38 ? mammal ? http://i.imgu... ? ?
39 ? mammal ? http://i.imgu... ? ?
40 ? mammal ? http://i.imgu... ? ?

Hint

If the two data sets consist of equally-named attributes (other than the ones used to perform the merging), Orange will check by default for consistency of the values of these attributes and report an error in case of non-matching values. In order to avoid the consistency checking, make sure that new attributes are created for each data set: you may use the '*Columns with the same name in different files represent different variables*' option in the *File* widget for loading the data.

Outliers



Simple outlier detection by comparing distances between instances.

Signals

Inputs:

- **Data**

  A data set

- **Distances**

  A distance matrix

Outputs:

- **Outliers**

  A data set containing instances scored as outliers

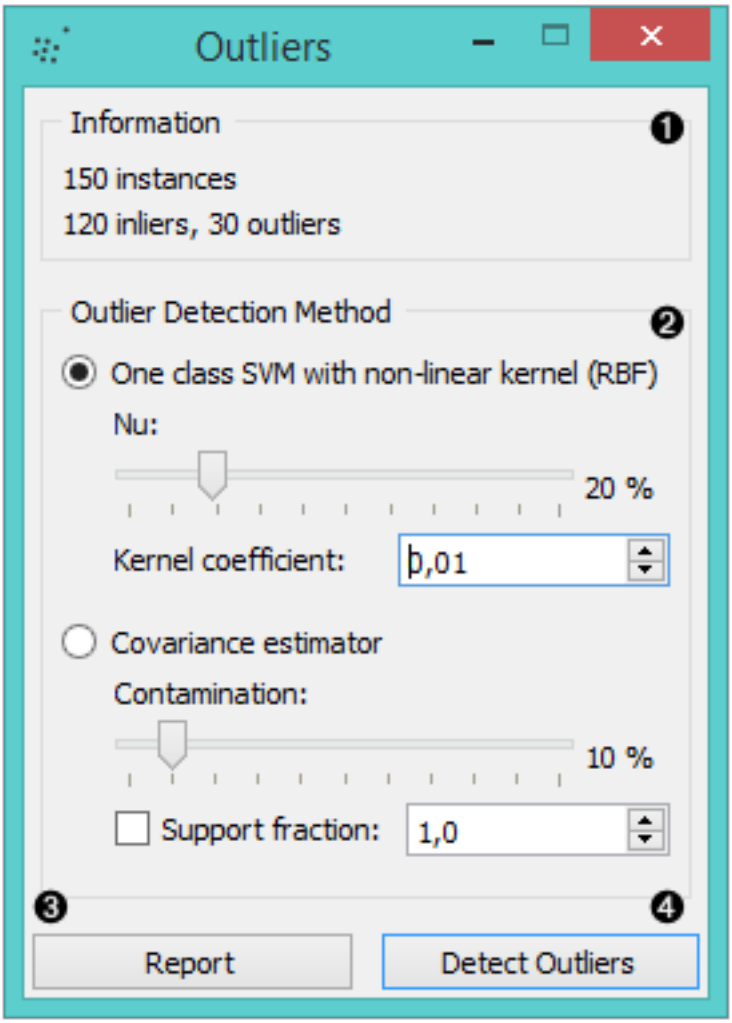
- **Inliers**

  A data set containing instances not scored as outliers

Description

The **Outliers** widget applies one of the two methods for outlier detection. Both methods apply classification to the data set, one with SVM (multiple kernels) and the other with elliptical envelope. *One-class SVM with non-linear kernels (RBF)* performs well with non-Gaussian distributions, while *Covariance estimator* works only for data with Gaussian distribution.

1. Information on the input data, number of inliers and outliers based on the selected model.
2. Select the *Outlier detection method*:
   - **One class SVM with non-linear kernel (RBF)**: classifies data as similar or different from the core class
     - **Nu** is a parameter for the upper bound on the fraction of training errors and a lower bound of the fraction of support vectors
     - **Kernel coefficient** is a gamma parameter, which specifies how much influence a single data instance has
   - **Covariance estimator**: fits ellipsis to central points with Mahalanobis distance metric

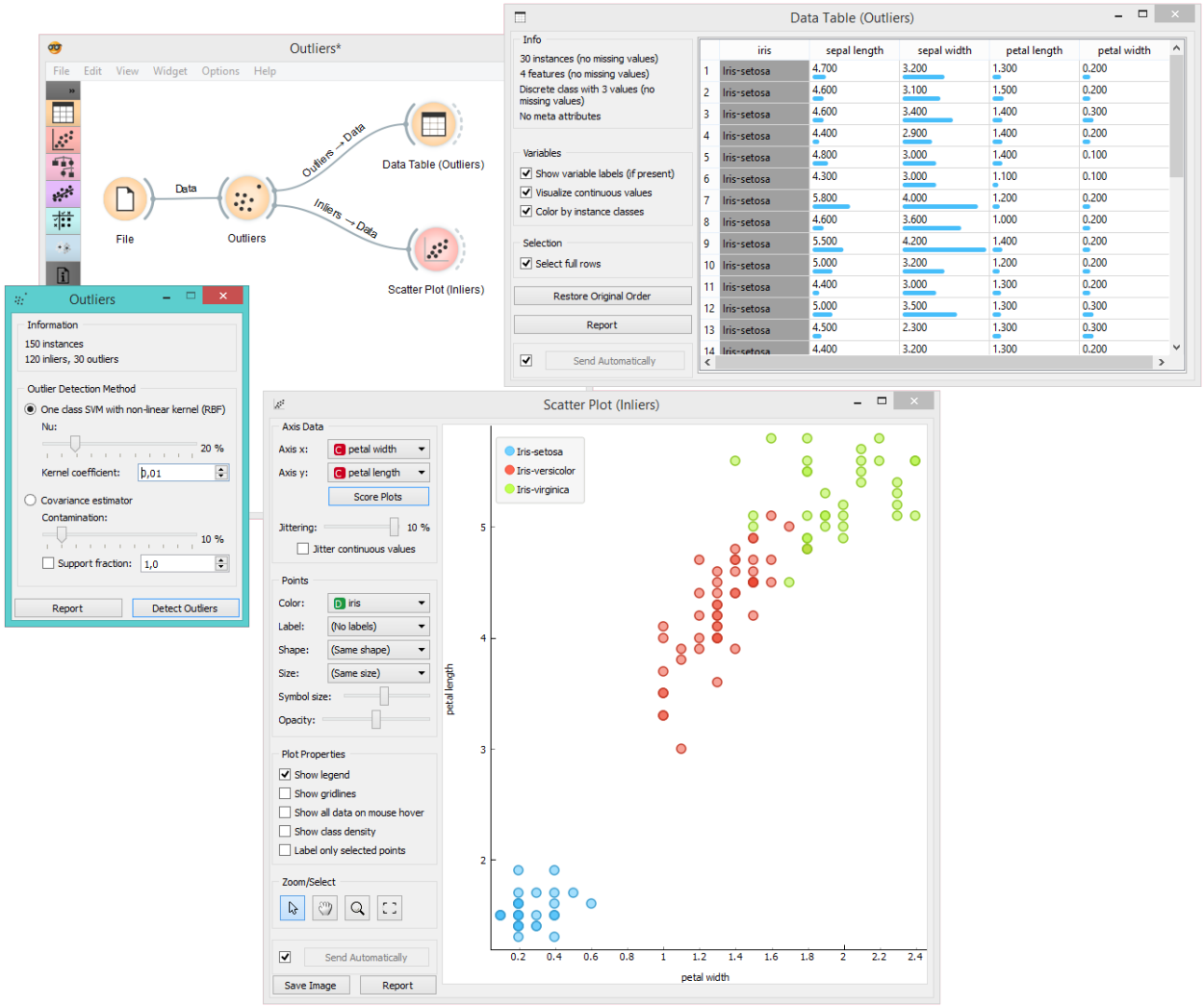

Outliers
Information
1
150 instances
120 inliers, 30 outliers
Outlier Detection Method
2
One class SVM with non-linear kernel (RBF)
Nu:
20 %
Kernel coefficient:
0,01
Covariance estimator
Contamination:
10 %
Support fraction:
1,0
3
4
Report
Detect Outliers

- **Contamination** is the proportion of outliers in the data set
- **Support fraction** specifies the proportion of points included in the estimate

3. Produce a report.
4. Click *Detect outliers* to output the data.

Example

Below, is a simple example of how to use this widget. We used the *Iris* data set to detect the outliers. We chose the *one class SVM with non-linear kernel (RBF)* method, with Nu set at 20% (less training errors, more support vectors). Then we observed the outliers in the *Data Table* widget, while we sent the inliers to the *Scatter Plot*.



Preprocess

Preprocesses data with selected methods.

Signals

Inputs:

- **Data**

  A data set.

Outputs:

- **Preprocessor**

  A preprocessing method.

- **Preprocessed Data**

  Data preprocessed with selected methods.

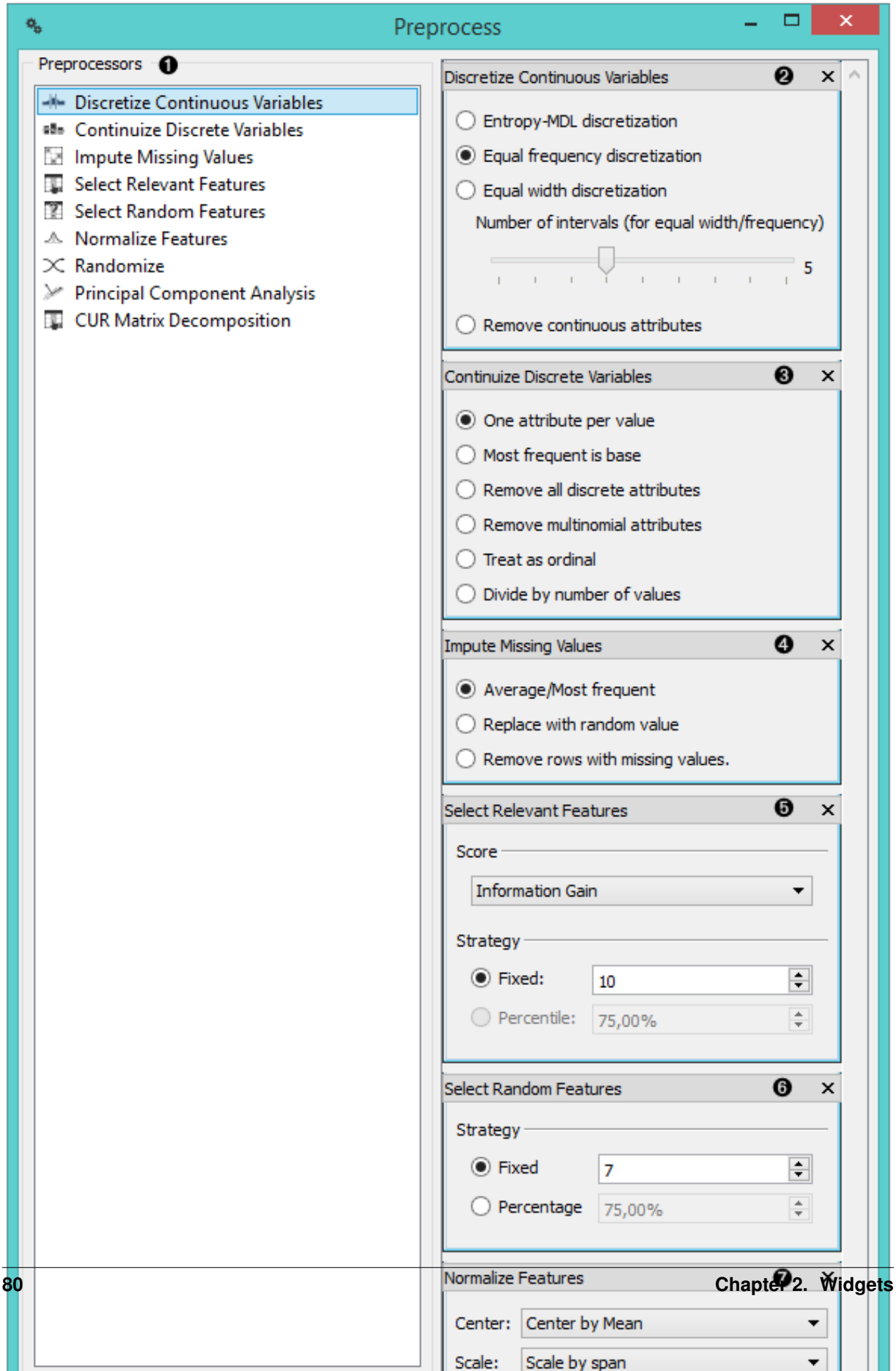
Description

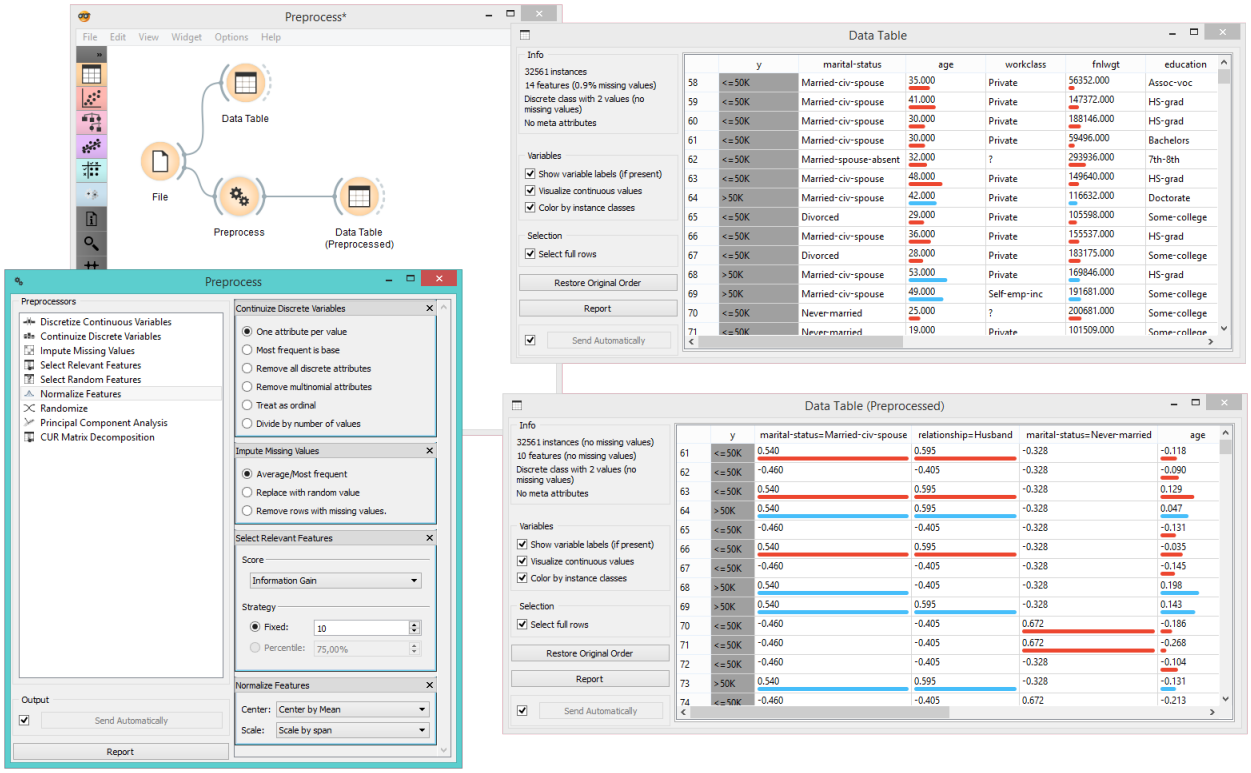
Preprocessing is crucial for achieving better-quality analysis results. The **Preprocess** widget offers five preprocessing methods to improve data quality. In this widget, you can immediately discretize continuous values or continuize discrete ones, impute missing values, select relevant features or center and scale them. Basically, this widget combines four separate widgets for simpler processing.

1. List of preprocessors. You drag the preprocessors you wish to use to the right side of the widget.
2. Discretization of continuous values
3. Continuization of discrete values
4. Impute missing values or remove them.
5. Select the most relevant features by information gain, gain ratio, Gini index.
6. Select random features
7. Normalize features
8. Randomize
9. When the box is ticked (*Send Automatically*), the widget will communicate changes automatically. Alternatively, click *Send*.
10. Produce a report.

Example

In the example below, we have used the *adult* data set and preprocessed the data. We continuized discrete values (age, education and marital status...) as *one attribute per value*, we imputed missing values (replacing ? with average values), selected 10 most relevant attributes by *Information gain*, centered them by mean and scaled by span. We can observe the changes in the *Data Table* and compare it to the non-processed data.


Preprocess
Preprocessors
1
Discretize Continuous Variables
Continuize Discrete Variables
Impute Missing Values
Select Relevant Features
Select Random Features
Normalize Features
Randomize
Principal Component Analysis
CUR Matrix Decomposition
Discretize Continuous Variables
2
Entropy-MDL discretization
Equal frequency discretization
Equal width discretization
Number of intervals (for equal width/frequency)
5
Remove continuous attributes
Continuize Discrete Variables
3
One attribute per value
Most frequent is base
Remove all discrete attributes
Remove multinomial attributes
Treat as ordinal
Divide by number of values
Impute Missing Values
4
Average/Most frequent
Replace with random value
Remove rows with missing values.
Select Relevant Features
5
Score
Information Gain
Strategy
Fixed:
10
Percentile:
75,00%
Select Random Features
6
Strategy
Fixed
7
Percentage
75,00%
Normalize Features
7
Center:
Center by Mean
Scale:
Scale by span


Preprocess*
File Edit View Widget Options Help
Data Table
File
Preprocess
Data Table (Preprocessed)
Data Table
Info
32561 instances
14 features (0.9% missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
y marital-status age workclass fnlwgt education
Preprocess
Preprocessors
Discretize Continuous Variables
Continuize Discrete Variables
Impute Missing Values
Select Relevant Features
Select Random Features
Normalize Features
Randomize
Principal Component Analysis
CUR Matrix Decomposition
Continuize Discrete Variables
One attribute per value
Most frequent is base
Remove all discrete attributes
Remove multinomial attributes
Treat as ordinal
Divide by number of values
Impute Missing Values
Average/Most frequent
Replace with random value
Remove rows with missing values.
Select Relevant Features
Score
Information Gain
Strategy
Fixed: 10
Percentile: 75.00%
Normalize Features
Center: Center by Mean
Scale: Scale by span
Output
Send Automatically
Report
Data Table (Preprocessed)
Info
32561 instances (no missing values)
10 features (no missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
y marital-status=Married-civ-spouse relationship=Husband marital-status=Never-married age

Purge Domain

Removes unused attribute values and useless attributes, sorts the remaining values.

Signals

Inputs:

- **Data**

  A data set.

Outputs:

- **Data**

  A filtered data set

Description

Definitions of nominal attributes sometimes contain values which don't appear in the data. Even if this does not happen in the original data, filtering the data, selecting examplary subsets and alike can remove all examples for which the attribute has some particular value. Such values clutter data presentation, especially various visualizations, and should be removed.

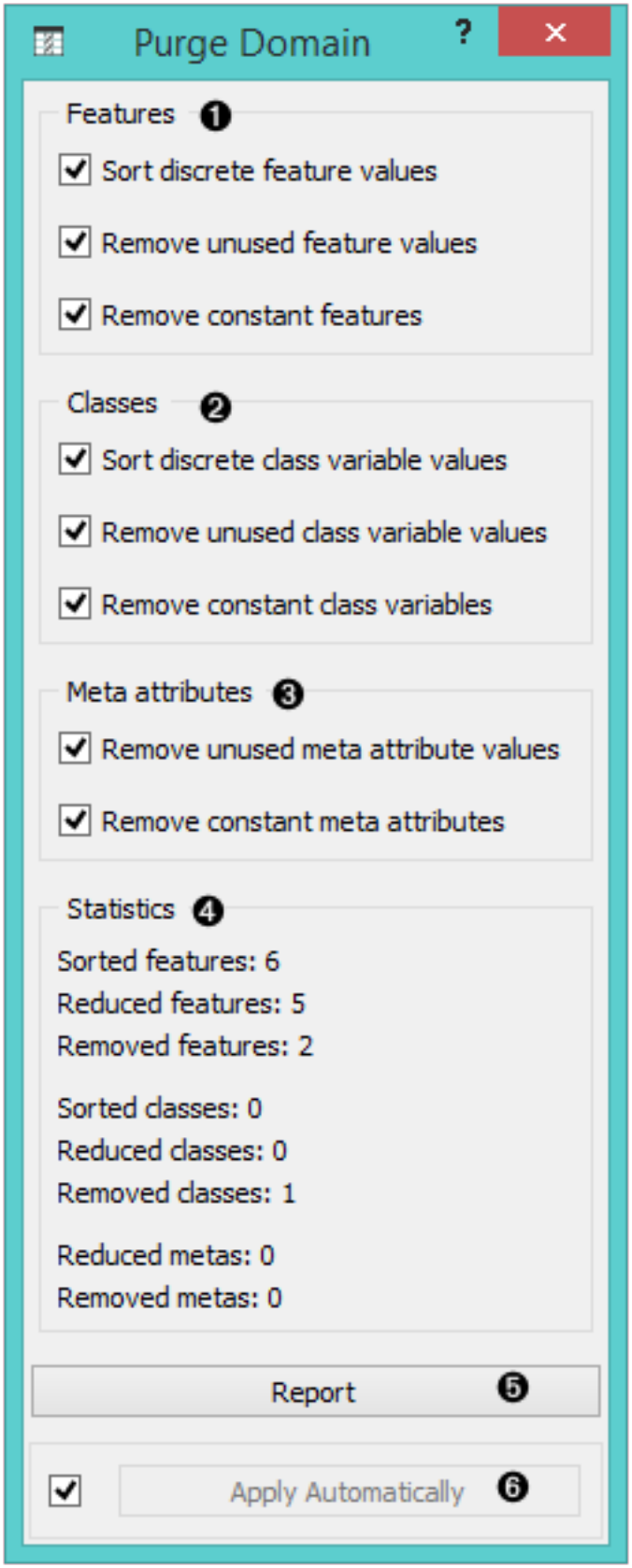
After purging an attribute, it may become single-valued or, in extreme case, have no values at all (if the value of this attribute was undefined for all examples). In such cases, the attribute can be removed.

A different issue is the order of attribute values: if the data is read from a file in a format in which values are not declared in advance, they are sorted "in order of appearance". Sometimes we would prefer to have them sorted alphabetically.

1. Purge attributes.
2. Purge classes.
3. Purge meta attributes.
4. Information on the filtering process
5. Produce a report.
6. If *Apply automatically* is ticked, the widget will output data at each change of widget settings.

Such purification is done by the widget **Purge Domain**. Ordinary attributes and class attributes are treated separately. For each, we can decide if we want the values sorted or not. Next, we may allow the widget to remove attributes with less than two values or remove the class attribute if there are less than two classes. Finally, we can instruct the widget to check which values of attributes actually appear in the data and remove the unused values. The widget cannot remove values if it is not allowed to remove the attributes, since having attributes without values makes no sense.

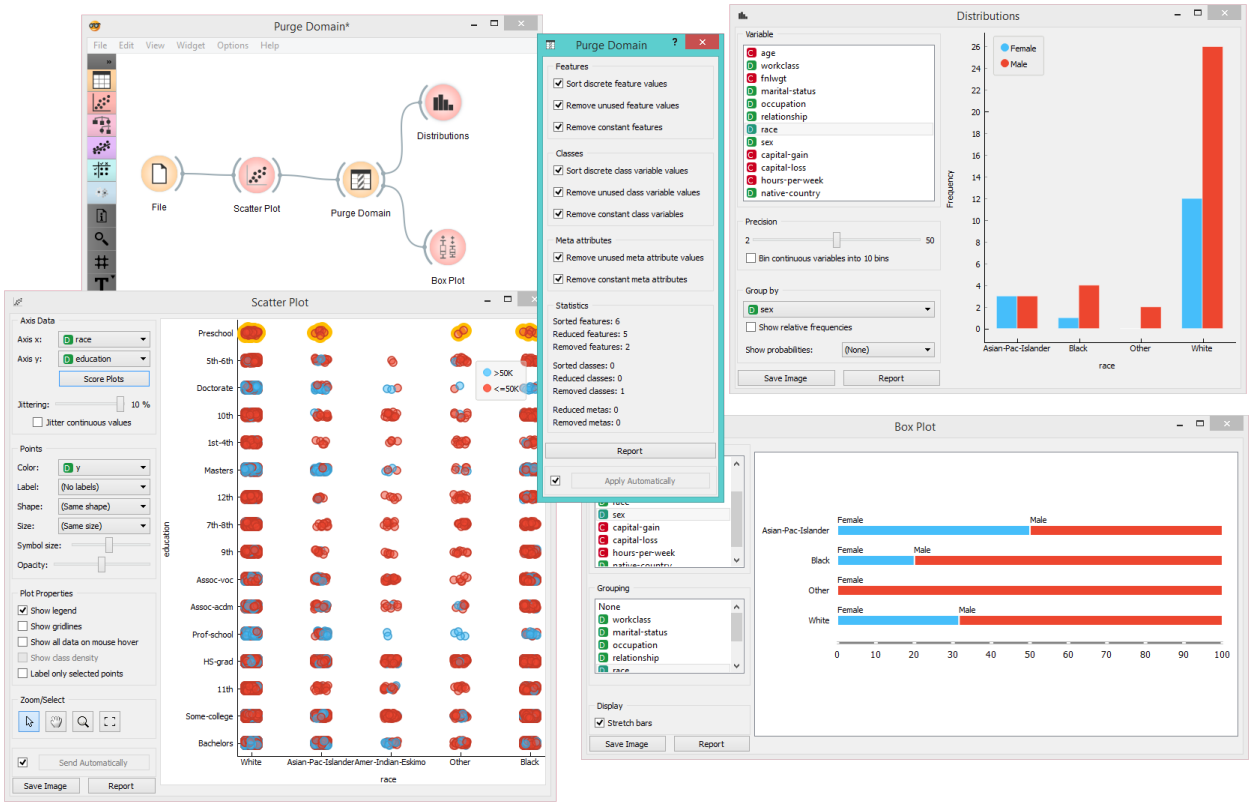
The new, reduced attributes get the prefix "R", which distinguishes them from the original ones. The values of new attributes can be computed from the old ones, but not the other way around. This means that if you construct a classifier from the new attributes, you can use it to classify the examples described by the original attributes. But not the opposite: constructing a classifier from the old attributes and using it on examples described by the reduced ones won't work. Fortunately, the latter is seldom the case. In a typical setup, one would explore the data, visualize it, filter it, purify it... and then test the final model on the original data.


Purge Domain
Features
1
Sort discrete feature values
Remove unused feature values
Remove constant features
Classes
2
Sort discrete class variable values
Remove unused class variable values
Remove constant class variables
Meta attributes
3
Remove unused meta attribute values
Remove constant meta attributes
Statistics
4
Sorted features: 6
Reduced features: 5
Removed features: 2
Sorted classes: 0
Reduced classes: 0
Removed classes: 1
Reduced metas: 0
Removed metas: 0
Report
5
Apply Automatically
6

Example

The **Purge Domain** widget would typically appear after data filtering, for instance when selecting a subset of visualized examples.

In the above schema, we play with the *adult.tab* data set: we visualize it and select a portion of the data, which contains only four out of the five original classes. To get rid of the empty class, we put the data through **Purge Domain** before going on to the *Box Plot* widget. The latter shows only the four classes which are in the **Purge Data** output. To see the effect of data purification, uncheck *Remove unused class variable values* and observe the effect this has on *Box Plot*.



Rank

Ranking of attributes in classification or regression data sets.

Signals

Inputs:

- **Data**

  An input data set.

- **Scorer** (multiple)

  Models that implement the feature scoring interface, such as linear / logistic regression, random forest, stochastic gradient descent, etc.

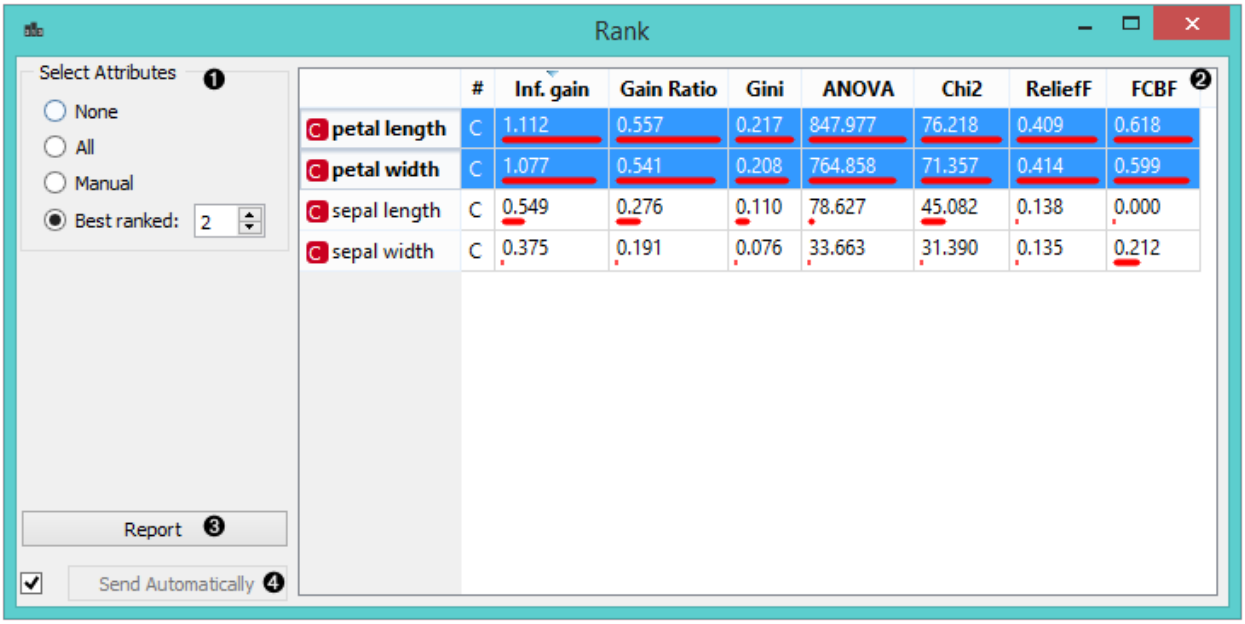
Outputs:

- **Reduced Data**

  A data set whith selected attributes.

Description

The **Rank** widget considers class-labeled data sets (classification or regression) and scores the attributes according to their correlation with the class.



1. Select attributes from the data table.
2. Data table with attributes (rows) and their scores by different scoring methods (columns)
3. Produce a report.
4. If '*Send Automatically*' is ticked, the widget automatically communicates changes to other widgets.

Scoring methods

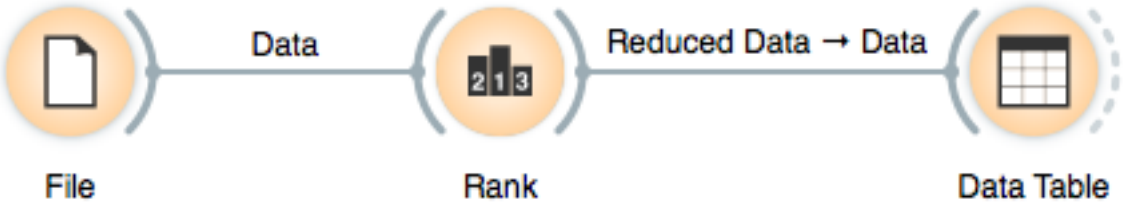
1. Information Gain: the expected amount of information (reduction of entropy)
2. Gain Ratio: a ratio of the information gain and the attribute's intrinsic information, which reduces the bias towards multivalued features that occurs in information gain
3. Gini: the inequality among values of a frequency distribution
4. ANOVA: the difference between average vaules of the feature in different classes
5. Chi2: dependence between the feature and the class as measure by the chi-square statistice
6. ReliefF: the ability of an attribute to distinguish between classes on similar data instances

7. FCBF (Fast Correlation Based Filter): entropy-based measure, which also identifies redundancy due to pairwise correlations between features

Additionally, you can connect certain learners that enable scoring the features according to how important they are in models that the learners build (e.g. *Linear / Logistic Regression*, *Random Forest*, *SGD*, . . .).

Example: Attribute Ranking and Selection

Below, we have used the **Rank** widget immediately after the *File* widget to reduce the set of data attributes and include only the most informative ones:



Notice how the widget outputs a data set that includes only the best-scored attributes:

Example: Feature Subset Selection for Machine Learning

What follows is a bit more complicated example. In the workflow below, we first split the data into a training set and a test set. In the upper branch, the training data passes through the **Rank** widget to select the most informative attributes, while in the lower branch there is no feature selection. Both feature selected and original data sets are passed to their own *Test & Score* widgets, which develop a *Naive Bayes* classifier and score it on a test set.

For data sets with many features, a naive Bayesian classifier feature selection, as shown above, would often yield a better predictive accuracy.

Color

Set color legend for variables.

Signals

Inputs:

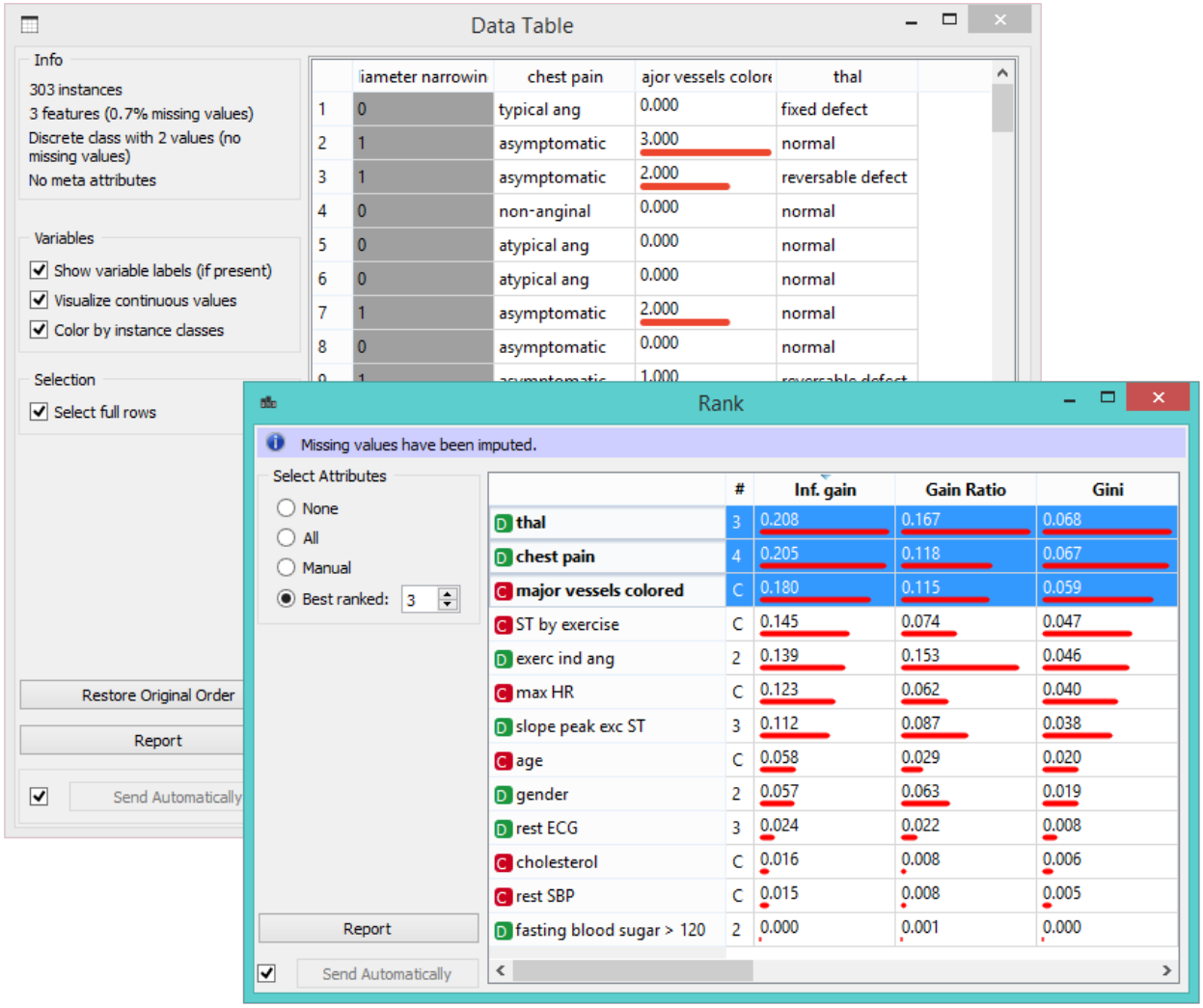
- **Data**

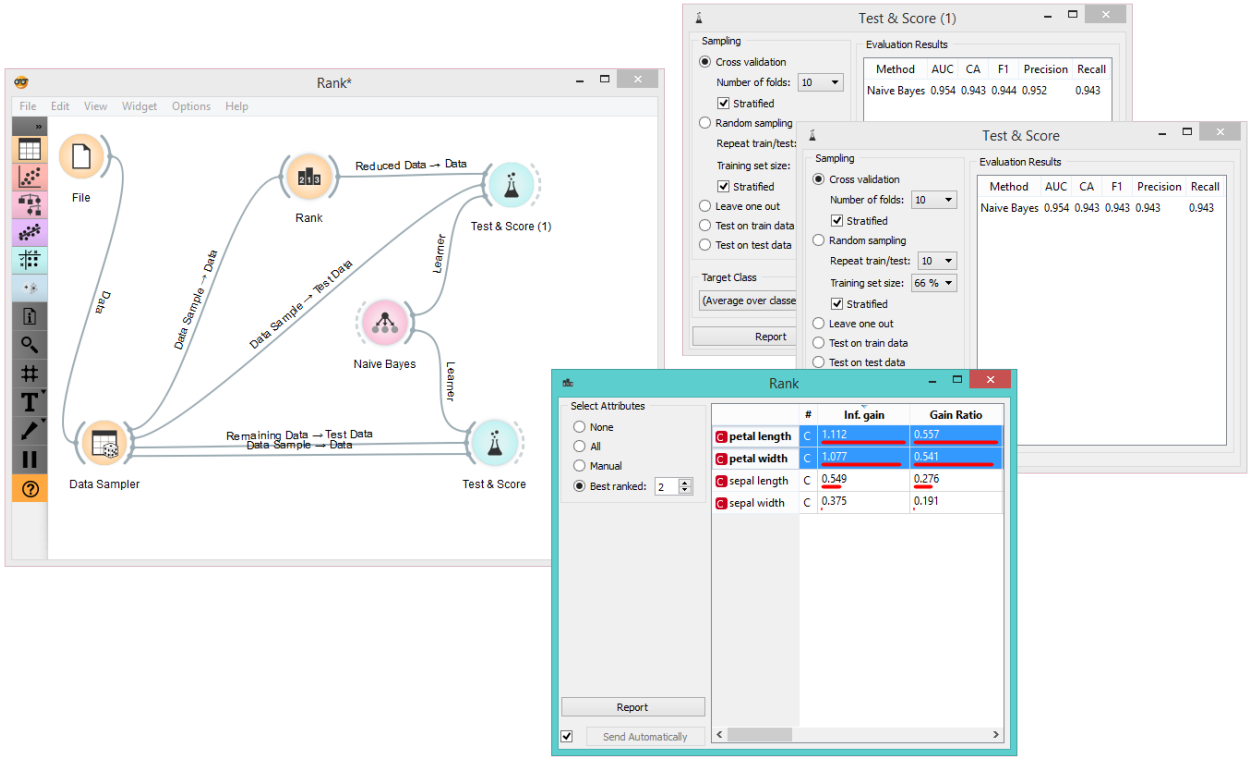
  An input data set.

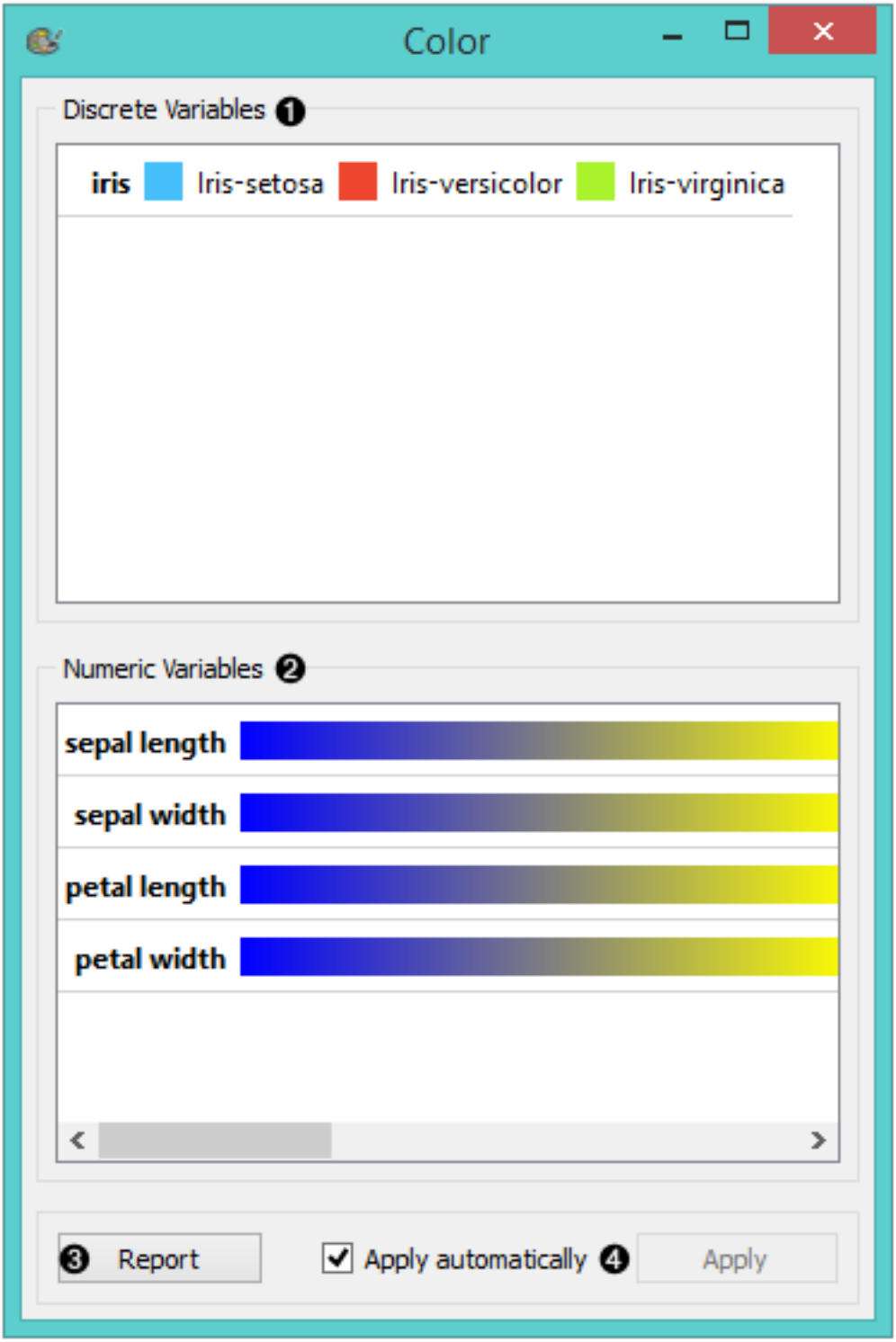
Outputs:

- **Data**

  A data set with a new color legend.


Data Table
Info
303 instances
3 features (0.7% missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
iameter narrowin
chest pain
ajor vessels colore
thal
1 0 typical ang 0.000 fixed defect
2 1 asymptomatic 3.000 normal
3 1 asymptomatic 2.000 reversable defect
4 0 non-anginal 0.000 normal
5 0 atypical ang 0.000 normal
6 0 atypical ang 0.000 normal
7 1 asymptomatic 2.000 normal
8 0 asymptomatic 0.000 normal
Rank
Missing values have been imputed.
Select Attributes
None
All
Manual
Best ranked: 3
Report
Send Automatically
Inf. gain Gain Ratio Gini
thal 3 0.208 0.167 0.068
chest pain 4 0.205 0.118 0.067
major vessels colored C 0.180 0.115 0.059
ST by exercise C 0.145 0.074 0.047
exerc ind ang 2 0.139 0.153 0.046
max HR C 0.123 0.062 0.040
slope peak exc ST 3 0.112 0.087 0.038
age C 0.058 0.029 0.020
gender 2 0.057 0.063 0.019
rest ECG 3 0.024 0.022 0.008
cholesterol C 0.016 0.008 0.006
rest SBP C 0.015 0.008 0.005
fasting blood sugar > 120 2 0.000 0.001 0.000


Rank*
File Edit View Widget Options Help
File
Rank
Reduced Data → Data
Test & Score (1)
Data
Data Sample → Data
Data Sample → Test Data
Learner
Naive Bayes
Learner
Remaining Data → Test Data
Data Sample → Data
Data Sampler
Test & Score
Test & Score (1)
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Leave one out
Test on train data
Test on test data
Target Class
(Average over classe
Report
Evaluation Results
Method AUC CA F1 Precision Recall
Naive Bayes 0.954 0.943 0.944 0.952 0.943
Test & Score
Naive Bayes 0.954 0.943 0.943 0.943 0.943
Rank
Select Attributes
None
All
Manual
Best ranked: 2
Report
Send Automatically
Inf. gain Gain Ratio
petal length 1.112 0.557
petal width 1.077 0.541
sepal length 0.549 0.276
sepal width 0.375 0.191

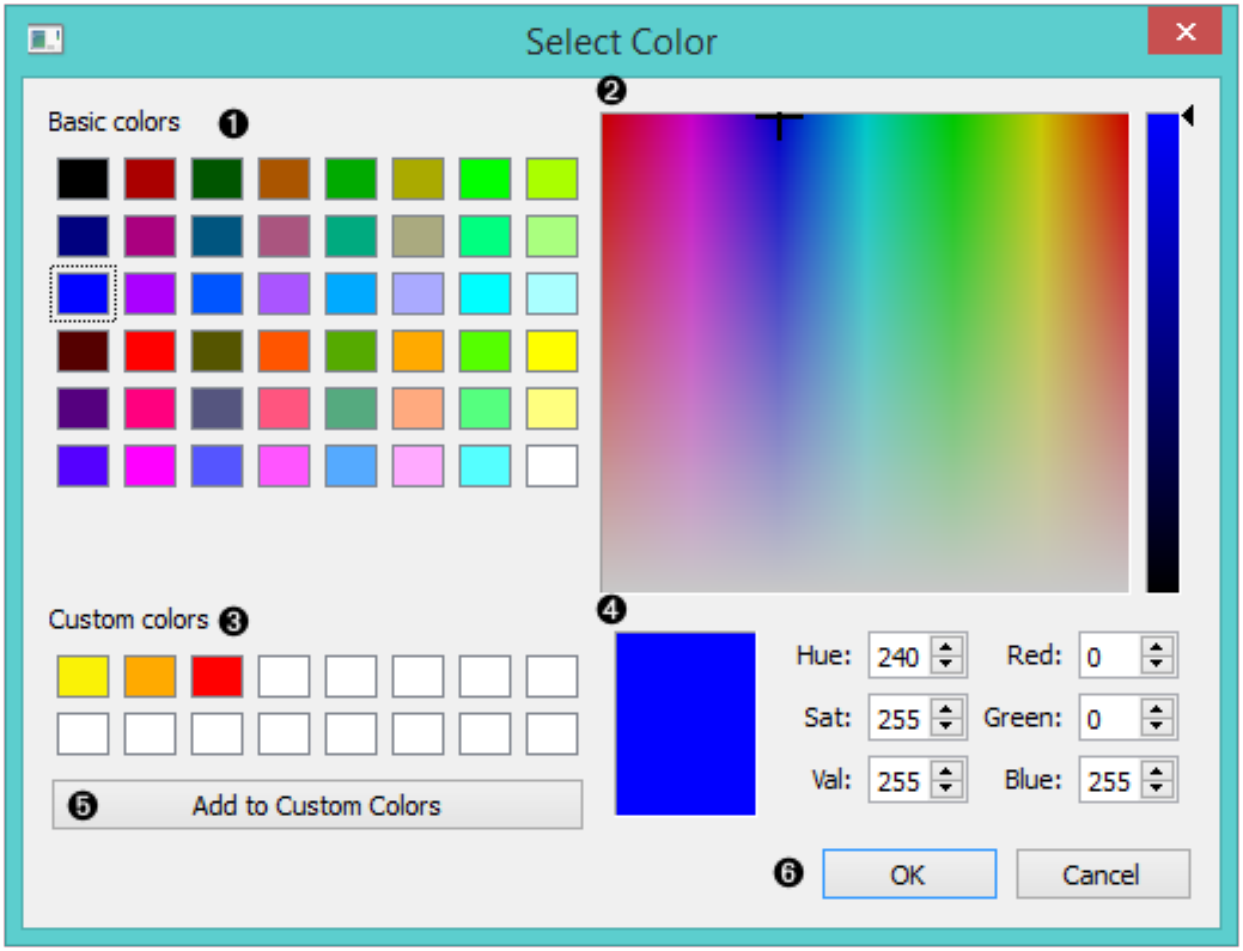

Color
Discrete Variables
1
iris
Iris-setosa
Iris-versicolor
Iris-virginica
Numeric Variables
2
sepal length
sepal width
petal length
petal width
3
Report
Apply automatically
4
Apply

Description

The **Color** widget enables you to set the color legend in your visualizations according to your own preferences. This option provides you with the tools for emphasizing your results and offers a great variety of color options for presenting your data. It can be combined with most visualizations widgets.

1. A list of discrete variables. You can set the color of each variable by double-clicking on it and opening the *Color palette* or the *Select color* window. The widget also enables text-editing. By clicking on a variable, you can change its name.
2. A list of continuous variables. You can customize the color gradients by double-clicking on them. The widget also enables text-editing. By clicking on a variable, you can change its name. If you hover over the right side side of the gradient, *Copy to all* appears. You can then apply your customized color gradient to all variables.
3. Produce a report.
4. Apply changes. If *Apply automatically* is ticked, changes will be communicated automatically. Alternatively, just click *Apply*.

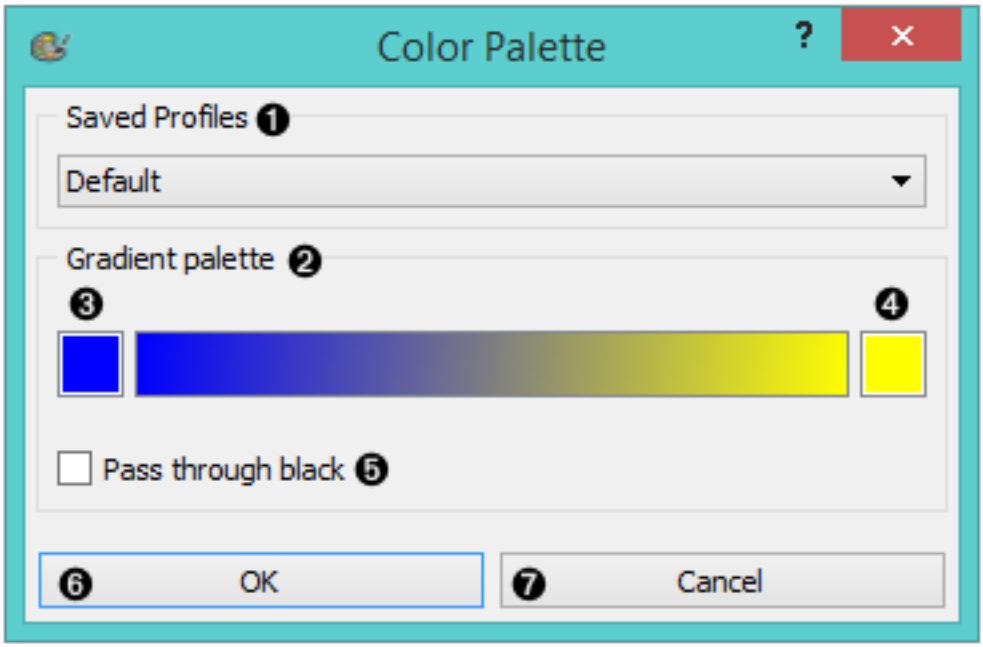
Discrete variables



1. Choose a desired color from the palette of basic colors.
2. Move the cursor to choose a custom color from the color palette.

3. Choose a custom color from your previously saved color choices.
4. Specify the custom color by:
   - entering the red, green, and blue components of the color as values between 0 (darkest) and 255 (brightest)
   - entering the hue, saturation and luminescence components of the color as values in the range 0 to 255
5. Add the created color to your custom colors.
6. Click *OK* to save your choices or *Cancel* to exit the the color palette.

Numeric variables

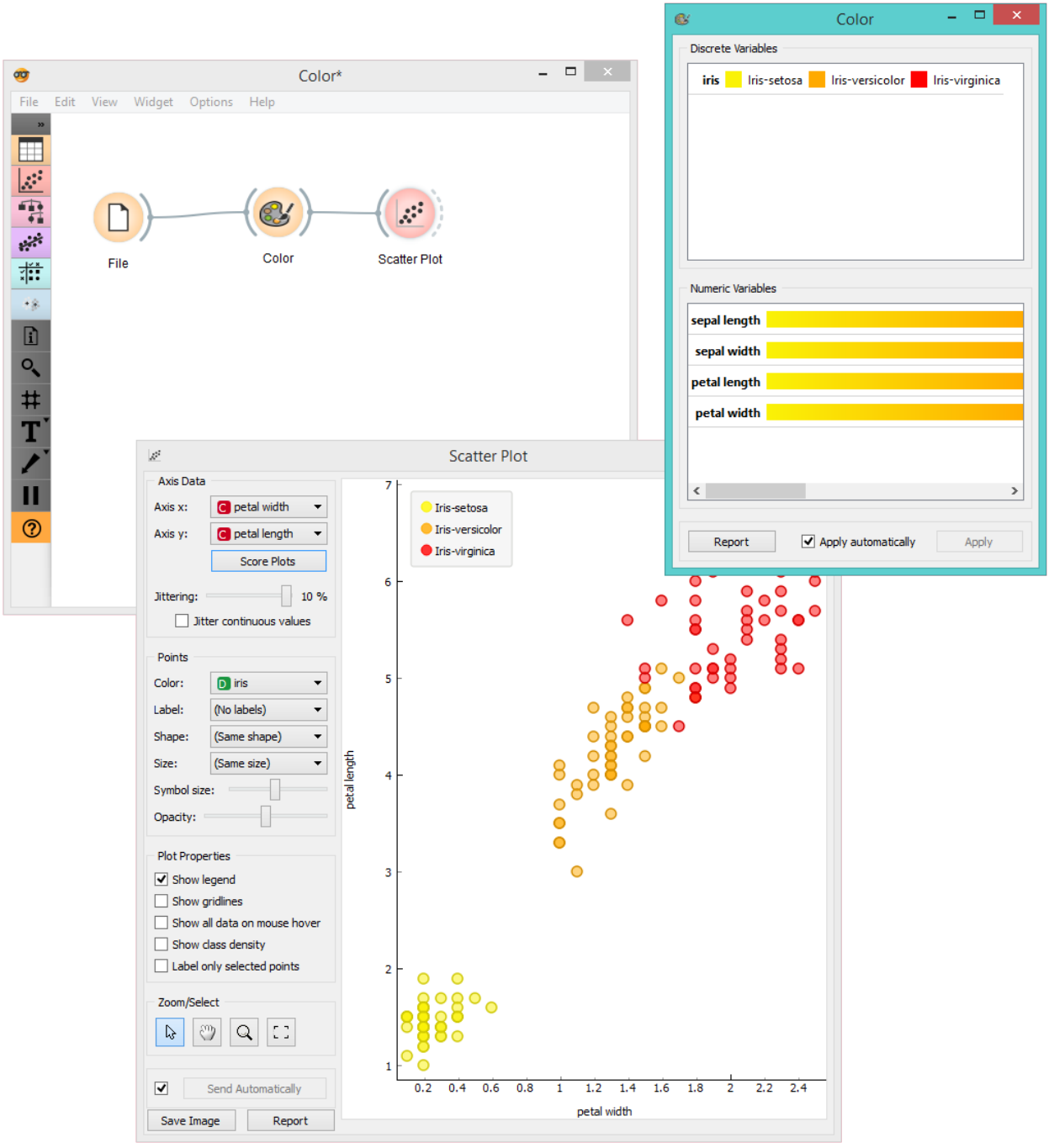


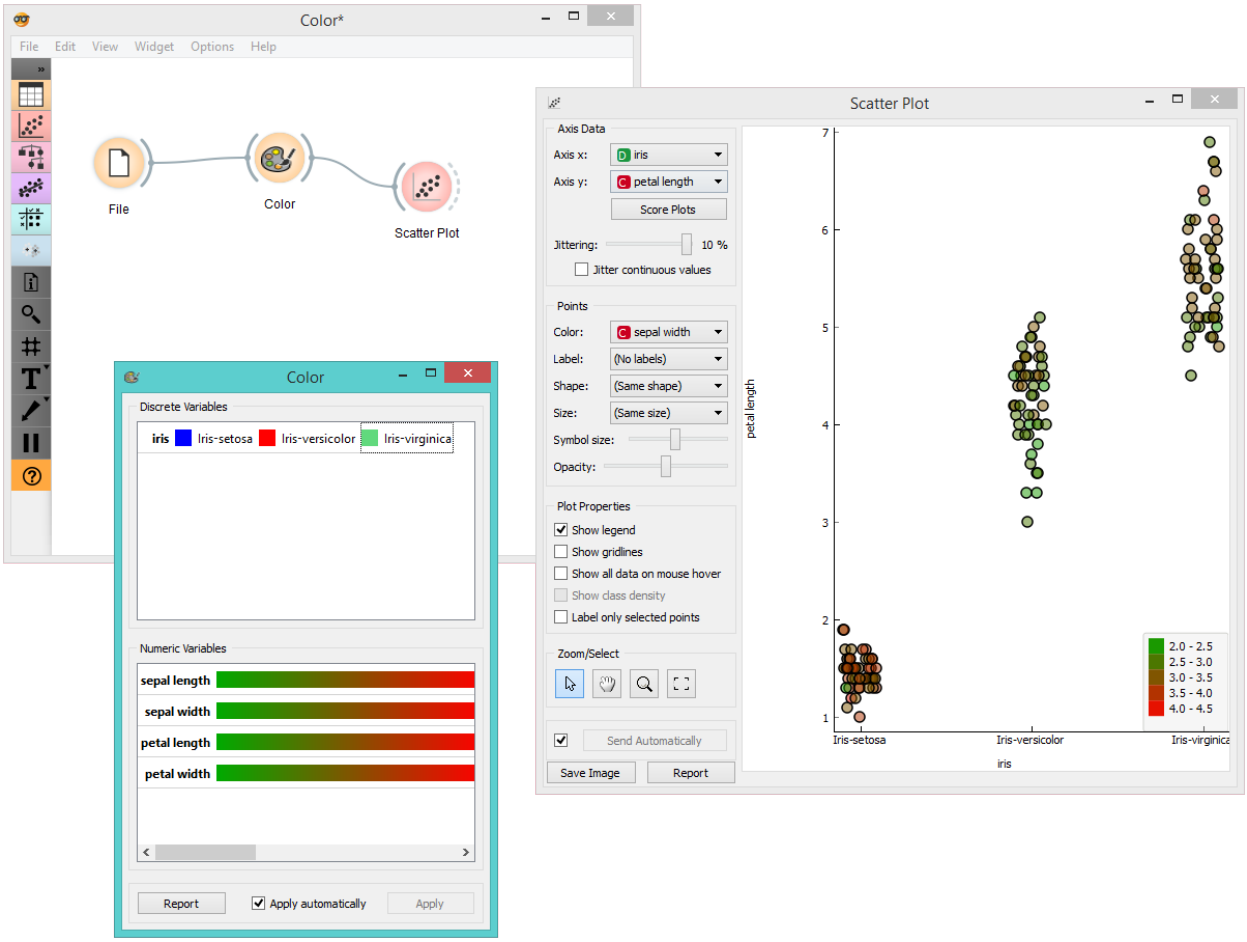
1. Choose a gradient from your saved profiles. The default profile is already set.
2. The gradient palette
3. Select the left side of the gradient. Double clicking the color opens the *Select Color* window.
4. Select the right side of the gradient. Double clicking the color opens the *Select Color* window.
5. Pass through black.
6. Click *OK* to save your choices or *Cancel* to exit the color palette.

Example

We chose to work with the *Iris* data set. We opened the color palette and selected three new colors for the three types of Irises. Then we opened the *Scatter Plot* widget and viewed the changes made to the scatter plot.

For our second example, we wished to demonstrate the use of the **Color** widget with continuous variables. We put different types of Irises on the x axis and petal length on the y axis. We created a new color gradient and named it greed (green + red). In order to show that sepal length is not a deciding factor in differentiating between different types of Irises, we chose to color the points according to sepal width.


Color*
File Edit View Widget Options Help
File
Color
Scatter Plot
Color
Discrete Variables
iris Iris-setosa Iris-versicolor Iris-virginica
Numeric Variables
sepal length
sepal width
petal length
petal width
Report
Apply automatically
Apply
Scatter Plot
Axis Data
Axis x: petal width
Axis y: petal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal length
petal width
1 2 3 4 5 6 7
0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4


Color*
File Edit View Widget Options Help
File
Color
Scatter Plot
Color
Discrete Variables
iris Iris-setosa Iris-versicolor Iris-virginica
Numeric Variables
sepal length
sepal width
petal length
petal width
Report
Apply automatically
Apply
Scatter Plot
Axis Data
Axis x: iris
Axis y: petal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: sepal width
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
petal length
iris
Iris-setosa
Iris-versicolor
Iris-virginica
2.0 - 2.5
2.5 - 3.0
3.0 - 3.5
3.5 - 4.0
4.0 - 4.5

Visualize

Box Plot

Shows distribution of attribute values.

Signals

Inputs:

- **Data**

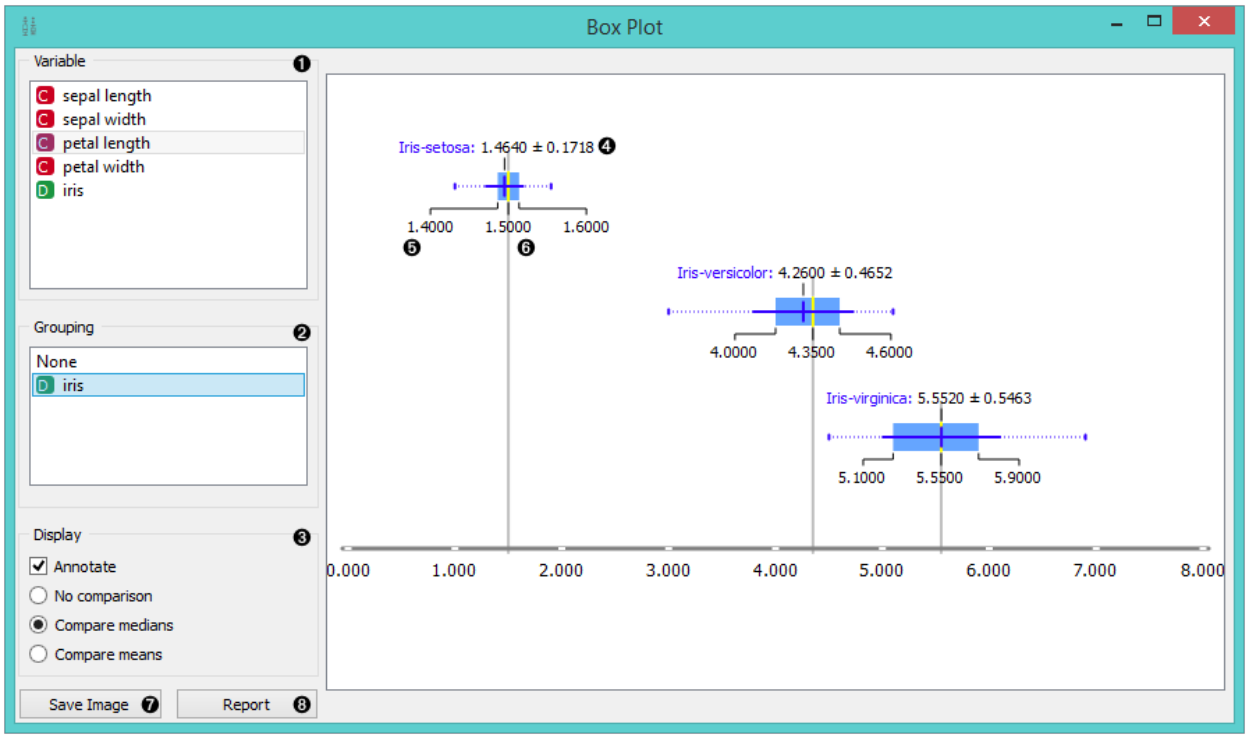
  An input data set

Outputs:

- (None)

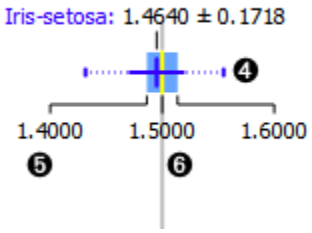
Description

The **Box Plot** widget shows the distributions of attribute values. It is a good practice to check any new data with this widget to quickly discover any anomalies, such as duplicated values (e.g. gray and grey), outliers, and alike.



1. Select the variable you want to see plotted.

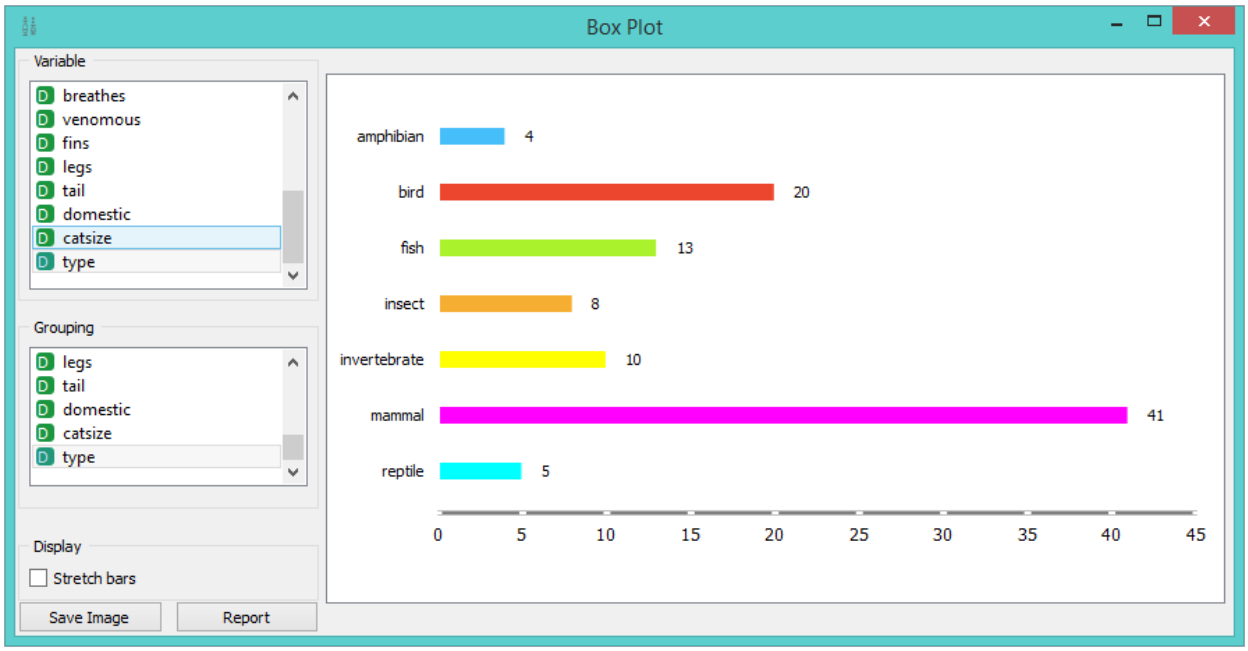
2. Choose *Grouping* to see box plots displayed by class.
3. When instances are grouped by class, you can change the display mode. Annotated boxes will display the end values, the mean and the median, while compare medians and compare means will, naturally, compare the selected value between class groups.



For continuous attributes the widget displays:

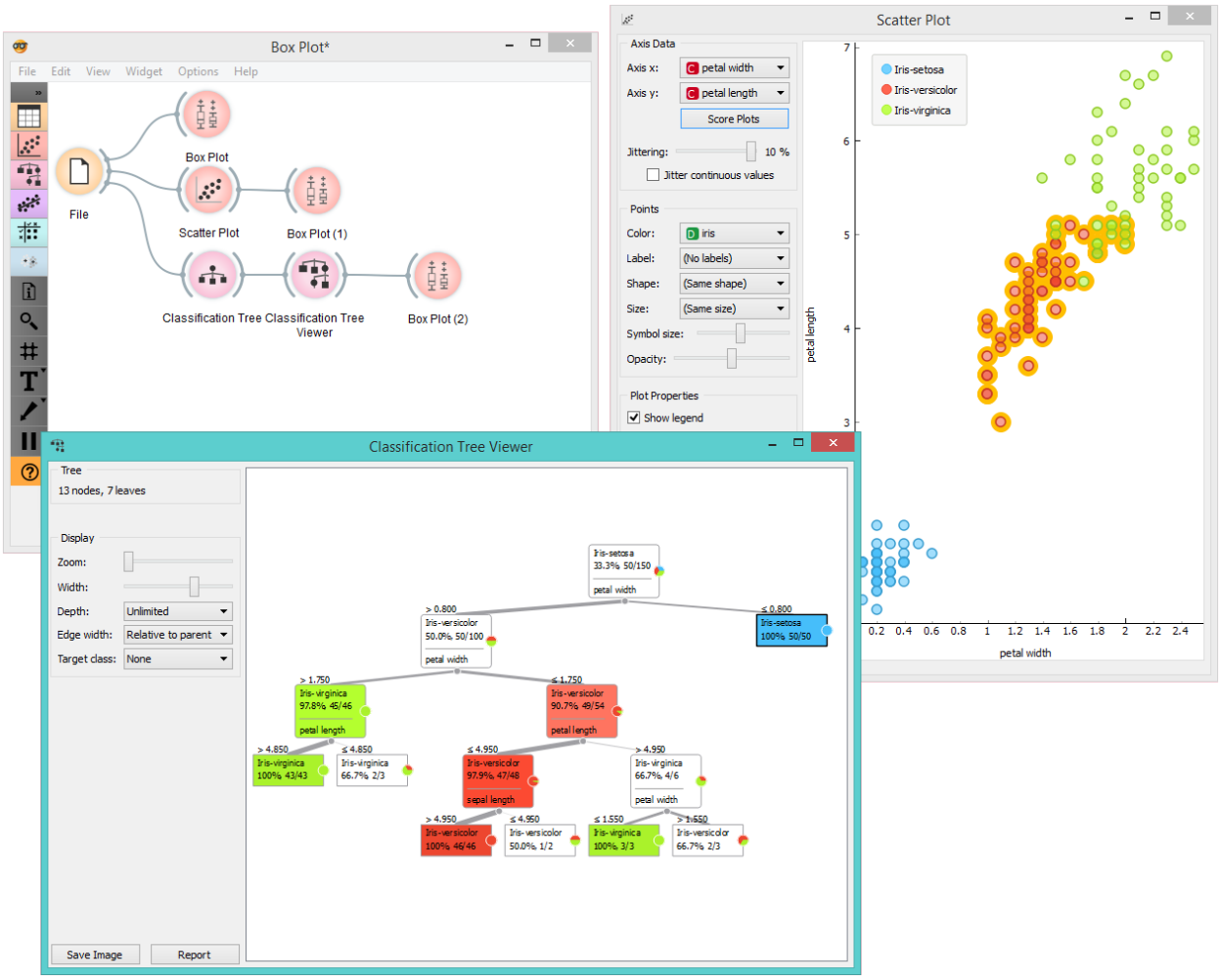
4. The mean (the dark blue vertical line)
5. Border values for the standard deviation of the mean. The blue highlighted area is the entire standard deviation of the mean.
6. The median (yellow vertical line). The thin blue line represents the area between the first (25%) and the third (75%) quantile, while the thin dotted line represents the entire range of values (from the lowest to the highest value in the data set for the selected parameter).
7. Save image.
8. Produce a report.

For discrete attributes, the bars represent the number of instances with each particular attribute value. The plot shows the number of different animal types in the *Zoo* data set: there are 41 mammals, 13 fish, 20 birds and so on.



Example

The **Box Plot** widget is most commonly used immediately after the *File* widget to observe the statistical properties of a data set. It is also useful for finding the properties of a specific data set, for instance a set of instances manually defined in another widget (e.g. *Scatterplot*) or instances belonging to some cluster or a classification tree node, as shown in the schema below.



Distributions

Displays value distributions for a single attribute.

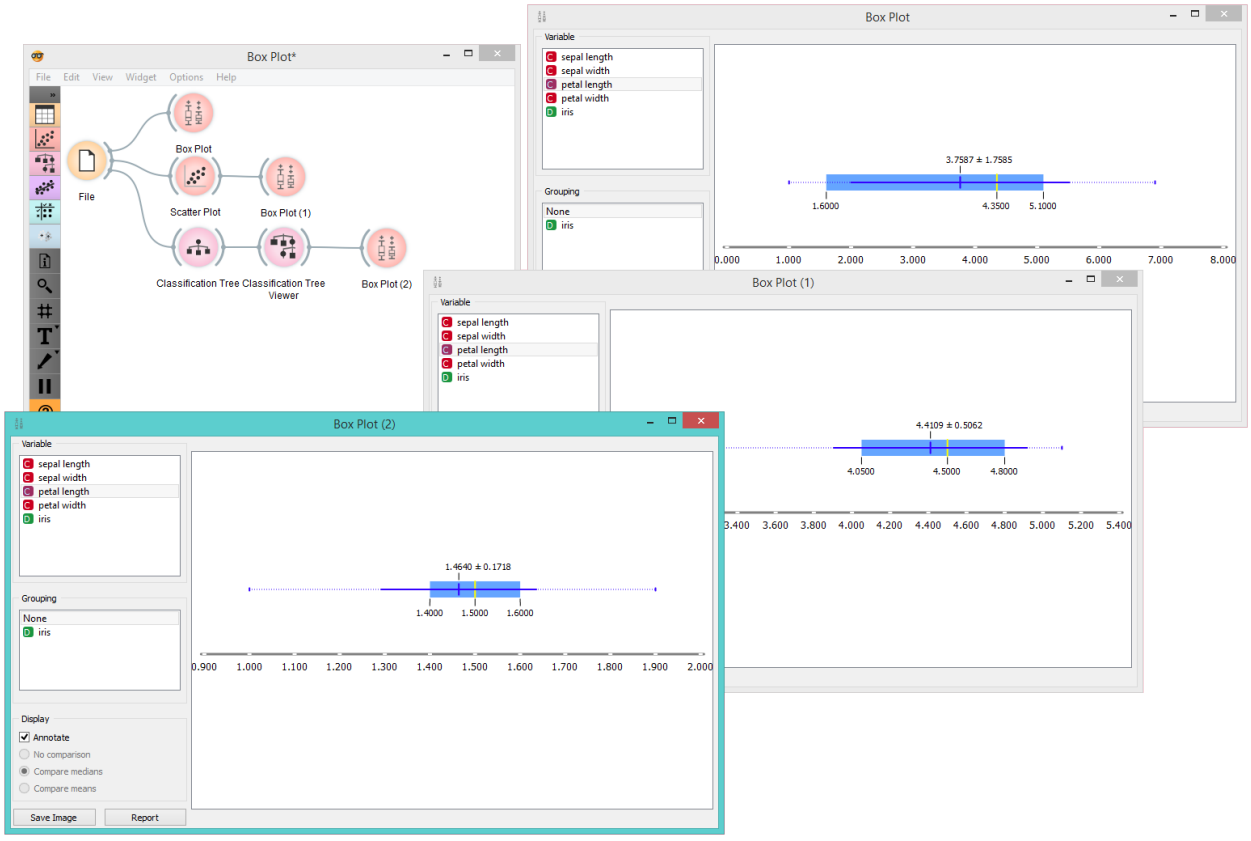
Signals

Inputs:

- **Data**

  An input data set.

Outputs:

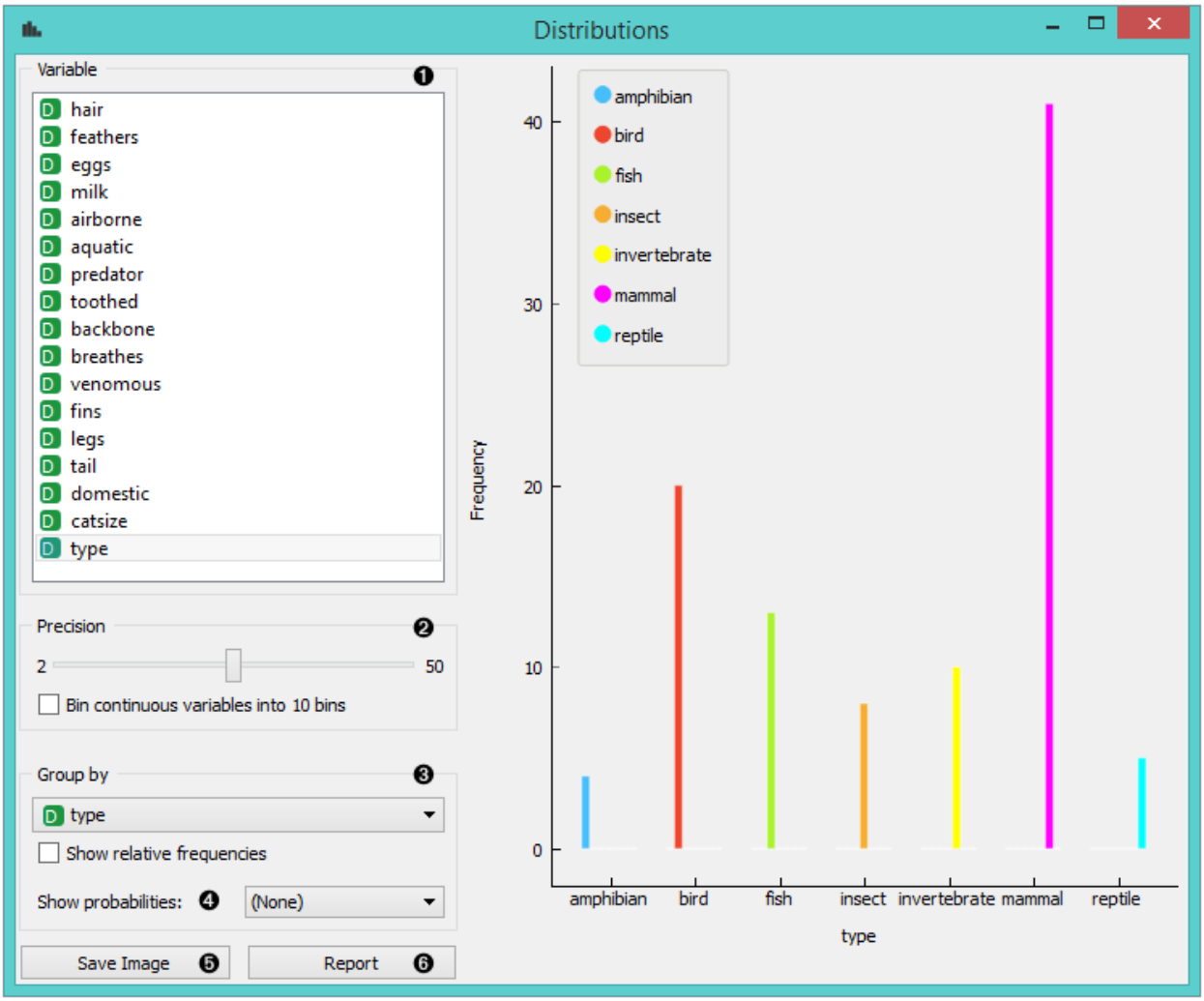

Box Plot*
File Edit View Widget Options Help
File
Box Plot
Scatter Plot
Box Plot (1)
Classification Tree
Classification Tree Viewer
Box Plot (2)
Box Plot
Variable
sepal length
sepal width
petal length
petal width
iris
Grouping
None
iris
3.7587 ± 1.7585
1.6000
4.3500
5.1000
0.000 1.000 2.000 3.000 4.000 5.000 6.000 7.000 8.000
Box Plot (1)
4.4109 ± 0.5062
4.0500
4.5000
4.8000
3.400 3.600 3.800 4.000 4.200 4.400 4.600 4.800 5.000 5.200 5.400
Box Plot (2)
Display
Annotate
No comparison
Compare medians
Compare means
Save Image
Report
1.4640 ± 0.1718
1.4000
1.5000
1.6000
0.900 1.000 1.100 1.200 1.300 1.400 1.500 1.600 1.700 1.800 1.900 2.000

- (None)

Description

The **Distributions** widget displays the value distribution of discrete or continuous attributes. If the data contains a class variable, distributions may be conditioned on the class.

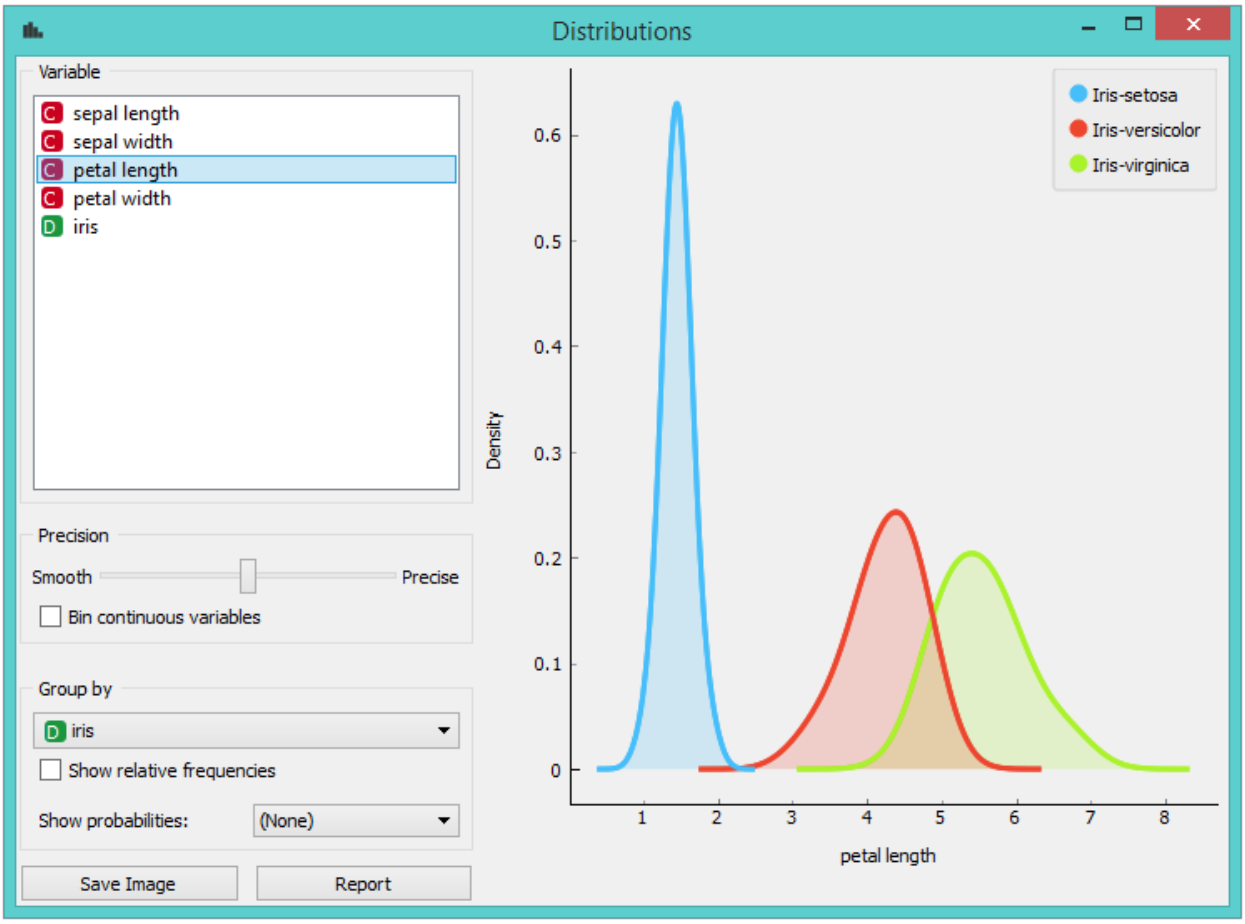
For discrete attributes, the graph displayed by the widget shows how many times (e.g., in how many instances) each attribute value appears in the data. If the data contains a class variable, class distributions for each of the attribute values will be displayed as well (like in the snapshot below). In order to create this graph, we used the *Zoo* data set.



1. A list of variables for distributions display
2. If *Bin continuous variables* is ticked, the widget will discretize continuous variables by assigning them to intervals. The number of intervals is set by precision scale. Alternatively, you can set smoothness for the distribution curves of continuous variables.
3. The widget may be requested to display value distributions only for instances of certain class (*Group by*). *Show relative frequencies* will scale the data by percentage of the data set.
4. Show probabilities.

5. *Save image* saves the graph to your computer in a .svg or .png format.
6. Produce a report.

For continuous attributes, the attribute values are displayed as a function graph. Class probabilities for continuous attributes are obtained with gaussian kernel density estimation, while the appearance of the curve is set with the *Precision* bar (smooth or precise). For the purpose of this example, we used the *Iris* data set.



In class-less domains, the bars are displayed in gray. Here we set *Bin continuous variables into 10 bins*, which distributes variables into 10 intervals and displays averages of these intervals as histograms (see 2. above). We used the *Housing* data set.

Heat Map

Plots a heat map for a pair of attributes.

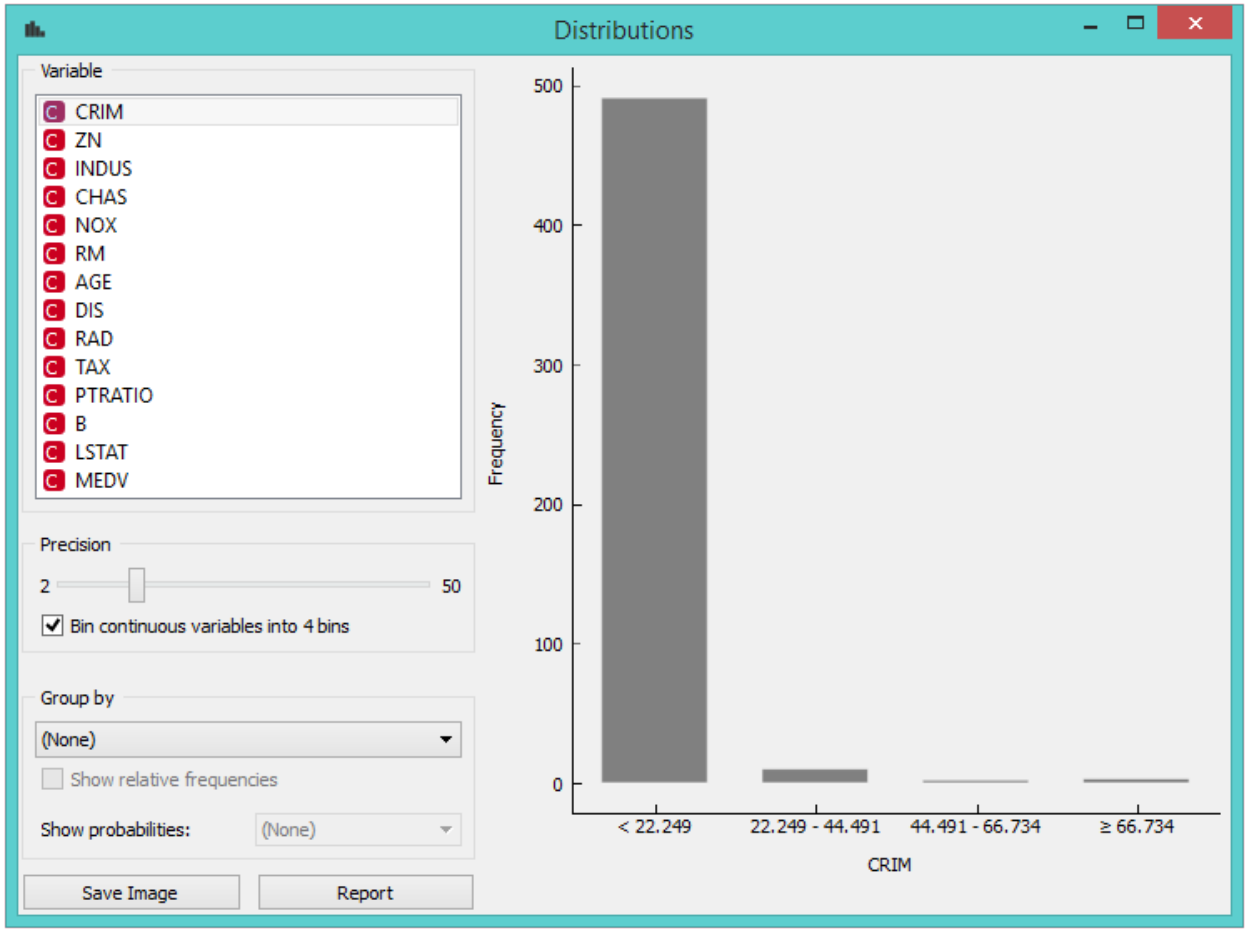
Signals

Inputs:

- **Data**

  An input data set.

Outputs:


Distributions
Variable
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
MEDV
Precision
2
50
Bin continuous variables into 4 bins
Group by
(None)
Show relative frequencies
Show probabilities:
(None)
Save Image
Report
Frequency
500
400
300
200
100
0
< 22.249
22.249 - 44.491
44.491 - 66.734
≥ 66.734
CRIM

- **Selected Data**

  A subset of instances that the user has manually selected from the map.

Description

Heat map is a graphical method for visualizing attribute values by class in a two-way matrix. It only works on data sets containing continuous variables. The values are represented by color: the higher a certain value is, the darker the represented color. By combining class and attributes on x and y axes, we see where the attribute values are the strongest and where the weakest, thus enabling us to find typical features (discrete) or value range (continuous) for each class.

1. The color scheme legend. **Low** and **High** are thresholds for the color palette (low for attributes with low values and high for attributes with high values).
2. Merge data.
3. Sort columns and rows: - **No Sorting** (lists attributes as found in the data set) - **Clustering** (clusters data by similarity) - **Clustering with ordered leaves** (maximizes the sum of similarities of adjacent elements)
4. Set what is displayed in the plot in **Annotation & Legend**. - If *Show legend* is ticked, a color chart will be displayed above the map. - If *Stripes with averages* is ticked, a new line with attribute averages will be displayed on the left. - **Row Annotations** adds annotations to each instance on the right. - **Column Label Positions** places column labels in a selected place (None, Top, Bottom, Top and Bottom).
5. If *Keep aspect ratio* is ticked, each value will be displayed with a square (proportionate to the map).
6. If *Send Automatically* is ticked, changes are communicated automatically. Alternatively, click *Send*.
7. *Save image* saves the image to your computer in a .svg or .png format.
8. Produce a report.

Example

The **Heat Map** below displays attribute values for the *Housing* data set. The aforementioned data set concerns the housing values in the suburbs of Boston. The first thing we see in the map are the 'B' and 'Tax' attributes, which are the only two colored in dark orange. The 'B' attribute provides information on the proportion of blacks by town and the 'Tax' attribute informs us about the full-value property-tax rate per $10,000. In order to get a clearer heat map, we then use the *Select Columns* widget and remove the two attributes from the data set. Then we again feed the data to the **Heat map**. The new projection offers additional information. By removing 'B' and 'Tax', we can see other deciding factors, namely 'Age' and 'ZN'. The 'Age' attribute provides information on the proportion of owner-occupied units built prior to 1940 and the 'ZN' attribute informs us about the proportion of non-retail business acres per town.

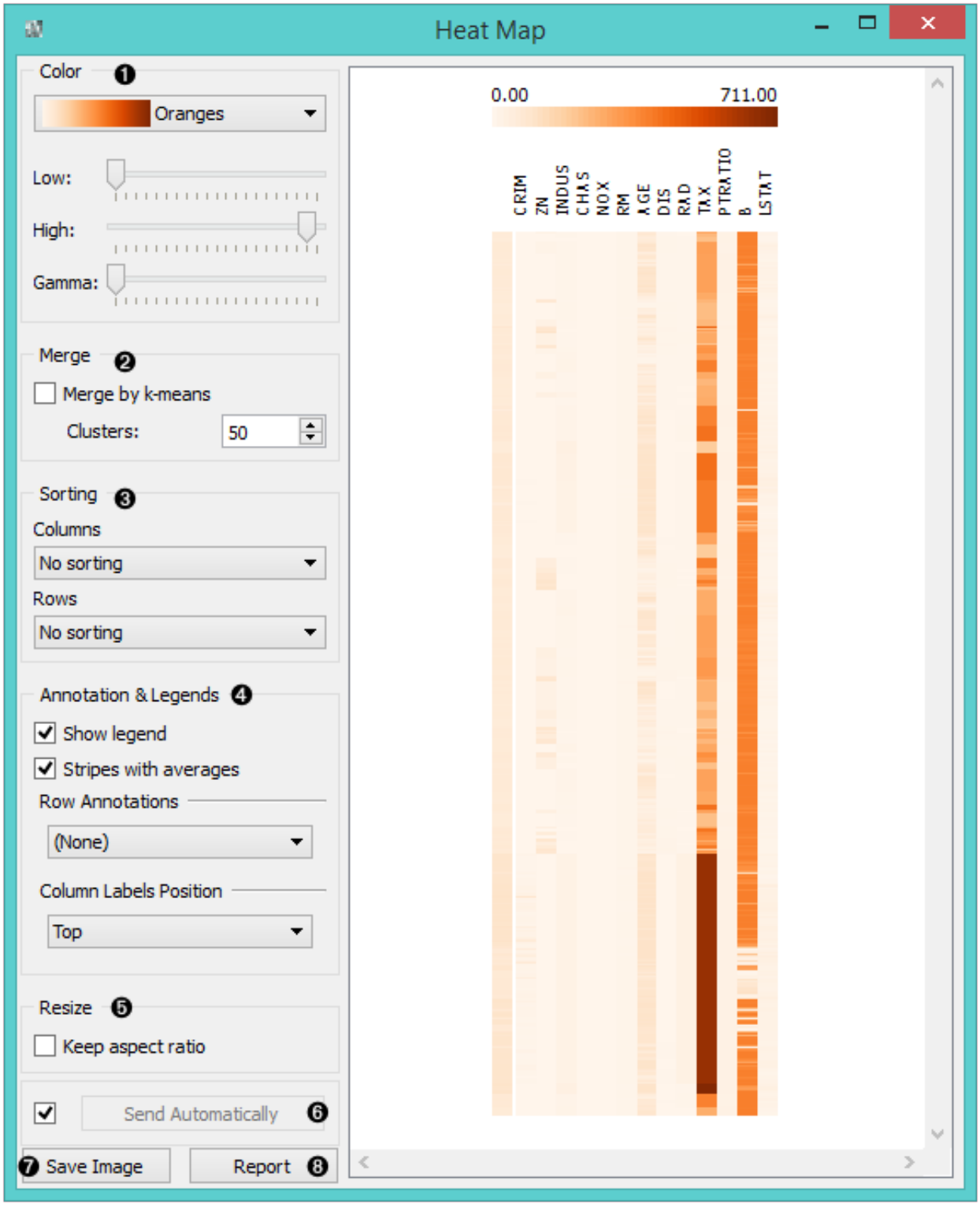
The **Heat Map** widget is a nice tool for discovering relevant features in the data. By removing some of the more pronounced features, we came across new information, which was hiding in the background.

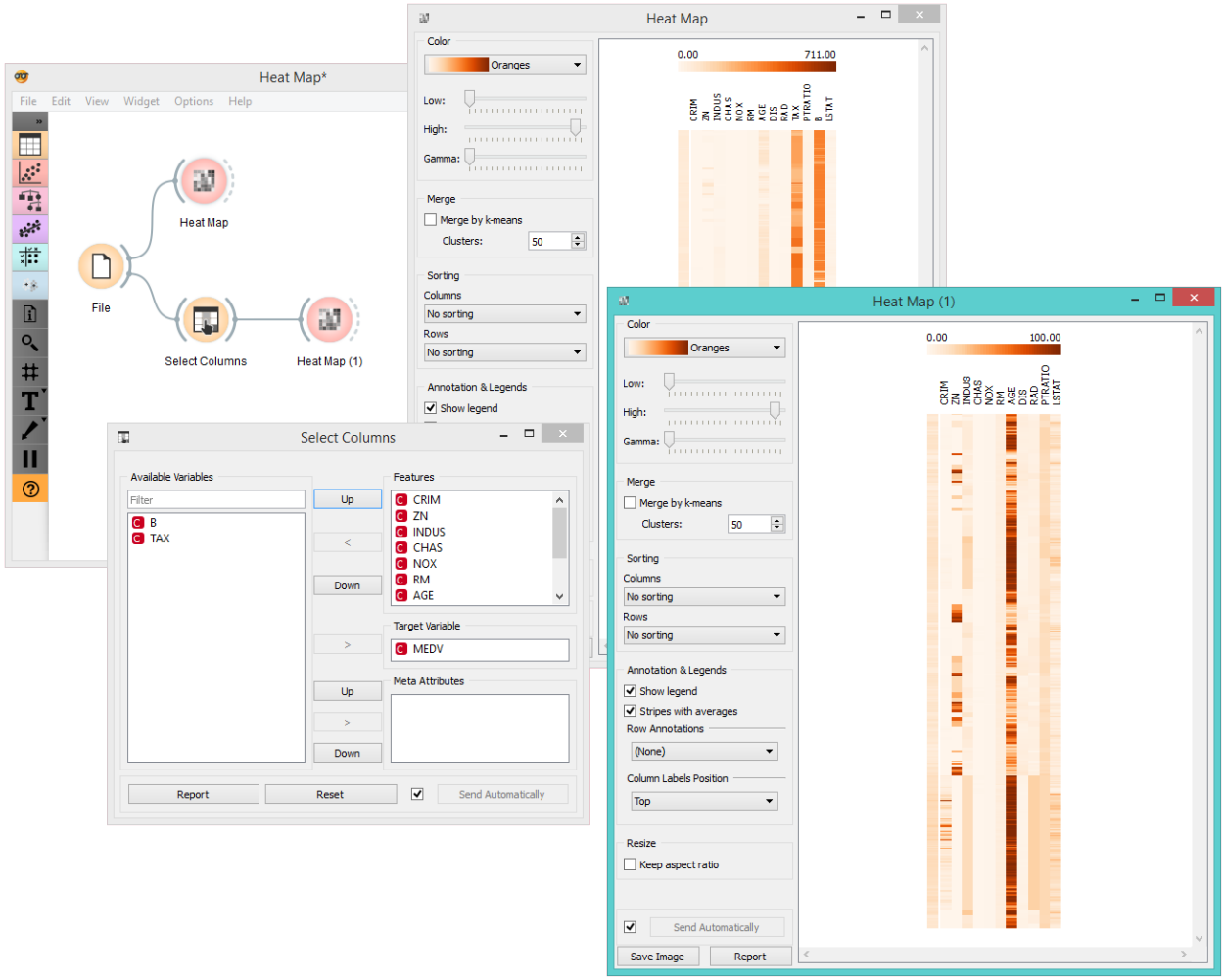
References

Housing Data Set

Scatter Plot

Scatterplot visualization with explorative analysis and intelligent data visualization enhancements.


Heat Map
Color
Oranges
Low:
High:
Gamma:
Merge
Merge by k-means
Clusters:
50
Sorting
Columns
No sorting
Rows
No sorting
Annotation & Legends
Show legend
Stripes with averages
Row Annotations
(None)
Column Labels Position
Top
Resize
Keep aspect ratio
Send Automatically
Save Image
Report
0.00
711.00
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT


Heat Map*
File Edit View Widget Options Help
Heat Map
File
Select Columns
Heat Map (1)
Heat Map
Color
Oranges
Low:
High:
Gamma:
Merge
Merge by k-means
Clusters: 50
Sorting
Columns
No sorting
Rows
No sorting
Annotation & Legends
Show legend
0.00
711.00
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT
Select Columns
Available Variables
Filter
B
TAX
Up
Down
Features
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
Target Variable
MEDV
Meta Attributes
Report
Reset
Send Automatically
Heat Map (1)
Color
Oranges
Low:
High:
Gamma:
Merge
Merge by k-means
Clusters: 50
Sorting
Columns
No sorting
Rows
No sorting
Annotation & Legends
Show legend
Stripes with averages
Row Annotations
(None)
Column Labels Position
Top
Resize
Keep aspect ratio
Send Automatically
Save Image
Report
0.00
100.00
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO LSTAT

Signals

Inputs:

- **Data**

  An input data set.

- **Data Subset**

  A subset of instances from the input data set.

- **Features**

  A list of attributes.

Outputs:

- **Selected Data**

  A subset of instances that the user manually selected from the scatterplot.

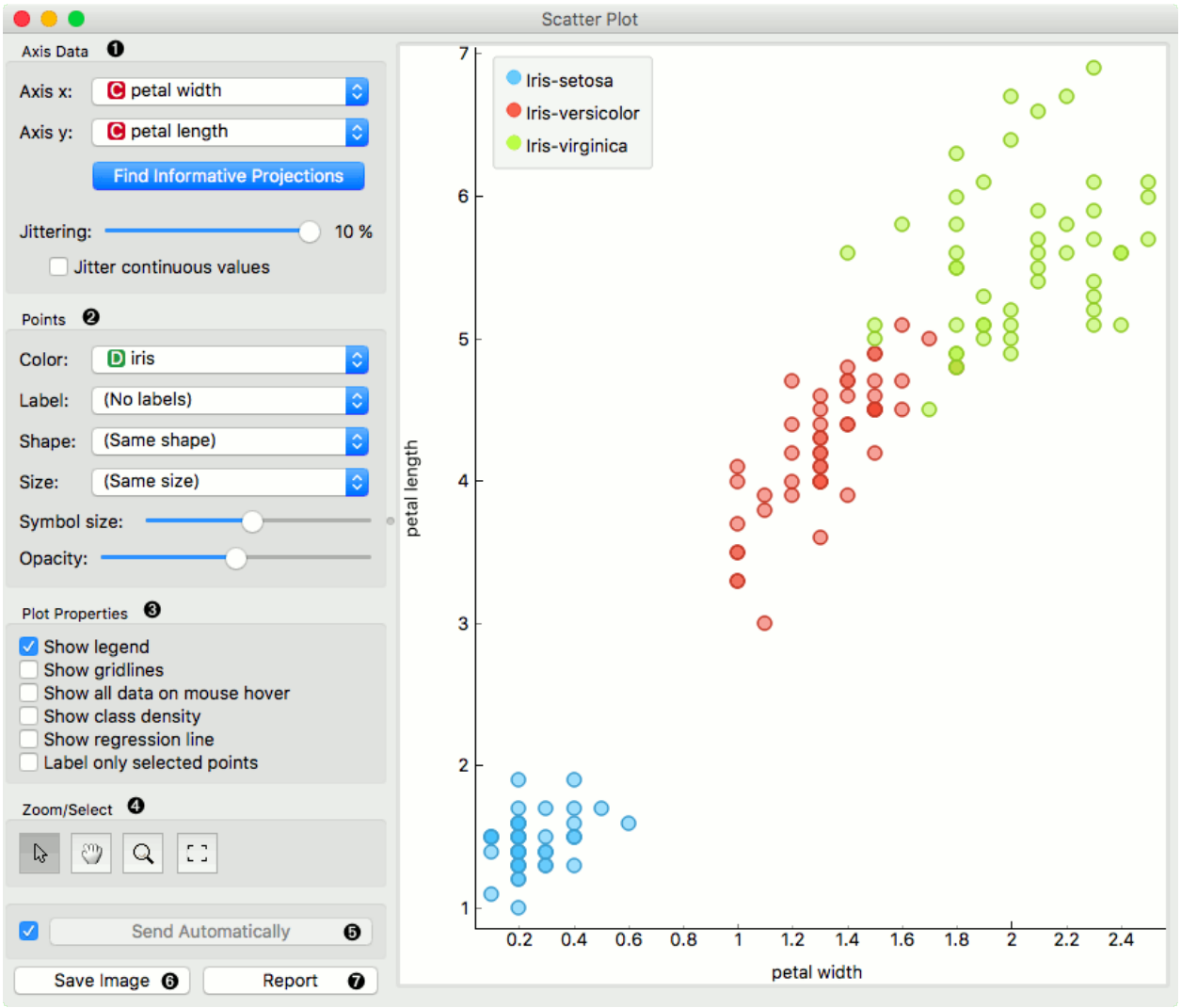
- **Data**

  Data with an additional column showing whether a point is selected.

Description

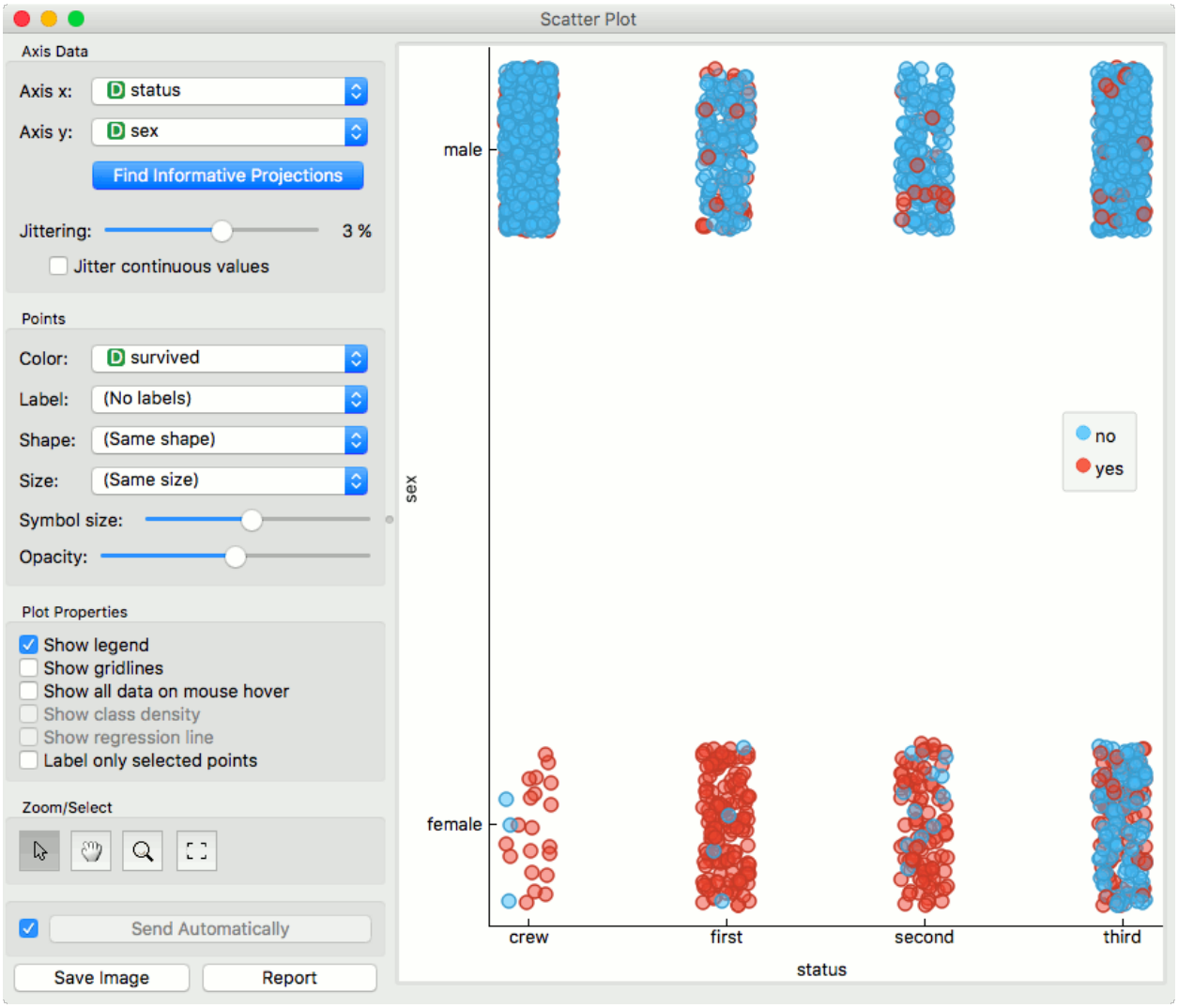
The **Scatterplot** widget provides a 2-dimensional scatterplot visualization for both continuous and discrete-valued attributes. The data is displayed as a collection of points, each having the value of the x-axis attribute determining the position on the horizontal axis and the value of the y-axis attribute determining the position on the vertical axis. Various properties of the graph, like color, size and shape of the points, axis titles, maximum point size and jittering can be adjusted on the left side of the widget. A snapshot below shows the scatterplot of the *Iris* data set with the coloring matching of the class attribute.

1. Select the x and y attribute. Optimize your projection by using **Rank Projections**. This feature scores attribute pairs by average classification accuracy and returns the top scoring pair with a simultaneous visualization update. Set jittering to prevent the dots overlapping. If *Jitter continuous values* is ticked, continuous instances will be dispersed.
2. Set the color of the displayed points (you will get colors for discrete values and grey-scale points for continuous). Set label, shape and size to differentiate between points. Set symbol size and opacity for all data points. Set the desired colors scale.
3. Adjust *plot properties*:
   - *Show legend* displays a legend on the right. Click and drag the legend to move it.
   - *Show gridlines* displays the grid behind the plot.
   - *Show all data on mouse hover* enables information bubbles if the cursor is placed on a dot.
   - *Show class density* colors the graph by class (see the screenshot below).
   - *Show regression line* draws the regression line for pair of continuous attributes.
   - *Label only selected points* allows you to select individual data instances and label them.
4. *Select, zoom, pan and zoom to fit* are the options for exploring the graph. The manual selection of data instances works as an angular/square selection tool. Double click to move the projection. Scroll in or out for zoom.
5. If *Send automatically* is ticked, changes are communicated automatically. Alternatively, press *Send*.
6. *Save Image* saves the created image to your computer in a .svg or .png format.


Scatter Plot
Axis Data
Axis x:
petal width
Axis y:
petal length
Find Informative Projections
Jittering:
10 %
Jitter continuous values
Points
Color:
iris
Label:
(No labels)
Shape:
(Same shape)
Size:
(Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal length
petal width
1
2
3
4
5
6
7
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
2.4

7. Produce a report.

For discrete attributes, jittering circumvents the overlap of points which have the same value for both axes, and therefore the density of points in the region corresponds better to the data. As an example, the scatterplot for the Titanic data set, reporting on the gender of the passengers and the traveling class is shown below; without jittering, the scatterplot would display only eight distinct points.



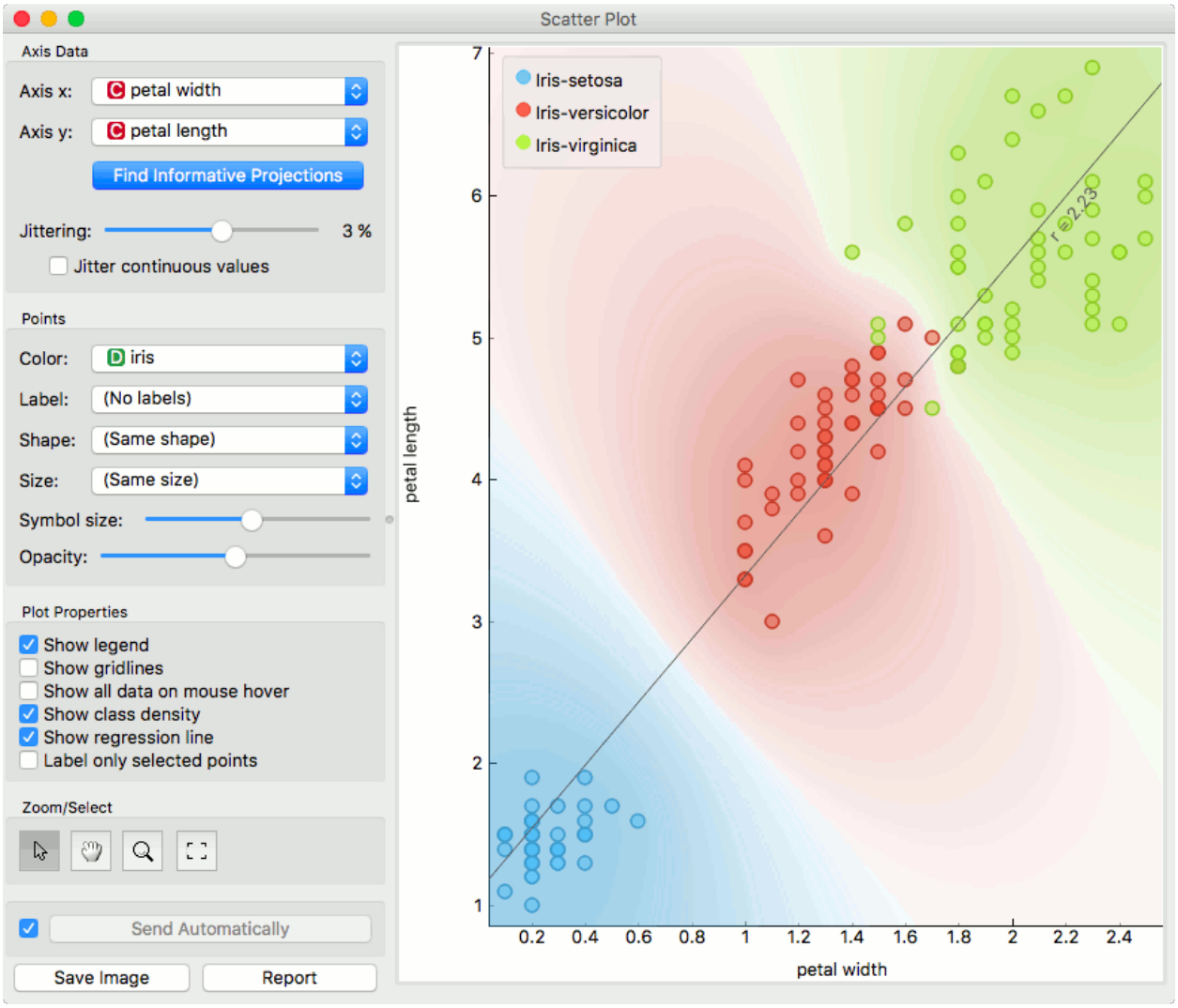
Here is an example of the **Scatter Plot** widget if the *Show class density* and *Show regression line* boxes are ticked.

Intelligent Data Visualization

If a data set has many attributes, it is impossible to manually scan through all the pairs to find interesting or useful scatterplots. Orange implements intelligent data visualization with the **Find Informative Projections** option in the widget. The goal of optimization is to find scatterplot projections where instances are well separated.

To use this method, go to the *Find Informative Projections* option in the widget, open the subwindow and press *Start Evaluation*. The feature will return a list of attribute pairs by average classification accuracy score.

Below, there is an example demonstrating the utility of ranking. The first scatterplot projection was set as the default sepal width to sepal length plot (we used the Iris data set for simplicity). Upon running *Find Informative Projections*


Scatter Plot
Axis Data
Axis x: petal width
Axis y: petal length
Find Informative Projections
Jittering: 3 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
r = 2.23
petal length
petal width
1
2
3
4
5
6
7
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
2.4

optimization, the scatterplot converted to a much better projection of petal width to petal length plot.

Selection

Selection can be used to manually defined subgroups in the data. Use Shift modifier when selecting data instances to put them into a new group. Shift + Ctrl (or Shift + Cmd on macOs) appends instances to the last group.

Signal data outputs a data table with an additional column that contains group indices.

Explorative Data Analysis

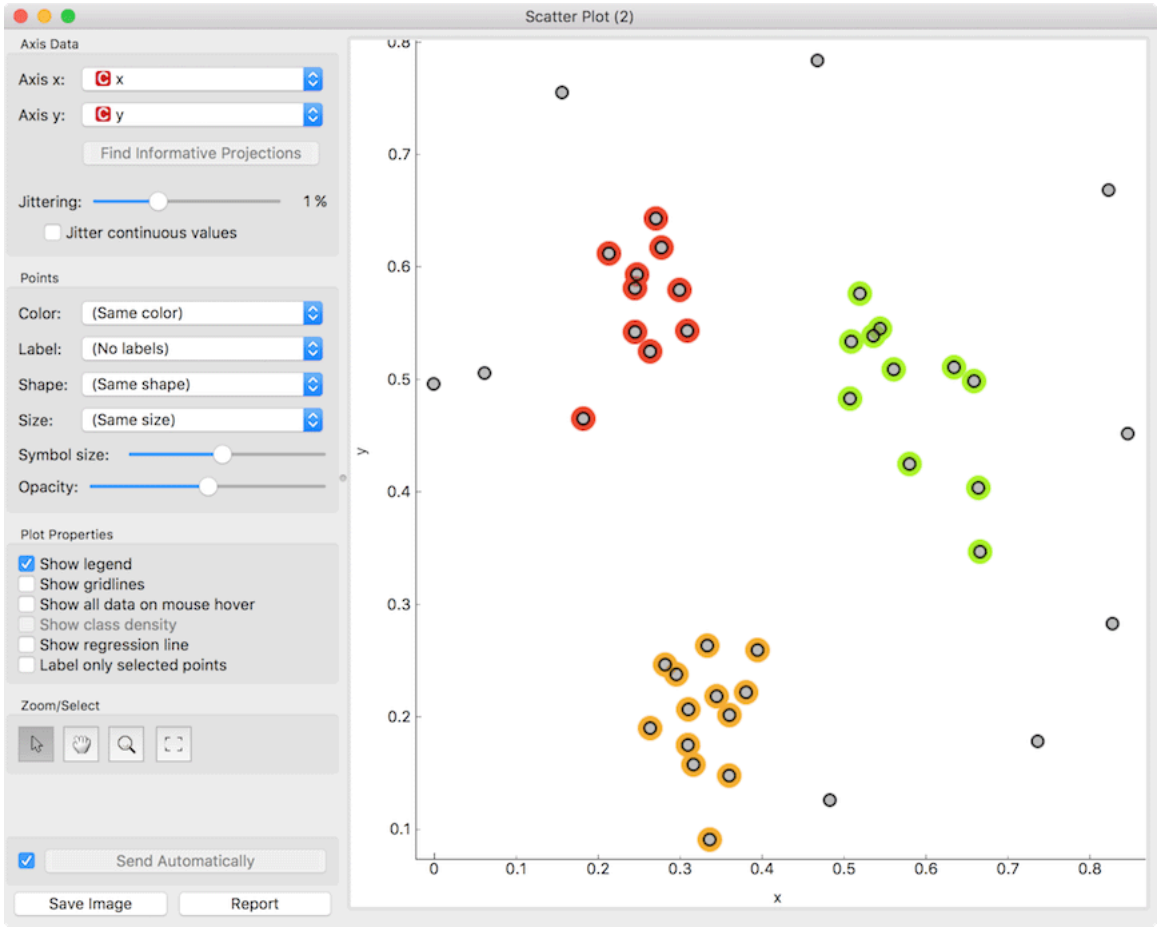
The **Scatterplot**, as the rest of Orange widgets, supports zooming-in and out of part of the plot and a manual selection of data instances. These functions are available in the lower left corner of the widget. The default tool is *Select*, which selects data instances within the chosen rectangular area. *Pan* enables you to move the scatterplot around the pane. With *Zoom* you can zoom in and out of the pane with a mouse scroll, while *Reset zoom* resets the visualization to its optimal size. An example of a simple schema, where we selected data instances from a rectangular region and sent them to the *Data Table* widget, is shown below. Notice that the scatterplot doesn't show all 52 data instances, because some data instances overlap (they have the same values for both attributes used).

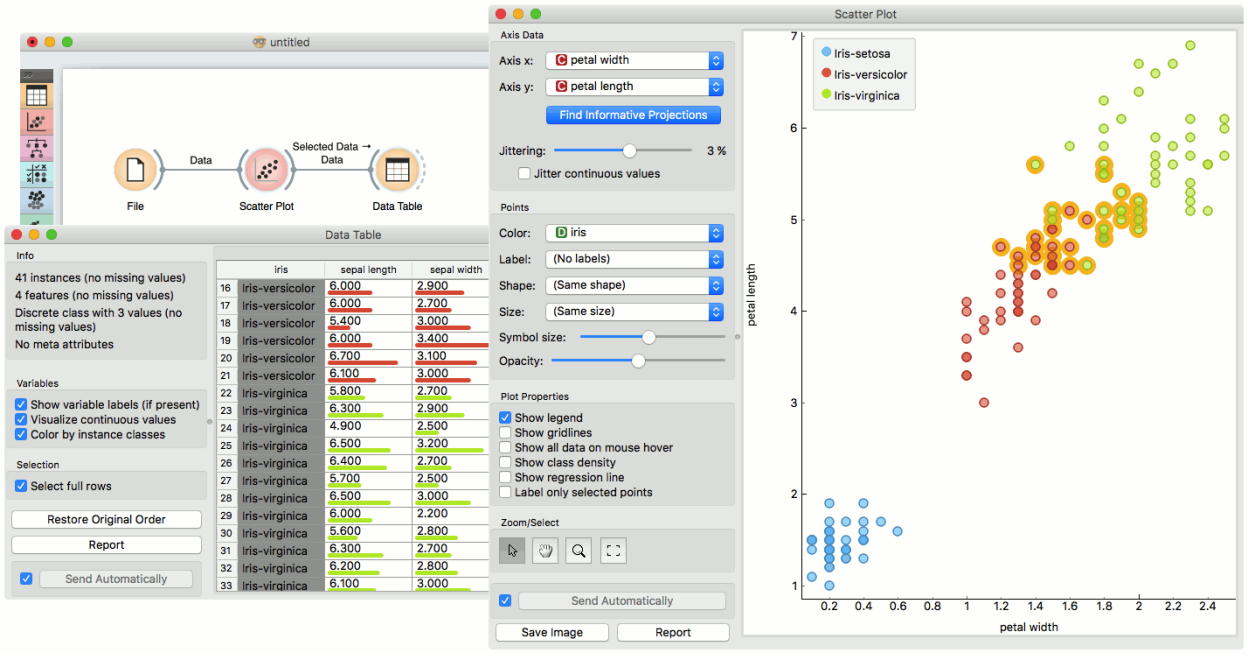
Example

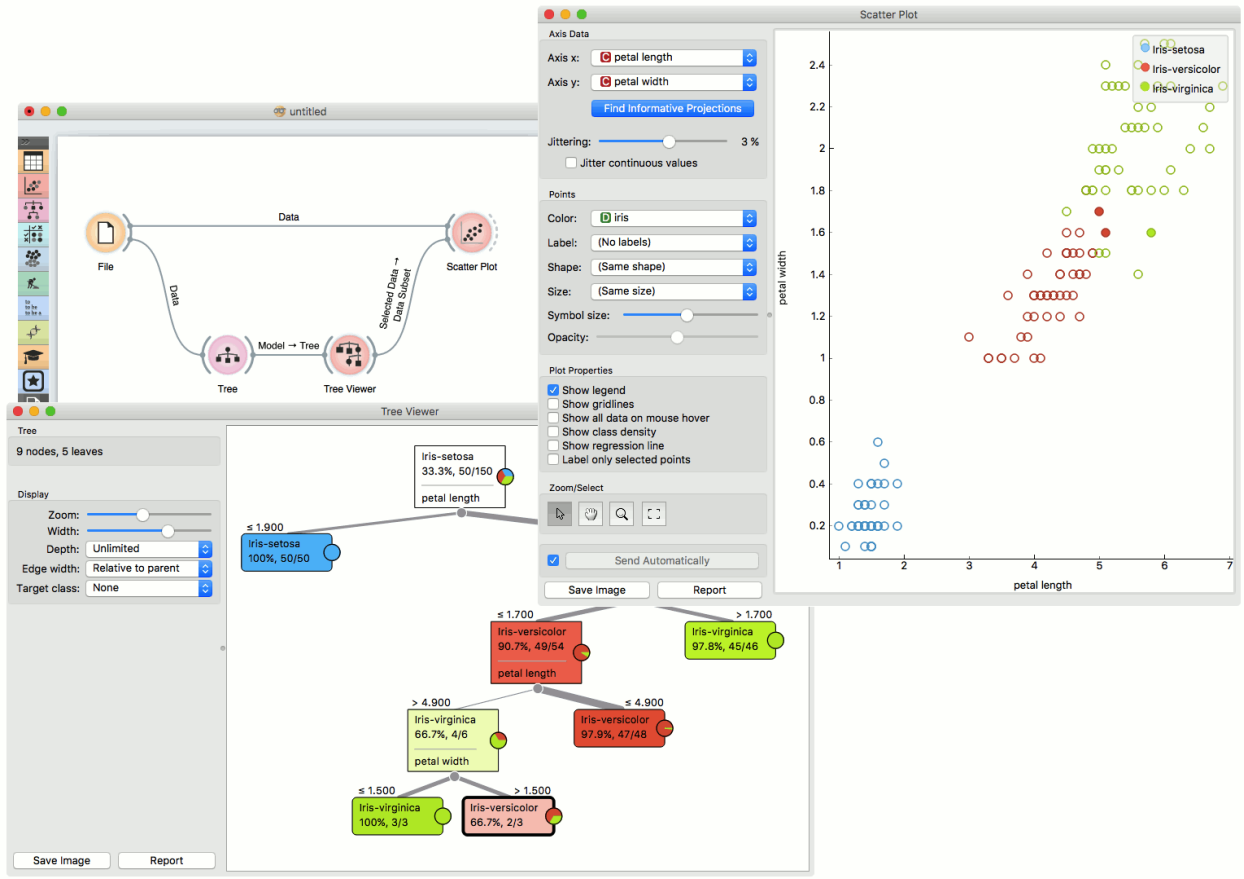
The **Scatterplot** can be combined with any widget that outputs a list of selected data instances. In the example below, we combine *Tree* and **Scatterplot** to display instances taken from a chosen decision tree node (clicking on any node of the tree will send a set of selected data instances to the scatterplot and mark selected instances with filled symbols).

Venn Diagram

Plots a Venn diagram for two or more data subsets.


Scatter Plot (2)
Axis Data
Axis x: x
Axis y: y
Find Informative Projections
Jittering: 1 %
Jitter continuous values
Points
Color: (Same color)
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
x
y


untitled
File
Data
Scatter Plot
Selected Data
Data
Data Table
Data Table
Info
41 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
Scatter Plot
Axis Data
Axis x: petal width
Axis y: petal length
Find Informative Projections
Jittering: 3 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal length
petal width


untitled
File
Data
Scatter Plot
Tree
Model → Tree
Tree Viewer
Selected Data → Data Subset
Tree Viewer
Tree
9 nodes, 5 leaves
Display
Zoom:
Width:
Depth: Unlimited
Edge width: Relative to parent
Target class: None
Save Image
Report
Iris-setosa
33.3%, 50/150
petal length
≤ 1.900
Iris-setosa
100%, 50/50
≤ 1.700
Iris-versicolor
90.7%, 49/54
petal length
> 1.700
Iris-virginica
97.8%, 45/46
> 4.900
Iris-virginica
66.7%, 4/6
petal width
≤ 4.900
Iris-versicolor
97.9%, 47/48
≤ 1.500
Iris-virginica
100%, 3/3
> 1.500
Iris-versicolor
66.7%, 2/3
Scatter Plot
Axis Data
Axis x: petal length
Axis y: petal width
Find Informative Projections
Jittering: 3 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal width
petal length

Signals

Inputs:

- **Data**

  An input data set

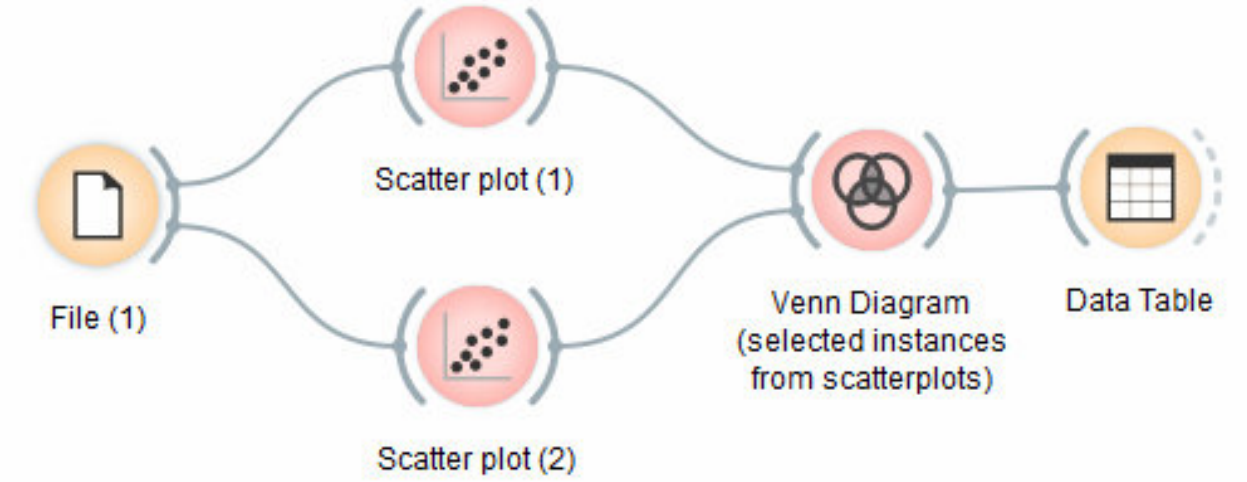
Outputs:

- **Selected Data**

  A subset of instances that the user has manually selected from the diagram.

Description

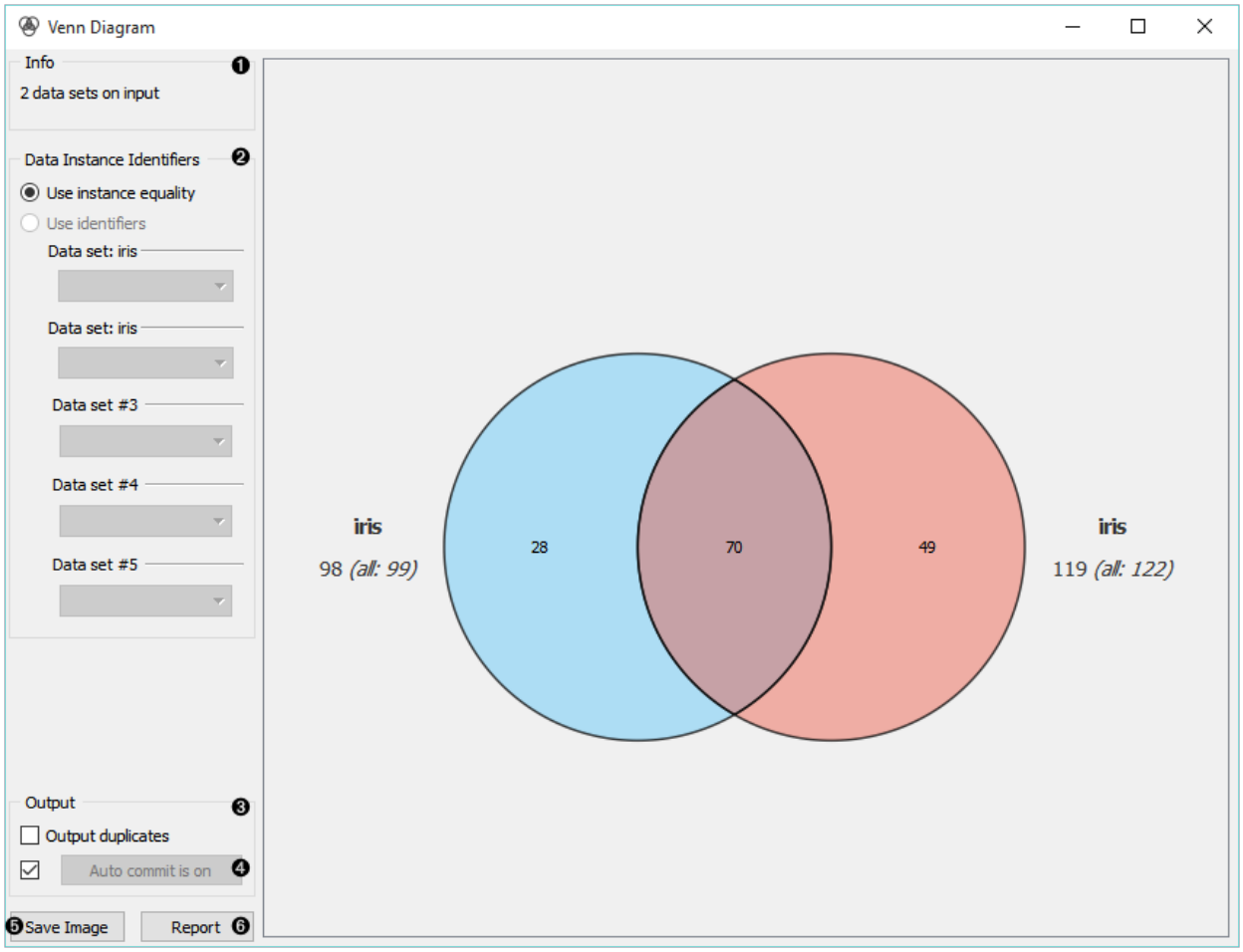
The **Venn Diagram** widget displays logical relations between data sets. This projection shows two or more data sets represented by circles of different colors. The intersections are subsets that belong to more than one data set. To further analyze or visualize the subset, click on the intersection.

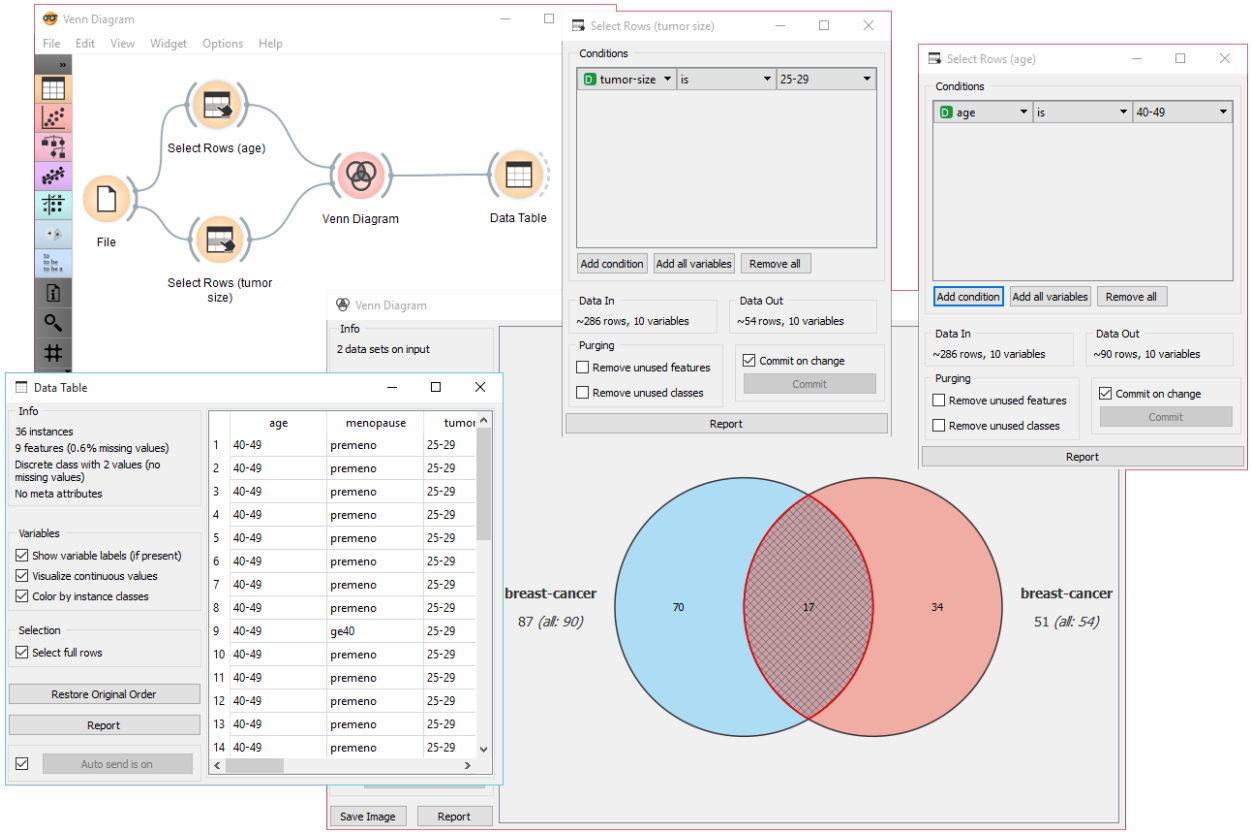


1. Information on the input data.
2. Select the identifiers by which to compare the data.
3. Tick *Output duplicates* if you wish to remove duplicates.
4. If *Auto commit* is on, changes are automatically communicated to other widgets. Alternatively, click *Commit*.
5. *Save Image* saves the created image to your computer in a .svg or .png format.
6. Produce a report.

Examples

The easiest way to use the **Venn Diagram** is to select data subsets and find matching instances in the visualization. We use the *breast-cancer* data set to select two subsets with *Select Rows* widget - the first subset is that of breast cancer patients aged between 40 and 49 and the second is that of patients with a tumor size between 20 and 29. The **Venn Diagram** helps us find instances that correspond to both criteria, which can be found in the intersection of the two circles.


Venn Diagram
Info
2 data sets on input
Data Instance Identifiers
Use instance equality
Use identifiers
Data set: iris
Data set: iris
Data set #3
Data set #4
Data set #5
Output
Output duplicates
Auto commit is on
Save Image
Report
iris
98 (all: 99)
28
70
49
iris
119 (all: 122)



The **Venn Diagram** widget can be also used for exploring different prediction models. In the following example, we analysed 3 prediction methods, namely *Naive Bayes*, *SVM* and *Random Forest*, according to their misclassified instances. By selecting misclassifications in the three *Confusion Matrix* widgets and sending them to Venn diagram, we can see all the misclassification instances visualized per method used. Then we open **Venn Diagram** and select, for example, the misclassified instances that were identified by all three methods (in our case 2). This is represented as an intersection of all three circles. Click on the intersection to see this two instances marked in the *Scatterplot* widget. Try selecting different diagram sections to see how the scatterplot visualization changes.

Linear Projection

A linear projection method with explorative data analysis.

Signals

Inputs:

- **Data**

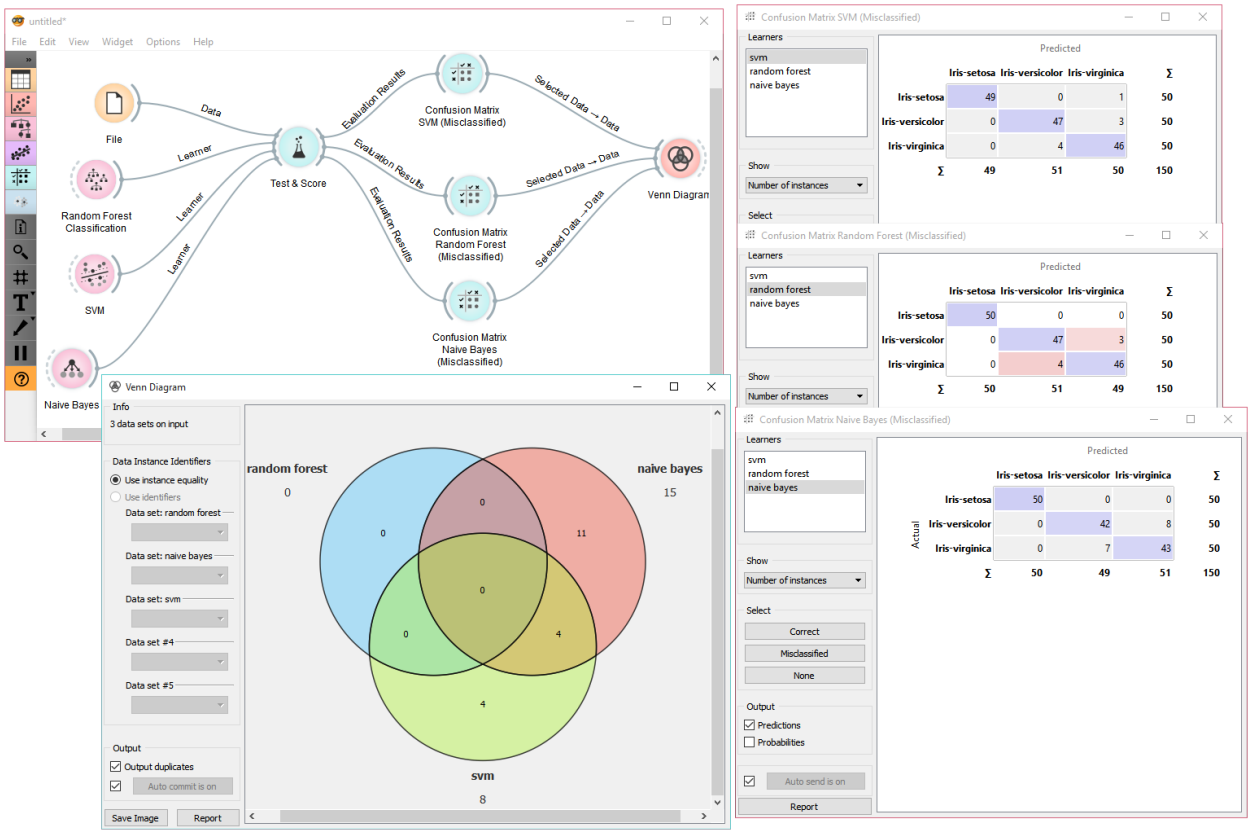
  An input data set

- **Data Subset**

  A subset of data instances

Outputs:

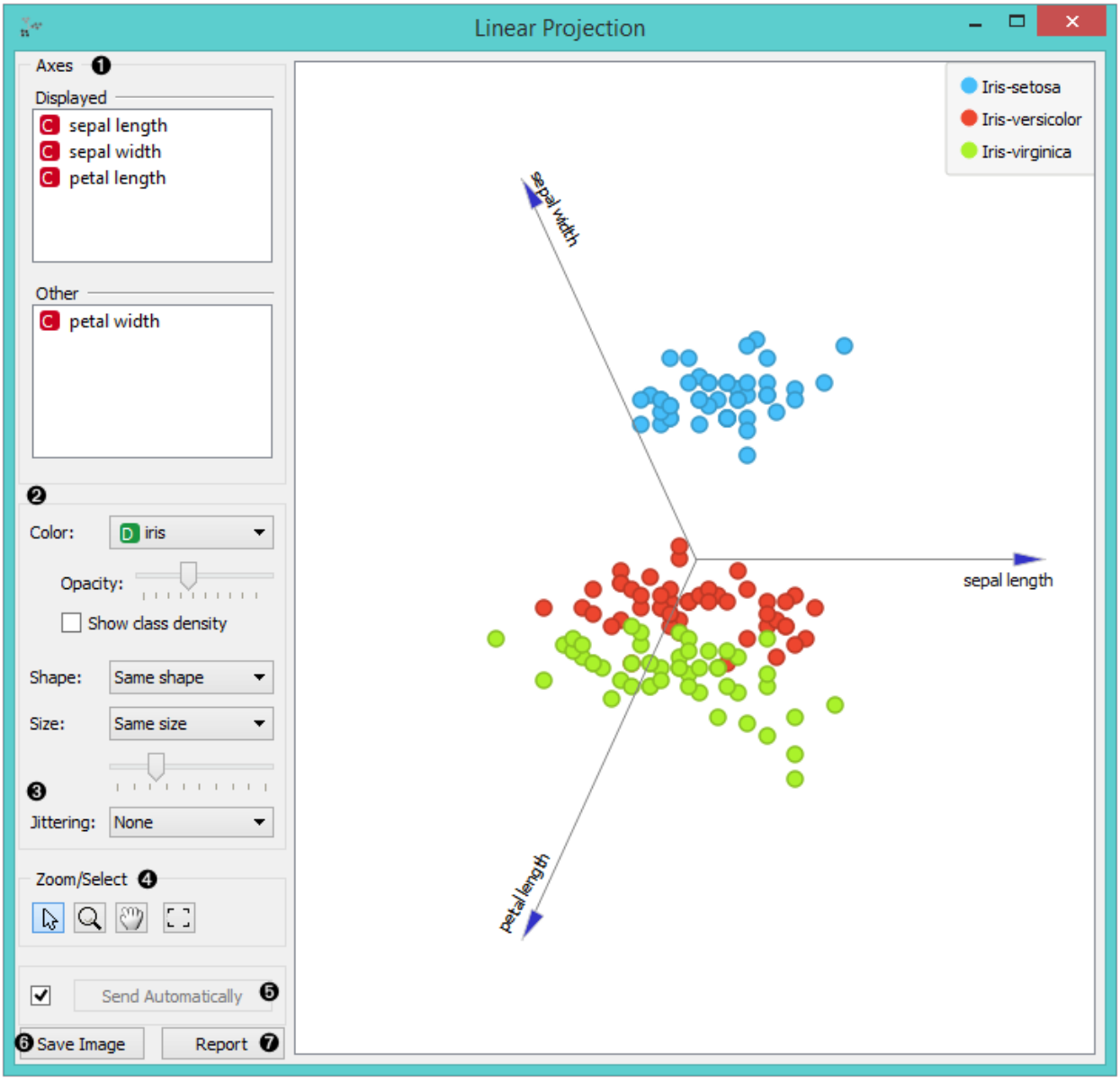
- **Selected Data**


untitled*
File
Random Forest Classification
SVM
Naive Bayes
Test & Score
Confusion Matrix SVM (Misclassified)
Confusion Matrix Random Forest (Misclassified)
Confusion Matrix Naive Bayes (Misclassified)
Venn Diagram
Learners
svm
random forest
naive bayes
Predicted
Iris-setosa
Iris-versicolor
Iris-virginica
Show
Number of instances
Select
Correct
Misclassified
None
Output
Predictions
Probabilities
Auto send is on
Report
Info
3 data sets on input
Data Instance Identifiers
Use instance equality
Use identifiers
Output duplicates
Save Image
random forest
naive bayes
svm

A data subset that the user has manually selected in the projection.

Description

This widget displays linear projections of class-labeled data. Consider, for a start, a projection of the *Iris* data set shown below. Notice that it is the sepal width and sepal length that already separate *Iris setosa* from the other two, while the petal length is the attribute best separating *Iris versicolor* from *Iris virginica*.

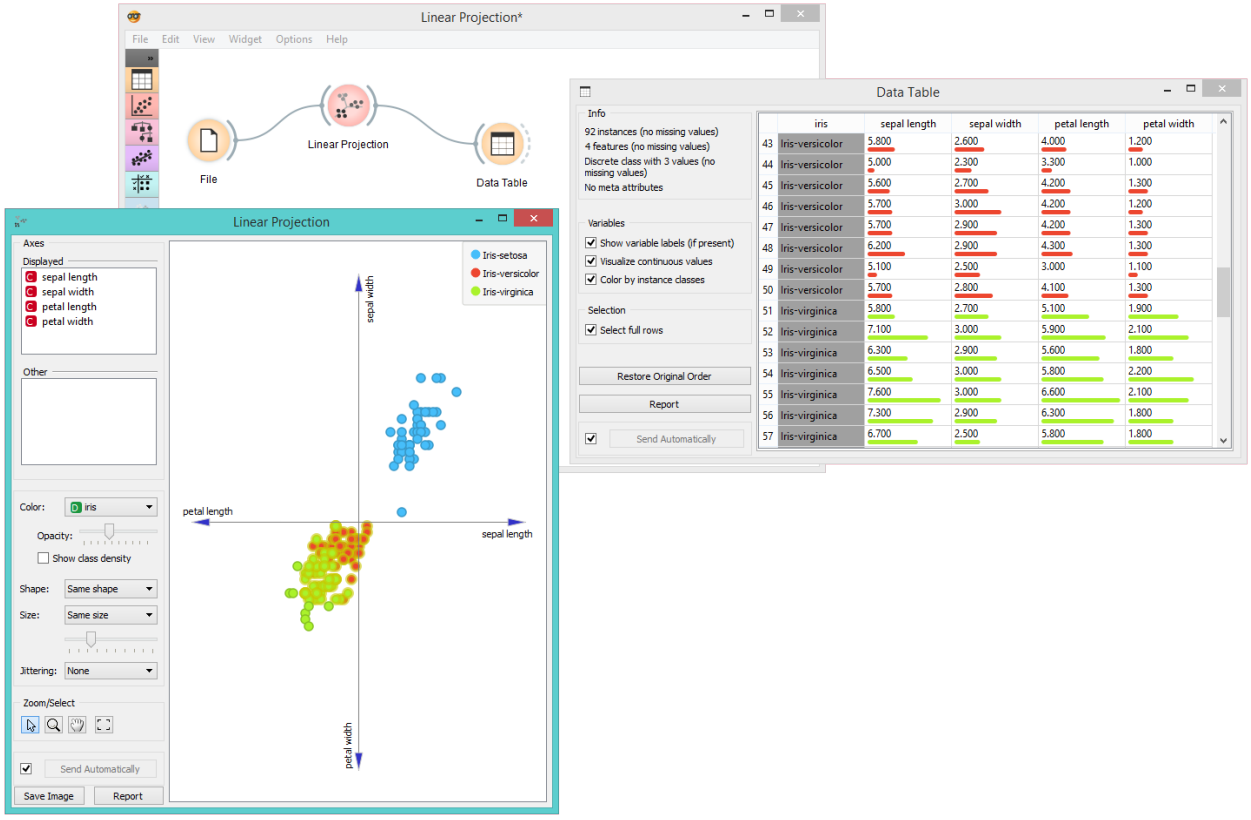


1. Axes in the projection that are displayed and other available axes.
2. Set the color of the displayed dots (you will get colored dots for discrete values and grey-scale dots for continuous). Set opacity, shape and size to differentiate between instances.
3. Set jittering to prevent the dots from overlapping (especially for discrete attributes).

4. *Select*, *zoom*, *pan* and *zoom to fit* options for exploring the graph. Manual selection of data instances works as a non-angular/free-hand selection tool. Double click to move the projection. Scroll in or out for zoom.
5. When the box is ticked (*Auto commit is on*), the widget will communicate the changes automatically. Alternatively, click *Commit*.
6. *Save Image* saves the created image to your computer in a .svg or .png format.
7. Produce a report.

Example

The **Linear Projection** widget works just like other visualization widgets. Below, we connected it to the *File* widget to see the set projected on a 2-D plane. Then we selected the data for further analysis and connected it to the *Data Table* widget to see the details of the selected subset.



References

Koren Y., Carmel L. (2003). Visualization of labeled data using linear transformations. In Proceedings of IEEE Information Visualization 2003, (InfoVis'03). Available here.

Boulesteix A.-L., Strimmer K. (2006). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics, 8(1), 32-44. Abstract here.

Scatter Map

Plots a scatter map for a pair of continuous attributes.

Signals

Inputs:

- **Data**

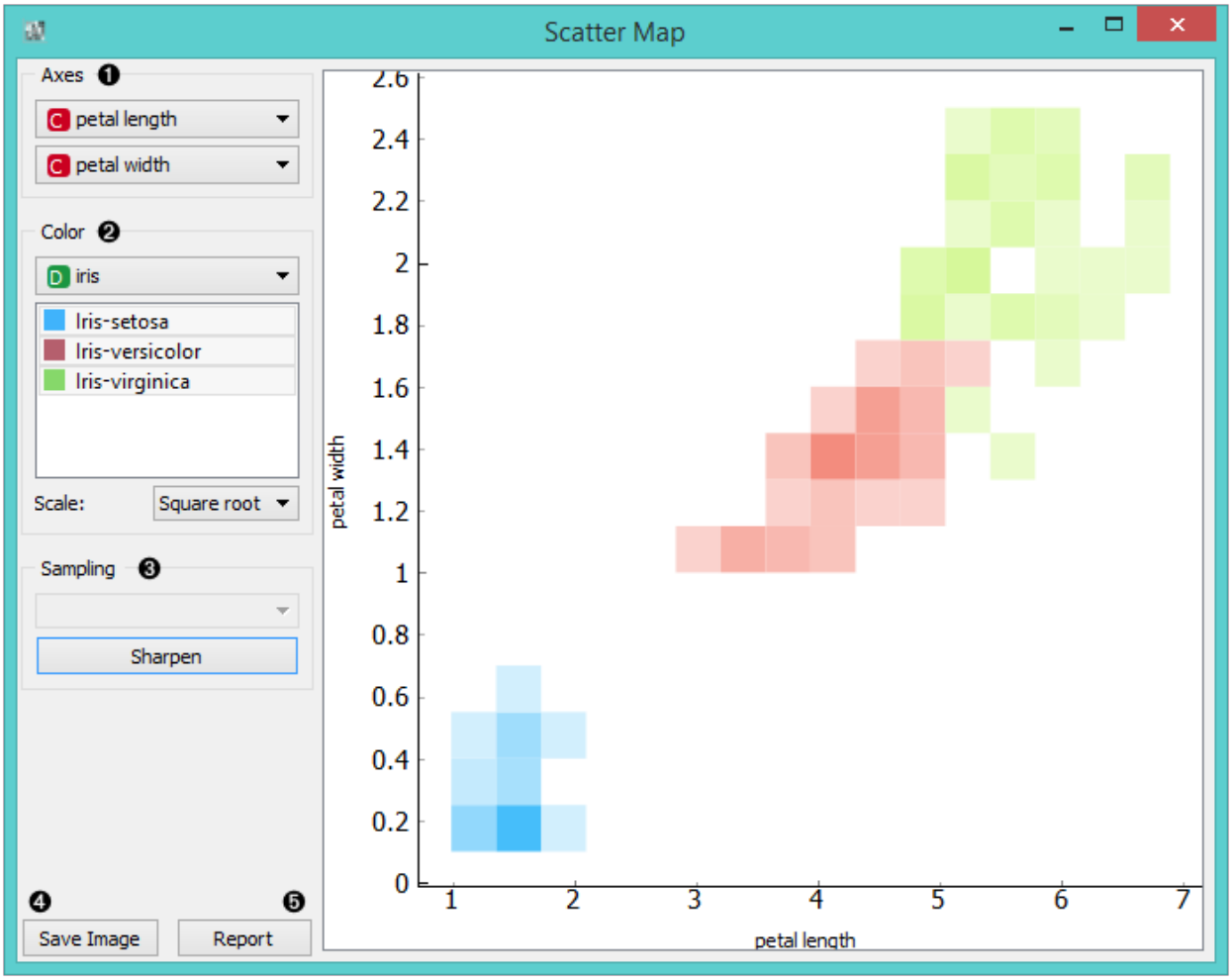
  An input data set

Outputs:

- None

Description

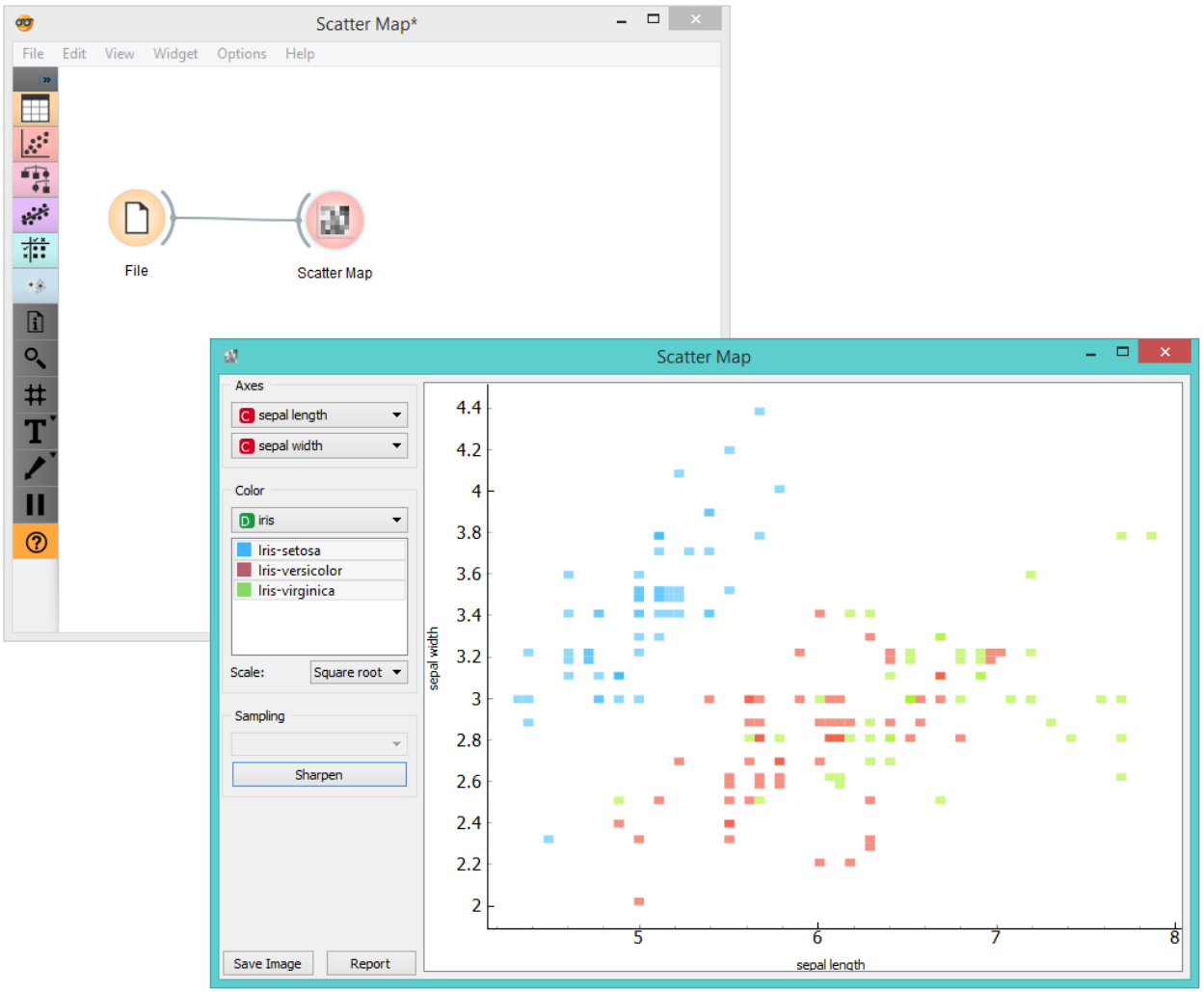
A Scatter map is a graphical method for visualizing frequencies in a two-way matrix by color. The higher the occurrence of a certain value, the darker the represented color. By combining two values on x and y axes, we see where the attribute combination is the strongest and where the weakest, thus enabling the user to find strong correlations or representative instances.



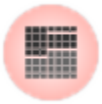
1. Select the x and y attribute to be plotted.
2. Color the plot by attribute. You can also select which attribute instances you wish to see in the visualization by clicking on them. At the bottom, you can select the color scale strength (linear, square root or logarithmic).
3. *Sampling* is enabled only when the widget is connected to the *SQL Table* widget. You can set the sampling time for large data to speed up the analysis. *Sharpen* works for all data types and it will resize (sharpen) the squares in the plot.
4. *Save Image* saves the created image to your computer in a .svg or .png format.
5. Produce a report.

Example

Below, you can see an example workflow for the **Scatter Map** widget. Notice that the widget only works with continuous data, so you need to first continuize the data attributes you want to visualize. The Scatter map below displays two attributes from the *Iris* data set, namely the petal width and petal length. Here, we can see the distribution of width and length values per Iris type. You can see that the variety *Iris setosa* is distinctly separated from the other two varieties by petal width and length and that the most typical values for these attributes are around 0.2 for petal width and between 1.4 and 1.7 for petal length. This shows that petal width and length are good attributes for telling Iris setosa apart from the other two varieties.



Sieve Diagram



Plots a sieve diagram for a pair of attributes.

Signals

Inputs:

- **Data**

  An input data set

Outputs:

- None

Description

A **Sieve diagram** is a graphical method for visualizing frequencies in a two-way contingency table and comparing them to expected frequencies under assumption of independence. It was proposed by Riedwyl and Schüpbach in a technical report in 1983 and later called a parquet diagram (Riedwyl and Schüpbach, 1994). In this display, the area of each rectangle is proportional to the expected frequency, while the observed frequency is shown by the number of squares in each rectangle. The difference between observed and expected frequency (proportional to the standard Pearson residual) appears as the density of shading, using color to indicate whether the deviation from independence is positive (blue) or negative (red).

1. Select the attributes you want to display in the sieve plot.
2. Score combinations enables you to fin the best possible combination of attributes.
3. *Save Image* saves the created image to your computer in a .svg or .png format.
4. Produce a report.

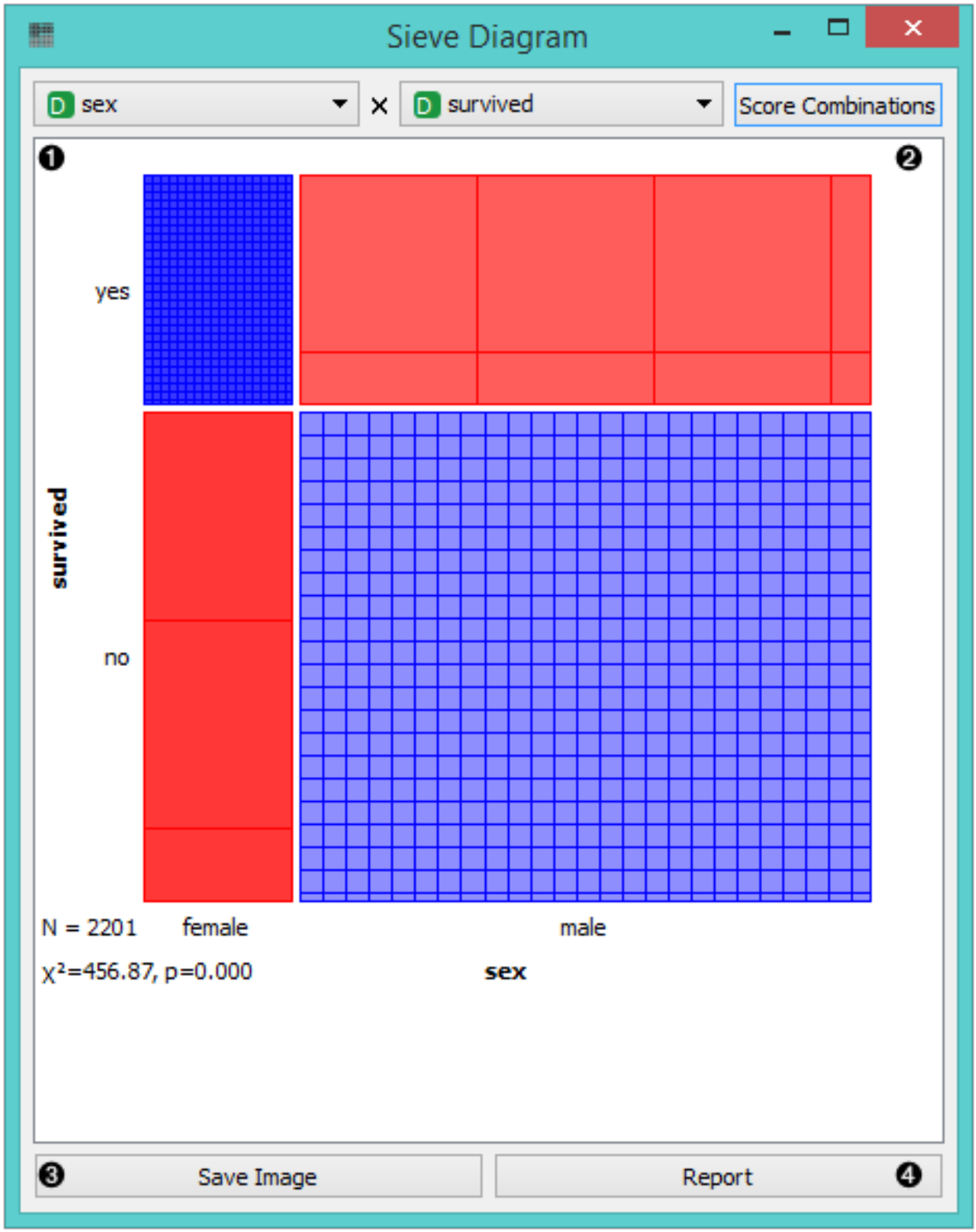
The snapshot below shows a sieve diagram for the *Titanic* data set and has the attributes *sex* and *survived* (the latter is a class attribute in this data set). The plot shows that the two variables are highly associated, as there are substantial differences between observed and expected frequencies in all of the four quadrants. For example, and as highlighted in the balloon, the chance for surviving the accident was much higher for female passengers than expected (0.06 vs. 0.15).

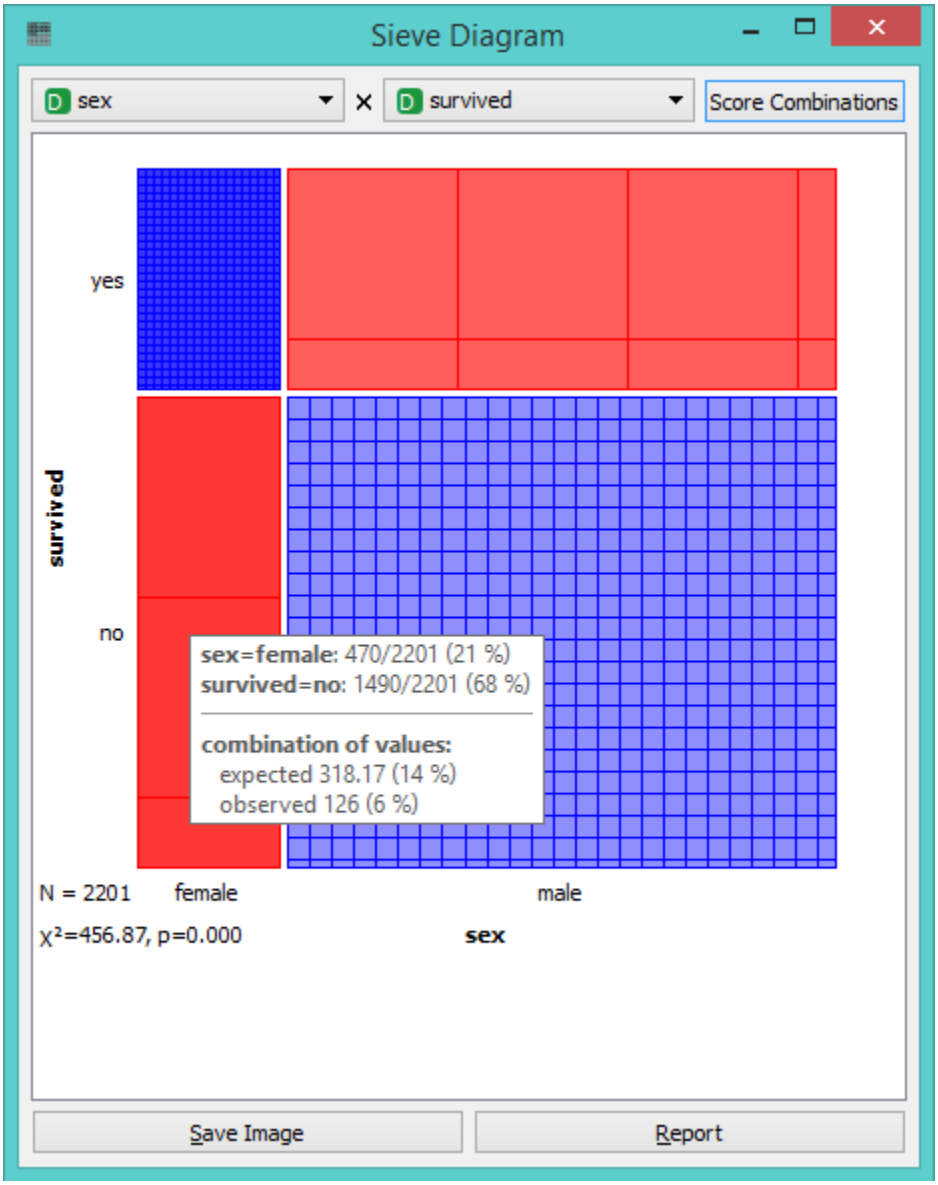
Pairs of attributes with interesting associations have a strong shading, such as the diagram shown in the above snapshot. For contrast, a sieve diagram of the least interesting pair (age vs. survival) is shown below.

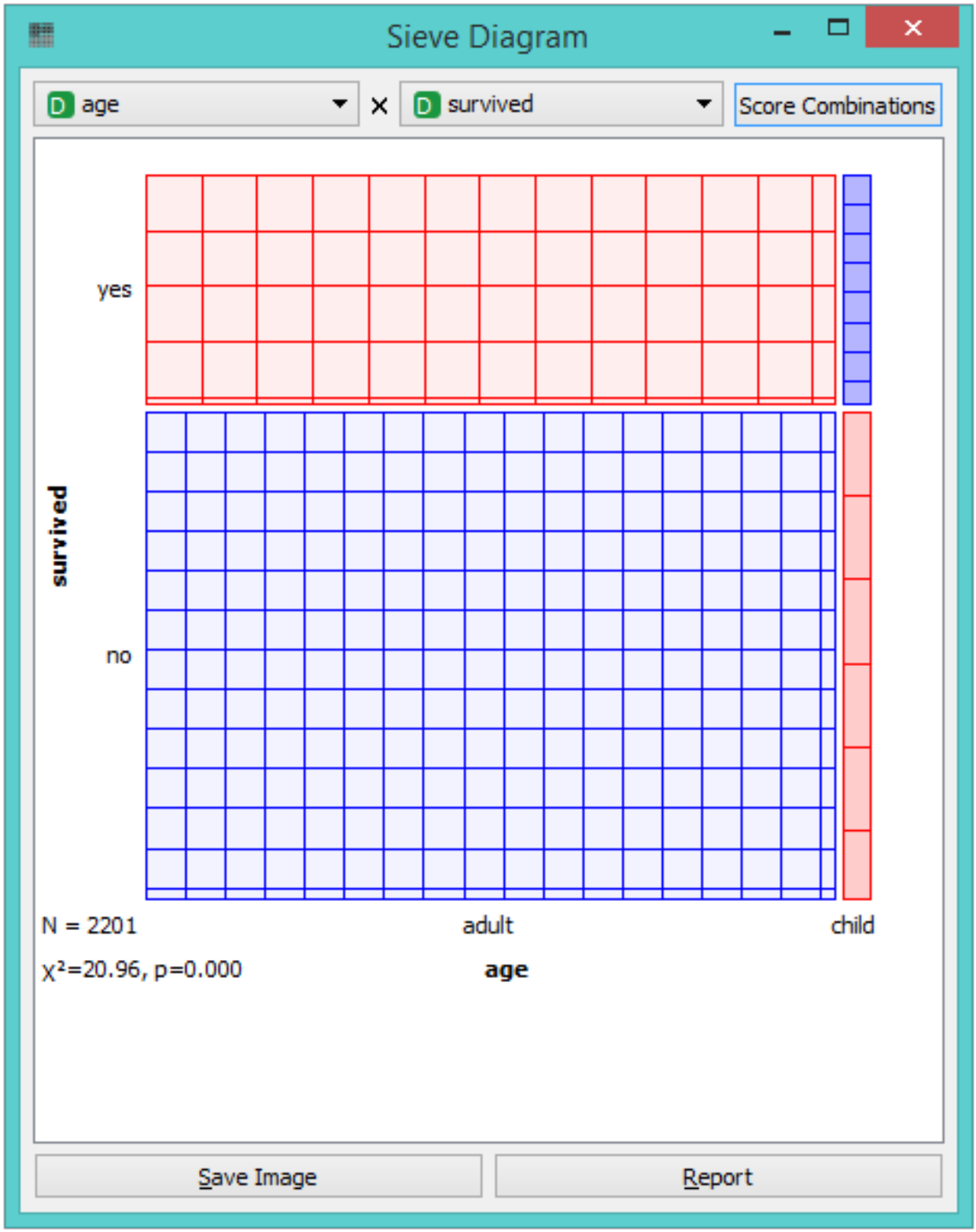
Example

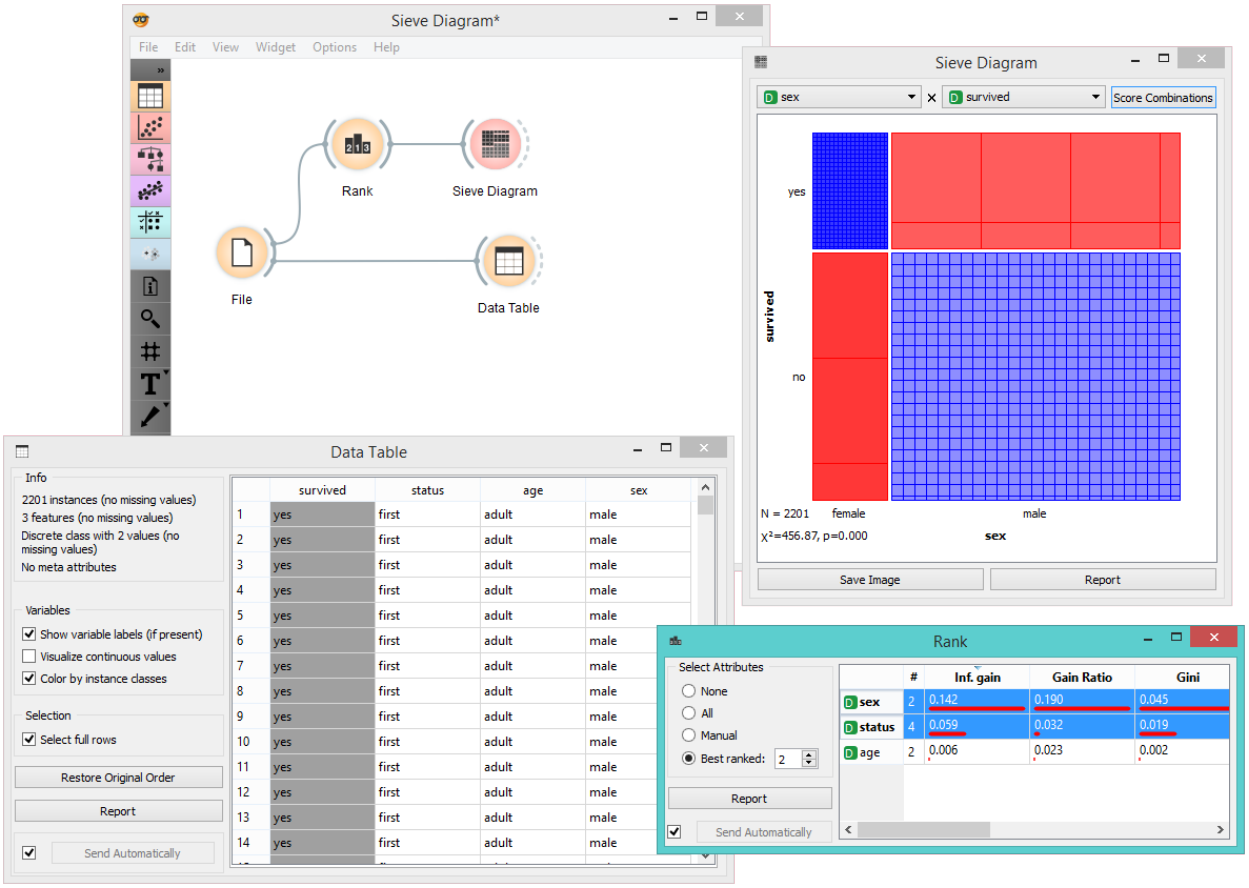
Below, we see a simple schema using the *Titanic* data set, where we use the *Rank* widget to select the best attributes (the ones with the highest information gain, gain ratio or gini index) and feed them into the **Sieve Diagram**. This displays the sieve plot for the two best attributes, which in our case are sex and status. We see that the survival rate on the Titanic was very high for women of the first class and very low for female crew members.

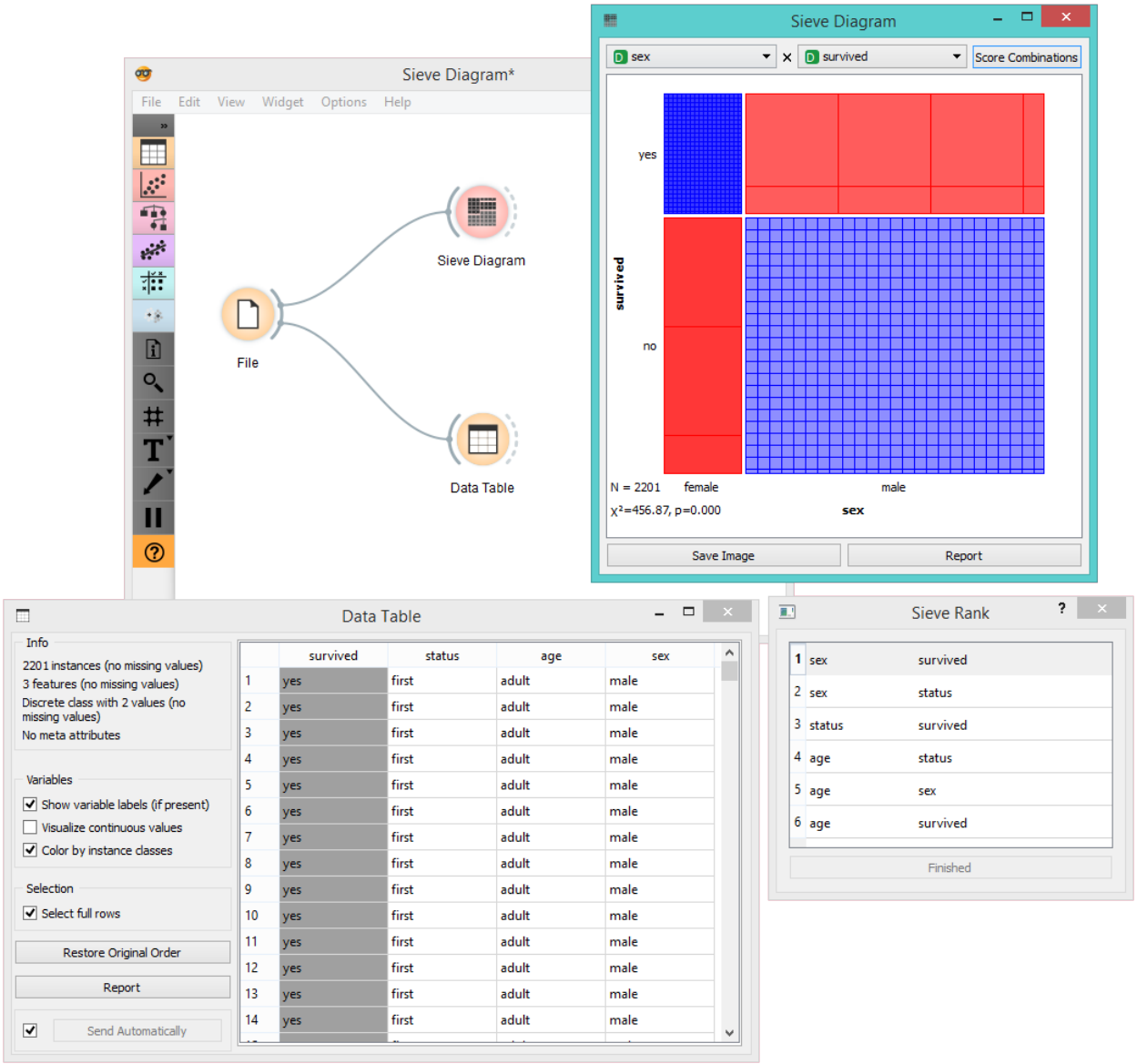
The **Sieve Diagram** also features the *Score Combinations* option, which makes the ranking of attributes even easier.


Sieve Diagram
sex
survived
Score Combinations
1
2
yes
no
survived
N = 2201
female
male
χ²=456.87, p=0.000
sex
3
Save Image
Report
4


Sieve Diagram
sex
survived
Score Combinations
yes
no
survived
sex=female: 470/2201 (21 %)
survived=no: 1490/2201 (68 %)
combination of values:
expected 318.17 (14 %)
observed 126 (6 %)
N = 2201
female
male
χ²=456.87, p=0.000
sex
Save Image
Report


Sieve Diagram
age
survived
Score Combinations
yes
no
survived
N = 2201
adult
child
χ²=20.96, p=0.000
age
Save Image
Report


Sieve Diagram*
File Edit View Widget Options Help
Rank
Sieve Diagram
File
Data Table
Sieve Diagram
sex
survived
Score Combinations
yes
no
survived
N = 2201
female
male
χ²=456.87, p=0.000
sex
Save Image
Report
Data Table
Info
2201 instances (no missing values)
3 features (no missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
survived status age sex
1 yes first adult male
2 yes first adult male
3 yes first adult male
4 yes first adult male
5 yes first adult male
6 yes first adult male
7 yes first adult male
8 yes first adult male
9 yes first adult male
10 yes first adult male
11 yes first adult male
12 yes first adult male
13 yes first adult male
14 yes first adult male
Rank
Select Attributes
None
All
Manual
Best ranked: 2
Report
Send Automatically
Inf. gain Gain Ratio Gini
sex 2 0.142 0.190 0.045
status 4 0.059 0.032 0.019
age 2 0.006 0.023 0.002


Sieve Diagram*
File Edit View Widget Options Help
Sieve Diagram
File
Data Table
Sieve Diagram
sex
survived
Score Combinations
yes
no
survived
N = 2201
female
male
X²=456.87, p=0.000
sex
Save Image
Report
Data Table
Info
2201 instances (no missing values)
3 features (no missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
survived status age sex
1 yes first adult male
2 yes first adult male
3 yes first adult male
4 yes first adult male
5 yes first adult male
6 yes first adult male
7 yes first adult male
8 yes first adult male
9 yes first adult male
10 yes first adult male
11 yes first adult male
12 yes first adult male
13 yes first adult male
14 yes first adult male
Sieve Rank
1 sex survived
2 sex status
3 status survived
4 age status
5 age sex
6 age survived
Finished

References

Riedwyl, H., and Schüpbach, M. (1994). Parquet diagram to plot contingency tables. In Softstat '93: Advances in Statistical Software, F. Faulbaum (Ed.). New York: Gustav Fischer, 293-299.

Pythagorean Tree

Pythagorean tree visualisation for classification or regression trees.

Signals

Inputs:

- **Tree**

  A decision tree model.

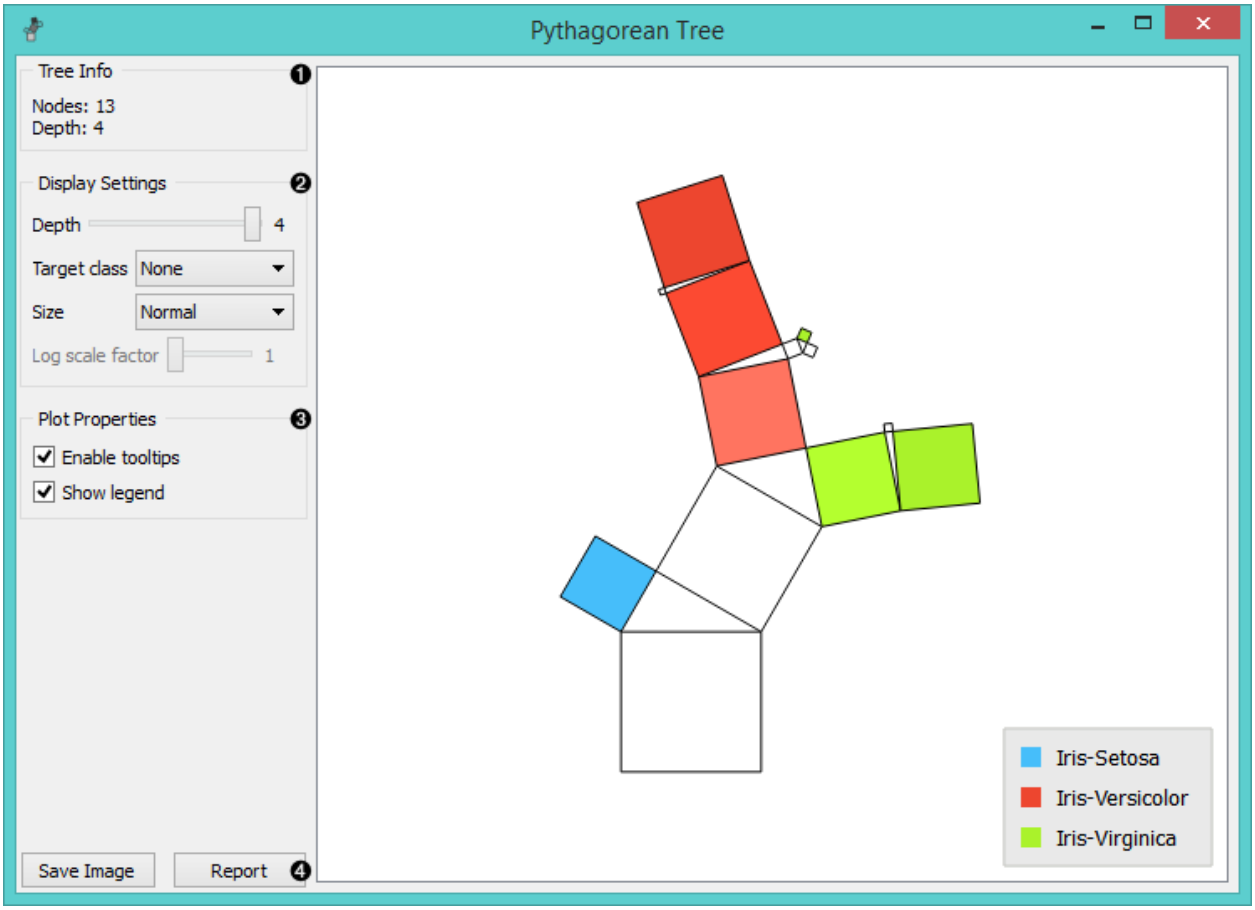
- **Selected Data**

  A subset of instances that the user has manually selected from the Pythagorean tree.

Description

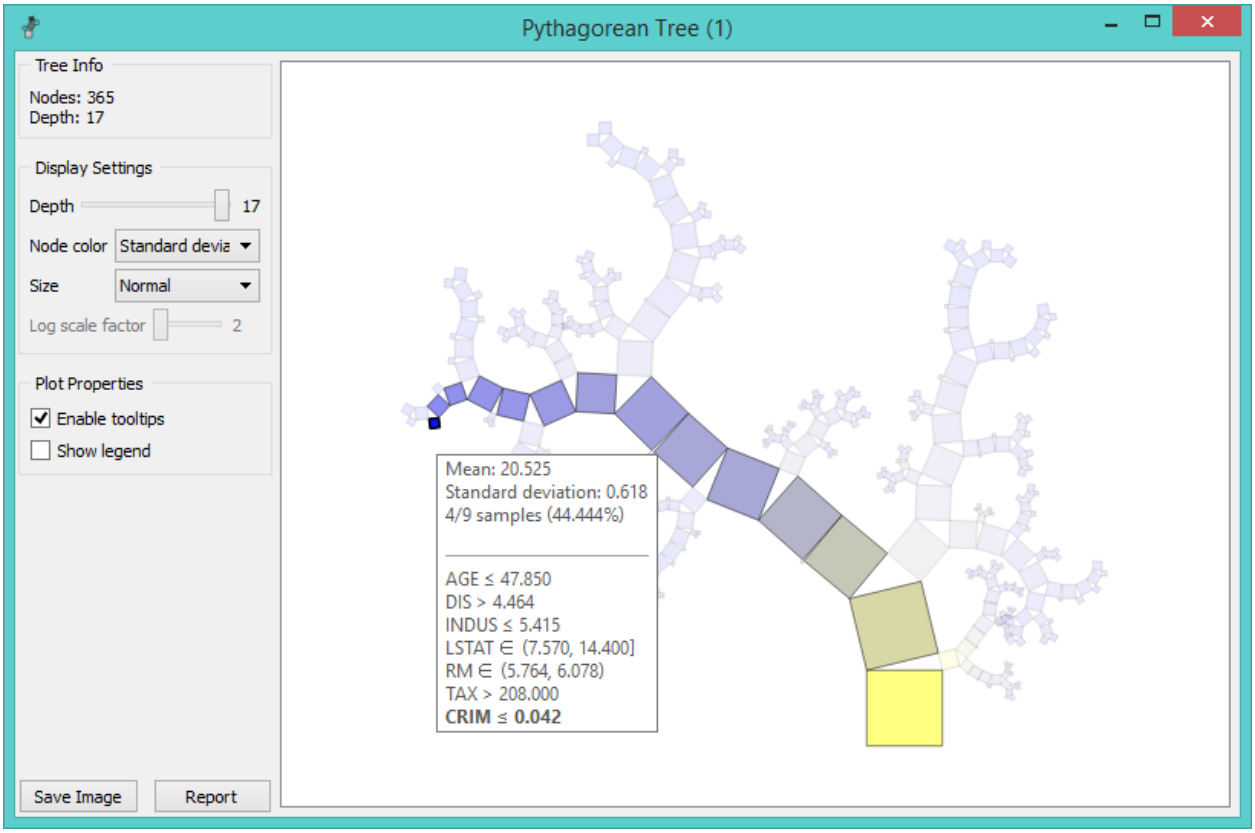
Pythagorean Trees are plane fractals that can be used to depict general tree hierarchies as presented in an article by Fabian Beck and co-authors. In our case, they are used for visualizing and exploring tree models, such as *Tree*.

1. Information on the input tree model.
2. Visualization parameters:
   - *Depth*: set the depth of displayed trees.
   - *Target class* (for classification trees): the intensity of the color for nodes of the tree will correspond to the probability of the target class. If *None* is selected, the color of the node will denote the most probable class.
   - *Node color* (for regression trees): node colors can correspond to mean or standard deviation of class value of the training data instances in the node.
   - *Size*: define a method to compute the size of the square representing the node. *Normal* will keep node sizes correspond to the size of training data subset in the node. *Square root* and *Logarithmic* are the respective transformations of the node size.
   - *Log scale factor* is only enabled when *logarithmic* transformation is selected. You can set the log factor between 1 and 10.
3. Plot properties:
   - *Enable tooltips*: display node information upon hovering.
   - *Show legend*: shows color legend for the plot.
4. **Reporting:**
   - *Save Image*: save the visualization to a SVG or PNG file.


Pythagorean Tree
Tree Info
Nodes: 13
Depth: 4
Display Settings
Depth
4
Target class
None
Size
Normal
Log scale factor
1
Plot Properties
Enable tooltips
Show legend
Save Image
Report
Iris-Setosa
Iris-Versicolor
Iris-Virginica

- *Report*: add visualization to the report.

Pythagorean Tree can visualize both classification and regression trees. Below is an example for regression tree. The only difference between the two is that regression tree doesn't enable coloring by class, but can color by class mean or standard deviation.



Example

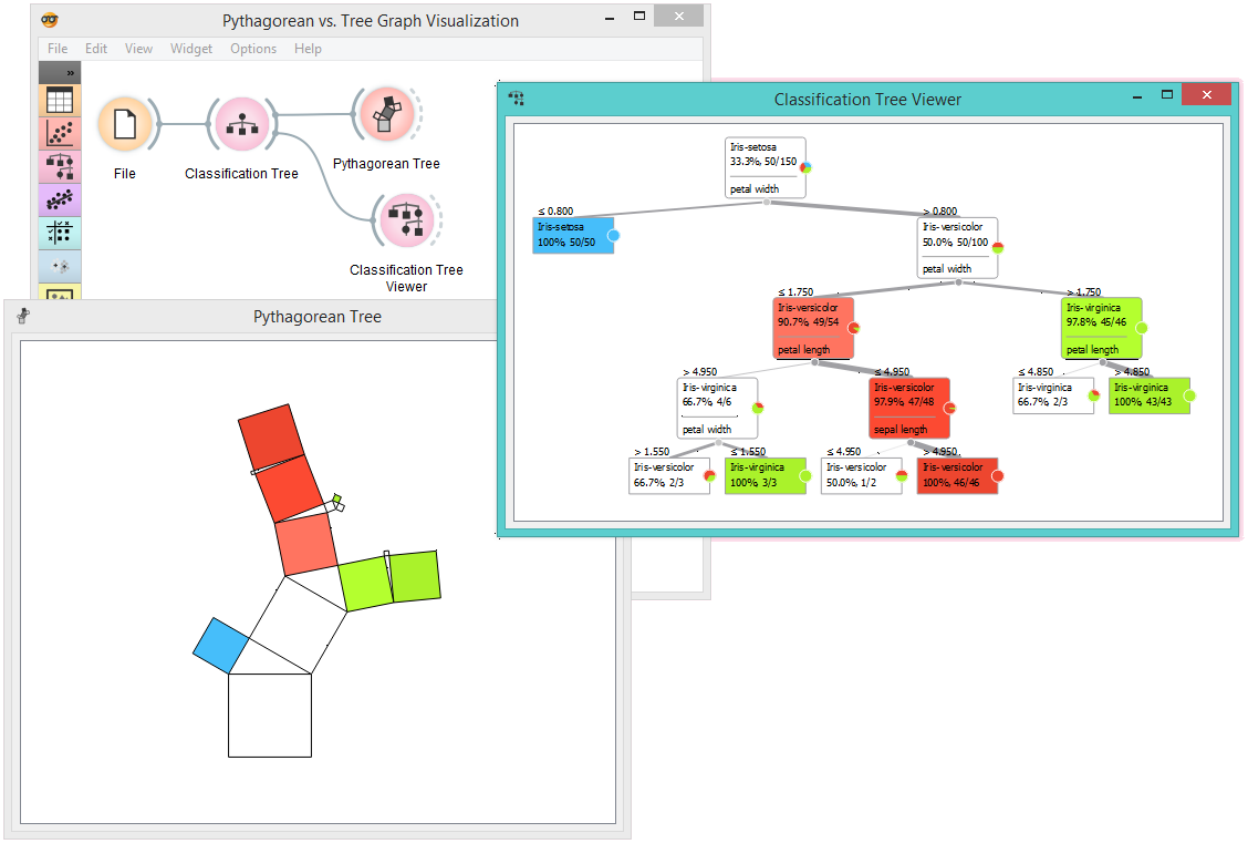
The workflow from the screenshot below demonstrates the difference between *Tree Viewer* and Pythagorean Tree. They can both visualize *Tree*, but Pythagorean visualization takes less space and is more compact, even for a small Iris flower data set. For both visualization widgets, we have hidden the control area on the left by clicking on the splitter between control and visualization area.

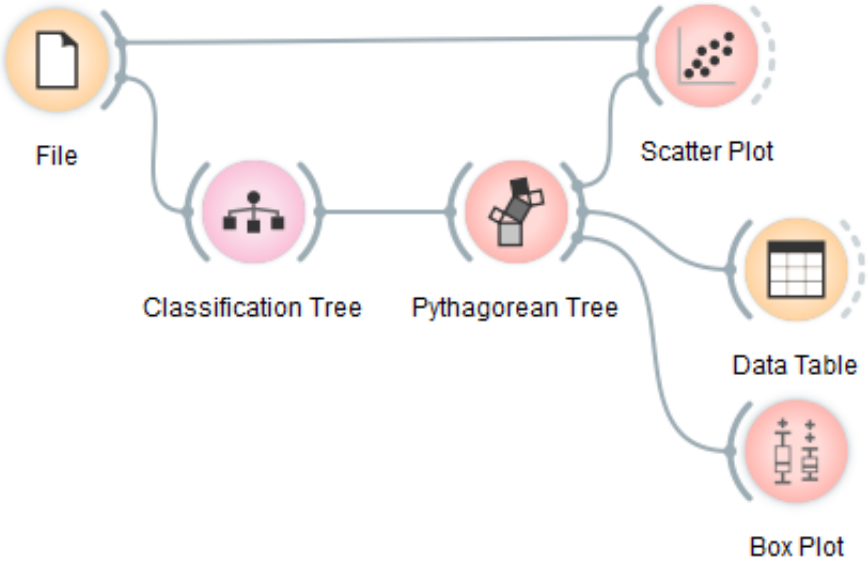
Pythagorean Tree is interactive: click on any of the nodes (squares) to select training data instances that were associated with that node. The following workflow explores these feature.

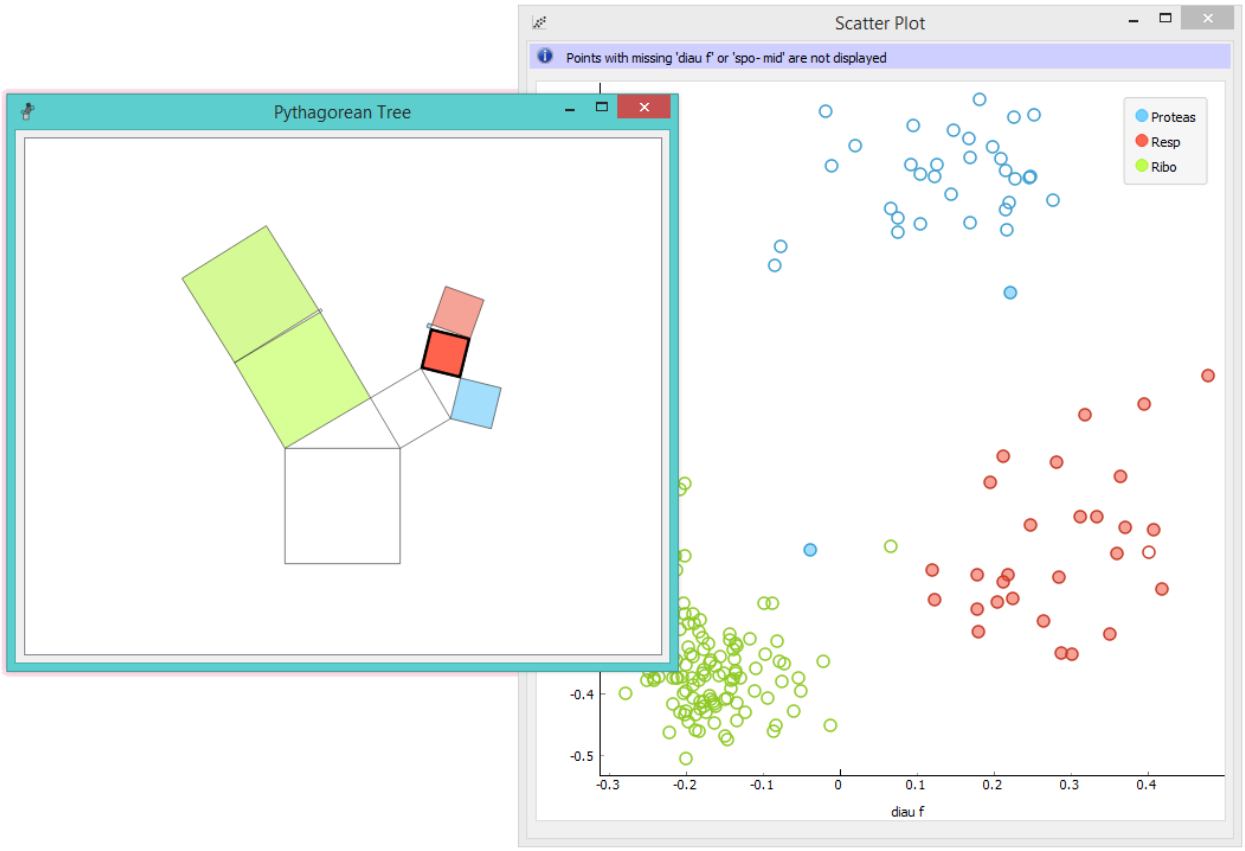
The selected data instances are shown as a subset in the *Scatter Plot*, sent to the *Data Table* and examined in the *Box Plot*. We have used brown-selected data set in this example. The tree and scatter plot are shown below; the selected node in the tree has a black outline.

References

Beck, F., Burch, M., Munz, T., Di Silvestro, L. and Weiskopf, D. (2014). Generalized Pythagoras Trees for Visualizing Hierarchies. In IVAPP '14 Proceedings of the 5th International Conference on Information Visualization Theory and Applications, 17-28.


Pythagorean vs. Tree Graph Visualization
File Edit View Widget Options Help
File
Classification Tree
Pythagorean Tree
Classification Tree Viewer
Pythagorean Tree
Classification Tree Viewer
Iris-setosa
33.3% 50/150
petal width
≤ 0.800
Iris-setosa
100% 50/50
> 0.800
Iris-versicolor
50.0% 50/100
petal width
≤ 1.750
Iris-versicolor
90.7% 49/54
petal length
> 1.750
Iris-virginica
97.8% 45/46
petal length
> 4.950
Iris-virginica
66.7% 4/6
petal width
≤ 4.950
Iris-versicolor
97.9% 47/48
sepal length
≤ 4.850
Iris-virginica
66.7% 2/3
> 4.850
Iris-virginica
100% 43/43
> 1.550
Iris-versicolor
66.7% 2/3
≤ 1.650
Iris-virginica
100% 3/3
≤ 4.950
Iris-versicolor
50.0% 1/2
> 4.950
Iris-versicolor
100% 46/46


File
Scatter Plot
Classification Tree
Pythagorean Tree
Data Table
Box Plot



Pythagorean Forest

Pythagorean forest for visualising random forests.

Signals

Inputs:

- **Random Forest**

  Classification / regression tree models as random forest.

Outputs:

- **Tree**

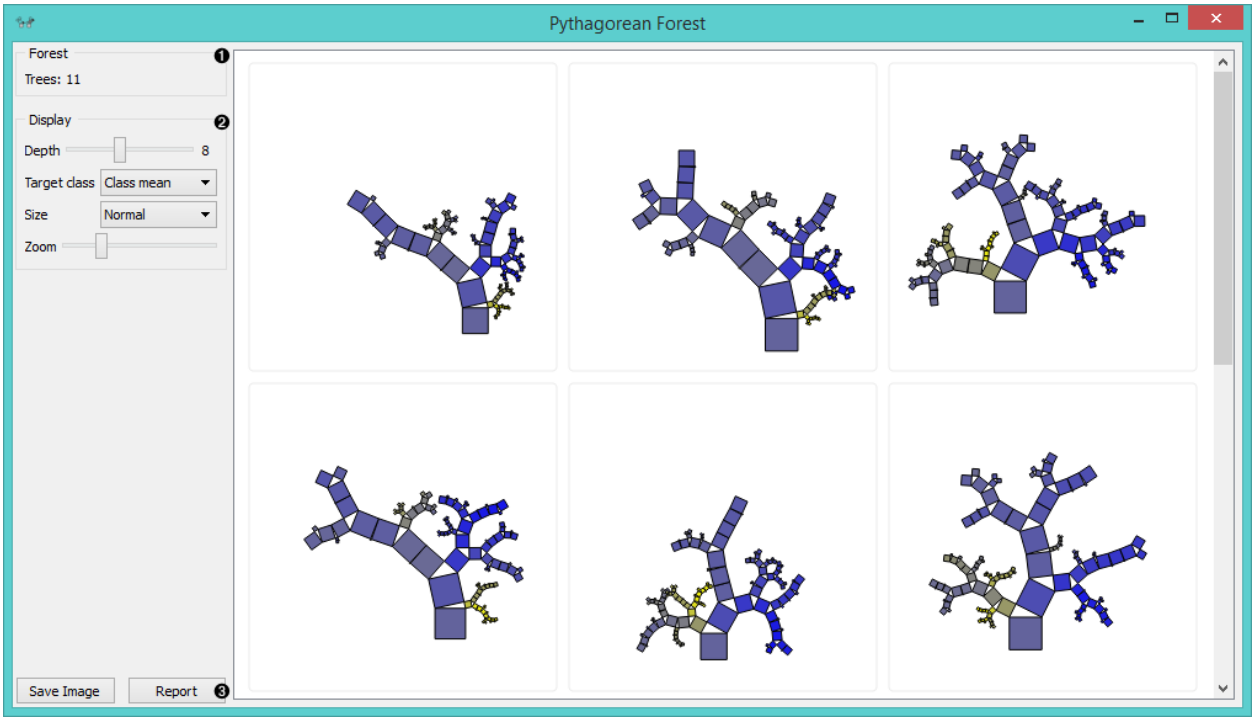
  A selected classification / regression tree model.

Description

Pythagorean Forest shows all learned decision tree models from *Random Forest* widget. It displays then as Pythagorean trees, each visualization pertaining to one randomly constructed tree. In the visualization, you can select

a tree and display it in *Pythagorean Tree* wigdet. The best tree is the one with the shortest and most strongly colored branches. This means few attributes split the branches well.

Widget displays both classification and regression results. Classification requires discrete target variable in the data set, while regression requires a continuous target variable. Still, they both should be fed a *Tree* on the input.

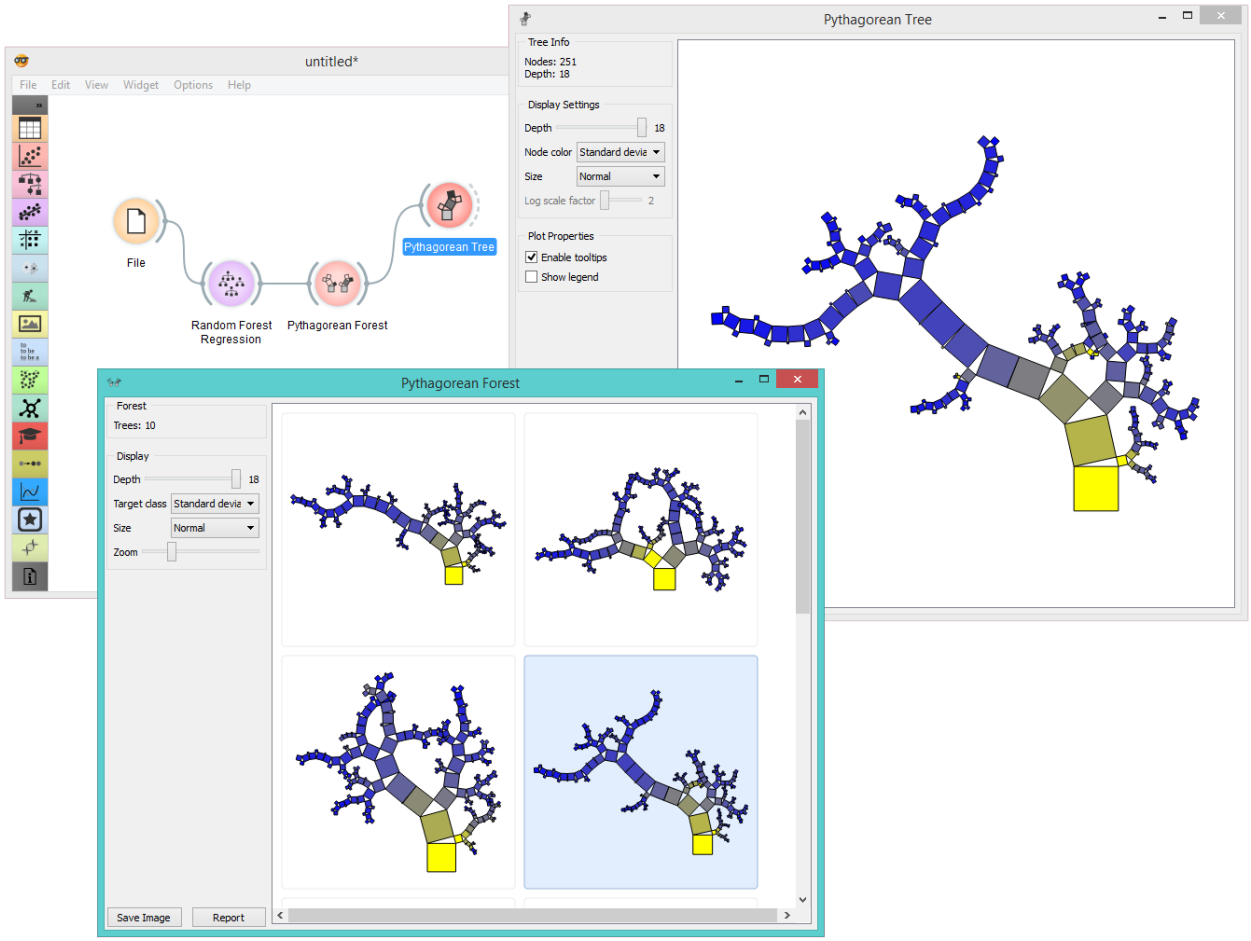


1. Information on the input random forest model.
2. Display parameters:
   - *Depth*: set the depth to which the trees are grown.
   - *Target class*: set the target class for coloring the trees. If *None* is selected, tree will be white. If the input is a classification tree, you can color nodes by their respective class. If the input is a regression tree, the options are *Class mean*, which will color tree nodes by the class mean value and *Standard deviation*, which will color then by the standard deviation value of the node.
   - *Size*: set the size of the nodes. *Normal* will keep nodes the size of the subset in the node. *Square root* and *Logarithmic* are the respective transformations of the node size.
   - *Zoom*: allows you to se the size of the tree visualizations.
3. *Save Image*: save the visualization to your computer as a *.svg* or *.png* file. *Report*: produce a report.

Example

Pythagorean Forest is great for visualizing several built trees at once. In the example below, we've used *housing* data set and plotted all 10 trees we've grown with *Random Forest*. When changing the parameters in Random Forest, visualization in Pythagorean Forest will change as well.

Then we've selected a tree in the visualization and inspected it further with *Pythagorean Tree* widget.


untitled*
File Edit View Widget Options Help
File
Random Forest Regression
Pythagorean Forest
Pythagorean Tree
Pythagorean Tree
Tree Info
Nodes: 251
Depth: 18
Display Settings
Depth 18
Node color Standard devia
Size Normal
Log scale factor 2
Plot Properties
Enable tooltips
Show legend
Pythagorean Forest
Forest
Trees: 10
Display
Depth 18
Target class Standard devia
Size Normal
Zoom
Save Image
Report

References

Beck, F., Burch, M., Munz, T., Di Silvestro, L. and Weiskopf, D. (2014). Generalized Pythagoras Trees for Visualizing Hierarchies. In IVAPP '14 Proceedings of the 5th International Conference on Information Visualization Theory and Applications, 17-28.

CN2 Rule Viewer

CN2 Rule Viewer

Signals

Inputs:

- **Data**

  Data set to filter.

- **CN2 Rule Classifier**

  CN2 Rule Classifier, including a list of induced rules.

Outputs:

- **Filtered Data**

  If data is connected, upon active selection (at least one rule is selected), filtered data is emitted. Output are data instances covered by all selected rules.

Description

A widget that displays CN2 classification rules. If data is also connected, upon rule selection, one can analyze which instances abide to the conditions.

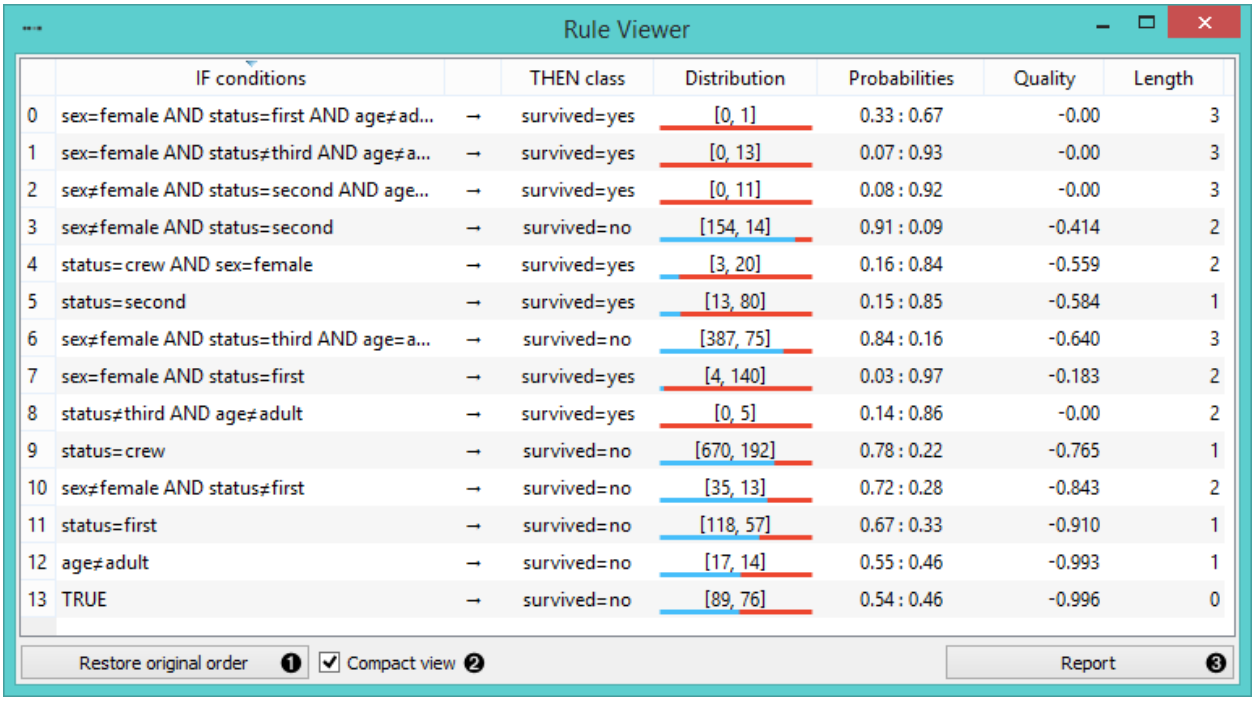
1. Original order of induced rules can be restored.
2. When rules are many and complex, the view can appear packed. For this reason, *compact view* was implemented, which allows a flat presentation and a cleaner inspection of rules.
3. Click *Report* to bring up a detailed description of the rule induction algorithm and its parameters, the data domain, and induced rules.

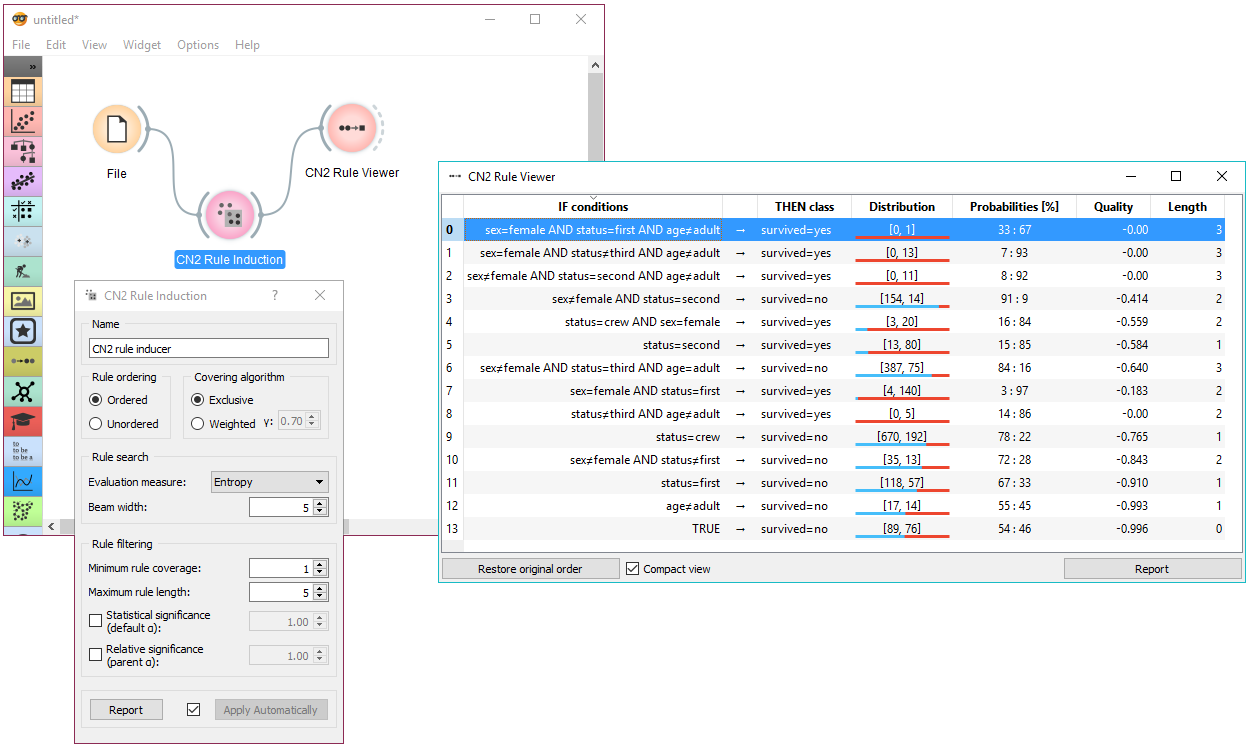
Additionally, upon selection, rules can be copied to clipboard by pressing the default system shortcut (ctrl+C, cmd+C).

Examples

In the schema below, the most common use of the widget is presented. First, the data is read and a CN2 rule classifier is trained. We are using *titanic* data set for the rule constrution. The rules are then viewed using the *Rule Viewer*. To explore different CN2 algorithms and understand how adjusting parameters influences the learning process, **Rule Viewer** should be kept open and in sight, while setting the CN2 learning algorithm (the presentation will be updated promptly).

Selecting a rule outputs filtered data instances. These can be viewed in a *Data Table*.


Rule Viewer
IF conditions
THEN class
Distribution
Probabilities
Quality
Length
0 sex=female AND status=first AND age≠ad... → survived=yes [0, 1] 0.33 : 0.67 -0.00 3
1 sex=female AND status≠third AND age≠a... → survived=yes [0, 13] 0.07 : 0.93 -0.00 3
2 sex≠female AND status=second AND age... → survived=yes [0, 11] 0.08 : 0.92 -0.00 3
3 sex≠female AND status=second → survived=no [154, 14] 0.91 : 0.09 -0.414 2
4 status=crew AND sex=female → survived=yes [3, 20] 0.16 : 0.84 -0.559 2
5 status=second → survived=yes [13, 80] 0.15 : 0.85 -0.584 1
6 sex≠female AND status=third AND age=a... → survived=no [387, 75] 0.84 : 0.16 -0.640 3
7 sex=female AND status=first → survived=yes [4, 140] 0.03 : 0.97 -0.183 2
8 status≠third AND age≠adult → survived=yes [0, 5] 0.14 : 0.86 -0.00 2
9 status=crew → survived=no [670, 192] 0.78 : 0.22 -0.765 1
10 sex≠female AND status≠first → survived=no [35, 13] 0.72 : 0.28 -0.843 2
11 status=first → survived=no [118, 57] 0.67 : 0.33 -0.910 1
12 age≠adult → survived=no [17, 14] 0.55 : 0.46 -0.993 1
13 TRUE → survived=no [89, 76] 0.54 : 0.46 -0.996 0
Restore original order
Compact view
Report


untitled*
File Edit View Widget Options Help
File
CN2 Rule Viewer
CN2 Rule Induction
CN2 Rule Induction
Name
CN2 rule inducer
Rule ordering
Ordered
Unordered
Covering algorithm
Exclusive
Weighted
Rule search
Evaluation measure:
Entropy
Beam width:
Rule filtering
Minimum rule coverage:
Maximum rule length:
Statistical significance (default α):
Relative significance (parent α):
Report
Apply Automatically
CN2 Rule Viewer
IF conditions
THEN class
Distribution
Probabilities [%]
Quality
Length
0 sex=female AND status=first AND age≠adult → survived=yes [0, 1] 33 : 67 -0.00 3
1 sex=female AND status≠third AND age≠adult → survived=yes [0, 13] 7 : 93 -0.00 3
2 sex≠female AND status=second AND age≠adult → survived=yes [0, 11] 8 : 92 -0.00 3
3 sex≠female AND status=second → survived=no [154, 14] 91 : 9 -0.414 2
4 status=crew AND sex=female → survived=yes [3, 20] 16 : 84 -0.559 2
5 status=second → survived=yes [13, 80] 15 : 85 -0.584 1
6 sex≠female AND status=third AND age=adult → survived=no [387, 75] 84 : 16 -0.640 3
7 sex=female AND status=first → survived=yes [4, 140] 3 : 97 -0.183 2
8 status≠third AND age≠adult → survived=yes [0, 5] 14 : 86 -0.00 2
9 status=crew → survived=no [670, 192] 78 : 22 -0.765 1
10 sex≠female AND status≠first → survived=no [35, 13] 72 : 28 -0.843 2
11 status=first → survived=no [118, 57] 67 : 33 -0.910 1
12 age≠adult → survived=no [17, 14] 55 : 45 -0.993 1
13 TRUE → survived=no [89, 76] 54 : 46 -0.996 0
Restore original order
Compact view
Report

Mosaic Display

Display data in a mosaic plot.

Signals

Inputs:

- **Data**

  An input data set.

- **Data subset**

  An input data subset.

Outputs:

- **Selected data**

  A subset of instances that the user has manually selected from the plot.

Description

The **Mosaic plot** is a graphical representation of a two-way frequency table or a contingency table. It is used for visualizing data from two or more qualitative variables and was introduced in 1981 by Hartigan and Kleiner and expanded and refined by Friendly in 1994. It provides the user with the means to more efficiently recognize relationships between different variables. If you wish to read up on the history of Mosaic Display, additional reading is available here.

1. Select the variables you wish to see plotted.
2. Select interior coloring. You can color the interior according to class or you can use the *Pearson residual*, which is the difference between observed and fitted values, divided by an estimate of the standard deviation of the observed value. If *Compare to total* is clicked, a comparison is made to all instances.
3. *Save image* saves the created image to your computer in a .svg or .png format.
4. Produce a report.

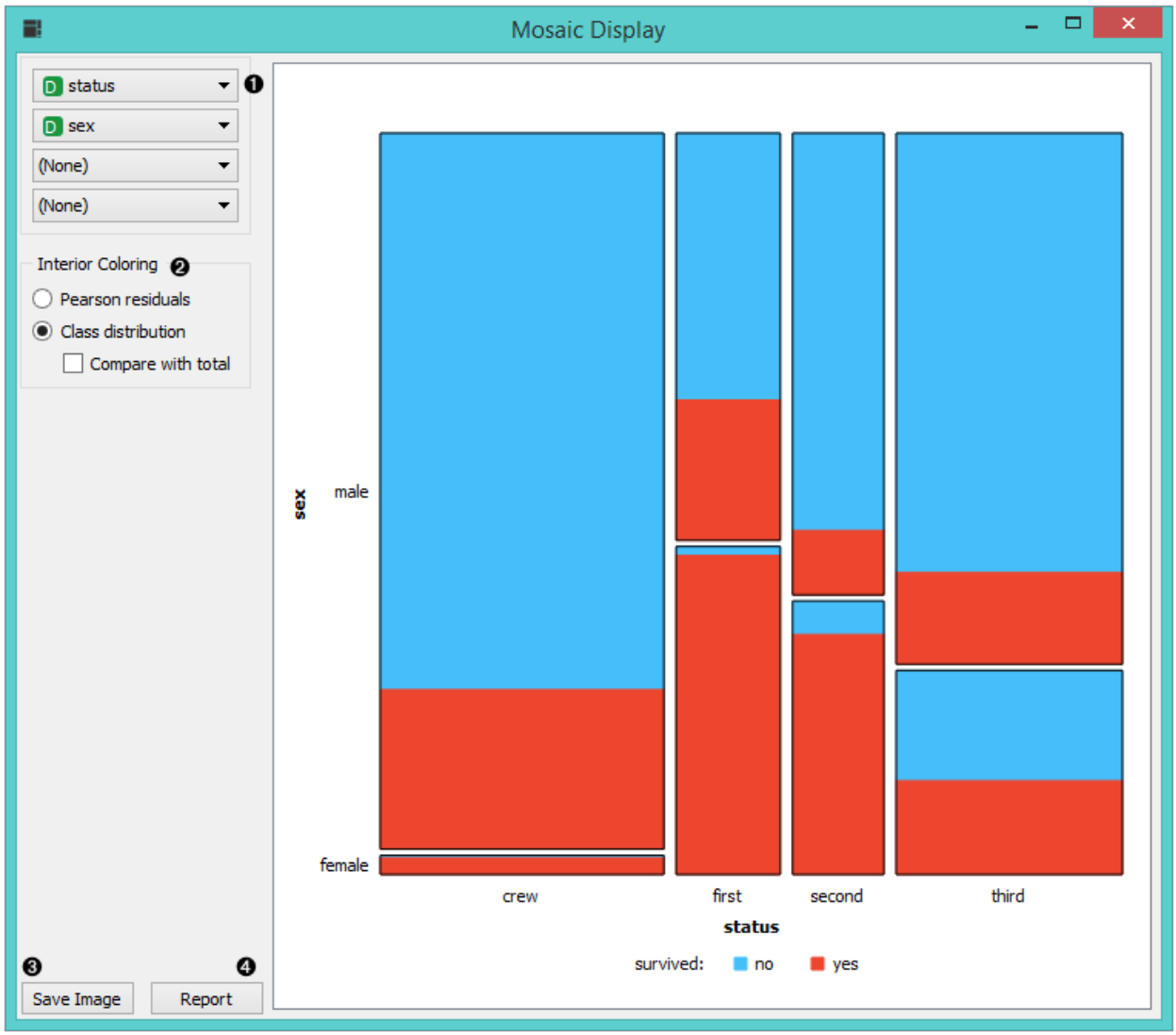
Example

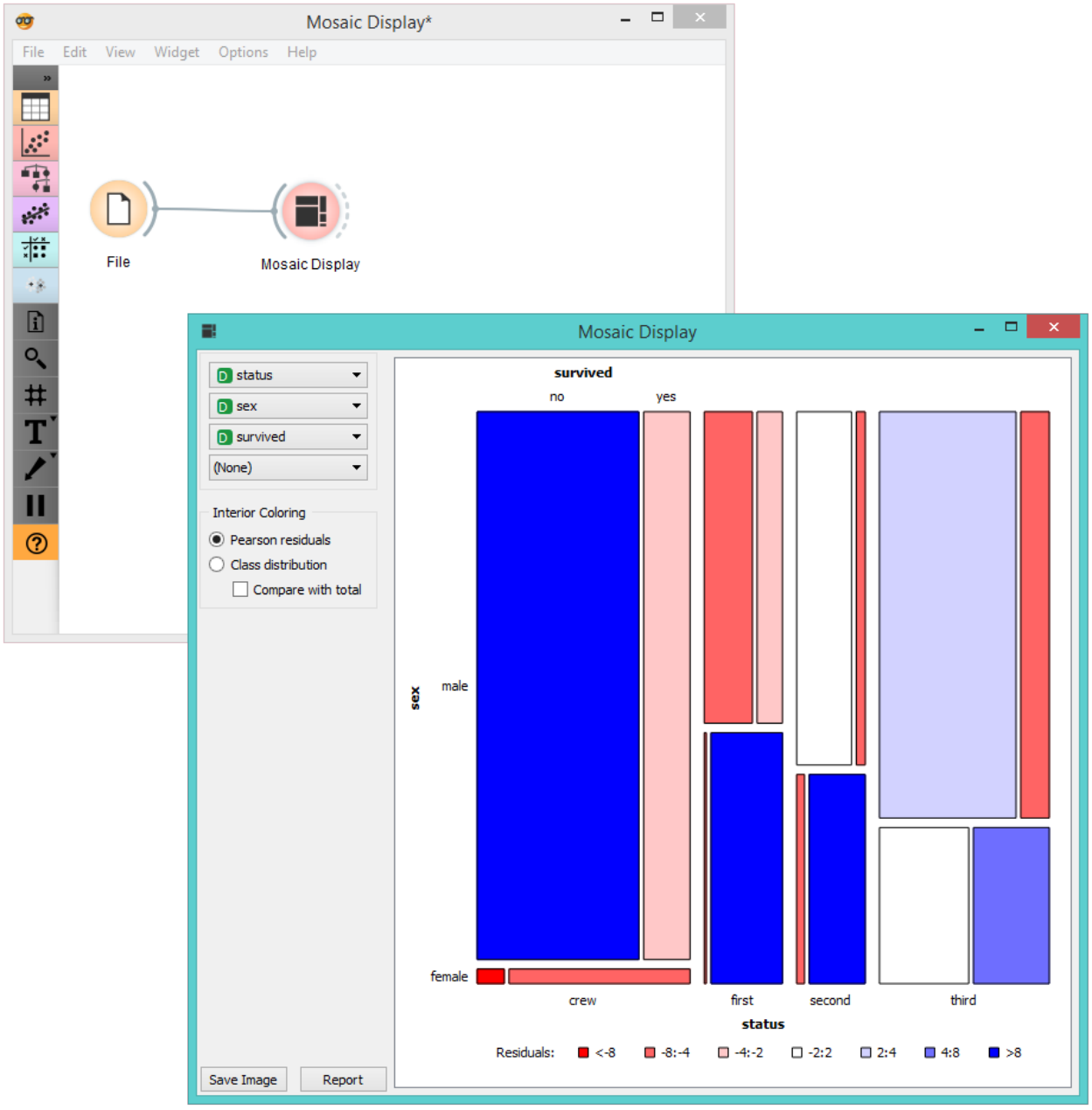
We loaded the *titanic* data set and connected it to the **Mosaic Display** widget. We decided to focus on two variables, namely status, sex and survival. We colored the interiors according to Pearson residuals in order to demonstrate the difference between observed and fitted values.

We can see that the survival rates for men and women clearly deviate from the fitted value.

Silhouette Plot

A graphical representation of consistency within clusters of data.


Mosaic Display
status
sex
(None)
(None)
Interior Coloring
Pearson residuals
Class distribution
Compare with total
Save Image
Report
sex
male
female
crew
first
second
third
status
survived:
no
yes


Mosaic Display*
File Edit View Widget Options Help
File
Mosaic Display
Mosaic Display
status
sex
survived
(None)
Interior Coloring
Pearson residuals
Class distribution
Compare with total
Save Image
Report
survived
no
yes
sex
male
female
crew
first
second
third
status
Residuals: <-8 -8:-4 -4:-2 -2:2 2:4 4:8 >8

Signals

Inputs

- **Data**

  A data set.

Outputs

- **Selected Data**

  A subset of instances that the user has manually selected from the plot.

- **Other Data**

  Remaining data.

Description

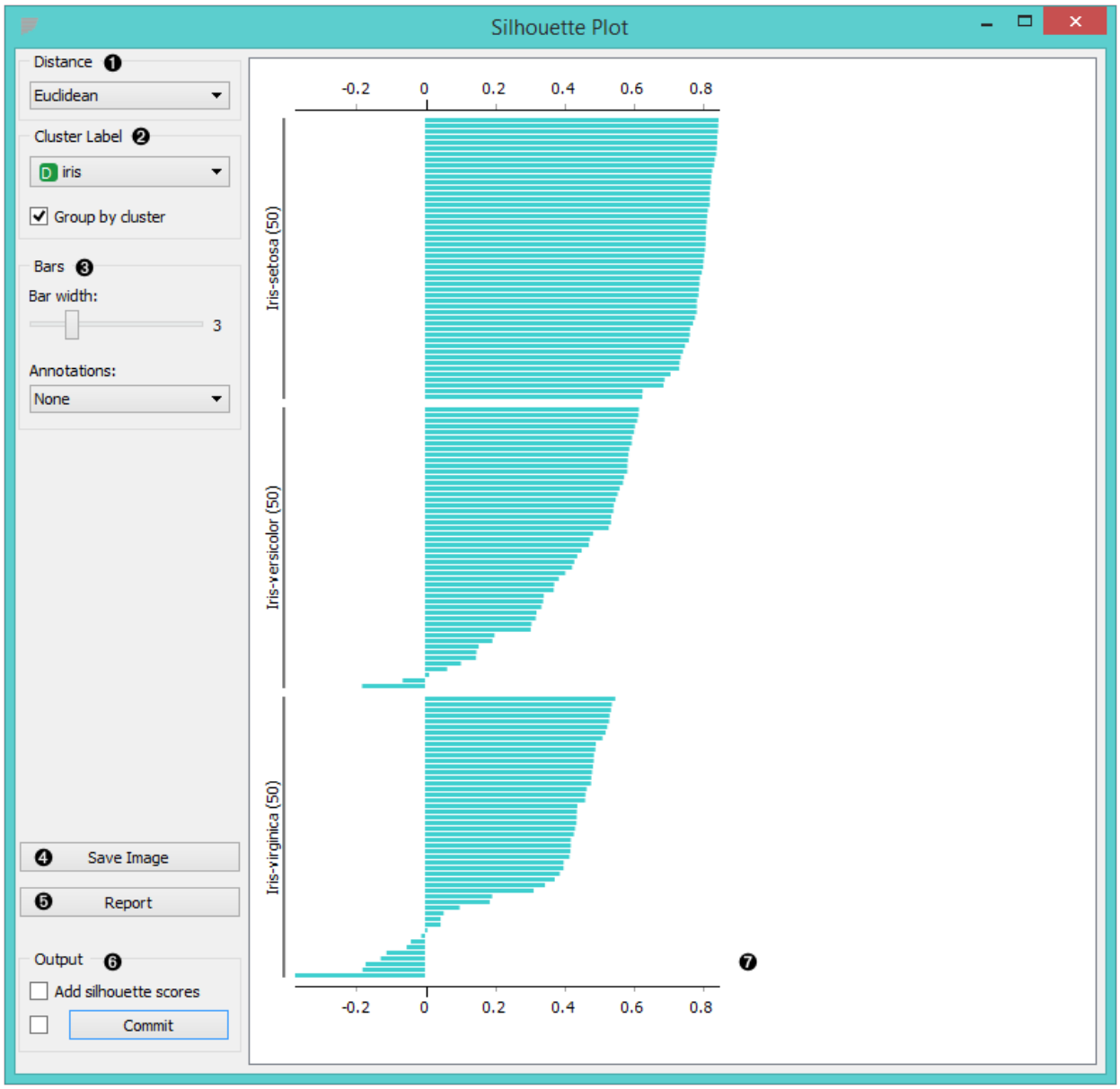
The **Silhouette Plot** widget offers a graphical representation of consistency within clusters of data and provides the user with the means to visually assess cluster quality. The silhouette score is a measure of how similar an object is to its own cluster in comparison to other clusters and is crucial in the creation of a silhoutte plot. The silhouette score close to 1 indicates that the data instance is close to the center of the cluster and instances posessing the silhouette scores close to 0 are on the border between two clusters.

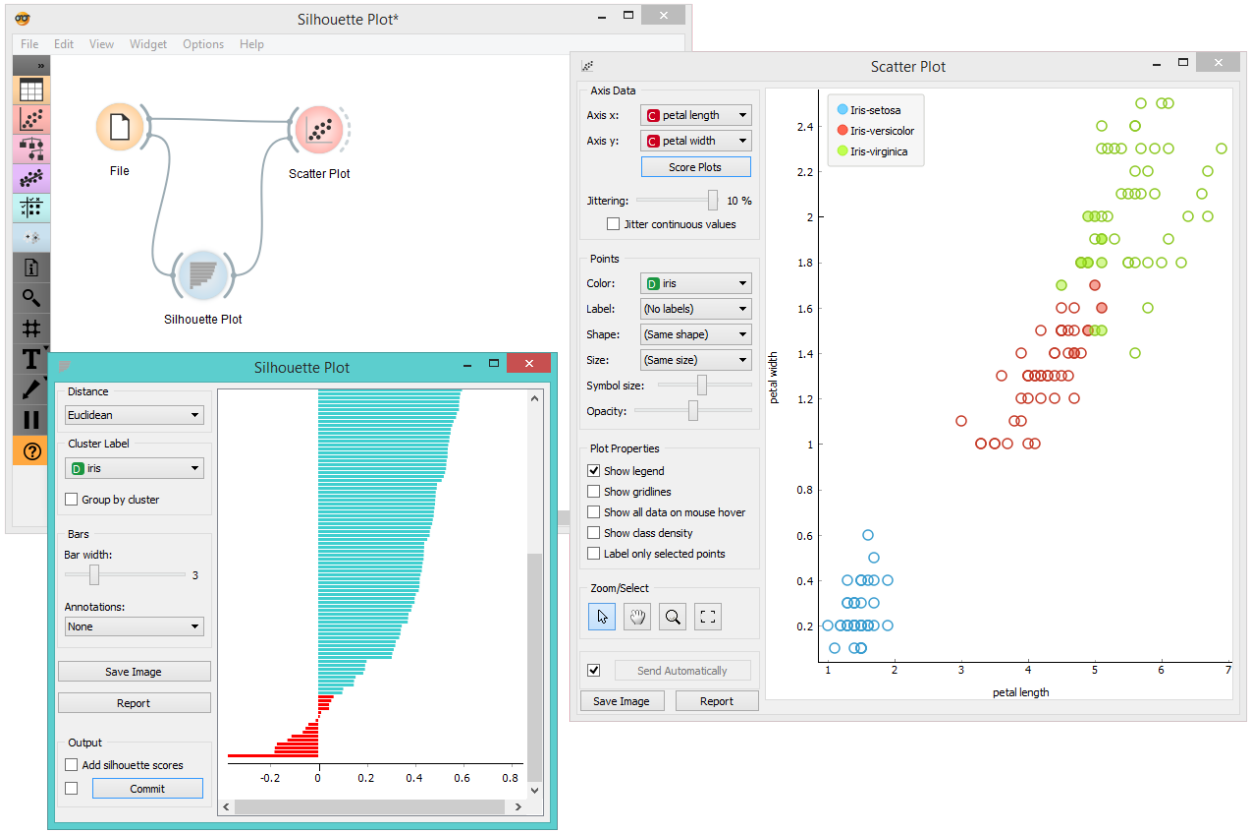
1. Choose the distance metric. You can choose between:
   - Euclidean ("straight line", distance between two points)
   - Manhattan (the sum of absolute differences for all attributes)
2. Select the cluster label. You can decide whether to group the instances by cluster or not.
3. Display options:
   - *Choose bar width*.
   - *Annotations*: annotate the silhouette plot.
4. *Save Image* saves the created silhouette plot to your computer in a *.png* or *.svg* format.
5. Produce a report.
6. Output:
   - *Add silhouette scores* (good clusters have higher silhoutte scores)
   - By clicking *Commit*, changes are comminicated to the output of the widget. Alternatively, tick the box on the left and changes will be communicated automatically.
7. The created silhouette plot.

Example

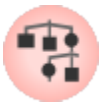
In the snapshot below, we have decided to use the **Silhoutte Plot** on the *iris* data set. We selected data intances with low silhouette scores and passed them on as a subset to the *Scatter Plot* widget. This visualization only confirms the accuracy of the **Silhouette Plot** widget, as you can clearly see that the subset lies in the border between two clusters.

If you are interested in other uses of the **Silhouette Plot** widget, feel free to explore our blog post.


Silhouette Plot
Distance ❶
Euclidean
Cluster Label ❷
iris
Group by cluster
Bars ❸
Bar width:
3
Annotations:
None
❹ Save Image
❺ Report
Output ❻
Add silhouette scores
Commit
-0.2
0
0.2
0.4
0.6
0.8
Iris-setosa (50)
Iris-versicolor (50)
Iris-virginica (50)
❼



Tree Viewer



A visualization of classification and regression trees.

Signals

Inputs:

- **Tree**

  A decision tree.

Outputs:

- **Selected Data**

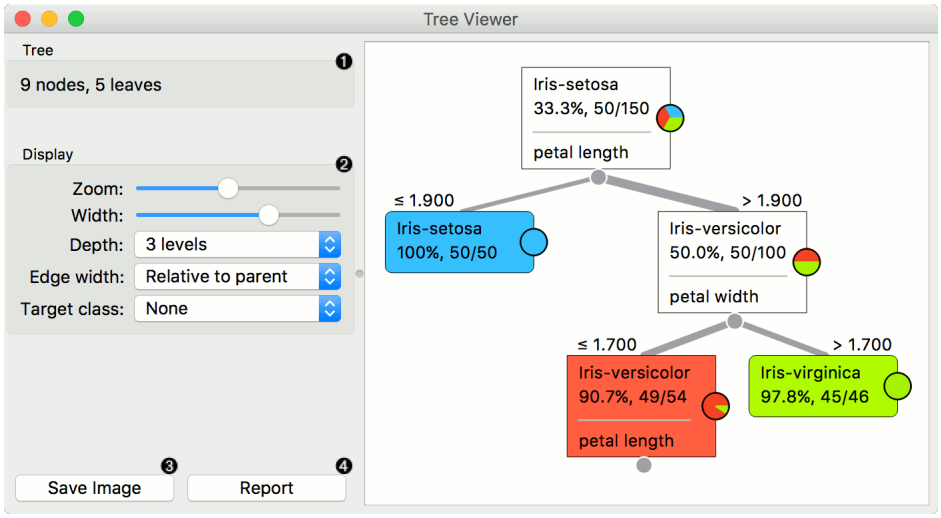
  Data from a selected tree node.

- **Data**

  Data set with an additional attribute for selection labels.

Description

This is a versatile widget with 2-D visualization of classification and regression trees. The user can select a node, instructing the widget to output the data associated with the node, thus enabling explorative data analysis.

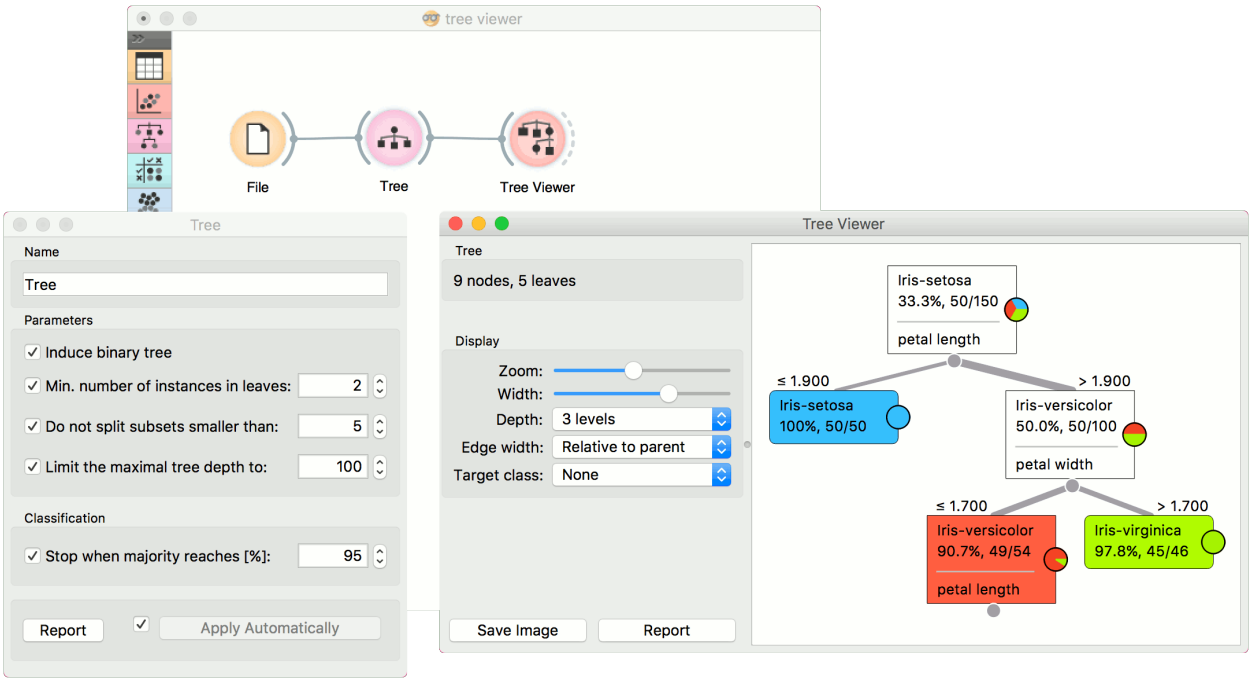


1. Information on the input.
2. Display options:
   - Zoom in and zoom out
   - Select the tree width. The nodes display information bubbles when hovering over them.
   - Select the depth of your tree.
   - **Select edge width. The edges between the nodes in the tree graph are drawn based on the selected edge width.**
     - All the edges will be of equal width if *Fixed* is chosen.
     - When *Relative to root* is selected, the width of the edge will correspond to the proportion of instances in the corresponding node with respect to all the instances in the training data. Under this selection, the edge will get thinner and thinner when traversing toward the bottom of the tree.
     - *Relative to parent* makes the edge width correspond to the proportion of instances in the nodes with respect to the instances in their parent node.
   - Define the target class, which you can change based on classes in the data.
3. Press *Save image* to save the created tree graph to your computer as a *.svg* or *.png* file.
4. Produce a report.

Examples

Below, is a simple classification schema, where we have read the data, constructed the decision tree and viewed it in our **Tree Viewer**. If both the viewer and *Tree* are open, any re-run of the tree induction algorithm will immediately affect the visualization. You can thus use this combination to explore how the parameters of the induction algorithm influence the structure of the resulting tree.

Clicking on any node will output the related data instances. This is explored in the schema below that shows the subset in the data table and in the *Scatterplot*. Make sure that the tree data is passed as a data subset; this can be done by



connecting the **Scatterplot** to the *File* widget first, and connecting it to the **Tree Viewer** widget next. Selected data will be displayed as bold dots.

Tree Viewer can also export labelled data. Connect *Data Table* to **Tree Viewer** and set the link between widgets to *Data* instead of *Selected Data*. This will send the entire data to **Data Table** with an additional meta column labelling selected data instances (*Yes* for selected and *No* for the remaining).

Finally, **Tree Viewer** can be used also for visualizing regression trees. Connect *Random Forest* to *File* widget using *housing.tab* data set. Then connect *Pythagorean Forest* to **Random Forest**. In **Pythagorean Forest** select a regression tree you wish to further analyze and pass it to the **Tree Viewer**. The widget will display the constructed tree. For visualizing larger trees, especially for regression, *Pythagorean Tree* could be a better option.

Geo Map

Show data points on a world map.

Signals

Inputs:

- **Data**

  An input data set.

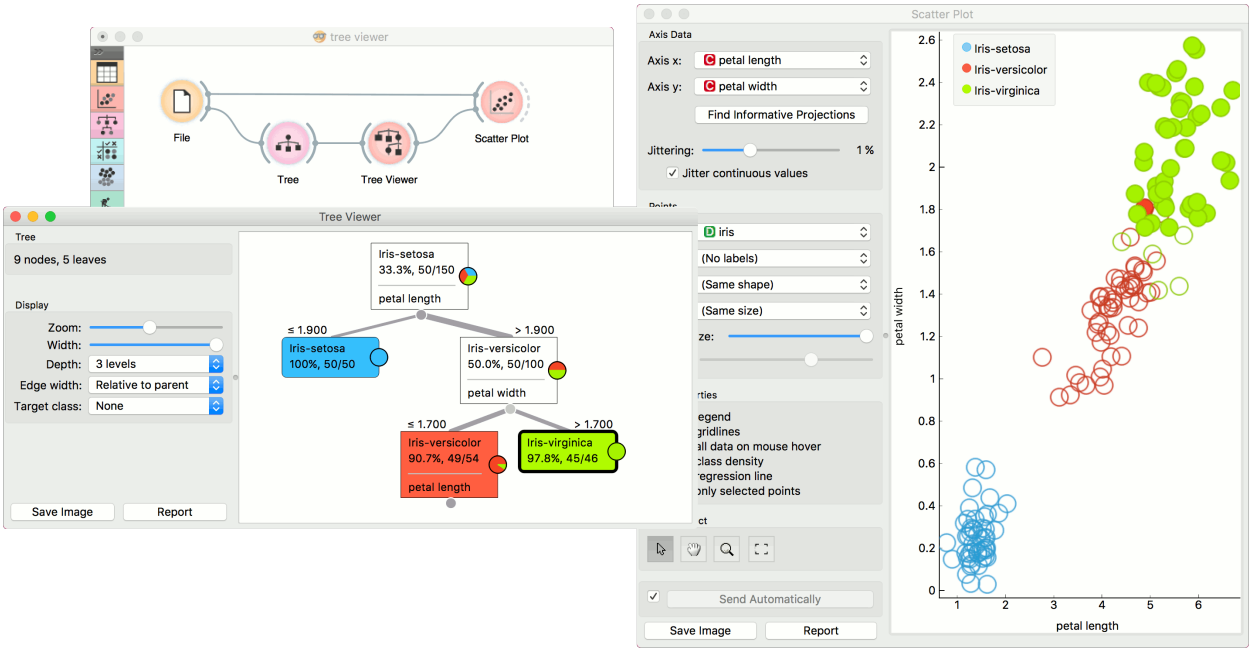
- **Data Subset**

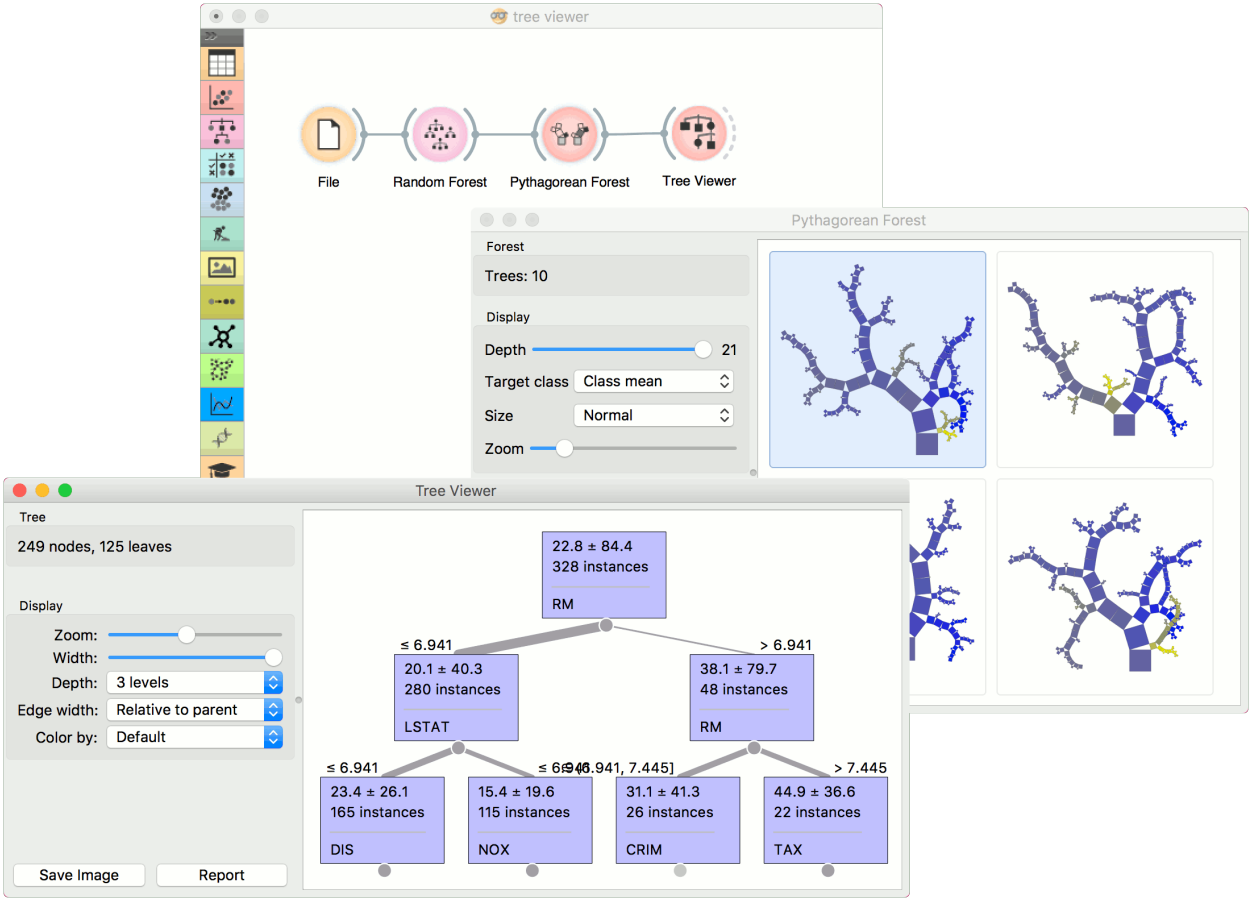
  A subset of instances from the input data set.

- **Learner**

  A learning algorithm (classification or regression).

Outputs:


tree viewer
File
Tree
Tree Viewer
Scatter Plot
Tree Viewer
Tree
9 nodes, 5 leaves
Display
Zoom:
Width:
Depth: 3 levels
Edge width: Relative to parent
Target class: None
Save Image
Report
Iris-setosa
33.3%, 50/150
petal length
≤ 1.900
> 1.900
Iris-setosa
100%, 50/50
Iris-versicolor
50.0%, 50/100
petal width
≤ 1.700
> 1.700
Iris-versicolor
90.7%, 49/54
petal length
Iris-virginica
97.8%, 45/46
Scatter Plot
Axis Data
Axis x: petal length
Axis y: petal width
Find Informative Projections
Jittering: 1 %
Jitter continuous values
iris
(No labels)
(Same shape)
(Same size)
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal width
petal length


tree viewer
File
Random Forest
Pythagorean Forest
Tree Viewer
Pythagorean Forest
Forest
Trees: 10
Display
Depth 21
Target class Class mean
Size Normal
Zoom
Tree Viewer
Tree
249 nodes, 125 leaves
Display
Zoom:
Width:
Depth: 3 levels
Edge width: Relative to parent
Color by: Default
Save Image
Report
22.8 ± 84.4
328 instances
RM
≤ 6.941
> 6.941
20.1 ± 40.3
280 instances
LSTAT
38.1 ± 79.7
48 instances
RM
≤ 6.941
≤ 6.946
(6.941, 7.445]
> 7.445
23.4 ± 26.1
165 instances
DIS
15.4 ± 19.6
115 instances
NOX
31.1 ± 41.3
26 instances
CRIM
44.9 ± 36.6
22 instances
TAX

- **Selected Data**

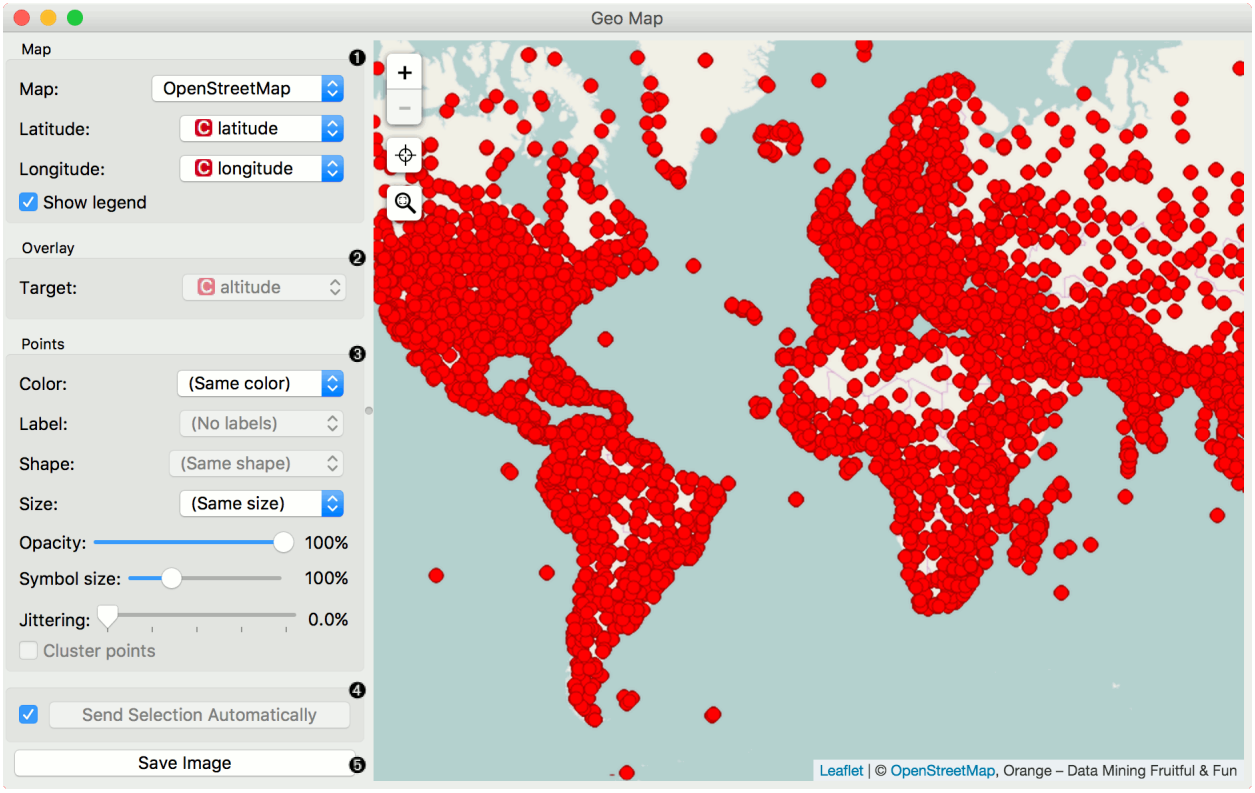
  A subset of instances that the user has manually selected from the map.

- **Data**

  Data set with an appended meta attribute specifying selected and unselected data.

Description

Geo Map widget maps geo-spatial data on a world map. It only works on data sets containing latitude and longitude variables. It also enables class predictions when a learner is provided on the input.



1. Define map properties: - Set the type of *map*: Black and White, OpenStreetMap, Topographic, Satellite, Print, Light, Dark, Railways and Watercolor. - Set latitude and longitude attributes, if the widget didn't recognize them automatically. Latitude values should be between -90(S) and 90(N) and longitude values between -180(W) and 180(E).
2. Overlay: - Set the target (class) for predictive mapping. A learner has to be provided on the input. The classifier is trained on latitude and longitude pairs only (i.e. it maps lat/lon pairs to the selected attribute).
3. Set point parameters: - *Color*: color of data points by attribute values - *Label*: label data points with an attribute (available when zoomed in) - *Shape*: shape of data points by attribute (available when zoomed in) - *Size*: size of data points by attribute - Opacity: set transparency of data points - Symbol size: size of data points (small to large) - Jittering: disperse overlaid data points - Cluster points: cluster neighboring points with naive greedy clustering (available when less than 600 points are in view)
4. If *Send Selection Automatically* is ticked, changes are communicated automatically. Alternatively, click *Send Selection*. *Save image* saves the image to your computer in a .svg or .png format.

Note: To select a subset of points from the map, hold Shift and draw a rectangle around the point you want to output.

Examples

In the first example we will model class predictions on a map. We will use *philadelphia-crime* data set, load it with *File* widget and connect it to **Map**. We can already observe the mapped points in Map. Now, we connect *Tree* to Map and set target variable to Type. This will display the predicted type of crime for a specific region of Philadelphia city (each region will be colored with a corresponding color code, explained in a legend on the right).

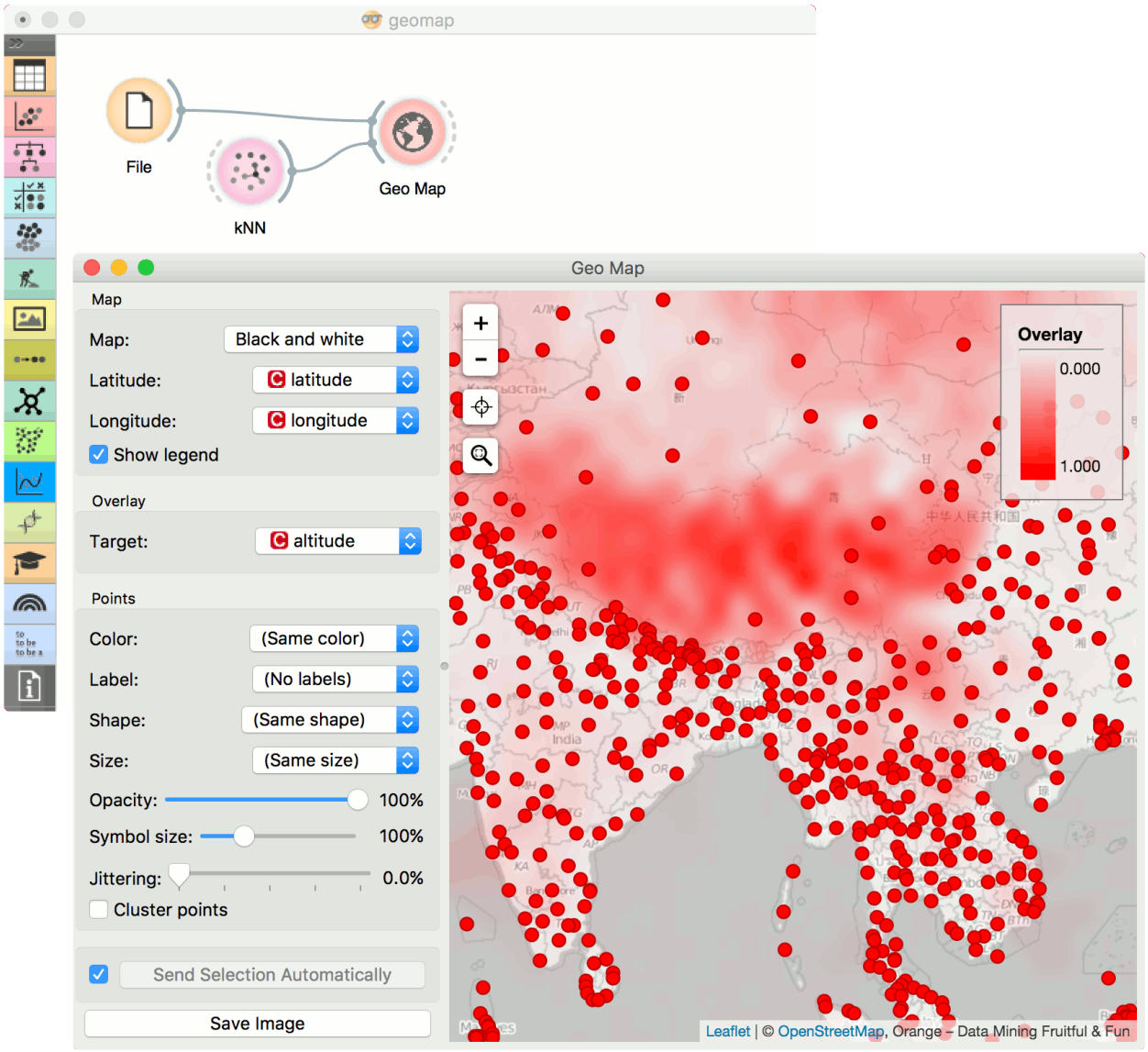
The second example uses global-airports.csv data. Say we somehow want to predict the altitude of the area based soley on the latitude and longitude. We again load the data with *File* widget and connect it to Map. Then we use a regressor, say, *KNN* and connect it to Map as well. Now we set target to altitude and use Black and White map type. The model guessed the Himalaya, but mades some errors elsewhere.

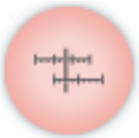
Nomogram

Nomograms for visualization of Naive Bayes and Logistic Regression classifiers.

Signals

Inputs:


geomap
File
kNN
Geo Map
Geo Map
Map
Map: Black and white
Latitude: latitude
Longitude: longitude
Show legend
Overlay
Target: altitude
Points
Color: (Same color)
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Opacity: 100%
Symbol size: 100%
Jittering: 0.0%
Cluster points
Send Selection Automatically
Save Image
Overlay
0.000
1.000
Leaflet | © OpenStreetMap, Orange – Data Mining Fruitful & Fun



- **Classifier**

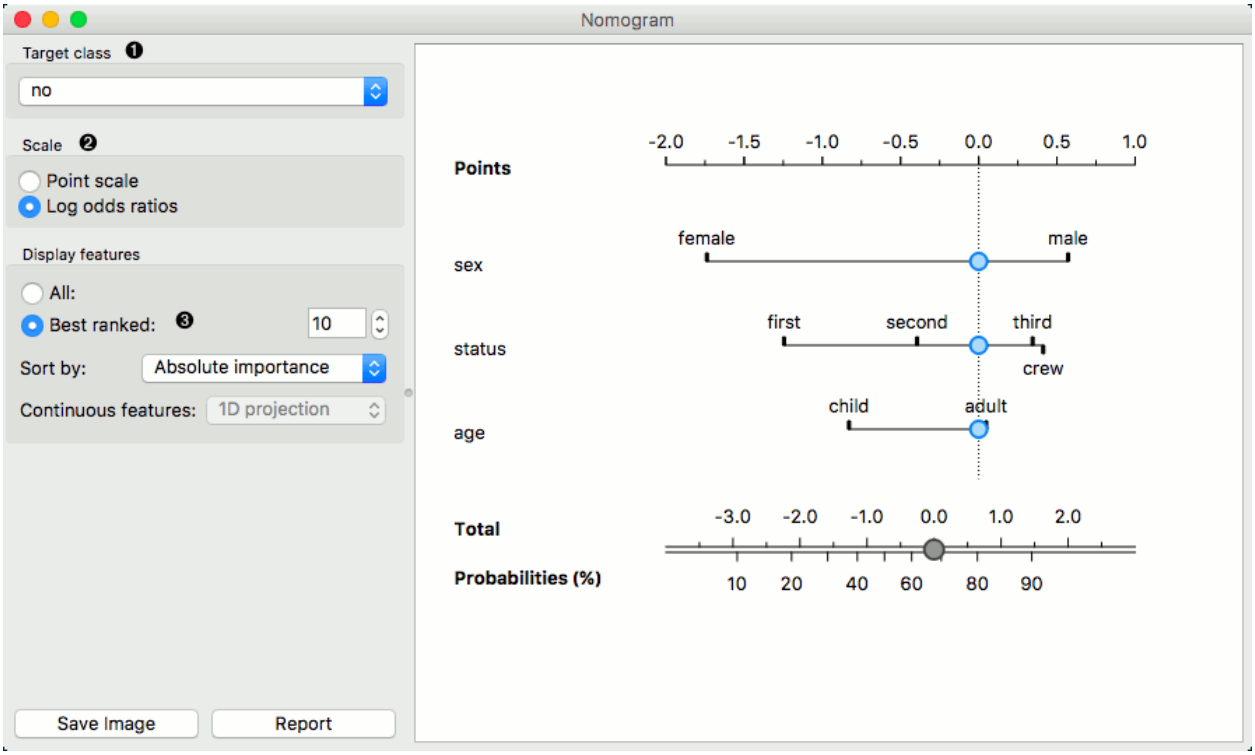
  A trained classifier (Naive Bayes or Logistic regression).

- **Data**

  Data instance.

Description

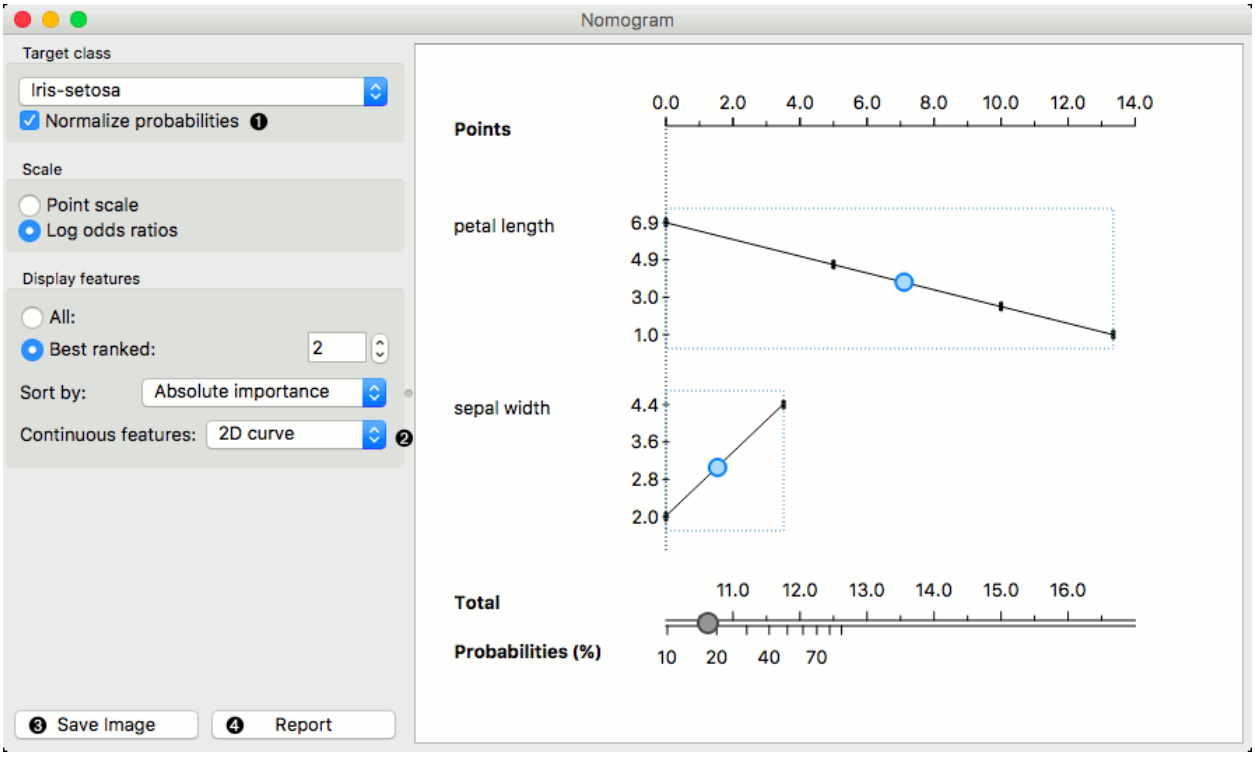
The **Nomogram** enables some classifier's (more precisely Naive Bayes classifier and Logistic Regression classifier) visual representation. It offers an insight into the structure of the training data and effects of the attributes on the class probabilities. Besides visualization of the classifier, the widget offers interactive support to prediction of class probabilities. A snapshot below shows the nomogram of the Titanic data set, that models the probability for a passenger not to survive the disaster of the Titanic.



1. Select the target class you want to model the probability for.
2. By default Scale is set to Log odds ration. For easier understanding and interpretation option Point scale can be used. The unit is obtained by re-scaling the log odds so that the maximal absolute log odds ratio in the nomogram represents 100 points.
3. When there are to many attributes in the plotted data set, you can choose to display only best ranked ones. It is possible to choose from 'No sorting', 'Name', 'Absolute importance', 'Positive influence' and 'Negative influence' for Naive Bayes representation and from 'No sorting', 'Name' and 'Absolute importance' for Logistic Regression representation.

To represent nomogram for Logistic Regressing classifier Iris data set is used:

1. The probability for the chosen target class is computed by 1. vs. all principle, which should be taken in consideration when dealing with multiclass data (alternating probabilities do not sum to 1). To avoid this inconvenience, you can choose to normalize probabilities.



2. Continuous attributes can be plotted in 2D (only for Logistic Regression).
3. Save image.
4. Produce a report.

Example

The **Nomogram** widget should be used immediately after trained classifier widget (e.g. *Naive Bayes*. It can also be passed a data instance using any widget that enables selection (e.g. *Data Table*) as shown in the workflow below.

Referring to the Titanic data set once again, 1490 (68%) of passengers on Titanic, of 2201 in total, died. To make a prediction, the contribution of each attribute is measured as a point score and the individual point scores are summed to determine the probability. When the value of the attribute is unknown, its contribution is 0 points. Therefore, not knowing anything about the passenger, the total point score is 0, and the corresponding probability equals to the unconditional prior. The nomogram in the example shows the case when we know that the passenger is a male adult from the first class. The points sum to -0.36, with a corresponding probability of not surviving of about 53%.

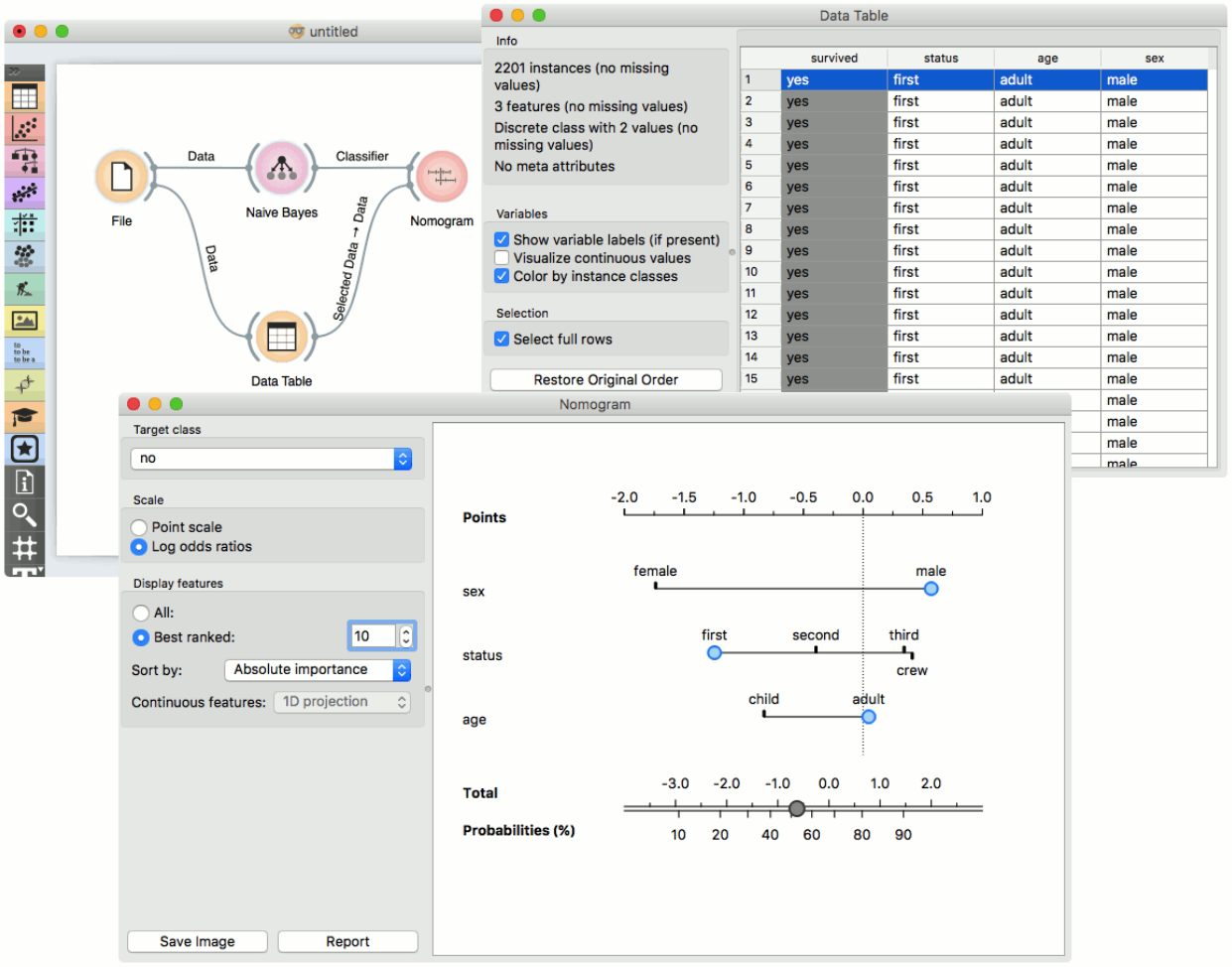
Model

Naive Bayes

A fast and simple probabilistic classifier based on Bayes' theorem with the assumption of feature independence.

Signals

Inputs:


untitled
File
Data
Naive Bayes
Classifier
Nomogram
Data
Selected Data → Data
Data Table
Data Table
Info
2201 instances (no missing values)
3 features (no missing values)
Discrete class with 2 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
survived status age sex
1 yes first adult male
2 yes first adult male
3 yes first adult male
4 yes first adult male
5 yes first adult male
6 yes first adult male
7 yes first adult male
8 yes first adult male
9 yes first adult male
10 yes first adult male
11 yes first adult male
12 yes first adult male
13 yes first adult male
14 yes first adult male
15 yes first adult male
Nomogram
Target class
no
Scale
Point scale
Log odds ratios
Display features
All:
Best ranked: 10
Sort by: Absolute importance
Continuous features: 1D projection
Save Image
Report
Points
-2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0
sex
female
male
status
first
second
third
crew
age
child
adult
Total
-3.0 -2.0 -1.0 0.0 1.0 2.0
Probabilities (%)
10 20 40 60 80 90

- **Data**

  A data set

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A naive bayes learning algorithm with settings as specified in the dialog.

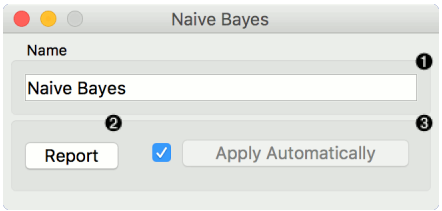
- **Model**

  A trained classifier. Output signal sent only if input *Data* is present.

Description

Naive Bayes learns a Naive Bayesian model from the data.

It only works for classification tasks.



This widget has two options: the name under which it will appear in other widgets and producing a report. The default name is *Naive Bayes*. When you change it, you need to press *Apply*.

Examples

Here, we present two uses of this widget. First, we compare the results of the **Naive Bayes** with another model, the *Random Forest*. We connect *iris* data from *File* to *Test&Score*. We also connect **Naive Bayes** and *Random Forest* to **Test & Score** and observe their prediction scores.

The second schema shows the quality of predictions made with **Naive Bayes**. We feed the *Test&Score* widget a Naive Bayes learner and then send the data to the *Confusion Matrix*. We also connect *Scatterplot* with **File**. Then we select the misclassified instances in the **Confusion Matrix** and show feed them to *Scatterplot*. The bold dots in the scatterplot are the misclassified instances from **Naive Bayes**.

Logistic Regression

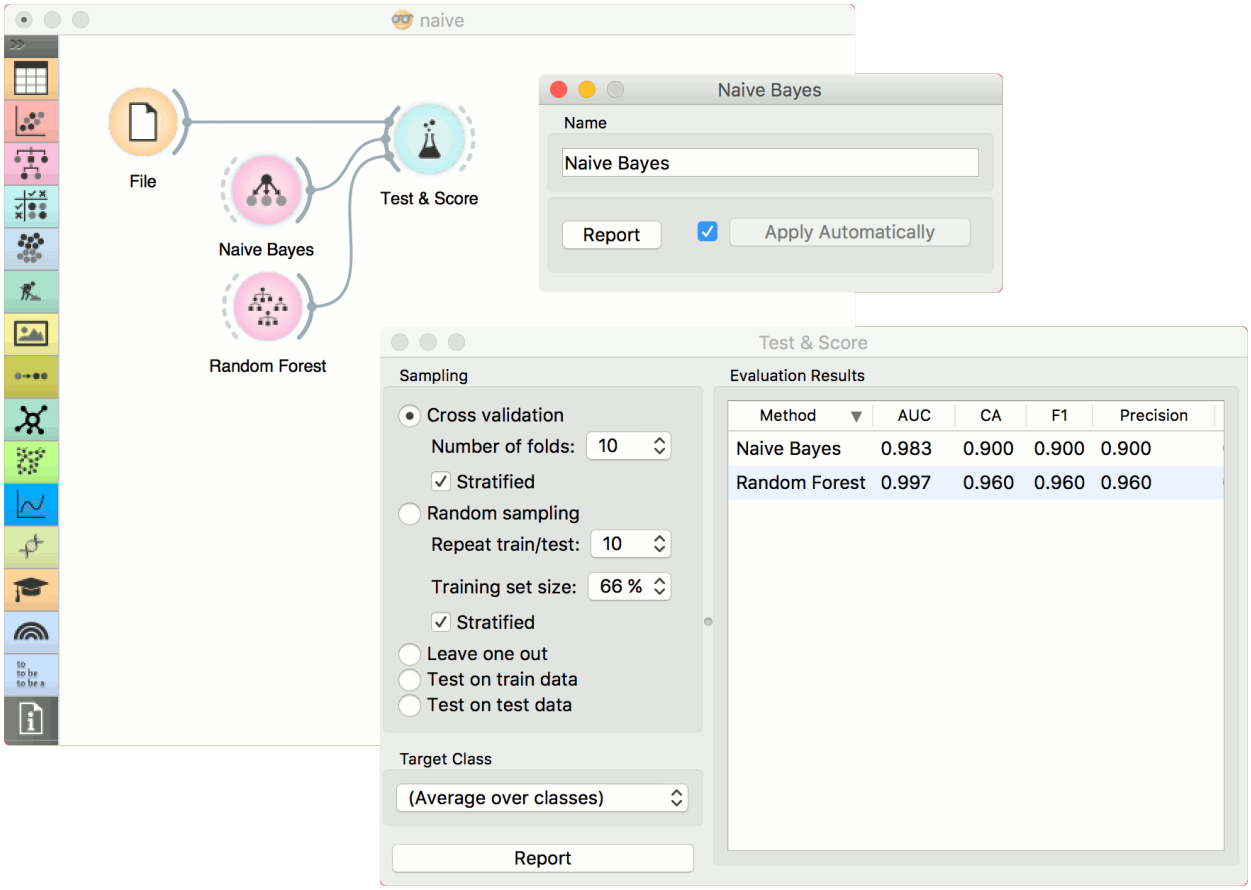
The logistic regression classification algorithm with LASSO (L1) or ridge (L2) regularization.

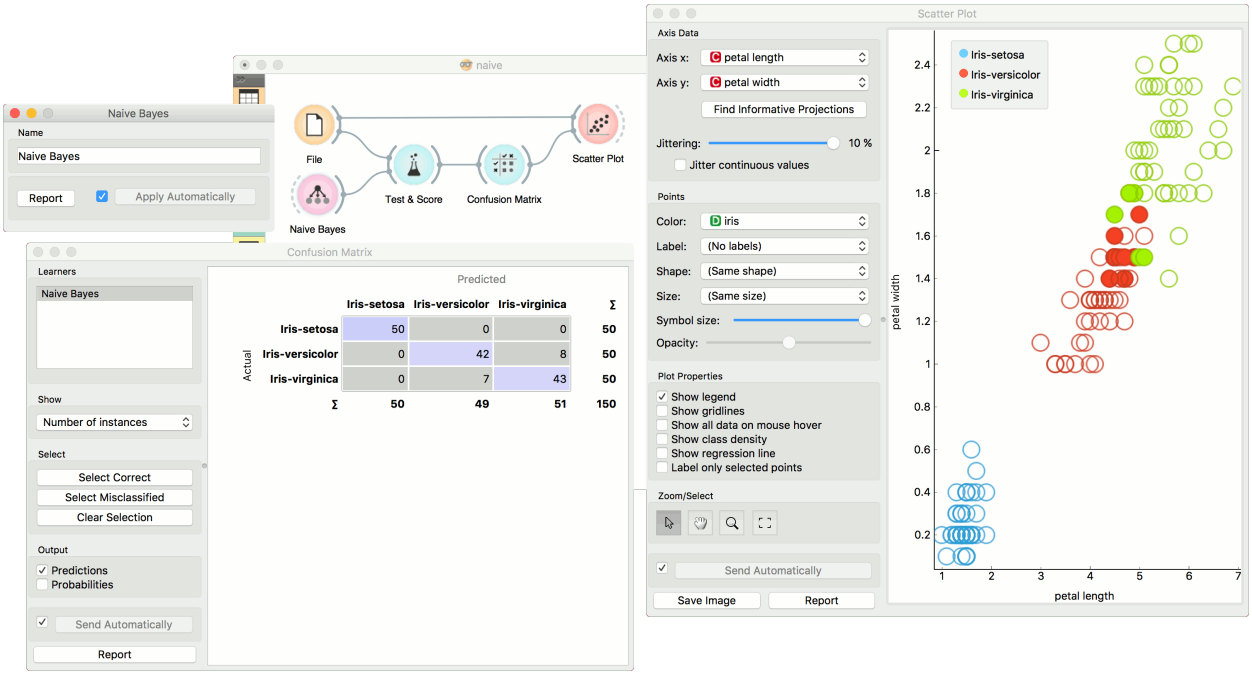
Signals

Inputs:

- **Data**

  A data set


naive
File
Naive Bayes
Test & Score
Random Forest
Naive Bayes
Name
Naive Bayes
Report
Apply Automatically
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision
Naive Bayes 0.983 0.900 0.900 0.900
Random Forest 0.997 0.960 0.960 0.960


Naive Bayes
Name
Naive Bayes
Report
Apply Automatically
naive
File
Test & Score
Scatter Plot
Naive Bayes
Confusion Matrix
Confusion Matrix
Learners
Naive Bayes
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predictions
Probabilities
Send Automatically
Report
Predicted
Iris-setosa Iris-versicolor Iris-virginica Σ
Iris-setosa 50 0 0 50
Iris-versicolor 0 42 8 50
Iris-virginica 0 7 43 50
Σ 50 49 51 150
Scatter Plot
Axis Data
Axis x: petal length
Axis y: petal width
Find Informative Projections
Jittering: 10 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Show regression line
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal width
petal length

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A logistic regression learning algorithm with settings as specified in the dialog.

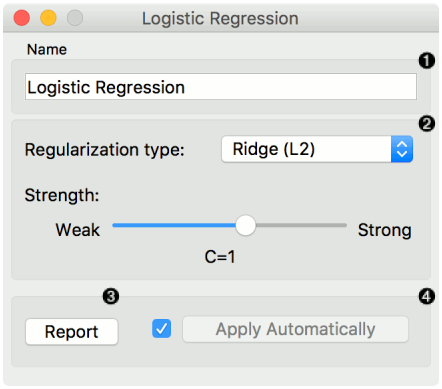
- **Logistic Regression Classifier**

  A trained classifier. Output signal sent only if input *Data* is present.

Description

Logistic Regression learns a Logistic Regression model from the data.

It only works for classification tasks.



1. A name under which the learner appears in other widgets. The default name is "Logistic Regression".
2. Regularization type (either L1 or L2). Set the cost strength (default is C=1).
3. Press *Apply* to commit changes. If *Apply Automatically* is ticked, changes will be communicated automatically.

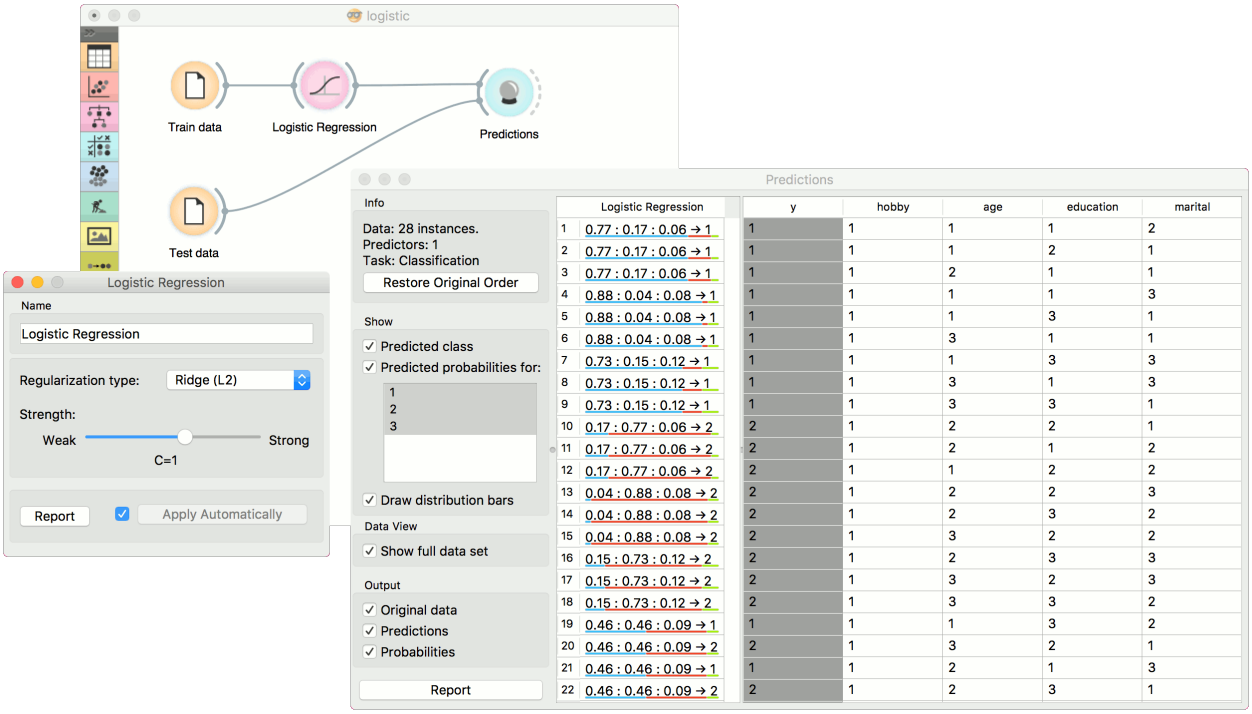
Example

The widget is used just as any other widget for inducing a classifier. This is an example demonstrating prediction results with logistic regression on the *hayes-roth* data set. We first load *hayes-roth_learn* in the *File* widget and pass the data to **Logistic Regression**. Then we pass the trained model to *Predictions*.

Now we want to predict class value on a new data set. We load *hayes-roth_test* in the second **File** widget and connect it to **Predictions**. We can now observe class values predicted with **Logistic Regression** directly in **Predictions**.

Tree

A tree algorithm with forward pruning.



Signals

Inputs:

- **Data**

  A data set

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A decision tree learning algorithm with settings as specified in the dialog.

- **Model**

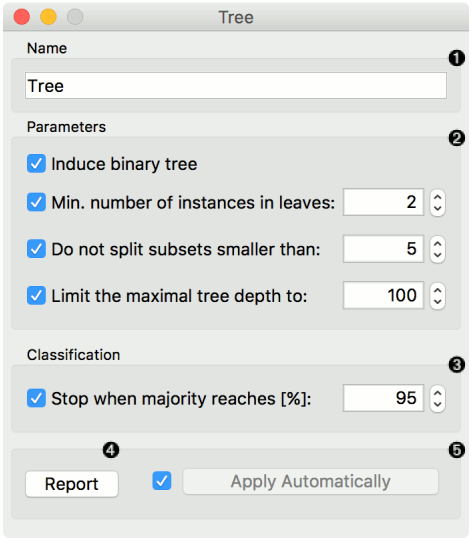
  A subset of data instances from the training set that were used as support vectors in the trained model.

Description

Tree is a simple algorithm that splits the data into nodes by class purity. It is a precursor to *Random Forest*. Tree in Orange is designed in-house and can handle both discrete and continuous data sets.

It can also be used for both classification and regression tasks.

1. The learner can be given a name under which it will appear in other widgets. The default name is “Tree”.



2. Tree parameters: - **Induce binary tree**: build a binary tree (split into two child nodes) - **Min. number of instances in leaves**: if checked, the algorithm will never construct a split which would put less than the specified number of training examples into any of the branches. - **Do not split subsets smaller than**: forbids the algorithm to split the nodes with less than the given number of instances. - **Limit the maximal tree depth**: limits the depth of the classification tree to the specified number of node levels.
3. **Stop when majority reaches [%]**: stop splitting the nodes after a specified majority threshold is reached
4. Produce a report. After changing the settings, you need to click *Apply*, which will put the new learner on the output and, if the training examples are given, construct a new classifier and output it as well. Alternatively, tick the box on the left and changes will be communicated automatically.

Examples

There are two typical uses for this widget. First, you may want to induce a model and check what it looks like in *Tree Viewer*.

The second schema trains a model and evaluates its performance against *Logistic Regression*.

We used the *iris* data set in both examples. However, **Tree** works for regression tasks as well. Use *housing* data set and pass it to **Tree**. The selected tree node from *Tree Viewer* is presented in the *Scatter Plot* and we can see that the selected examples exhibit the same features.

kNN

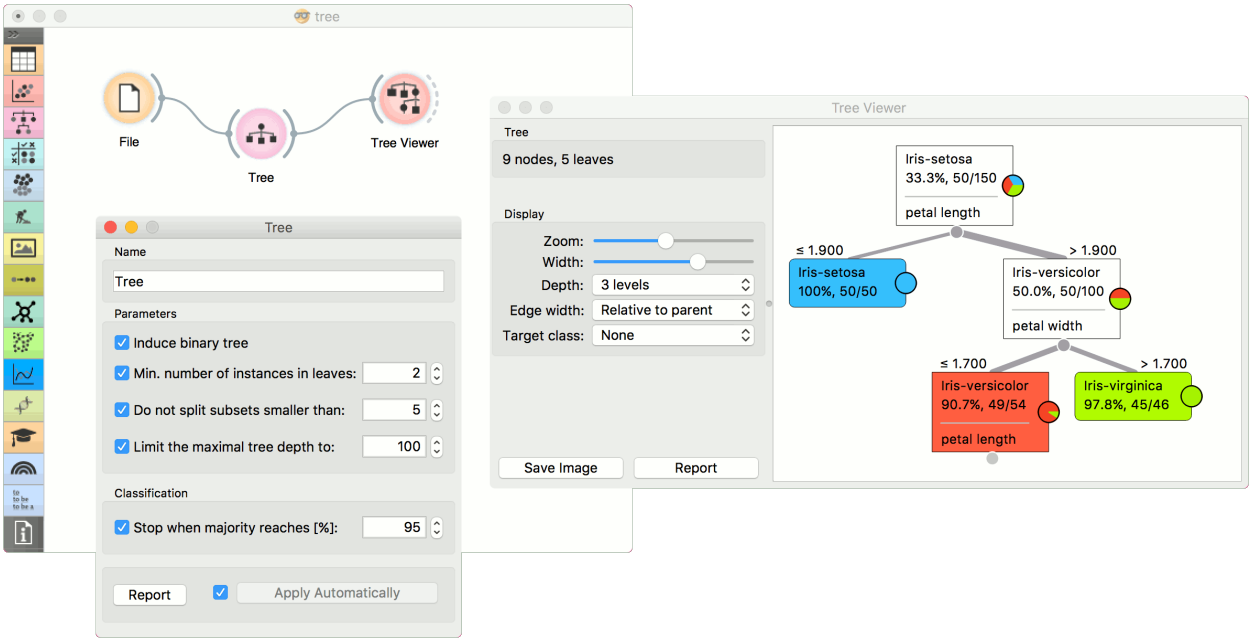
Predict according to the nearest training instances.

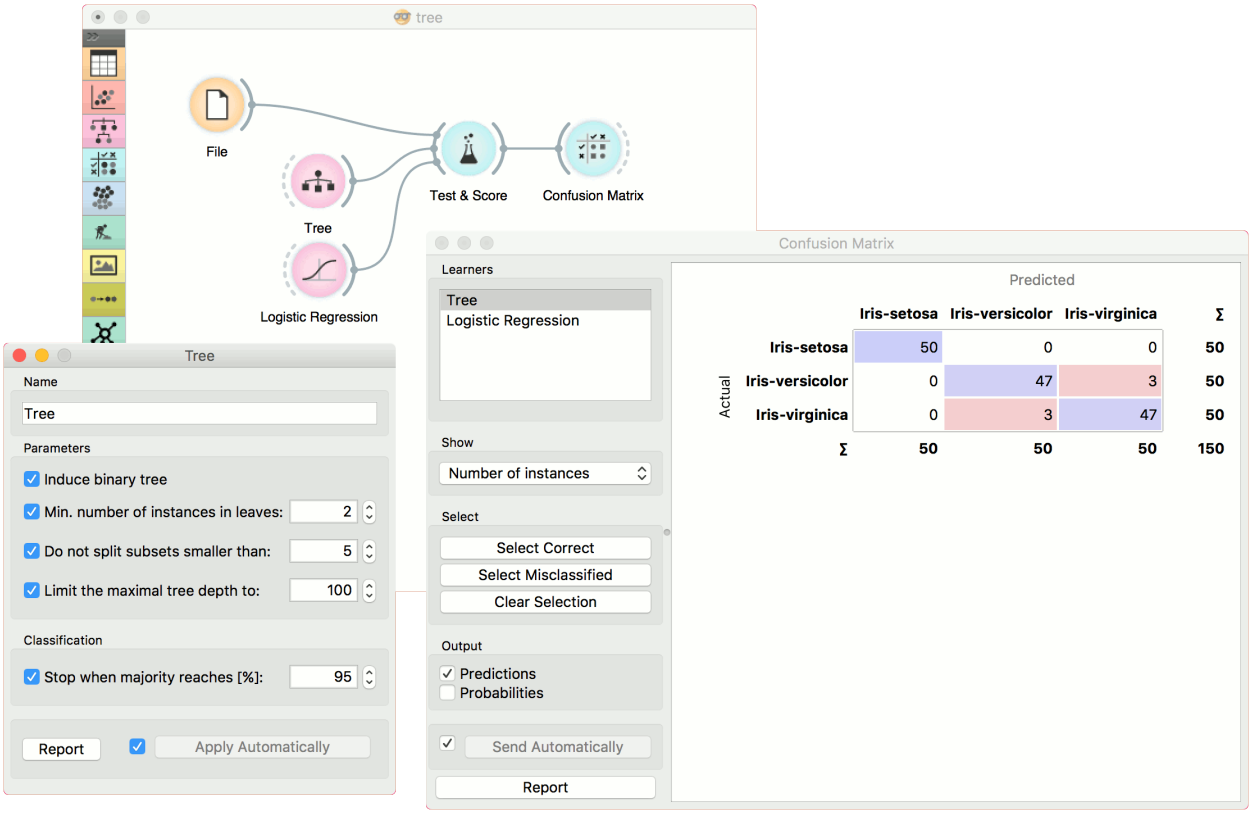
Signals

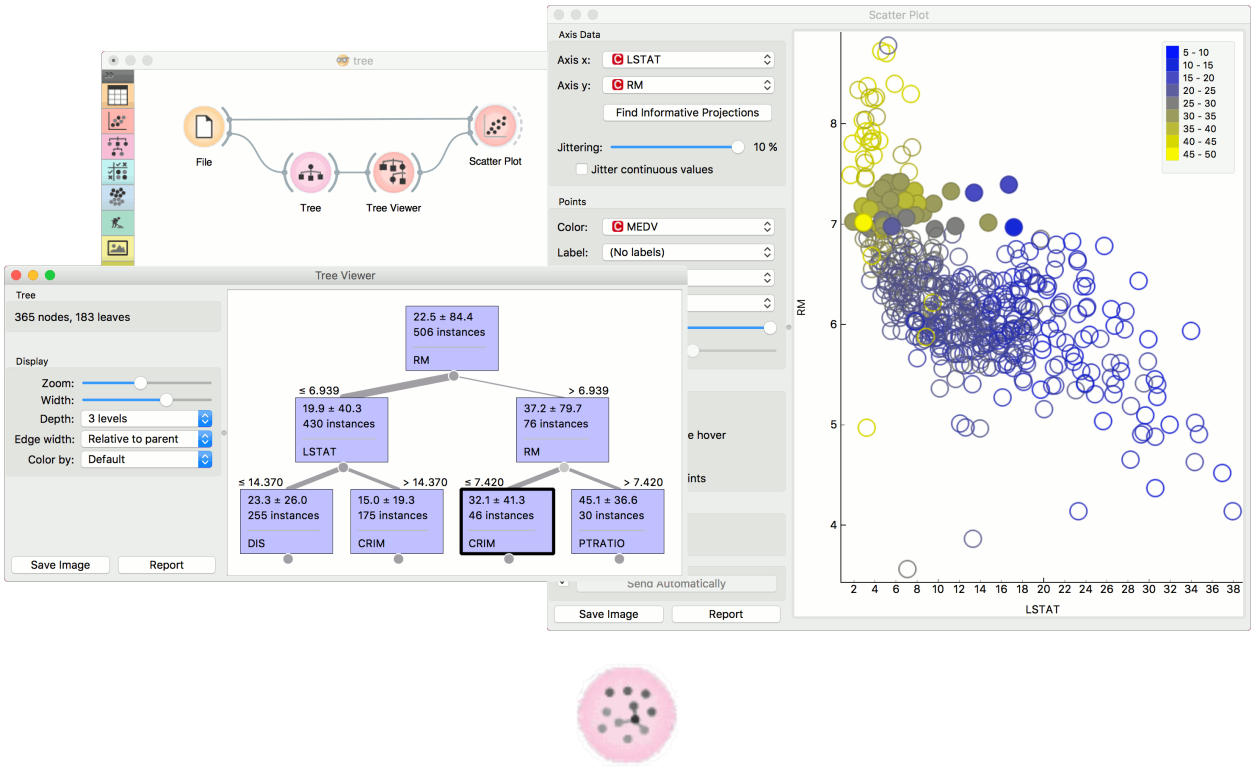
Inputs:

- **Data**

  A data set


tree
File
Tree
Tree Viewer
Tree
Name
Tree
Parameters
Induce binary tree
Min. number of instances in leaves: 2
Do not split subsets smaller than: 5
Limit the maximal tree depth to: 100
Classification
Stop when majority reaches [%]: 95
Report
Apply Automatically
Tree Viewer
Tree
9 nodes, 5 leaves
Display
Zoom:
Width:
Depth: 3 levels
Edge width: Relative to parent
Target class: None
Save Image
Report
Iris-setosa
33.3%, 50/150
petal length
≤ 1.900
> 1.900
Iris-setosa
100%, 50/50
Iris-versicolor
50.0%, 50/100
petal width
≤ 1.700
> 1.700
Iris-versicolor
90.7%, 49/54
petal length
Iris-virginica
97.8%, 45/46


tree
File
Tree
Logistic Regression
Test & Score
Confusion Matrix
Tree
Name
Tree
Parameters
Induce binary tree
Min. number of instances in leaves: 2
Do not split subsets smaller than: 5
Limit the maximal tree depth to: 100
Classification
Stop when majority reaches [%]: 95
Report
Apply Automatically
Confusion Matrix
Learners
Tree
Logistic Regression
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predictions
Probabilities
Send Automatically
Report
Predicted
Actual
Iris-setosa
Iris-versicolor
Iris-virginica
Σ
50
0
0
50
0
47
3
50
0
3
47
50
50
50
50
150



- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

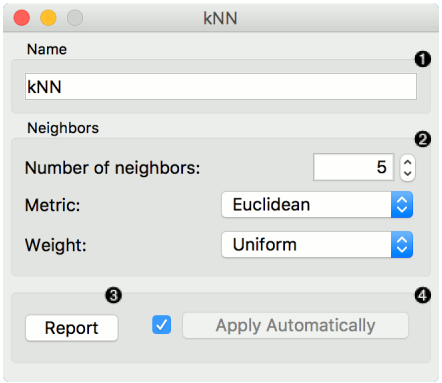
  A kNN learning algorithm with settings as specified in the dialog.

- **Model**

  A trained model. Output signal sent only if input *Data* is present.

Description

The **kNN** widget uses the kNN algorithm that searches for k closest training examples in feature space and uses their average as prediction.

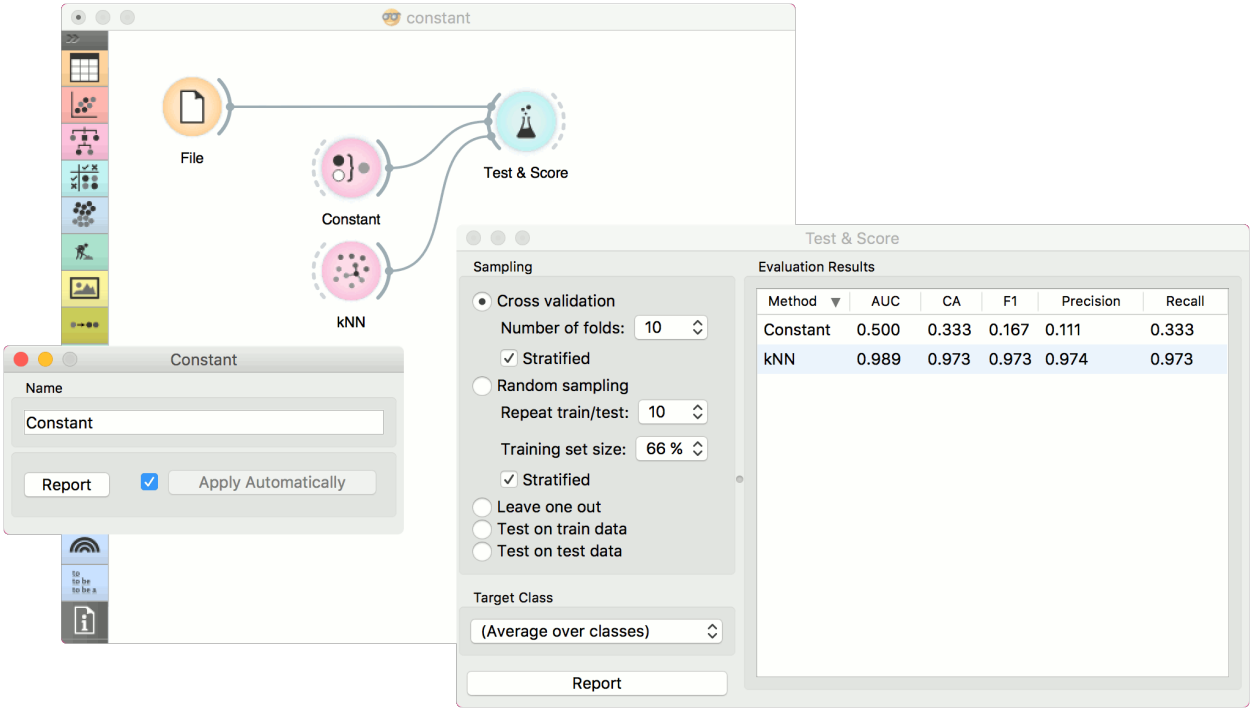


1. A name under which it will appear in other widgets. The default name is “kNN”.
2. Set the number of nearest neighbors, the distance parameter (metric) and weights as model criteria. Metric can be:
   - Euclidean (“straight line”, distance between two points)
   - Manhattan (sum of absolute differences of all attributes)
   - Maximal (greatest of absolute differences between attributes)
   - Mahalanobis (distance between point and distribution).

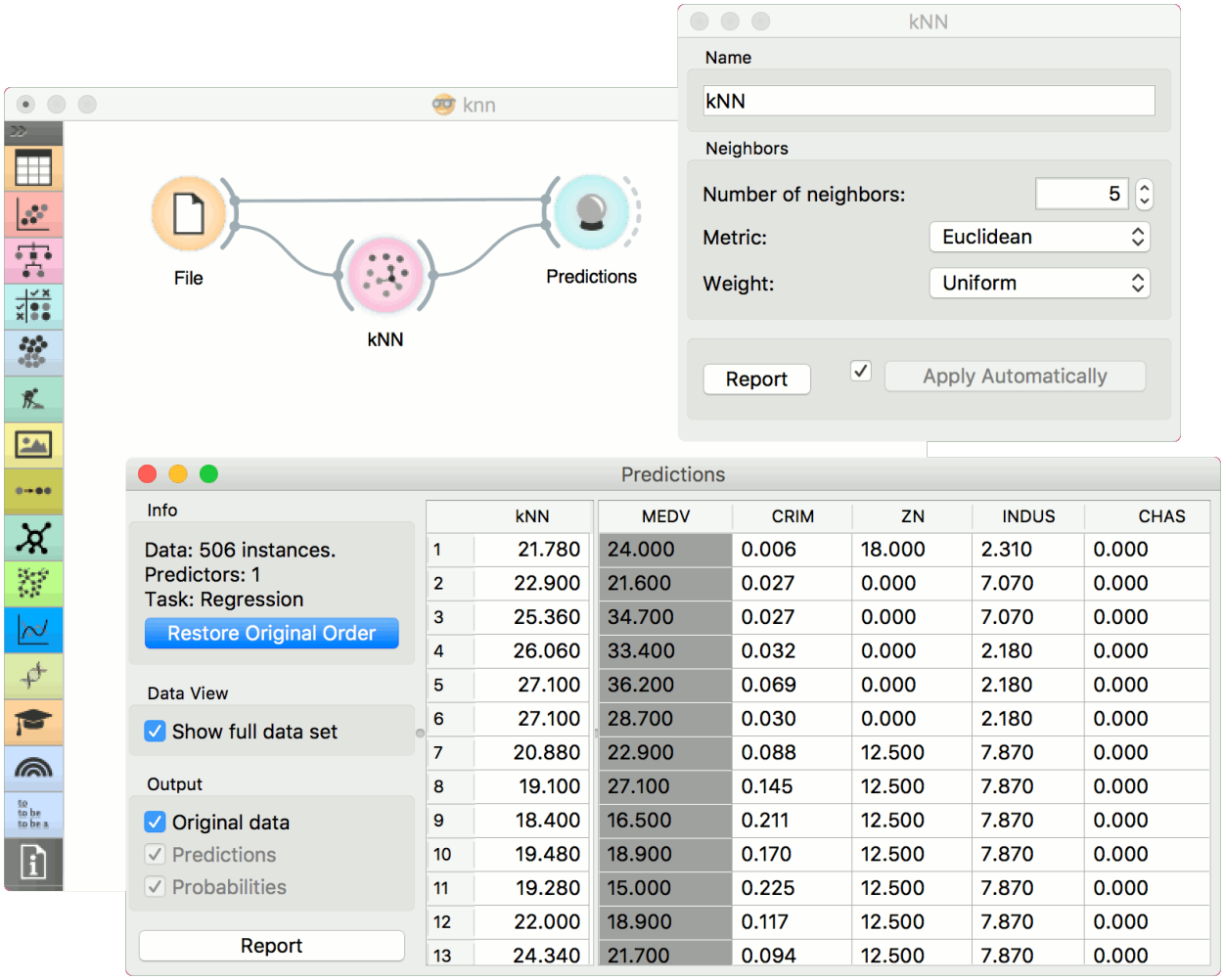
   The *Weights* you can use are:
   - **Uniform**: all points in each neighborhood are weighted equally.
   - **Distance**: closer neighbors of a query point have a greater influence than the neighbors further away.
3. Produce a report.
4. When you change one or more settings, you need to click *Apply*, which will put a new learner on the output and, if the training examples are given, construct a new model and output it as well. Changes can also be applied automatically by clicking the box on the left side of the *Apply* button.

Examples

The first example is a classification task on *iris* data set. We compare the results of k-Nearest neighbors with the default model *Constant*, which always predicts the majority class.



The second example is a regression task. This workflow shows how to use the *Learner* output. For the purpose of this example, we used the *housing* data set. We input the **kNN** prediction model into *Predictions* and observe the predicted values.



	kNN	MEDV	CRIM	ZN	INDUS	CHAS
1	21.780	24.000	0.006	18.000	2.310	0.000
2	22.900	21.600	0.027	0.000	7.070	0.000
3	25.360	34.700	0.027	0.000	7.070	0.000
4	26.060	33.400	0.032	0.000	2.180	0.000
5	27.100	36.200	0.069	0.000	2.180	0.000
6	27.100	28.700	0.030	0.000	2.180	0.000
7	20.880	22.900	0.088	12.500	7.870	0.000
8	19.100	27.100	0.145	12.500	7.870	0.000
9	18.400	16.500	0.211	12.500	7.870	0.000
10	19.480	18.900	0.170	12.500	7.870	0.000
11	19.280	15.000	0.225	12.500	7.870	0.000
12	22.000	18.900	0.117	12.500	7.870	0.000
13	24.340	21.700	0.094	12.500	7.870	0.000

Load Model

Load a model from an input file.

Signals

Inputs:

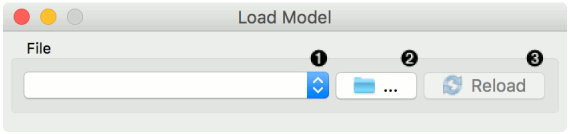
- None

Outputs:

- **Model**

  A model with selected parameters.

Description



1. Choose from a list of previously used models.
2. Browse for saved models.
3. Reload the selected model.

Example

When you want to use a custom-set model that you've saved before, open the **Load Model** widget and select the desired file with the *Browse* icon. This widget loads the exisiting model into *Predictions* widget. Data sets used with **Load Model** have to contain compatible attributes!

Constant

Predict the most frequent class or mean value from the training set.

Signals

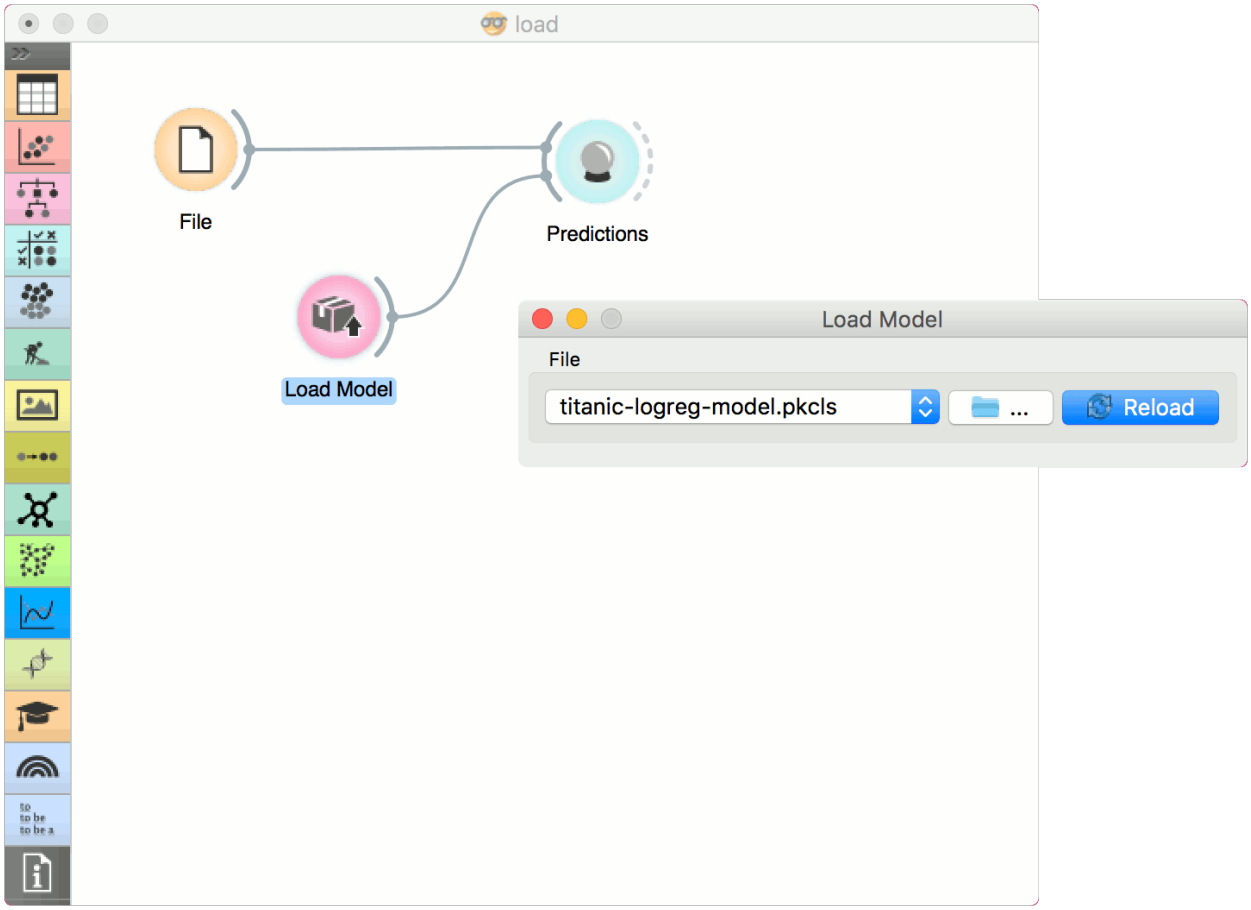
Inputs:

- **Data**

  A data set

- **Preprocessor**

  Preprocessing method(s)


load
File
Predictions
Load Model
Load Model
File
titanic-logreg-model.pkcls
...
Reload

Outputs:

- **Learner**

  A majority/mean learning algorithm

- **Model**

  A trained model. Output signal sent only if input *Data* is present.

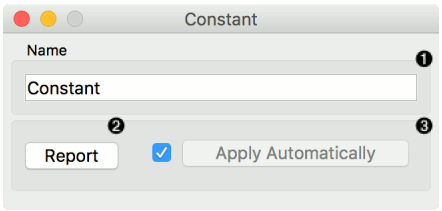
Description

This learner produces a model that always predicts the majority for classification tasks and mean value for regression tasks.

For classification, when predicting the class value with *Predictions*, the widget will return relative frequencies of the classes in the training set. When there are two or more majority classes, the classifier chooses the predicted class randomly, but always returns the same class for a particular example.

For regression, it *learns* the mean of the class variable and returns a predictor with the same mean value.

The widget is typically used as a baseline for other models.



This widget provides the user with two options:

1. The name under which it will appear in other widgets. Default name is “Constant”.
2. Produce a report.

If you change the widget’s name, you need to click *Apply*. Alternatively, tick the box on the left side and changes will be communicated automatically.

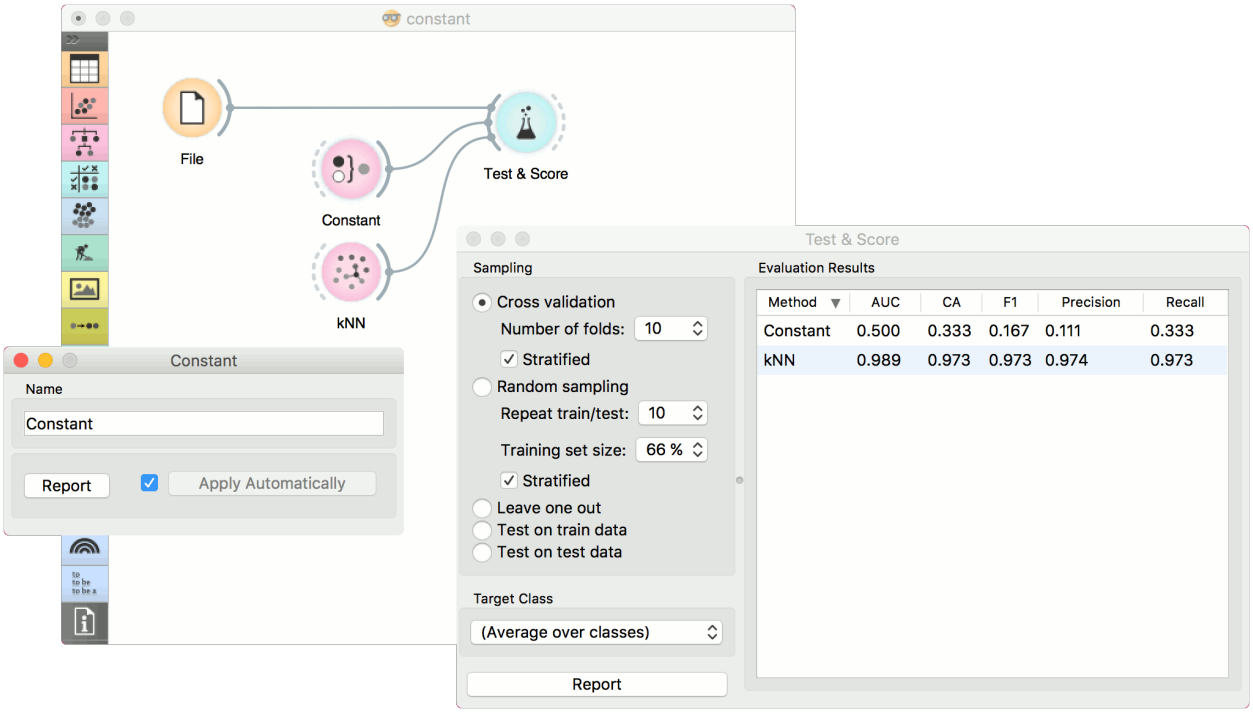
Examples

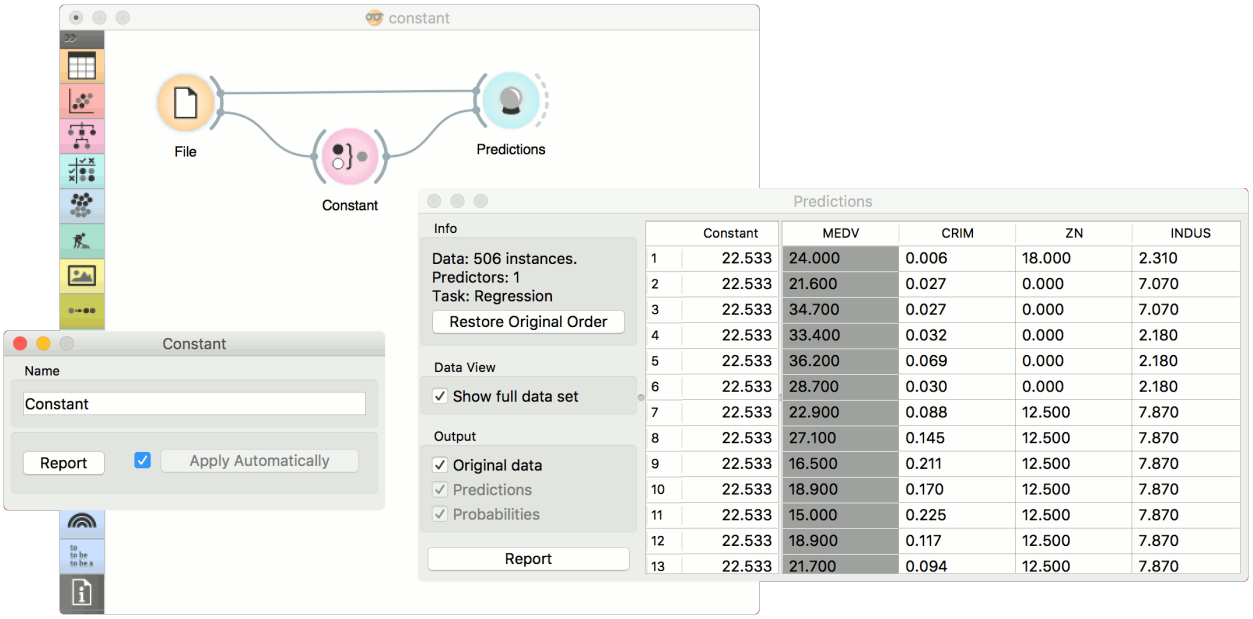
In a typical classification example, we would use this widget to compare the scores of other learning algorithms (such as kNN) with the default scores. Use *iris* data set and connect it to *Test & Score*. Then connect **Constant** and *kNN* to *Test & Score* and observe how well *kNN* performs against a constant baseline.

For regression, we use **Constant** to construct a predictor in *Predictions*. We used the *housing* data set. In **Predictions**, you can see that *Mean Learner* returns one (mean) value for all instances.

Random Forest

Predict using an ensemble of decision trees.


constant
File
Constant
kNN
Test & Score
Constant
Name
Constant
Report
Apply Automatically
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision Recall
Constant 0.500 0.333 0.167 0.111 0.333
kNN 0.989 0.973 0.973 0.974 0.973


constant
File
Constant
Predictions
Constant
Name
Constant
Report
Apply Automatically
Predictions
Info
Data: 506 instances.
Predictors: 1
Task: Regression
Restore Original Order
Data View
Show full data set
Output
Original data
Predictions
Probabilities
Report
Constant MEDV CRIM ZN INDUS
1 22.533 24.000 0.006 18.000 2.310
2 22.533 21.600 0.027 0.000 7.070
3 22.533 34.700 0.027 0.000 7.070
4 22.533 33.400 0.032 0.000 2.180
5 22.533 36.200 0.069 0.000 2.180
6 22.533 28.700 0.030 0.000 2.180
7 22.533 22.900 0.088 12.500 7.870
8 22.533 27.100 0.145 12.500 7.870
9 22.533 16.500 0.211 12.500 7.870
10 22.533 18.900 0.170 12.500 7.870
11 22.533 15.000 0.225 12.500 7.870
12 22.533 18.900 0.117 12.500 7.870
13 22.533 21.700 0.094 12.500 7.870

Signals

Inputs:

- **Data**

  A data set

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A random forest learning algorithm with settings as specified in the dialog.

- **Model**

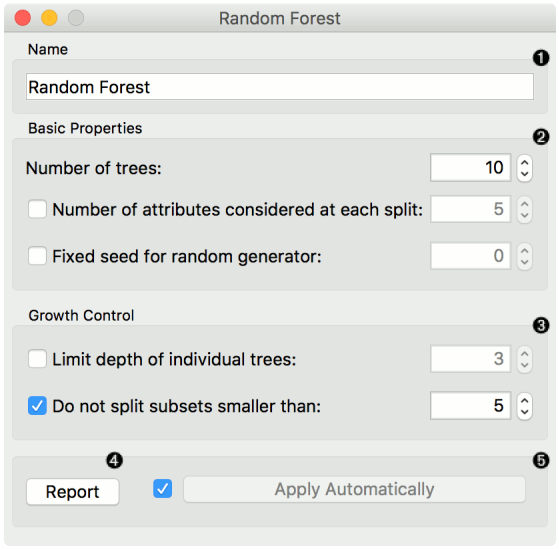
  A trained model. Output signal sent only if input *Data* is present.

Description

Random forest is an ensemble learning method used for classification, regression and other tasks. It was first proposed by Tin Kam Ho and further developed by Leo Breiman (Breiman, 2001) and Adele Cutler.

Random Forest builds a set of decision trees. Each tree is developed from a bootstrap sample from the training data. When developing individual trees, an arbitrary subset of attributes is drawn (hence the term "Random"), from which the best attribute for the split is selected. The final model is based on the majority vote from individually developed trees in the forest.

Random Forest works for both classification and regression tasks.

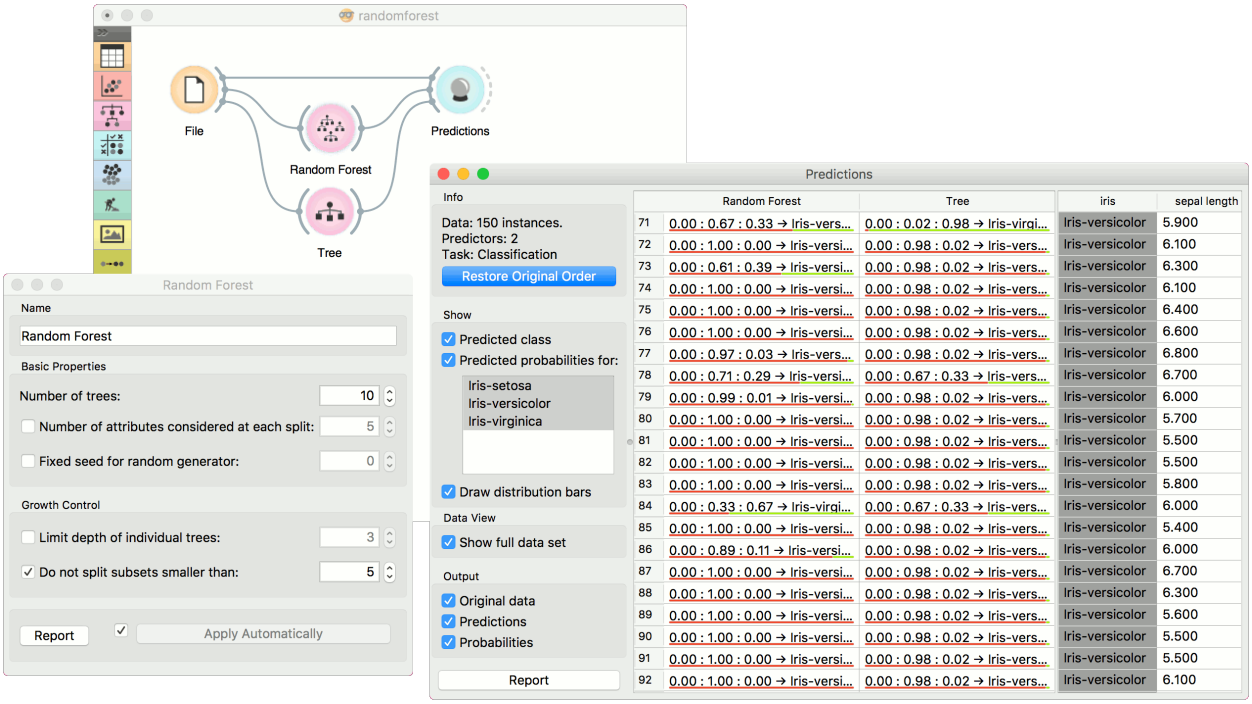


1. Specify the name of the model. The default name is "Random Forest".
2. Specify how many decision trees will be included in the forest (*Number of trees in the forest*), and how many attributes will be arbitrarily drawn for consideration at each node. If the latter is not specified (option *Number of attributes...* left unchecked), this number is equal to the square root of the number of attributes in the data. You can also choose to fix the seed for tree generation (*Fixed seed for random generator*), which enables replicability of the results.

3. Original Brieman's proposal is to grow the trees without any pre-prunning, but since pre-pruning often works quite well and is faster, the user can set the depth to which the trees will be grown (*Limit depth of individual trees*). Another pre-pruning option is to select the smallest subset that can be split (*Do not split subsets smaller than*).
4. Produce a report.
5. Click *Apply* to communicate the changes to other widgets. Alternatively, tick the box on the left side of the *Apply* button and changes will be communicated automatically.

Examples

For classification tasks, we use *iris* data set. Connect it to *Predictions*. Then, connect *File* to **Random Forest** and *Tree* and connect them further to *Predictions*. Finally, observe the predictions for the two models.



For regressions tasks, we will use *housing* data. Here, we will compare different models, namely **Random Forest**, *Linear Regression* and *Constant*, in the *Test&Score* widget.

References

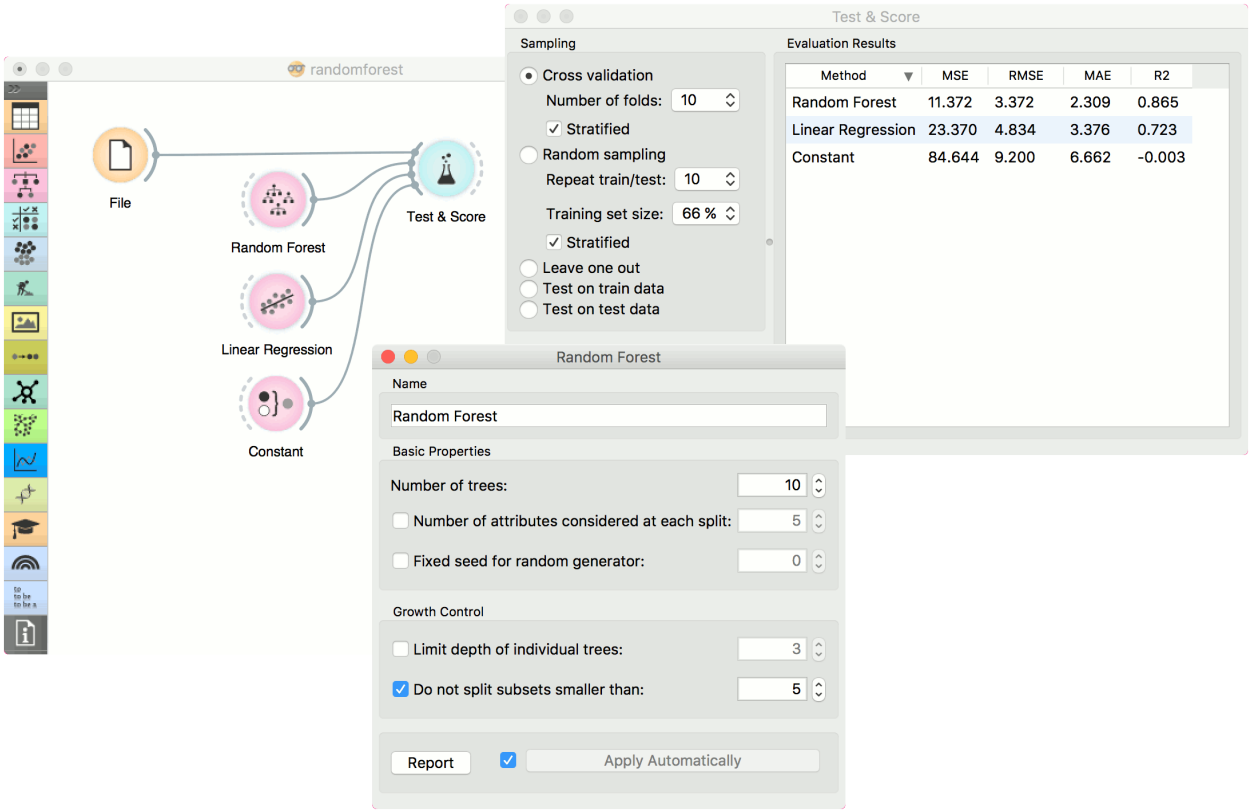
Breiman, L. (2001). Random Forests. In Machine Learning, 45(1), 5-32. Available here

Save Model

Save a trained model to an output file.

Signals

Inputs:


randomforest
File
Random Forest
Linear Regression
Constant
Test & Score
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Evaluation Results
Method MSE RMSE MAE R2
Random Forest 11.372 3.372 2.309 0.865
Linear Regression 23.370 4.834 3.376 0.723
Constant 84.644 9.200 6.662 -0.003
Random Forest
Name
Random Forest
Basic Properties
Number of trees: 10
Number of attributes considered at each split: 5
Fixed seed for random generator: 0
Growth Control
Limit depth of individual trees: 3
Do not split subsets smaller than: 5
Report
Apply Automatically

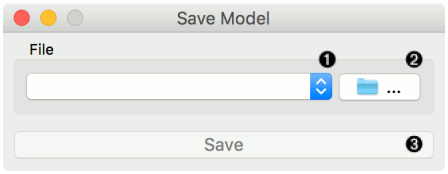
- **Model**

  A model with selected parameters

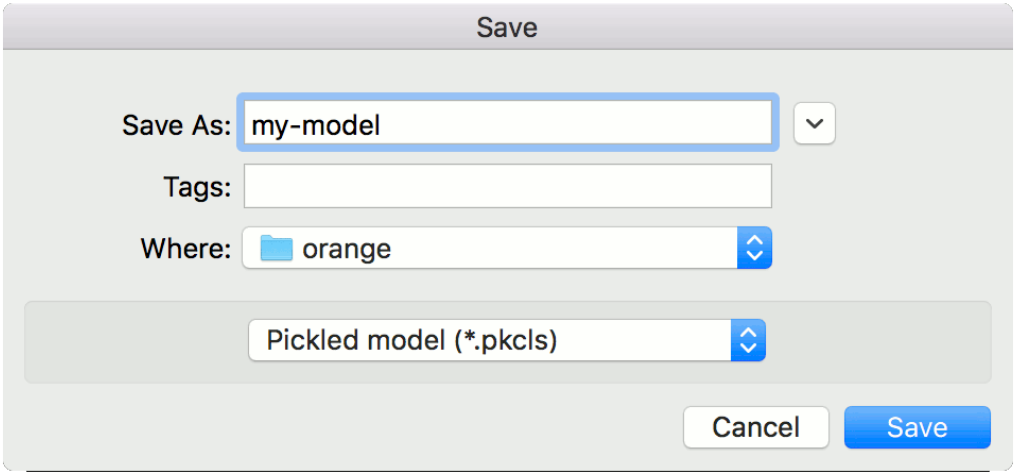
Outputs:

- None

Description



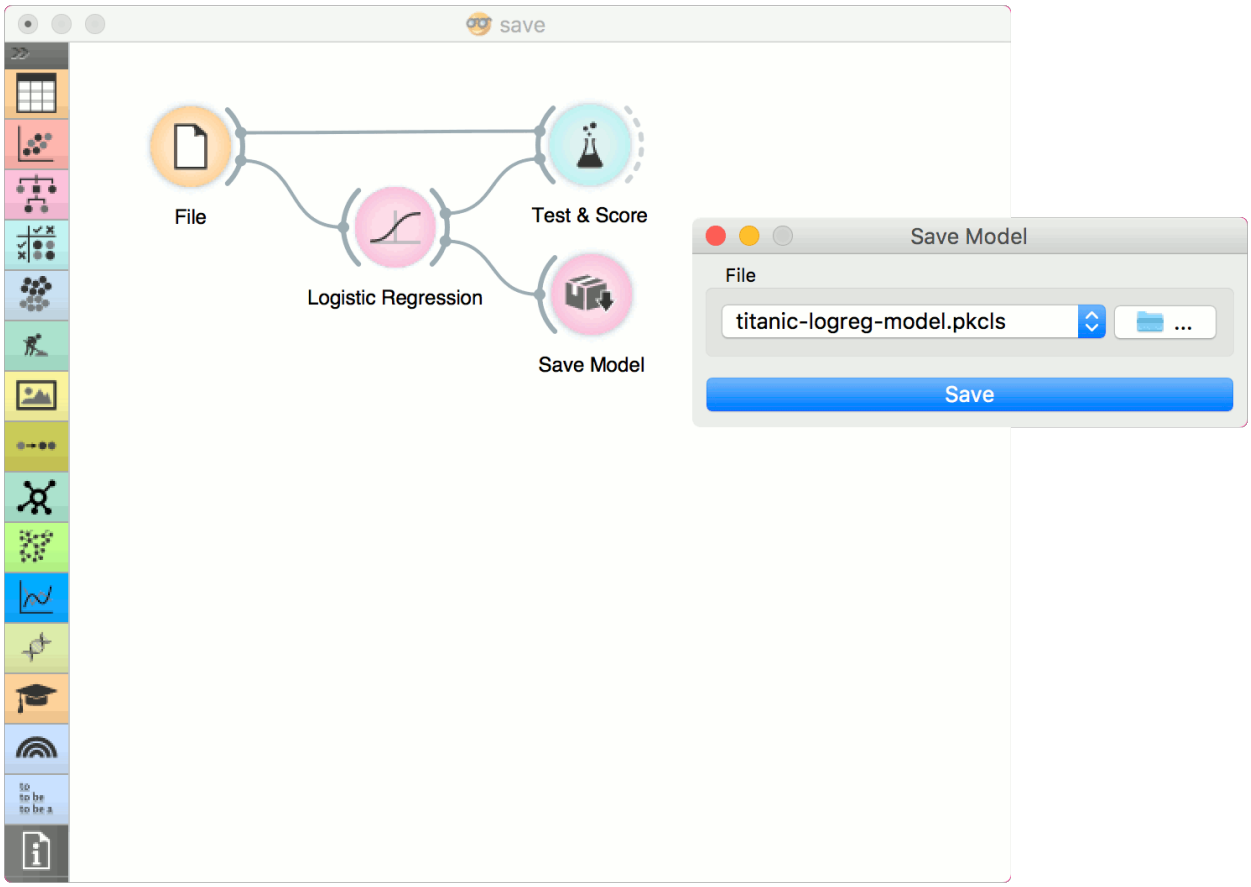
1. Choose from previously saved models.
2. Save the created model with the *Browse* icon. Click on the icon and enter the name of the file. The model will be saved to a pickled file.



3. Save the model.

Example

When you want to save a custom-set model, feed the data to the model (e.g. *Logistic Regression*) and connect it to **Save Model**. Name the model; load it later into workflows with *Load Model*. Data sets used with **Load Model** have to contain compatible attributes.


save
File
Test & Score
Logistic Regression
Save Model
Save Model
File
titanic-logreg-model.pkcls
...
Save

SVM

Support Vector Machines map inputs to higher-dimensional feature spaces.

Signals

Inputs:

- **Data**

  A data set.

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A support vector machine learning algorithm with settings as specified in the dialog.

- **Model**

  A trained model. Output signal sent only if input *Data* is present.

- **Support Vectors**

  A subset of data instances from the training set that were used as support vectors in the trained model.

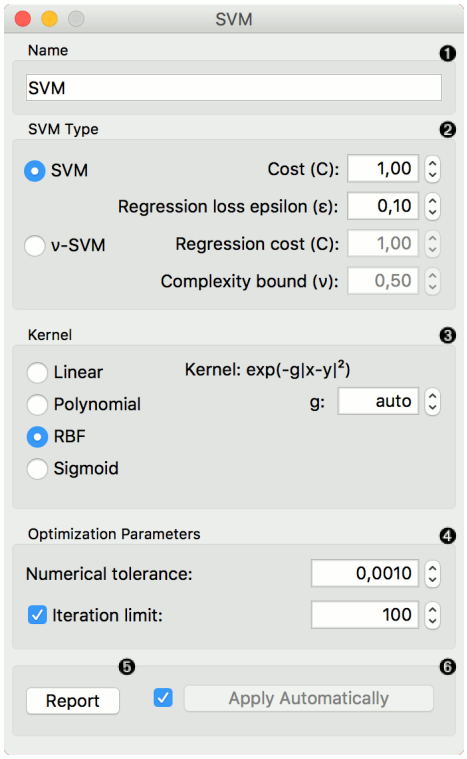
Description

Support vector machine (SVM) is a machine learning technique that separates the attribute space with a hyperplane, thus maximizing the margin between the instances of different classes or class values. The technique often yields supreme predictive performance results. Orange embeds a popular implementation of SVM from the LIBSVM package. This widget is its graphical user interface.

For regression tasks, **SVM** performs linear regression in a high dimension feature space using an ϵ-insensitive loss. Its estimation accuracy depends on a good setting of C, ϵ and kernel parameters. The widget outputs class predictions based on a SVM Regression.

The widget works for both classification and regression tasks.

1. The learner can be given a name under which it will appear in other widgets. The default name is "SVM".
2. SVM type with test error settings. *SVM* and ν-*SVM* are based on different minimization of the error function. On the right side, you can set test error bounds:
   - SVM:
     - Cost: penalty term for loss and applies for classification and regression tasks.
     - ϵ: a parameter to the epsilon-SVR model, applies to regression tasks. Defines the distance from true values within which no penalty is associated with predicted values.
   - ν-SVM:
     - Cost: penalty term for loss and applies only to regression tasks
     - ν: a parameter to the ν-SVR model, applies to classification and regression tasks. An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.



3. Kernel is a function that transforms attribute space to a new feature space to fit the maximum-margin hyperplane, thus allowing the algorithm to create the model with:

   - Linear
   - Polynomial
   - RBF and
   - Sigmoid

   kernels. Functions that specify the kernel are presented upon selecting them, and the constants involved are:

   - **g** for the gamma constant in kernel function (the recommended value is 1/k, where k is the number of the attributes, but since there may be no training set given to the widget the default is 0 and the user has to set this option manually),
   - **c** for the constant c0 in the kernel function (default 0), and
   - **d** for the degree of the kernel (default 3).

4. Set permitted deviation from the expected value in *Numerical Tolerance*. Tick the box next to *Iteration Limit* to set the maximum number of iterations permitted.
5. Produce a report.
6. Click *Apply* to commit changes. If you tick the box on the left side of the *Apply* button, changes will be communicated automatically.

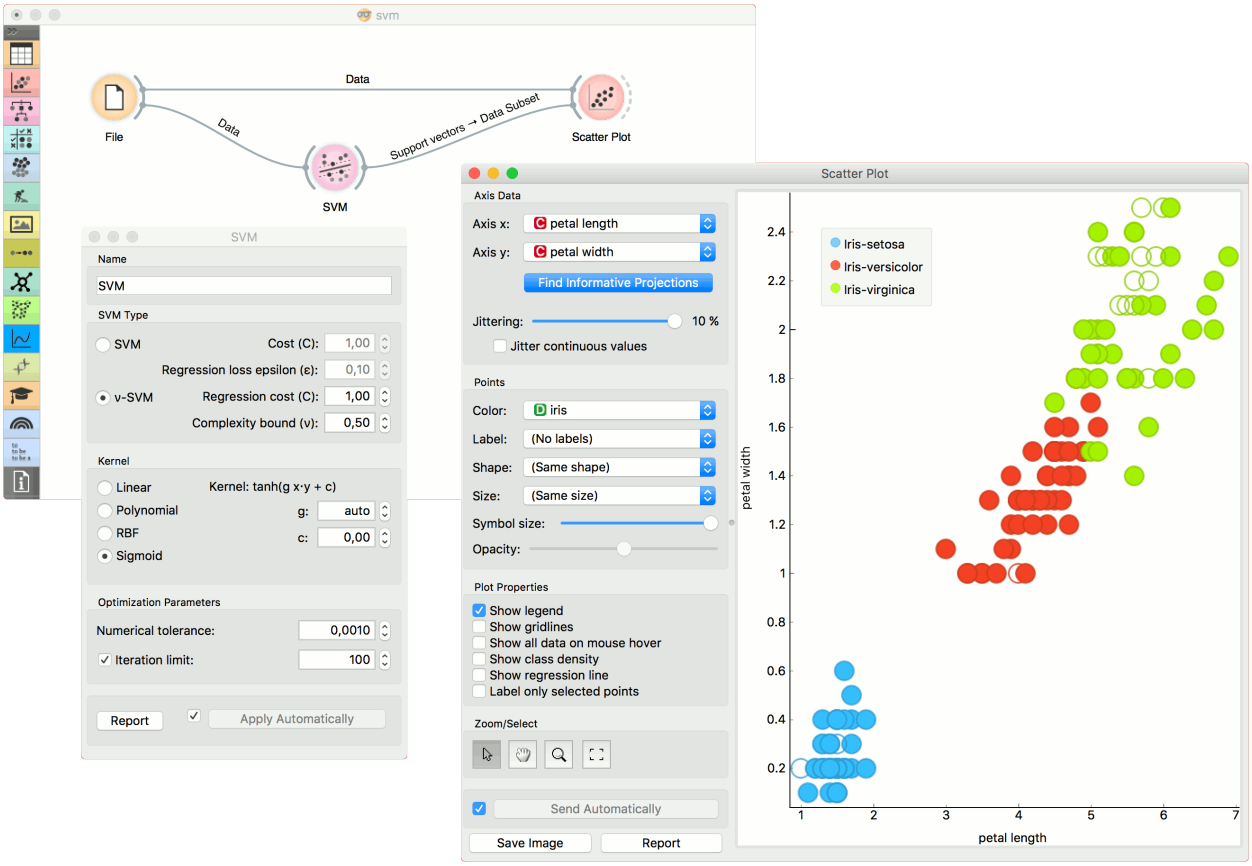
Examples

In the first (regression) example, we have used *housing* data set and split the data into two data subsets (*Data Sample* and *Remaining Data*) with *Data Sampler*. The sample was sent to SVM which produced a *Model*, which was then

used in *Predictions* to predict the values in *Remaining Data*. A similar schema can be used if the data is already in two separate files; in this case, two *File* widgets would be used instead of the *File* - *Data Sampler* combination.

widgets/model/images/SVM-predictions.png

The second example shows how to use **SVM** in combination with *Scatterplot*. The following workflow trains a SVM model on *iris* data and outputs support vectors, which are those data instances that were used as support vectors in the learning phase. We can observe which are these data instances in a scatter plot visualization. Note that for the workflow to work correctly, you must set the links between widgets as demonstrated in the screenshot below.

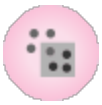


References

Introduction to SVM on StatSoft.

CN2 Rule Induction

Induce rules from data using CN2 algorithm.



Signals

Inputs

- **Data**

  Data set.

- **Preprocessor**

  Preprocessing method(s)

Outputs

- **Learner**

  The CN2 learning algorithm with settings as specified in the dialog.

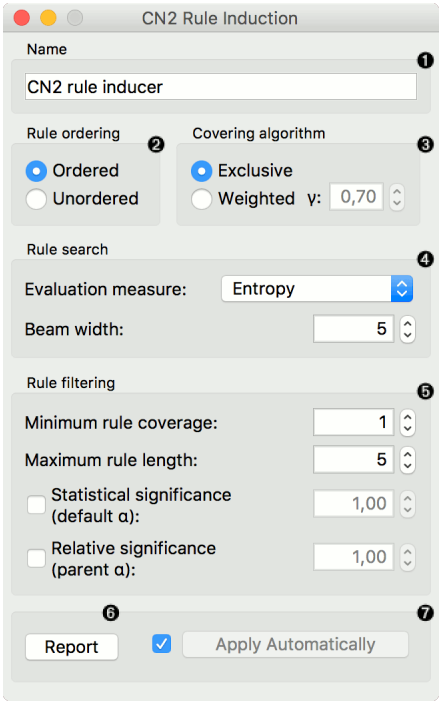
- **CN2 Rule Classifier**

  A trained model. Output signal sent only if input *Data* is present.

Description

The CN2 algorithm is a classification technique designed for the efficient induction of simple, comprehensible rules of form "if *cond* then predict *class*", even in domains where noise may be present.

CN2 Rule Induction works only for classification.



1. Name under which the learner appears in other widgets. The default name is *CN2 Rule Induction*.

2. *Rule ordering*:
   - **Ordered**: induce ordered rules (decision list). Rule conditions are found and the majority class is assigned in the rule head.
   - **Unordered**: induce unordered rules (rule set). Learn rules for each class individually, in regard to the original learning data.
3. *Covering algorithm*:
   - **Exclusive**: after covering a learning instance, remove it from further consideration.
   - **Weighted**: after covering a learning instance, decrease its weight (multiplication by *gamma*) and in-turn decrease its impact on further iterations of the algorithm.
4. *Rule search*:
   - **Evaluation measure**: select a heuristic to evaluate found hypotheses:
     (a) Entropy (measure of unpredictability of content)
     (b) Laplace Accuracy
     (c) Weighted Relative Accuracy
   - **Beam width**; remember the best rule found thus far and monitor a fixed number of alternatives (the beam).
5. *Rule filtering*:
   - **Minimum rule coverage**: found rules must cover at least the minimum required number of covered examples. Unordered rules must cover this many target class examples.
   - **Maximum rule length**: found rules may combine at most the maximum allowed number of selectors (conditions).
   - **Default alpha**: significance testing to prune out most specialised (less frequently applicable) rules in regard to the initial distribution of classes.
   - **Parent alpha**: significance testing to prune out most specialised (less frequently applicable) rules in regard to the parent class distribution.
6. Tick 'Apply Automatically' to auto-communicate changes to other widgets and to immediately train the classifier if learning data is connected. Alternatively, press 'Apply' after configuration.

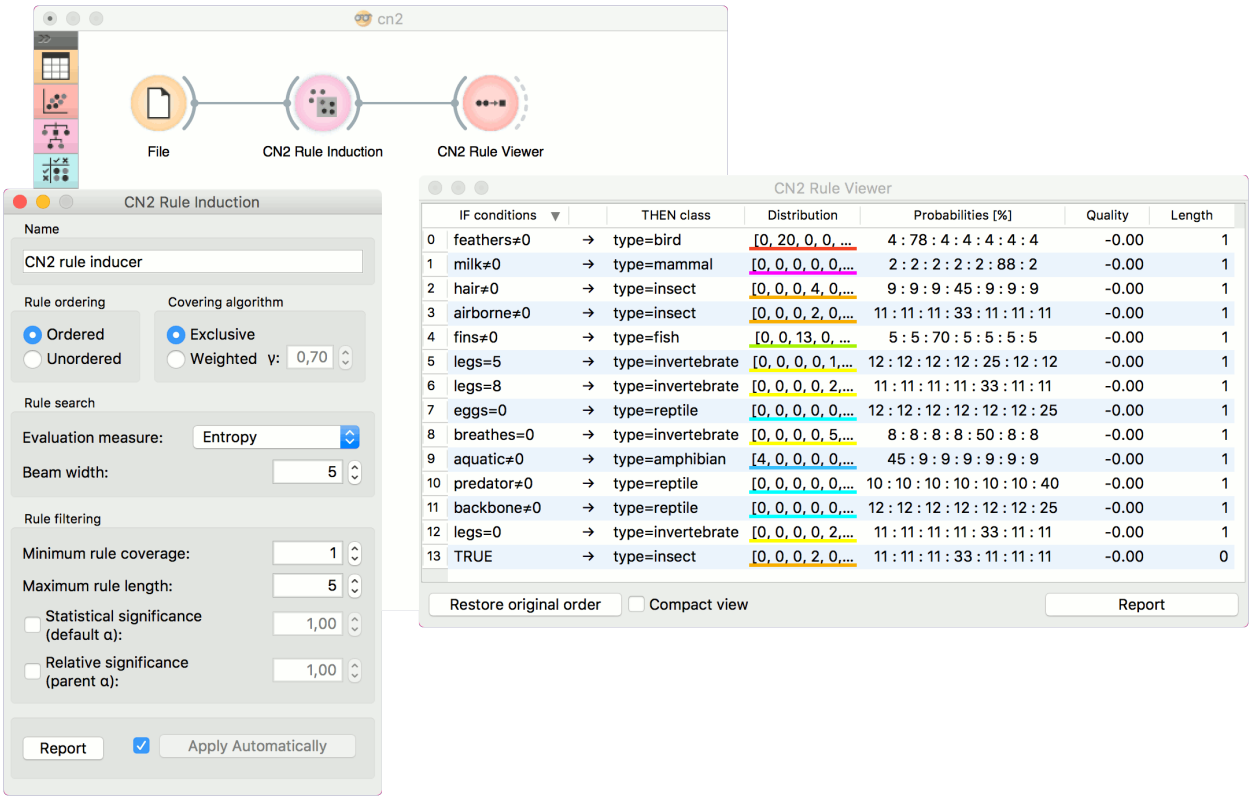
Examples

For the example below, we have used *zoo* data set and passed it to **CN2 Rule Induction**. We can review and interpret the built model with *CN2 Rule Viewer* widget.

The second workflow tests evaluates **CN2 Rule Induction** and *Tree* in *Test & Score*.

References

1. "Separate-and-Conquer Rule Learning", Johannes Fürnkranz, Artificial Intelligence Review 13, 3-54, 1999
2. "The CN2 Induction Algorithm", Peter Clark and Tim Niblett, Machine Learning Journal, 3 (4), pp261-283, (1989)
3. "Rule Induction with CN2: Some Recent Improvements", Peter Clark and Robin Boswell, Machine Learning - Proceedings of the 5th European Conference (EWSL-91), pp151-163, 1991
4. "Subgroup Discovery with CN2-SD", Nada Lavrač et al., Journal of Machine Learning Research 5 (2004), 153-188, 2004



AdaBoost

An ensemble meta-algorithm that combines weak learners and adapts to the 'hardness' of each training sample.

Signals

Inputs:

- **Data**

  A data set.

- **Preprocessor**

  Preprocessing method(s)

- **Learner**

  A learning algorithm.

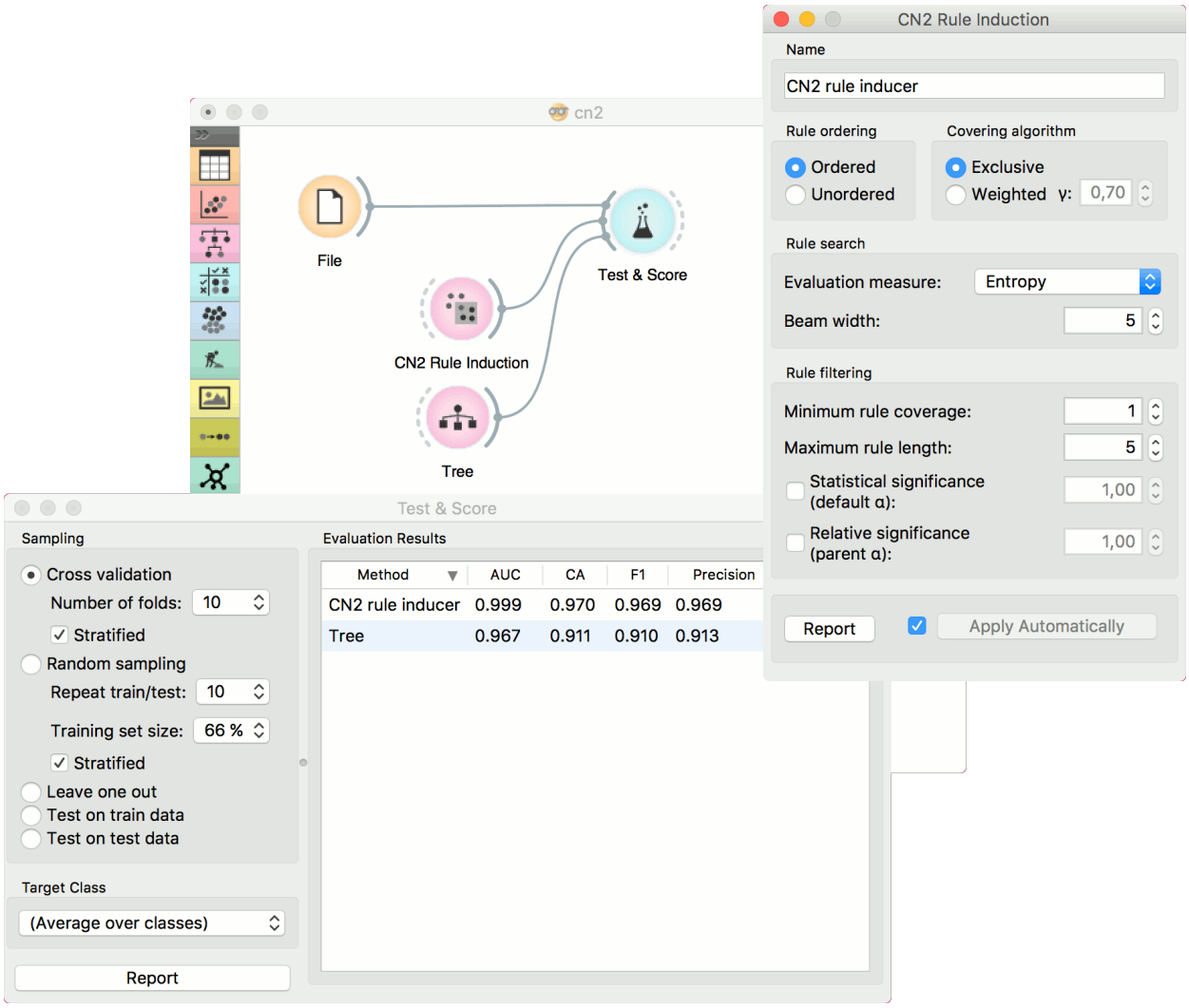
Outputs:

- **Learner**

  AdaBoost learning algorithm with settings as specified in the dialog.

- **Model**

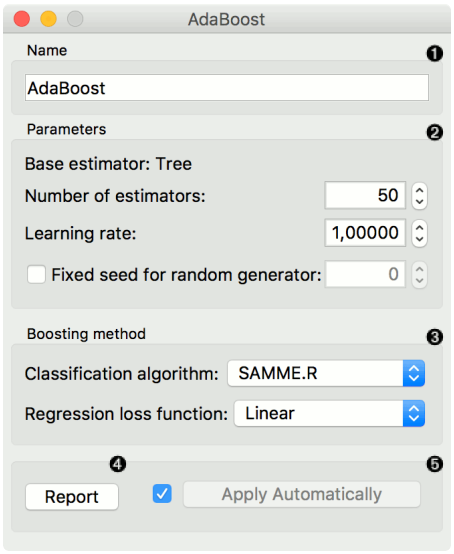
  A trained model. Output signal sent only if input *Data* is present.


cn2
File
Test & Score
CN2 Rule Induction
Tree
CN2 Rule Induction
Name
CN2 rule inducer
Rule ordering
Ordered
Unordered
Covering algorithm
Exclusive
Weighted γ: 0,70
Rule search
Evaluation measure: Entropy
Beam width: 5
Rule filtering
Minimum rule coverage: 1
Maximum rule length: 5
Statistical significance (default α): 1,00
Relative significance (parent α): 1,00
Report
Apply Automatically
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision
CN2 rule inducer 0.999 0.970 0.969 0.969
Tree 0.967 0.911 0.910 0.913

Description

The AdaBoost (short for "Adaptive boosting") widget is a machine-learning algorithm, formulated by Yoav Freund and Robert Schapire. It can be used with other learning algorithms to boost their performance. It does so by tweaking the weak learners.

AdaBoost works for both classification and regression.

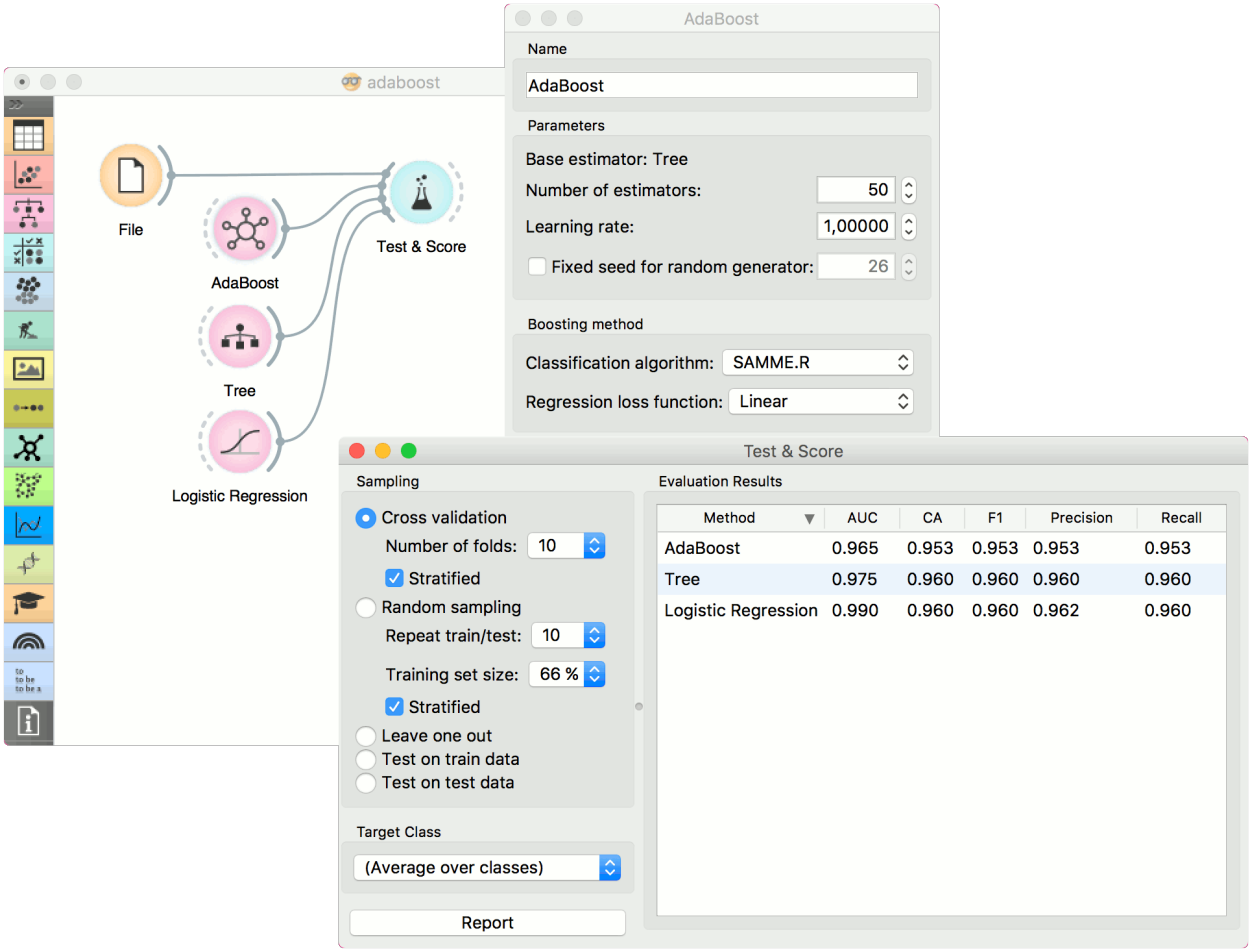


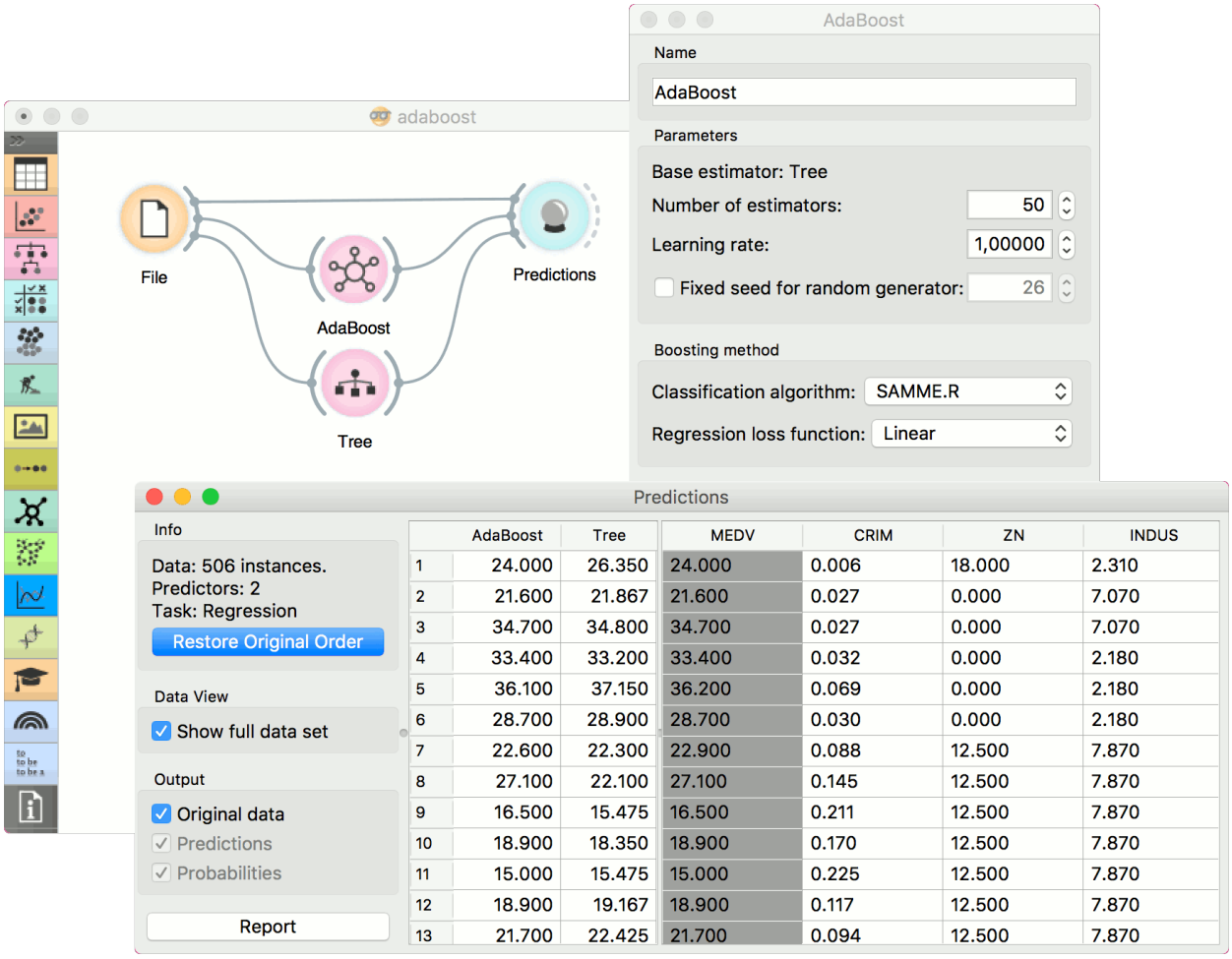
1. The learner can be given a name under which it will appear in other widgets. The default name is "AdaBoost".
2. Set the parameters. The base estimator is a tree and you can set:
   - *Number of estimators*
   - *Learning rate*: it determines to what extent the newly acquired information will override the old information (0 = the agent will not learn anything, 1 = the agent considers only the most recent information)
   - *Fixed seed for random generator*: set a fixed seed to enable reproducing the results.
3. Boosting method.
   - *Classification algorithm* (if classification on input): SAMME (updates base estimator's weights with classification results) or SAMME.R (updates base estimator's weight with probability estimates).
   - *Regression loss function* (if regression on input): Linear (), Square (), Exponential ().
4. Produce a report.
5. Click *Apply* after changing the settings. That will put the new learner in the output and, if the training examples are given, construct a new model and output it as well. To communicate changes automatically tick *Apply Automatically*.

Examples

For classification, we loaded the *iris* data set. We used *AdaBoost*, *Tree* and *Logistic Regression* and evaluated the models' performance in *Test & Score*.

For regression, we loaded the *housing* data set, sent the data instances to two different models (**AdaBoost** and *Tree*) and output them to the *Predictions* widget.


adaboost
File
AdaBoost
Tree
Logistic Regression
Test & Score
AdaBoost
Name
AdaBoost
Parameters
Base estimator: Tree
Number of estimators: 50
Learning rate: 1,00000
Fixed seed for random generator: 26
Boosting method
Classification algorithm: SAMME.R
Regression loss function: Linear
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision Recall
AdaBoost 0.965 0.953 0.953 0.953 0.953
Tree 0.975 0.960 0.960 0.960 0.960
Logistic Regression 0.990 0.960 0.960 0.962 0.960



	AdaBoost	Tree	MEDV	CRIM	ZN	INDUS
1	24.000	26.350	24.000	0.006	18.000	2.310
2	21.600	21.867	21.600	0.027	0.000	7.070
3	34.700	34.800	34.700	0.027	0.000	7.070
4	33.400	33.200	33.400	0.032	0.000	2.180
5	36.100	37.150	36.200	0.069	0.000	2.180
6	28.700	28.900	28.700	0.030	0.000	2.180
7	22.600	22.300	22.900	0.088	12.500	7.870
8	27.100	22.100	27.100	0.145	12.500	7.870
9	16.500	15.475	16.500	0.211	12.500	7.870
10	18.900	18.350	18.900	0.170	12.500	7.870
11	15.000	15.475	15.000	0.225	12.500	7.870
12	18.900	19.167	18.900	0.117	12.500	7.870
13	21.700	22.425	21.700	0.094	12.500	7.870

Stochastic Gradient Descent

Minimize an objective function using a stochastic approximation of gradient descent.

Signals

Inputs:

- **Data**

  A data set.

- **Preprocessor**

  Preprocessing method(s)

Outputs:

- **Learner**

  A SGD learning algorithm with settings as specified in the dialog.

- **Model**

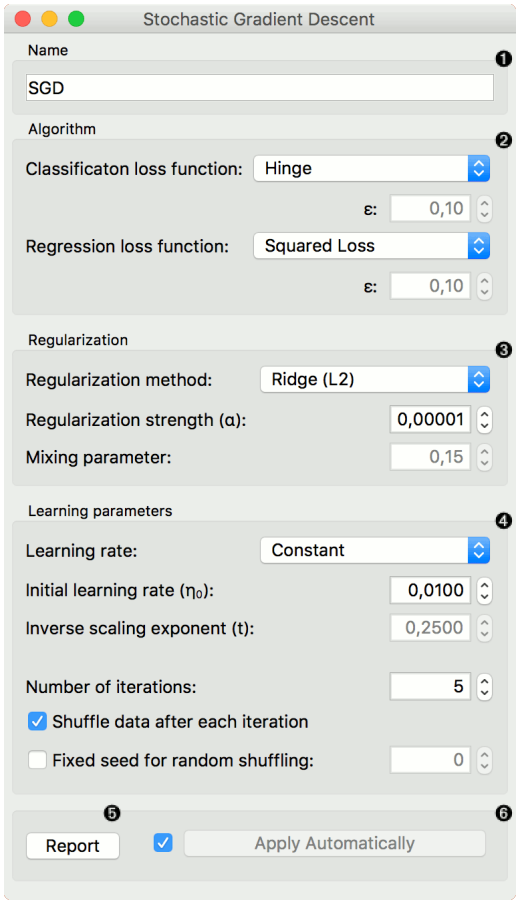
  A trained model. Output signal sent only if input *Data* is present.

Description

The **Stochastic Gradient Descent** widget uses stochastic gradient descent that minimizes a chosen loss function with a linear function. The algorithm approximates a true gradient by considering one sample at a time, and simultaneously updates the model based on the gradient of the loss function. For regression, it returns predictors as minimizers of the sum, i.e. M-estimators, and is especially useful for large-scale and sparse data sets.

1. Specify the name of the model. The default name is "SGD".
2. Algorithm parameters. Classification loss function:
   - Hinge (linear SVM)
   - Logistic Regression (logistic regression SGD)
   - Modified Huber (smooth loss that brings tolerance to outliers as well as probability estimates)
   - *Squared Hinge* (quadratically penalized hinge)
   - Perceptron (linear loss used by the perceptron algorithm)
   - Squared Loss (fitted to ordinary least-squares)
   - Huber (switches to linear loss beyond ϵ)
   - Epsilon insensitive (ignores errors within ϵ, linear beyond it)
   - *Squared epsilon insensitive* (loss is squared beyond ϵ-region).

   Regression loss function:
   - Squared Loss (fitted to ordinary least-squares)
   - Huber (switches to linear loss beyond ϵ)


Stochastic Gradient Descent
Name
SGD
Algorithm
Classificaton loss function:
Hinge
ε:
0,10
Regression loss function:
Squared Loss
ε:
0,10
Regularization
Regularization method:
Ridge (L2)
Regularization strength (α):
0,00001
Mixing parameter:
0,15
Learning parameters
Learning rate:
Constant
Initial learning rate (η0):
0,0100
Inverse scaling exponent (t):
0,2500
Number of iterations:
5
Shuffle data after each iteration
Fixed seed for random shuffling:
0
Report
Apply Automatically

- Epsilon insensitive (ignores errors within ϵ, linear beyond it)
- *Squared epsilon insensitive* (loss is squared beyond ϵ-region).

3. Regularization norms to prevent overfitting:
   - None.
   - Lasso (L1) (L1, leading to sparse solutions)
   - Ridge (L2) (L2, standard regularizer)
   - Elastic net (mixing both penalty norms).

   Regularization strength defines how much regularization will be applied (the less we regularize, the more we allow the model to fit the data) and the mixing parameter what the ratio between L1 and L2 loss will be (if set to 0 then the loss is L2, if set to 1 then it is L1).
4. Learning parameters.
   - Learning rate:
     - *Constant*: learning rate stays the same through all epochs (passes)
     - Optimal: a heuristic proposed by Leon Bottou
     - Inverse scaling: earning rate is inversely related to the number of iterations
   - Initial learning rate.
   - Inverse scaling exponent: learning rate decay.
   - Number of iterations: the number of passes through the training data.
   - If *Shuffle data after each iteration* is on, the order of data instances is mixed after each pass.
   - If *Fixed seed for random shuffling* is on, the algorithm will use a fixed random seed and enable replicating the results.
7. Produce a report.
8. Press *Apply* to commit changes. Alternatively, tick the box on the left side of the *Apply* button and changes will be communicated automatically.

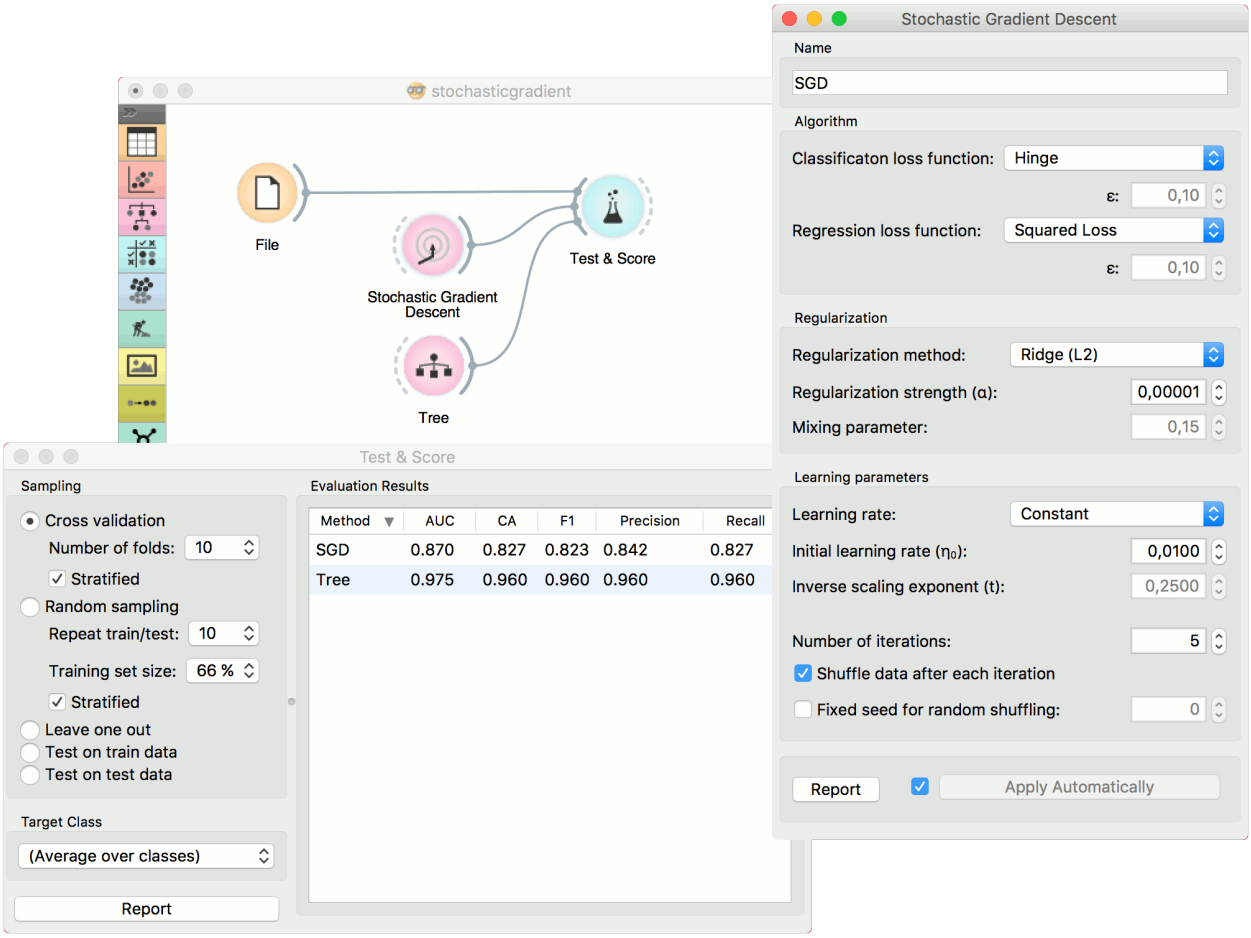
Examples

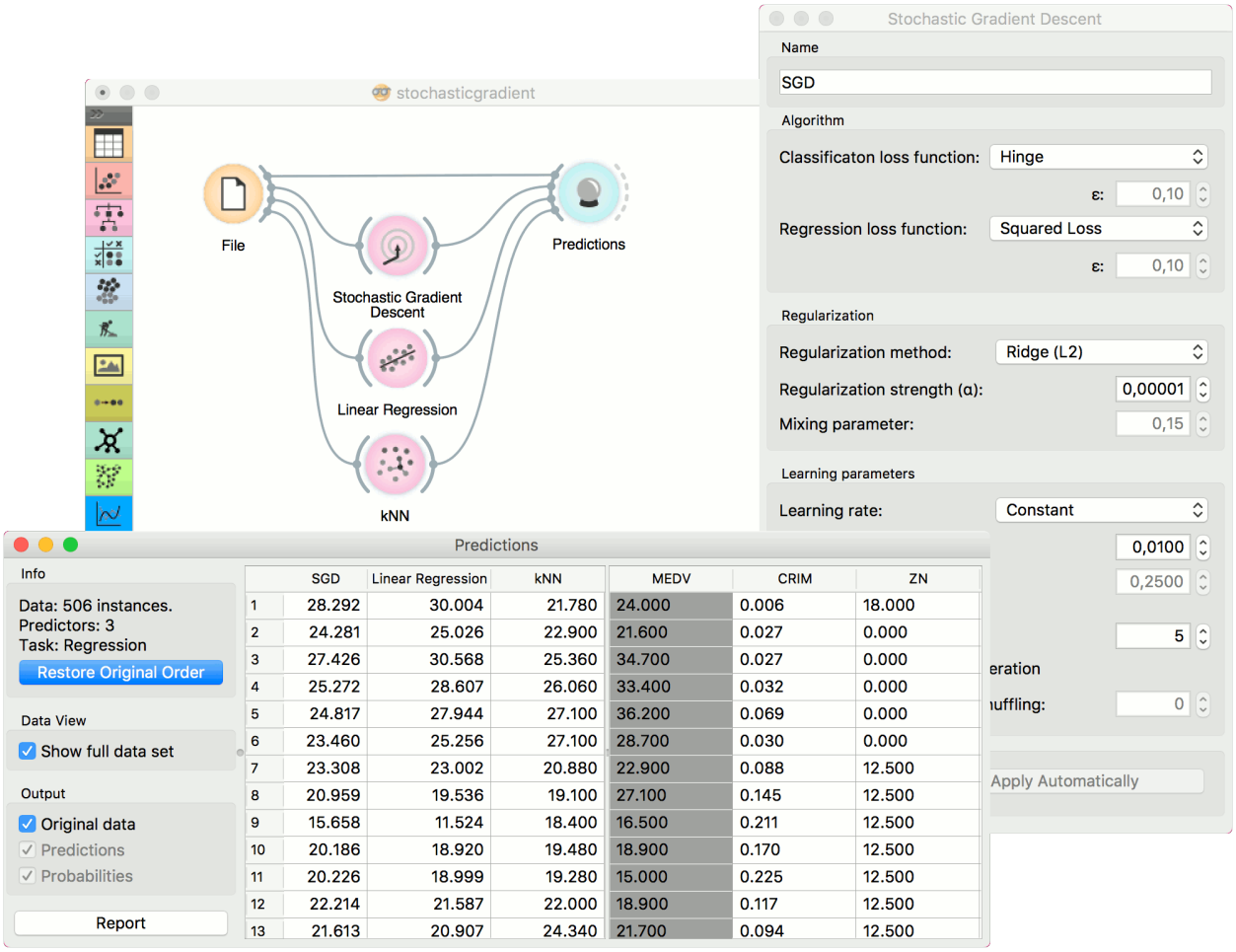
For the classification task, we will use *iris* data set and test two models on it. We connected *Stochastic Gradient Descent* and *Tree* to *Test & Score*. We also connected *File* to **Test & Score** and observed model performance in the widget.

For the regression task, we will compare three different models to see which predict what kind of results. For the purpose of this example, the *housing* data set is used. We connect the *File* widget to **Stochastic Gradient Descent**, *Linear Regression* and *kNN* widget and all four to the *Predictions* widget.

Linear Regression

A linear regression algorithm with optional L1 (LASSO), L2 (ridge) or L1L2 (elastic net) regularization.


stochasticgradient
File
Stochastic Gradient Descent
Test & Score
Tree
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision Recall
SGD 0.870 0.827 0.823 0.842 0.827
Tree 0.975 0.960 0.960 0.960 0.960
Stochastic Gradient Descent
Name
SGD
Algorithm
Classificaton loss function: Hinge
ε: 0,10
Regression loss function: Squared Loss
ε: 0,10
Regularization
Regularization method: Ridge (L2)
Regularization strength (α): 0,00001
Mixing parameter: 0,15
Learning parameters
Learning rate: Constant
Initial learning rate (η0): 0,0100
Inverse scaling exponent (t): 0,2500
Number of iterations: 5
Shuffle data after each iteration
Fixed seed for random shuffling: 0
Report
Apply Automatically


stochasticgradient
File
Predictions
Stochastic Gradient Descent
Linear Regression
kNN
Stochastic Gradient Descent
Name
SGD
Algorithm
Classificaton loss function:
Hinge
ε:
0,10
Regression loss function:
Squared Loss
ε:
0,10
Regularization
Regularization method:
Ridge (L2)
Regularization strength (α):
0,00001
Mixing parameter:
0,15
Learning parameters
Learning rate:
Constant
0,0100
0,2500
5
eration
uffling:
0
Apply Automatically
Predictions
Info
Data: 506 instances.
Predictors: 3
Task: Regression
Restore Original Order
Data View
Show full data set
Output
Original data
Predictions
Probabilities
Report
SGD
Linear Regression
kNN
MEDV
CRIM
ZN
1 28.292 30.004 21.780 24.000 0.006 18.000
2 24.281 25.026 22.900 21.600 0.027 0.000
3 27.426 30.568 25.360 34.700 0.027 0.000
4 25.272 28.607 26.060 33.400 0.032 0.000
5 24.817 27.944 27.100 36.200 0.069 0.000
6 23.460 25.256 27.100 28.700 0.030 0.000
7 23.308 23.002 20.880 22.900 0.088 12.500
8 20.959 19.536 19.100 27.100 0.145 12.500
9 15.658 11.524 18.400 16.500 0.211 12.500
10 20.186 18.920 19.480 18.900 0.170 12.500
11 20.226 18.999 19.280 15.000 0.225 12.500
12 22.214 21.587 22.000 18.900 0.117 12.500
13 21.613 20.907 24.340 21.700 0.094 12.500

Signals

Inputs:

- **Data**

  A data set

- **Preprocessor**

  A preprocessed data set.

Outputs:

- **Learner**

  A linear regression learning algorithm with settings as specified in the dialog.

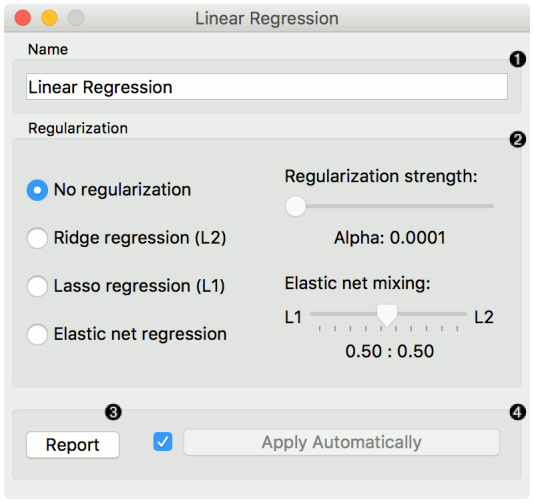
- **Predictor**

  A trained regressor. Output signal sent only if input *Data* is present.

Description

The **Linear Regression** widget constructs a learner/predictor that learns a linear function from its input data. The model can identify the relationship between a predictor xi and the response variable y. Additionally, Lasso and Ridge regularization parameters can be specified. Lasso regression minimizes a penalized version of the least squares loss function with L1-norm penalty and Ridge regularization with L2-norm penalty.

Linear regreesion works only on regression tasks.

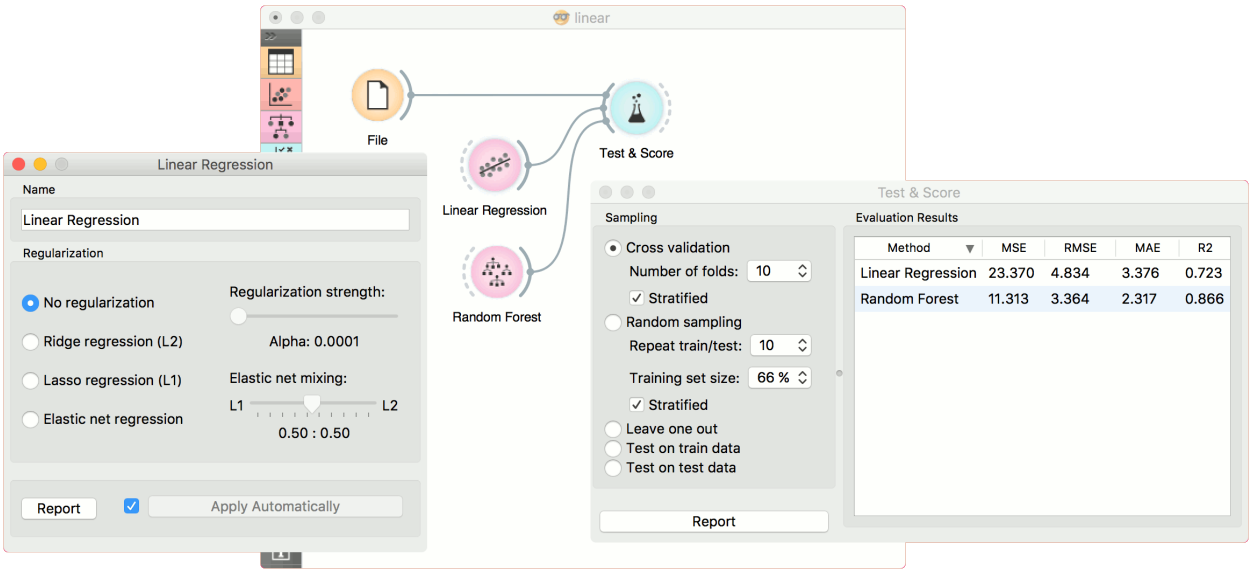


1. The learner/predictor name
2. Choose a model to train:
   - no regularization
   - a Ridge regularization (L2-norm penalty)
   - a Lasso bound (L1-norm penalty)
   - an Elastic net regularization
3. Produce a report.

4. Press *Apply* to commit changes. If *Apply Automatically* is ticked, changes are committed automatically.

Example

Below, is a simple workflow with *housing* data set. We trained **Linear Regression** and *Random Forest* and evaluated their performance in *Test&Score*.



Unsupervised

PCA

PCA linear transformation of input data.

Signals

Inputs:

- **Data**

  A data set.

Outputs:

- **Transformed Data**

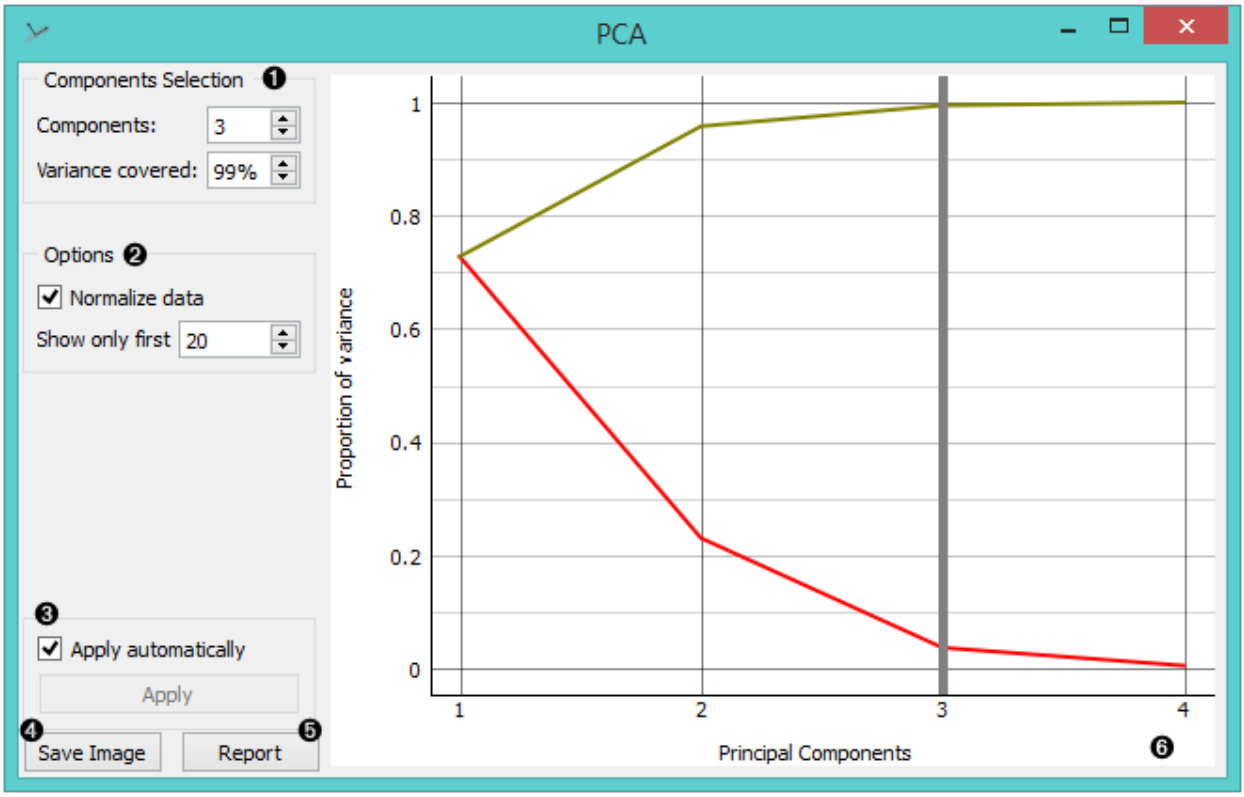
  PCA transformed input data.

- **Components**

  Eigenvectors.

Description

Principal Component Analysis (PCA) computes the PCA linear transformation of the input data. It outputs either a transformed data set with weights of individual instances or weights of principal components.

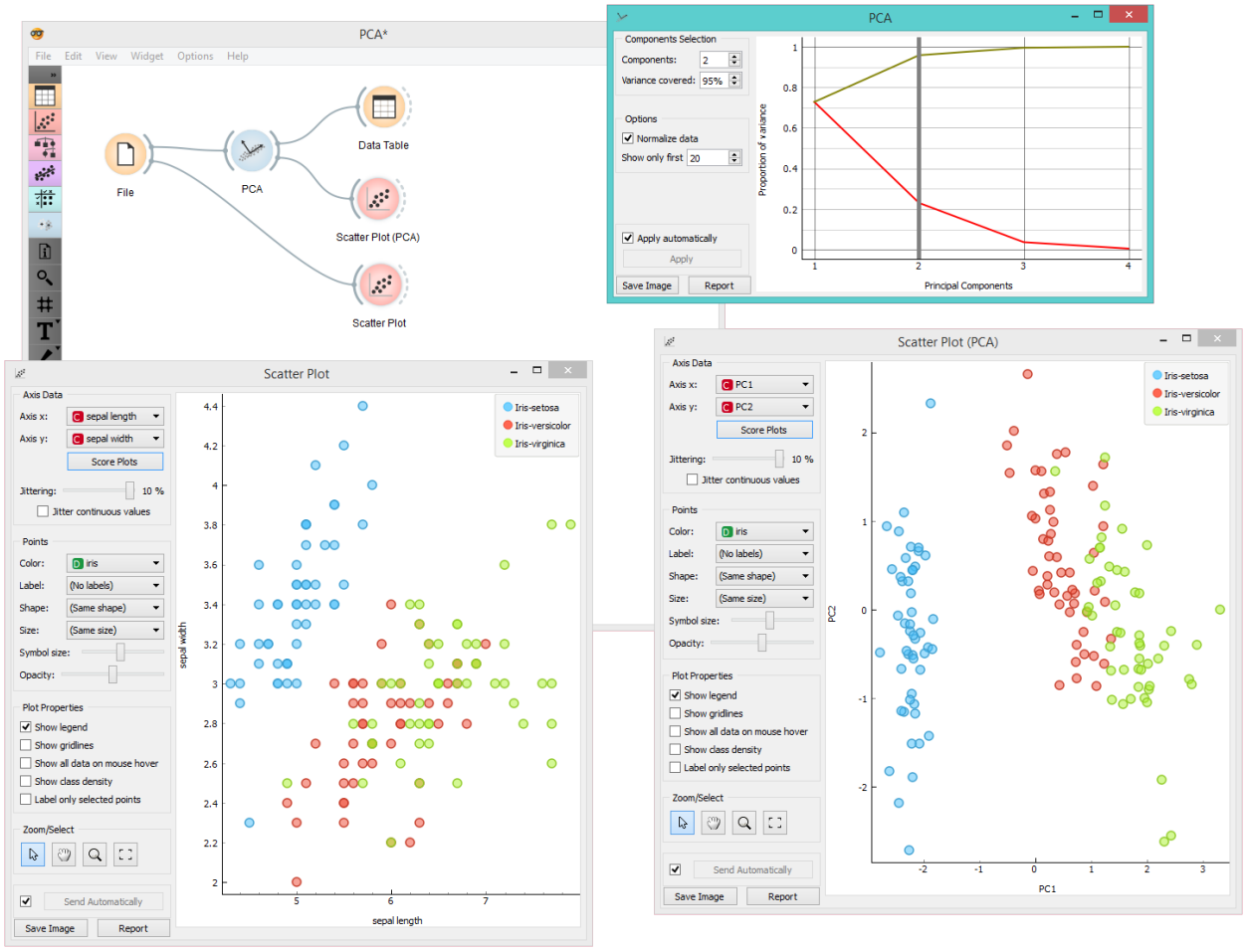


1. Select how many principal components you wish in your output. It is best to choose as few as possible with variance covered as high as possible. You can also set how much variance you wish to cover with your principal components.
2. You can normalize data to adjust the values to common scale.
3. When *Apply Automatically* is ticked, the widget will automatically communicate all changes. Alternatively, click *Apply*.
4. Press *Save Image* if you want to save the created image to your computer.
5. Produce a report.
6. Principal components graph, where the red (lower) line is the variance covered per component and the green (upper) line is cumulative variance covered by components.

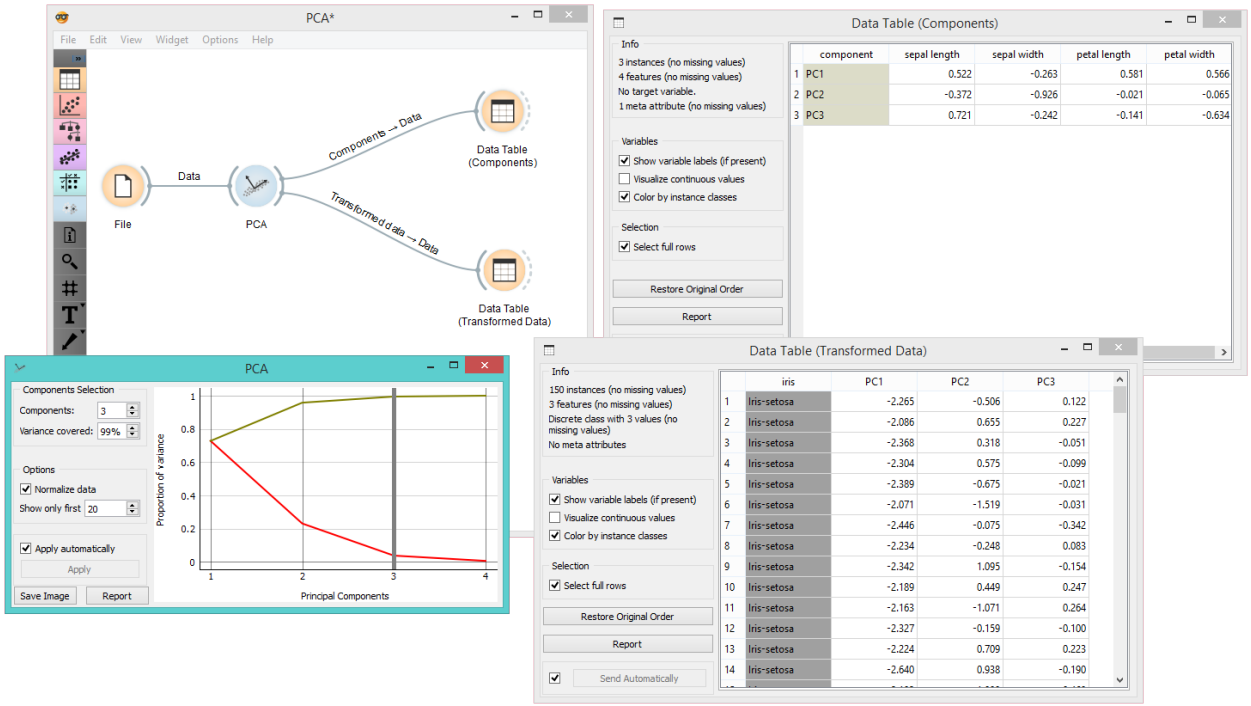
The number of components of the transformation can be selected either in the *Components Selection* input box or by dragging the vertical cutoff line in the graph.

Examples

PCA can be used to simplify visualizations of large data sets. Below, we used the *Iris* data set to show how we can improve the visualization of the data set with PCA. The transformed data in the *Scatter Plot* show a much clearer distinction between classes than the default settings.


PCA*
File Edit View Widget Options Help
File
PCA
Data Table
Scatter Plot (PCA)
Scatter Plot
PCA
Components Selection
Components: 2
Variance covered: 95%
Options
Normalize data
Show only first 20
Apply automatically
Apply
Save Image
Report
Proportion of variance
Principal Components
Scatter Plot
Axis Data
Axis x: sepal length
Axis y: sepal width
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
sepal width
sepal length
Scatter Plot (PCA)
Axis x: PC1
Axis y: PC2
PC2
PC1

The widget provides two outputs: transformed data and principal components. Transformed data are weights for individual instances in the new coordinate system, while components are the system descriptors (weights for princial components). When fed into the *Data Table*, we can see both outputs in numerical form. We used two data tables in order to provide a more clean visualization of the workflow, but you can also choose to edit the links in such a way that you display the data in just one data table. You only need to create two links and connect the *Transformed data* and *Components* inputs to the *Data* output.



Correspondence Analysis

Signals

Inputs:

- **Data**

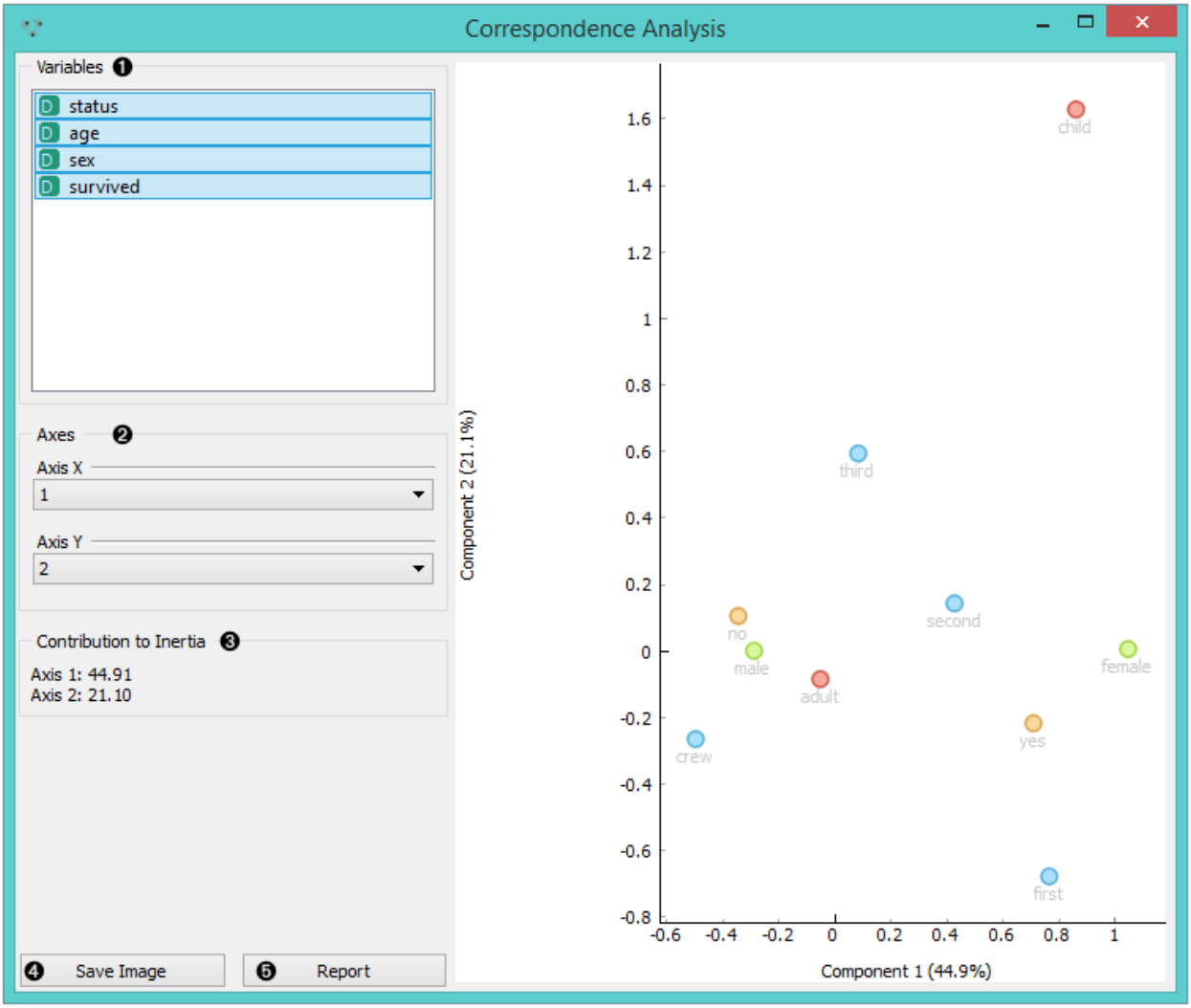
  A data set.

Outputs:

- None

Description

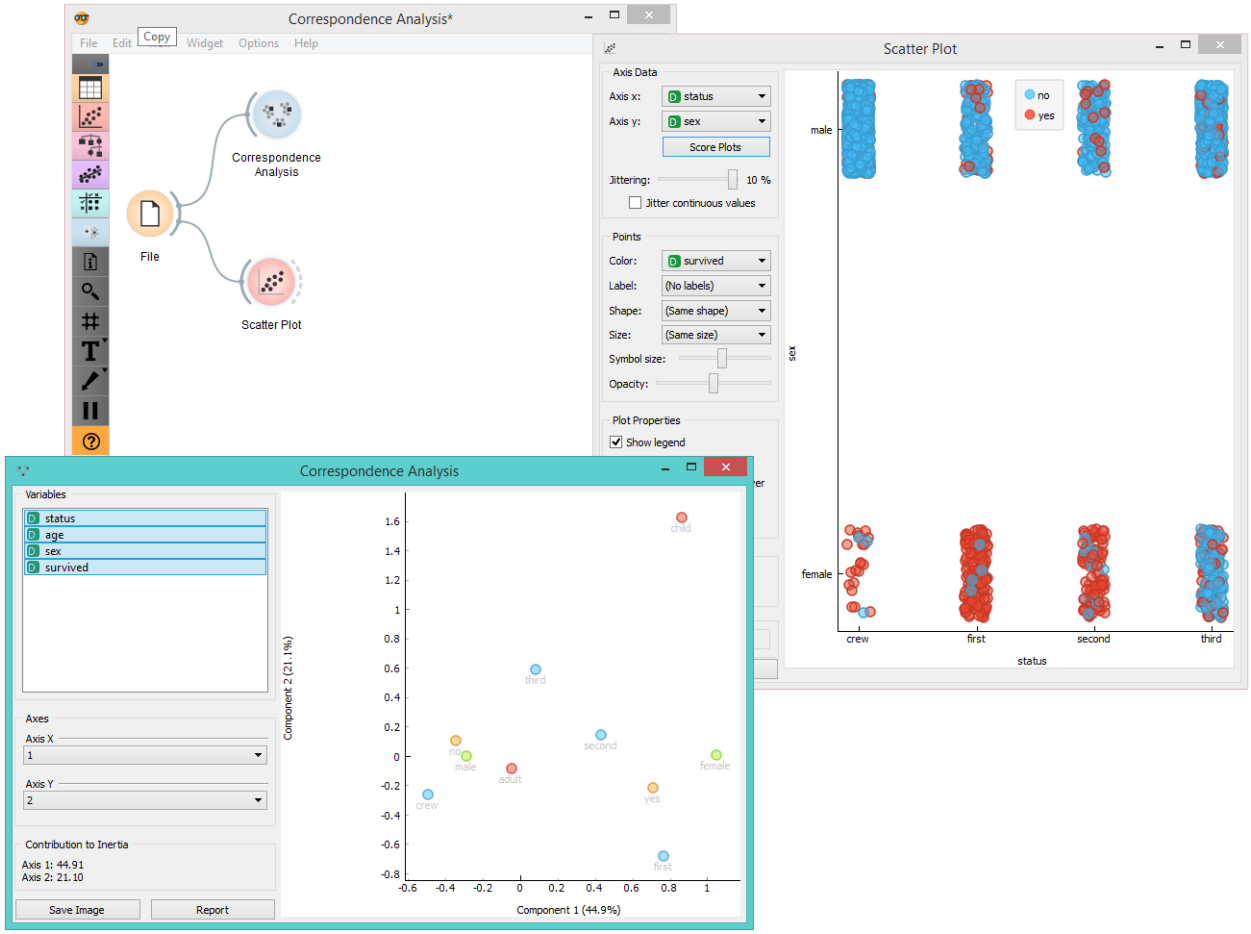
Correspondence Analysis (CA) computes the CA linear transformation of the input data. While it is similar to PCA, CA computes linear transformation on discrete rather than on continuous data.


Correspondence Analysis
Variables ❶
status
age
sex
survived
Axes ❷
Axis X
1
Axis Y
2
Contribution to Inertia ❸
Axis 1: 44.91
Axis 2: 21.10
❹ Save Image
❺ Report
Component 2 (21.1%)
Component 1 (44.9%)
child
third
second
no
male
female
adult
yes
crew
first

1. Select the variables you want to see plotted.
2. Select the component for each axis.
3. Inertia values (percentage of independence from transformation, i.e. variables are in the same dimension).
4. Produce a report.

Example

Below, is a simple comparison between the **Correspondence Analysis** and *Scatter plot* widgets on the *Titanic* data set. While the *Scatter plot* shows fairly well which class and sex had a good survival rate and which one didn't, **Correspondence Analysis** can plot several variables in a 2-D graph, thus making it easy to see the relations between variable values. It is clear from the graph that "no", "male" and "crew" are related to each other. The same goes for "yes", "female" and "first".



Distance Map

Visualizes distances between items.

Signals

Inputs:

- **Distances**

  A distance matrix.

Outputs:

- **Data**

  Instances corresponding to the selected elements of the matrix.

- **Features**

  Attributes corresponding to the selected elements of the matrix.

Description

The **Distance Map** visualizes distances between objects. The visualization is the same as if we printed out a table of numbers, except that the numbers are replaced by colored spots.

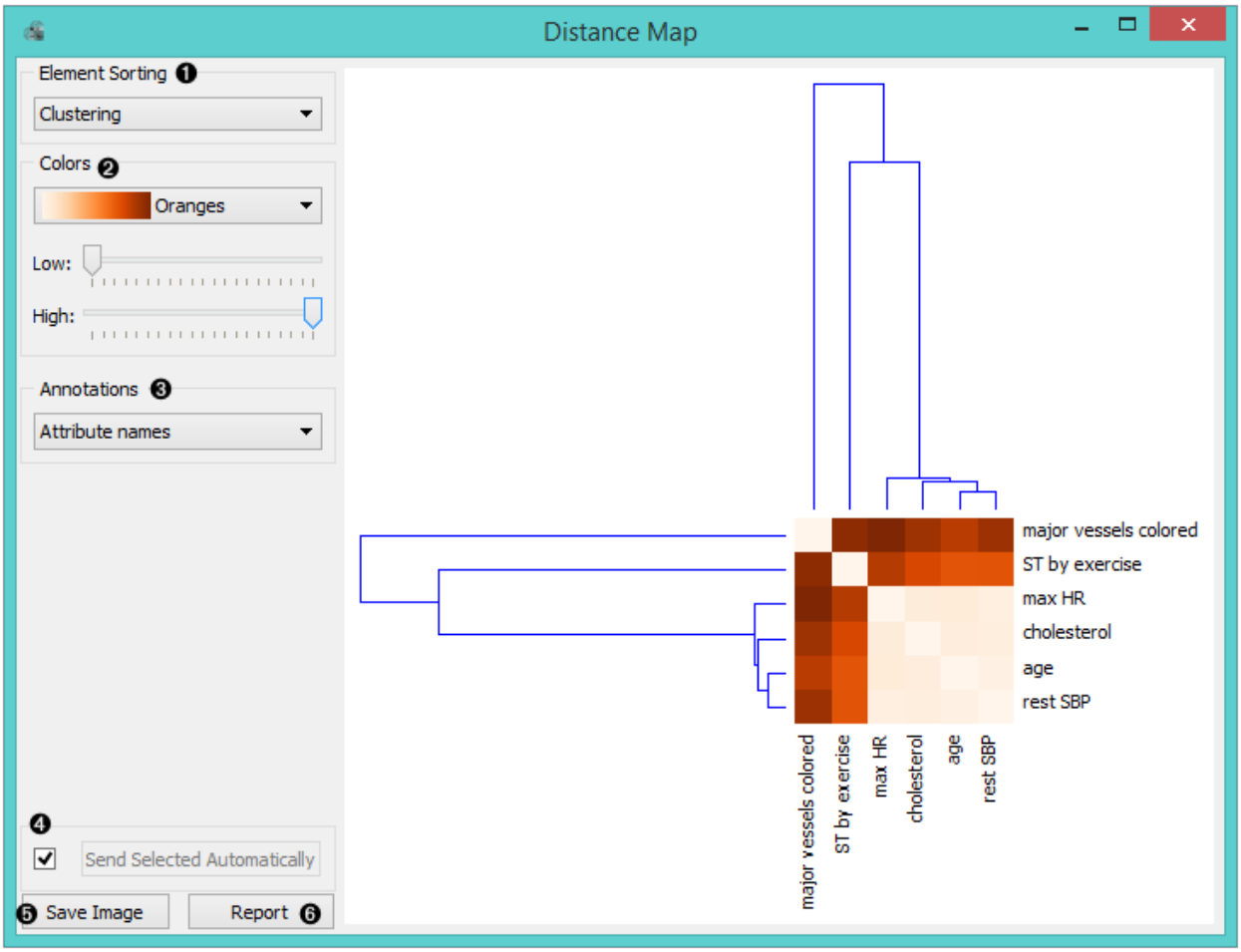
Distances are most often those between instances ("*rows*" in the *Distances* widget) or attributes ("*columns*" in *Distances* widget). The only suitable input for **Distance Map** is the *Distances* widget. For the output, the user can select a region of the map and the widget will output the corresponding instances or attributes. Also note that the **Distances** widget ignores discrete values and calculates distances only for continuous data, thus it can only display distance map for discrete data if you *Continuize* them first.

The snapshot shows distances between columns in the *heart disease* data, where smaller distances are represented with light and larger with dark orange. The matrix is symmetric and the diagonal is a light shade of orange - no attribute is different from itself. Symmetricity is always assumed, while the diagonal may also be non-zero.

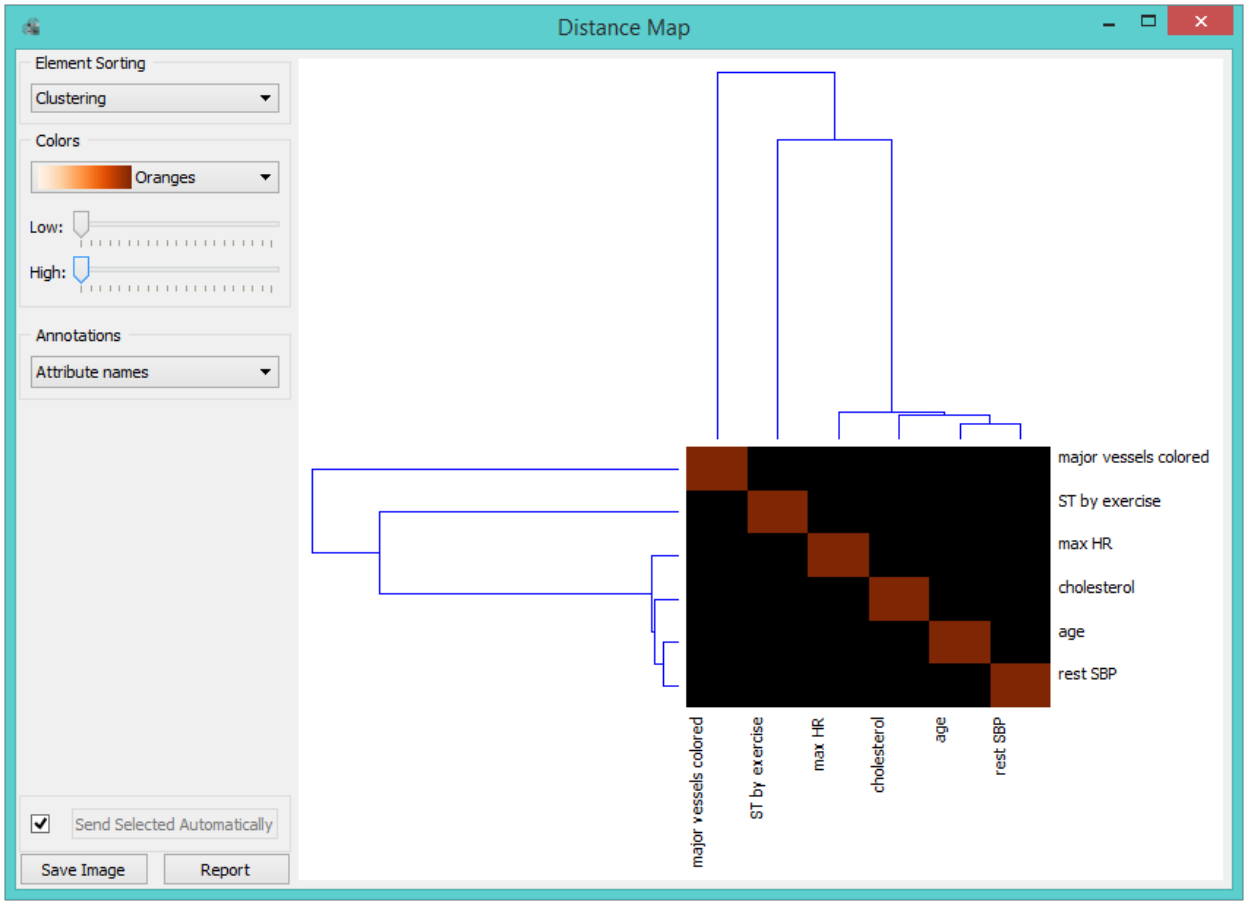
1. *Element sorting* arranges elements in the map by
   - None (lists instances as found in the data set)
   - **Clustering** (clusters data by similarity)
   - **Clustering with ordered leaves** (maximizes the sum of similarities of adjacent elements)
2. *Colors*
   - **Colors** (select the color palette for your distance map)
   - **Low** and **High** are thresholds for the color palette (low for instances or attributes with low distances and high for instances or attributes with high distances).
3. Select *Annotations*.
4. If *Send Selected Automatically* is on, the data subset is communicated automatically, otherwise you need to press *Send Selected*.
5. Press *Save Image* if you want to save the created image to your computer.
6. Produce a report.

Normally, a color palette is used to visualize the entire range of distances appearing in the matrix. This can be changed by setting the low and high threshold. In this way we ignore the differences in distances outside this interval and visualize the interesting part of the distribution.

Below, we visualized the most correlated attributes (distances by columns) in the *heart disease* data set by setting the color threshold for high distances to the minimum. We get a predominantly black square, where attributes with the lowest distance scores are represented by a lighter shade of the selected color schema (in our case: orange). Beside


Distance Map
Element Sorting
Clustering
Colors
Oranges
Low:
High:
Annotations
Attribute names
Send Selected Automatically
Save Image
Report
major vessels colored
ST by exercise
max HR
cholesterol
age
rest SBP

the diagonal line, we see that in our example *ST by exercise* and *major vessels colored* are the two attributes closest together.



The user can select a region in the map with the usual click-and-drag of the cursor. When a part of the map is selected, the widget outputs all items from the selected cells.

Examples

The first workflow shows a very standard use of the **Distance Map** widget. We select 70% of the original *Iris* data as our sample and view the distances between rows in **Distance Map**.

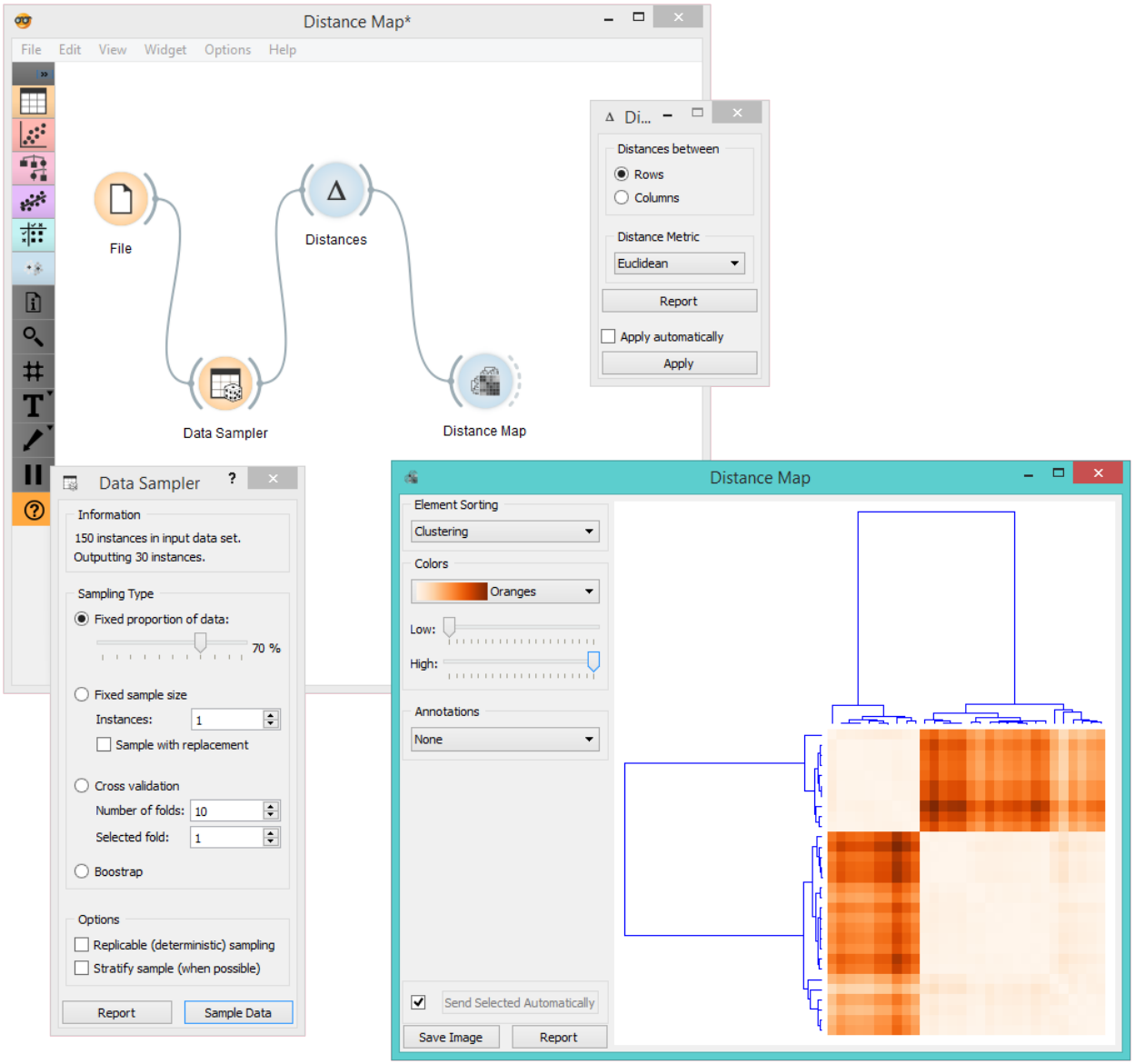
In the second example, we use the *heart disease* data again and select a subset of women only from the *Scatter Plot*. Then, we visualize distances between columns in the **Distance Map**. Since the subset also contains some discrete data, the *Distances* widget warns us it will ignore the discrete features, thus we will see only continuous instances/attributes in the map.

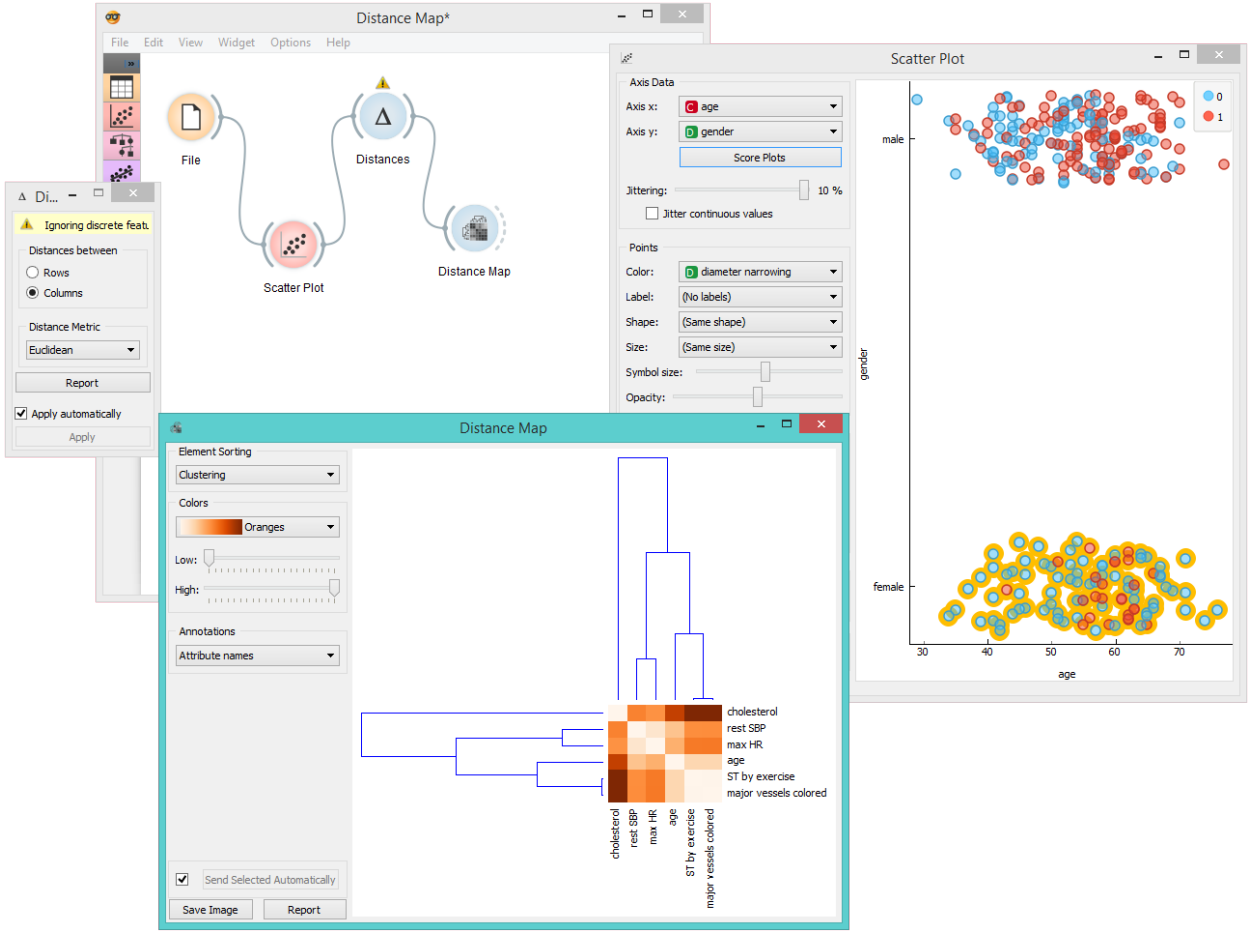
Distances

Computes distances between rows/columns in a data set.

Signals

Inputs:


Distance Map*
File Edit View Widget Options Help
File
Distances
Data Sampler
Distance Map
Di...
Distances between
Rows
Columns
Distance Metric
Euclidean
Report
Apply automatically
Apply
Data Sampler
Information
150 instances in input data set.
Outputting 30 instances.
Sampling Type
Fixed proportion of data:
70 %
Fixed sample size
Instances: 1
Sample with replacement
Cross validation
Number of folds: 10
Selected fold: 1
Boostrap
Options
Replicable (deterministic) sampling
Stratify sample (when possible)
Report
Sample Data
Distance Map
Element Sorting
Clustering
Colors
Oranges
Low:
High:
Annotations
None
Send Selected Automatically
Save Image
Report


Distance Map*
File Edit View Widget Options Help
File
Distances
Scatter Plot
Distance Map
Di...
Ignoring discrete featu
Distances between
Rows
Columns
Distance Metric
Euclidean
Report
Apply automatically
Apply
Scatter Plot
Axis Data
Axis x: age
Axis y: gender
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: diameter narrowing
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
male
female
gender
30
40
50
60
70
age
0
1
Distance Map
Element Sorting
Clustering
Colors
Oranges
Low:
High:
Annotations
Attribute names
Send Selected Automatically
Save Image
Report
cholesterol
rest SBP
max HR
age
ST by exercise
major vessels colored

- **Data**

  A data set

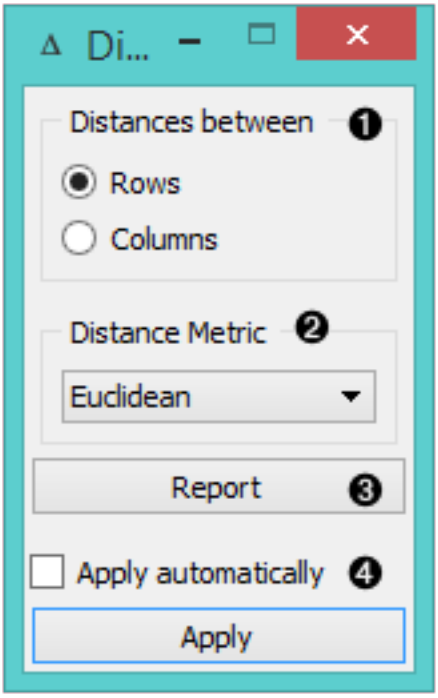
Outputs:

- **Distances**

  A distance matrix

Description

The **Distances** widget computes distances between rows or columns in a data set.



1. Choose whether to measure distances between rows or columns.
2. Choose the *Distance Metric*:
   - Euclidean ("straight line", distance between two points)
   - Manhattan (the sum of absolute differences for all attributes)
   - Cosine (the cosine of the angle between two vectors of an inner product space)
   - Jaccard (the size of the intersection divided by the size of the union of the sample sets)
   - Spearman (linear correlation between the rank of the values, remapped as a distance in a [0, 1] interval)
   - Spearman absolute (linear correlation between the rank of the absolute values, remapped as a distance in a [0, 1] interval)
   - Pearson (linear correlation between the values, remapped as a distance in a [0, 1] interval)
   - Pearson absolute (linear correlation between the absolute values, remapped as a distance in a [0, 1] interval)

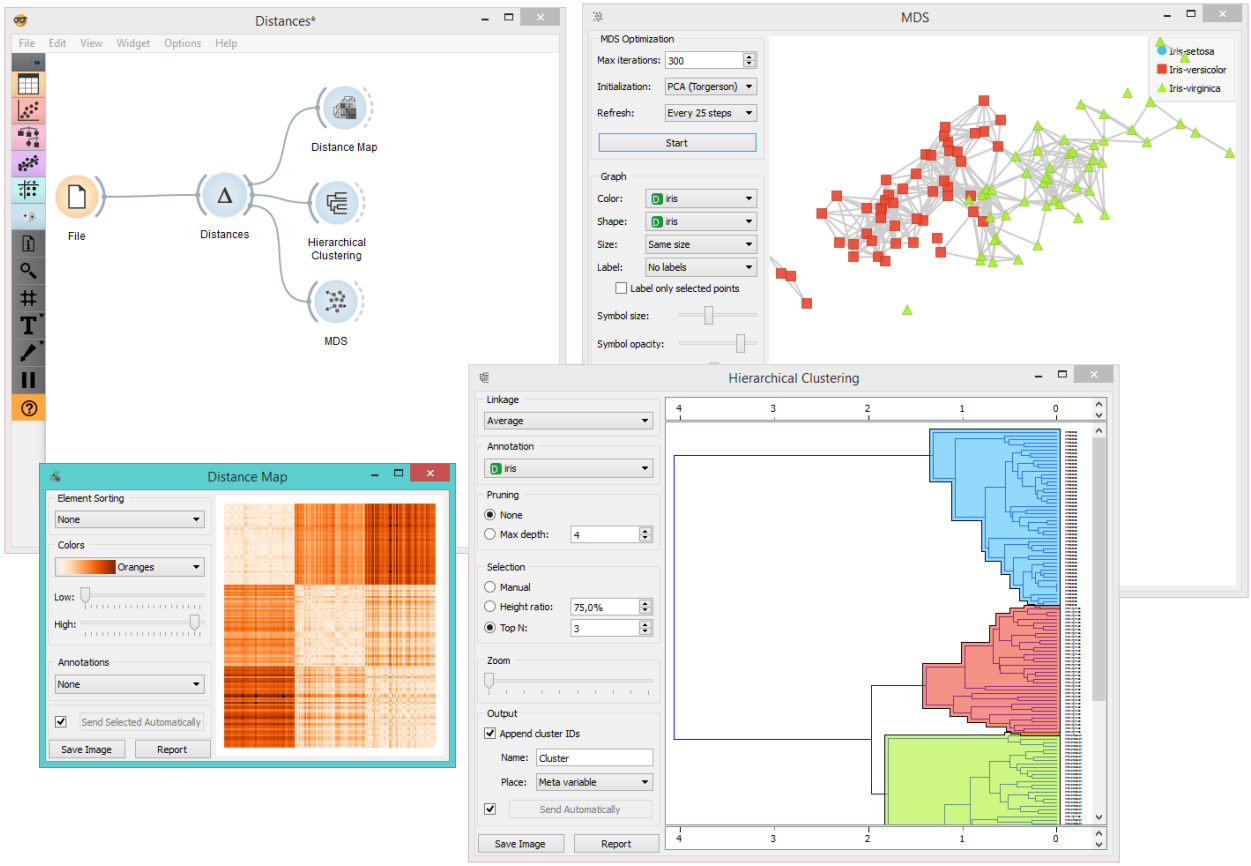
In case of missing values, the widget automatically imputes the average value of the row or the column.

Since the widget cannot compute distances between discrete and continuous attributes, it only uses continuous attributes and ignores the discrete ones. If you want to use discrete attributes, continuize them with the *Continuize* widget first.

3. Produce a report.
4. Tick *Apply Automatically* to automatically commit changes to other widgets. Alternatively, press '*Apply*'.

Example

This widget needs to be connected to another widget to display results, for instance to *Distance Map* to visualize distances, *Hierarchical Clustering* to cluster the attributes, or *MDS* to visualize the distances in a plane.



Distance Matrix

Visualizes distance measures in a distance matrix.

Signals

Inputs:

- **Distances**

  A distance matrix.

Outputs:

- **Distances**

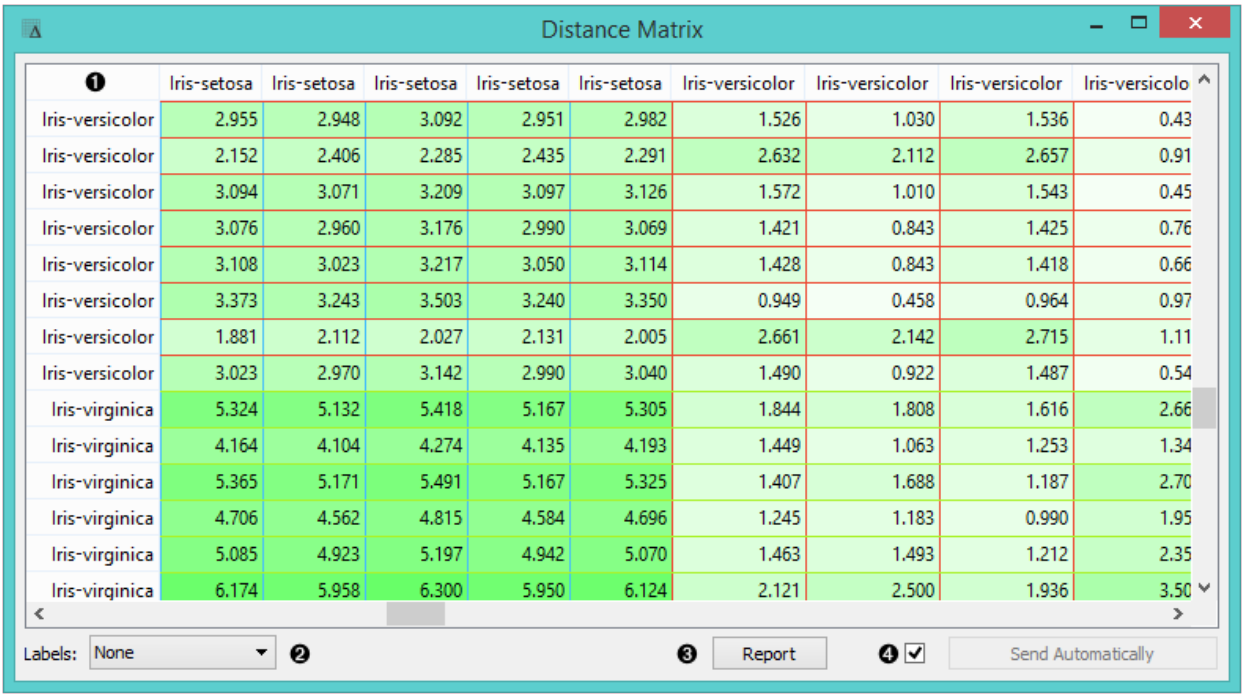
  A distance matrix.

- **Table**

  Distance measures in a distance matrix.

Description

The **Distance Matrix** widget creates a distance matrix, which is a two-dimensional array containing the distances, taken pairwise, between the elements of a set. The number of elements in the data set defines the size of the matrix. Data matrices are essential for hierarchical clustering and they are extremely useful in bioinformatics as well, where they are used to represent protein structures in a coordinate-independent manner.

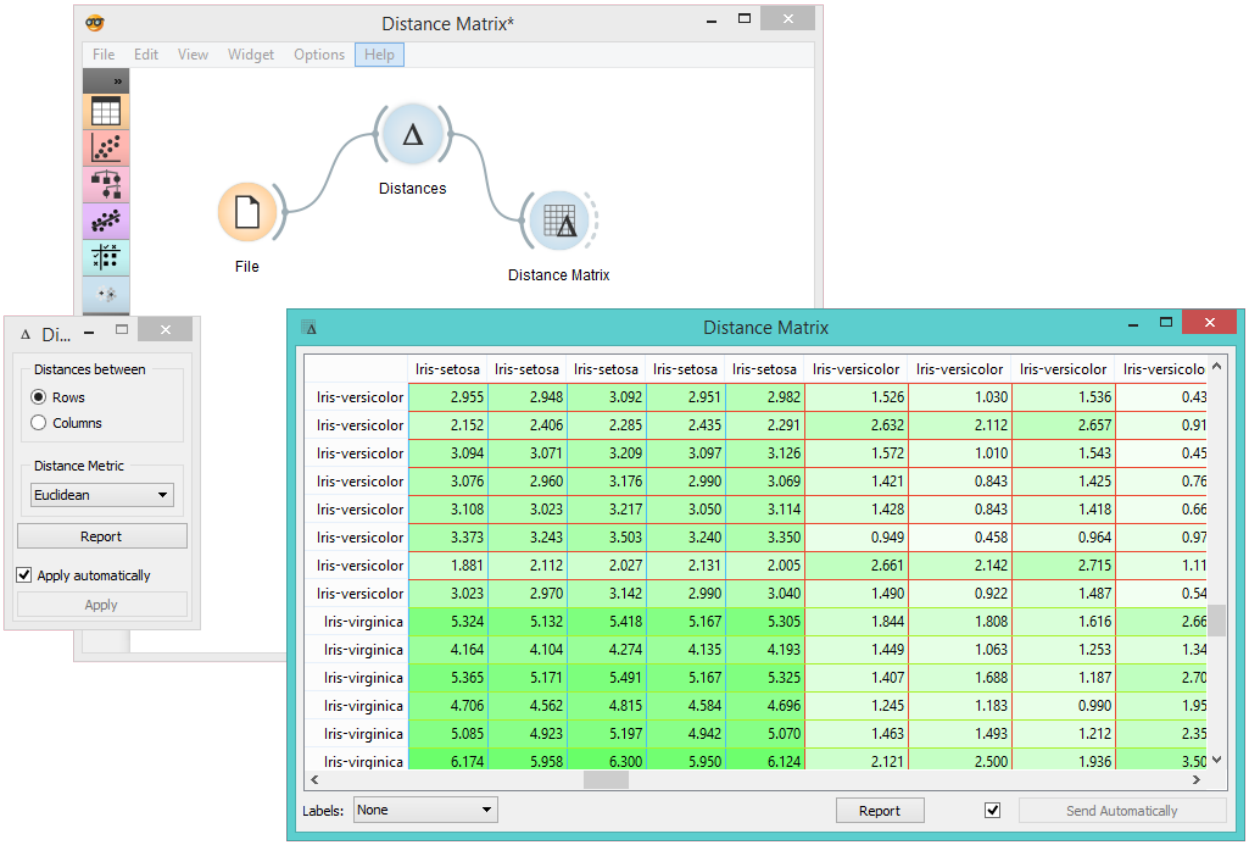


1. Elements in the data set and the distances between them
2. Label the table. The options are: *none*, *enumeration*, *according to variables*.
3. Produce a report.
4. Click *Send* to communicate changes to other widgets. Alternatively, tick the box in front of the *Send* button and changes will be communicated automatically (*Send Automatically*).

The only two suitable inputs for **Distance Matrix** are the *Distances* widget and the *Distance Transformation* widget. The output of the widget is a data table containing the distance matrix. The user can decide how to label the table and the distance matrix (or instances in the distance matrix) can then be visualized or displayed in a separate data table.

Example

The example below displays a very standard use of the **Distance Matrix** widget. We compute the distances between rows in the sample from the *Iris* data set and output them in the **Distance Matrix**. It comes as no surprise that Iris Virginica and Iris Setosa are the furthest apart.



	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-versicolor	Iris-versicolor	Iris-versicolor	Iris-versicolo
Iris-versicolor	2.955	2.948	3.092	2.951	2.982	1.526	1.030	1.536	0.43
Iris-versicolor	2.152	2.406	2.285	2.435	2.291	2.632	2.112	2.657	0.91
Iris-versicolor	3.094	3.071	3.209	3.097	3.126	1.572	1.010	1.543	0.45
Iris-versicolor	3.076	2.960	3.176	2.990	3.069	1.421	0.843	1.425	0.76
Iris-versicolor	3.108	3.023	3.217	3.050	3.114	1.428	0.843	1.418	0.66
Iris-versicolor	3.373	3.243	3.503	3.240	3.350	0.949	0.458	0.964	0.97
Iris-versicolor	1.881	2.112	2.027	2.131	2.005	2.661	2.142	2.715	1.11
Iris-versicolor	3.023	2.970	3.142	2.990	3.040	1.490	0.922	1.487	0.54
Iris-virginica	5.324	5.132	5.418	5.167	5.305	1.844	1.808	1.616	2.66
Iris-virginica	4.164	4.104	4.274	4.135	4.193	1.449	1.063	1.253	1.34
Iris-virginica	5.365	5.171	5.491	5.167	5.325	1.407	1.688	1.187	2.70
Iris-virginica	4.706	4.562	4.815	4.584	4.696	1.245	1.183	0.990	1.95
Iris-virginica	5.085	4.923	5.197	4.942	5.070	1.463	1.493	1.212	2.35
Iris-virginica	6.174	5.958	6.300	5.950	6.124	2.121	2.500	1.936	3.50

Distance Transformation

Transforms distances in a data set.

Signals

Inputs:

- **Distances**

  A distance matrix

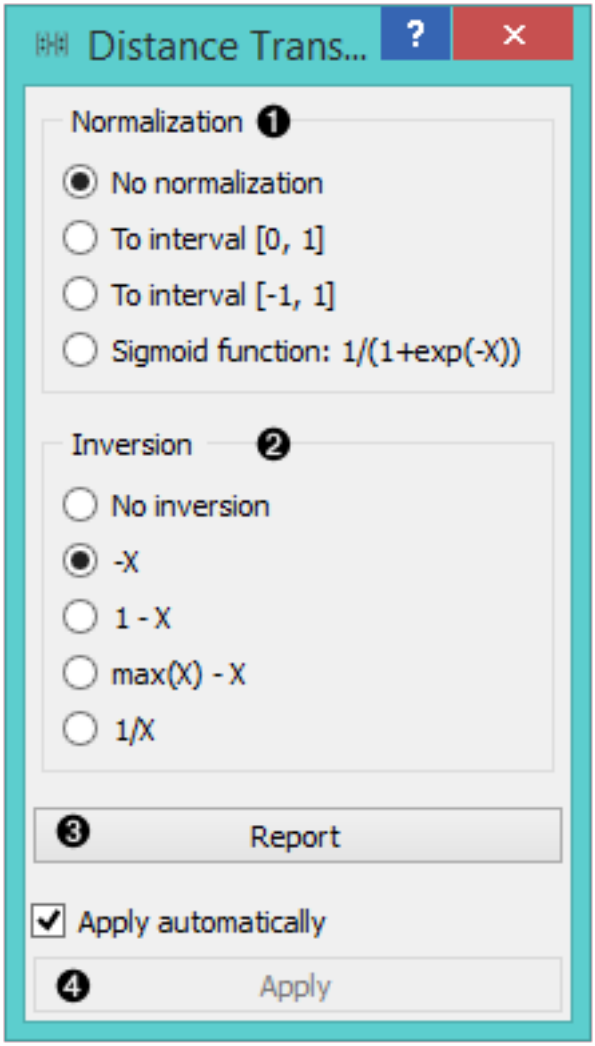
Outputs:

- **Distances**

  A distance matrix

Description

The **Distances Transformation** widget is used for the normalization and inversion of distance matrices. The normalization of data is necessary to bring all the variables into proportion with one another.



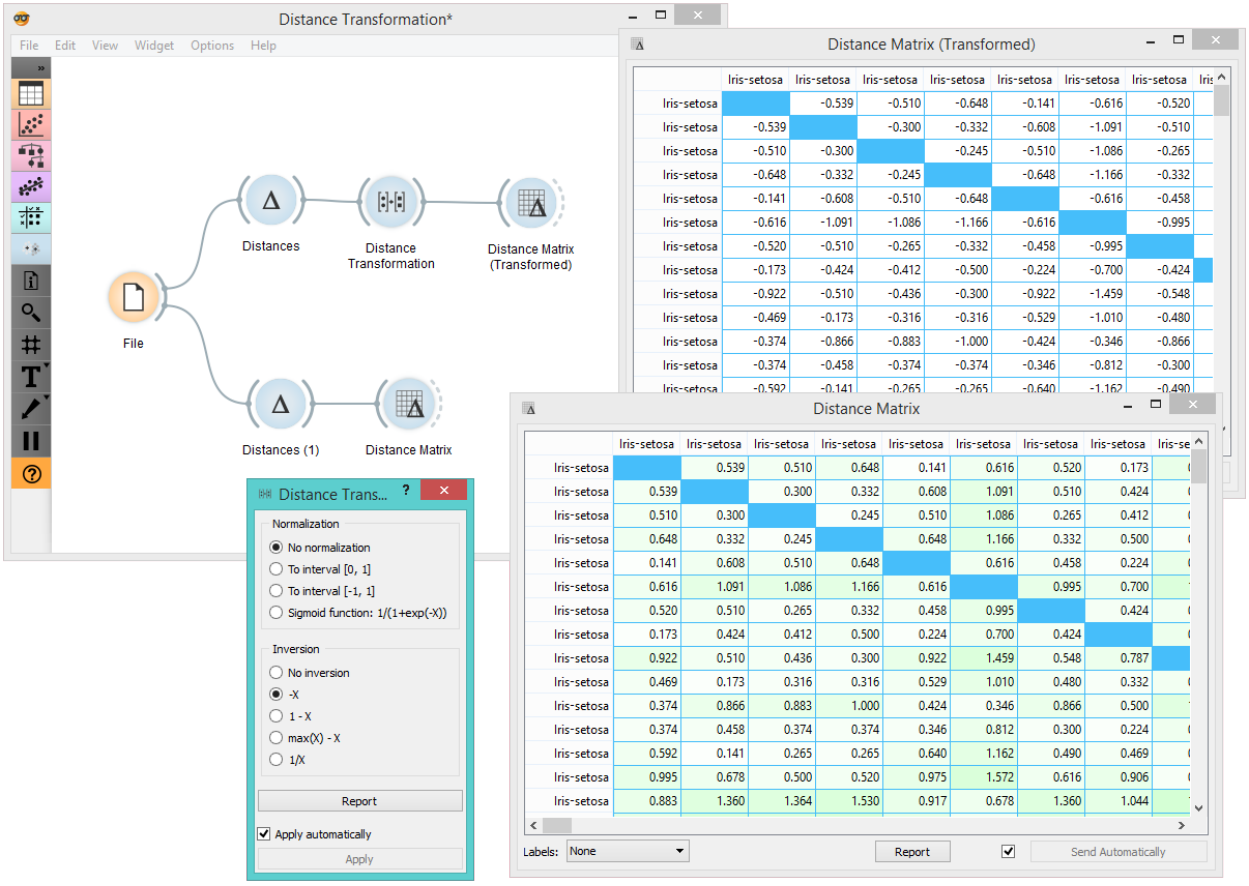
1. Choose the type of Normalization:
   - **No normalization**
   - **To interval [0, 1]**
   - **To interval [-1, 1]**

- Sigmoid function: 1/(1+exp(-X))

2. Choose the type of Inversion:
   - **No inversion**
   - **-X**
   - **1 - X**
   - **max(X) - X**
   - **1/X**
3. Produce a report.
4. After changing the settings, you need to click *Apply* to commit changes to other widgets. Alternatively, tick *Apply automatically*.

Example

In the snapshot below, you can see how transformation affects the distance matrix. We loaded the *Iris* data set and calculated the distances between rows with the help of the *Distances* widget. In order to demonstrate how **Distance Transformation** affects the *Distance Matrix*, we created the worflow below and compared the transformed distance matrix with the "original" one.



Distance File

Loads an existing distance file.

Signals

Inputs:

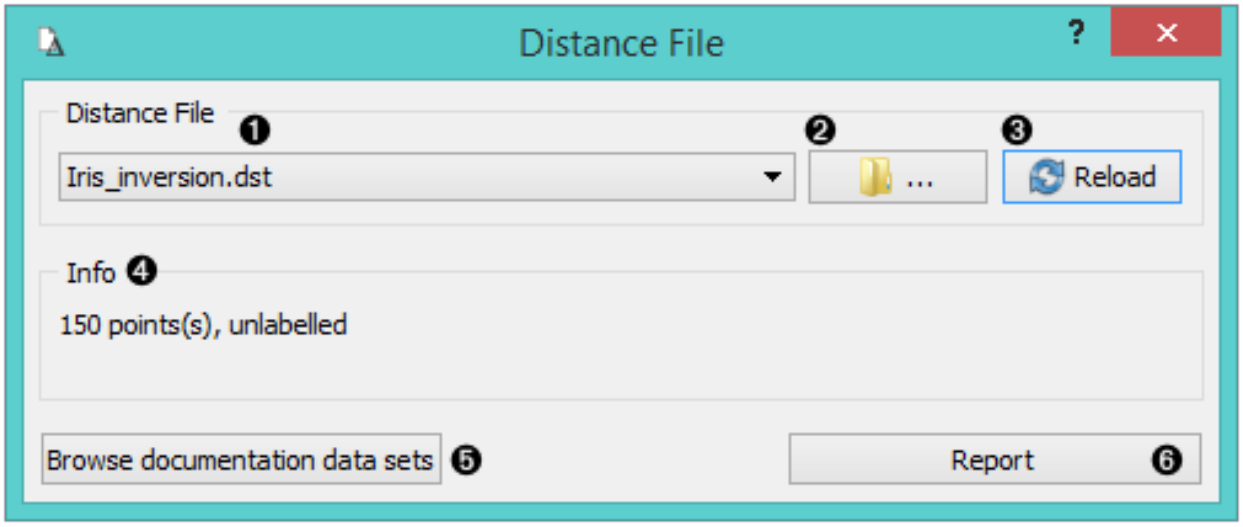
- None

Outputs:

- **Distance File**

  A distance matrix.

Description

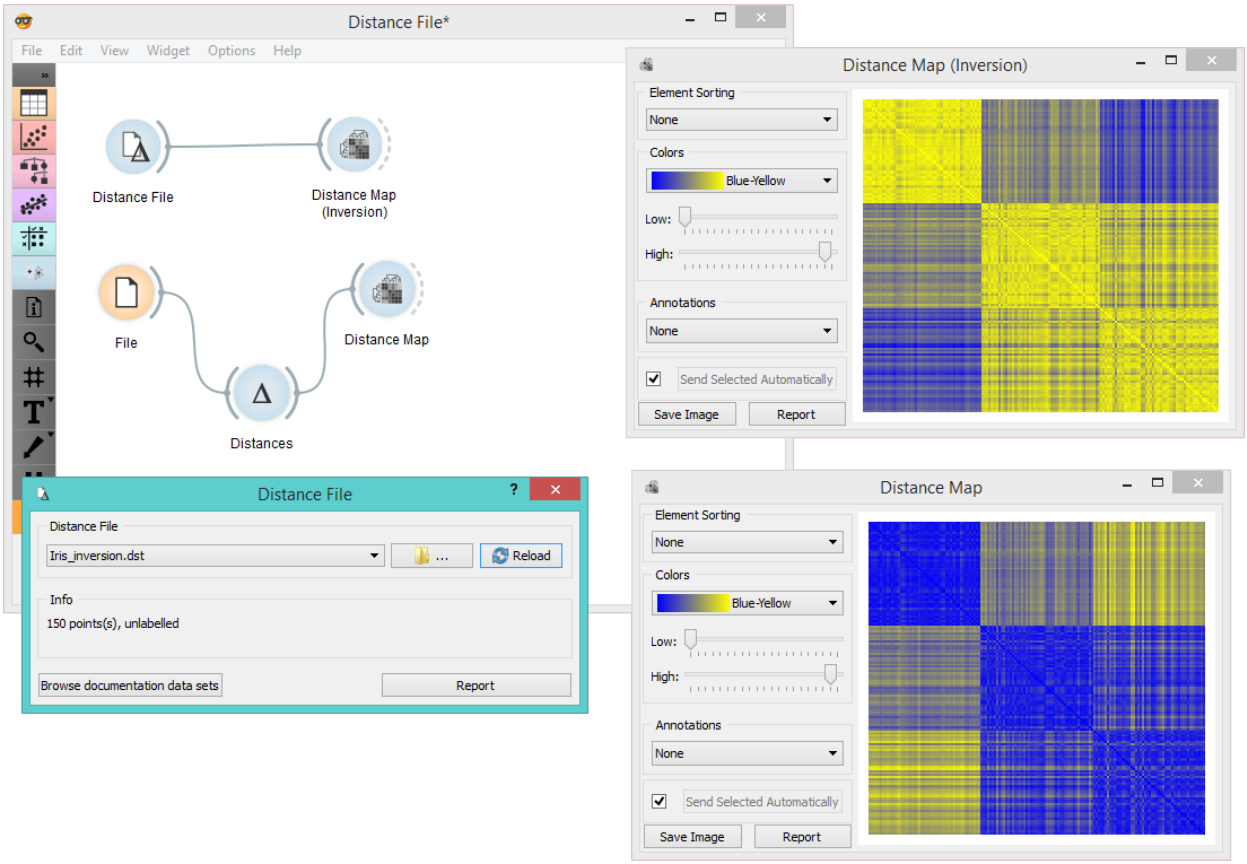


1. Choose from a list of previously saved distance files.
2. Browse for saved distance files.
3. Reload the selected distance file.
4. Information about the distance file (number of points, labelled/unlabelled)
5. Browse documentation data sets.
6. Produce a report.

Example

When you want to use a custom-set distance file that you've saved before, open the **Distance File** widget and select the desired file with the *Browse* icon. This widget loads the existing distance file. In the snapshot below, we loaded the

transformed *Iris* distance matrix from the *Save Distance Matrix* example. We displayed the transformed data matrix in the *Distance Map* widget. We also decided to display a distance map of the original *Iris* data set for comparison.



Save Distance Matrix

Saves a distance matrix.

Signals

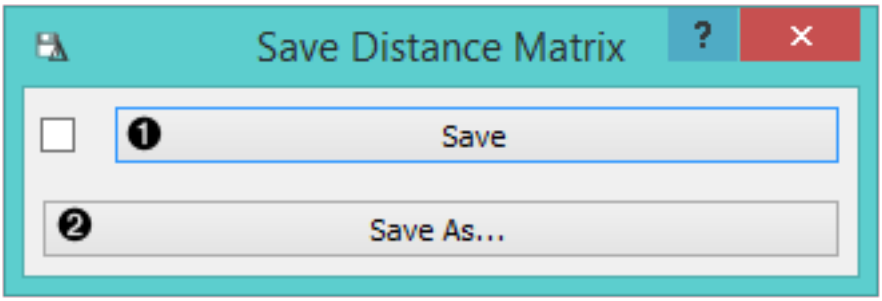
Inputs:

- **Distances**

  A distance matrix.

Outputs:

- None



Description

1. By clicking *Save*, you choose from previously saved distance matrices. Alternatively, tick the box on the left side of the *Save* button and changes will be communicated automatically.
2. By clicking *Save as*, you save the distance matrix to your computer, you only need to enter the name of the file and click *Save*. The distance matrix will be saved as type *.dst*.

Example

In the snapshot below, we used the *Distance Transformation* widget to transform the distances in the *Iris* data set. We then chose to save the transformed version to our computer, so we could use it later on. We decided to output all data instances. You can choose to output just a minor subset of the data matrix. Pairs are marked automatically. If you wish to know what happened to our changed file, go *here*

Hierarchical Clustering

Groups items using a hierarchical clustering algorithm.

Signals

Inputs:

- **Distances**

  A distance matrix

Outputs:

- **Selected Data**

  A data subset

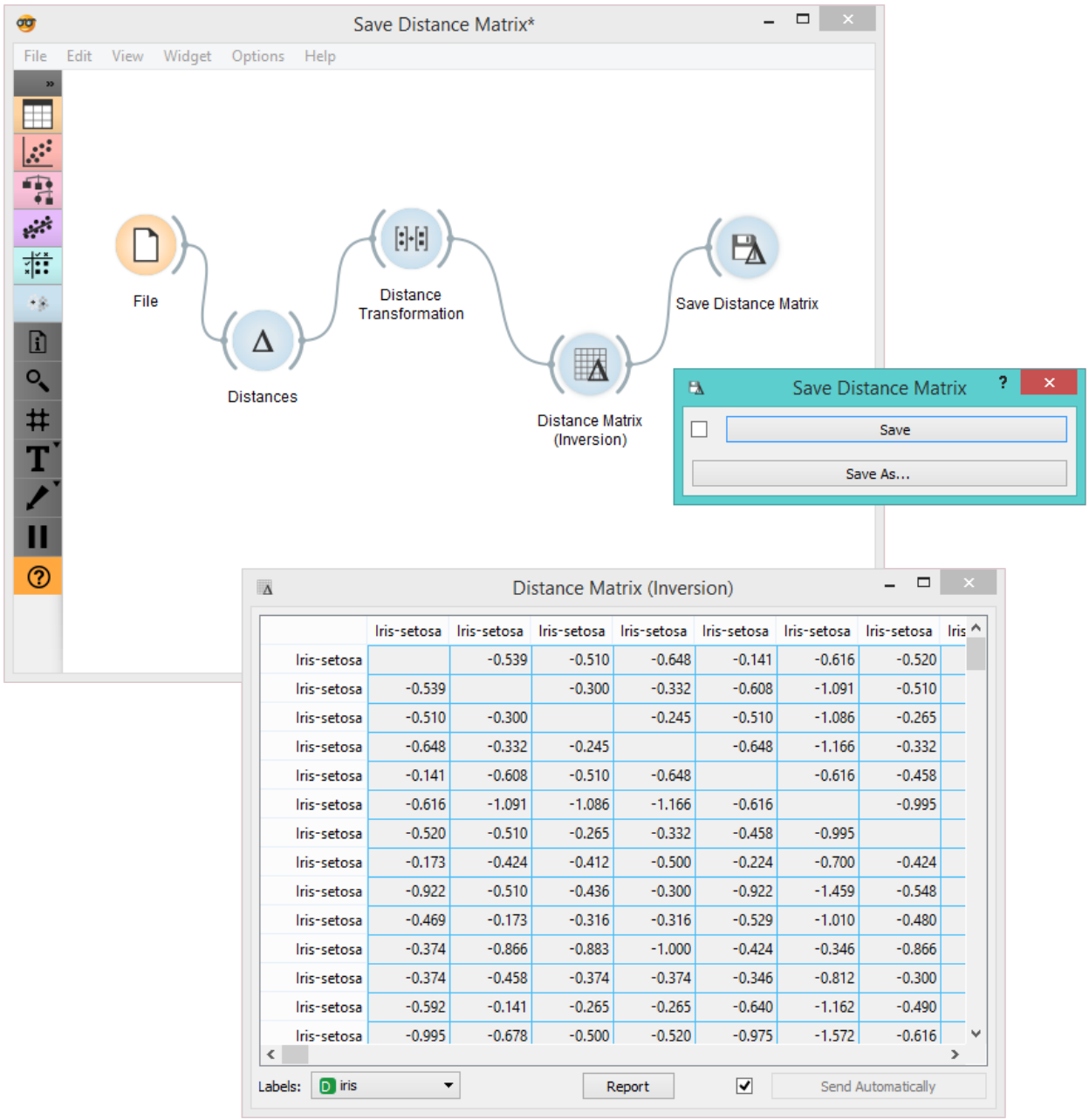
- **Other Data**

  Remaining data

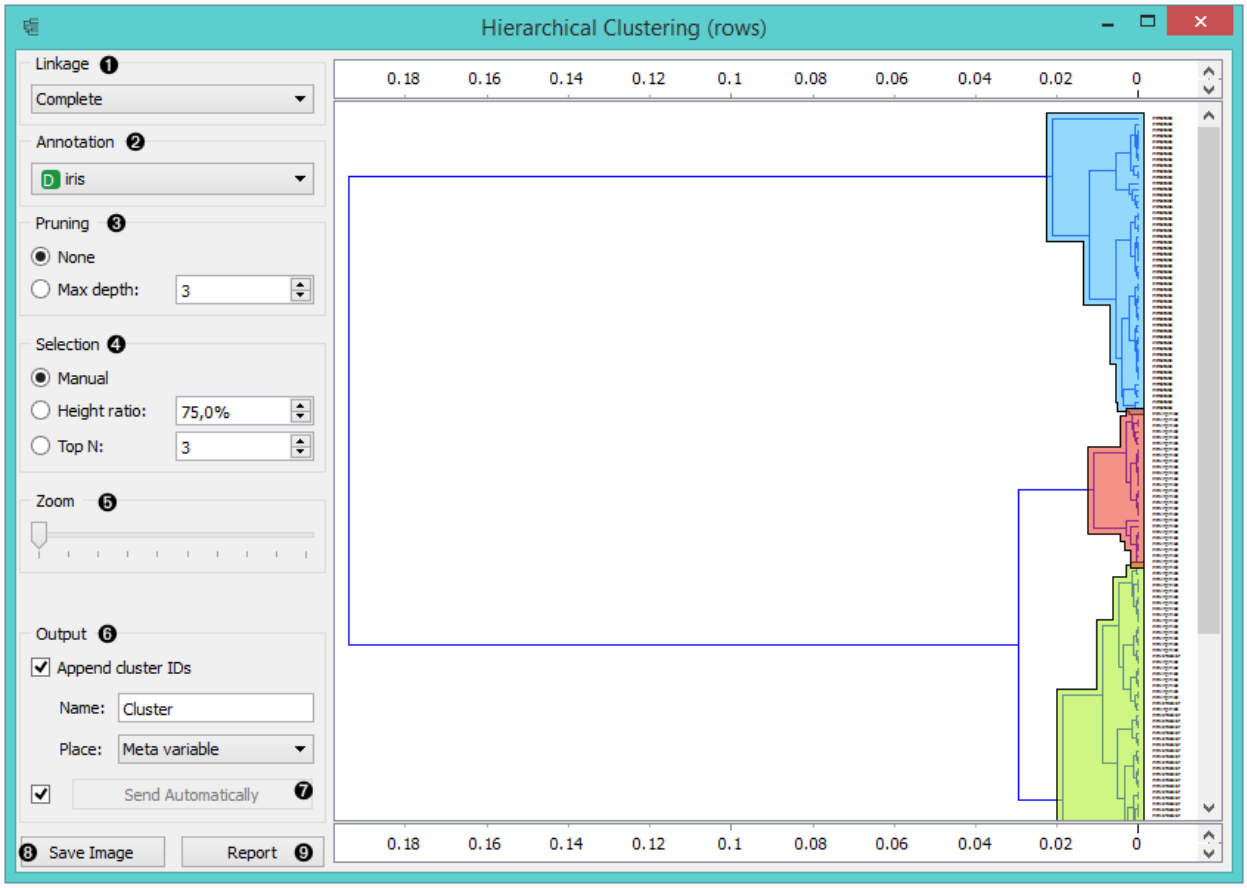
Description

The widget computes hierarchical clustering of arbitrary types of objects from a matrix of distances and shows a corresponding dendrogram.

1. The widget supports four ways of measuring distances between clusters:
   - **Single linkage** computes the distance between the closest elements of the two clusters



	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris
Iris-setosa		-0.539	-0.510	-0.648	-0.141	-0.616	-0.520	
Iris-setosa	-0.539		-0.300	-0.332	-0.608	-1.091	-0.510	
Iris-setosa	-0.510	-0.300		-0.245	-0.510	-1.086	-0.265	
Iris-setosa	-0.648	-0.332	-0.245		-0.648	-1.166	-0.332	
Iris-setosa	-0.141	-0.608	-0.510	-0.648		-0.616	-0.458	
Iris-setosa	-0.616	-1.091	-1.086	-1.166	-0.616		-0.995	
Iris-setosa	-0.520	-0.510	-0.265	-0.332	-0.458	-0.995		
Iris-setosa	-0.173	-0.424	-0.412	-0.500	-0.224	-0.700	-0.424	
Iris-setosa	-0.922	-0.510	-0.436	-0.300	-0.922	-1.459	-0.548	
Iris-setosa	-0.469	-0.173	-0.316	-0.316	-0.529	-1.010	-0.480	
Iris-setosa	-0.374	-0.866	-0.883	-1.000	-0.424	-0.346	-0.866	
Iris-setosa	-0.374	-0.458	-0.374	-0.374	-0.346	-0.812	-0.300	
Iris-setosa	-0.592	-0.141	-0.265	-0.265	-0.640	-1.162	-0.490	
Iris-setosa	-0.995	-0.678	-0.500	-0.520	-0.975	-1.572	-0.616	


Hierarchical Clustering (rows)
Linkage 1
Complete
Annotation 2
iris
Pruning 3
None
Max depth:
3
Selection 4
Manual
Height ratio:
75,0%
Top N:
3
Zoom 5
Output 6
Append cluster IDs
Name:
Cluster
Place:
Meta variable
Send Automatically 7
8 Save Image
Report 9
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

- **Average linkage** computes the average distance between elements of the two clusters
- **Weighted linkage** uses the WPGMA method
- **Complete linkage** computes the distance between the clusters' most distant elements

2. Labels of nodes in the dendrogram can be chosen in the **Annotation** box.
3. Huge dendrograms can be pruned in the *Pruning* box by selecting the maximum depth of the dendrogram. This only affects the display, not the actual clustering.
4. The widget offers three different selection methods:
   - **Manual** (Clicking inside the dendrogram will select a cluster. Multiple clusters can be selected by holding Ctrl/Cmd. Each selected cluster is shown in a different color and is treated as a separate cluster in the output.)
   - **Height ratio** (Clicking on the bottom or top ruler of the dendrogram places a cutoff line in the graph. Items to the right of the line are selected.)
   - **Top N** (Selects the number of top nodes.)
5. Use *Zoom* and scroll to zoom in or out.
6. If the items being clustered are instances, they can be added a cluster index (*Append cluster IDs*). The ID can appear as an ordinary **Attribute**, **Class attribute** or a **Meta attribute**. In the second case, if the data already has a class attribute, the original class is placed among meta attributes.
7. The data can be automatically output on any change (*Auto send is on*) or, if the box isn't ticked, by pushing *Send Data*.
8. Clicking this button produces an image that can be saved.
9. Produce a report.

Examples

The workflow below shows the output of **Hierarchical Clustering** for the *Iris* data set in *Data Table* widget. We see that if we choose *Append cluster IDs* in hierarchical clustering, we can see an additional column in the *Data Table* named *Cluster*. This is a way to check how hierarchical clustering clustered individual instances.

In the second example, we loaded the *Iris* data set again, but this time we added the *Scatter Plot*, showing all the instances from the *File* widget, while at the same time receiving the selected instances signal from **Hierarchical Clustering**. This way we can observe the position of the selected cluster(s) in the projection.

k-Means

Groups items using the k-Means clustering algorithm.

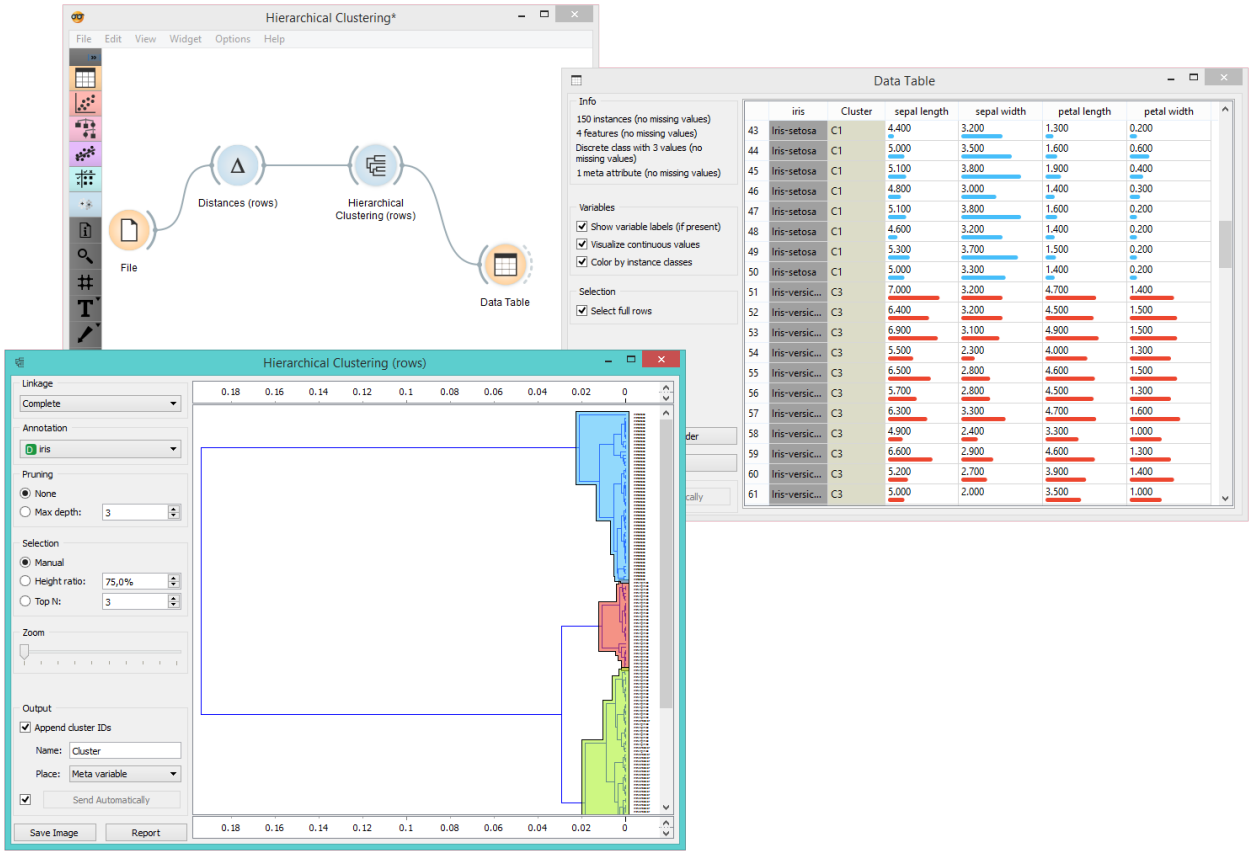
Signals

Inputs:

- **Data**

  A data set.

Outputs:



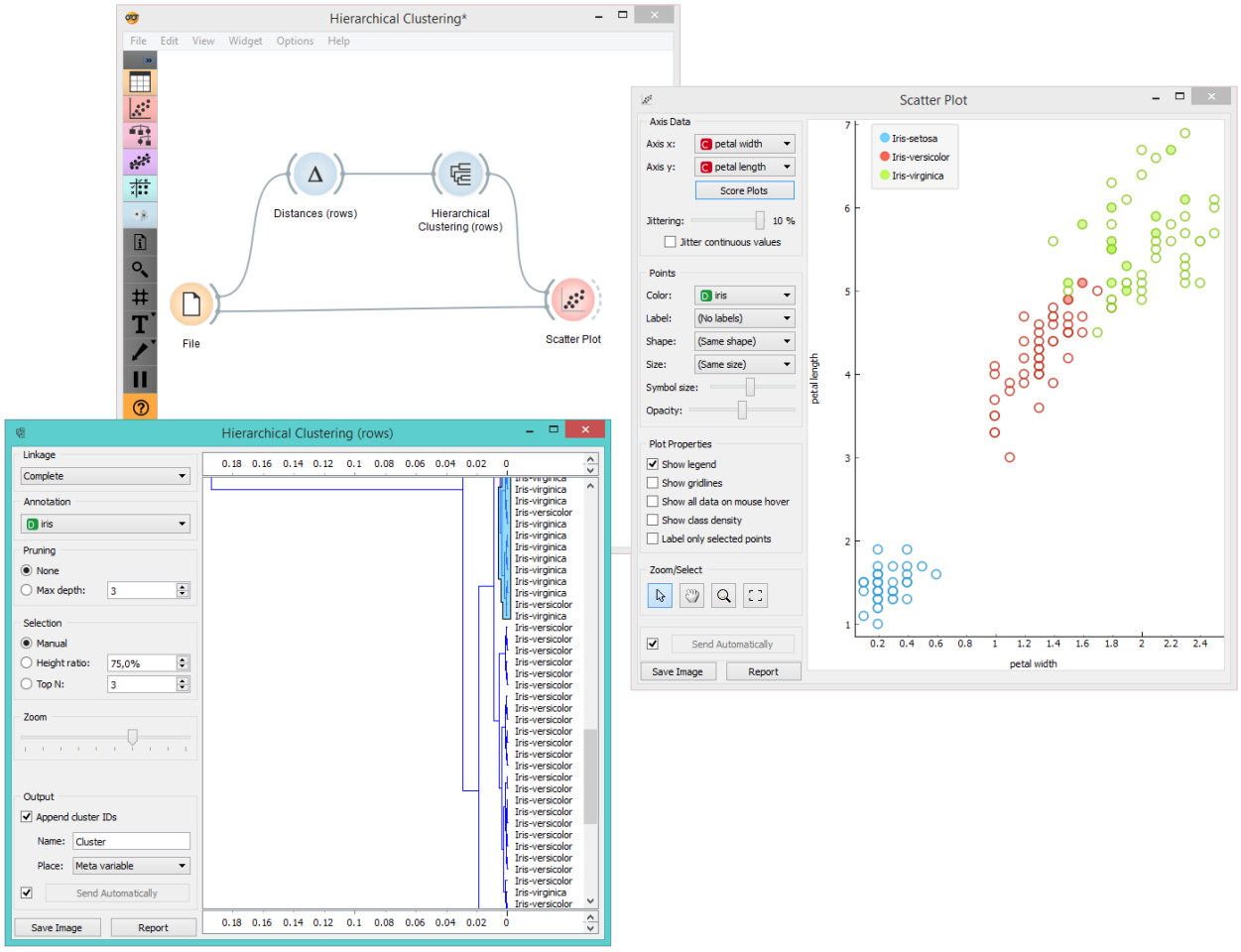
- **Data**

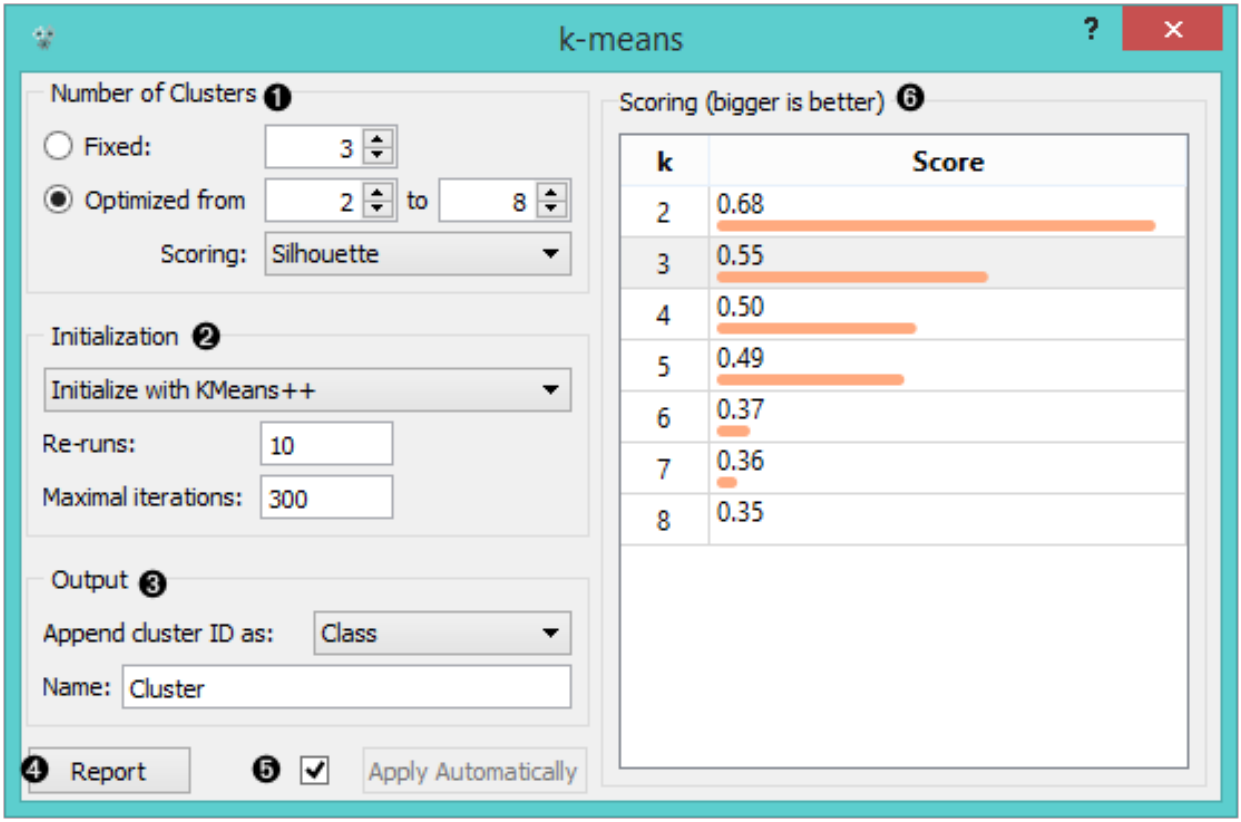
  A data set with cluster index as a class attribute.

Description

The widget applies the k-Means clustering algorithm to the data and outputs a new data set in which the cluster index is used as a class attribute. The original class attribute, if it exists, is moved to meta attributes. Scores of clustering results for various k are also shown in the widget.

1. Select the number of clusters.
   - **Fixed**: algorithm clusters data in a specified number of clusters.
   - **Optimized**: widget shows clustering scores for the selected cluster range.
   - Silhouette (contrasts average distance to elements in the same cluster with the average distance to elements in other clusters)
   - **Inter-cluster distance** (measures distances between clusters, normally between centroids)
   - **Distance to** centroids (measures distances to the arithmetic means of clusters)
2. Select the initialization method (the way the algorithm begins clustering):
   - k-Means++ (first center is selected randomly, subsequent are chosen from the remaining points with probability proportioned to squared distance from the closest center)
   - **Random initialization** (clusters are assigned randomly at first and then updated with further iterations)


Hierarchical Clustering*
File Edit View Widget Options Help
Distances (rows)
Hierarchical Clustering (rows)
File
Scatter Plot
Scatter Plot
Axis Data
Axis x: petal width
Axis y: petal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: iris
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
petal length
petal width
Hierarchical Clustering (rows)
Linkage
Complete
Annotation
iris
Pruning
None
Max depth: 3
Selection
Manual
Height ratio: 75,0%
Top N: 3
Zoom
Output
Append cluster IDs
Name: Cluster
Place: Meta variable
Send Automatically
Save Image
Report



Re-runs (how many times the algorithm is run) and **maximal iterations** (the maximum number of iteration within each algorithm run) can be set manually.

3. The widget outputs a new data set with appended cluster information. Select how to append cluster information (as class, feature or meta attribute) and name the column.
4. If *Apply Automatically* is ticked, the widget will commit changes automatically. Alternatively, click *Apply*.
5. Produce a report.
6. Check scores of clustering results for various k.

Examples

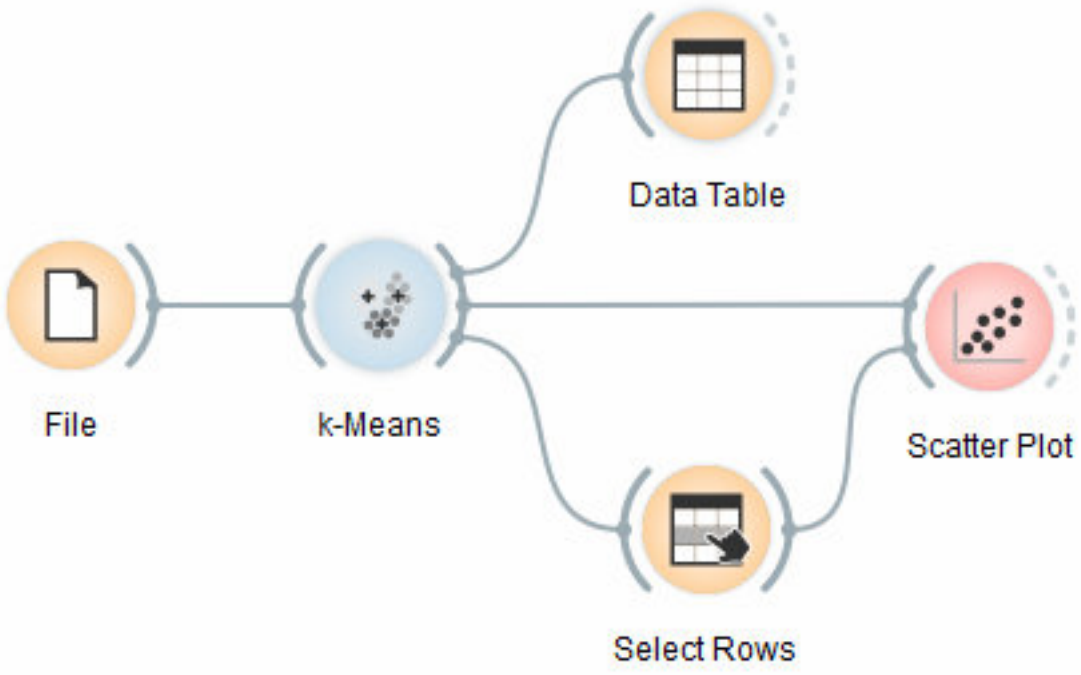
We are going to explore the widget with the following schema.

First, we load the *Iris* data set, divide it into three clusters and show it in the *Data Table*, where we can observe which instance went into which cluster. The interesting parts are the *Scatter Plot* and *Select Rows*.

Since **k-Means** added the cluster index as a class attribute, the scatter plot will color the points according to the clusters they are in.

What we are really interested in is how well the clusters induced by the (unsupervised) clustering algorithm match the actual classes in the data. We thus take *Select Rows* widget, in which we can select individual classes and have the corresponding points marked in the scatter plot. The match is perfect for *setosa*, and pretty good for the other two classes.

You may have noticed that we left the **Remove unused values/attributes** and **Remove unused classes** in *Select Rows* unchecked. This is important: if the widget modifies the attributes, it outputs a list of modified instances and the scatter plot cannot compare them to the original data.



Perhaps a simpler way to test the match between clusters and the original classes is to use the *Distributions* widget.

The only (minor) problem here is that this widget only visualizes normal (and not meta) attributes. We solve this by using *Select Columns*: we reinstate the original class *Iris* as the class and put the cluster index among the attributes.

The match is perfect for *setosa*: all instances of setosa are in the third cluster (blue). 48 *versicolors* are in the second cluster (red), while two ended up in the first. For *virginicae*, 36 are in the first cluster and 14 in the second.

MDS

Multidimensional scaling (MDS) projects items onto a plane fitted to given distances between points.

Signals

Inputs:

- **Distances**

  A distance matrix

- **Data**

  A data set

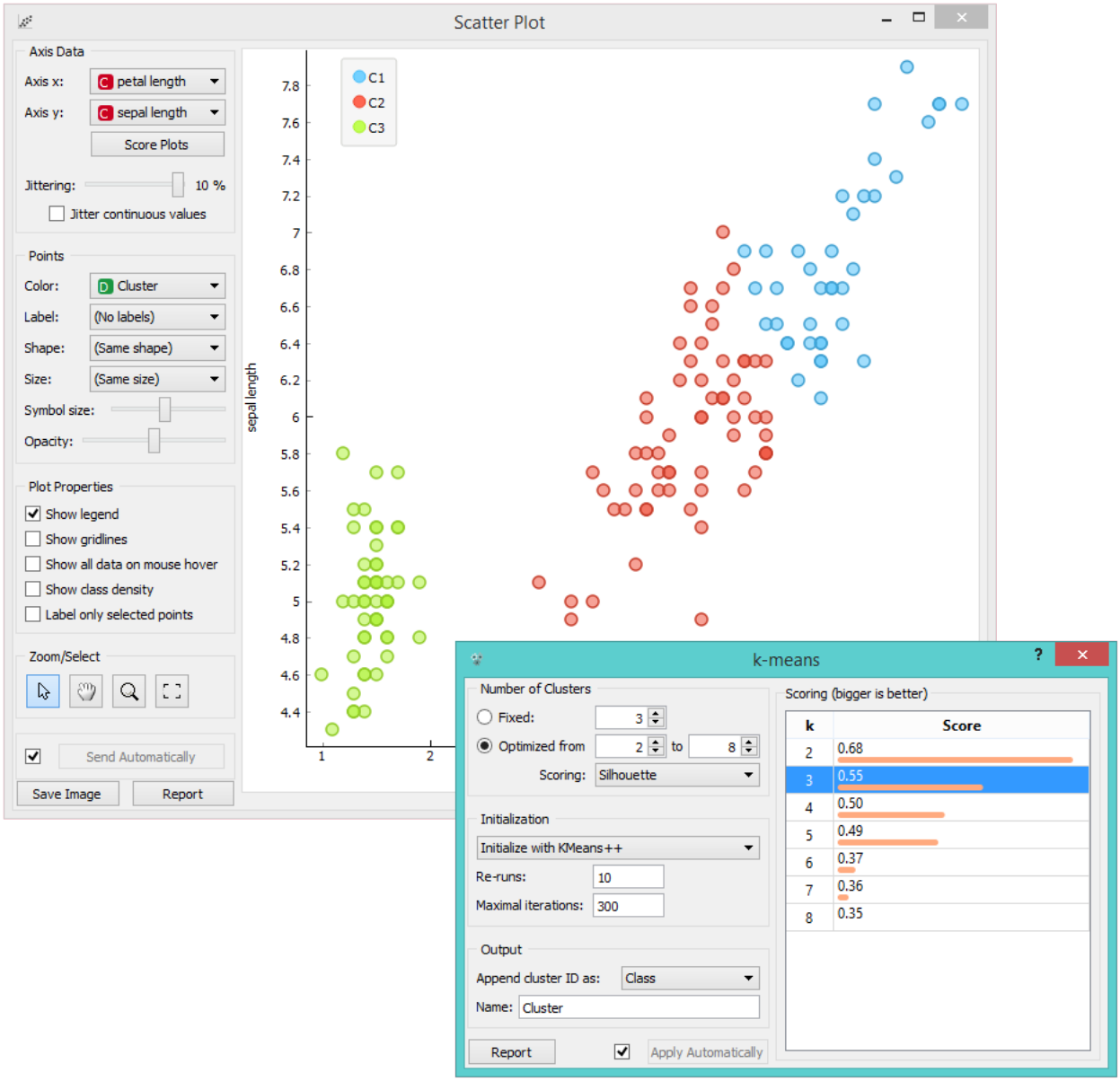
Outputs:

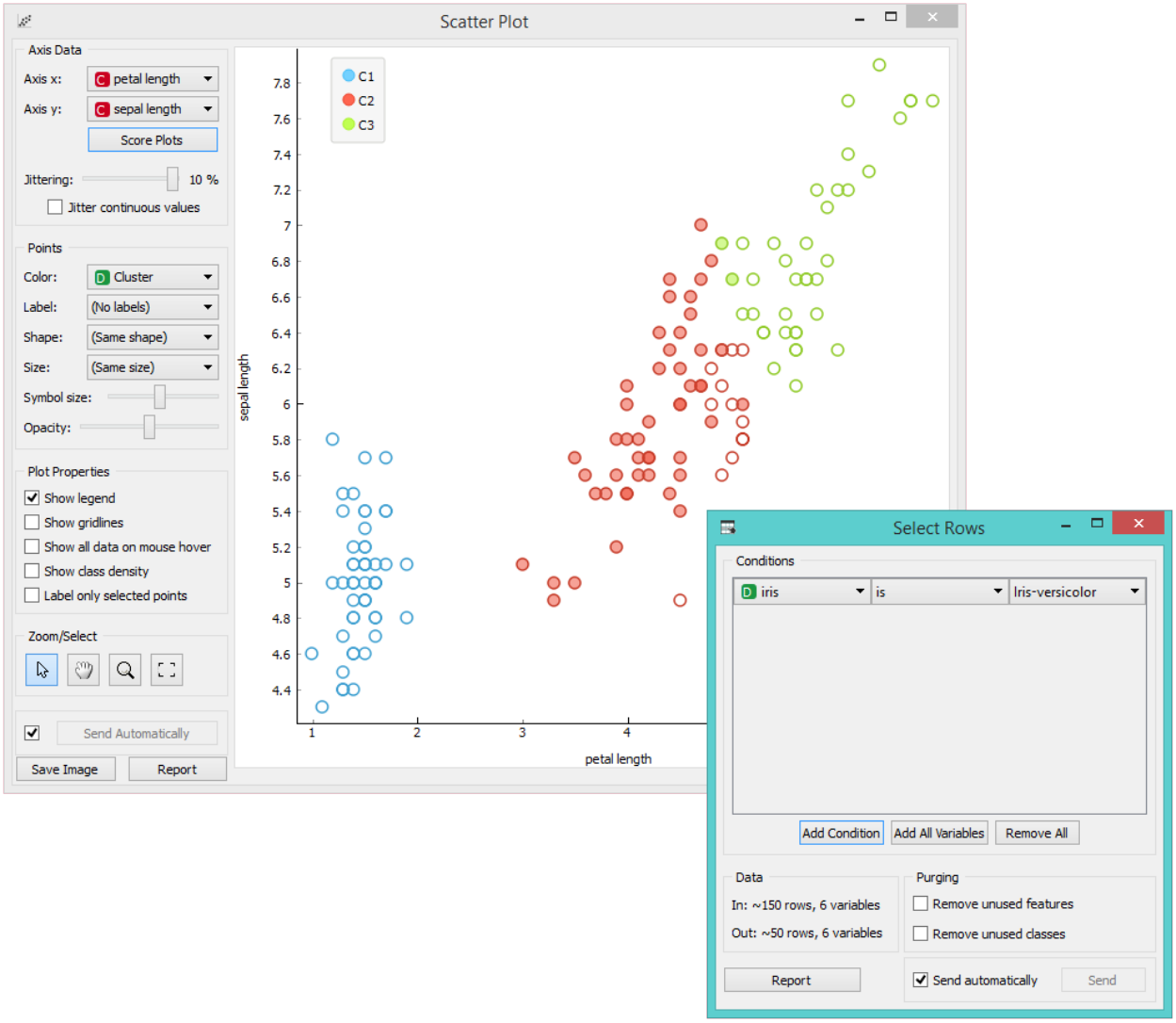
- **Data**

  A data set with MDS coordinates.

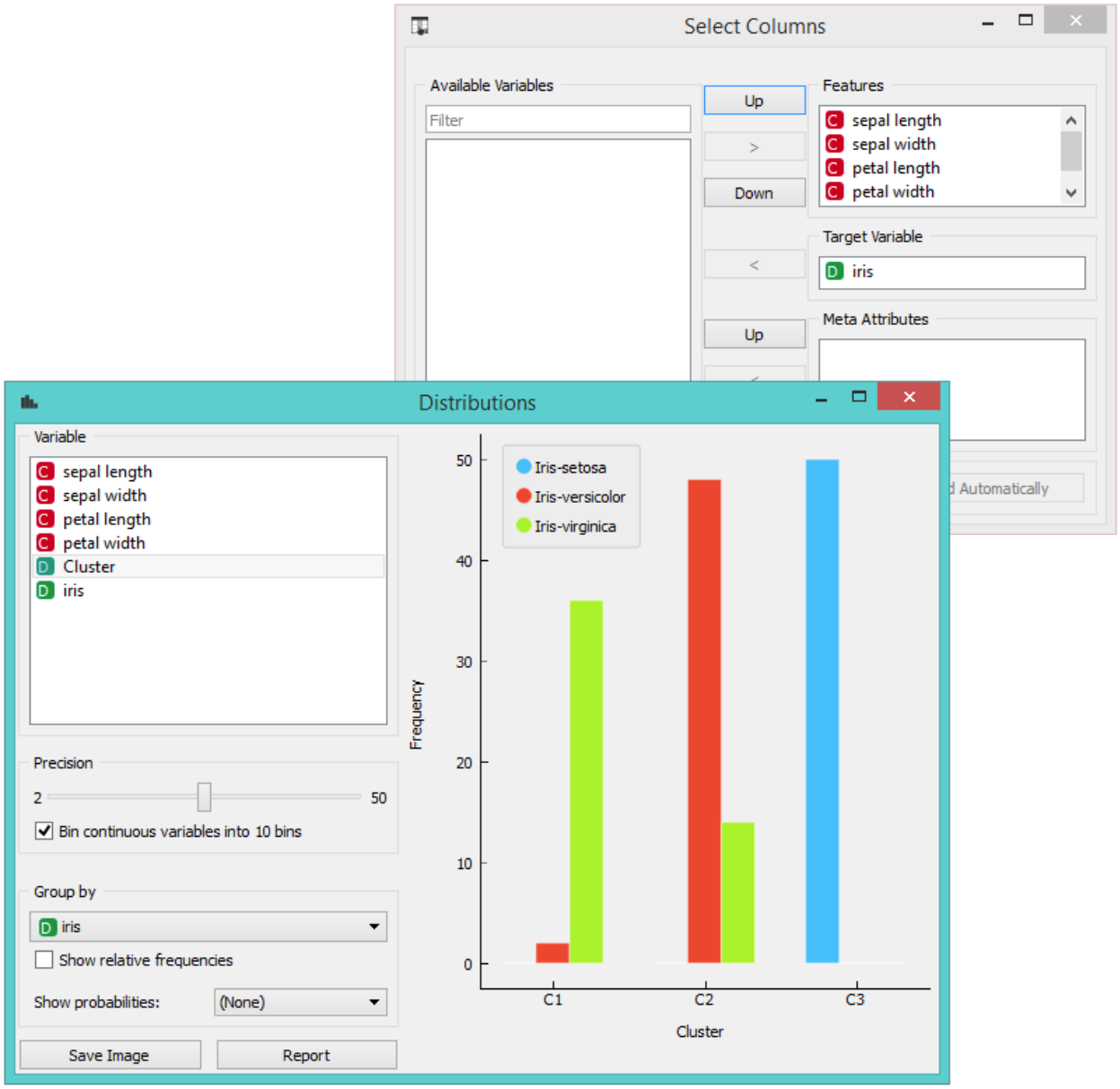
- **Data subset**

  Selected data


Scatter Plot
Axis Data
Axis x: petal length
Axis y: sepal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: Cluster
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
C1
C2
C3
sepal length
7.8
7.6
7.4
7.2
7
6.8
6.6
6.4
6.2
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4.4
1
2
k-means
Number of Clusters
Fixed: 3
Optimized from 2 to 8
Scoring: Silhouette
Initialization
Initialize with KMeans++
Re-runs: 10
Maximal iterations: 300
Output
Append cluster ID as: Class
Name: Cluster
Report
Apply Automatically
Scoring (bigger is better)
k
Score
2 0.68
3 0.55
4 0.50
5 0.49
6 0.37
7 0.36
8 0.35


Scatter Plot
Axis Data
Axis x: petal length
Axis y: sepal length
Score Plots
Jittering: 10 %
Jitter continuous values
Points
Color: Cluster
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
C1
C2
C3
sepal length
petal length
Select Rows
Conditions
iris
is
Iris-versicolor
Add Condition
Add All Variables
Remove All
Data
In: ~150 rows, 6 variables
Out: ~50 rows, 6 variables
Purging
Remove unused features
Remove unused classes
Report
Send automatically
Send

File
k-Means
Select Columns
Distributions

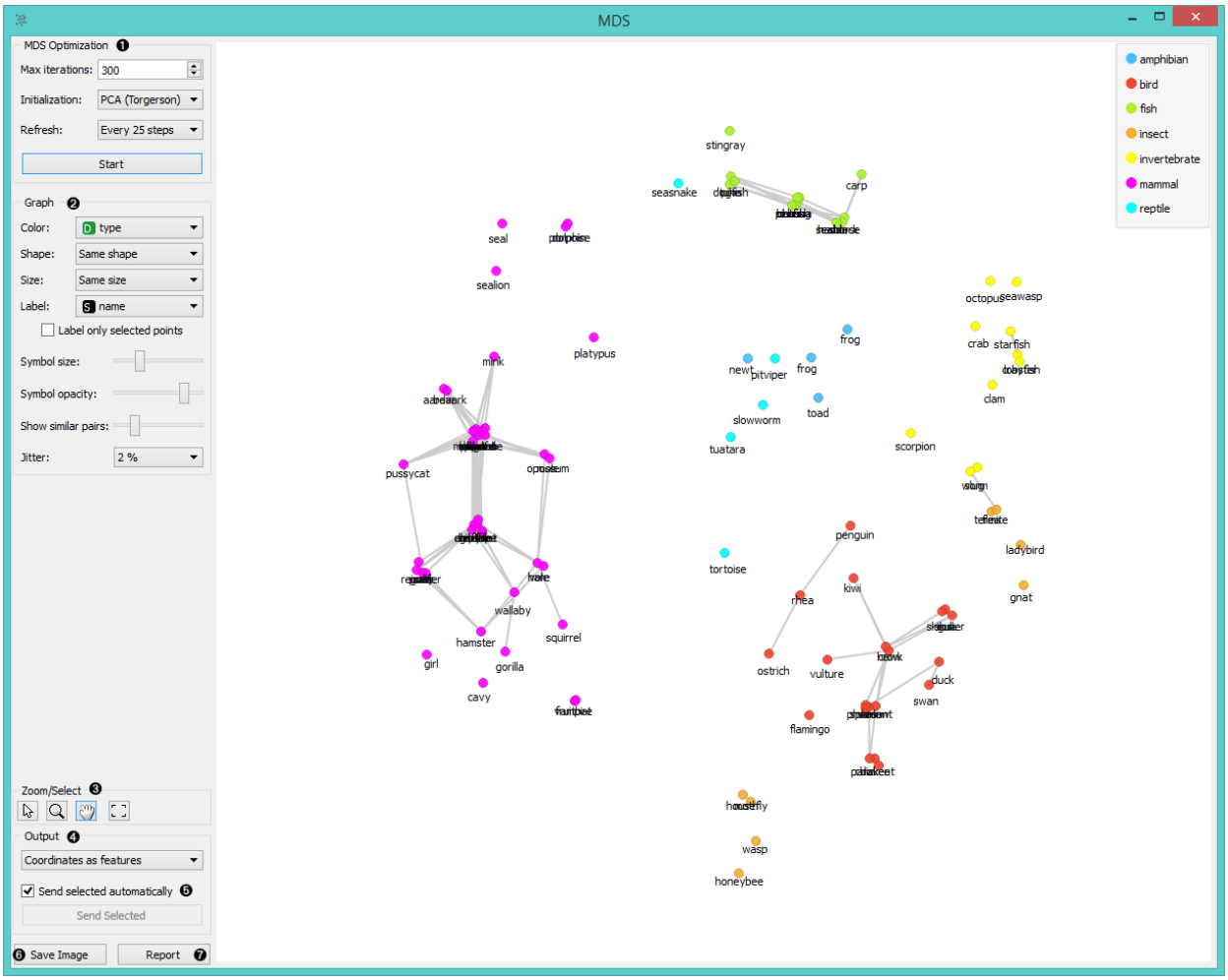

Select Columns
Available Variables
Filter
Up
>
Down
<
Up
Features
sepal length
sepal width
petal length
petal width
Target Variable
iris
Meta Attributes
Automatically
Distributions
Variable
sepal length
sepal width
petal length
petal width
Cluster
iris
Precision
2
50
Bin continuous variables into 10 bins
Group by
iris
Show relative frequencies
Show probabilities:
(None)
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica
Frequency
50
40
30
20
10
0
C1
C2
C3
Cluster

Description

Multidimensional scaling is a technique which finds a low-dimensional (in our case a two-dimensional) projection of points, where it tries to fit distances between points as well as possible. The perfect fit is typically impossible to obtain since the data is high-dimensional or the distances are not Euclidean.

In the input, the widget needs either a data set or a matrix of distances. When visualizing distances between rows, you can also adjust the color of the points, change their shape, mark them, and output them upon selection.

The algorithm iteratively moves the points around in a kind of a simulation of a physical model: if two points are too close to each other (or too far away), there is a force pushing them apart (or together). The change of the point's position at each time interval corresponds to the sum of forces acting on it.



1. The widget redraws the projection during optimization. Optimization is run automatically in the beginning and later by pushing *Start*.

    - **Max iterations**: The optimization stops either when the projection changes only minimally at the last iteration or when a maximum number of iterations has been reached.
    - **Initialization**: PCA (Torgerson) positions the initial points along principal coordinate axes. *Random* sets the initial points to a random position and then readjusts them.
    - **Refresh**: Set how often you want to refresh the visualization. It can be at *Every iteration*, *Every 5/10/25/50 steps* or never (*None*). Setting a lower refresh interval makes the animation more visually appealing, but

can be slow if the number of points is high.

2. Defines how the points are visualized. These options are available only when visalizing distances between rows (selected in the *Distances* widget).
   - **Color**: Color of points by attribute (gray for continuous, colored for discrete).
   - **Shape**: Shape of points by attribute (only for discrete).
   - **Size**: Set the size of points (*Same size* or select an attribute) or let the size depend on the value of the continuous attribute the point represents (Stress).
   - **Label**: Discrete attributes can serve as a label.
   - **Symbol size**: Adjust the size of the dots.
   - **Symbol opacity**: Adjust the transparency level of the dots.
   - **Show similar pairs**: Adjust the strength of network lines.
   - **Jitter**: Set jittering to prevent the dots from overlapping.
3. Adjust the graph with *Zoom/Select*. The arrow enables you to select data instances. The magnifying glass enables zooming, which can be also done by scrolling in and out. The hand allows you to move the graph around. The rectangle readjusts the graph proportionally.
4. Select the desired output:
   - **Original features only** (input data set)
   - **Coordinates only** (MDS coordinates)
   - **Coordinates as features** (input data set + MDS coordinates as regular attributes)
   - **Coordinates as meta attributes** (input data set + MDS coordinates as meta attributes)
5. Sending the instances can be automatic if *Send selected automatically* is ticked. Alternatively, click *Send selected*.
6. **Save Image** allows you to save the created image either as .svg or .png file to your device.
7. Produce a report.

The MDS graph performs many of the functions of the Visualizations widget. It is in many respects similar to the *Scatter Plot* widget, so we recommend reading that widget's description as well.

Example

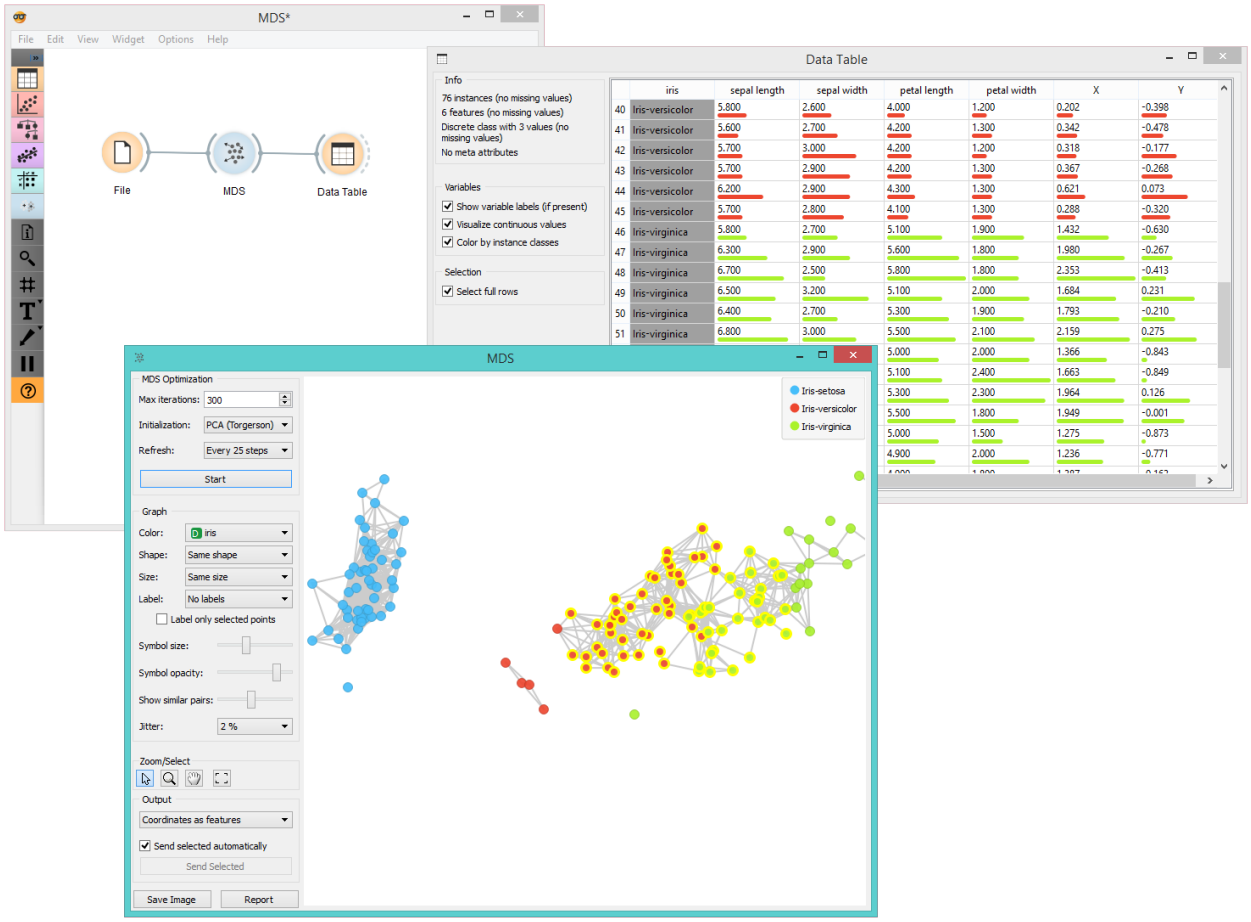
The above graphs were drawn using the following simple schema. We used the *iris.tab* data set. Using the *Distances* widget we input the distance matrix into the **MDS** widget, where we see the *Iris* data displayed in a 2-dimensional plane. We can see the appended coordinates in the *Data Table* widget.

References

Wickelmaier, F. (2003). An Introduction to MDS. Sound Quality Research Unit, Aalborg University. Available here.

Manifold Learning

Nonlinear dimensionality reduction.


MDS*
File Edit View Widget Options Help
File
MDS
Data Table
Data Table
Info
76 instances (no missing values)
6 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
iris sepal length sepal width petal length petal width X Y
40 Iris-versicolor 5.800 2.600 4.000 1.200 0.202 -0.398
41 Iris-versicolor 5.600 2.700 4.200 1.300 0.342 -0.478
42 Iris-versicolor 5.700 3.000 4.200 1.200 0.318 -0.177
43 Iris-versicolor 5.700 2.900 4.200 1.300 0.367 -0.268
44 Iris-versicolor 6.200 2.900 4.300 1.300 0.621 0.073
45 Iris-versicolor 5.700 2.800 4.100 1.300 0.288 -0.320
46 Iris-virginica 5.800 2.700 5.100 1.900 1.432 -0.630
47 Iris-virginica 6.300 2.900 5.600 1.800 1.980 -0.267
48 Iris-virginica 6.700 2.500 5.800 1.800 2.353 -0.413
49 Iris-virginica 6.500 3.200 5.100 2.000 1.684 0.231
50 Iris-virginica 6.400 2.700 5.300 1.900 1.793 -0.210
51 Iris-virginica 6.800 3.000 5.500 2.100 2.159 0.275
5.000 2.000 1.366 -0.843
5.100 2.400 1.663 -0.849
5.300 2.300 1.964 0.126
5.500 1.800 1.949 -0.001
5.000 1.500 1.275 -0.873
4.900 2.000 1.236 -0.771
MDS
MDS Optimization
Max iterations: 300
Initialization: PCA (Torgerson)
Refresh: Every 25 steps
Start
Graph
Color: iris
Shape: Same shape
Size: Same size
Label: No labels
Label only selected points
Symbol size:
Symbol opacity:
Show similar pairs:
Jitter: 2 %
Zoom/Select
Output
Coordinates as features
Send selected automatically
Send Selected
Save Image
Report
Iris-setosa
Iris-versicolor
Iris-virginica

Signals

Inputs:

- **Data**

  A data set

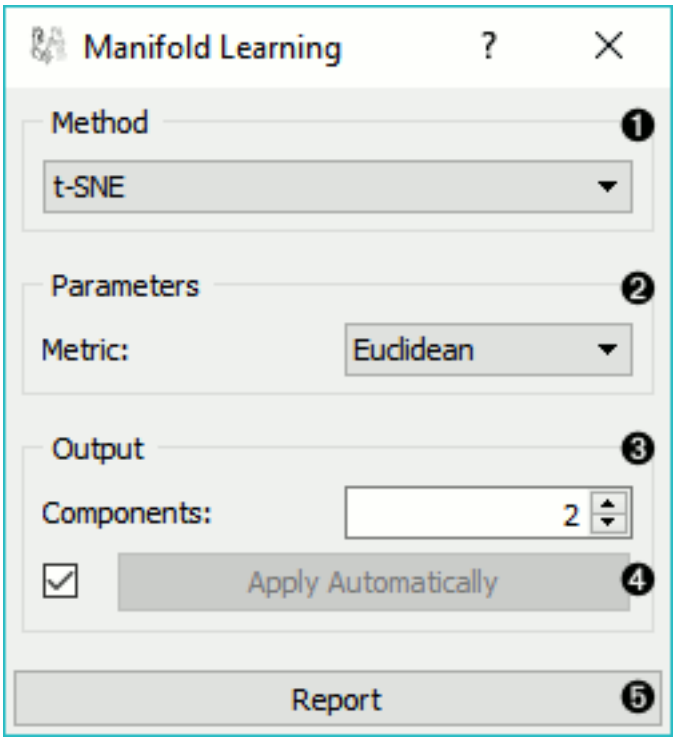
Outputs:

- **Transformed Data**

  A data set with new, reduced coordinates.

Description

Manifold Learning is a technique which finds a non-linear manifold within the higher-dimensional space. The widget then outputs new coordinates which correspond to a two-dimensional space. Such data can be later visualized with *Scatter Plot* or other visualization widgets.



1. Method for manifold learning:
   - t-SNE
   - MDS, see also *MDS widget*
   - Isomap
   - Locally Linear Embedding
   - Spectral Embedding
2. Set parameters for the method:
   - **t-SNE (distance measures):**

  - *Euclidean* distance
  - *Manhattan*
  - *Chebyshev*
  - *Jaccard*
  - *Mahalanobis*
  - *Cosine*
- **MDS (iterations and initialization):**
  - *max interations*: maximum number of optimization interations
  - *initialization*: method for initialization of the algorithm (PCA or random)
- **Isomap:**
  - number of *neighbors*
- **Locally Linear Embedding:**
  - ***method*:**
    - standard
    - modified
    - hessian eigenmap
    - local
  - number of *neighbors*
  - *max iterations*
- **Spectral Embedding:**
  - ***affinity*:**
    - nearest neighbors
    - RFB kernel

3. Output: the number of reduced features (components).
4. If *Apply automatically* is ticked, changes will be propagated automatically. Alternatively, click *Apply*.
5. Produce a report.

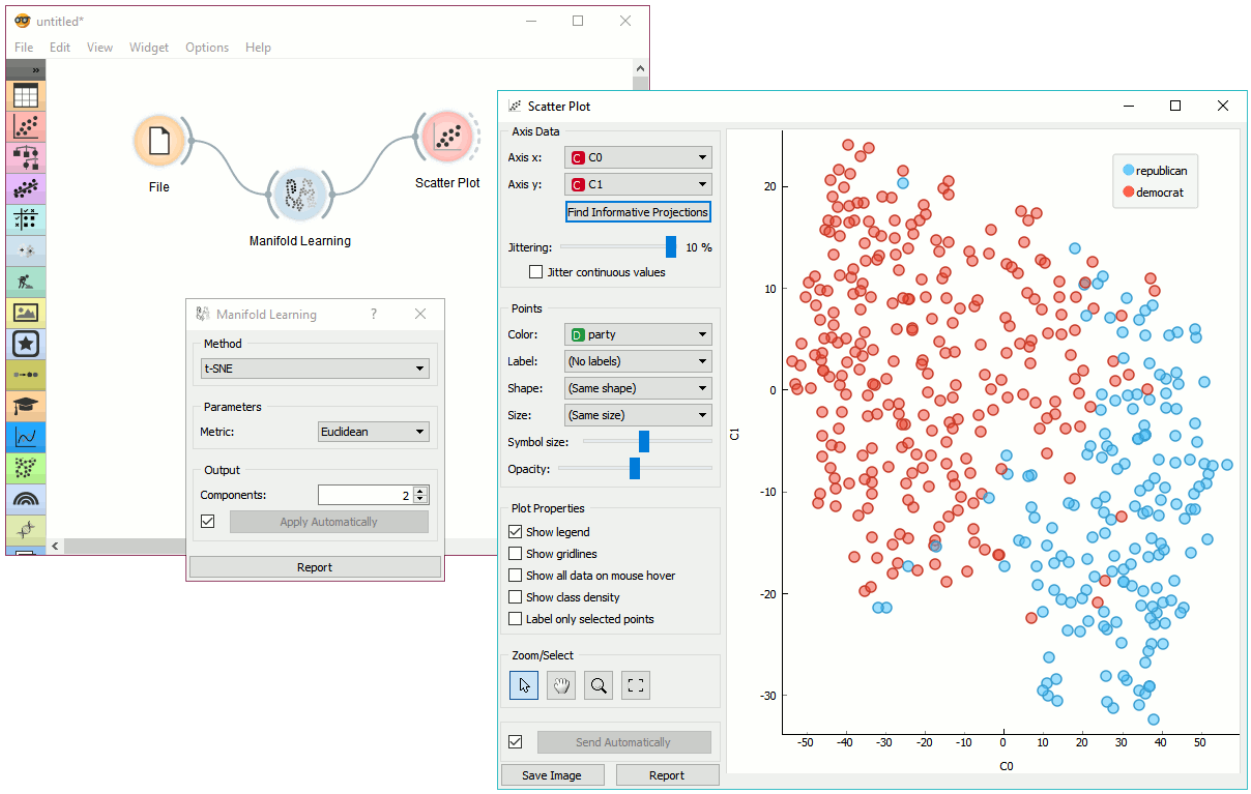
Manifold Learning widget produces different embeddings for high-dimensional data.

... figure:: images/collage-manifold.png

From left to right, top to bottom: t-SNE, MDS, Isomap, Locally Linear Embedding and Spectral Embedding.

Example

Manifold Learning widget transforms high-dimensional data into a lower dimensional approximation. This makes it great for visualizing data sets with many features. We used *voting.tab* to map 16-dimensional data onto a 2D graph. Then we used *Scatter Plot* to plot the embeddings.



Evaluation

Calibration Plot

Shows the match between classifiers' probability predictions and actual class probabilities.

Signals

Inputs:

- **Evaluation Results**

  Results of testing classification algorithms.

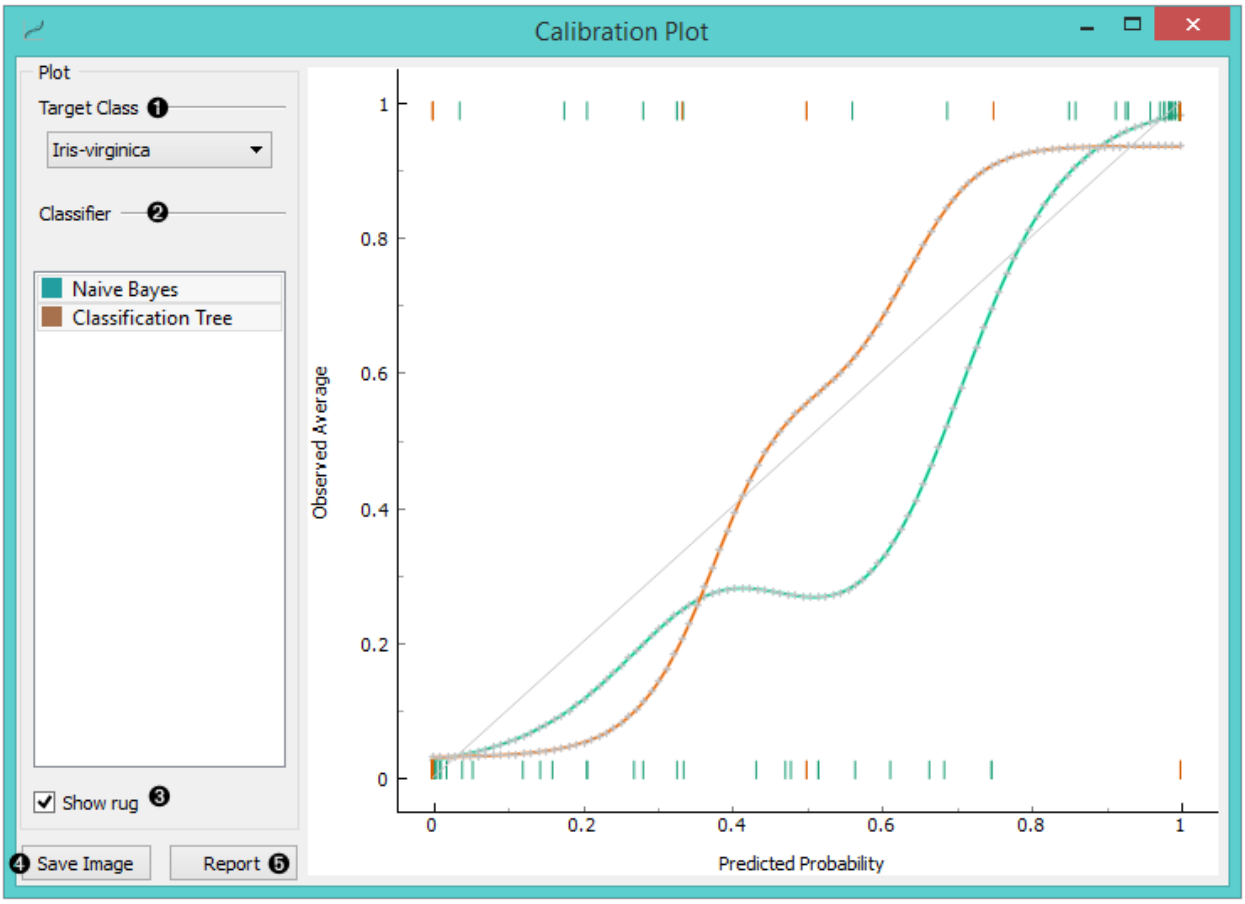
Outputs:

- None

Description

The Calibration Plot plots class probabilities against those predicted by the classifier(s).

1. Select the desired target class from the drop down menu.

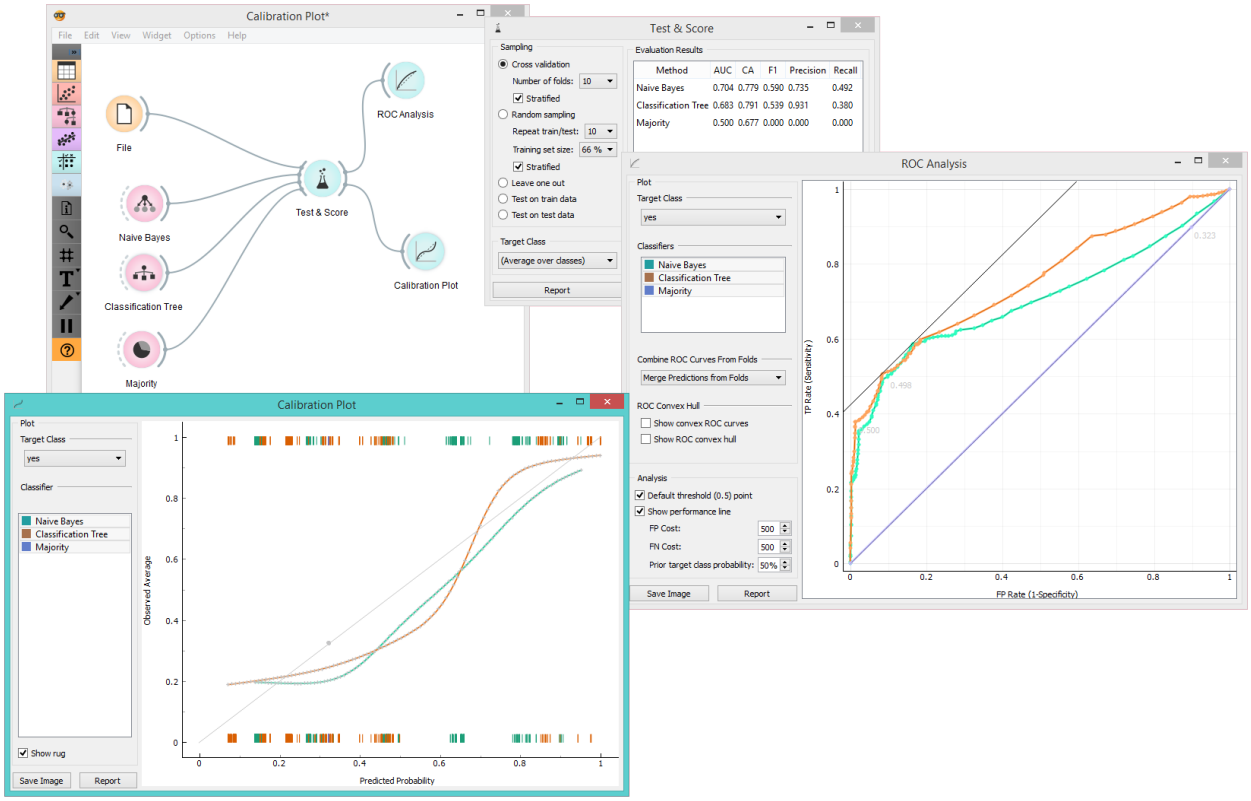

Calibration Plot
Plot
Target Class
Iris-virginica
Classifier
Naive Bayes
Classification Tree
Show rug
Save Image
Report
Observed Average
Predicted Probability
0
0.2
0.4
0.6
0.8
1

2. Choose which classifiers to plot. The diagonal represents optimal behaviour; the closer the classifier's curve gets, the more accurate its prediction probabilities are. Thus we would use this widget to see whether a classifier is overly optimistic (gives predominantly positive results) or pesimitistic (gives predominantly negative results).
3. If *Show rug* is enabled, ticks are displayed at the bottom and the top of the graph, which represent negative and positive examples respectively. Their position corresponds to the classifier's probability prediction and the color shows the classifier. At the bottom of the graph, the points to the left are those which are (correctly) assigned a low probability of the target class, and those to the right are incorrectly assigned high probabilities. At the top of the graph, the instances to the right are correctly assigned high probabilities and vice versa.
4. Press *Save Image* if you want to save the created image to your computer in a .svg or .png format.
5. Produce a report.

Example

At the moment, the only widget which gives the right type of signal needed by the **Calibration Plot** is *Test&Score*. The Calibration Plot will hence always follow Test&Score and, since it has no outputs, no other widgets follow it.

Here is a typical example, where we compare three classifiers (namely *Naive Bayes*, *Tree* and *Constant*) and input them into *Test&Score*. We used the *Titanic* data set. Test&Score then displays evaluation results for each classifier. Then we draw **Calibration Plot** and *ROC Analysis* widgets from Test&Score to further analyze the performance of classifiers. **Calibration Plot** enables you to see prediction accuracy of class probabilities in a plot.



Confusion Matrix

Shows proportions between the predicted and actual class.

Signals

Inputs:

- **Evaluation results**

  Results of testing the algorithms; typically from **Test Learners**

Outputs:

- **Selected Data**

  A data subset from the selected cells in the confusion matrix.

Description

The Confusion Matrix gives the number/proportion of instances between the predicted and actual class. The selection of the elements in the matrix feeds the corresponding instances into the output signal. This way, one can observe which specific instances were misclassified and how.

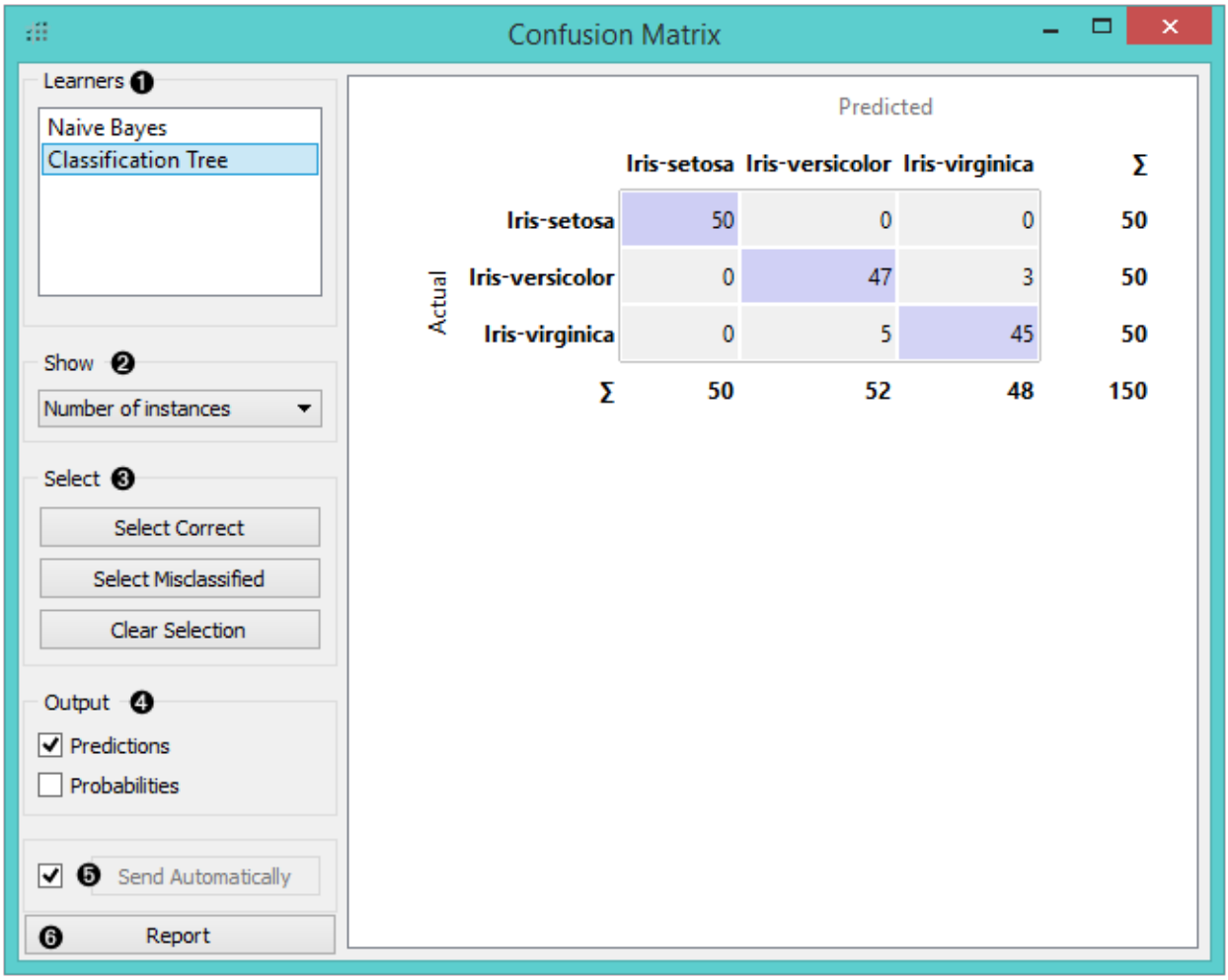
The widget usually gets the evaluation results from *Test & Score*; an example of the schema is shown below.

1. When evaluation results contain data on multiple learning algorithms, we have to choose one in the *Learners* box.

The snapshot shows the confusion matrix for *Tree* and *Naive Bayesian* models trained and tested on the *iris* data. The righthand side of the widget contains the matrix for the naive Bayesian model (since this model is selected on the left). Each row corresponds to a correct class, while columns represent the predicted classes. For instance, four instances of *Iris-versicolor* were misclassified as *Iris-virginica*. The rightmost column gives the number of instances from each class (there are 50 irises of each of the three classes) and the bottom row gives the number of instances classified into each class (e.g., 48 instances were classified into virginica).

2. In *Show*, we select what data we would like to see in the matrix.
   - **Number of instances** shows correctly and incorrectly classified instances numerically.
   - **Proportions of predicted** shows how many instances classified as, say, *Iris-versicolor* are in which true class; in the table we can read the 0% of them are actually setosae, 88.5% of those classified as versicolor are versicolors, and 7.7% are virginicae.
   - **Proportions of actual** shows the opposite relation: of all true versicolors, 92% were classified as versicolors and 8% as virginicae.
3. In *Select*, you can choose the desired output.
   - **Correct** sends all correctly classified instances to the output by selecting the diagonal of the matrix.
   - **Misclassified** selects the misclassified instances.
   - **None** annuls the selection.

     As mentioned before, one can also select individual cells of the table to select specific kinds of misclassified instances (e.g. the versicolors classified as virginicae).
4. When sending selected instances, the widget can add new attributes, such as predicted classes or their probabilities, if the corresponding options *Predictions* and/or *Probabilities* are checked.



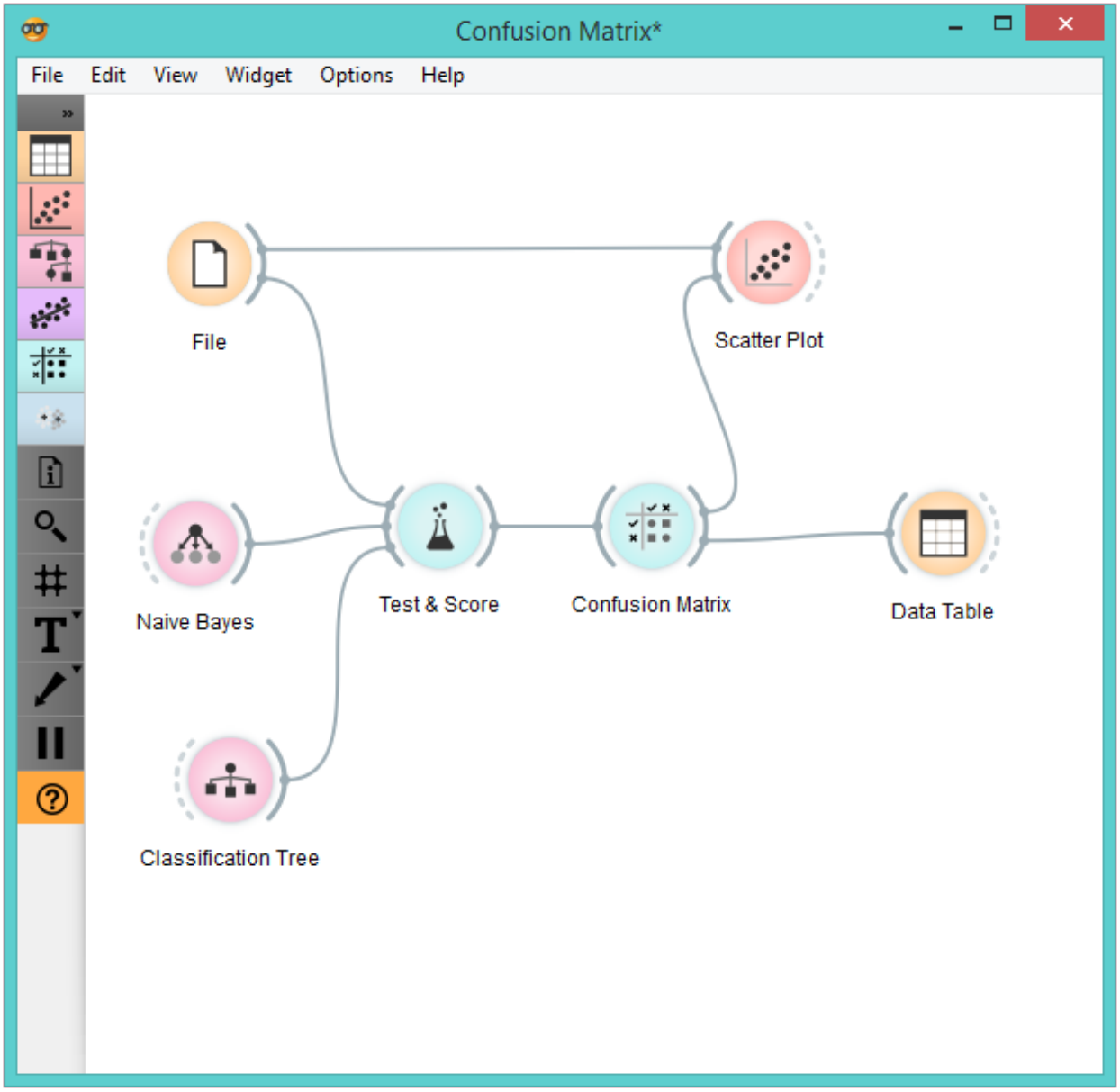
Predicted

Actual	Iris-setosa	Iris-versicolor	Iris-virginica	Σ
Iris-setosa	100.0 %	0.0 %	0.0 %	50
Iris-versicolor	0.0 %	88.7 %	6.4 %	50
Iris-virginica	0.0 %	11.3 %	93.6 %	50
Σ	50	53	47	150

5. The widget outputs every change if *Send Automatically* is ticked. If not, the user will need to click *Send Selected* to commit the changes.
6. Produce a report.

Example

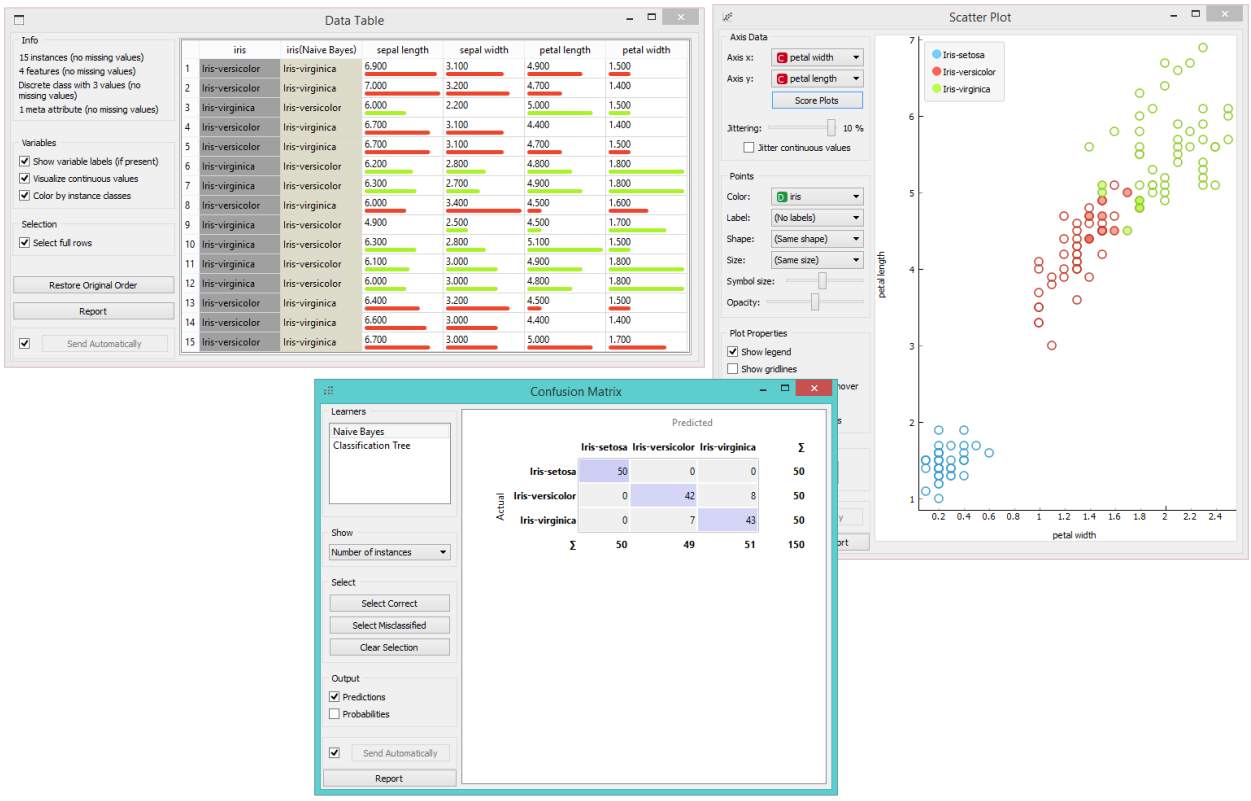
The following workflow demonstrates what this widget can be used for.



Test & Score gets the data from *File* and two learning algorithms from *Naive Bayes* and *Tree*. It performs cross-validation or some other train-and-test procedures to get class predictions by both algorithms for all (or some) data instances. The test results are fed into the **Confusion Matrix**, where we can observe how many instances were misclassified and in which way.

In the output, we used *Data Table* to show the instances we selected in the confusion matrix. If we, for instance, click *Misclassified*, the table will contain all instances which were misclassified by the selected method.

The *Scatterplot* gets two sets of data. From the *File* widget it gets the complete data, while the confusion matrix sends only the selected data, misclassifications for instance. The scatter plot will show all the data, with bold symbols representing the selected data.



Lift Curve

Measures the performance of a chosen classifier against a random classifier.

Signals

Inputs:

- **Evaluation Results**

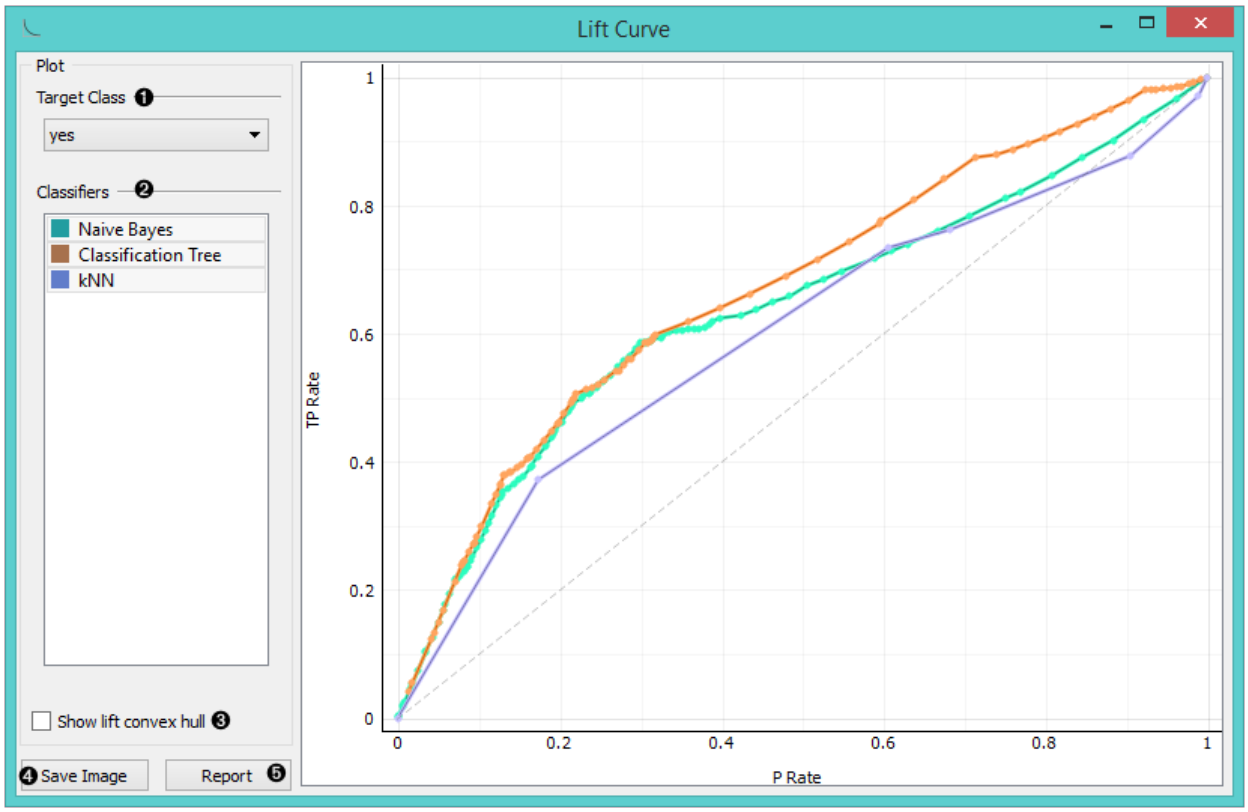
  Results of classifiers' tests on data.

Outputs:

- None

Description

The **Lift curve** shows the relation between the number of instances which were predicted positive and those that are indeed positive and thus measures the performance of a chosen classifier against a random classifier. The graph is constructed with the cumulative number of cases (in descending order of probability) on the x-axis and the cumulative number of true positives on the y-axis. Lift curve is often used in segmenting the population, e.g., plotting the number of responding customers against the number of all customers contacted. You can also determine the optimal classifier and its threshold from the graph.



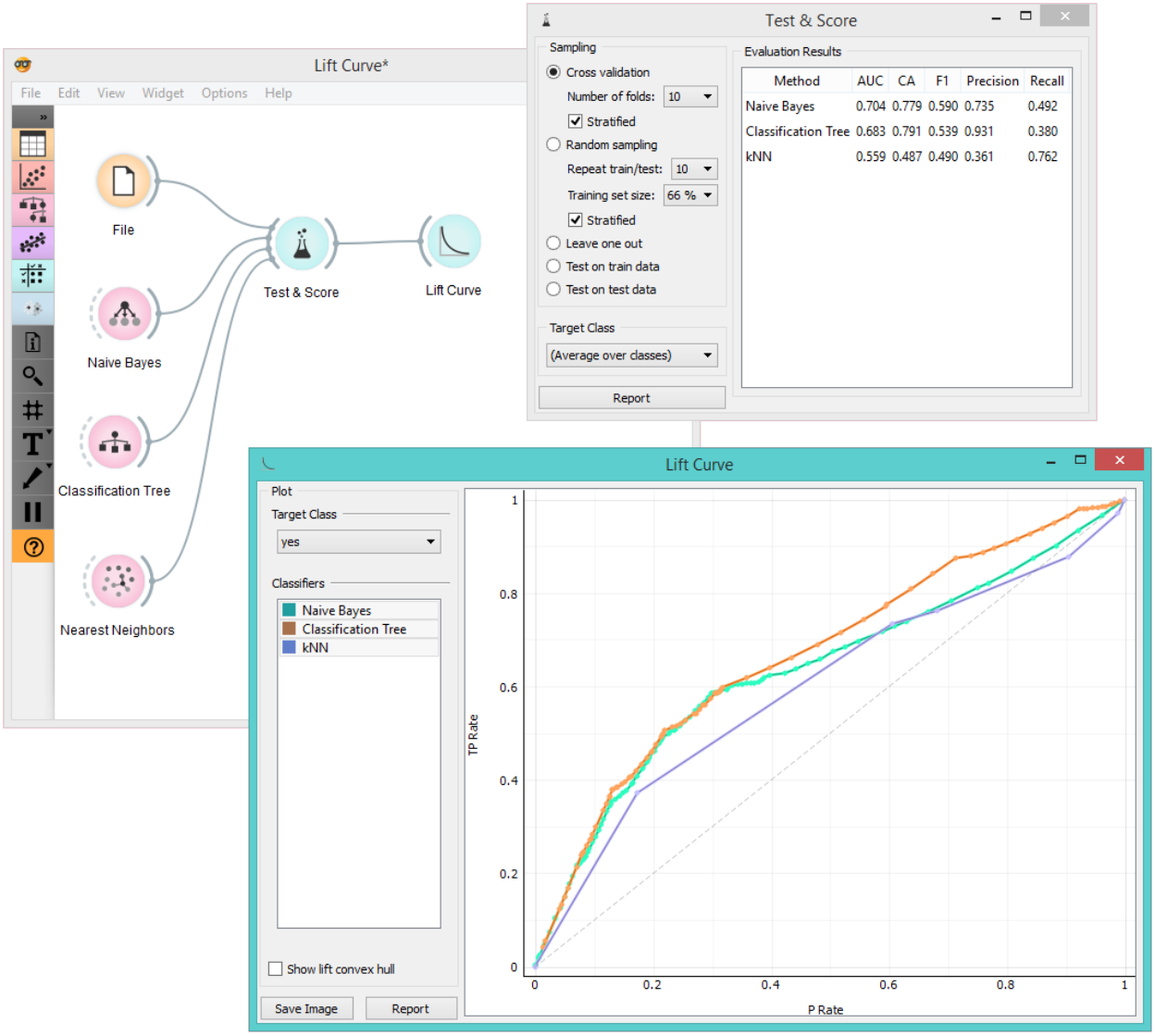
1. Choose the desired *Target class*. The default class is chosen alphabetically.
2. If test results contain more than one classifier, the user can choose which curves she or he wants to see plotted. Click on a classifier to select or deselect the curve.
3. *Show lift convex hull* plots a convex hull over lift curves for all classifiers (yellow curve). The curve shows the optimal classifier (or combination thereof) for each desired TP/P rate.
4. Press *Save Image* if you want to save the created image to your computer in a .svg or .png format.
5. Produce a report.
6. 2-D pane with **P rate** (population) as x-axis and **TP rate** (true positives) as a y-axis. The diagonal line represents the behaviour of a random classifier. Click and drag to move the pane and scroll in or out to zoom. Click on the "*A*" sign at the bottom left corner to realign the pane.

Note: The perfect classifier would have a steep slope towards 1 until all classes are guessed correctly and then run straight along 1 on y-axis to (1,1).

Example

At the moment, the only widget which gives the right type of the signal needed by the **Lift Curve** is *Test&Score*.

In the example below, we try to see the prediction quality for the class 'survived' on the *Titanic* data set. We compared three different classifiers in the Test Learners widget and sent them to Lift Curve to see their performance against a random model. We see the *Tree* classifier is the best out of the three, since it best aligns with *lift convex hull*. We also see that its performance is the best for the first 30% of the population (in order of descending probability), which we can set as the threshold for optimal classification.



References

Handouts of the University of Notre Dame on Data Mining - Lift Curve. Available here.

Predictions

Shows models' predictions on the data.

Signals

Inputs

- **Data**

  A data set.

- **Predictors**

  Predictors to be used on the data.

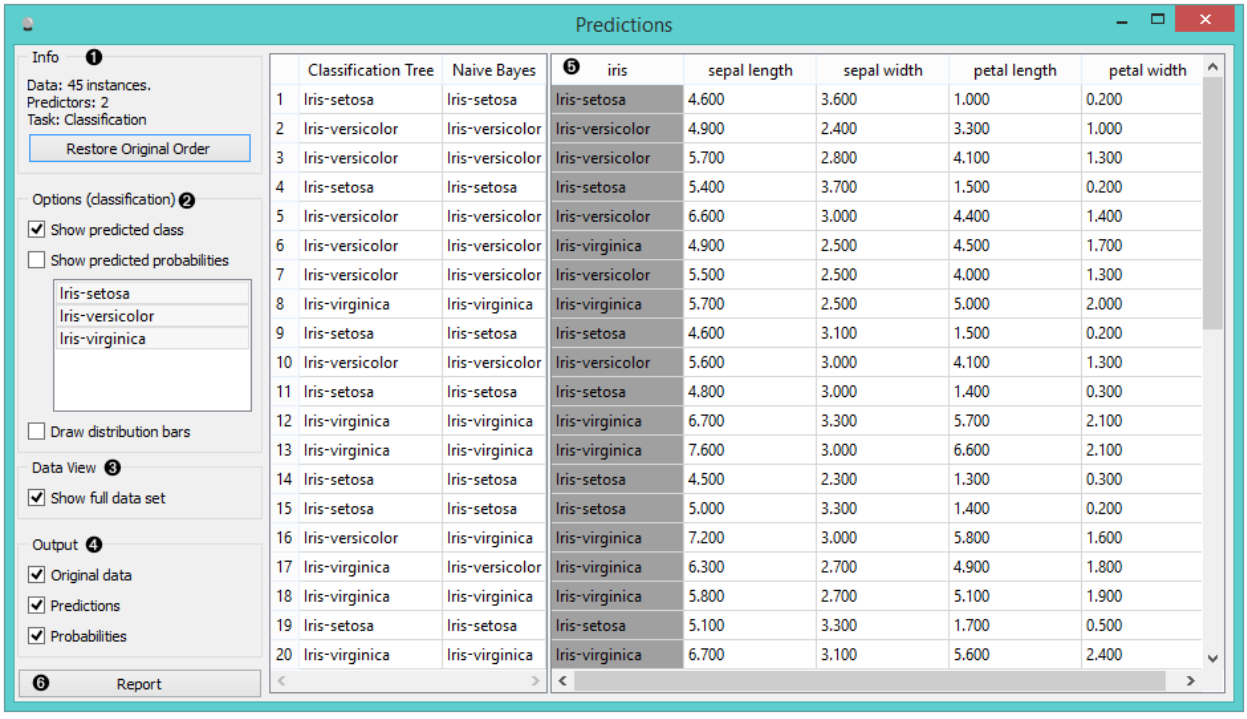
Outputs

- **Predictions**

  Original data with added predictions.

Description

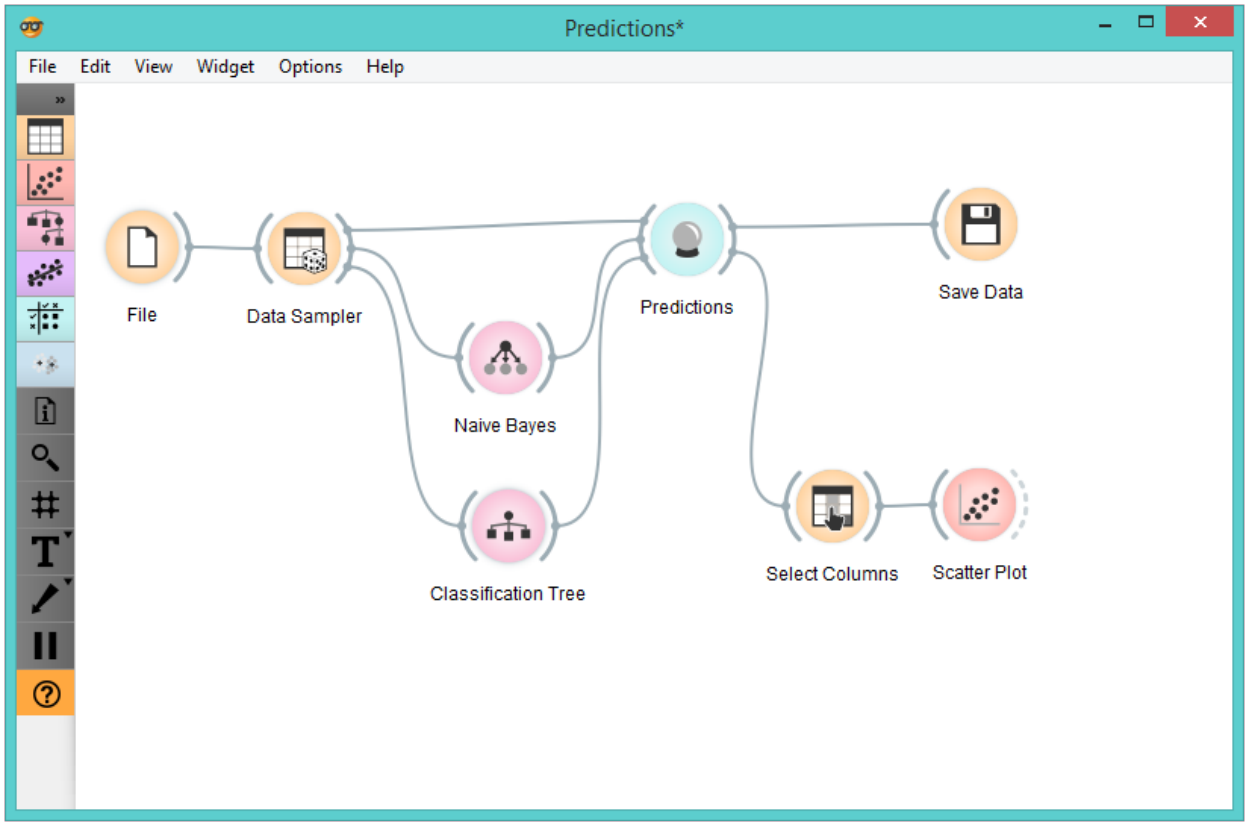
The widget receives a data set and one or more predictors (classifiers, not learning algorithms - see the example below). It outputs the data and the predictions.



1. Information on the input
2. The user can select the options for classification. If *Show predicted class* is ticked, the appended data table provides information on predicted class. If *Show predicted probabilities* is ticked, the appended data table provides information on probabilities predicted by the classifiers. The user can also select the predicted class he or she wants displayed in the appended data table. The option *Draw distribution bars* provides a nice visualization of the predictions.
3. By ticking the *Show full data set*, the user can append the entire data table to the *Predictions* widget.
4. Select the desired output.
5. The appended data table
6. Produce a report.

Despite its simplicity, the widget allows for quite an interesting analysis of decisions of predictive models; there is a simple demonstration at the bottom of the page. *Confusion Matrix* is a related widget and although many things can be done with any of them, there are tasks for which one of them might be much more convenient than the other. The output of the widget is another data set, where predictions are appended as new meta attributes. You can select which features you wish to output (original data, predictions, probabilities). The resulting data set can be appended to the widget, but you can still choose to display it in a separate data table.

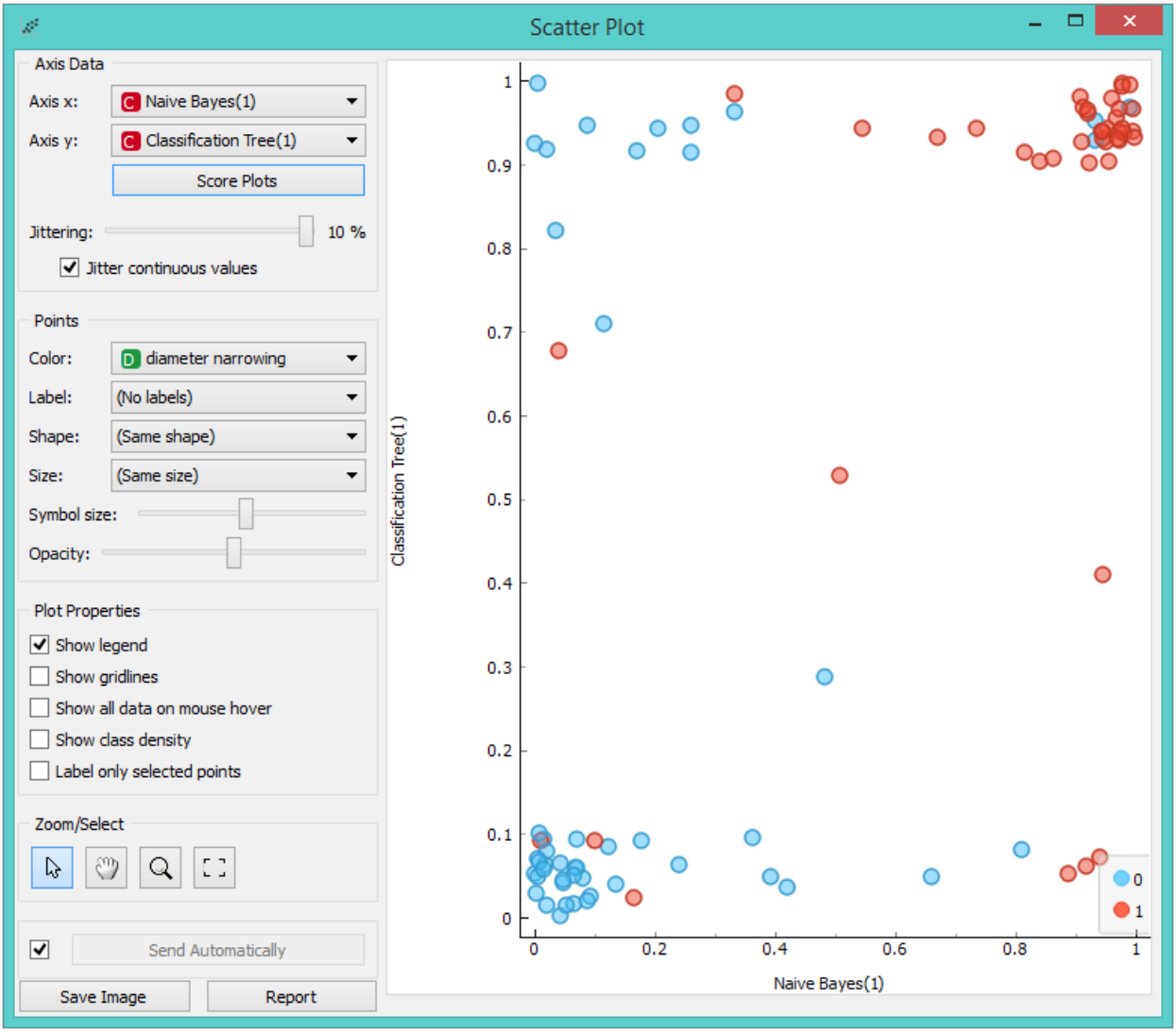
Example



We randomly split the *heart-disease* data into two subsets. The larger subset, containing 70 % of data instances, is sent to *Naive Bayes* and *Tree*, so they can produce the corresponding model. Models are then sent into **Predictions**, among with the remaining 30 % of the data. Predictions shows how these examples are classified.

To save the predictions, we simply attach the *Save* widget to **Predictions**. The final file is a data table and can be saved as in a .tab or .tsv format.

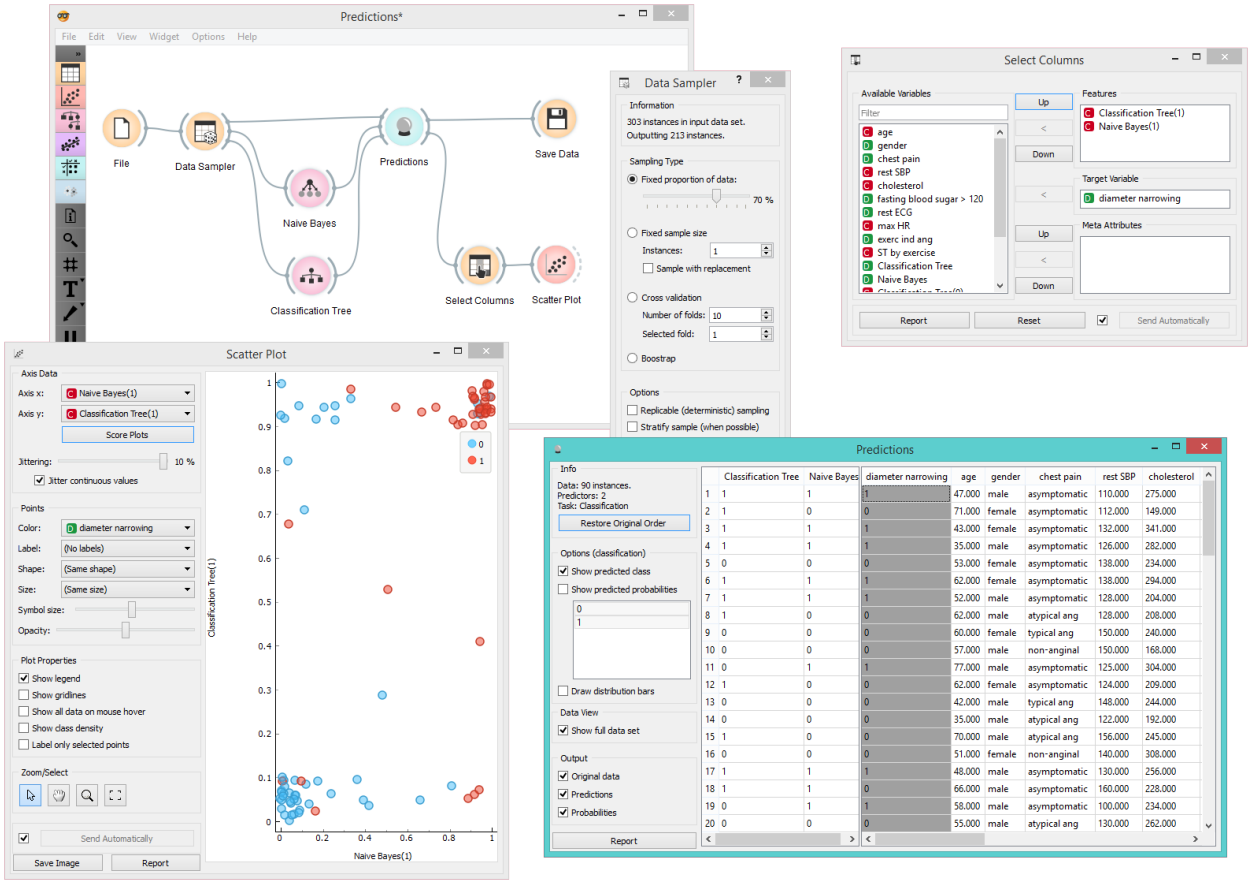
Finally, we can analyze the models' predictions. For that, we first take *Select Columns* with which we move the meta attributes with probability predictions to features. The transformed data is then given to the *Scatterplot*, which we set to use the attributes with probabilities as the x and y axes, while the class is (already by default) used to color the data points.



To get the above plot, we selected *Jitter continuous values*, since the decision tree gives just a few distinct probabilities. The blue points in the bottom left corner represent the people with no diameter narrowing, which were correctly classified by both models. The upper right red points represent the patients with narrowed vessels, which were correctly classified by both.

Note that this analysis is done on a rather small sample, so these conclusions may be ungrounded. Here is the entire workflow:

Another example of using this widget is given in the documentation for the widget *Confusion Matrix*.


Predictions*
File
Data Sampler
Naive Bayes
Predictions
Save Data
Classification Tree
Select Columns
Scatter Plot
Data Sampler
Information
303 instances in input data set.
Outputting 213 instances.
Sampling Type
Fixed proportion of data:
70 %
Fixed sample size
Instances:
Sample with replacement
Cross validation
Number of folds:
Selected fold:
Boostrap
Options
Replicable (deterministic) sampling
Stratify sample (when possible)
Select Columns
Available Variables
Filter
age
gender
chest pain
rest SBP
cholesterol
fasting blood sugar > 120
rest ECG
max HR
exerc ind ang
ST by exercise
Classification Tree
Naive Bayes
Up
Down
Features
Classification Tree(1)
Naive Bayes(1)
Target Variable
diameter narrowing
Meta Attributes
Report
Reset
Send Automatically
Scatter Plot
Axis Data
Axis x:
Naive Bayes(1)
Axis y:
Classification Tree(1)
Score Plots
Jittering:
10 %
Jitter continuous values
Points
Color:
diameter narrowing
Label:
(No labels)
Shape:
(Same shape)
Size:
(Same size)
Symbol size:
Opacity:
Plot Properties
Show legend
Show gridlines
Show all data on mouse hover
Show class density
Label only selected points
Zoom/Select
Send Automatically
Save Image
Report
Classification Tree(1)
Naive Bayes(1)
Predictions
Info
Data: 90 instances.
Predictors: 2
Task: Classification
Restore Original Order
Options (classification)
Show predicted class
Show predicted probabilities
Draw distribution bars
Data View
Show full data set
Output
Original data
Predictions
Probabilities
Report
Classification Tree
Naive Bayes
diameter narrowing
age
gender
chest pain
rest SBP
cholesterol

ROC Analysis

Plots a true positive rate against a false positive rate of a test.

Signals

Inputs:

- **Evaluation Results**

  Results of classifiers' tests on data

Outputs:

- None

Description

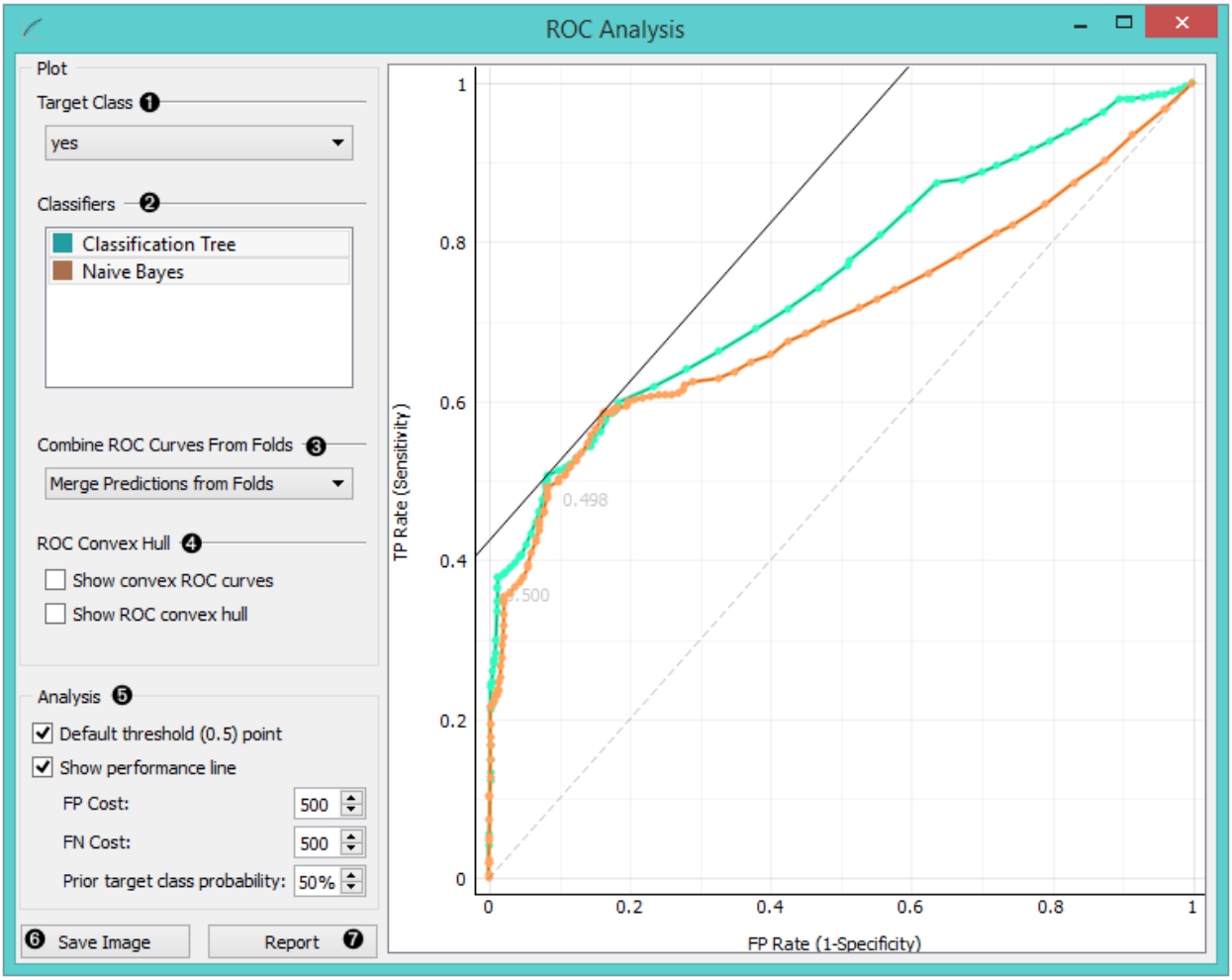
The widget shows ROC curves for the tested models and the corresponding convex hull. It serves as a mean of comparison between classification models. The curve plots a false positive rate on an x-axis (1-specificity; probability that target=1 when true value=0) against a true positive rate on a y-axis (sensitivity; probability that target=1 when true value=1). The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the classifier. Given the costs of false positives and false negatives, the widget can also determine the optimal classifier and threshold.

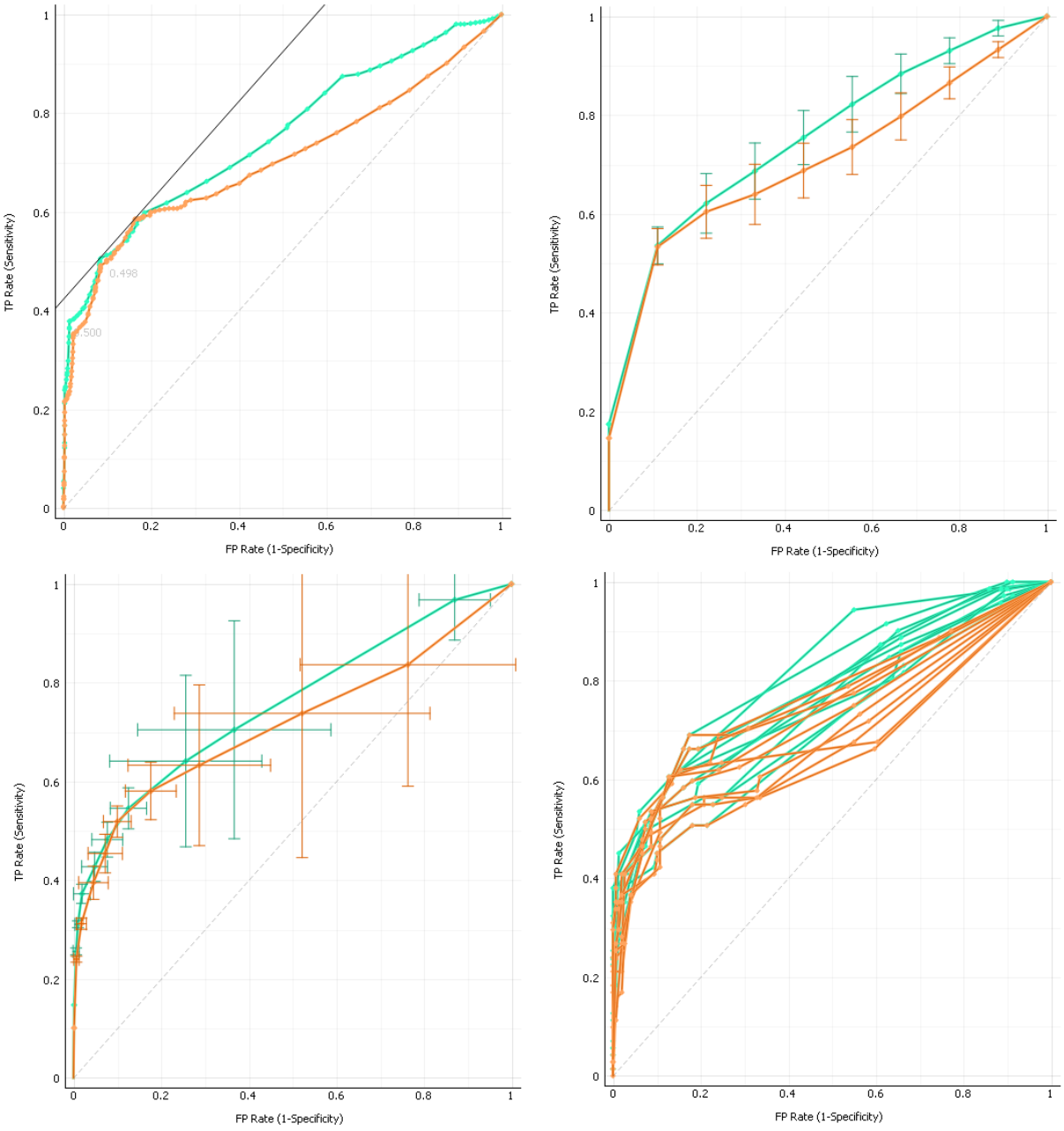
1. Choose the desired *Target Class*. The default class is chosen alphabetically.
2. If test results contain more than one classifier, the user can choose which curves she or he wants to see plotted. Click on a classifier to select or deselect it.
3. When the data comes from multiple iterations of training and testing, such as k-fold cross validation, the results can be (and usually are) averaged.

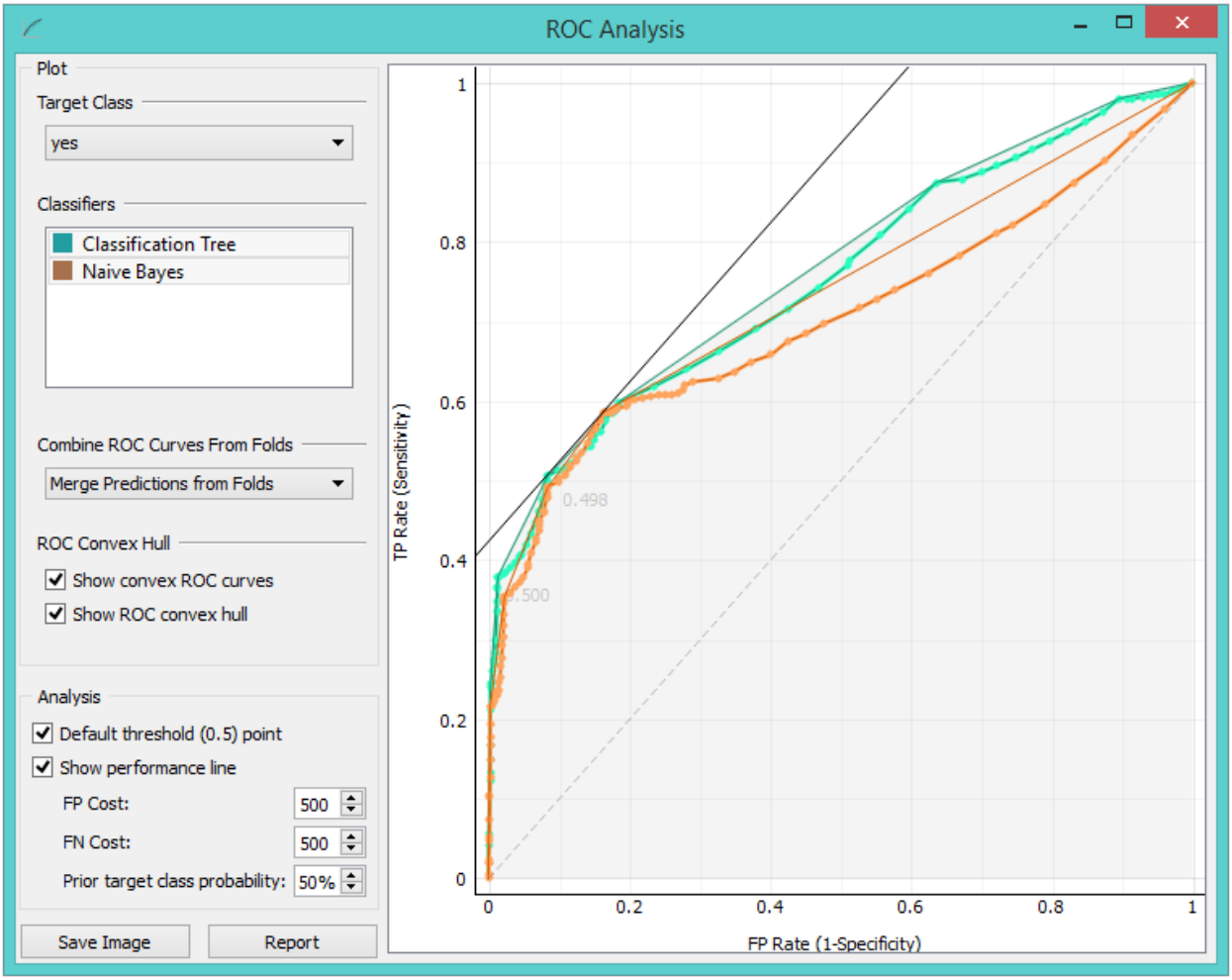
   The averaging options are:

   - **Merge predictions from folds** (top left), which treats all the test data as if they came from a single iteration
   - **Mean TP rate** (top right) averages the curves vertically, showing the corresponding confidence intervals
   - **Mean TP and FP at threshold** (bottom left) traverses over threshold, averages the positions of curves and shows horizontal and vertical confidence intervals
   - **Show individual curves** (bottom right) does not average but prints all the curves instead
4. Option *Show convex ROC curves* refers to convex curves over each individual classifier (the thin lines positioned over curves). *Show ROC convex hull* plots a convex hull combining all classifiers (the gray area below the curves). Plotting both types of convex curves makes sense since selecting a threshold in a concave part of the curve cannot yield optimal results, disregarding the cost matrix. Besides, it is possible to reach any point on the convex curve by combining the classifiers represented by the points on the border of the concave region.

   The diagonal dotted line represents the behaviour of a random classifier. The full diagonal line represents iso-performance. A black "*A*" symbol at the bottom of the graph proportionally readjusts the graph.


ROC Analysis
Plot
Target Class
yes
Classifiers
Classification Tree
Naive Bayes
Combine ROC Curves From Folds
Merge Predictions from Folds
ROC Convex Hull
Show convex ROC curves
Show ROC convex hull
Analysis
Default threshold (0.5) point
Show performance line
FP Cost:
500
FN Cost:
500
Prior target class probability:
50%
Save Image
Report
TP Rate (Sensitivity)
FP Rate (1-Specificity)
0.498
0.500
0
0.2
0.4
0.6
0.8
1


TP Rate (Sensitivity)
FP Rate (1-Specificity)
0.498
0.500

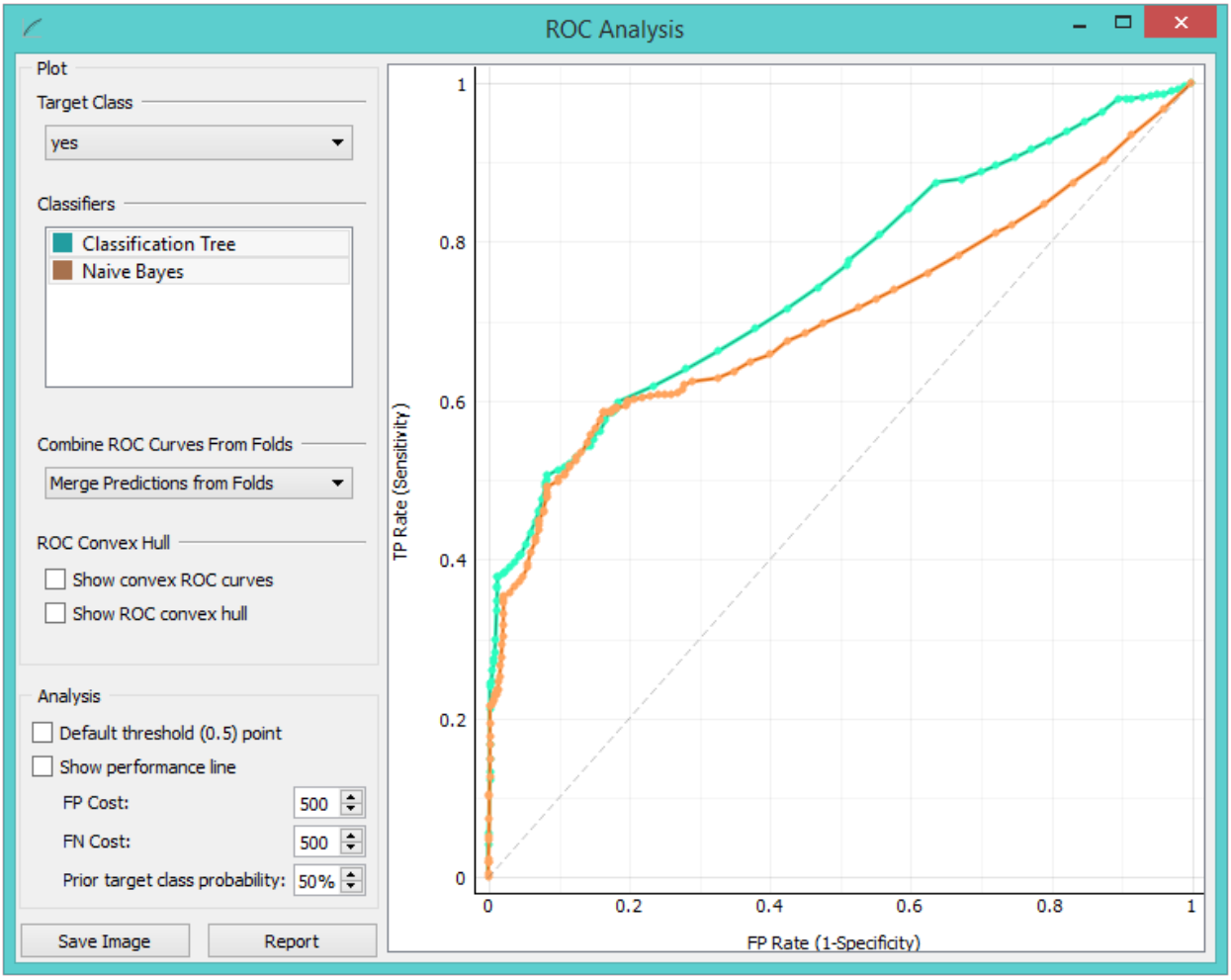

ROC Analysis
Plot
Target Class
yes
Classifiers
Classification Tree
Naive Bayes
Combine ROC Curves From Folds
Merge Predictions from Folds
ROC Convex Hull
Show convex ROC curves
Show ROC convex hull
Analysis
Default threshold (0.5) point
Show performance line
FP Cost:
500
FN Cost:
500
Prior target class probability:
50%
Save Image
Report
TP Rate (Sensitivity)
FP Rate (1-Specificity)
0.498
0.500
0
0.2
0.4
0.6
0.8
1

5. The final box is dedicated to the analysis of the curve. The user can specify the cost of false positives (FP) and false negatives (FN), and the prior target class probability.

   Default threshold (0.5) point shows the point on the ROC curve achieved by the classifier if it predicts the target class if its probability equals or exceeds 0.5.

   Show performance line shows iso-performance in the ROC space so that all the points on the line give the same profit/loss. The line further to the upper left is better than the one down and right. The direction of the line depends upon costs and probabilities. This gives a recipe for depicting the optimal threshold for the given costs: this is the point where the tangent with the given inclination touches the curve and it is marked in the plot. If we push the iso-performance higher or more to the left, the points on the iso-performance line cannot be reached by the learner. Going down or to the right, decreases the performance.

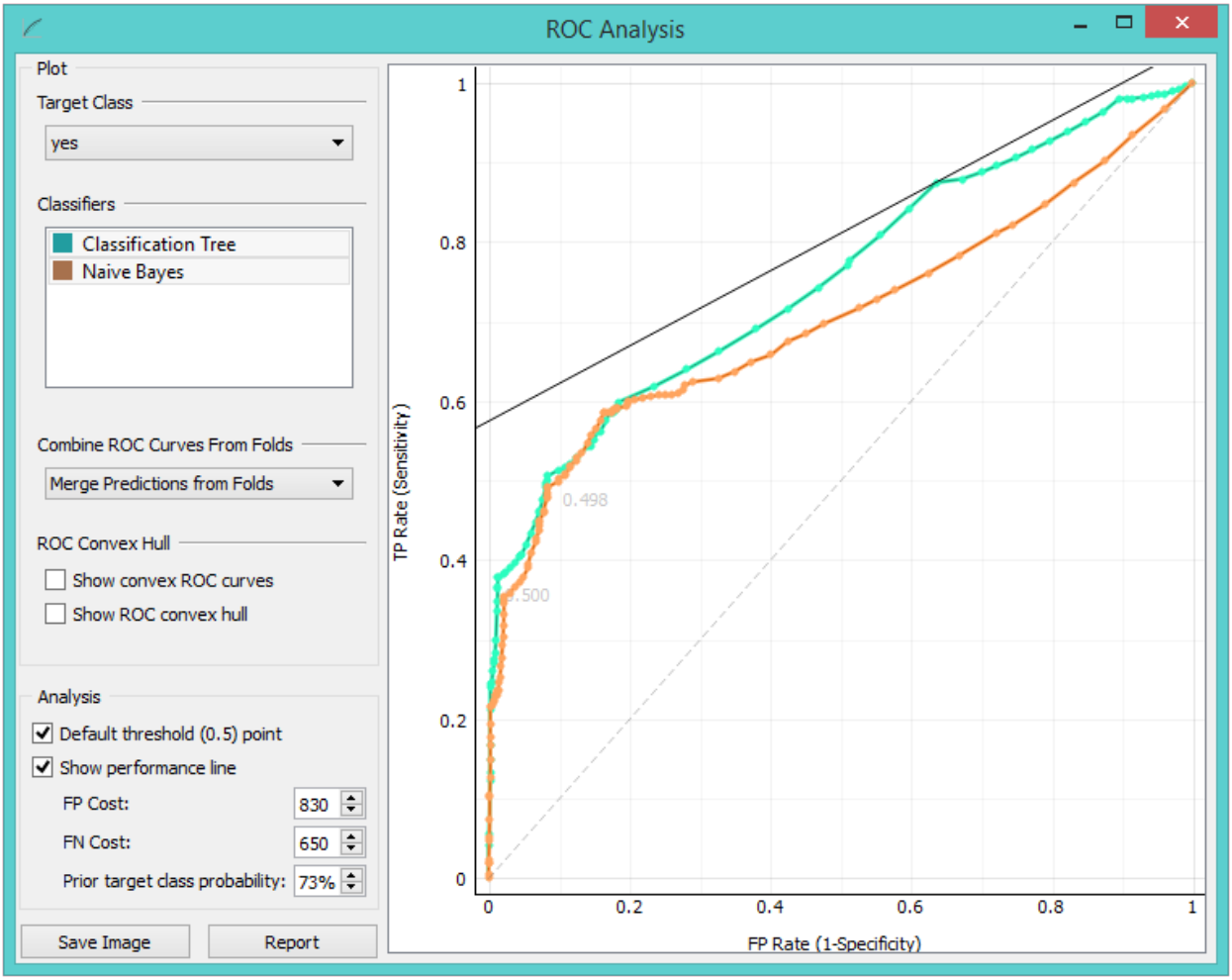
   The widget allows setting the costs from 1 to 1000. Units are not important, as are not the magnitudes. What matters is the relation between the two costs, so setting them to 100 and 200 will give the same result as 400 and 800.



   Defaults: both costs equal (500), Prior target class probability 50% (from the data).

   False positive cost: 830, False negative cost 650, Prior target class probability 73%.

6. Press *Save Image* if you want to save the created image to your computer in a .svg or .png format.
7. Produce a report.


ROC Analysis
Plot
Target Class
yes
Classifiers
Classification Tree
Naive Bayes
Combine ROC Curves From Folds
Merge Predictions from Folds
ROC Convex Hull
Show convex ROC curves
Show ROC convex hull
Analysis
Default threshold (0.5) point
Show performance line
FP Cost:
830
FN Cost:
650
Prior target class probability:
73%
Save Image
Report
TP Rate (Sensitivity)
FP Rate (1-Specificity)
0.498
0.500
0
0.2
0.4
0.6
0.8
1

Example

At the moment, the only widget which gives the right type of signal needed by the **ROC Analysis** is *Test&Score*. Below, we compare two classifiers, namely *Tree* and *Naive Bayes*, in **Test&Score** and then compare their performance in **ROC Analysis**, *Life Curve* and *Calibration Plot*.

Test & Score

Tests learning algorithms on data.

Signals

Inputs

- **Data**

  Data for training and, if there is no separate test data set, also testing.

- **Test Data**

  Separate data for testing.

- **Learner**

  One or more learning algorithms.

Outputs

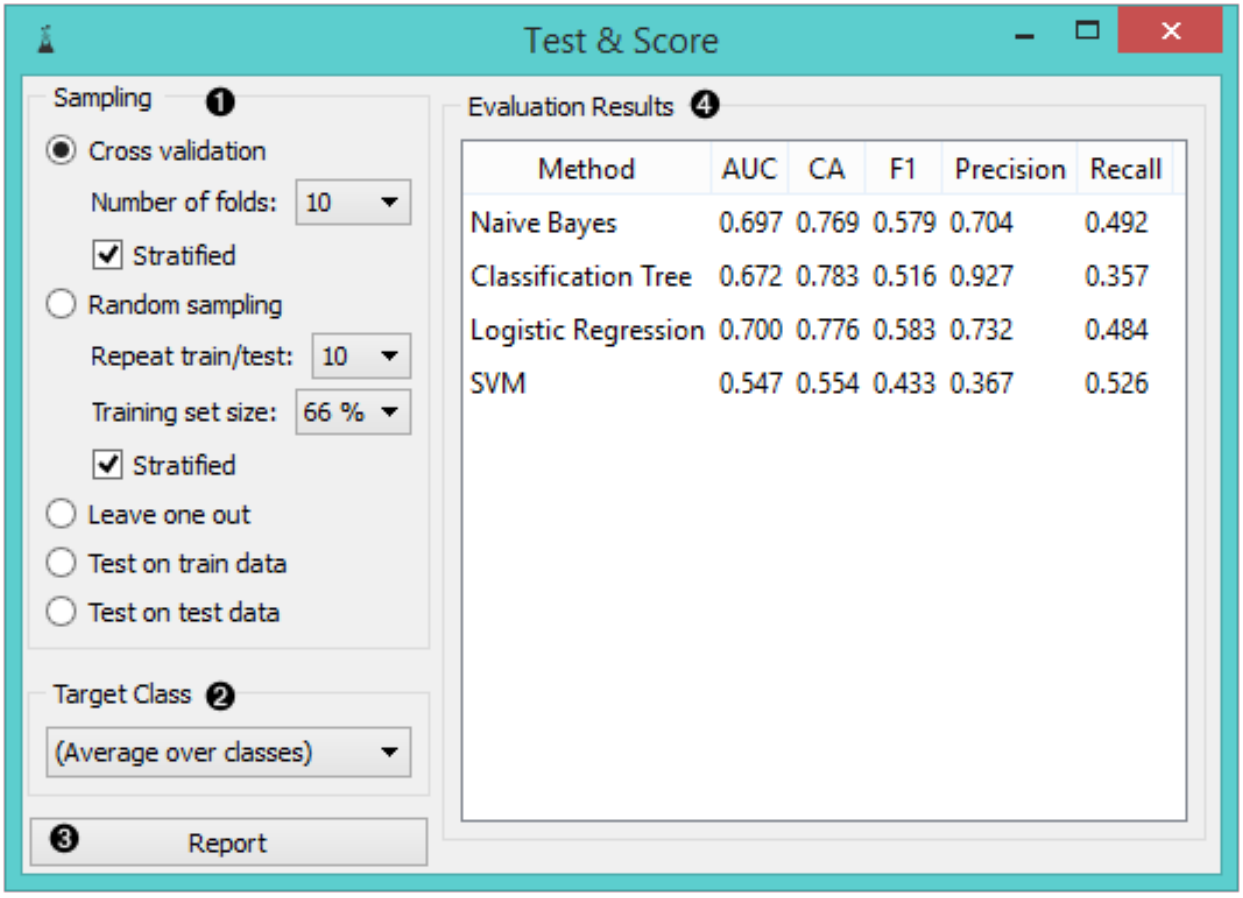
- **Evaluation results**

  Results of testing the algorithms.

Description

The widget tests learning algorithms. Different sampling schemes are available, including using separate test data. The widget does two things. First, it shows a table with different classifier performance measures, such as classification accuracy and area under the curve. Second, it outputs evaluation results, which can be used by other widgets for analyzing the performance of classifiers, such as *ROC Analysis* or *Confusion Matrix*.

The *Learner* signal has an uncommon property: it can be connected to more than one widget to test multiple learners with the same procedures.



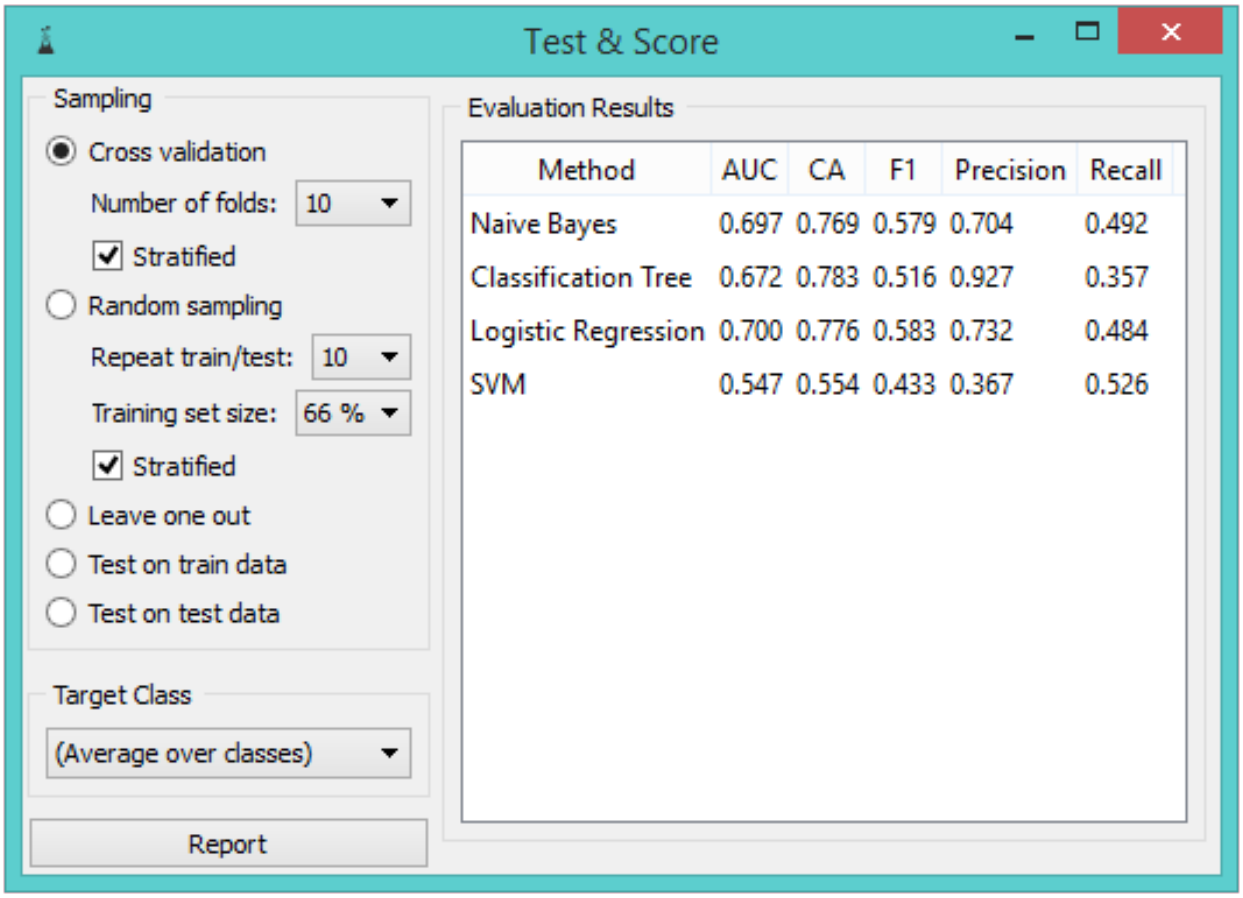
1. The widget supports various sampling methods.

   - Cross-validation splits the data into a given number of folds (usually 5 or 10). The algorithm is tested by holding out examples from one fold at a time; the model is induced from other folds and examples from the held out fold are classified. This is repeated for all the folds.

- **Leave-one-out** is similar, but it holds out one instance at a time, inducing the model from all others and then classifying the held out instances. This method is obviously very stable, reliable . . . and very slow.
- **Random sampling** randomly splits the data into the training and testing set in the given proportion (e.g. 70:30); the whole procedure is repeated for a specified number of times.
- **Test on train data** uses the whole data set for training and then for testing. This method practically always gives wrong results.
- **Test on test data**: the above methods use the data from *Data* signal only. To input another data set with testing examples (for instance from another file or some data selected in another widget), we select *Separate Test Data* signal in the communication channel and select Test on test data.

2. Only *Test on test data* requires a target class, e.g. having the disease or being of subvariety *Iris setosa*. When *Target class* is (None), the methods return the average value. Target class can be selected at the bottom of the widget.
3. Produce a report.
4. The widget will compute a number of performance statistics:

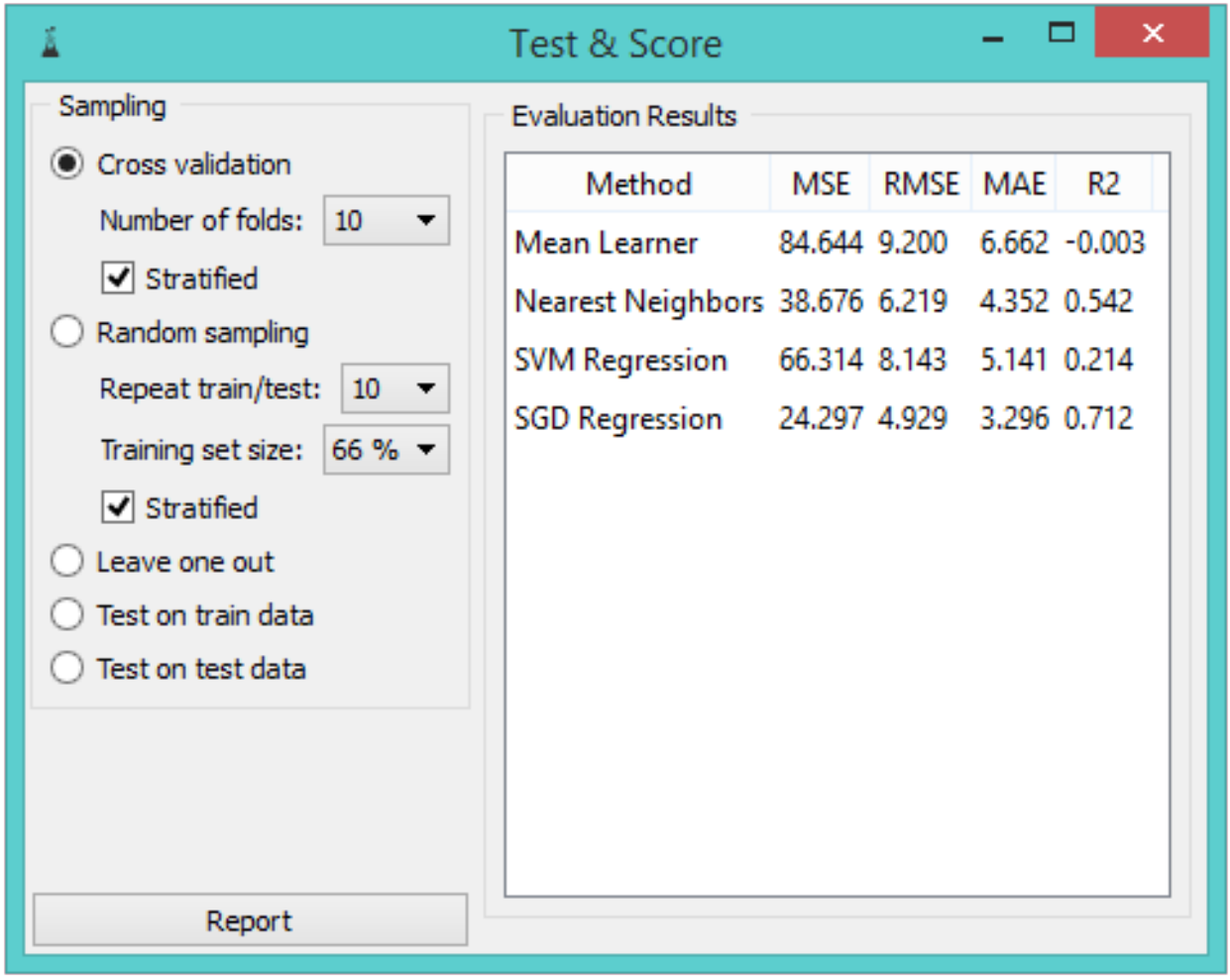
Classification



- Area under ROC is the area under the receiver-operating curve.
- Classification accuracy is the proportion of correctly classified examples.
- F-1 is a weighted harmonic mean of precision and recall (see below).

- Precision is the proportion of true positives among instances classified as positive, e.g. the proportion of *Iris virginica* correctly identified as Iris virginica.
- Recall is the proportion of true positives among all positive instances in the data, e.g. the number of sick among all diagnosed as sick.

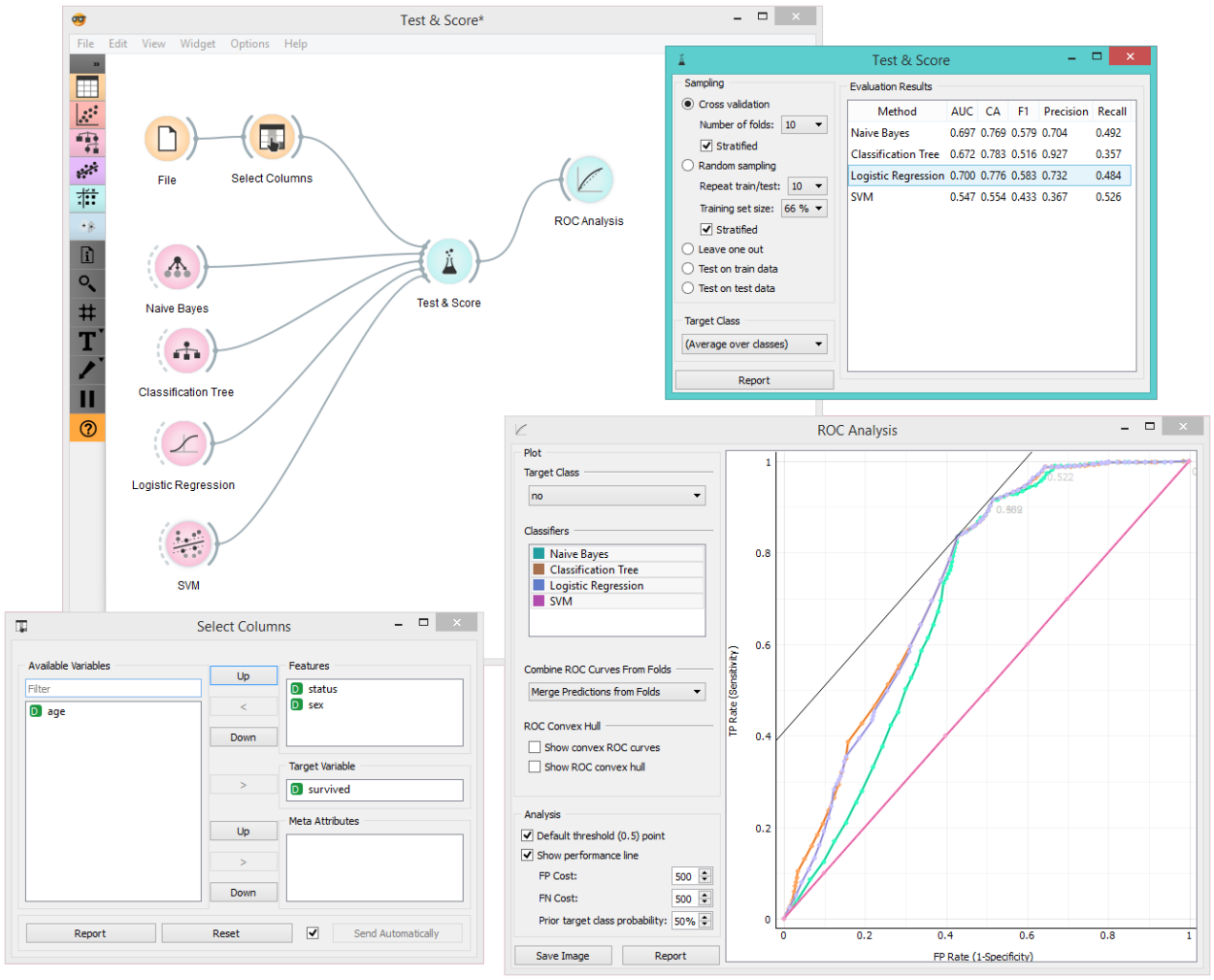
Regression



- MSE measures the average of the squares of the errors or deviations (the difference between the estimator and what is estimated).
- RMSE is the square root of the arithmetic mean of the squares of a set of numbers (a measure of imperfection of the fit of the estimator to the data)
- MAE is used to measure how close forecasts or predictions are to eventual outcomes.
- R2 is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.

Example

In a typical use of the widget, we give it a data set and a few learning algorithms and we observe their performance in the table inside the **Test & Score** widget and in the *ROC*. The data is often preprocessed before testing; in this case we did some manual feature selection (*Select Columns* widget) on *Titanic* data set, where we want to know only the sex and status of the survived and omit the age.



Another example of using this widget is presented in the documentation for the *Confusion Matrix* widget.