RapidMiner 4.2

User Guide

Operator Reference

Developer Tutorial

Rapid-I GmbH
Stockumer Str. 475
44227 Dortmund, Germany
http://www.rapidminer.com/

July 14, 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-world knowledge discovery processes typically consist of complex data preprocessing, machine learning, evaluation, and visualization steps. Hence a data mining platform should allow complex nested operator chains or trees, provide transparent data handling, comfortable parameter handling and optimization, be flexible, extendable and easy-to-use.

Depending on the task at hand, a user may want to interactively explore different knowledge discovery chains and continuously inspect intermediate results, or he may want to perform highly automated data mining processes off-line in batch mode. Therefore an ideal data mining platform should offer both, interactive and batch interfaces.

RapidMiner (formerly Yale) is an environment for machine learning and data mining processes. A modular operator concept allows the design of complex nested operator chains for a huge number of learning problems. The data handling is transparent to the operators. They do not have to cope with the actual data format or different data views - the RapidMiner core takes care of the necessary transformations. Today, RapidMiner is the world-wide leading open-source data mining solution and is widely used by researchers and companies.

RapidMiner introduces new concepts of transparent data handling and process modelling which eases process configuration for end users. Additionally clear interfaces and a sort of scripting language based on XML turns RapidMiner into an integrated developer environment for data mining and machine learning. Some of these aspects will be discussed in the next sections. Please refer to [12, 16, 21] for further explanations. We highly appreciate if you cite RapidMiner in your scientific work. Please do so by citing

> Mierswa, I. and Wurst, M. and Klinkenberg, R. and Scholz, M. and Euler, T., Yale (now: RapidMiner): Rapid Prototyping for

Complex Data Mining Tasks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), 2006.

1.1 Modeling Knowledge Discovery Processes as Operator Trees

Knowledge discovery (KD) processes are often viewed as sequential operator chains. In many applications, flat linear operator chains are insufficient to model the KD process and hence operator chains need to be nestable. For example a complex KD process containing a learning step, whose parameters are optimized using an inner cross-validation, and which as a whole is evaluated by an outer cross-validation. Nested operator chains are basically trees of operators.

In RapidMiner, the leafs in the operator tree of a KD process correspond to simple steps in the modeled process. Inner nodes of the tree correspond to more complex or abstract steps in the process. The root of the tree hence corresponds to the whole process.

Operators define their expected inputs and delivered outputs as well as their obligatory and optional parameters, which enables RapidMiner to automatically check the nesting of the operators, the types of the objects passed between the operators, and the mandatory parameters. This eases the design of complex data mining processes and enables RapidMiner to automatically check the nesting of the operators, the types of the objects passed between the operators, and the mandatory parameters.

Figure 1.1 shows a nested KD process for feature selection using a genetic algorithm with an inner cross-validation for evaluating candidate feature sets and an outer cross-validation for evaluating the genetic algorithm as a feature selector.

1.2 RapidMiner as a Data Mining Interpreter

RapidMiner uses XML (eXtensible Markup Language), a widely used language well suited for describing structured objects, to describe the operator trees modeling KD processes. XML has become a standard format for data exchange. Furthermore this format is easily readable by humans and machines. All RapidMiner processes are described in an easy XML format. You can see this XML description as a scripting language for data mining pocesses.

The graphical user interface and the XML based scripting language turn RapidMiner into an IDE and interpreter for machine learning and data mining. Fur-

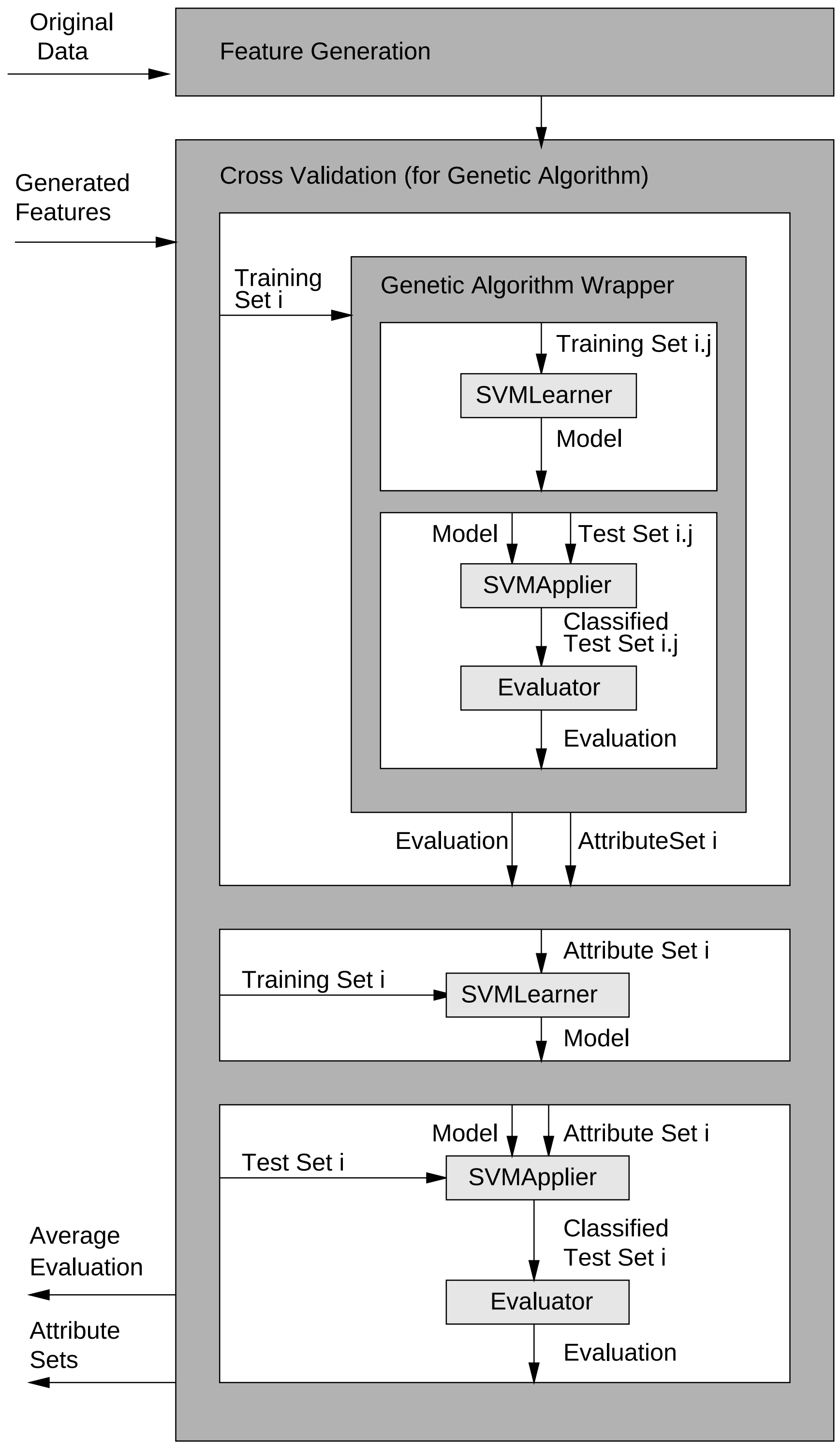


Figure 1.1: Nested operator chain for feature selection using a genetic algorithm.

thermore, the XML process configuration files define a standardized interchange format for data mining processes.

1.3 Different Ways of Using RapidMiner

RapidMiner can be started off-line, if the process configuration is provided as XML file. Alternatively, the GUI of RapidMiner can be used to design the XML description of the operator tree, to interactively control and inspect running processes, and to continuously monitor the visualization of the process results. Break points can be used to check intermediate results and the data flow between operators. Of course you can also use RapidMiner from your program. Clear interfaces define an easy way of applying single operators, operator chains, or complete operator trees on you input data. A command line version and a Java API allows invoking of RapidMiner from your programs without using the GUI. Since RapidMiner is entirely written in Java, it runs on any major platform/operating system.

1.4 Multi-Layered Data View Concept

RapidMiner's most important characteristic is the ability to nest operator chains and build complex operator trees. In order to support this characteristic the RapidMiner data core acts like a data base management system and provides a multi-layered data view concept on a central data table which underlies all views. For example, the first view can select a subset of examples and the second view can select a subset of features. The result is a single view which reflects both views. Other views can create new attributes or filter the data on the fly. The number of layered views is not limited.

This multi-layered view concept is also an efficient way to store different views on the same data table. This is especially important for automatic data preprocessing tasks like feature generation or selection. For example, the population of an evolutionary operator might consist of several data views - instead of several copies of parts of the data set.

No matter whether a data set is stored in memory, in a file, or in a database, RapidMiner internally uses a special type of data table to represent it. In order not to unnecessarily copy the data set or subsets of it, RapidMiner manages views on this table, so that only references to the relevant parts of the table need to be copied or passed between operators. These views are nestable as is for example required for nested cross-validations by maintaining a stack of views. In case of an example set, views on the rows of the table correspond to subsets of the example set, and views on the columns correspond to the selected features used to represent these examples.

1.5 Transparent Data Handling

RapidMiner supports flexible process (re)arrangements which allows the search for the best learning scheme and preprocessing for the data and learning task at hand. The simple adaptation and evaluation of different process designs allow the comparison of different solutions.

RapidMiner achieves a transparent data handling by supporting several types of data sources and hiding internal data transformations and partitioning from the user. Due to the modular operator concept often only one operator has to be replaced to evaluate its performance while the rest of the process design remains the same. This is an important feature for both scientific research and the optimization of real-world applications.

The input objects of an operator may be consumed or passed on to following or enclosing operators. If the input objects are not required by this operator, they are simply passed on, and may be used by later or outer operators. This increases the flexibility of RapidMiner by easing the match of the interfaces of consecutive operators and allowing to pass objects from one operator through several other operators to the goal operator.

Objects typically passed between operators are example sets, prediction models, evaluation vectors, etc. Operators may add information to input objects, e.g. labels to previously unlabeled examples, or new features in a feature generation operator, and deliver these extended objects.

1.6 Meta Data

To guide the transformation of the feature space or the automatical search for the best preprocessing, the user can define additional meta data on the data set at hand. Meta data include the type of attributes or their unit (SI). This information is for example used by the feature generation / construction algorithms provided by RapidMiner. The definition of meta information on your data is optional and if it is omitted RapidMiner tries to guess the correct data types.

1.7 Large Number of Built-in Data Mining Operators

RapidMiner provides more than 400 operators including:

Machine learning algorithms: a huge number of learning schemes for regression and classification tasks including support vector machines (SVM), decision tree and rule learners, lazy learners, Bayesian learners, and Logistic learners. Several algorithms for association rule mining and clustering are also part of RapidMiner. Furthermore, we added several meta learning schemes including Bayesian Boosting.

Weka operators: all Weka operations like learning schemes and attribute evaluators of the Weka learning environment are also available and can be used like all other RapidMiner operators.

Data preprocessing operators: discretization, example and feature filtering, missing and infinite value replenishment, normalization, removal of useless features, sampling, dimensionality reduction, and more.

Feature operators: selection algorithms like forward selection, backward elimination, and several genetic algorithms, operators for feature extraction from time series, feature weighting, feature relevance, and generation of new features.

Meta operators: optimization operators for process design, e.g. example set iterations or several parameter optimization schemes.

Performance evaluation: cross-validation and other evaluation schemes, several performance criteria for classification and regression, operators for parameter optimization in enclosed operators or operator chains.

Visualization: operators for logging and presenting results. Create online 2D and 3D plots of your data, learned models and other process results.

In- and output: flexible operators for data in- and output, support of several file formats including arff, C4.5, csv, bibtex, dBase, and reading directly from databases.

1.8 Extending RapidMiner

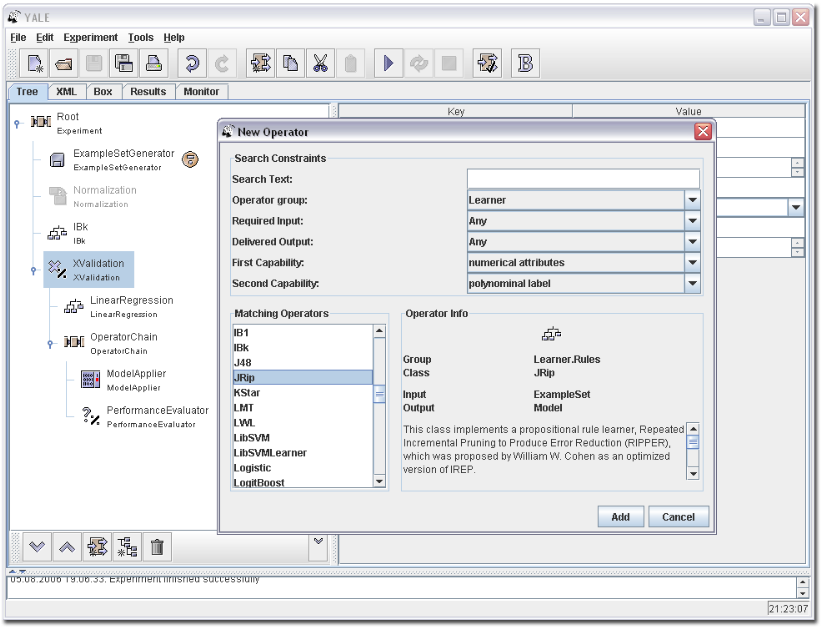
RapidMiner supports the implementation of user-defined operators. In order to implement an operator, the user simply needs to define the expected inputs, the delivered outputs, the mandatory and optional parameters, and the core functionality of the operator. Everything else is done by RapidMiner. The operator description in XML allows RapidMiner to automatically create corresponding GUI elements. This is explained in detail in chapter 6. An easy-to-use plugin mechanism is provided to add future operators or operators written by the RapidMiner community into RapidMiner. Several plugins are already provided in the download section of RapidMiner.

External programs can be integrated by implementing wrapper operators and can then be transparently used in any RapidMiner process.

1.9 Example Applications

RapidMiner has already been applied for machine learning and knowledge discovery tasks in a number of domains including feature generation and selection [5, 10, 22, 23], concept drift handling [9, 8, 7, 11], and transduction [2, 6]. In addition to the above-mentioned, current application domains of RapidMiner also include the pre-processing of and learning from time series [13, 17, 18], meta learning [19, 20], clustering, and text processing and classification. There exist several plugins to provide operators for these special learning tasks. Among these, there are some unusual plugins like GAStruct which can be used to optimize the design layout for chemical plants [14, 15].

The Figures 1.2 and 1.3 show screenshots from two process definitions performed with the GUI version of RapidMiner. Figure 1.2 depicts the process tree and the panel for setting the parameters of one of its operators in a feature selection process using backward elimination as feature selector. Figure 1.3 demonstrates the continuous result display of a parameter optimization process.



Search Text:
Operator group: Learner
Required Input: Any
Delivered Output: Any
First Capability: numerical attributes
Second Capability: polynominal label

IB1
IBk
J48
JRip
KStar
LMT
LWL
LibSVM
LibSVMLearner
Logistic
LogitBoost

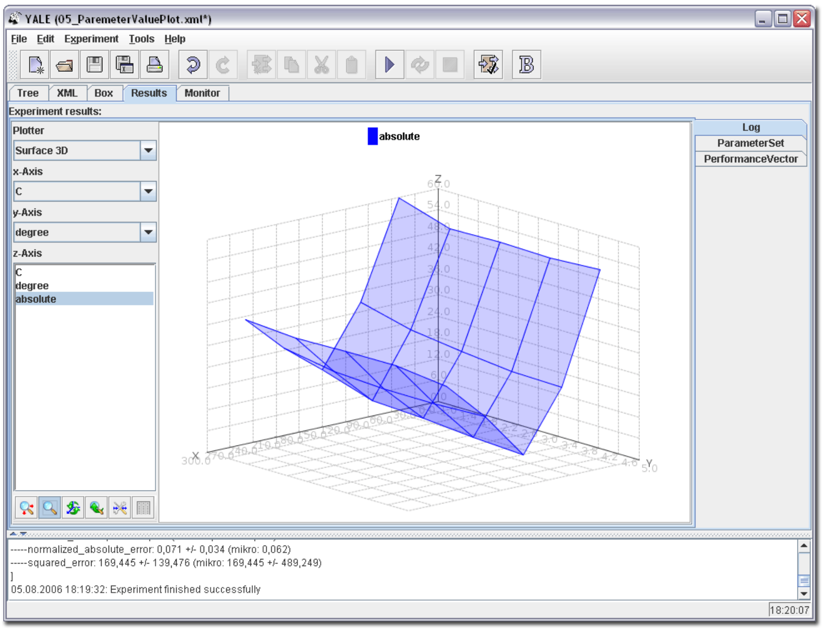
Group Learner.Rules
Class JRip
Input ExampleSet
Output Model

This class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W. Cohen as an optimized version of IREP.

05.08.2006 19:06:33: Experiment finished successfully

21:23:07

Figure 1.2: RapidMiner screenshot of the process tree and the panel for setting the parameters of a feature selection operator.



Plotter
Surface 3D
x-Axis
C
y-Axis
degree
z-Axis
C
degree
absolute

```
-----normalized_absolute_error: 0,071 +/- 0,034 (mikro: 0,062)
-----squared_error: 169,445 +/- 139,476 (mikro: 169,445 +/- 489,249)
]
05.08.2006 18:19:32: Experiment finished successfully
```

Figure 1.3: RapidMiner screenshot of the continuous result display of a parameter optimization process.

Use RapidMiner and explore your data! Simplify the construction of data mining processes and the evaluation of different approaches. Try to find the best combination of preprocessing and learning steps or let RapidMiner do that automatically for you. Have fun!

1.10 How this tutorial is organized

First you should read this chapter in order to get an idea of the concepts of RapidMiner. Thereafter, we suggest that you read the GUI manual of RapidMiner and make the online tutorial. It will be much easier to understand the details explained in the next chapters. Chapter 3 describes possible first steps and the basics of RapidMiner. In Chapter 4 we discuss more advanced processes. You should read at least these two chapters to create your own process setups. Chapter 5 provides information about all RapidMiner core operators. It is an operator reference, i.e. you can look up the details and description of all operators. Chapter 6 can be omitted if you want to use RapidMiner on your data or just want to perform some basic process definitions. In this chapter we describe ways to extend RapidMiner by writing your own operators or building own plugins.

Chapter 2

Installation and starting notes

2.1 Download

The latest version of RapidMiner is available on the RapidMiner homepage:

> http://www.rapidminer.com/.

The RapidMiner homepage also contains this document, the RapidMiner javadoc, example datasets, plugins, and example configuration files.

2.2 Installation

This section describes the installation of RapidMiner on your machine. You may install RapidMiner for all users of your system or for your own account locally.

Basically, there are two different ways of installing RapidMiner:

- Installation of a Windows executable
- Installation of a Java version (any platform)

Both ways are described below. More information about the installation of RapidMiner can be found at http://www.rapidminer.com/.

2.2.1 Installing the Windows executable

Just perform a double click on the downloaded file

```
    rapidminer-XXX-install.exe
```

and follow the installation instructions. As a result, there will be a new menu entry in the Windows startmenu. RapidMiner is started by clicking on this entry.

2.2.2 Installing the Java version (any platform)

RapidMiner is completely written in Java, which makes it run on almost every platform. Therefore it requires a Java Runtime Environment (JRE) version 5.0 (aka 1.5.0) or higher to be installed properly. The runtime environment JRE is available at http://java.sun.com/. It must be installed before RapidMiner can be installed.

In order to install RapidMiner, choose an installation directory and unzip the downloaded archive using WinZIP or tar or similar programs:

```
> unzip rapidminer-XXX-bin.zip
```

for the binary version or

```
> unzip rapidminer-XXX-src.zip
```

for the version containing both the binaries and the sources. This will create the RapidMiner home directory which contains the files listed in table 2.1.

2.3 Starting RapidMiner

If you have used the Windows installation executable, you can start RapidMiner just as any other Windows program by selecting the corresponding menu item from the start menu.

On some operating systems you can start RapidMiner by double-clicking the file `rapidminer.jar` in the `lib` subdirectory of RapidMiner. If that does not work, you can type `java -jar rapidminer.jar` on the command prompt. You can also use the startscripts `scripts/rapidminer` (commandline version) or `scripts/RapidMinerGUI` (graphical user interface version) for Unix or `scripts/rapidminer.bat` and `scripts/RapidMinerGUI.bat` for Windows.

If you intend to make frequent use of the commandline version of RapidMiner, you might want to modify your local startup scripts adding the `scripts` directory to your `PATH` environment variable. If you decide to do so, you can

`etc/`	Configuration files
`lib/`	Java libraries and jar files
`lib/rapidminer.jar`	The core RapidMiner java archive
`lib/plugins`	Plugin files (Java archives)
`licenses/`	The GPL for RapidMiner and library licenses
`resources/`	Resource files (source version only)
`sample/`	Some sample processes and data
`scripts/`	Executables
`scripts/rapidminer`	The commandline Unix startscript
`scripts/rapidminer.bat`	The commandline Windows startscript
`scripts/RapidMinerGUI`	The GUI Unix startscript
`scripts/RapidMinerGUI.bat`	The GUI Windows startscript
`src/`	Java source files (source version only)
`INSTALL`	Installation notes
`README`	Readme files for used libraries
`CHANGES`	Changes from previous versions
`LICENSE`	The GPL

Table 2.1: The RapidMiner directory structure.

start a process by typing `rapidminer <processfile>` from anywhere on your system. If you intend to make frequent use of the GUI, you might want to create a desktop link or a start menu item to `scripts/RapidMinerGUI` or `scripts/RapidMinerGUI.bat`. Please refer to your window manager documentation on information about this. Usually it is sufficient to drag the icon onto the desktop and choose ”‘Create link”’ or something similar.

Congratulations: RapidMiner is now installed. In order to check if RapidMiner is correctly working, you can go to the `sample` subdirectory and test your installation by invoking RapidMiner on the file `Empty.xml` which contains the simplest process setup that can be conducted with RapidMiner. In order to do so, type

```
cd sample
rapidminer Empty.xml
```

The contents of the file `Empty.xml` is shown in figure 2.1.

```
<operator name=”Root” class=”Process”>
</operator>
```

Figure 2.1: Installation test

Though this process does, as you might guess, nothing, you should see the message "Process finished successfully" after a few moments if everything goes well. Otherwise the words "Process not successful" or another error message can be read. In this case something is wrong with the installation. Please refer to the Installation section of our website http://www.rapidminer.com/ for further installation details and for pictures describing the installation process.

2.4 Memory Usage

Since performing complex data mining tasks and machine learning methods on huge data sets might need a lot of main memory, it might be that RapidMiner stops a running process with a note that the size of the main memory was not sufficient. In many cases, things are not as worse at this might sound at a first glance. Java does not use the complete amount of available memory per default and memory must be explicitely allowed to be used by Java.

On the installation page of our web site http://www.rapidminer.com/ you can find a description how the amount of memory usable by RapidMiner can be increased. This is, by the way, not necessary for the Windows executable of RapidMiner since the amount of available memory is automatically calculated and properly set in this case.

2.5 Plugins

In order to install RapidMiner plugins, it is sufficient to copy them to the `lib/plugins` subdirectory of the RapidMiner installation directory. RapidMiner scans all `jar` files in this directory. In case a plugin comes in an archive containing more than a single `jar` file (maybe documentation or samples), please only put the `jar` file into the `lib/plugins` directory and refer to the plugin documentation about what to do with the other files. For an introduction of how to create your own plugin, please refer to section 6.9 of this tutorial.

For Windows systems, there might also be an executable installer ending on .exe which can be used to automatically install the plugin into the correct directory. In both cases the plugin will become available after the next start of RapidMiner.

2.6 General settings

During the start up process of RapidMiner you can see a list of configuration files that are checked for settings. These are the files `rapidminerrc` and `rapidminerrc.OS` where OS is the name of your operating system, e.g. "Linux" or "Windows 2000". Four locations are scanned in the following order

1. The RapidMiner home directory (the directory in which it is installed.)
2. The directory `.rapidminer` in your home directory.
3. The current working directory.
4. Finally, the file specified by the java property *rapidminer.rcfile* is read. Properties can be passed to java by using the `-D` option:

   ```
   java -Drapidminer.rcfile=/my/rapidminer/rcfile -jar rapidminer.jar
   ```

Parameters in the home directory can override global parameters. The most important options are listed in table 2.2 and take the form *key=value*. Comments start with a #. Users that are familiar with the Java language recognize this file format as the Java property file format.

A convenient dialog for setting these properties is available in the file menu of the GUI version of RapidMiner.

2.7 External Programs

The properties discussed in the last section are used to determine the behavior of the RapidMiner core. Additionally to this, plugins can require names and paths of executables used for special learning methods and external tools. These paths are also defined as properties. The possibility of using external programs such as machine learning methods is discussed in the operator reference (chapter 5). These programs must have been properly installed and must be executable without RapidMiner, before they can be used in any RapidMiner process setup. By making use of the `rapidminerrc.OS` file, paths can be set in a platform dependent manner.

2.8 Database Access

It is very simple to access your data from a database management system like Oracle, Microsoft SQL Server, PostgreSQL, or mySQL. RapidMiner supports a wide range of systems without any additional effort. If your database management system is not natively supported, you simply have to add the JDBC driver for your system to the directory `lib/jdbc` or to your CLASSPATH variable.

Key	**Description**
`rapidminer.general.capabilities.warn`	indicates if only a warning should be shown if a learner does not have sufficient capabilities
`rapidminer.general.randomseed`	the default random seed
`rapidminer.tools.sendmail.command`	the sendmail command to use for sending notification emails
`rapidminer.tools.gnuplot.command`	the full path to the gnuplot executable (for GUI only)
`rapidminer.tools.editor`	external editor for Java source code
`rapidminer.gui.attributeeditor.rowlimit`	limit number of examples in attribute editor (for performance reasons)
`rapidminer.gui.beep.success`	beeps on process success
`rapidminer.gui.beep.error`	beeps on error
`rapidminer.gui.beep.breakpoint`	beeps on reaching a breakpoint
`rapidminer.gui.processinfo.show`	indicates if some information should be displayed after process loading
`rapidminer.gui.plaf`	the pluggable look and feel; may be `system`, `cross_platform`, or classname
`rapidminer.gui.plotter.colors.classlimit`	limits the number of nominal values for colorized plotters, e.g. color histograms
`rapidminer.gui.plotter.legend.classlimit`	limits the number of nominal values for plotter legends
`rapidminer.gui.plotter.matrixplot.size`	the pixel size of plotters used in matrix plots
`rapidminer.gui.plotter.rows.maximum`	limits the sample size of data points used for plotting
`rapidminer.gui.undolist.size`	limit for number of states in the undo list
`rapidminer.gui.update.check`	indicates if automatic update checks should be performed

Table 2.2: The most important rapidminerrc options.

If you want to ease the access to your database even further, you might think of defining some basic properties and description in the file

```
resources/jdbc_properties.xml
```

although this is not necessary to work on your databases and basically only eases the usage of the database wizard for the convenient connection to your database and query creation.

Chapter 3

First steps

This chapter describes some basic concepts of RapidMiner. In the description, we assume that most of the processes are performed in batch mode (or command line mode). Of course you can also use RapidMiner in the Graphical User Interface mode which is more convenient and offers a large amount of additional features. A short documentation of the GUI mode is separately available in the download section of the RapidMiner website. RapidMiner provides an online tutorial which also describes the usage of the GUI mode and the basic concepts of machine learning with RapidMiner. Probably, you will not need to read all sections of this tutorial after making the online tutorial and reading the short GUI manual. However, you should at least read this section to get a first idea about some of the RapidMiner concepts.

All examples described in this tutorial are part of the sample directories of RapidMiner. Although only few of these examples are discussed here, you should take a look at all of them since they will give you some helpful hints. We suggest that you start approximately the first half of the process definitions in each of the sample directories in the order the directories are named, i.e. first the first half of directory `01_IO`, then the first half of `02_Learner` and so on. After this round, you should again start with the first directory and perform the second half of the process setups. This way the more complicated processes will be performed after you had a look at almost all of the simple building blocks and operators.

3.1 First example

Let us start with a simple example `03_XValidation_Numerical.xml` which you can find in the `04_Validation` subdirectory. This example process loads an example set from a file, generates a model using a support vector machine

(SVM) and evaluates the performance of the SVM on this dataset by estimating the expected absolute and squared error by means of a ten-fold cross-validation. In the following we will describe what the parameters mean without going into detail too much. We will describe the used operators later in this section.

```
<operator name="Root" class="Process">
  <parameter key="logfile" value="XValidation.log"/>
  <operator name="Input" class="ExampleSource">
    <parameter key="attributes" value="../data/polynomial.aml"/>
  </operator>
  <operator name="XVal" class="XValidation">
    <operator name="Training" class="LibSVMLearner">
      <parameter key="svm_type" value="epsilon−SVR"/>
      <parameter key="kernel_type" value="poly"/>
      <parameter key="C" value="1000.0"/>
    </operator>
    <operator name="ApplierChain" class="OperatorChain">
      <operator name="Test" class="ModelApplier">
      </operator>
      <operator name="Evaluation" class="PerformanceEvaluator">
        <parameter key="squared_error" value="true"/>
      </operator>
    </operator>
  </operator>
</operator>
```

Figure 3.1: Simple example configuration file. This is the `03_XValidation_Numerical.xml` sample process

But first of all let's start the process. We assume that your current folder contains the file `03_XValidation_Numerical.xml` (see figure 3.1). Now start RapidMiner by typing

```
rapidminer 03_XValidation_Numerical.xml
```

or by opening that file with the GUI and pressing the start button. After a short while you should read the words "Process finished successfully". Congratulations, you just made your first RapidMiner process. If you read "Process not successful" instead, something went wrong. In either case you should get some information messages on your console (using RapidMiner in batch mode) or in the message viewer (GUI mode). In the latter case it should give you information about what went wrong. All kinds of debug messages as well as information messages and results like the calculated relative error are written to this output. Have a look at it now.

The log message starts with the process tree and contains a lot of warnings, because most of the parameters are not set. Don't panic, reasonable default values are used for all of them. At the end, you will find the process tree again.

The number in squared brackets following each operator gives the number of times the operator was applied. It is one for the outer operators and ten within the ten-fold cross-validation. Every time an operator is applied a message is written to the log messages indicating its input objects (like example sets and models). When the operator terminates its application it writes the output to the log stream again. You can find the average performance estimated by the cross-validation close to the end of the messages.

Taking a look at the process tree in the log messages once again, you will quickly understand how the configuration file is structured. There is one `operator` tag for each operator specifying its name and class. Names must be unique and have the only purpose of distinguishing between instances of the same class. Operator chains like the cross-validation chain may contain one or more inner operators. Parameters can be specified in the form of key-value pairs using a `parameter` tag.

We will now focus on the operators without going into detail too much. If you are interested in the the operator classes, their input and output objects, parameters, and possible inner operators you may consult the reference section of this tutorial (chapter 5).

The outermost operator called "Root" is a Process operator, a subclass of a simple OperatorChain. An operator chain works in a very simple manner. It applies its inner operators successively passing their respective output to the next inner operator. The output of an operator chain is the output of the last inner operator. While usual operator chains do not take any parameters, this particular operator chain (being the outermost operator) has some parameters that are important for the process as a whole, e.g. the name of the log file (*logfile*) and the name of the directory for temporary files (*temp_dir*).

The ExampleSource operator loads an example set from a file. An additional file containing the attribute descriptions is specified (`data/polynomial.xml`). References to the actual data files are specified in this file as well (see section 3.4 for a description of the files). Then the resulting example set is passed to the cross-validation chain.

The XValidation evaluates the learning method by splitting the input example set into ten subsets $S_1, \ldots, S_{10}$. The inner operators are applied ten times. In run number i the first inner operator, which is a LibSVMLearner, generates a model using the training set $\bigcup_{j \neq i} S_j$. The second inner operator, an evaluation chain, evaluates this model by applying it to the remaining test set S_i. The ModelApplier predicts labels for the test set and the PerformanceEvaluator compares them to the real labels. Afterwards the absolute and squared errors are calculated. Finally the cross-validation chain returns the average absolute and squared errors over the ten runs and their variances.

The processing of RapidMiner operator trees is similar to a depth first search of normal trees. In contrast to this usual way of traversing a tree, RapidMiner allows loops during the run (each learning child is used 10 times, the applier chain is used 10 times, too). Additionally, inner nodes may perform some operations before they pass the output of the first children to the next child. The traversal through a RapidMiner operator tree containing leaf operators and simple operator chains only is actually equivalent to the usual depth first search traversal.

3.2 Process configuration files

Process configuration files are XML documents containing only four types of tags (extension: `.xml`). If you use the GUI version of RapidMiner, you can display the configuration file by clicking on the XML tab. Process files define the process tree consisting of operators and the parameters for these operators. Parameters are single values or lists of values. Descriptions can be used to comment your operators.

<operator>

The `operator` tag represents one instance of an operator class. Exactly two attributes must be present:

name: A unique name identifying this particular operator instance

class: The operator class. See the operator reference (chapter 5) for a list of operators.

For instance, an operator tag for an operator that reads an example set from a file might look like this:

<**operator** *name*="MyExampleSource" *class*="ExampleSource">
</**operator**>

If `class` is a subclass of OperatorChain, then nested operators may be contained within the opening and closing tag.

<parameter> and <list>

As discussed above, a parameter can have a single value or a set of values. For single value parameters the `<parameter>` tag is used. The attributes of the `<parameter>` tag are as follows:

key: The unique name of the parameter.

value: The value of the parameter.

In order to specify a filename for the example above, there might be used the following parameter:

```
<operator name="MyExampleSource" class="ExampleSource">
  <parameter key="attributes" value="myexamples.dat"/>
</operator>
```

If the parameter accepts a list of values, the `<list>` tag must be used. The list must have a `key` attribute, just as the `<parameter>` tag. The elements of the list are specified by nested `<parameter>` tags, e.g. in case of a FeatureGeneration operator (see section 5.8.42).

```
<list key="functions">
  <parameter key="sum"   value="+(a1,a2)"/>
  <parameter key="product" value="*(a3,a4)"/>
  <parameter key="nested" value="+(*(a1,a3),a4)"/>
</list>
```

<description>

All operators can have an inner tag named `<description>`. It has only one attribute named `text`. This attribute contains a comment for the enclosing operator. If the root operator of the process has an inner description tag, the text is displayed after loading the process setup.

```
<operator name="MyExampleSource" class="ExampleSource">
  <description text="Loads the data from file." />
</operator>
```

3.3 Parameter Macros

All text based parameters might contain so called macros which will be replaced by RapidMiner during runtime. For example, you can write a learned model into a file with the operator ModelWriter (see 5.3.31). If you want to do this for each learned model in a cross validation run, each model would be overwritten by the next one. How can this be prevented?

To save all models for each iteration in an own file, you need parameter macros. In a parameter value, the character '%' has a special meaning. Parameter values are expanded as follows:

%{a} is replaced by the number of times the operator was applied.

%{b} is replaced by the number of times the operator was applied plus one, i.e. $\%\mathrm{a}+1$. This is a shortcut for %p[1].

%{p[number }] is replaced by the number of times the operator was applied plus the given number, i.e. $\%\mathrm{a}+number$.

%{t} is replaced by the system time.

%{n} is replaced by the name of the operator.

%{c} is replaced by the class of the operator.

%{%} becomes %.

%{process_name} becomes the name of the process file (without path and extension).

%{process_file} becomes the name of the process file (with extension).

%{process_path} becomes the path of the process file.

For example to enumerate your files with ascending numbers, please use the following value for the key *model-file*:

```
<operator name="ModelWriter" class="ModelWriter">
  <parameter key="model_file" value="model_%{a}.mod"/>
</operator>
```

The macro %{a} will be replaced by the number of times the operator was applied, in case of model write after the learner of a 10-fold cross validation it will hence be replaced by the numbers 1 to 10.

You can also define own macros with help of the MacroDefinition operator (see 5.2.7).

3.4 File formats

RapidMiner can read a number of input files. Apart from data files it can read and write models, parameter sets and attribute sets. Generally, RapidMiner is able to read all files it generates. Some of the file formats are less important for the user, since they are mainly used for intermediate results. The most important file formats are those for "examples" or "instances". These data sets are provided by the user and almost all processes contain an operator that reads them.

3.4.1 Data files and the attribute description file

If the data files are in the popular arff format (extension: `.arff`), which provides some meta data, they can be read by the ArffExampleSource (see section 5.3.2). Other operators for special file formats are also available. Additionally, data can be read from a data base using the DatabaseExampleSource (see section 5.3.17). In that case, meta data is read from the data base as well.

The ExampleSource operator allows for a variety of other file formats in which instances are separated by newline characters. It is the main data input operator for RapidMiner. Comment characters can be specified arbitrarily and attributes can be spread over several files. This is especially useful in cases where attribute data and the label are kept in different files.

Sparse data files can be read using the SparseFormatExampleSource. We call data sparse if almost all values are equal to a default, e.g. zero.

The ExampleSource (for dense data) and some sparse formats need an attribute description file (extension: `.aml`) in order to retrieve meta data about the instances. This file is a simple XML document defining the properties of the attributes (like their name and range) and their source files. The data may be spread over several files. Therefore, the actual data files do not have to be specified as a parameter of the input operator.

The outer tag must be an `<attributeset>` tag. The only attribute of this tag may be `default_source=`*filename*. This file will be used as a default file if it is not specified explicitly with the attribute.

The inner tags can be any number of `<attribute>` tags plus at most one tag for each special attribute. The most frequently used special attributes are `<label>`, `<weight>`, `<id>`, and `<cluster>`. Note that arbitrary names for special attributes may be used. Though the set of special attributes used by the core RapidMiner operators is limited to the ones mentioned above, plugins or any other additional operators may use more special attributes. Please refer to the operator documentation to learn more about the specific special attributes used or generated by these operators.

The following XML attributes may be set to specify the properties of the RapidMiner attribute declared by the corresponding XML tag (mandatory XML attributes are set in italic font):

name: The unique name of the attribute.

sourcefile: The name of the file containing the data. If this name is not specified, the default file is used (specified for the parent `attributeset` tag).

sourcecol: The column within this file (numbering starts at 1). Can be omitted for sparse data file formats.

sourcecol_end: If this parameter is set, its value must be greater than the value of `sourcecol`. In that case, $sourcecol - sourcecol\_end$ attributes are generated with the same properties. Their names are generated by appending numbers to the value of `name`. If the `blocktype` is `value_series`, then `value_series_start` and `value_series_end` respectively are used for the first and last attribute blocktype in the series.

valuetype: One out of `nominal`, `numeric`, `integer`, `real`, `ordered`, `binominal`, `polynominal`, and `file_path`

blocktype: One out of `single_value`, `value_series`, `value_series_start`, `value_series_end`, `interval`, `interval_start`, and `interval_end`.

Each *nominal* attribute, i.e. each attribute with a nominal (binominal, polynominal) value type definition, should define the possible values with help of inner tags

```
<value>nominal_value_1</value>
<value>nominal_value_2</value>
...
```

See figure 3.2 for an example attribute description file. For classification learners that can handle only binary classifications (e.g. "yes" and "no") the first defined value in the list of nominal values is assumed to be the negative label. That includes the classification "yes" is not necessarily the positive label (depending on the order). This is important, for example, for the calculation of some performance measurements like precision and recall.

Note: Omitting the inner value tags for nominal attributes will usually *seem* to work (and indeed, in many cases no problems might occur) but since the internal representation of nominal values depend on this definition it might happend that the nominal values of learned models do not fit the given data set. Since this might lead to drastically reduced prediction accuracies you should always define the nominal values for nominal attributes.

Note: You do not need to specify a label attribute in cases where you only want to predict a label with a learned model. Simply describe the attributes in the same manner as in the learning process setup, the label attribute can be omitted.

```
<attributeset default_source=”golf.dat”>
  <attribute
    name      =”Outlook”
    sourcecol =”1”
    valuetype =”nominal”
    blocktype =”single_value”
    classes   =”rain overcast sunny”
  />
  <attribute
    name      =”Temperature”
    sourcecol =”2”
    valuetype =”integer”
    blocktype =”single_value”
  />
  <attribute
    name      =”Humidity”
    sourcecol =”3”
    valuetype =”integer”
    blocktype =”single_value”
  />
  <attribute
    name      =”Wind”
    sourcecol =”4”
    valuetype =”nominal”
    blocktype =”single_value”
    classes   =”true false”
  />
  <label
    name      =”Play”
    sourcecol =”5”
    valuetype =”nominal”
    blocktype =”single_value”
    classes   =”yes no”
  />
</attributeset>
```

Figure 3.2: An example attribute set description file in XML syntax.

Dense data files

The data files are in a very simple format (extension: `.dat`). By default, comments start with #. When a comment character is encountered, the rest of the line is discarded. Empty lines – after comment removal – are ignored. If the data is spread over several files, a non empty line is read from every file. If the end of one of the files is reached, reading stops. The lines are split into tokens that are whitespace separated by default, separated by a comma, or separated by semicolon. The number of the tokens are mapped to the `sourcecol` attributes specified in the attribute description file. Additional or other separators can be specified as a regular expression using the respective parameters of the ExampleSource (see section 5.3.21). The same applies for comment characters.

Sparse data files

If almost all of the entries in a data file are zero or have a default nominal value, it may be well suitable to use a SparseFormatExampleSource (see section 5.3.41). This operator can read an attribute description file as described above. If the *attribute_description_file* parameter is supplied, the attribute descriptions are read from this file and the `default_source` is used as the single data file. The `sourcecol` and `sourcefile` attributes are ignored. If the *attribute_description_file* parameter is not supplied, the data is read from the file *data_file* and attributes are generated with default value types. Regular attributes are supposed to be real numbers and the label is supposed to be nominal. In that case, the *dimension* parameter, which specifies the number of regular attributes, must be set.

Comments in the data file start with a '#'-character, empty lines are ignored. Lines are split into whitespace separated tokens of the form `index:value` where *value* is the attribute value, i.e. a number or a string, and *index* is either an index number referencing a regular attribute or a prefix for a special attribute defined by the parameter list *prefix_map* of the SparseFormatExampleSource. Please note that index counting starts with 1.

The SparseFormatExampleSource parameter *format* specifies the way labels are read.

xy The label is the last token in the line.

yx The label is the first token in the line.

prefix The label is treated like all other special attributes.

separate_file The label is read from a separate file. In that case, parameter *label_file* must be set.

no_label The example set is unlabeled.

All attributes that are not found in a line are supposed to have default values. The default value for numerical data is 0, the default vallue for nominal attributes is the first string specified by the `classes` attribute in the attribute description file.

Example Suppose you have a sparse file which looks like this:

```
w:1.0 5:1   305:5 798:1 yes
w:0.2 305:2 562:1       yes
w:0.8 49:1  782:1 823:2 no
...
```

You may want each example to have a special attribute “weight”, a nominal label taking the values “yes” and “no”, and 1 000 regular numerical attributes. Most of them are 0. The best way to read this file, is to use a SparseFormatExampleSource and set the parameter value of *format* to `xy` (since the label is the last token in each line) and use a *prefix_map* that maps the prefix “w” to the attribute “weight”. See figure 3.3 for a configuration file.

```
<operator name="SparseFormatExampleSource" class="SparseFormatExampleSource">
  <parameter key="dimension"      value="1000"/>
  <parameter key="attribute_file" value="mydata.dat"/>
  <parameter key="format"         value="xy"/>
  <list key="prefix_map">
    <parameter key="w" value="weight"/>
  </list>
</operator>
```

Figure 3.3: Configuration of a SparseFormatExampleSource

3.4.2 Model files

Model files contain the models generated by learning operators in previous RapidMiner runs (extension: `.mod`). Models can be written to a file by using the operator ModelWriter. They can be read by using a ModelLoader and applied by using a ModelApplier.

3.4.3 Attribute construction files

An AttributeConstructionsWriter writes an attribute set to a text file (extension: `.att`). Later, this file can be used by an AttributeConstruc-

TIONSLOADER operator to generate the same set of attributes in another process and/or for another set of data.

The attribute generation files can be generated by hand as well. Every line is of the form

<**attribute** *name*="attribute_name" construction="generation_description"/>

The generation description is defined by functions, with prefix-order notation. The functions can be nested as well. An example of a nested generation description might be: $f(g(a), h(b), c)$. See page 5.8.42 for a reference of the available functions.

Example of an attribute constructions file:

```
<constructions version="4.0">
    <attribute name="a2" construction="a2"/>
    <attribute name="gensym8" construction="∗(∗(a1, a2), a3)"/>
    <attribute name="gensym32" construction="∗(a2, a2)"/>
    <attribute name="gensym4" construction="∗(a1, a2)"/>
    <attribute name="gensym19" construction="∗(a2, ∗(∗(a1, a2), a3))"/>
</constructions>
```

3.4.4 Parameter set files

For example, the GRIDPARAMETEROPTIMIZATION operator generates a set of optimal parameters for a particular task (extension: `.par`). Since parameters of several operators can be optimized at once, each line of the parameter set files is of the form

```
OperatorName.parameter_name = value
```

These files can be generated by hand as well and can be read by a PARAMETERSETLOADER and set by a PARAMETERSETTER.

3.4.5 Attribute weight files

All operators for feature weighting and selection generate a set of feature weights (extension: `.wgt`). Attribute selection is seen as attribute weighting which allows more flexible operators. For each attribute the weight is stored, where a weight of 0 means that the attribute was not used at all. For writing the files to a file the operator ATTRIBUTEWEIGHTSWRITER can be used. In such a weights file each line is of the form

```
<weight name="attribute_name" value="weight"/>
```

These files can be generated by hand as well and can be read by an AttributeWeightsLoader and used on example sets with the operator AttributeWeightsApplier. They can also be read and adapted with the InteractiveAttributeWeighting operator. Feature operators like forward selection, genetic algorithms and the weighting operators can deliver an example set with the selection / weighting already applied or the original example set (optional). In the latter case the weights can adapted and changed before they are applied.

Example of an attribute weight file:

```
<attributeweights version="4.0">
    <weight name="a1" value="0.8"/>
    <weight name="a2" value="1.0"/>
    <weight name="a3" value="0.0"/>
    <weight name="a4" value="0.5"/>
    <weight name="a5" value="0.0"/>
</attributeweights>
```

3.5 File format summary

Table 3.1 summarizes all file formats and the corresponding file extensions.

Extension	**Description**
.aml	attribute description file (standard XML meta data format)
.arff	attribute relation file format (known from Weka)
.att	attribute set file
.bib	BibTeX data file format
.clm	cluster model file (clustering plugin)
.cms	cluster model set file (clustering plugin)
.cri	population criteria file
.csv	comma separated values data file format
.dat	(dense) data files
.ioc	IOContainer file format
.log	log file / process log file
.mat	matrix file (clustering plugin)
.mod	model file
.obf	obfuscation map
.par	parameter set file
.per	performance file
.res	results file
.sim	similarity matrix file (clustering plugin)
.thr	threshold file
.wgt	attribute weight file
.wls	word list file (word vector tool plugin)
.xrff	extended attribute relation file format (known from Weka)

Table 3.1: The most important file formats for RapidMiner.

Chapter 4

Advanced processes

At this point, we assume that you are familiar with the simple example from section 3.1. You should know how to read a dataset from a file, what a learner and a model applier do, and how a cross-validation chain works. These operators will be used frequently and without further explanation in this chapter. After reading this chapter you should be able to understand most of the sample process definitions provided in the `sample` directory of RapidMiner. You should have a look at these examples and play around to get familiar with RapidMiner.

4.1 Feature selection

Let us assume that we have a dataset with numerous attributes. We would like to test, whether all of these attributes are really relevant, or whether we can get a better model by omitting some of the original attributes. This task is called *feature selection* and the *backward elimination* algorithm is an approach that can solve it for you.

Here is how backward elimination works within RapidMiner: Enclose the cross-validation chain by a FeatureSelection operator. This operator repeatedly applies the cross-validation chain, which now is its inner operator, until the specified stopping criterion is complied with. The backward elimination approach iteratively removes the attribute whose removal yields the largest performance improvement. The stopping criterion may be for example that there has been no improvement for a certain number of steps. See section 5.8.45 for a detailed description of the algorithm. Figure 4.1 shows the configuration file.

You should try some of the following things:

- Use *forward selection* instead of backward elimination by changing the parameter value of *selection_direction* from `backward` to `forward`. This approach starts with an empty attribute set and iteratively adds the attribute whose inclusion improves the performance the most.

- Use the GeneticAlgorithm operator for feature selection instead of the FeatureSelection operator (see section 5.8.54).

- Replace the cross validation by a filter based evaluation. The sample process `FeatureSelectionFilter.xml` uses such a fast feature set evaluation.

- Compare the results of the three approaches above to the BruteForce operator. The brute force approach tests all subsets of the original attributes, i.e. all combinations of attributes, to select an optimal subset. While this operator is prohibitively expensive for large attribute sets, it can be used to find an optimal solution on small attribute sets in order to estimate the quality of the results of other approaches.

```
<operator name="Global" class="Process">
  <parameter key="logfile"      value="advanced1.log"/>

  <operator name="Input" class="ExampleSource">
    <parameter key="attributes" value="data/polynomial.aml"/>
  </operator>

  <operator name="BackwardElimination" class="FeatureSelection">
    <parameter key="selection_direction" value="backward"/>

    <operator name="XVal" class="XValidation">
      <parameter key="number_of_validations" value="5"/>

      <operator name="Learner" class="LibSVMLearner">
        <parameter key="kernel_type" value="poly"/>
        <parameter key="C"           value="1000.0"/>
        <parameter key="svm_type"    value="epsilon-SVR"/>
      </operator>
      <operator name="ApplierChain" class="OperatorChain">
        <operator name="Applier" class="ModelApplier"/>
        <operator name="Evaluator" class="PerformanceEvaluator">
          <parameter key="squared_error" value="true"/>
        </operator>
      </operator>
    </operator>
  </operator>
</operator>
```

Figure 4.1: A feature selection process

4.2 Splitting up Processes

If you are not a computer scientist but a data mining user, you are probably interested in a real-world application of RapidMiner. May be, you have a small labeled dataset and would like to train a model with an optimal attribute set. Later you would like to apply this model to your huge unlabeled database. Actually you have two separate processes.

4.2.1 Learning a model

This phase is basically the same as described in the preceeding section. We append two operators to the configuration file that write the results of the process into files. First, we write the attribute set to the file `selected_attributes.att` using an AttributeSetWriter. Second, we once again train a model, this time using the entire example set, and we write it to the file `model.mod` with help of a ModelWriter. For the configuration file see figure 4.2. Execute the process and take a look at the file `attributes.att`. It should contain the selected subset of the originally used attributes, one per line.

4.2.2 Applying the model

In order to apply this learned model to new unlabeled dataset, you first have to load this example set as usual using an ExampleSource. You can now load the trained model using a ModelLoader. Unfortunately, your unlabeled data probably still uses the original attributes, which are incompatible with the model learned on the reduced attribute set. Hence, we have to transform the examples to a representation that only uses the selected attributes, which we saved to the file `attributes.att`. The AttributeSetLoader loads this file and generates (or rather selects) the attributes accordingly. Now we can apply the model and finally write the labeled data to a file. See figure 4.3 for the corresponding configuration file.

As you can see, you can easily use different dataset source files even in different formats as long as you use consistent names for the attributes. You could also split the process into three parts:

1. Find an optimal attribute set and train the model.
2. Generate or select these attributes for the unlabeled data and write them to temporary files.
3. Apply the model from step one to the temporary files from step two and write the labeled data to a result file.

```
<operator name="Global" class="Process">
  <parameter key="logfile"       value="advanced2.log"/>

  <operator name="Input" class="ExampleSource">
    <parameter key="attributes" value="data/polynomial.aml"/>
  </operator>

  <operator name="BackwardElimination" class="FeatureSelection">
    <parameter key="selection_direction" value="backward"/>

    <operator name="XVal" class="XValidation">
      <parameter key="number_of_validations" value="5"/>

      <operator name="Learner" class="LibSVMLearner">
        <parameter key="kernel_type" value="poly"/>
        <parameter key="C"           value="1000.0"/>
        <parameter key="svm_type"  value="epsilon−SVR"/>
      </operator>
      <operator name="ApplierChain" class="OperatorChain">
        <operator name="Applier" class="ModelApplier"/>
        <operator name="Evaluator" class="PerformanceEvaluator">
          <parameter key="squared_error" value="true"/>
        </operator>
      </operator>
    </operator>
  </operator>

  <operator name="AttributeWeightsWriter" class="AttributeWeightsWriter">
        <parameter key=" attribute_weights_file" value=" selected_attributes .wgt"/>
    </operator>
  <operator name="Learner" class="LibSVMLearner">
    <parameter key="kernel_type" value="poly"/>
    <parameter key="C"           value="1000.0"/>
    <parameter key="svm_type"  value="epsilon−SVR"/>
    <parameter key="model_file" value="model.mod"/>
  </operator>
  <operator name="ModelOutput" class="ModelWriter">
    <parameter key="model_file" value="model.mod"/>
  </operator>
</operator>
```

Figure 4.2: Training a model and writing it to a file

Of course it is also possible to merge all process modules into one big process definition.

```
<operator name="Global" class="Process">
  <parameter key="logfile"          value="advanced3.log"/>

  <operator name="Input" class="ExampleSource">
    <parameter key="attributes" value="polynomial_unlabeled.aml"/>
  </operator>

  <operator name="AttributeWeightsLoader" class="AttributeWeightsLoader">
        <parameter key="attribute_weights_file" value=" selected_attributes .wgt"/>
    </operator>

    <operator name="AttributeWeightSelection" class="AttributeWeightSelection">
        <parameter key="weight" value="0.0"/>
        <parameter key="weight_relation"          value="greater"/>
    </operator>

  <operator name="ModelLoader" class="ModelLoader">
    <parameter key="model_file" value="model.mod"/>
  </operator>

  <operator name="Applier" class="ModelApplier"/>

  <operator class="ExampleSetWriter" name="ExampleSetWriter">
    <parameter key="example_set_file" value="polynom.labelled.dat"/>
  </operator>
</operator>
```

Figure 4.3: Applying the model to unlabeled data

4.3 Parameter and performance analysis

In this section we show how one can easily record performance values of an operator or operator chain depending on parameter values. In order to achieve this, the RapidMiner process setup described in this section makes use of two new RapidMiner operators: GridParameterOptimization (see section 5.5.10) and ProcessLog (see section 5.10.8).

We will see how to analyze the performance of a support vector machine (SVM) with a polynomial kernel depending on the two parameters degree d and ε.[1]

We start with the building block we should now be familiar with: a validation chain containing a LibSVMLearner, a ModelApplier, and a PerformanceEvaluator. Now we would like to vary the parameters.

[1] The performance of a polynomial SVM also depends on other parameters like e.g. C, but this is not the focus of this process.

Since we want to optimize more than one parameter, we cannot pass this information to the GridParameterOptimization operator using the usual `<parameter>` tag. As the latter is designed to take a single value, we must use the `<list>` tag, which can take several parameters. Similar to the `<parameter>` tag the `<list>` tag must have a key. In case of the GridParameterOptimization this key is (slightly confusingly in this context) named *parameters* (the list of parameters which should be optimized). Each parameter that might be optimized, needs a `<parameter>` tag entry in the `<list>`. The `key` of a `<parameter>` tag has the form `OperatorName.parameter_name` and the `value` is a comma separated list of values. In our case, the operator is named "Training" and the parameters are *degree* and *epsilon*. This leads to the following xml fragment:

```
<list key="parameters">
  <parameter key="Training.degree" value="1,2,3,4"/>
  <parameter key="Training.epsilon" value="0.01,0.03,0.05,0.1"/>
</list>
```

Figure 4.4 shows the entire example process setup.

In GUI mode you do not have to bother about the XML code, just click on the Edit List button next to the *parameters* parameter of the GridParameterOptimization operator and add the two parameters to the list.

If the value lists hold n_1 and n_2 values, respectively, the GridParameterOptimization will apply its inner operators $n_1 \cdot n_2$ times. Finally the GridParameterOptimization operator returns an optimal parameter value combination and the best performance vector. If this is desired, the optimal parameter set can be written to a file (for a specification see section 3.4.4) and reread from another process using a ParameterSetLoader (see section 5.3.34) and set using a ParameterSetter (see section 5.5.18).

In order to create a chart showing the absolute error over the parameters d and ε, we use the ProcessLog operator. Each time this operator is applied, it creates a record containing a set of data that we can specify. If the operator is applied n times and we specify m parameters, we have a table with n rows and m columns at the end of the process. Various plots and charts may be generated from this table.

Similar to the optimization operator, the ProcessLog operator accepts a `<list>` of parameters specifying the values that should be recorded. This list has the key *log*. In our case, we are interested in three values: the values of the parameters *degree* and *epsilon* and in the performance of the models generated with these parameters. Therefore, we add one `<parameter>` tag to the *log* parameter `<list>` for each value we are interested in. (Again, in GUI mode, simply click on the Edit List button next to the *log* parameter of the

```
<operator name="Global" class="Process">
  <parameter key="logfile"          value="advanced4.log"/>

  <operator name="Input" class="ExampleSource">
    <parameter key="attributes" value="data/polynomial.aml"/>
  </operator>

  <operator name="GridParameterOptimization" class="ParameterOptimization">

    <list key="parameters">
      <parameter key="Learner.epsilon" value="0.01,0.03,0.05,0.075,0.1"/>
      <parameter key="Learner.degree" value="1,2,3,4"/>
    </list>

    <operator name="Validation" class="SimpleValidation">
      <parameter key="split_ratio" value="0.5"/>

      <operator name="Learner" class="LibSVMLearner">
        <parameter key="kernel_type" value="poly"/>
      </operator>
      <operator name="ApplierChain" class="OperatorChain">
        <operator name="Applier" class="ModelApplier"/>
        <operator name="Evaluator" class="PerformanceEvaluator">
          <parameter key="absolute_error"       value="true"/>
          <parameter key="main_criterion" value="absolute_error"/>
        </operator>
      </operator>
    </operator>

    <operator name="ProcessLog" class="ProcessLog">
      <parameter key="filename" value="svm_degree_epsilon.log"/>
        <list key="log">
          <parameter key="degree"
                 value="operator.Learner.parameter.degree"/>
          <parameter key="epsilon"
                 value="operator.Learner.parameter.epsilon"/>
          <parameter key="absolute"
                 value="operator.Validation.value.performance"/>
        </list>
    </operator>
  </operator>
</operator>
```

Figure 4.4: Parameter and performance analysis

ProcessLog operator.) The keys of the parameters nested in this list may have arbitrary values. They are used as column names and labels in charts only. We choose “degree”, “epsilon”, and “performance”. The value of the parameters specifies, how to retrieve the values. They are of the form

operator.OperatorName.{parameter|value}.Name[2]

Two types of values can be recorded:

1. parameters that are specified by the process configuration or varied by the GridParameterOptimization operator and
2. values that are generated or measured in the course of the process.

degree and *epsilon* are parameters of the operator named “Training”. The performance is a value generated by the operator named “XValidation”. Hence, our parameter list looks like this:

```
<list key="log">
  <parameter key="degree"
     value="operator.Training.parameter.degree"/>
  <parameter key="epsilon"
     value="operator.Training.parameter.epsilon"/>
  <parameter key="performance"
     value="operator.XValidation.value.performance"/>
</list>
```

For a list of values that are provided by the individual operators, please refer to the operator reference (chapter 5).

Some plots may be generated online by using the GUI. This includes color and 3D plots like the one shown in figure 4.5.

4.4 Support and tips

RapidMiner is a complex data mining suite and provides a platform for a large variety of process designs. We suggest that you work with some of the building blocks described in this chapter and replace some operators and parameter settings. You should have a look at the sample process definitions delivered with RapidMiner and learn about other operators. However, the complexity

[2] If you wonder why this string starts with the constant prefix “operator”, this is because it is planned to extend the ProcessLog operator by the possibility to log values taken from an input object passed to the operator.

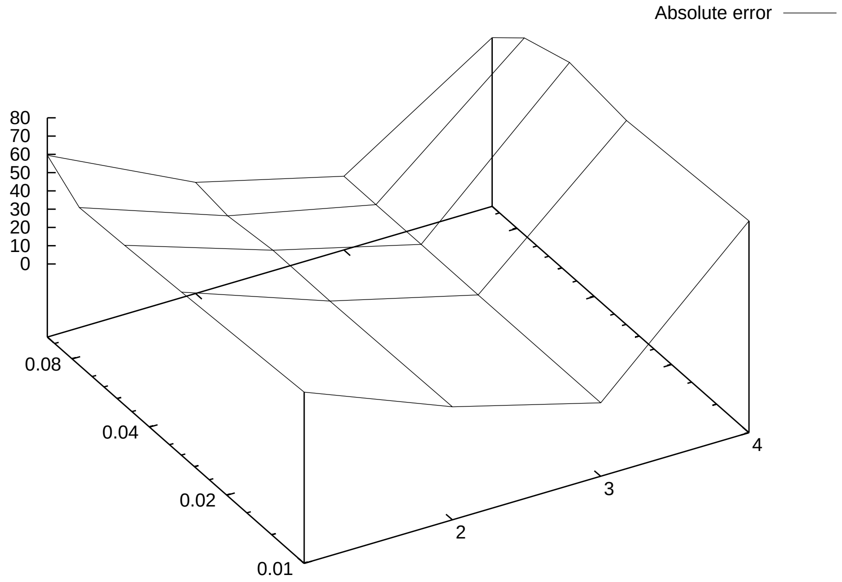


Figure 4.5: The performance of a SVM (plot generated by `gnuplot`)

of RapidMiner might sometimes be very frustrating if you cannot manage to design the data mining processes you want to. Please do not hesitate to use the user forum and ask for help. You can also submit a support request. Both user forum and support request tracker are available on our website

http://www.rapidminer.com/

Beside this, we also offer services like support and consulting for our professional users. Please contact us if you are interested in this form of professional support.

We conclude this chapter with some tips:

- You should make use of the automatic process validation available in the graphical user interface. This avoids a wrong process setup, missing parameter values, etc.

- Work on a small subsample of your data during the process design and switch to the complete dataset if you are sure the process will properly run.

- You do not have to write the attribute description files (XML) by hand. Just use the Attribute Editor of the GUI version or the configuration wizard of the ExampleSource operator.

- Make use of breakpoints in the design phase. This helps to understand the data flow of RapidMiner and find potential problems, etc.
- Start with small process setups and known building blocks and check if each new operator / operator chain performs in the way you have expected.

Chapter 5

Operator reference

This chapter describes the built-in operators that come with RapidMiner. Each operator section is subdivided into several parts:

1. The group and the icon of the operator.

2. An enumeration of the required input and the generated output objects. The input objects are usually consumed by the operator and are not part of the output. In some cases this behaviour can be changed by using a parameter `keep_...`. Operators may also receive more input objects than required. In that case the unused input objects will be appended to the output and can be used by the next operator.

3. The parameters that can be used to configure the operator. Ranges and default values are specified. Required parameters are indicated by bullets (•) and optional parameters are indicated by an open bullet (◦)

4. A list of values that can be logged using the ProcessLog operator (see page 543).

5. If the operator represents a learning scheme, the capabilities of the learner are described. The learning capapabilities of most meta learning schemes depend on the inner learner.

6. If the operator represents an operator chain a short description of the required inner operators is given.

7. A short and a long textual description of the operator.

The reference is divided into sections according to the operator groups known from the graphical user interface. Within each section operators are alphabetically listed.

5.1 Basic operators

5.1.1 ModelApplier

Required input:	**Generated output:**
• Model • ExampleSet	• ExampleSet

Parameters:

- ○ **keep_model:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **application_parameters:** Model parameters for application (usually not needed). (list)
- ○ **create_view:** Indicates that preprocessing models should only create a new view on the data instead of actually changing the data itself. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Applies a model to an example set. This might be a prediction or another data transformation model.

Description: This operator applies a Model to an ExampleSet. All parameters of the training process should be stored within the model. However, this operator is able to take any parameters for the rare case that the model can use some parameters during application. Models can be read from a file by using a ModelLoader (see section 5.3.30).

5.1.2 ModelGrouper

Required input:	**Generated output:**
• Model	• Model

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Groups the input models into a single combined model which might be necessary for example for applying preprocessing models.

Description: This operator groups all input models together into a grouped (combined) model. This model can be completely applied on new data or written into a file as once. This might become useful in cases where preprocessing and prediction models should be applied together on new and unseen data.

This operator replaces the automatic model grouping known from previous versions of RapidMiner. The explicit usage of this grouping operator gives the user more control about the grouping procedure. A grouped model can be ungrouped with the ModelUngrouper (see section 5.1.3) operator.

5.1.3 ModelUngrouper

Required input:

- Model

Generated output:

- Model

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Ungroups a previously grouped model into the single models which might then be handled on their own.

Description: This operator ungroups a previously grouped model (ModelGrouper (see section 5.1.2)) and delivers the grouped input models.

This operator replaces the automatic model grouping known from previous versions of RapidMiner. The explicit usage of this ungrouping operator gives the user more control about the ungrouping procedure. Single models can be grouped with the ModelGrouper (see section 5.1.2) operator.

5.1.4 ModelUpdater

Required input:

- Model
- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Updates a model according to an example set. Please note that this operator can only be used for updatable models, otherwise an error will be shown.

Description: This operator updates a Model with an ExampleSet. Please note that the model must return true for Model in order to be usable with this operator.

5.1.5 OperatorChain

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: A chain of operators that is subsequently applied.

Description: A simple operator chain which can have an arbitrary number of inner operators. The operators are subsequently applied and their output is used as input for the succeeding operator. The input of the operator chain is used as input for the first inner operator and the output of the last operator is used as the output of the operator chain.

5.2 Core operators

The operators described in this section are basic operators in a sense that they are used in many process definitions without being specific to a certain group of operators.

5.2.1 CommandLineOperator

Group: Core

Parameters:

- **command:** Command to execute. (string)
- ○ **log_stdout:** If set to true, the stdout stream of the command is redirected to the logfile. (boolean; default: true)
- ○ **log_stderr:** If set to true, the stderr stream of the command is redirected to the logfile. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator simply executes a command in a shell of the underlying operating system, basically any system command or external program.

Description: This operator executes a system command. The command and all its arguments are specified by the parameter `command`. The standard output stream and the error stream of the process can be redirected to the logfile.

Please note also that the command is system dependent. Characters that have special meaning on the shell like e.g. the pipe symbol or brackets and braces do not have a special meaning to Java.

The method `Runtime.exec(String)` is used to execute the command. Please note, that this (Java) method parses the string into tokens before it is executed. These tokens are *not* interpreted by a shell (which?). If the desired command involves piping, redirection or other shell features, it is best to create a small shell script to handle this.

5.2.2 Experiment

Group: Core

Please use the operator 'Process' instead.

Parameters:

- **logverbosity:** Log verbosity level.
- **logfile:** File to write logging information to. (filename)
- **resultfile:** File to write inputs of the ResultWriter operators to. (filename)
- **random_seed:** Global random seed for random generators (-1 for initialization by system time). (integer; -2147483648-+∞; default: 2001)
- **notification_email:** Email address for the notification mail. (string)
- **encoding:** The encoding of the process XML description. (string; default: 'SYSTEM')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **memory:** The current memory usage.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: The root operator chain, which needs to be the outer most operator of any experiment.

Description: Each process must contain exactly one operator of this class and it must be the root operator of the process. The only purpose of this operator is to provide some parameters that have global relevance.

5.2.3 FileEcho

Group: Core

Parameters:

- **file:** The file into which this operator should write the specified text. (filename)
- **text:** The text which should be written into the file. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator simply writes the given text into the specified file (can be useful in combination with a process branch).

Description: This operator simply writed the specified text into the specified file. This can be useful in combination with the ProcessBranch (see section 5.5.20) operator. For example, one could write the success or non-success of a process into the same file depending on the condition specified by a process branch.

5.2.4 IOConsumer

Group: Core

Parameters:

- ○ **io_object:** The class of the object(s) which should be removed.
- ○ **deletion_type:** Defines the type of deletion.
- ○ **delete_which:** Defines which input object should be deleted (only used for deletion type 'delete_one'). (integer; 1-+∞; default: 1)
- ○ **except:** Defines which input object should not be deleted (only used for deletion type 'delete_one_but_number'). (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operators simply consumes some unused outputs.

Description: Most RapidMiner operators should define their desired input and delivered output in a senseful way. In some cases operators can produce additional output which is indicated with a boolean parameter. Other operators are able to deliver their input as output instead of consuming it (parameter keep_....). However, in some cases it might be usefull to delete unwanted output to ensure that following operators use the correct input object. Furthermore, some operators produce additional unneeded and therefore unconsumed output. In an iterating operator chain this unneeded output will grow with each iteration. Therefore, the IOConsumeOperator can be used to delete one (the n-th) object of a given type (indicated by delete_one), all input objects of a given type (indicated by delete_all), all input objects but those of a given type (indicated by delete_all_but), or all input objects of the given type except for the n-th object of the type.

5.2.5 IOMultiplier

Group: Core

Parameters:

- **number_of_copies:** The number of copies which should be created. (integer; 1-+∞; default: 1)
- **io_object:** The class of the object(s) which should be multiplied.
- **multiply_type:** Defines the type of multiplying.
- **multiply_which:** Defines which input object should be multiplied (only used for deletion type 'multiply_one'). (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operators simply multiplies selected input objects.

Description: In some cases you might want to apply different parts of the process on the same input object. You can use this operator to create `k` copies of the given input object.

5.2.6 IOSelector

Group: Core

Generated output:

- IOObject

Parameters:

- **io_object:** The class of the object(s) which should be removed.
- **select_which:** Defines which input object should be selected. (integer; 1-+∞; default: 1)
- **delete_others:** Indicates if the other non-selected objects should be deleted. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operators simply selects one of the input objects of the specified type and discards the rest.

Description: This operator allows to choose special IOObjects from the given input. Bringing an IOObject to the front of the input queue allows the next operator to directly perform its action on the selected object. Please note that counting for the parameter value starts with one, but usually the IOObject which was added at last gets the number one, the object added directly before get number two and so on.

The user can specify with the parameter delete_others what will happen to the non-selected input objects of the specified type: if this parameter is set to true, all other IOObjects of the specified type will be removed by this operator. Otherwise (default), the objects will all be kept and the selected objects will just be brought into front.

5.2.7 MacroDefinition

Group: Core

Parameters:

- ○ **macros:** The list of macros defined by the user. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to define arbitrary macros which can be used by

Description: (Re-)Define macros for the current process. Macros will be replaced in the value strings of parameters by the macro values defined in the parameter list of this operator. Please note that this features is basically only supported for string type parameter values (strings or files) and not for numerical or list types.

In the parameter list of this operator, you have to define the macro name (without the enclosing brackets) and the macro value. The defined macro can then be used in all succeeding operators as parameter value for string type parameters. A macro must then be enclosed by "% {" and "}".

There are several predefined macros:

- % {process_name}: will be replaced by the name of the process (without path and extension)
- % {process_file}: will be replaced by the file name of the process (with extension)
- % {process_path}: will be replaced by the complete absolute path of the process file

In addition to those the user might define arbitrary other macros which will be replaced by arbitrary string during the process run. Please note also that several other short macros exist, e.g. % {a} for the number of times the current operator was applied. Please refer to the section about macros in the RapidMiner tutorial. Please note also that other operators like the FeatureIterator (see section 5.5.8) also adds specific macros.

5.2.8 MaterializeDataInMemory

Group: Core

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **datamanagement:** Determines, how the data is represented internally.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a fresh and clean copy of the data. Might be useful after large preprocessing chains with a lot of views or even data copies, especially in combination with a memory clean up operation followed afterwards.

Description: Creates a fresh and clean copy of the data in memory. Might be very useful in combination with the MemoryCleanUp (see section 5.2.9) operator after large preprocessing trees using lot of views or data copies.

5.2.9 MemoryCleanUp

Group: Core

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Frees unused memory. Might be useful after large preprocessing chains with a lot of (now) unused views or even data copies. Can be very useful after the data set was (again) materialized in memory.

Description: Cleans up unused memory resources. Might be very useful in combination with the MaterializeDataInMemory (see section 5.2.8) operator after large preprocessing trees using lot of views or data copies. Internally, this operator simply invokes a garbage collection from the underlying Java programming language.

5.2.10 Process

Group: Core

Parameters:

- **logverbosity:** Log verbosity level.
- **logfile:** File to write logging information to. (filename)
- **resultfile:** File to write inputs of the ResultWriter operators to. (filename)
- **random_seed:** Global random seed for random generators (-1 for initialization by system time). (integer; -2147483648-+∞; default: 2001)
- **notification_email:** Email address for the notification mail. (string)
- **encoding:** The encoding of the process XML description. (string; default: 'SYSTEM')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **memory:** The current memory usage.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: The root operator chain, which needs to be the outer most operator of any process.

Description: Each process must contain exactly one operator of this class and it must be the root operator of the process. The only purpose of this operator is to provide some parameters that have global relevance.

5.2.11 SingleMacroDefinition

Group: Core

Parameters:

- **macro:** The macro name defined by the user. (string)
- **value:** The macro value defined by the user. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to define a single arbitrary macro which can be used by

Description: (Re-)Define macros for the current process. Macros will be replaced in the value strings of parameters by the macro values defined as a parameter of this operator. Please note that this features is basically only supported for string type parameter values (strings or files) and not for numerical or list types. In contrast to the usual MacroDefinitionOperator, this operator only supports the definition of a single macro and can hence be used inside of parameter iterations.

You have to define the macro name (without the enclosing brackets) and the macro value. The defined macro can then be used in all succeeding operators as parameter value for string type parameters. A macro must then be enclosed by “% {“ and “}“.

There are several predefined macros:

- % {process_name}: will be replaced by the name of the process (without path and extension)
- % {process_file}: will be replaced by the file name of the process (with extension)
- % {process_path}: will be replaced by the complete absolute path of the process file

In addition to those the user might define arbitrary other macros which will be replaced by arbitrary string during the process run. Please note also that several other short macros exist, e.g. % {a} for the number of times the current operator was applied. Please refer to the section about macros in the RapidMiner tutorial.

5.3 Input/Output operators

The operators described in this section deal with input and output of all kinds of results and in-between results. Models, example sets, attribute sets and parameter sets can be read from and written to disc. Hence, it is possible to split up process into, for instance, a training process setup and a evaluation or application process setup.

5.3.1 ArffExampleSetWriter

Group: IO.Examples

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **example_set_file:** File to save the example set to. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes the values of all examples into an ARFF-file.

Description: Writes values of all examples into an ARFF file which can be used by the machine learning library Weka. The ARFF format is described in the ArffExampleSource (see section 5.3.2) operator which is able to read ARFF files to make them usable with RapidMiner.

5.3.2 ArffExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **data_file:** The path to the data file. (filename)
- ○ **label_attribute:** The (case sensitive) name of the label attribute (string)
- ○ **id_attribute:** The (case sensitive) name of the id attribute (string)
- ○ **weight_attribute:** The (case sensitive) name of the weight attribute (string)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)
- ○ **sample_size:** The exact number of samples which should be read (-1 = use sample ratio; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (only for permutation, -1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read arff files.

Description: This operator can read ARFF files known from the machine learning library Weka. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software.

ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information. The Header of the ARFF file contains the name of the relation (@RELATION, ignored by RapidMiner) and a list of the attributes, each of which is defined by a starting @ATTRIBUTE followed by its name and its type.

Attribute declarations take the form of an orderd sequence of @ATTRIBUTE statements. Each attribute in the data set has its own @ATTRIBUTE statement which uniquely defines the name of that attribute and it's data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared all that attributes values will be found in the third comma delimited column.

The possible attribute types are:

- `numeric`
- `integer`
- `real`
- `nominalValue1,nominalValue2,...` for nominal attributes
- `string` for nominal attributes without distinct nominal values (it is however recommended to use the nominal definition above as often as possible)
- `date [date-format]` (currently not supported by RapidMiner)

Valid examples for attribute definitions are

`@ATTRIBUTE petalwidth REAL`

`@ATTRIBUTE class Iris-setosa,Iris-versicolor,Iris-virginica`

The ARFF Data section of the file contains the data declaration line @DATA followed by the actual example data lines. Each example is represented on a single line, with carriage returns denoting the end of the example. Attribute values for each example are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the n-th @ATTRIBUTE declaration is always the n-th field of the example line). Missing values are represented by a single question mark, as in:

`4.4,?,1.5,?,Iris-setosa`

A percent sign (names or example values containing spaces must be quoted with single quotes ('). Please note that the sparse ARFF format is currently only supported for numerical attributes. Please use one of the other options for sparse data files provided by RapidMiner if you also need sparse data files for nominal attributes.

Please have a look at the Iris example ARFF file provided in the data subdirectory of the sample directory of RapidMiner to get an idea of the described data format.

5.3.3 AttributeConstructionsLoader

Group: IO.Attributes

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attribute_constructions_file:** Filename for the attribute constructions file. (filename)
- ○ **keep_all:** If set to true, all the original attributes are kept, otherwise they are removed from the example set. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Loads all attributes of an example set from a file. Each line holds the construction description of one attribute.

Description: Loads an attribute set from a file and constructs the desired features. If keep_all is false, original attributes are deleted before the new ones are created. This also means that a feature selection is performed if only a subset of the original features was given in the file.

5.3.4 AttributeConstructionsWriter

Group: IO.Attributes

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attribute_constructions_file:** Filename for the attribute construction description file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes all attributes of an example set to a file. Each line holds the construction description of one attribute.

Description: Writes all attributes of an example set to a file. Each line holds the construction description of one attribute. This file can be read in another process using the FeatureGeneration (see section 5.8.42) or AttributeConstructionsLoader (see section 5.3.3).

5.3.5 AttributeWeightsLoader

Group: IO.Attributes

Generated output:

- AttributeWeights

Parameters:

- **attribute_weights_file:** Filename of the attribute weights file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads the weights of all attributes of an example set from a file. Each line must hold the name and the weight of one attribute.

Description: Reads the weights for all attributes of an example set from a file and creates a new AttributeWeights IOObject. This object can be used for scaling the values of an example set with help of the AttributeWeightsApplier (see section 5.8.12) operator.

5.3.6 AttributeWeightsWriter

Group: IO.Attributes

Required input:

- AttributeWeights

Generated output:

- AttributeWeights

Parameters:

- **attribute_weights_file:** Filename for the attribute weight file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes the weights of all attributes of an example set to a file. Each line holds the name and the weight of one attribute.

Description: Writes the weights of all attributes of an example set to a file. Therefore a AttributeWeights object is needed in the input of this operator. Each line holds the name of one attribute and its weight. This file can be read in another process using the AttributeWeightsLoader (see section 5.3.5) and the AttributeWeightsApplier (see section 5.8.12).

5.3.7 BibtexExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- ○ **label_attribute:** The (case sensitive) name of the label attribute (string)
- ○ **id_attribute:** The (case sensitive) name of the id attribute (string)
- ○ **weight_attribute:** The (case sensitive) name of the weight attribute (string)

- ◦ **datamanagement:** Determines, how the data is represented internally.
- **data_file:** The file containing the data (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read BibTeX files.

Description: This operator can read BibTeX files. It uses Stefan Haustein's kdb tools.

5.3.8 C45ExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **c45_filestem:** The path to either the C4.5 names file, the data file, or the filestem (without extensions). Both files must be in the same directory. (filename)
- ◦ **datamanagement:** Determines, how the data is represented internally.
- ◦ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read data and meta given in C4.5 format.

Description: Loads data given in C4.5 format (names and data file). Both files must be in the same directory. You can specify one of the C4.5 files (either the data or the names file) or only the filestem.

For a dataset named "foo", you will have two files: foo.data and foo.names. The .names file describes the dataset, while the .data file contains the examples which make up the dataset.

The files contain series of identifiers and numbers with some surrounding syntax. A | (vertical bar) means that the remainder of the line should be ignored as a comment. Each identifier consists of a string of characters that does not include comma, question mark or colon. Embedded whitespce is also permitted but multiple whitespace is replaced by a single space.

The .names file contains a series of entries that describe the classes, attributes and values of the dataset. Each entry can be terminated with a period, but the period can be omited if it would have been the last thing on a line. The first entry in the file lists the names of the classes, separated by commas. Each successive line then defines an attribute, in the order in which they will appear in the .data file, with the following format:

```
attribute-name : attribute-type
```

The attribute-name is an identifier as above, followed by a colon, then the attribute type which must be one of

- `continuous` If the attribute has a continuous value.
- `discrete [n]` The word 'discrete' followed by an integer which indicates how many values the attribute can take (not recommended, please use the method depicted below for defining nominal attributes).
- `[list of identifiers]` This is a discrete, i.e. nominal, attribute with the values enumerated (this is the prefered method for discrete attributes). The identifiers should be separated by commas.
- `ignore` This means that the attribute should be ignored - it won't be used. This is not supported by RapidMiner, please use one of the attribute selection operators after loading if you want to ignore attributes and remove them from the loaded example set.

Here is an example .names file:

```
good, bad.
```

```
dur: continuous.
wage1: continuous.
wage2: continuous.
wage3: continuous.
cola: tc, none, tcf.
hours: continuous.
pension: empl\_contr, ret\_allw, none.
stby\_pay: continuous.
shift\_diff: continuous.
educ\_allw: yes, no.
...
```

Foo.data contains the training examples in the following format: one example per line, attribute values separated by commas, class last, missing values represented by "?". For example:

```
2,5.0,4.0,?,none,37,?,?,5,no,11,below\_average,yes,full,yes,full,good
3,2.0,2.5,?,?,35,none,?,?,?,10,average,?,?,yes,full,bad
3,4.5,4.5,5.0,none,40,?,?,?,no,11,average,?,half,?,?,good
3,3.0,2.0,2.5,tc,40,none,?,5,no,10,below\_average,yes,half,yes,full,bad
...
```

5.3.9 CSVExampleSetWriter

Group: IO.Examples

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **csv_file:** The CSV file which should be written. (filename)
- ○ **column_separator:** The column separator. (string; default: ';')
- ○ **write_attribute_names:** Indicates if the attribute names should be written as first row. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can write csv files.

Description: This operator can be used to write data into CSV files (Comma Separated Values). The values and columns are separated by “;“. Missing data values are indicated by empty cells.

5.3.10 CSVExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **filename:** Name of the file to read the data from. (filename)
- ○ **read_attribute_names:** Read attribute names from file (assumes the attribute names are in the first line of the file). (boolean; default: true)
- ○ **label_name:** Name of the label attribute (if empty, the column defined by label_column will be used) (string)
- ○ **label_column:** Column number of the label attribute (only used if label_name is empty; 0 = none; negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **id_name:** Name of the id attribute (if empty, the column defined by id_column will be used) (string)
- ○ **id_column:** Column number of the id attribute (only used if id_name is empty; 0 = none; negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **weight_name:** Name of the weight attribute (if empty, the column defined by weight_column will be used) (string)
- ○ **weight_column:** Column number of the weight attribute (only used if weight_name is empty; 0 = none, negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)

- **sample_size:** The exact number of samples which should be read (-1 = use sample ratio; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)
- **datamanagement:** Determines, how the data is represented internally.
- **column_separators:** Column separators for data files (regular expression) (string; default: ',\s*|;\s*')
- **use_comment_characters:** Indicates if qa comment character should be used. (boolean; default: true)
- **comment_chars:** Lines beginning with these characters are ignored. (string; default: '#')
- **use_quotes:** Indicates if quotes should be regarded (slower!). (boolean; default: true)
- **trim_lines:** Indicates if lines should be trimmed (empty spaces are removed at the beginning and the end) before the column split is performed. (boolean; default: false)
- **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read csv files.

Description: This operator can read csv files. All values must be separated by “,“, by “;“, or by white space like tabs. The first line is used for attribute names as default.

For other file formats or column separators you can use in almost all cases the operator SimpleExampleSource (see section 5.3.40) or, if this is not sufficient, the operator ExampleSource (see section 5.3.21).

5.3.11 CachedDatabaseExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- ◦ **configure_operator:** Configure this operator by means of a Wizard.
- ◦ **database_system:** Indicates the used database system
- **database_url:** The complete URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database' (string)
- **username:** Database username. (string)
- ◦ **password:** Password for the database. (password)
- ◦ **query:** SQL query. If not set, the query is read from the file specified by 'query_file'. (string)
- ◦ **query_file:** File containing the query. Only evaluated if 'query' is not set. (filename)
- ◦ **table_name:** Use this table if work_on_database is true or no other query is specified. (string)
- ◦ **label_attribute:** The (case sensitive) name of the label attribute (string)
- ◦ **id_attribute:** The (case sensitive) name of the id attribute (string)
- ◦ **weight_attribute:** The (case sensitive) name of the weight attribute (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator reads an example set from an SQL database by incrementally caching it (recommended).

Description: This operator reads an ExampleSet from an SQL database. The SQL query can be passed to RapidMiner via a parameter or, in case of long SQL statements, in a separate file. Please note that column names are often case sensitive. Databases may behave differently here.

The most convenient way of defining the necessary parameters is the configuration wizard. The most important parameters (database URL and user name) will be automatically determined by this wizard and it is also possible to define the special attributes like labels or ids.

Please note that this operator supports two basic working modes:

1. reading the data from the database and creating an example table in main memory

2. keeping the data in the database and directly working on the database table

The latter possibility will be turned on by the parameter "work_on_database". Please note that this working mode is still regarded as experimental and errors might occur. In order to ensure proper data changes the database working mode is only allowed on a single table which must be defined with the parameter "table_name". IMPORTANT: If you encounter problems during data updates (e.g. messages that the result set is not updatable) you probably have to define a primary key for your table.

If you are not directly working on the database, the data will be read with an arbitrary SQL query statement (SELECT ... FROM ... WHERE ...) defined by "query" or "query_file". The memory mode is the recommended way of using this operator. This is especially important for following operators like learning schemes which would often load (most of) the data into main memory during the learning process. In these cases a direct working on the database is not recommended anyway.

Warning As the java `ResultSetMetaData` interface does not provide information about the possible values of nominal attributes, the internal indices the nominal values are mapped to will depend on the ordering they appear in the table. This may cause problems only when processes are split up into a training process and an application or testing process. For learning schemes which are capable of handling nominal attributes, this is not a problem. If a learning scheme like a SVM is used with nominal data, RapidMiner pretends that nominal attributes are numerical and uses indices for the nominal values as their numerical value. A SVM may perform well if there are only two possible values. If a test set is read in another process, the nominal values may be assigned different indices, and hence the SVM trained is useless. This is not a problem for label attributes, since the classes can be specified using the `classes` parameter and hence, all learning schemes intended to use with nominal data are safe to use.

5.3.12 ChurnReductionExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

◦ **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)

◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates data for testing purposes based on a churn reduction data set.

Description: Generates a random example set for testing purposes. The data represents a direct mailing example set.

5.3.13 ClusterModelReader

Group: IO.Clustering

Generated output:

- HierarchicalClusterModel

Parameters:

- **cluster_model_file:** the file from which the cluster model is read (filename)

◦ **flat:** load a flat model or flatten it (boolean; default: false)

  - ○ **add_ids:** if true, new ids are generated for each cluster model, otherwise, the ids in the file are used (boolean; default: false)
  - ○ **convert_labels:** if true, all non-letter characters are replaced in cluster descriptions (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads a single cluster model from a file.

Description: Reads a single cluster model from a file.

5.3.14 ClusterModelWriter

Group: IO.Clustering

Required input:	**Generated output:**
• ClusterModel	• ClusterModel

Parameters:

- **cluster_model_file:** the file to which the cluster model is stored (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes a cluster model to a file.

Description: Write a single cluster model to a file.

5.3.15 DBaseExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

○ **label_attribute:** The (case sensitive) name of the label attribute (string)
○ **id_attribute:** The (case sensitive) name of the id attribute (string)
○ **weight_attribute:** The (case sensitive) name of the weight attribute (string)
○ **datamanagement:** Determines, how the data is represented internally.
• **data_file:** The file containing the data (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read dBase files.

Description: This operator can read dbase files. It uses Stefan Haustein's kdb tools.

5.3.16 DatabaseExampleSetWriter

Group: IO.Examples

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **configure_operator:** Configure this operator by means of a Wizard.
○ **database_system:** Indicates the used database system

- **database_url:** The complete URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database' (string)
- **username:** Database username. (string)

○ **password:** Password for the database. (password)

○ **table_name:** Use this table if work_on_database is true or no other query is specified. (string)

○ **overwrite_mode:** Indicates if an existing table should be overwritten or if data should be appended.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes the values of all examples to a single table in a database.

Description: This operator writes an ExampleSet into an SQL database. The user can specify the database connection and a table name. Please note that the table will be created during writing if it does not exist.

The most convenient way of defining the necessary parameters is the configuration wizard. The most important parameters (database URL and user name) will be automatically determined by this wizard. At the end, you only have to define the table name and then you are ready.

This operator only supports the writing of the complete example set consisting of all regular and special attributes and all examples. If this is not desired perform some preprocessing operators like attribute or example filter before applying this operator.

5.3.17 DatabaseExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- ○ **configure_operator:** Configure this operator by means of a Wizard.
- ○ **work_on_database:** (EXPERIMENTAL!) If set to true, the data read from the database is NOT copied to main memory. All operations that change data will modify the database. (boolean; default: false)
- ○ **database_system:** Indicates the used database system
- • **database_url:** The complete URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database' (string)
- • **username:** Database username. (string)
- ○ **password:** Password for the database. (password)
- ○ **query:** SQL query. If not set, the query is read from the file specified by 'query_file'. (string)
- ○ **query_file:** File containing the query. Only evaluated if 'query' is not set. (filename)
- ○ **table_name:** Use this table if work_on_database is true or no other query is specified. (string)
- ○ **label_attribute:** The (case sensitive) name of the label attribute (string)
- ○ **id_attribute:** The (case sensitive) name of the id attribute (string)
- ○ **weight_attribute:** The (case sensitive) name of the weight attribute (string)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **classes:** Whitespace separated list of possible class values of the label attribute. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator reads an example set from an SQL database.

Description: This operator reads an ExampleSet from an SQL database. The SQL query can be passed to RapidMiner via a parameter or, in case of long SQL statements, in a separate file. Please note that column names are often case sensitive. Databases may behave differently here.

The most convenient way of defining the necessary parameters is the configuration wizard. The most important parameters (database URL and user name) will be automatically determined by this wizard and it is also possible to define the special attributes like labels or ids.

Please note that this operator supports two basic working modes:

1. reading the data from the database and creating an example table in main memory

2. keeping the data in the database and directly working on the database table

The latter possibility will be turned on by the parameter "work_on_database". Please note that this working mode is still regarded as experimental and errors might occur. In order to ensure proper data changes the database working mode is only allowed on a single table which must be defined with the parameter "table_name". IMPORTANT: If you encounter problems during data updates (e.g. messages that the result set is not updatable) you probably have to define a primary key for your table.

If you are not directly working on the database, the data will be read with an arbitrary SQL query statement (SELECT ... FROM ... WHERE ...) defined by "query" or "query_file". The memory mode is the recommended way of using this operator. This is especially important for following operators like learning schemes which would often load (most of) the data into main memory during the learning process. In these cases a direct working on the database is not recommended anyway.

Warning As the java `ResultSetMetaData` interface does not provide information about the possible values of nominal attributes, the internal indices the nominal values are mapped to will depend on the ordering they appear in the table. This may cause problems only when processes are split up into a training process and an application or testing process. For learning schemes which are capable of handling nominal attributes, this is not a problem. If a learning scheme like a SVM is used with nominal data, RapidMiner pretends that nominal attributes are numerical and uses indices for the nominal values as their numerical value. A SVM may perform well if there are only two possible values. If a test set is read in another process, the nominal values may be assigned different indices, and hence the SVM trained is useless. This is not a problem for label attributes, since the classes can be specified using the `classes` parameter and hence, all learning schemes intended to use with nominal data are safe to use.

5.3.18 DirectMailingExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- ◦ **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates data for testing purposes based on a direct mailing data set.

Description: Generates a random example set for testing purposes. The data represents a direct mailing example set.

5.3.19 ExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- **target_function:** Specifies the target function of this example set
- ◦ **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)

- ○ **number_of_attributes:** The number of attributes. (integer; 1-+∞; default: 5)
- ○ **attributes_lower_bound:** The minimum value for the attributes. (real; -∞-+∞)
- ○ **attributes_upper_bound:** The maximum value for the attributes. (real; -∞-+∞)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **datamanagement:** Determines, how the data is represented internally.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates an example set based on numerical attributes.

Description: Generates a random example set for testing purposes. Uses a subclass of `TargetFunction` to create the examples from the attribute values. Possible target functions are: random, sum (of all attributes), polynomial (of the first three attributes, degree 3), non linear, sinus, sinus frequency (like sinus, but with frequencies in the argument), random classification, sum classification (like sum, but positive for positive sum and negative for negative sum), interaction classification (positive for negative x or positive y and negative z), sinus classification (positive for positive sinus values).

5.3.20 ExampleSetWriter

Group: IO.Examples

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **example_set_file:** File to save the example set to. (filename)
- ○ **attribute_description_file:** File to save the attribute descriptions to. (filename)

- **format:** Format to use for output.
- **special_format:** Format string to use for output. (string)
- **fraction_digits:** The number of fraction digits in the output file (-1: all possible digits). (integer; -1-+∞; default: -1)
- **quote_whitespace:** Indicates if nominal values containing whitespace characters should be quoted with double quotes. (boolean; default: true)
- **zipped:** Indicates if the data file content should be zipped. (boolean; default: false)
- **append:** Indicates if the data should be appended to an possible existing data file. Otherwise the existing file will be overwritten. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes the values of all examples to a file.

Description: Writes values of all examples in an ExampleSet to a file. Dense, sparse, and user defined formats (specified by the parameter 'format') can be used. Attribute description files may be generated for dense and sparse format as well. These formats can be read using the ExampleSource (see section 5.3.21) and SparseFormatExampleSource (see section 5.3.41) operators.

dense: Each line of the generated data file is of the form

```
regular attributes <special attributes>
```

For example, each line could have the form

```
value1 value2 ... valueN <id> <label> <prediction> ... <confidences>
```

Values in parenthesis are optional and are only printed if they are available. The confidences are only given for nominal predictions. Other special attributes might be the example weight or the cluster number.

sparse: Only non 0 values are written to the file, prefixed by a column index. See the description of SparseFormatExampleSource (see section 5.3.41) for details.

special: Using the parameter 'special_format', the user can specify the exact format. The $ sign has a special meaning and introduces a command (the following character) Additional arguments to this command may be supplied enclosing it in square brackets.

- **$a:** All attributes separated by the default separator
- **$a[separator]:** All attributes separated by separator
- **$s[separator][indexSeparator]:** Sparse format. For all non zero attributes the following strings are concatenated: the column index, the value of indexSeparator, the attribute value. Attributes are separated by separator.
- **$v[name]:** The value of the attribute with the given name (both regular and special attributes)
- **$k[index]:** The value of the attribute with the given index
- **$l:** The label
- **$p:** The predicted label
- **$d:** All prediction confidences for all classes in the form conf(class)=value
- **$d[class]:** The prediction confidence for the defined class as a simple number
- **$i:** The id
- **$w:** The weight
- **$b:** The batch number
- **$n:** The newline character
- **$t:** The tabulator character
- **$$:** The dollar sign
- **$[:** The '[' character
- **$]:** The ']' character

Make sure the format string ends with $n if you want examples to be separated by newlines!

5.3.21 ExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- ○ **configure_operator:** Configure this operator by means of a Wizard.
- • **attributes:** Filename for the xml attribute description file. This file also contains the names of the files to read the data from. (attribute filename)
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)
- ○ **sample_size:** The exact number of samples which should be read (-1 = use sample ratio; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **column_separators:** Column separators for data files (regular expression) (string; default: ',\s*|;\s*|\s+')
- ○ **use_comment_characters:** Indicates if qa comment character should be used. (boolean; default: true)
- ○ **comment_chars:** Any content in a line after one of these characters will be ignored. (string; default: '#')
- ○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')
- ○ **use_quotes:** Indicates if quotes should be regarded. (boolean; default: true)
- ○ **trim_lines:** Indicates if lines should be trimmed (empty spaces are removed at the beginning and the end) before the column split is performed. (boolean; default: false)
- ○ **permutate:** Indicates if the loaded data should be permutated. (boolean; default: false)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (only for permutation, -1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator reads an example set from file. The operator can be configured to read almost all file formats.

Description: This operator reads an example set from (a) file(s). Probably you can use the default parameter values for the most file formats (including the format produced by the ExampleSetWriter, CSV, ...). Please refer to section 3.4 for details on the attribute description file set by the parameter *attributes* used to specify attribute types.

This operator supports the reading of data from multiple source files. Each attribute (including special attributes like labels, weights, ...) might be read from another file. Please note that only the minimum number of lines of all files will be read, i.e. if one of the data source files has less lines than the others, only this number of examples will be read.

The split points can be defined with regular expressions (please refer to the Java API). The default split parameter ",\s*|;\s*|\s+" should work for most file formats. This regular expression describes the following column separators

- the character "," followed by a whitespace of arbitrary length (also no white space)
- the character ";" followed by a whitespace of arbitrary length (also no white space)
- a whitespace of arbitrary length (min. 1)

A logical XOR is defined by "|". Other useful separators might be "\t" for tabulars, " " for a single whitespace, and "\s" for any whitespace.

Quoting is also possible with ". However, since using quotes might slow down the parsing it is therefore recommended to ensure that the split characters are not included in the data columns and that quotes are not needed.

Additionally you can specify comment characters which can be used at arbitrary locations of the data lines. Any content after the comment character will be ignored. Unknown attribute values can be marked with empty strings (if this is possible for your column separators) or by a question mark (recommended).

5.3.22 ExcelExampleSetWriter

Group: IO.Examples

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **excel_file:** The Excel spreadsheet file which should be written. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator writes an example set to Excel spreadsheet files.

Description: This operator can be used to write data into Microsoft Excel spreadsheets. This operator creates Excel files readable by Excel 95, 97, 2000, XP, 2003 and newer. Missing data values are indicated by empty cells.

5.3.23 ExcelExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **excel_file:** The Excel spreadsheet file which should be loaded. (filename)

◦ **sheet_number:** The number of the sheet which should be imported. (integer; 0-+∞; default: 0)

◦ **first_row_as_names:** Indicates if the first row should be used for the attribute names. (boolean; default: false)

◦ **label_column:** Indicates which column should be used for the label attribute (0: no label) (integer; 0-+∞; default: 0)

◦ **id_column:** Indicates which column should be used for the Id attribute (0: no id) (integer; 0-+∞; default: 0)

◦ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

◦ **datamanagement:** Determines, how the data is represented internally.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator reads an example set from Excel spreadsheet files.

Description: This operator can be used to load data from Microsoft Excel spreadsheets. This operator is able to reads data from Excel 95, 97, 2000, XP, and 2003. The user has to define which of the spreadsheets in the workbook should be used as data table. The table must have a format so that each line is an example and each column represents an attribute. Please note that the first line might be used for attribute names which can be indicated by a parameter.

The data table can be placed anywhere on the sheet and is allowed to contain arbitrary formatting instructions, empty rows, and empty columns. Missing data values are indicated by empty cells or by cells containing only “¿‘.

5.3.24 GnuplotWriter

Group: IO.Other

Parameters:

- **output_file:** The gnuplot file. (filename)
- **name:** The name of the process log operator which produced the data table. (string)
- ○ **title:** The title of the plot. (string; default: 'Created by RapidMiner')
- **x_axis:** The values of the x-axis. (string)
- ○ **y_axis:** The values of the y-axis (for 3d plots). (string)
- **values:** A whitespace separated list of values which should be plotted. (string)
- ○ **additional_parameters:** Additional parameters for the gnuplot header. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates gnuplot files from the data generated by a process log operator.

Description: Writes the data generated by a ProcessLogOperator to a file in gnuplot format.

5.3.25 IOContainerReader

Group: IO.Other

Parameters:

- ● **filename:** Name of file to write the output to. (filename)
- ○ **method:** Append or prepend the contents of the file to this operators input or replace this operators input?
- ○ **logfile:** Name of file to read log information from (optional). (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads an IOContainer from a file.

Description: Reads all elements of an `IOContainer` from a file. The file must be written by an IOContainerWriter (see section 5.3.26).

The operator additionally supports to read text from a logfile, which will be given to the RapidMiner `LogService`. Hence, if you add a IOContainerWriter to the end of an process and set the logfile in the process root operator, the output of applying the IOContainerReader will be quite similar to what the original process displayed.

5.3.26 IOContainerWriter

Group: IO.Other

Parameters:

- **filename:** Name of file to write the output to. (filename)

○ **zipped:** Indicates if the file content should be zipped. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes all current IO objects to a file.

Description: Writes all elements of the current `IOContainer`, i.e. all objects passed to this operator, to a file. Although this operator uses an XML serialization mechanism, the files produced for different RapidMiner versions might not be compatible. At least different Java versions should not be a problem anymore.

5.3.27 IOObjectReader

Group: IO.Other

Generated output:

- IOObject

Parameters:

- **object_file:** Filename of the object file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generic reader for all types of IOObjects.

Description: Generic reader for all types of IOObjects. Reads an IOObject from a file.

5.3.28 IOObjectWriter

Group: IO.Other

Parameters:

- **object_file:** Filename of the object file. (filename)
- ◦ **io_object:** The class of the object(s) which should be saved.
- ◦ **write_which:** Defines which input object should be written. (integer; 1-+∞; default: 1)
- ◦ **output_type:** Indicates the type of the output
- ◦ **continue_on_error:** Defines behavior on errors (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generic writer for all types of IOObjects.

Description: Generic writer for all types of IOObjects. Writes one of the input objects into a given file.

5.3.29 MassiveDataGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- ◦ **number_examples:** The number of generated examples. (integer; 0-+∞; default: 10000)
- ◦ **number_attributes:** The number of attributes. (integer; 0-+∞; default: 10000)
- ◦ **sparse_fraction:** The fraction of default attributes. (real; 0.0-1.0)

- ○ **sparse_representation:** Indicates if the example should be internally represented in a sparse format. (boolean; default: true)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates huge amounts of data for testing purposes.

Description: Generates huge amounts of data in either sparse or dense format. This operator can be used to check if huge amounts of data can be handled by RapidMiner for a given process setup without creating the correct format / writing special purpose input operators.

5.3.30 ModelLoader

Group: IO.Models

Generated output:

- Model

Parameters:

- **model_file:** Filename containing the model to load. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads a model from a given file.

Description: Reads a Model from a file that was generated by an operator like `Learner` in a previous process. Once a model is generated, it can be applied several times to newly acquired data using a model loader, an ExampleSource (see section 5.3.21), and a ModelApplier (see section 5.1.1).

5.3.31 ModelWriter

Group: IO.Models

Required input:

- Model

Generated output:

- Model

Parameters:

- **model_file:** Filename for the model file. (filename)
- ◦ **overwrite_existing_file:** Overwrite an existing file. If set to false then an index is appended to the filename. (boolean; default: true)
- ◦ **output_type:** Indicates the type of the output

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes a model to a given file.

Description: Writes the input model in the file specified by the corresponding parameter. Since models are often written into files and loaded and applied in other processes or applications, this operator offers three different writing modes for models:

- *XML*: in this mode, the models are written as plain text XML files. The file size is usually the biggest in this mode (might be several hundred mega bytes so you should be cautious) but this model type has the advantage that the user can inspect and change the files.
- *XML Zipped (default)*: In this mode, the models are written as zipped XML files. Users can simply unzip the files and read or change the contents. The file sizes are smallest for most models. For these reasons, this

mode is the default writing mode for models although the loading times are the longest due to the XML parsing and unzipping.

- *Binary*: In this mode, the models are written in an proprietary binary format. The resulting model files cannot be inspected by the user and the file sizes are usually slightly bigger then for the zipped XML files. The loading time, however, is smallers than the time needed for the other modes.

This operator is also able to keep old files if the overwriting flag is set to false. However, this could also be achieved by using some of the parameter macros provided by RapidMiner like

5.3.32 MultipleLabelGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- **number_examples:** The number of generated examples. (integer; 1-$+\infty$; default: 100)
- **regression:** Defines if multiple labels for regression tasks should be generated. (boolean; default: false)
- **attributes_lower_bound:** The minimum value for the attributes. (real; $-\infty$-$+\infty$)
- **attributes_upper_bound:** The maximum value for the attributes. (real; $-\infty$-$+\infty$)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates an example set based on numerical attributes and with more than one label.

Description: Generates a random example set for testing purposes with more than one label.

5.3.33 NominalExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- ◦ **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)
- ◦ **number_of_attributes:** The number of attributes. (integer; 0-+∞; default: 5)
- ◦ **number_of_values:** The number of nominal values for each attribute. (integer; 0-+∞; default: 5)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates an example set based on nominal attributes.

Description: Generates a random example set for testing purposes. All attributes have only (random) nominal values and a classification label.

5.3.34 ParameterSetLoader

Group: IO.Other

Generated output:

- ParameterSet

Parameters:

- **parameter_file:** A file containing a parameter set. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads a parameter set from a file.

Description: Reads a set of parameters from a file that was written by a ParameterOptimizationOperator. It can then be applied to the operators of the process using a ParameterSetter (see section 5.5.18).

5.3.35 ParameterSetWriter

Group: IO.Other

Required input:

- ParameterSet

Generated output:

- ParameterSet

Parameters:

- **parameter_file:** A file containing a parameter set. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes a parameter set into a file.

Description: Writes a parameter set into a file. This can be created by one of the parameter optimization operators, e.g. GridParameterOptimization (see section 5.5.10). It can then be applied to the operators of the process using a ParameterSetter (see section 5.5.18).

5.3.36 PerformanceLoader

Group: IO.Results

Generated output:

- PerformanceVector

Parameters:

- **performance_file:** Filename for the performance file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to load a performance vector from a file.

Description: Reads a performance vector from a given file. This performance vector must have been written before with a PerformanceWriter (see section 5.3.37).

5.3.37 PerformanceWriter

Group: IO.Results

Required input:

- PerformanceVector

Generated output:

- PerformanceVector

Parameters:

- **performance_file:** Filename for the performance file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to write the input performance into a file.

Description: Writes the input performance vector in a given file. You also might want to use the ResultWriter (see section 5.3.38) operator which writes all current results in the main result file.

5.3.38 ResultWriter

Group: IO.Results

Parameters:

○ **result_file:** Appends the descriptions of the input objects to this file. If empty, use the general file defined in the process root operator. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used at each point in an operator chain and and writes current results to the console or to a file.

Description: This operator can be used at each point in an operator chain. It returns all input it receives without any modification. Every input object which implements the ResultObject interface (which is the case for almost all objects generated by the core RapidMiner operators) will write its results to the file specified by the parameter *result_file*. If the definition of this parameter is ommited then the global result file parameter with the same name of the ProcessRootOperator (the root of the process) will be used. If this file is also not specified the results are simply written to the console (standard out).

5.3.39 SPSSExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **filename:** Name of the file to read the data from. (filename)

◦ **datamanagement:** Determines, how the data is represented internally.

◦ **attribute_naming_mode:** Determines which SPSS variable properties should be used for attribute naming.

◦ **use_value_labels:** Use SPSS value labels as values. (boolean; default: true)

◦ **recode_user_missings:** Recode SPSS user missings to missing values. (boolean; default: true)

◦ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)

◦ **sample_size:** The exact number of samples which should be read (-1 = all; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)

◦ **local_random_seed:** Use the given random seed instead of global random numbers (for sampling by ratio, -1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read SPSS data files.

Description: This operator can read spss files.

5.3.40 SimpleExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **filename:** Name of the file to read the data from. (filename)
- ○ **read_attribute_names:** Read attribute names from file (assumes the attribute names are in the first line of the file). (boolean; default: false)
- ○ **label_name:** Name of the label attribute (if empty, the column defined by label_column will be used) (string)
- ○ **label_column:** Column number of the label attribute (only used if label_name is empty; 0 = none; negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **id_name:** Name of the id attribute (if empty, the column defined by id_column will be used) (string)
- ○ **id_column:** Column number of the id attribute (only used if id_name is empty; 0 = none; negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **weight_name:** Name of the weight attribute (if empty, the column defined by weight_column will be used) (string)
- ○ **weight_column:** Column number of the weight attribute (only used if weight_name is empty; 0 = none, negative values are counted from the last column) (integer; -2147483648-+∞; default: 0)
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)
- ○ **sample_size:** The exact number of samples which should be read (-1 = use sample ratio; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **column_separators:** Column separators for data files (regular expression) (string; default: ',\s*|;\s*|\s+')
- ○ **use_comment_characters:** Indicates if qa comment character should be used. (boolean; default: true)
- ○ **comment_chars:** Lines beginning with these characters are ignored. (string; default: '#')

- **use_quotes:** Indicates if quotes should be regarded (slower!). (boolean; default: false)
- **trim_lines:** Indicates if lines should be trimmed (empty spaces are removed at the beginning and the end) before the column split is performed. (boolean; default: false)
- **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator reads an example set from file. It is a simpler version of the ExampleSource operator.

Description: This operator reads an example set from (a) file(s). Probably you can use the default parameter values for the most file formats (including the format produced by the ExampleSetWriter, CSV, ...). In fact, in many cases this operator is more appropriate for CSV based file formats than the CSVExampleSource (see section 5.3.10) operator itself.

In contrast to the usual ExampleSource operator this operator is able to read the attribute names from the first line of the data file. However, there is one restriction: the data can only be read from one file instead of multiple files. If you need a fully flexible operator for data loading you should use the more powerful ExampleSource operator.

The column split points can be defined with regular expressions (please refer to the Java API). The default split parameter ",\s*|;\s*|\s+" should work for most file formats. This regular expression describes the following column separators

- the character "," followed by a whitespace of arbitrary length (also no white space)
- the character ";" followed by a whitespace of arbitrary length (also no white space)
- a whitespace of arbitrary length (min. 1)

A logical XOR is defined by "|". Other useful separators might be "\t" for tabulars, " " for a single whitespace, and "\s" for any whitespace.

Quoting is also possible with “. However, using quotes slows down parsing and is therefore not recommended. The user should ensure that the split characters are not included in the data columns and that quotes are not needed. Additionally you can specify comment characters which can be used at arbitrary locations of the data lines. Unknown attribute values can be marked with empty strings or a question mark.

5.3.41 SparseFormatExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- ○ **format:** Format of the sparse data file.
- ○ **attribute_description_file:** Name of the attribute description file. (filename)
- ○ **data_file:** Name of the data file. Only necessary if not specified in the attribute description file. (filename)
- ○ **label_file:** Name of the data file containing the labels. Only necessary if format is 'format_separate_file'. (filename)
- ○ **dimension:** Dimension of the example space. Only necessary if parameter 'attribute_description_file' is not set. (integer; -1-+∞; default: -1)
- ○ **sample_size:** The maximum number of examples to read from the data files (-1 = all) (integer; -1-+∞; default: -1)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')
- ○ **prefix_map:** Maps prefixes to names of special attributes. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads an example file in sparse format.

Description: Reads an example file in sparse format, i.e. lines have the form

```
label index:value index:value index:value...
```

Index may be an integer (starting with 1) for the regular attributes or one of the prefixes specified by the parameter list `prefix_map`. Four possible `formats` are supported

format_xy: The label is the last token in each line

format_yx: The label is the first token in each line

format_prefix: The label is prefixed by 'l:'

format_separate_file: The label is read from a separate file specified by `label_file`

no_label: The example set is unlabeled.

A detailed introduction to the sparse file format is given in section 3.4.1.

5.3.42 StataExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **filename:** Name of the file to read the data from. (filename)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **attribute_naming_mode:** Determines which variable properties should be used for attribute naming.
- ○ **handle_value_labels:** Specifies how to handle attributes with value labels, i.e. whether to ignore the labels or how to use them.
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)
- ○ **sample_size:** The exact number of samples which should be read (-1 = all; if not -1, sample_ratio will not have any effect) (integer; -1-$+\infty$; default: -1)

- **local_random_seed:** Use the given random seed instead of global random numbers (for sampling by ratio, -1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read Stata data files.

Description: This operator can read stata files. Currently only stata files of version 113 or 114 are supported.

5.3.43 TeamProfitExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates data for testing purposes based on a team profit data set.

Description: Generates a random example set for testing purposes. The data represents a team profit example set.

5.3.44 ThresholdLoader

Group: IO.Other

Generated output:

- Threshold

Parameters:

- **threshold_file:** Filename for the threshold file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Loads a threshold from a file (used for transforming soft into crisp predictions).

Description: Reads a threshold from a file. The first line must hold the threshold, the second the value of the first class, and the second the value of the second class. This file can be written in another process using the ThresholdWriter (see section 5.3.45).

5.3.45 ThresholdWriter

Group: IO.Other

Required input:

- Threshold

Generated output:

- Threshold

Parameters:

- **threshold_file:** Filename for the threshold file. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes a threshold to a file (used for transforming soft into crisp predictions).

Description: Writes the given threshold into a file. The first line holds the threshold, the second the value of the first class, and the second the value of the second class. This file can be read in another process using the ThresholdLoader (see section 5.3.44).

5.3.46 TransfersExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- ○ **number_examples:** The number of generated examples. (integer; 1-$+\infty$; default: 100)
- ○ **create_fraud_label:** Indicates if a label should be created for possible frauds. (boolean; default: false)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates data for testing purposes based on a transfers data set.

Description: Generates a random example set for testing purposes. The data represents a team profit example set.

5.3.47 UpSellingExampleSetGenerator

Group: IO.Generator

Generated output:

- ExampleSet

Parameters:

- **number_examples:** The number of generated examples. (integer; 1-+∞; default: 100)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates data for testing purposes based on an up-selling data set.

Description: Generates a random example set for testing purposes. The data represents an up-selling example set.

5.3.48 WekaModelLoader

Group: IO.Models

Generated output:

- Model

Parameters:

- **model_file:** Filename containing the Weka model to load. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Reads a Weka model from a given file.

Description: This operator reads in model files which were saved from the Weka toolkit. For models learned within RapidMiner please use always the ModelLoader (see section 5.3.30) operator even it the used learner was originally a Weka learner.

5.3.49 XrffExampleSetWriter

Group: IO.Examples

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **example_set_file:** File to save the example set to. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Writes the values of all examples into an XRFF-file.

Description: Writes values of all examples into an XRFF file which can be used by the machine learning library Weka. The XRFF format is described in the XrffExampleSource (see section 5.3.50) operator which is able to read XRFF files to make them usable with RapidMiner.

Please note that writing attribute weights is not supported, please use the other RapidMiner operators for attribute weight loading and writing for this purpose.

5.3.50 XrffExampleSource

Group: IO.Examples

Generated output:

- ExampleSet

Parameters:

- **data_file:** The path to the data file. (filename)
- ○ **id_attribute:** The (case sensitive) name of the id attribute (string)
- ○ **datamanagement:** Determines, how the data is represented internally.
- ○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')
- ○ **sample_ratio:** The fraction of the data set which should be read (1 = all; only used if sample_size = -1) (real; 0.0-1.0)
- ○ **sample_size:** The exact number of samples which should be read (-1 = use sample ratio; if not -1, sample_ratio will not have any effect) (integer; -1-+∞; default: -1)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (only for permutation, -1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can read xrff files.

Description: This operator can read XRFF files known from Weka. The XRFF (eXtensible attribute-Relation File Format) is an XML-based extension of the

ARFF format in some sense similar to the original RapidMiner file format for attribute description files (.aml).

Here you get a small example for the IRIS dataset represented as XRFF file:

```
<?xml version="1.0" encoding="utf-8"?>
<dataset name="iris" version="3.5.3">
 <header>
    <attributes>
       <attribute name="sepallength" type="numeric"/>
       <attribute name="sepalwidth" type="numeric"/>
       <attribute name="petallength" type="numeric"/>
       <attribute name="petalwidth" type="numeric"/>
       <attribute class="yes" name="class" type="nominal">
          <labels>
             <label>Iris-setosa</label>
             <label>Iris-versicolor</label>
             <label>Iris-virginica</label>
          </labels>
       </attribute>
    </attributes>
 </header>

 <body>
    <instances>
       <instance>
          <value>5.1</value>
          <value>3.5</value>
          <value>1.4</value>
          <value>0.2</value>
          <value>Iris-setosa</value>
       </instance>
       <instance>
          <value>4.9</value>
          <value>3</value>
          <value>1.4</value>
          <value>0.2</value>
          <value>Iris-setosa</value>
       </instance>
       ...
    </instances>
 </body>
</dataset>
```

Please note that the sparse XRFF format is currently not supported, please use one of the other options for sparse data files provided by RapidMiner.

Since the XML representation takes up considerably more space since the data is wrapped into XML tags, one can also compress the data via gzip. RapidMiner automatically recognizes a file being gzip compressed, if the file's extension is .xrff.gz instead of .xrff.

Similar to the native RapidMiner data definition via .aml and almost arbitrary data files, the XRFF format contains some additional features. Via the class="yes" attribute in the attribute specification in the header, one can define which attribute should used as a prediction label attribute. Although the RapidMiner terminus for such classes is "label" instead of "class" we support the terminus class in order to not break compatibility with original XRFF files.

Please note that loading attribute weights is currently not supported, please use the other RapidMiner operators for attribute weight loading and writing for this purpose.

Instance weights can be defined via a weight XML attribute in each instance tag. By default, the weight is 1. Here's an example:

```
<instance weight="0.75">
 <value>5.1</value>
 <value>3.5</value>
 <value>1.4</value>
 <value>0.2</value>
 <value>Iris-setosa</value>
</instance>
```

Since the XRFF format does not support id attributes one have to use one of the RapidMiner operators in order to change on of the columns to the id column if desired. This has to be done after loading the data.

5.4 Learning schemes

Acquiring knowledge is fundamental for the development of intelligent systems. The operators described in this section were designed to automatically discover hypotheses to be used for future decisions. They can learn models from the given data and apply them to new data to predict a label for each observation in an unpredicted example set. The ModelApplier can be used to apply these models to unlabelled data.

Additionally to some learning schemes and meta learning schemes directly implemented in RapidMiner, all learning operators provided by Weka are also available as RapidMiner learning operators.

5.4.1 AdaBoost

Group: Learner.Supervised.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **iterations:** The maximum number of iterations. (integer; 1-+∞; default: 10)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance.
- **time:** The time elapsed since this operator started.

Learner capabilities: weighted examples

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Boosting operator allowing all learners (not restricted to Weka learners).

Description: This AdaBoost implementation can be used with all learners available in RapidMiner, not only the ones which originally are part of the Weka package.

5.4.2 AdditiveRegression

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **iterations:** The number of iterations. (integer; 1-+∞; default: 10)
- **shrinkage:** Reducing this learning rate prevent overfitting but increases the learning time. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Additive regression operator allowing all learners (not restricted to Weka learners).

Description: This operator uses regression learner as a base learner. The learner starts with a default model (mean or mode) as a first prediction model. In each iteration it learns a new base model and applies it to the example set. Then, the residuals of the labels are calculated and the next base model is learned. The learned meta model predicts the label by adding all base model predictions.

5.4.3 AgglomerativeClustering

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **measure:** similarity measure to apply
- ○ **mode:** the cluster similarity criterion (class) to use
- ○ **min_items:** The minimal number of items in a cluster. Clusters with less items are merged. (integer; 1-+∞; default: 2)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Agglomerative buttom-up clustering

Description: This operator performs generic agglomorative clustering based on a set of ids and a similarity measure. The algorithm implemented here is currently very simple and not very efficient (cubic).

5.4.4 AgglomerativeFlatClustering

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- ○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ○ **measure:** similarity measure to apply
- ○ **mode:** the cluster similarity criterion (class) to use
- ○ **k:** the maximal number of clusters (integer; 2-$+\infty$; default: 2)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Agglomerative buttom-up clustering producing a flat clustering

Description: This operator performs generic agglomorative clustering based on a set of ids and a similarity measure. Clusters are merged as long as their number is lower than a given maximum number of clusters. The algorithm implemented here is currently very simple and not very efficient (cubic).

5.4.5 AssociationRuleGenerator

Group: Learner.Unsupervised.Itemsets

Required input:

- FrequentItemSets

Generated output:

- FrequentItemSets
- AssociationRules

Parameters:

- **keep_frequent_item_sets:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **min_confidence:** The minimum confidence of the rules (real; 0.0-1.0)
- **gain_theta:** The Parameter Theta in Gain calculation (real; -∞-+∞)
- **laplace_k:** The Parameter k in LaPlace function calculation (real; 1.0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator generated a set of association rules for a given set of frequent item sets.

Description: This operator generates association rules from frequent item sets. In RapidMiner, the process of frequent item set mining is divided into two parts: first, the generation of frequent item sets and second, the generation of association rules from these sets.

For the generation of frequent item sets, you can use for example the operator FPGrowth (see section 5.4.26). The result will be a set of frequent item sets which could be used as input for this operator.

5.4.6 AttributeBasedVote

Group: Learner.Supervised.Lazy

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Short description: Actually no learning scheme since the prediction is the average of all attribute values.

Description: AttributeBasedVotingLearner is very lazy. Actually it does not learn at all but creates an AttributeBasedVotingModel. This model simply calculates the average of the attributes as prediction (for regression) or the mode of all attribute values (for classification). AttributeBasedVotingLearner is especially useful if it is used on an example set created by a meta learning scheme, e.g. by VOTE (see section 5.4.67).

5.4.7 Bagging

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

◦ **sample_ratio:** Fraction of examples used for training. Must be greater than 0 and should be lower than 1. (real; 0.0-1.0)

◦ **iterations:** The number of iterations (base models). (integer; 1-+∞; default: 10)

◦ **average_confidences:** Specifies whether to average available prediction confidences or not. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Bagging operator allowing all learners (not restricted to Weka learners).

Description: This Bagging implementation can be used with all learners available in RapidMiner, not only the ones which originally are part of the Weka package.

5.4.8 BasicRuleLearner

Group: Learner.Supervised.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **pureness:** The desired pureness, i.e. the necessary amount of the major class in a covered subset in order become pure. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a set of rules minimizing the training error without pruning.

Description: This operator builds an unpruned rule set of classification rules. It is based on the paper Cendrowska, 1987: PRISM: An algorithm for inducing modular rules.

5.4.9 BayesianBoosting

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **rescale_label_priors:** Specifies whether the proportion of labels should be equal by construction after first iteration . (boolean; default: false)
- **use_subset_for_training:** Fraction of examples used for training, remaining ones are used to estimate the confusion matrix. Set to 1 to turn off test set. (real; 0.0-1.0)
- **iterations:** The maximum number of iterations. (integer; 1-+∞; default: 10)
- **allow_marginal_skews:** Allow to skew the marginal distribution (P(x)) during learning. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance.
- **time:** The time elapsed since this operator started.

Learner capabilities: weighted examples

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Boosting operator based on Bayes' theorem.

Description: This operator trains an ensemble of classifiers for boolean target attributes. In each iteration the training set is reweighted, so that previously discovered patterns and other kinds of prior knowledge are "sampled out" [25]. An inner classifier, typically a rule or decision tree induction algorithm, is sequentially applied several times, and the models are combined to a single global model. The number of models to be trained maximally are specified by the parameter `iterations`.

If the parameter `rescale_label_priors` is set, then the example set is reweighted, so that all classes are equally probable (or frequent). For two-class problems this turns the problem of fitting models to maximize weighted relative accuracy into the more common task of classifier induction [24]. Applying a rule induction algorithm as an inner learner allows to do subgroup discovery. This option is also recommended for data sets with class skew, if a "very weak learner" like a decision stump is used. If `rescale_label_priors` is not set, then the operator performs boosting based on probability estimates.

The estimates used by this operator may either be computed using the same set as for training, or in each iteration the training set may be split randomly, so that a model is fitted based on the first subset, and the probabilities are estimated based on the second. The first solution may be advantageous in situations where data is rare. Set the parameter `ratio_internal_bootstrap` to 1 to use the same set for training as for estimation. Set this parameter to a value of lower than 1 to use the specified subset of data for training, and the remaining examples for probability estimation.

If the parameter `allow_marginal_skews` is *not* set, then the support of each subset defined in terms of common base model predictions does not change from one iteration to the next. Analogously the class priors do not change. This is the procedure originally described in [25] in the context of subgroup discovery.

Setting the `allow_marginal_skews` option to `true` leads to a procedure that changes the marginal weights/probabilities of subsets, if this is beneficial in a boosting context, and stratifies the two classes to be equally likely. As for AdaBoost, the total weight upper-bounds the training error in this case. This bound is reduced more quickly by the BayesianBoosting operator, however.

In sum, to reproduce the sequential sampling, or knowledge-based sampling, from [25] for subgroup discovery, two of the default parameter settings of this operator have to be changed: `rescale_label_priors` must be set to `true`, and `allow_marginal_skews` must be set to `false`. In addition, a boolean (binomial) label has to be used.

The operator requires an example set as its input. To sample out prior knowledge of a different form it is possible to provide another model as an optional additional input. The predictions of this model are used to weight produce an initial weighting of the training set. The ouput of the operator is a classification model applicable for estimating conditional class probabilities or for plain crisp classification. It contains up to the specified number of inner base models. In the case of an optional initial model, this model will also be stored in the output model, in order to produce the same initial weighting during model application.

5.4.10 BestRuleInduction

Group: Learner.Supervised.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **max_depth:** An upper bound for the number of literals. (integer; 1-$+\infty$; default: 2)
- **utility_function:** The function to be optimized by the rule.
- **max_cache:** Bounds the number of rules considered per depth to avoid high memory consumption, but leads to incomplete search. (integer; 1-$+\infty$; default: 10000)
- **relative_to_predictions:** Searches for rules with a maximum difference to the predited label. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label, weighted examples

Short description: Returns a best conjunctive rule with respect to the WRAcc metric for boolean prediction problems and polynomial attributes.

Description: This operator returns the best rule regarding WRAcc using exhaustive search. Features like the incorporation of other metrics and the search for more than a single rule are prepared.

The search strategy is BFS, with save pruning whenever applicable. This operator can easily be extended to support other search strategies.

5.4.11 Binary2MultiClassLearner

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **classification_strategies:** What strategy should be used for multi class classifications?
- ◦ **random_code_multiplicator:** A multiplicator regulating the codeword length in random code modus. (real; 1.0-+∞)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal label

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Builds a classification model for multiple classes based on a binary learner.

Description: A metaclassifier for handling multi-class datasets with 2-class classifiers. This class supports several strategies for multiclass classification including procedures which are capable of using error-correcting output codes for increased accuracy.

5.4.12 BregmanHardClustering

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- ○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ○ **k:** The number of clusters which should be found. (integer; 2-+∞; default: 2)
- ○ **max_runs:** The maximal number of runs of this operator with random initialization that are performed. (integer; 1-+∞; default: 5)
- ○ **max_optimization_steps:** The maximal number of iterations performed for one run of this operator. (integer; 1-+∞; default: 100)
- ○ **bregman_divergence:** The Bregman Divergence
- ○ **local_random_seed:** The local random seed (-1: use global random seed) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with Bregman Hard Clustering

Description: This operator represents an implementation of the Bregman Hard Clustering.

5.4.13 CHAID

Group: Learner.Supervised.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-$+\infty$; default: 2)
- ○ **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-$+\infty$)
- ○ **maximal_depth:** The maximum tree depth (-1: no bound) (integer; -1-$+\infty$; default: 10)
- ○ **confidence:** The confidence level used for the pessimistic error calculation of pruning. (real; 1.0E-7-0.5)
- ○ **no_pruning:** Disables the pruning and delivers an unpruned tree. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a pruned decision tree based on a chi squared attribute relevance test.

Description: The CHAID decision tree learner works like the DecisionTree (see section 5.4.20) with one exception: it used a chi squared based criterion instead of the information gain or gain ratio criteria.

5.4.14 ClassificationByRegression

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal label, binominal label, numerical label

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: This operator chain must contain a regression learner and allows to learn classifications tasks with more than two classes.

Description: For a classified dataset (with possibly more than two classes) builds a classifier using a regression method which is specified by the inner operator. For each class i a regression model is trained after setting the label to $+1$ if the label equals i and to -1 if it is not. Then the regression models are combined into a classification model. In order to determine the prediction for an unlabeled example, all models are applied and the class belonging to the regression model which predicts the greatest value is chosen.

5.4.15 ClusterModel2ExampleSet

Group: Learner.Unsupervised.Clustering

Required input:	**Generated output:**
• ClusterModel • ExampleSet	• ExampleSet

Parameters:

- **keep_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_label:** should the cluster values be added as label as well (boolean; default: false)
- **delete_unlabeled:** delete the unlabelled examples (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Labels an example set with the cluster ids from a given cluster model.

Description: Labels an example set with the cluster ids from a given cluster model.

5.4.16 ClusterModel2Similarity

Group: Learner.Unsupervised.Clustering.Similarity

Required input:	**Generated output:**
• ClusterModel	• SimilarityMeasure

Parameters:

- **keep_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **measure:** measure used to convert a cluster model into a similarity

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Calculates a similarity measure from given cluster models.

Description: This operator converts a (hierarchical) cluster model to a similarity measure.

5.4.17 CostBasedThresholdLearner

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **class_weights:** The weights for all classes (first column: class names, second column: weight), empty: using 1 for all classes. The costs for not classifying at all are defined with class name '?'. (list)
- **predict_unknown_costs:** Use this cost value for predicting an example as unknown (-1: use same costs as for correct class). (real; -1.0-$+\infty$)
- **training_ratio:** Use this amount of input data for model learning and the rest for threshold optimization. (real; 0.0-1.0)
- **number_of_iterations:** Defines the number of optimization iterations. (integer; 1-$+\infty$; default: 200)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Determines confidence thresholds based on misclassification costs, also possible to define costs for the option non-classified.

Description: This operator uses a set of class weights and also allows a weight for the fact that an example is not classified at all (marked as unknown). Based on the predictions of the model of the inner learner this operator optimized a set of thresholds regarding the defined weights.

This operator might be very useful in cases where it is better to not classify an example then to classify it in a wrong way. This way, it is often possible to get very high accuracies for the remaining examples (which are actually classified) for the cost of having some examples which must still be manually classified.

5.4.18 DBScanClustering

Group: Learner.Unsupervised.Clustering

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)

○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)

○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)

○ **min_pts:** The minimal number of points in each cluster. (integer; 0-+∞; default: 2)

○ **max_distance:** maximal distance (real; 0.0-+∞)

- ○ **measure:** similarity measure to apply

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with DBSCAN

Description: This operator represents a simple implementation of the DBSCAN algorithm. [4]).

5.4.19 DecisionStump

Group: Learner.Supervised.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- ○ **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-$+\infty$; default: 1)
- ○ **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns only a root node of a decision tree. Can be very efficient when boosted.

Description: This operator learns decision stumps, i.e. a small decision tree with only one single split. This decision stump works on both numerical and nominal attributes.

5.4.20 DecisionTree

Group: Learner.Supervised.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-+∞; default: 2)
- **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-+∞)
- **maximal_depth:** The maximum tree depth (-1: no bound) (integer; -1-+∞; default: 10)
- **confidence:** The confidence level used for the pessimistic error calculation of pruning. (real; 1.0E-7-0.5)
- **no_pruning:** Disables the pruning and delivers an unpruned tree. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a pruned decision tree which can handle both numerical and nominal attributes.

Description: This operator learns decision trees from both nominal and numerical data. Decision trees are powerful classification methods which often can also easily be understood. This decision tree learner works similar to Quinlan's C4.5 or CART.

The actual type of the tree is determined by the criterion, e.g. using gain_ratio or Gini for CART / C4.5.

5.4.21 DefaultLearner

Group: Learner.Supervised.Lazy

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **method:** The method to compute the default.
- ○ **constant:** Value returned when method = constant. (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Short description: Learns a default value.

Description: This learner creates a model, that will simply predict a default value for all examples, i.e. the average or median of the true labels (or the mode in case of classification) or a fixed specified value. This learner can be used to compare the results of "real" learning schemes with guessing.

5.4.22 EvoSVM

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **kernel_type:** The SVM kernel type
- **kernel_gamma:** The SVM kernel parameter gamma (RBF, anova). (real; 0.0-+∞)
- **kernel_sigma1:** The SVM kernel parameter sigma1 (Epanechnikov, Gaussian Combination, Multiquadric). (real; 0.0-+∞)
- **kernel_sigma2:** The SVM kernel parameter sigma2 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_sigma3:** The SVM kernel parameter sigma3 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_degree:** The SVM kernel parameter degree (polynomial, anova, Epanechnikov). (real; 0.0-+∞)
- **kernel_shift:** The SVM kernel parameter shift (polynomial, Multiquadric). (real; -∞-+∞)
- **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)
- **C:** The SVM complexity constant (0: calculates probably good value). (real; 0.0-+∞)
- **epsilon:** The width of the regression tube loss function of the regression SVM (real; 0.0-+∞)
- **start_population_type:** The type of start population initialization.
- **max_generations:** Stop after this many evaluations (integer; 1-+∞; default: 10000)
- **generations_without_improval:** Stop after this number of generations without improvement (-1: optimize until max_iterations). (integer; -1-+∞; default: 30)
- **population_size:** The population size (-1: number of examples) (integer; -1-+∞; default: 1)

- **tournament_fraction:** The fraction of the population used for tournament selection. (real; 0.0-+∞)
- **keep_best:** Indicates if the best individual should survive (elititst selection). (boolean; default: true)
- **mutation_type:** The type of the mutation operator.
- **selection_type:** The type of the selection operator.
- **crossover_prob:** The probability for crossovers. (real; 0.0-1.0)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **hold_out_set_ratio:** Uses this amount as a hold out set to estimate generalization error after learning (currently only used for multi-objective classification). (real; 0.0-1.0)
- **show_convergence_plot:** Indicates if a dialog with a convergence plot should be drawn. (boolean; default: false)
- **return_optimization_performance:** Indicates if final optimization fitness should be returned as performance. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, numerical label, weighted examples

Short description: EvoSVM uses an Evolutionary Strategy for optimization.

Description: This is a SVM implementation using an evolutionary algorithm (ES) to solve the dual optimization problem of a SVM. It turns out that on many datasets this simple implementation is as fast and accurate as the usual SVM implementations. In addition, it is also capable of learning with Kernels which are not positive semi-definite and can also be used for multi-objective learning which makes the selection of C unecessary before learning.

Mierswa, Ingo. Evolutionary Learning with Kernels: A Generic Solution for Large Margin Problems. In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2006), 2006.

5.4.23 ExampleSet2ClusterConstraintList

Group: Learner.Unsupervised.Clustering

Required input:	Generated output:
• ExampleSet	• ClusterConstraintList

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **type:** the type of constraints to create
- **link_mode:** the policy how to choose link constraints
- **link_max_must:** the maximal number of MUST_LINK constraints to create (integer; 0-$+\infty$; default: 100)
- **link_max_cannot:** the maximal number of CANNOT_LINK constraints to create (integer; 0-$+\infty$; default: 100)
- **link_weight:** the global weight of the created link constraints (real; 0.0-$+\infty$)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with constrained k-means

Description: Creates a ClusterConstraintList of the specified type from a (possibly partially) labeled ExampleSet. For the type 'link' you can choose, if you want the LinkClusterConstraints to be created randomly or orderly, always bounded by the maximal number of constraints to create. Choosing 'random walk' the Must-Link-constraints for each label will form a connected component.

5.4.24 ExampleSet2ClusterModel

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **cluster_attribute:** The name of the cluster attribute (the attribute along which the clusters are build. (string; default: 'cluster')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering based on one nominal attribute.

Description: Operator that clusters items along one given nominal attribute.

5.4.25 ExampleSet2Similarity

Group: Learner.Unsupervised.Clustering.Similarity

Required input:

- ExampleSet

Generated output:

- SimilarityMeasure

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)

- **measure:** similarity measure to apply

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Calculates a similarity measure from the given data (attribute based).

Description: This class represents an operator that creates a similarity measure based on an ExampleSet.

5.4.26 FPGrowth

Group: Learner.Unsupervised.Itemsets

Required input:	**Generated output:**
• ExampleSet	• FrequentItemSets

Parameters:

- **keep_example_set:** indicates if example set is kept (boolean; default: false)
- **find_min_number_of_itemsets:** Indicates if the support should be decreased until the specified minimum number of frequent item sets is found. Otherwise, FPGrowth simply uses the defined support. (boolean; default: true)
- **min_number_of_itemsets:** Indicates the minimum number of itemsets which should be determined if the corresponding parameter is activated. (integer; 0-$+\infty$; default: 100)
- **min_support:** The minimal support necessary in order to be a frequent item (set). (real; 0.0-1.0)
- **max_items:** The upper bound for the length of the item sets (-1: no upper bound) (integer; -1-$+\infty$; default: -1)
- **must_contain:** The items any generated rule must contain as regular expression. Empty if none. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This learner efficiently calculates all frequent item sets from the given data.

Description: This operator calculates all frequent items sets from a data set by building a FPTree data structure on the transaction data base. This is a very compressed copy of the data which in many cases fits into main memory even for large data bases. From this FPTree all frequent item set are derived. A major advantage of FPGrowth compared to Apriori is that it uses only 2 data scans and is therefore often applicable even on large data sets.

Please note that the given data set is only allowed to contain binominal attributes, i.e. nominal attributes with only two different values. Simply use the provided preprocessing operators in order to transform your data set. The necessary operators are the discretization operators for changing the value types of numerical attributes to nominal and the operator Nominal2Binominal for transforming nominal attributes into binominal / binary ones. The frequent item sets are mined for the positive entries in your data base, i.e. for those nominal values which are defined as positive in your data base. If you use an attribute description file (.aml) for the ExampleSource (see section 5.3.21) operator this corresponds to the second value which is defined via the classes attribute or inner value tags.

This operator has two basic working modes: finding at least the specified number of item sets with highest support without taking the min_support into account (default) or finding all item sets with a support large than min_support.

5.4.27 FlattenClusterModel

Group: Learner.Unsupervised.Clustering

Required input:	**Generated output:**
• ClusterModel	• ClusterModel

Parameters:

- ○ **k:** the maximal number of clusters (integer; 2-+∞; default: 2)
- ○ **performance:** return the highest cluster similarity as performance (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a flat cluster model from a hierarchical one.

Description: Creates a flat cluster model from a hierarchical one by expanding nodes in the order of their weight until the desired number of clusters is reached.

5.4.28 GPLearner

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **kernel_type:** The kind of kernel.
- ○ **kernel_lengthscale:** The lengthscale r for rbf kernel functions (exp-1.0 * r-2 * ||x - bla||). (real; 0.0-$+\infty$)
- ○ **kernel_degree:** The degree used in the poly kernel. (real; 0.0-$+\infty$)
- ○ **kernel_bias:** The bias used in the poly kernel. (real; 0.0-$+\infty$)
- ○ **kernel_sigma1:** The SVM kernel parameter sigma1 (Epanechnikov, Gaussian Combination, Multiquadric). (real; 0.0-$+\infty$)
- ○ **kernel_sigma2:** The SVM kernel parameter sigma2 (Gaussian Combination). (real; 0.0-$+\infty$)
- ○ **kernel_sigma3:** The SVM kernel parameter sigma3 (Gaussian Combination). (real; 0.0-$+\infty$)
- ○ **kernel_shift:** The SVM kernel parameter shift (polynomial, Multiquadric). (real; -∞-$+\infty$)
- ○ **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-$+\infty$)
- ○ **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-$+\infty$)

- ○ **max_basis_vectors:** Maximum number of basis vectors to be used. (integer; 1-+∞; default: 100)
- ○ **epsilon_tol:** Tolerance for gamma induced projections (real; 0.0-+∞)
- ○ **geometrical_tol:** Tolerance for geometry induced projections (real; 0.0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, numerical label

Short description: An implementation of Gaussian Processes.

Description: Gaussian Process (GP) Learner. The GP is a probabilistic method both for classification and regression.

5.4.29 HyperHyper

Group: Learner.Supervised.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **local_random_seed:** The local random seed (-1: use global random seed) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, numerical label, weighted examples

Short description: This is a minimal SVM implementation. The model is built with only one positive and one negative example. Typically this operater is used in combination with a boosting method.

Description: This is a minimal SVM implementation. The model is built with only one positive and one negative example. Typically this operater is used in combination with a boosting method.

5.4.30 ID3

Group: Learner.Supervised.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- ◦ **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-$+\infty$; default: 2)
- ◦ **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label, weighted examples

Short description: Learns an unpruned decision tree from nominal attributes only.

Description: This operator learns decision trees without pruning using nominal attributes only. Decision trees are powerful classification methods which often can also easily be understood. This decision tree learner works similar to Quinlan's ID3.

5.4.31 ID3Numerical

Group: Learner.Supervised.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- ○ **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-+∞; default: 2)
- ○ **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns an unpruned decision tree from nominal and numerical data.

Description: This operator learns decision trees without pruning using both nominal and numerical attributes. Decision trees are powerful classification methods which often can also easily be understood. This decision tree learner works similar to Quinlan's ID3.

5.4.32 IteratingGSS

Group: Learner.Supervised.Rules

This operator will not be supported in future releases

Required input:

- ExampleSet

Generated output:

- Model
- IGSSResult

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **epsilon:** approximation parameter (real; 0.01-1.0)
- **delta:** desired confidence (real; 0.01-1.0)
- **min_utility_pruning:** minimum utility used for pruning (real; -1.0-1.0)
- **min_utility_useful:** minimum utility for the usefulness of a rule (real; -1.0-1.0)
- **stepsize:** the number of examples drawn before the next hypothesis update (integer; 1-10000; default: 100)
- **large:** the number of examples a hypothesis must cover before normal approximation is used (integer; 1-10000; default: 100)
- **max_complexity:** the maximum complexity of hypothesis (integer; 1-10; default: 1)
- **min_complexity:** the minimum complexity of hypothesis (integer; 1-10; default: 1)
- **iterations:** the number of iterations (integer; 1-50; default: 10)
- **use_binomial:** Switch to binomial utility funtion before increasing complexity (boolean; default: false)
- **utility_function:** the utility function to be used
- **use_kbs:** use kbs to reweight examples after each iteration (boolean; default: true)

- **rejection_sampling:** use rejection sampling instead of weighted examples (boolean; default: true)
- **useful_criterion:** criterion to decide if the complexity is increased
- **example_factor:** used by example criterion to determine usefulness of a hypothesis (real; 1.0-5.0)
- **force_iterations:** make all iterations even if termination criterion is met (boolean; default: false)
- **generate_all_hypothesis:** generate h-¿Y+/Y- or h-¿Y+ only. (boolean; default: false)
- **reset_weights:** Set weights back to 1 when complexity is increased. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: Combines Generic Sequential Sampling by Scheffer/Wrobel with Knowledge-Based Sampling by Scholz.

Description: This operator implements the IteratingGSS algorithmus presented in the diploma thesis 'Effiziente Entdeckung unabhaengiger Subgruppen in grossen Datenbanken' at the Department of Computer Science, University of Dortmund.

5.4.33 JMySVMLearner

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **kernel_type:** The SVM kernel type
- **kernel_gamma:** The SVM kernel parameter gamma (radial, anova). (real; 0.0-+∞)
- **kernel_sigma1:** The SVM kernel parameter sigma1 (epanechnikov, gaussian combination, multiquadric). (real; 0.0-+∞)
- **kernel_sigma2:** The SVM kernel parameter sigma2 (gaussian combination). (real; 0.0-+∞)
- **kernel_sigma3:** The SVM kernel parameter sigma3 (gaussian combination). (real; 0.0-+∞)
- **kernel_shift:** The SVM kernel parameter shift (multiquadric). (real; 0.0-+∞)
- **kernel_degree:** The SVM kernel parameter degree (polynomial, anova, epanechnikov). (real; 0.0-+∞)
- **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)
- **kernel_cache:** Size of the cache for kernel evaluations im MB (integer; 0-+∞; default: 200)
- **C:** The SVM complexity constant. Use -1 for different C values for positive and negative. (real; -1.0-+∞)
- **convergence_epsilon:** Precision on the KKT conditions (real; 0.0-+∞)
- **max_iterations:** Stop after this many iterations (integer; 1-+∞; default: 100000)
- **scale:** Scale the example values and store the scaling parameters for test set. (boolean; default: true)
- **calculate_weights:** Indicates if attribute weights should be returned. (boolean; default: false)
- **return_optimization_performance:** Indicates if final optimization fitness should be returned as performance. (boolean; default: false)
- **estimate_performance:** Indicates if this learner should also return a performance estimation. (boolean; default: false)
- **L_pos:** A factor for the SVM complexity constant for positive examples (real; 0.0-+∞)
- **L_neg:** A factor for the SVM complexity constant for negative examples (real; 0.0-+∞)
- **epsilon:** Insensitivity constant. No loss if prediction lies this close to true value (real; 0.0-+∞)
- **epsilon_plus:** Epsilon for positive deviation only (real; 0.0-+∞)

- **epsilon_minus:** Epsilon for negative deviation only (real; 0.0-+∞)
- **balance_cost:** Adapts Cpos and Cneg to the relative size of the classes (boolean; default: false)
- **quadratic_loss_pos:** Use quadratic loss for positive deviation (boolean; default: false)
- **quadratic_loss_neg:** Use quadratic loss for negative deviation (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, numerical label, weighted examples

Short description: JMySVMLearner provides an internal Java implementation of the mySVM by Stefan Rueping.

Description: This learner uses the Java implementation of the support vector machine *mySVM* by Stefan Rüping. This learning method can be used for both regression and classification and provides a fast algorithm and good results for many learning tasks.

5.4.34 KMeans

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)

- ○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ○ **k:** The number of clusters which should be detected. (integer; 2-+∞; default: 2)
- ○ **max_runs:** The maximal number of runs of k-Means with random initialization that are performed. (integer; 1-+∞; default: 10)
- ○ **max_optimization_steps:** The maximal number of iterations performed for one run of k-Means. (integer; 1-+∞; default: 100)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with k-means

Description: This operator represents a simple implementation of k-means.

5.4.35 KMedoids

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- ○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ○ **k:** The number of clusters which should be found. (integer; 2-+∞; default: 2)

- ○ **max_runs:** The maximal number of runs of this operator with random initialization that are performed. (integer; 1-$+\infty$; default: 5)
- ○ **max_optimization_steps:** The maximal number of iterations performed for one run of this operator. (integer; 1-$+\infty$; default: 100)
- ○ **measure:** similarity measure to apply
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with k-medoids

Description: Simple implementation of k-medoids.

5.4.36 KernelKMeans

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- ○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ○ **k:** The number of clusters which should be found. (integer; 2-$+\infty$; default: 2)
- ○ **max_runs:** The maximal number of runs of this operator with random initialization that are performed. (integer; 1-$+\infty$; default: 5)

○ **max_optimization_steps:** The maximal number of iterations performed for one run of this operator. (integer; 1-$+\infty$; default: 100)

○ **scale:** Indicates if the examples are scaled before clustering is applied. (boolean; default: true)

○ **cache_size_mb:** The size of the kernel cache. (integer; 0-$+\infty$; default: 50)

- **kernel_type:** The kernel type, i.e. the similarity measure which should be applied.

○ **kernel_gamma:** The SVM kernel parameter gamma (radial). (real; 0.0-$+\infty$)

○ **kernel_degree:** The SVM kernel parameter degree (polynomial). (integer; 0-$+\infty$; default: 2)

○ **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-$+\infty$)

○ **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with kernel k-means

Description: Simple implementation of kernel k-means [3].

5.4.37 KernelLogisticRegression

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **kernel_type:** The kernel type

- **kernel_gamma:** The kernel parameter gamma (RBF, anova). (real; 0.0-+∞)
- **kernel_sigma1:** The kernel parameter sigma1 (Epanechnikov, Gaussian Combination, Multiquadric). (real; 0.0-+∞)
- **kernel_sigma2:** The kernel parameter sigma2 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_sigma3:** The kernel parameter sigma3 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_degree:** The kernel parameter degree (polynomial, anova, Epanechnikov). (real; 0.0-+∞)
- **kernel_shift:** The kernel parameter shift (polynomial, Multiquadric). (real; -∞-+∞)
- **kernel_a:** The kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The kernel parameter b (neural). (real; -∞-+∞)
- **C:** The complexity constant. (real; 1.0E-8-+∞)
- **start_population_type:** The type of start population initialization.
- **max_generations:** Stop after this many evaluations (integer; 1-+∞; default: 10000)
- **generations_without_improval:** Stop after this number of generations without improvement (-1: optimize until max_iterations). (integer; -1-+∞; default: 30)
- **population_size:** The population size (-1: number of examples) (integer; -1-+∞; default: 1)
- **tournament_fraction:** The fraction of the population used for tournament selection. (real; 0.0-+∞)
- **keep_best:** Indicates if the best individual should survive (elititst selection). (boolean; default: true)
- **mutation_type:** The type of the mutation operator.
- **selection_type:** The type of the selection operator.
- **crossover_prob:** The probability for crossovers. (real; 0.0-1.0)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **show_convergence_plot:** Indicates if a dialog with a convergence plot should be drawn. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, weighted examples

Short description: A kernel logistic regression learner for binary classification tasks.

Description: This operator determines a logistic regression model.

5.4.38 LibSVMLearner

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

◦ **svm_type:** SVM for classification (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class)

◦ **kernel_type:** The type of the kernel functions

◦ **degree:** The degree for a polynomial kernel function. (integer; 1-+∞; default: 3)

◦ **gamma:** The parameter gamma for polynomial, rbf, and sigmoid kernel functions (0 means 1/#attributes). (real; 0.0-+∞)

◦ **coef0:** The parameter coef0 for polynomial and sigmoid kernel functions. (real; -∞-+∞)

◦ **C:** The cost parameter C for c_svc, epsilon_svr, and nu_svr. (real; 0.0-+∞)

◦ **nu:** The parameter nu for nu_svc, one_class, and nu_svr. (real; 0.0-0.5)

◦ **cache_size:** Cache size in Megabyte. (integer; 0-+∞; default: 80)

◦ **epsilon:** Tolerance of termination criterion. (real; -∞-+∞)

◦ **p:** Tolerance of loss function of epsilon-SVR. (real; -∞-+∞)

- ○ **class_weights:** The weights w for all classes (first column: class name, second column: weight), i.e. set the parameters C of each class w * C (empty: using 1 for all classes where the weight was not defined). (list)
- ○ **shrinking:** Whether to use the shrinking heuristics. (boolean; default: true)
- ○ **calculate_confidences:** Indicates if proper confidence values should be calculated. (boolean; default: false)
- ○ **confidence_for_multiclass:** Indicates if the class with the highest confidence should be selected in the multiclass setting. Uses binary majority vote over all 1-vs-1 classifiers otherwise (selected class must not be the one with highest confidence in that case). (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, numerical label

Short description: LibSVMLearner encapsulates the Java libsvm, an SVM learner.

Description: Applies the libsvm[1] learner by Chih-Chung Chang and Chih-Jen Lin. The SVM is a powerful method for both classification and regression. This operator supports the SVM types `C-SVC` and `nu-SVC` for classification tasks and `epsilon-SVR` and `nu-SVR` for regression tasks. Supports also multiclass learning and probability estimation based on Platt scaling for proper confidence values after applying the learned model on a classification data set.

5.4.39 LinearRegression

Group: Learner.Supervised.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **feature_selection:** The feature selection method used during regression.
- **eliminate_colinear_features:** Indicates if the algorithm should try to delete colinear features during the regression. (boolean; default: true)
- **min_standardized_coefficient:** The minimum standardized coefficient for the removal of colinear feature elimination. (real; 0.0-$+\infty$)
- **ridge:** The ridge parameter used during ridge regression. (real; 0.0-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, numerical label, weighted examples

Short description: Linear regression.

Description: This operator calculates a linear regression model. It uses the Akaike criterion for model selection.

5.4.40 LogisticRegression

Group: Learner.Supervised.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **add_intercept:** Determines whether to include an intercept. (boolean; default: true)

- ○ **return_model_performance:** Determines whether to return the performance. (boolean; default: false)
- ○ **start_population_type:** The type of start population initialization.
- ○ **max_generations:** Stop after this many evaluations (integer; 1-+∞; default: 10000)
- ○ **generations_without_improval:** Stop after this number of generations without improvement (-1: optimize until max_iterations). (integer; -1-+∞; default: 300)
- ○ **population_size:** The population size (-1: number of examples) (integer; -1-+∞; default: 3)
- ○ **tournament_fraction:** The fraction of the population used for tournament selection. (real; 0.0-+∞)
- ○ **keep_best:** Indicates if the best individual should survive (elititst selection). (boolean; default: true)
- ○ **mutation_type:** The type of the mutation operator.
- ○ **selection_type:** The type of the selection operator.
- ○ **crossover_prob:** The probability for crossovers. (real; 0.0-1.0)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **show_convergence_plot:** Indicates if a dialog with a convergence plot should be drawn. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, weighted examples

Short description: A logistic regression learner for binary classification tasks.

Description: This operator determines a logistic regression model.

5.4.41 MPCKMeans

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet
- ClusterConstraintList

Generated output:

- ClusterModel

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)

○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)

○ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)

○ **k:** The number of clusters which should be found. (integer; 2-$+\infty$; default: 2)

○ **max_runs:** The maximal number of runs of this operator with random initialization that are performed. (integer; 1-$+\infty$; default: 5)

○ **max_optimization_steps:** The maximal number of iterations performed for one run of this operator. (integer; 1-$+\infty$; default: 100)

○ **k_from_labels:** set k to the number of unique labels in the example set (boolean; default: false)

○ **metric_update:** choose whether to learn diagonal or full or not to learn metric matrices

○ **metric_mode:** use a single metric for all clusters or one metric for each cluster

○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with the constrained clusterer MPCKMeans

Description: This is an implementation of the ”Metric Pairwise Constraints K-Means” algorithm (see ”Mikhail Bilenko, Sugato Basu, and Raymond J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In Proceedings of the 21st International Conference on Machine Learning, ICML, pages 8188, Banff, Canada, July 2004.”) that uses a list of LinkClusterConstraints created from a (possibly partially) labeled ExampleSet to learn a parameterized euklidean distance metric.

5.4.42 MetaCost

Group: Learner.Supervised.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

• **cost_matrix:** The cost matrix in Matlab single line format (string)

◦ **use_subset_for_training:** Fraction of examples used for training. Must be greater than 0 and should be lower than 1. (real; 0.0-1.0)

◦ **iterations:** The number of iterations (base models). (integer; 1-+∞; default: 10)

◦ **sampling_with_replacement:** Use sampling with replacement (true) or without (false) (boolean; default: true)

◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Builds a classification model using cost values from a given matrix.

Description: This operator uses a given cost matrix to compute label predictions according to classification costs. The method used by this operator is similar to MetaCost as described by Pedro Domingos.

5.4.43 MultiCriterionDecisionStump

Group: Learner.Supervised.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **utility_function:** The function to be optimized by the rule.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label, weighted examples

Short description: A quick DecisionStump clone that allows to specify different utility functions.

Description: A DecisionStump clone that allows to specify different utility functions. It is quick for nominal attributes, but does not yet apply pruning for continuos attributes. Currently it can only handle boolean class labels.

5.4.44 MyKLRLearner

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **kernel_type:** The SVM kernel type
- **kernel_gamma:** The SVM kernel parameter gamma (radial, anova). (real; 0.0-+∞)
- **kernel_sigma1:** The SVM kernel parameter sigma1 (epanechnikov, gaussian combination, multiquadric). (real; 0.0-+∞)
- **kernel_sigma2:** The SVM kernel parameter sigma2 (gaussian combination). (real; 0.0-+∞)
- **kernel_sigma3:** The SVM kernel parameter sigma3 (gaussian combination). (real; 0.0-+∞)
- **kernel_shift:** The SVM kernel parameter shift (multiquadric). (real; 0.0-+∞)
- **kernel_degree:** The SVM kernel parameter degree (polynomial, anova, epanechnikov). (real; 0.0-+∞)
- **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)
- **kernel_cache:** Size of the cache for kernel evaluations im MB (integer; 0-+∞; default: 200)
- **C:** The SVM complexity constant. Use -1 for different C values for positive and negative. (real; -1.0-+∞)
- **convergence_epsilon:** Precision on the KKT conditions (real; 0.0-+∞)
- **max_iterations:** Stop after this many iterations (integer; 1-+∞; default: 100000)
- **scale:** Scale the example values and store the scaling parameters for test set. (boolean; default: true)
- **calculate_weights:** Indicates if attribute weights should be returned. (boolean; default: false)

- **return_optimization_performance:** Indicates if final optimization fitness should be returned as performance. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label

Short description: MyKLRLearner provides an internal Java implementation of the myKLR by Stefan Rueping.

Description: This is the Java implementation of *myKLR* by Stefan Rüping. myKLR is a tool for large scale kernel logistic regression based on the algorithm of Keerthi/etal/2003 and the code of mySVM.

5.4.45 NaiveBayes

Group: Learner.Supervised.Bayes

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **laplace_correction:** Use Laplace correction to prevent high influence of zero probabilities. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Returns classification model using estimated normal distributions.

Description: Naive Bayes learner.

5.4.46 NearestNeighbors

Group: Learner.Supervised.Lazy

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **k:** The used number of nearest neighbors. (integer; 1-+∞; default: 1)
- ◦ **measure_types:** The measure type
- ◦ **weighted_vote:** Indicates if the votes should be weighted by similarity. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: Classification with k-NN based on an explicit similarity measure.

Description: A k nearest neighbor implementation.

5.4.47 NeuralNet

Group: Learner.Supervised.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **input_layer_type:** The default layer type for the input layers.
- ◦ **output_layer_type:** The default layer type for the output layers.
- ◦ **default_number_of_hidden_layers:** The number of hidden layers. Only used if no layers are defined by the list hidden_layer_types. (integer; 1-$+\infty$; default: 1)
- ◦ **default_hidden_layer_size:** The default size of hidden layers. Only used if no layers are defined by the list hidden_layer_types. -1 means size (number of attributes + number of classes) / 2 (integer; -1-$+\infty$; default: -1)
- ◦ **default_hidden_layer_type:** The default layer type for the hidden layers. Only used if the parameter list hidden_layer_types is not defined.
- ◦ **hidden_layer_types:** Describes the name, the size, and the type of all hidden layers (list)
- ◦ **training_cycles:** The number of training cycles used for the neural network training. (integer; 1-$+\infty$; default: 200)
- ◦ **learning_rate:** The learning rate determines by how much we change the weights at each step. (real; 0.0-1.0)
- ◦ **momentum:** The momentum simply adds a fraction of the previous weight update to the current one (prevent local maxima and smoothes optimization directions). (real; 0.0-1.0)
- ◦ **error_epsilon:** The optimization is stopped if the training error gets below this epsilon value. (real; 0.0-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label, numerical label

Short description: Learns a neural net from the input data.

Description: This operator learns a model by means of a feed-forward neural network. The learning is done via backpropagation. The user can define the structure of the neural network with the parameter list "hidden_layer_types". Each list entry describes a new hidden layer. The key of each entry must correspond to the layer type which must be one out of

- linear
- sigmoid (default)
- tanh
- sine
- logarithmic
- gaussian

The key of each entry must be a number defining the size of the hidden layer. A size value of -1 or 0 indicates that the layer size should be calculated from the number of attributes of the input example set. In this case, the layer size will be set to (number of attributes + number of classes) / 2 + 1.

If the user does not specify any hidden layers, a default hidden layer with sigmoid type and size (number of attributes + number of classes) / 2 + 1 will be created and added to the net.

The type of the input nodes is sigmoid. The type of the output node is sigmoid is the learning data describes a classification task and linear for numerical regression tasks.

5.4.48 OneR

Group: Learner.Supervised.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a single rule using only one attribute.

Description: This operator concentrates on one single attribute and determines the best splitting terms for minimizing the training error. The result will be a single rule containing all these terms.

5.4.49 Perceptron

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **rounds:** The number of datascans used to adapt the hyperplane. (integer; 0-+∞; default: 3)
- **learning_rate:** The hyperplane will adapt with this rate to each example. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, weighted examples

Short description: Single Perceptron finding seperating hyperplane if one exists

Description: The perceptron is a type of artificial neural network invented in 1957 by Frank Rosenblatt. It can be seen as the simplest kind of feedforward neural network: a linear classifier. Beside all biological analogies, the single layer perceptron is simply a linear classifier which is efficiently trained by a simple update rule: for all wrongly classified data points, the weight vector is either increased or decreased by the corresponding example values.

5.4.50 PsoSVM

Group: Learner.Supervised.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **show_convergence_plot:** Indicates if a dialog with a convergence plot should be drawn. (boolean; default: false)
- **kernel_type:** The SVM kernel type
- **kernel_gamma:** The SVM kernel parameter sigma (radial kernel). (real; 0.0-$+\infty$)
- **kernel_degree:** The SVM kernel parameter degree (polynomial). (real; 0.0-$+\infty$)
- **kernel_shift:** The SVM kernel parameter shift (polynomial). (real; $-\infty$-$+\infty$)
- **kernel_a:** The SVM kernel parameter a (neural). (real; $-\infty$-$+\infty$)
- **kernel_b:** The SVM kernel parameter b (neural). (real; $-\infty$-$+\infty$)
- **C:** The SVM complexity constant (0: calculates probably good value). (real; 0.0-$+\infty$)

- **max_evaluations:** Stop after this many evaluations (integer; 1-+∞; default: 500)
- **generations_without_improval:** Stop after this number of generations without improvement (-1: optimize until max_iterations). (integer; -1-+∞; default: 10)
- **population_size:** The population size (-1: number of examples) (integer; -1-+∞; default: 10)
- **inertia_weight:** The (initial) weight for the old weighting. (real; 0.0-+∞)
- **local_best_weight:** The weight for the individual's best position during run. (real; 0.0-+∞)
- **global_best_weight:** The weight for the population's best position during run. (real; 0.0-+∞)
- **dynamic_inertia_weight:** If set to true the inertia weight is improved during run. (boolean; default: true)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label

Short description: PsoSVM uses a Particle Swarm Optimization for optimization.

Description: This is a SVM implementation using a particle swarm optimization (PSO) approach to solve the dual optimization problem of a SVM. It turns out that on many datasets this simple implementation is as fast and accurate as the usual SVM implementations.

5.4.51 RVMLearner

Group: Learner.Supervised.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **rvm_type:** Regression RVM
- **kernel_type:** The type of the kernel functions.
- **max_iteration:** The maximum number of iterations used. (integer; 1-+∞; default: 100)
- **min_delta_log_alpha:** Abort iteration if largest log alpha change is smaller than this (real; 0.0-+∞)
- **alpha_max:** Prune basis function if its alpha is bigger than this (real; 0.0-+∞)
- **kernel_lengthscale:** The lengthscale used in all kernels. (real; 0.0-+∞)
- **kernel_degree:** The degree used in the poly kernel. (real; 0.0-+∞)
- **kernel_bias:** The bias used in the poly kernel. (real; 0.0-+∞)
- **kernel_sigma1:** The SVM kernel parameter sigma1 (Epanechnikov, Gaussian Combination, Multiquadric). (real; 0.0-+∞)
- **kernel_sigma2:** The SVM kernel parameter sigma2 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_sigma3:** The SVM kernel parameter sigma3 (Gaussian Combination). (real; 0.0-+∞)
- **kernel_shift:** The SVM kernel parameter shift (polynomial, Multiquadric). (real; -∞-+∞)
- **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, binominal label, numerical label

Short description: An implementation of a relevance vector machine.

Description: Relevance Vector Machine (RVM) Learner. The RVM is a probabilistic method both for classification and regression. The implementation of the relevance vector machine is based on the original algorithm described by Tipping/2001. The fast version of the marginal likelihood maximization (Tipping/Faul/2003) is also available if the parameter "rvm_type" is set to "Constructive-Regression-RVM".

5.4.52 RandomFlatClustering

Group: Learner.Unsupervised.Clustering

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)

◦ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)

◦ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)

◦ **k:** the maximal number of clusters (integer; 2-+∞; default: 2)

◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Flat random clustering

Description: Returns a random clustering. Note that this algorithm does not garantuee that all clusters are non-empty.

5.4.53 RandomForest

Group: Learner.Supervised.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **number_of_trees:** The number of learned random trees. (integer; 1-$+\infty$; default: 10)
- **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-$+\infty$; default: 2)
- **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-$+\infty$)
- **maximal_depth:** The maximum tree depth (-1: no bound) (integer; -1-$+\infty$; default: 10)
- **subset_ratio:** Ratio of randomly chosen attributes to test (-1: use log(m) + 1 features) (real; -1.0-1.0)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a set of random trees, i.e. for each split only a random subset of attributes is available. The resulting model is a voting model of all trees.

Description: This operators learns a random forest. The resulting forest model contains serveral single random tree models.

5.4.54 RandomTree

Group: Learner.Supervised.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-+∞; default: 2)
- **minimal_gain:** The minimal gain which must be achieved in order to produce a split. (real; 0.0-+∞)
- **maximal_depth:** The maximum tree depth (-1: no bound) (integer; -1-+∞; default: 10)
- **confidence:** The confidence level used for the pessimistic error calculation of pruning. (real; 1.0E-7-0.5)
- **no_pruning:** Disables the pruning and delivers an unpruned tree. (boolean; default: false)
- **subset_ratio:** Ratio of randomly chosen attributes to test (-1: use log(m) + 1 features) (real; -1.0-1.0)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a single decision tree. For each split only a random subset of attributes is available.

Description: This operator learns decision trees from both nominal and numerical data. Decision trees are powerful classification methods which often can also easily be understood. The random tree learner works similar to Quinlan's C4.5 or CART but it selects a random subset of attributes before it is applied. The size of the subset is defined by the parameter subset_ratio.

5.4.55 RelativeRegression

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **relative_attribute:** Indicates which attribute should be used as a base for the relative comparison (counting starts with 1 or -1; negative: counting starts with the last; positive: counting starts with the first). (integer; $-\infty$-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Learns a regression model for predictions relative to another attribute value.

Description: This meta regression learner transforms the label on-the-fly relative to the value of the specified attribute. This is done right before the inner regression learner is applied. This can be useful in order to allow time series predictions on data sets with large trends.

5.4.56 RelevanceTree

Group: Learner.Supervised.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **minimal_leaf_size:** The minimal size of all leaves. (integer; 1-$+\infty$; default: 2)
- **maximal_depth:** The maximum tree depth (-1: no bound) (integer; -1-$+\infty$; default: 10)
- **confidence:** The confidence level used for pruning. (real; 1.0E-7-0.5)
- **no_pruning:** Disables the pruning and delivers an unpruned tree. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [AttributeWeights].

Short description: Learns a pruned decision tree based on an arbitrary feature relevance test (attribute weighting scheme as inner operator).

Description: Learns a pruned decision tree based on arbitrary feature relevance measurements defined by an inner operator (use for example InfoGainRatioWeighting (see section 5.8.60) for C4.5 and ChiSquaredWeighting (see section 5.8.21) for CHAID. Works only for nominal attributes.

5.4.57 RuleLearner

Group: Learner.Supervised.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **criterion:** Specifies the used criterion for selecting attributes and numerical splits.
- **sample_ratio:** The sample ratio of training data used for growing and pruning. (real; 0.0-1.0)
- **pureness:** The desired pureness, i.e. the necessary amount of the major class in a covered subset in order become pure. (real; 0.0-1.0)
- **minimal_prune_benefit:** The minimum amount of benefit which must be exceeded over unpruned benefit in order to be pruned. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Learns a pruned set of rules with respect to the information gain.

Description: This operator works similar to the propositional rule learner named Repeated Incremental Pruning to Produce Error Reduction (RIPPER, Cohen 1995). Starting with the less prevalent classes, the algorithm iteratively grows and prunes rules until there are no positive examples left or the error rate is greater than 50

In the growing phase, for each rule greedily conditions are added to the rule until the rule is perfect (i.e. 100possible value of each attribute and selects the condition with highest information gain.

In the prune phase, for each rule any final sequences of the antecedents is pruned with the pruning metric p/(p+n).

5.4.58 SimilarityComparator

Group: Learner.Unsupervised.Clustering.Similarity

Required input:

- SimilarityMeasure

Generated output:

- PerformanceVector

Parameters:

- ◦ **keep_similarity_measure:** Indicates if this input object should also be returned as output. (boolean; default: false)
- • **measure:** similarity measure to apply
- ◦ **sampling_rate:** the sampling rate used for comparision (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator compares two similarity measures using diverse metrics.

Description: Operator that compares two similarity measures using diverse metrics.

5.4.59 Stacking

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **keep_all_attributes:** Indicates if all attributes (including the original ones) in order to learn the stacked model. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Uses the first inner learner to build a stacked model on top of the predictions of the other inner learners.

Description: This class uses n+1 inner learners and generates n different models by using the last n learners. The predictions of these n models are taken to create n new features for the example set, which is finally used to serve as an input of the first inner learner.

5.4.60 SubgroupDiscovery

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **ratio_internal_bootstrap:** Fraction of examples used for training (internal bootstrapping). If activated (value ¡ 1) only the rest is used to estimate the biases. (real; 0.0-1.0)
- **iterations:** The maximum number of iterations. (integer; 1-+∞; default: 10)
- **ROC_convex_hull_filter:** A parameter whether to discard all rules not lying on the convex hull in ROC space. (boolean; default: true)
- **additive_reweight:** If enabled then resampling is done by additive reweighting, otherwise by multiplicative reweighting. (boolean; default: true)
- **gamma:** Factor used for multiplicative reweighting. Has no effect in case of additive reweighting. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A Subgroup Discovery meta learning scheme

Description: Subgroup discovery learner.

5.4.61 SupportVectorClustering

Group: Learner.Unsupervised.Clustering

Required input:	Generated output:
• ExampleSet	• ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **min_pts:** The minimal number of points in each cluster. (integer; 0-+∞; default: 2)
- **kernel_type:** The SVM kernel type
- **kernel_gamma:** The SVM kernel parameter gamma (radial). (real; 0.0-+∞)
- **kernel_degree:** The SVM kernel parameter degree (polynomial). (integer; 0-+∞; default: 2)
- **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)
- **kernel_cache:** Size of the cache for kernel evaluations im MB (integer; 0-+∞; default: 200)
- **convergence_epsilon:** Precision on the KKT conditions (real; 0.0-+∞)
- **max_iterations:** Stop after this many iterations (integer; 1-+∞; default: 100000)
- **p:** The fraction of allowed outliers. (real; 0.0-1.0)
- **r:** Use this radius instead of the calculated one (-1 for calculated radius). (real; -1.0-+∞)
- **number_sample_points:** The number of virtual sample points to check for neighborship. (integer; 1-+∞; default: 20)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Clustering with support vectors

Description: An implementation of Support Vector Clustering based on [1].

5.4.62 TopDownClustering

Group: Learner.Unsupervised.Clustering

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **max_leaf_size:** The maximal number of items in each cluster leaf (integer; 1-+∞; default: 1)
- ○ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [ClusterModel].

Short description: Hierarchical clustering by applying an inner clusterer scheme recursively

Description: A top-down generic clustering that can be used with any (flat) clustering as inner operator. Note though, that the outer operator cannot set or get the maximal number of clusters, the inner operator produces. These value has to be set in the inner operator.

5.4.63 TopDownRandomClustering

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ◦ **max_leaf_size:** The maximal number of items in each cluster leaf (integer; 1-+∞; default: 1)
- ◦ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute (boolean; default: true)
- ◦ **max_k:** the maximal number of clusters at each level (integer; 2-+∞; default: 2)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: Random top down clustering

Description: Creates a random top down clustering. Used for testing purposes.

5.4.64 TransformedRegression

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **transformation_method:** Type of transformation to use on the labels (log, exp, transform to mean 0 and variance 1, rank, or none).
- **z_scale:** Scale transformed values to mean 0 and standard deviation 1? (boolean; default: false)
- **interpolate_rank:** Interpolate prediction if predicted rank is not an integer? (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: This learner performs regression by transforming the labels and calling an inner regression learner.

Description: This meta learner applies a transformation on the label before the inner regression learner is applied.

5.4.65 Tree2RuleConverter

Group: Learner.Supervised.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Determines a set of rules from a given decision tree model.

Description: This meta learner uses an inner tree learner and creates a rule model from the learned decision tree.

5.4.66 UPGMAClustering

Group: Learner.Unsupervised.Clustering

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)

◦ **distance_measure:** Specifies the way the distance of two examples is calculated.

◦ **cluster_distance_measure:** Specifies the way the distance of two clusters is calculated.

◦ **add_cluster_attribute:** if true, a cluster id is generated as new special attribute (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Efficient implementation of an agglomerative buttom up hierarchical clusterer

Description: This operator generates a tree each node of which represents a cluster. UPGMA stands for Unweighted Pair Group Method using Arithmetic Means. Since the way cluster distances are calculated can be specified using parameters, this name is slightly misleading. Unfortunately, the name of the algorithm changes depending on the parameters used.

Starting with initial clusters of size 1, the algorithm unites two clusters with minimal distance forming a new tree node. This is iterated until there is only one cluster left which forms the root of the tree.

This operator does not generate a special cluster attribute and does not modify the input example set at all, since it generates too many clusters. The tree generated by this cluster is considered the interesting result of the algorithm.

5.4.67 Vote

Group: Learner.Supervised.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators:

- Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Uses a majority vote (for classification) or the average (for regression) on top of the predictions of the other inner learners.

Description: This class uses n+1 inner learners and generates n different models by using the last n learners. The predictions of these n models are taken to create n new features for the example set, which is finally used to serve as an input of the first inner learner.

5.4.68 W-ADTree

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **B:** Number of boosting iterations. (Default = 10) (real; -∞-+∞)
- ○ **E:** Expand nodes: -3(all), -2(weight), -1(z_pure), ¿=0 seed for random walk (Default = -3) (real; -∞-+∞)
- ○ **D:** Save the instance data with the model (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label, weighted examples

Short description: Class for generating an alternating decision tree.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In: Proceeding of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, 124-133, 1999.

5.4.69 W-AODE

Group: Learner.Supervised.Weka.Bayes

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Output debugging information (boolean; default: false)
- **F:** Impose a frequency limit for superParents (default is 1) (real; $-\infty$-$+\infty$)
- **M:** Use m-estimate instead of laplace correction (boolean; default: false)
- **W:** Specify a weight to use with m-estimate (default is 1) (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label, updatable, weighted examples

Short description: AODE achieves highly accurate classification by averaging over all of a small space of alternative naive-Bayes-like models that have weaker (and hence less detrimental) independence assumptions than naive Bayes.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: G. Webb, J. Boughton, Z. Wang (2005). Not So Naive Bayes: Aggregating One-Dependence Estimators. Machine Learning. 58(1):5-24.

5.4.70 W-AODEsr

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Output debugging information (boolean; default: false)
- **C:** Impose a critcal value for specialization-generalization relationship (default is 50) (real; $-\infty$-$+\infty$)
- **F:** Impose a frequency limit for superParents (default is 1) (real; $-\infty$-$+\infty$)
- **L:** Using Laplace estimation (default is m-esimation (m=1)) (string)
- **M:** Weight value for m-estimation (default is 1.0) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label, updatable, weighted examples

Short description: AODEsr augments AODE with Subsumption Resolution.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Fei Zheng, Geoffrey I. Webb: Efficient Lazy Elimination for Averaged-One Dependence Estimators. In: Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006), 1113-1120, 2006.

5.4.71 W-AdaBoostM1

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **P:** Percentage of weight mass to base training on. (default 100, reduce to around 90 speed up) (real; -∞-+∞)
- ◦ **Q:** Use resampling for boosting. (boolean; default: false)
- ◦ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ◦ **I:** Number of iterations. (default 10) (real; -∞-+∞)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for boosting a nominal class classifier using the Adaboost M1 method.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.

5.4.72 W-AdditiveRegression

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Specify shrinkage rate. (default = 1.0, ie. no shrinkage) (real; $-\infty$-$+\infty$)
- **I:** Number of iterations. (default 10) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Meta classifier that enhances the performance of a regression base classifier.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: J.H. Friedman (1999). Stochastic Gradient Boosting.

5.4.73 W-Apriori

Group: Learner.Unsupervised.Itemsets.Weka

Required input:

- ExampleSet

Generated output:

- WekaAssociator

Parameters:

- **N:** The required number of rules. (default = 10) (real; -∞-+∞)
- **T:** The metric type by which to rank rules. (default = confidence) (real; -∞-+∞)
- **C:** The minimum confidence of a rule. (default = 0.9) (real; -∞-+∞)
- **D:** The delta by which the minimum support is decreased in each iteration. (default = 0.05) (real; -∞-+∞)
- **U:** Upper bound for minimum support. (default = 1.0) (real; -∞-+∞)
- **M:** The lower bound for the minimum support. (default = 0.1) (real; -∞-+∞)
- **S:** If used, rules are tested for significance at the given level. Slower. (default = no significance testing) (real; -∞-+∞)
- **I:** If set the itemsets found are also output. (default = no) (boolean; default: false)
- **R:** Remove columns that contain all missing values (default = no) (boolean; default: false)
- **V:** Report progress iteratively. (default = no) (boolean; default: false)
- **A:** If set class association rules are mined. (default = no) (boolean; default: false)
- **c:** The class index. (default = last) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Class implementing an Apriori-type algorithm.

Description: Performs the Weka association rule learner with the same name. The operator returns a result object containing the rules found by the association learner. In contrast to models generated by normal learners, the association rules cannot be applied to an example set. Hence, there is no way to evaluate the performance of association rules yet. See the Weka javadoc for further operator and parameter descriptions.

Further information: R. Agrawal, R. Srikant: Fast Algorithms for Mining Association Rules in Large Databases. In: 20th International Conference on Very Large Data Bases, 478-499, 1994.

Bing Liu, Wynne Hsu, Yiming Ma: Integrating Classification and Association Rule Mining. In: Fourth International Conference on Knowledge Discovery and Data Mining, 80-86, 1998.

5.4.74 W-BFTree

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Random number seed. (default 1) (real; -∞-+∞)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **P:** The pruning strategy. (default: POSTPRUNED) (string; default: 'POSTPRUNED')
- **M:** The minimal number of instances at the terminal nodes. (default 2) (real; -∞-+∞)
- **N:** The number of folds used in the pruning. (default 5) (real; -∞-+∞)
- **H:** Don't use heuristic search for nominal attributes in multi-class problem (default yes). (boolean; default: false)
- **G:** Don't use Gini index for splitting (default yes), if not information is used. (boolean; default: false)
- **R:** Don't use error rate in internal cross-validation (default yes), but root mean squared error. (boolean; default: false)

- **A:** Use the 1 SE rule to make pruning decision. (default no). (boolean; default: false)
- **C:** Percentage of training data size (0-1] (default 1). (string; default: '1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Class for building a best-first decision tree classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Haijian Shi (2007). Best-first decision tree learning. Hamilton, NZ.

Jerome Friedman, Trevor Hastie, Robert Tibshirani (2000). Additive logistic regression : A statistical view of boosting. Annals of statistics. 28(2):337-407.

5.4.75 W-BIFReader

Group: Learner.Supervised.Weka.Net

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Do not use ADTree data structure (string; default: '-Q')
- **B:** BIF file to compare with (string)
- **Q:** Search algorithm (string; default: 'weka.classifiers.bayes.net.search.local.K2')

- ○ **E:** Estimator algorithm (string; default: 'weka.classifiers.bayes.net.estimate.SimpleEstimator')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Builds a description of a Bayes Net classifier stored in XML BIF 0.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Fabio Cozman, Marek Druzdzel, Daniel Garcia (1998). XML BIF version 0.3. URL http://www-2.cs.cmu.edu/ fgcozman/Research/InterchangeFormat/.

5.4.76 W-Bagging

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **P:** Size of each bag, as a percentage of the training set size. (default 100) (real; $-\infty$-$+\infty$)
- ○ **O:** Calculate the out of bag error. (boolean; default: false)
- ○ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- ○ **I:** Number of iterations. (default 10) (real; $-\infty$-$+\infty$)

- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for bagging a classifier to reduce variance.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Leo Breiman (1996). Bagging predictors. Machine Learning. 24(2):123-140.

5.4.77 W-BayesNet

Group: Learner.Supervised.Weka.Bayes

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Do not use ADTree data structure (string; default: '-Q')
- **B:** BIF file to compare with (string)

- **Q:** Search algorithm (string; default: 'weka.classifiers.bayes.net.search.local.K2')
- **E:** Estimator algorithm (string; default: 'weka.classifiers.bayes.net.estimate.SimpleEstimator')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Bayes Network learning using various search algorithms and quality measures.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.78 W-BayesNetGenerator

Group: Learner.Supervised.Weka.Net

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Generate network (instead of instances) (boolean; default: false)
- **N:** Nr of nodes (real; -∞-+∞)
- **A:** Nr of arcs (real; -∞-+∞)
- **M:** Nr of instances (real; -∞-+∞)
- **C:** Cardinality of the variables (real; -∞-+∞)
- **S:** Seed for random number generator (real; -∞-+∞)
- **F:** The BIF file to obtain the structure from. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Bayes Network learning using various search algorithms and quality measures.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.79 W-BayesianLogisticRegression

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Show Debugging Output (string; default: '-Tl')
- **P:** Distribution of the Prior (1=Gaussian, 2=Laplacian) (default: 1=Gaussian) (real; $-\infty$-$+\infty$)
- **H:** Hyperparameter Selection Method (1=Norm-based, 2=CV-based, 3=specific value) (default: 1=Norm-based) (real; $-\infty$-$+\infty$)
- **V:** Specified Hyperparameter Value (use in conjunction with -H 3) (default: 0.27) (real; $-\infty$-$+\infty$)
- **R:** Hyperparameter Range (use in conjunction with -H 2) (format: R:start-end,multiplier OR L:val(1), val(2), ..., val(n)) (default: R:0.01-316,3.16) (string; default: 'R:0.01-316,3.16')
- **Tl:** Tolerance Value (default: 0.0005) (real; $-\infty$-$+\infty$)

- **S:** Threshold Value (default: 0.5) (real; -∞-+∞)
- **F:** Number Of Folds (use in conjuction with -H 2) (default: 2) (real; -∞-+∞)
- **I:** Max Number of Iterations (default: 100) (real; -∞-+∞)
- **N:** Normalize the data (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: binominal attributes, numerical attributes, binominal label

Short description: Implements Bayesian Logistic Regression for both Gaussian and Laplace Priors.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Alexander Genkin, David D. Lewis, David Madigan (2004). Large-scale bayesian logistic regression for text categorization. URL http://www.stat.rutgers.edu/ madigan/PAPERS/shortFat-v3a.pdf.

5.4.80 W-CLOPE

Group: Learner.Unsupervised.Clustering.Weka

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)

- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **R:** Repulsion (default 2.6) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Yiling Yang, Xudong Guan, Jinyuan You: CLOPE: a fast and effective clustering algorithm for transactional data.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

Further information: Yiling Yang, Xudong Guan, Jinyuan You: CLOPE: a fast and effective clustering algorithm for transactional data. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 682-687, 2002.

5.4.81 W-CitationKNN

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **R:** Number of Nearest References (default 1) (string; default: '1')
- **C:** Number of Nearest Citers (default 1) (string; default: '1')
- **H:** Rank of the Hausdorff Distance (default 1) (string; default: '1')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Modified version of the Citation kNN multi instance classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Jun Wang, Zucker, Jean-Daniel: Solving Multiple-Instance Problem: A Lazy Learning Approach. In: 17th International Conference on Machine Learning, 1119-1125, 2000.

5.4.82 W-ClassBalancedND

Group: Learner.Supervised.Weka.Nesteddichotomies

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A meta classifier for handling multi-class datasets with 2-class classifiers by building a random class-balanced tree structure.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Lin Dong, Eibe Frank, Stefan Kramer: Ensembles of Balanced Nested Dichotomies for Multi-class Problems. In: PKDD, 84-95, 2005.

Eibe Frank, Stefan Kramer: Ensembles of nested dichotomies for multi-class problems. In: Twenty-first International Conference on Machine Learning, 2004.

5.4.83 W-ClassificationViaClustering

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A simple meta-classifier that uses a clusterer for classification.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.84 W-Cobweb

Group: Learner.Unsupervised.Clustering.Weka

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **A:** Acuity. (default=1.0) (real; $-\infty$-$+\infty$)
- **C:** Cutoff. (default=0.002) (real; $-\infty$-$+\infty$)
- **S:** Random number seed. (default 42) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Class implementing the Cobweb and Classit clustering algorithms.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

Further information: D. Fisher (1987). Knowledge acquisition via incremental conceptual clustering. Machine Learning. 2(2):139-172.

J. H. Gennari, P. Langley, D. Fisher (1990). Models of incremental concept formation. Artificial Intelligence. 40:11-61.

5.4.85 W-ComplementNaiveBayes

Group: Learner.Supervised.Weka.Bayes

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **N:** Normalize the word weights for each class (boolean; default: false)
- ◦ **S:** Smoothing value to avoid zero WordGivenClass probabilities (default=1.0). (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for building and using a Complement class Naive Bayes classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Jason D. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger: Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In: ICML, 616-623, 2003.

5.4.86 W-ConjunctiveRule

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **N:** Set number of folds for REP One fold is used as pruning set. (default 3) (real; $-\infty$-$+\infty$)
- ○ **R:** Set if NOT uses randomization (default:use randomization) (boolean; default: false)
- ○ **E:** Set whether consider the exclusive expressions for nominal attributes (default false) (boolean; default: false)
- ○ **M:** Set the minimal weights of instances within a split. (default 2.0) (real; $-\infty$-$+\infty$)
- ○ **P:** Set number of antecedents for pre-pruning if -1, then REP is used (default -1) (real; $-\infty$-$+\infty$)
- ○ **S:** Set the seed of randomization (default 1) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: This class implements a single conjunctive rule learner that can predict for numeric and nominal class labels.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.87 W-CostSensitiveClassifier

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **M:** Minimize expected misclassification cost. Default is to reweight training instances according to costs per class (boolean; default: false)
- **C:** File name of a cost matrix to use. If this is not supplied, a cost matrix will be loaded on demand. The name of the on-demand file is the relation name of the training data plus ”.cost”, and the path to the on-demand file is specified with the -N option. (string)
- **N:** Name of a directory to search for cost files when loading costs on demand (default current directory). (string; default: '/home/ingo/workspace/yale')
- **cost-matrix:** The cost matrix in Matlab single line format. (string)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A metaclassifier that makes its base classifier cost-sensitive.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.88 W-DMNBtext

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, updatable, weighted examples

Short description: Class for building and using a Discriminative Multinomial Naive Bayes classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Jiang Su,Harry Zhang,Charles X. Ling,Stan Matwin: Discriminative Parameter Learning for Bayesian Networks. In: ICML 2008', 2008.

5.4.89 W-DTNB

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **X:** Use cross validation to evaluate features. Use number of folds = 1 for leave one out CV. (Default = leave one out CV) (real; $-\infty$-$+\infty$)
- **E:** Performance evaluation measure to use for selecting attributes. (Default = accuracy for discrete class and rmse for numeric class) (string)
- **I:** Use nearest neighbour instead of global table majority. (boolean; default: false)
- **R:** Display decision table rules. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for building and using a decision table/naive bayes hybrid classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Mark Hall, Eibe Frank: Combining Naive Bayes and Decision Tables. In: Proceedings of the 21st Florida Artificial Intelligence Society Conference (FLAIRS), ???-???, 2008.

5.4.90 W-Dagging

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **F:** The number of folds for splitting the training set into smaller chunks for the base classifier. (default 10) (real; $-\infty$-$+\infty$)
- **verbose:** Whether to print some more information during building the classifier. (default is off) (boolean; default: false)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: This meta classifier creates a number of disjoint, stratified folds out of the data and feeds each chunk of data to a copy of the supplied base classifier.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ting, K. M., Witten, I. H.: Stacking Bagged and Dagged Models. In: Fourteenth international Conference on Machine Learning, San Francisco, CA, 367-375, 1997.

5.4.91 W-DataNearBalancedND

Group: Learner.Supervised.Weka.Nesteddichotomies

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Random number seed. (default 1) (real; -∞-+∞)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A meta classifier for handling multi-class datasets with 2-class classifiers by building a random data-balanced tree structure.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Lin Dong, Eibe Frank, Stefan Kramer: Ensembles of Balanced Nested Dichotomies for Multi-class Problems. In: PKDD, 84-95, 2005.

Eibe Frank, Stefan Kramer: Ensembles of nested dichotomies for multi-class problems. In: Twenty-first International Conference on Machine Learning, 2004.

5.4.92 W-DecisionStump

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: Class for building and using a decision stump.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.93 W-DecisionTable

Group: Learner.Supervised.Weka.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Full class name of search method, followed by its options. eg: "weka.attributeSelection.BestFi -D 1" (default weka.attributeSelection.BestFirst) (string; default: 'weka.attributeSelection.Best -D 1 -N 5')
- **X:** Use cross validation to evaluate features. Use number of folds = 1 for leave one out CV. (Default = leave one out CV) (real; $-\infty$-$+\infty$)
- **E:** Performance evaluation measure to use for selecting attributes. (Default = accuracy for discrete class and rmse for numeric class) (string)
- **I:** Use nearest neighbour instead of global table majority. (boolean; default: false)
- **R:** Display decision table rules. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: Class for building and using a simple decision table majority classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ron Kohavi: The Power of Decision Tables. In: 8th European Conference on Machine Learning, 174-189, 1995.

5.4.94 W-Decorate

Group: Learner.Supervised.Weka.Meta

Required input:	Generated output:
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **E:** Desired size of ensemble. (default 10) (real; $-\infty$-$+\infty$)
- ◦ **R:** Factor that determines number of artificial examples to generate. Specified proportional to training set size. (default 1.0) (real; $-\infty$-$+\infty$)
- ◦ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- ◦ **I:** Number of iterations. (default 10) (real; $-\infty$-$+\infty$)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: DECORATE is a meta-learner for building diverse ensembles of classifiers by using specially constructed artificial training examples.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: P. Melville, R. J. Mooney: Constructing Diverse Classifier Ensembles Using Artificial Training Examples. In: Eighteenth International Joint Conference on Artificial Intelligence, 505-510, 2003.

P. Melville, R. J. Mooney (2004). Creating Diversity in Ensembles Using Artificial Data. Information Fusion: Special Issue on Diversity in Multiclassifier Systems..

5.4.95 W-EM

Group: Learner.Unsupervised.Clustering.Weka

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **N:** number of clusters. If omitted or -1 specified, then cross validation is used to select the number of clusters. (real; $-\infty$-$+\infty$)
- **I:** max iterations. (default 100) (real; $-\infty$-$+\infty$)
- **V:** verbose. (boolean; default: false)
- **M:** minimum allowable standard deviation for normal density computation (default 1e-6) (real; $-\infty$-$+\infty$)
- **O:** Display model in old format (good when there are many clusters) (boolean; default: false)
- **S:** Random number seed. (default 100) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Simple EM (expectation maximisation) class.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

5.4.96 W-END

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **I:** Number of iterations. (default 10) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A meta classifier for handling multi-class datasets with 2-class classifiers by building an ensemble of nested dichotomies.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Lin Dong, Eibe Frank, Stefan Kramer: Ensembles of Balanced Nested Dichotomies for Multi-class Problems. In: PKDD, 84-95, 2005.

Eibe Frank, Stefan Kramer: Ensembles of nested dichotomies for multi-class problems. In: Twenty-first International Conference on Machine Learning, 2004.

5.4.97 W-EditableBayesNet

Group: Learner.Supervised.Weka.Net

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** Do not use ADTree data structure (string; default: '-Q')
- ○ **B:** BIF file to compare with (string)
- ○ **Q:** Search algorithm (string; default: 'weka.classifiers.bayes.net.search.local.K2')
- ○ **E:** Estimator algorithm (string; default: 'weka.classifiers.bayes.net.estimate.SimpleEstimator')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Bayes Network learning using various search algorithms and quality measures.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.98 W-EnsembleSelection

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **L:** Specifies the Model Library File, continuing the list of all models. (string)
- **B:** Set the number of bags, i.e., number of iterations to run the ensemble selection algorithm. (real; $-\infty$-$+\infty$)
- **E:** Set the ratio of library models that will be randomly chosen to populate each bag of models. (real; $-\infty$-$+\infty$)
- **V:** Set the ratio of the training data set that will be reserved for validation. (real; $-\infty$-$+\infty$)
- **H:** Set the number of hillclimbing iterations to be performed on each model bag. (real; $-\infty$-$+\infty$)
- **I:** Set the the ratio of the ensemble library that the sort initialization algorithm will be able to choose from while initializing the ensemble for each model bag (real; $-\infty$-$+\infty$)
- **X:** Sets the number of cross-validation folds. (real; $-\infty$-$+\infty$)
- **P:** Specify the metric that will be used for model selection during the hillclimbing algorithm. Valid metrics are: accuracy, rmse, roc, precision, recall, fscore, all (string; default: 'rmse')

- **A:** Specifies the algorithm to be used for ensemble selection. Valid algorithms are: "forward" (default) for forward selection. "backward" for backward elimination. "both" for both forward and backward elimination. "best" to simply print out top performer from the ensemble library "library" to only train the models in the ensemble library (string; default: 'forward')
- **R:** Flag whether or not models can be selected more than once for an ensemble. (string; default: '-G')
- **G:** Whether sort initialization greedily stops adding models when performance degrades. (string; default: '-S')
- **O:** Flag for verbose output. Prints out performance of all selected models. (boolean; default: false)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Combines several classifiers using the ensemble selection method.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Rich Caruana, Alex Niculescu, Geoff Crew,, Alex Ksikes: Ensemble Selection from Libraries of Models. In: 21st International Conference on Machine Learning, 2004.

5.4.99 W-FLR

Group: Learner.Supervised.Weka.Misc

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **R:** Set vigilance parameter rhoa. (a float in range [0,1]) (real; $-\infty$-$+\infty$)
- ○ **B:** Set boundaries File Note: The boundaries file is a simple text file containing a row with a Fuzzy Lattice defining the metric space. For example, the boundaries file could contain the following the metric space for the iris dataset: [4.3 7.9] [2.0 4.4] [1.0 6.9] [0.1 2.5] in Class: -1 This lattice just contains the min and max value in each dimension. In other kind of problems this may not be just a min-max operation, but it could contain limits defined by the problem itself. Thus, this option should be set by the user. If ommited, the metric space used contains the mins and maxs of the training split. (string)
- ○ **Y:** Show Rules (string; default: '')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label

Short description: Fuzzy Lattice Reasoning Classifier (FLR) v5.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: I. N. Athanasiadis, V. G. Kaburlasos, P. A. Mitkas, V. Petridis: Applying Machine Learning Techniques on Air Quality Data for Real-Time Decision Support. In: 1st Intl. NAISO Symposium on Information Technologies in Environmental Engineering (ITEE-2003), Gdansk, Poland, 2003.

V. G. Kaburlasos, I. N. Athanasiadis, P. A. Mitkas, V. Petridis (2003). Fuzzy Lattice Reasoning (FLR) Classifier and its Application on Improved Estimation of Ambient Ozone Concentration.

5.4.100 W-FT

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Binary splits (convert nominal attributes to binary ones) (boolean; default: false)
- **P:** Use error on probabilities instead of misclassification error for stopping criterion of LogitBoost. (boolean; default: false)
- **I:** Set fixed number of iterations for LogitBoost (instead of using cross-validation) (real; $-\infty$-$+\infty$)
- **F:** Set Funtional Tree type to be generate: 0 for FT, 1 for FTLeaves and 2 for FTInner (real; $-\infty$-$+\infty$)
- **M:** Set minimum number of instances at which a node can be split (default 15) (real; $-\infty$-$+\infty$)
- **W:** Set beta for weight trimming for LogitBoost. Set to 0 (default) for no weight trimming. (real; $-\infty$-$+\infty$)
- **A:** The AIC is used to choose the best iteration. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Classifier for building 'Functional trees', which are classification trees that could have logistic regression functions at the inner nodes and/or leaves.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Joao Gama (2004). Functional Trees.

Niels Landwehr, Mark Hall, Eibe Frank (2005). Logistic Model Trees.

5.4.101 W-FarthestFirst

Group: Learner.Unsupervised.Clustering.Weka

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **N:** number of clusters. (default = 2). (real; -∞-+∞)
- **S:** Random number seed. (default 1) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Cluster data using the FarthestFirst algorithm.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

Further information: Hochbaum, Shmoys (1985). A best possible heuristic for the k-center problem. Mathematics of Operations Research. 10(2):180-184.

Sanjoy Dasgupta: Performance Guarantees for Hierarchical Clustering. In: 15th Annual Conference on Computational Learning Theory, 351-363, 2002.

5.4.102 W-GaussianProcesses

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

○ **L:** Level of Gaussian Noise. (default: 1.0) (real; $-\infty$-$+\infty$)

○ **N:** Whether to 0=normalize/1=standardize/2=neither. (default: 0=normalize) (real; $-\infty$-$+\infty$)

○ **K:** The Kernel to use. (default: weka.classifiers.functions.supportVector.PolyKernel) (string; default: 'weka.classifiers.functions.supportVector.RBFKernel -C 250007 -G 1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label

Short description: Implements Gaussian Processes for regression without hyperparameter-tuning.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: David J.C. Mackay (1998). Introduction to Gaussian Processes. Dept. of Physics, Cambridge University, UK.

5.4.103 W-GeneralizedSequentialPatterns

Group: Learner.Unsupervised.Itemsets.Weka

Required input:	**Generated output:**
• ExampleSet	• WekaAssociator

Parameters:

- ○ **D:** If set, algorithm is run in debug mode and may output additional info to the console (boolean; default: false)
- ○ **S:** The miminum support threshold. (default: 0.9) (real; -∞-+∞)
- ○ **I:** The attribute number representing the data sequence ID. (default: 0) (real; -∞-+∞)
- ○ **F:** The attribute numbers used for result filtering. (default: -1) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Class implementing a GSP algorithm for discovering sequential patterns in a sequential data set.

Description: Performs the Weka association rule learner with the same name. The operator returns a result object containing the rules found by the association learner. In contrast to models generated by normal learners, the association rules cannot be applied to an example set. Hence, there is no way to evaluate the performance of association rules yet. See the Weka javadoc for further operator and parameter descriptions.

Further information: Ramakrishnan Srikant, Rakesh Agrawal (1996). Mining Sequential Patterns: Generalizations and Performance Improvements.

5.4.104 W-Grading

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **M:** Full name of meta classifier, followed by options. (default: "weka.classifiers.rules.Zero") (string; default: 'weka.classifiers.rules.ZeroR ')

○ **X:** Sets the number of cross-validation folds. (real; -∞-+∞)

○ **S:** Random number seed. (default 1) (real; -∞-+∞)

○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Implements Grading.

Description: Performs the ensemble learning scheme of Weka with the same name. An arbitrary number of other Weka learning schemes must be embedded as inner operators. See the Weka javadoc for further classifier and parameter descriptions.

Further information: A.K. Seewald, J. Fuernkranz: An Evaluation of Grading Classifiers. In: Advances in Intelligent Data Analysis: 4th International Conference, Berlin/Heidelberg/New York/Tokyo, 115-124, 2001.

5.4.105 W-GridSearch

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **E:** Determines the parameter used for evaluation: CC = Correlation coefficient RMSE = Root mean squared error RRSE = Root relative squared error MAE = Mean absolute error RAE = Root absolute error COMB = Combined = (1-abs(CC)) + RRSE + RAE ACC = Accuracy (default: CC) (string; default: 'CC')
- **y-property:** The Y option to test (without leading dash). (default: classifier.ridge) (string; default: 'classifier.ridge')
- **y-min:** The minimum for Y. (default: -10) (real; -∞-+∞)
- **y-max:** The maximum for Y. (default: +5) (real; -∞-+∞)
- **y-step:** The step size for Y. (default: 1) (real; -∞-+∞)
- **y-base:** The base for Y. (default: 10) (real; -∞-+∞)
- **y-expression:** The expression for Y. Available parameters: BASE FROM TO STEP I - the current iteration value (from 'FROM' to 'TO' with stepsize 'STEP') (default: 'pow(BASE,I)') (string; default: 'pow(BASE,I)')
- **filter:** The filter to use (on X axis). Full classname of filter to include, followed by scheme options. (default: weka.filters.supervised.attribute.PLSFilter)

(string; default: 'weka.filters.supervised.attribute.PLSFilter -C 20 -M -A PLS1 -P center')

- **x-property:** The X option to test (without leading dash). (default: filter.numComponents) (string; default: 'filter.numComponents')
- **x-min:** The minimum for X. (default: +5) (real; $-\infty$-$+\infty$)
- **x-max:** The maximum for X. (default: +20) (real; $-\infty$-$+\infty$)
- **x-step:** The step size for X. (default: 1) (real; $-\infty$-$+\infty$)
- **x-base:** The base for X. (default: 10) (real; $-\infty$-$+\infty$)
- **x-expression:** The expression for the X value. Available parameters: BASE MIN MAX STEP I - the current iteration value (from 'FROM' to 'TO' with stepsize 'STEP') (default: 'pow(BASE,I)') (string; default: 'I')
- **extend-grid:** Whether the grid can be extended. (default: no) (boolean; default: false)
- **max-grid-extensions:** The maximum number of grid extensions (-1 is unlimited). (default: 3) (string)
- **sample-size:** The size (in percent) of the sample to search the inital grid with. (default: 100) (real; $-\infty$-$+\infty$)
- **traversal:** The type of traversal for the grid. (default: COLUMN-WISE) (string; default: 'COLUMN-WISE')
- **log-file:** The log file to log the messages to. (default: none) (string; default: '/home/ingo/workspace/yale')
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, numerical label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Performs a grid search of parameter pairs for the a classifier (Y-axis, default is LinearRegression with the "Ridge" parameter) and the PLSFilter (X-axis, "number of Components") and chooses the best pair found for the actual predicting.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.106 W-HNB

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: Contructs Hidden Naive Bayes classification model with high classification accuracy and AUC.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: H. Zhang, L. Jiang, J. Su: Hidden Naive Bayes. In: Twentieth National Conference on Artificial Intelligence, 919-924, 2005.

5.4.107 W-HotSpot

Group: Learner.Unsupervised.Itemsets.Weka

Required input:	**Generated output:**
• ExampleSet	• WekaAssociator

Parameters:

- ◦ **c:** The target index. (default = last) (string; default: 'last')
- ◦ **V:** The target value (nominal target only, default = first) (string; default: 'first')
- ◦ **L:** Minimize rather than maximize. (boolean; default: false)
- ◦ -**S:** Minimum value count (nominal target)/segment size (numeric target). Values between 0 and 1 are interpreted as a percentage of the total population; values ¿ 1 are interpreted as an absolute number of instances (default = 0.3) (string)
- ◦ -**M:** Maximum branching factor (default = 2) (string)
- ◦ -**I:** Minimum improvement in target value in order to add a new branch/test (default = 0.01 (1
- ◦ -**D:** Output debugging info (duplicate rule lookup hash table stats) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: HotSpot learns a set of rules (displayed in a tree-like structure) that maximize/minimize a target variable/value of interest.

Description: Performs the Weka association rule learner with the same name. The operator returns a result object containing the rules found by the association learner. In contrast to models generated by normal learners, the association rules cannot be applied to an example set. Hence, there is no way to evaluate the performance of association rules yet. See the Weka javadoc for further operator and parameter descriptions.

5.4.108 W-HyperPipes

Group: Learner.Supervised.Weka.Misc

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Class implementing a HyperPipe classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.109 W-IB1

Group: Learner.Supervised.Weka.Lazy

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, updatable

Short description: Nearest-neighbour classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: D. Aha, D. Kibler (1991). Instance-based learning algorithms. Machine Learning. 6:37-66.

5.4.110 W-IBk

Group: Learner.Supervised.Weka.Lazy

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **I:** Weight neighbours by the inverse of their distance (use when k ¿ 1) (boolean; default: false)

- **F:** Weight neighbours by 1 - their distance (use when k ¿ 1) (boolean; default: false)
- **K:** Number of nearest neighbours (k) used in classification. (Default = 1) (real; -∞-+∞)
- **E:** Minimise mean squared error rather than mean absolute error when using -X option with numeric prediction. (boolean; default: false)
- **W:** Maximum number of training instances maintained. Training instances are dropped FIFO. (Default = no window) (real; -∞-+∞)
- **X:** Select the number of nearest neighbours between 1 and the k value specified using hold-one-out evaluation on the training data (use when k ¿ 1) (boolean; default: false)
- **A:** The nearest neighbour search algorithm to use (default: weka.core.neighboursearch.LinearNNSearch). (string; default: 'weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, updatable, weighted examples

Short description: K-nearest neighbours classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: D. Aha, D. Kibler (1991). Instance-based learning algorithms. Machine Learning. 6:37-66.

5.4.111 W-Id3

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: Class for constructing an unpruned decision tree based on the ID3 algorithm.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: R. Quinlan (1986). Induction of decision trees. Machine Learning. 1(1):81-106.

5.4.112 W-IsotonicRegression

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, numerical label, weighted examples

Short description: Learns an isotonic regression model.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.113 W-J48

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **U:** Use unpruned tree. (boolean; default: false)
- **C:** Set confidence threshold for pruning. (default 0.25) (real; $-\infty$-$+\infty$)
- **M:** Set minimum number of instances per leaf. (default 2) (real; $-\infty$-$+\infty$)
- **R:** Use reduced error pruning. (boolean; default: false)
- **N:** Set number of folds for reduced error pruning. One fold is used as pruning set. (default 3) (string)
- **B:** Use binary splits only. (boolean; default: false)

- **S:** Don't perform subtree raising. (boolean; default: false)
- **L:** Do not clean up after the tree has been built. (boolean; default: false)
- **A:** Laplace smoothing for predicted probabilities. (boolean; default: false)
- **Q:** Seed for random data shuffling (default 1). (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for generating a pruned or unpruned C4.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

5.4.114 W-J48graft

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **U:** Use unpruned tree. (boolean; default: false)
- **C:** Set confidence threshold for pruning. (default 0.25) (real; $-\infty$-$+\infty$)
- **M:** Set minimum number of instances per leaf. (default 2) (real; $-\infty$-$+\infty$)
- **B:** Use binary splits only. (boolean; default: false)

- ○ **S:** Don't perform subtree raising. (boolean; default: false)
- ○ **L:** Do not clean up after the tree has been built. (boolean; default: false)
- ○ **A:** Laplace smoothing for predicted probabilities. (note: this option only affects initial tree; grafting process always uses laplace). (boolean; default: false)
- ○ **E:** Relabel when grafting. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for generating a grafted (pruned or unpruned) C4.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Geoff Webb: Decision Tree Grafting From the All-Tests-But-One Partition. In: , San Francisco, CA, 1999.

5.4.115 W-JRip

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **F:** Set number of folds for REP One fold is used as pruning set. (default 3) (real; $-\infty$-$+\infty$)

- **N:** Set the minimal weights of instances within a split. (default 2.0) (real; -∞-+∞)
- **O:** Set the number of runs of optimizations. (Default: 2) (real; -∞-+∞)
- **D:** Set whether turn on the debug mode (Default: false) (boolean; default: false)
- **S:** The seed of randomization (Default: 1) (real; -∞-+∞)
- **E:** Whether NOT check the error rate¿=0.5 in stopping criteria (default: check) (boolean; default: false)
- **P:** Whether NOT use pruning (default: use pruning) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: This class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: William W. Cohen: Fast Effective Rule Induction. In: Twelfth International Conference on Machine Learning, 115-123, 1995.

5.4.116 W-JythonClassifier

Group: Learner.Supervised.Weka.Classifiers

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **J:** The Jython module to load (full path) Options after '–' will be passed on to the Jython module. (string; default: '/home/ingo/workspace/yale')
- **P:** The paths to add to 'sys.path' (comma-separated list). (string)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: A wrapper class for Jython code.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.117 W-KStar

Group: Learner.Supervised.Weka.Lazy

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Manual blend setting (default 20(real; $-\infty$-$+\infty$)
- **E:** Enable entropic auto-blend setting (symbolic class only) (boolean; default: false)
- **M:** Specify the missing value treatment mode (default a) Valid options are: a(verage), d(elete), m(axdiff), n(ormal) (string; default: 'a')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, updatable

Short description: K* is an instance-based classifier, that is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 108-114, 1995.

5.4.118 W-LBR

Group: Learner.Supervised.Weka.Lazy

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: Lazy Bayesian Rules Classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Zijian Zheng, G. Webb (2000). Lazy Learning of Bayesian Rules. Machine Learning. 4(1):53-84.

5.4.119 W-LMT

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Binary splits (convert nominal attributes to binary ones) (boolean; default: false)
- **R:** Split on residuals instead of class values (boolean; default: false)
- **C:** Use cross-validation for boosting at all nodes (i.e., disable heuristic) (boolean; default: false)
- **P:** Use error on probabilities instead of misclassification error for stopping criterion of LogitBoost. (boolean; default: false)
- **I:** Set fixed number of iterations for LogitBoost (instead of using cross-validation) (real; -∞-+∞)
- **M:** Set minimum number of instances at which a node can be split (default 15) (real; -∞-+∞)
- **W:** Set beta for weight trimming for LogitBoost. Set to 0 (default) for no weight trimming. (real; -∞-+∞)
- **A:** The AIC is used to choose the best iteration. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Classifier for building 'logistic model trees', which are classification trees with logistic regression functions at the leaves.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Niels Landwehr, Mark Hall, Eibe Frank (2005). Logistic Model Trees. Machine Learning. 95(1-2):161-205.

Marc Sumner, Eibe Frank, Mark Hall: Speeding up Logistic Model Tree Induction. In: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, 675-683, 2005.

5.4.120 W-LWL

Group: Learner.Supervised.Weka.Lazy

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **A:** The nearest neighbour search algorithm to use (default: weka.core.neighboursearch.LinearNN (string; default: 'weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last"')
- **K:** Set the number of neighbours used to set the kernel bandwidth. (default all) (real; -∞-+∞)
- **U:** Set the weighting kernel shape to use. 0=Linear, 1=Epanechnikov, 2=Tricube, 3=Inverse, 4=Gaussian. (default 0 = Linear) (real; -∞-+∞)

- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- ○ **W:** Full name of base classifier. (default: weka.classifiers.trees.DecisionStump) (string; default: 'weka.classifiers.trees.DecisionStump')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, updatable, weighted examples

Short description: Locally weighted learning.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Eibe Frank, Mark Hall, Bernhard Pfahringer: Locally Weighted Naive Bayes. In: 19th Conference in Uncertainty in Artificial Intelligence, 249-256, 2003.

C. Atkeson, A. Moore, S. Schaal (1996). Locally weighted learning. AI Review..

5.4.121 W-LeastMedSq

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **S:** Set sample size (default: 4) (real; $-\infty$-$+\infty$)

- **G:** Set the seed used to generate samples (default: 0) (string; default: '0')
- **D:** Produce debugging output (default no debugging output) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label

Short description: Implements a least median sqaured linear regression utilising the existing weka LinearRegression class to form predictions.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Peter J. Rousseeuw, Annick M. Leroy (1987). Robust regression and outlier detection. .

5.4.122 W-LinearRegression

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Produce debugging output. (default no debugging output) (boolean; default: false)
- **S:** Set the attribute selection method to use. 1 = None, 2 = Greedy. (default 0 = M5' method) (real; $-\infty$-$+\infty$)
- **C:** Do not try to eliminate colinear attributes. (boolean; default: false)

- **R:** Set ridge parameter (default 1.0e-8). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label, weighted examples

Short description: Class for using linear regression for prediction.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.123 W-Logistic

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Turn on debugging output. (boolean; default: false)
- **R:** Set the ridge in the log-likelihood. (real; -∞-+∞)
- **M:** Set the maximum number of iterations (default -1, until convergence). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for building and using a multinomial logistic regression model with a ridge estimator.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: le Cessie, S., van Houwelingen, J.C. (1992). Ridge Estimators in Logistic Regression. Applied Statistics. 41(1):191-201.

5.4.124 W-LogisticBase

Group: Learner.Supervised.Weka.Lmt

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: weighted examples

Short description: The weka learner W-LogisticBase

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.125 W-LogitBoost

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **Q:** Use resampling instead of reweighting for boosting. (boolean; default: false)
- ○ **P:** Percentage of weight mass to base training on. (default 100, reduce to around 90 speed up) (real; -∞-+∞)
- ○ **F:** Number of folds for internal cross-validation. (default 0 – no cross-validation) (real; -∞-+∞)
- ○ **R:** Number of runs for internal cross-validation. (default 1) (real; -∞-+∞)
- ○ **L:** Threshold on the improvement of the likelihood. (default -Double.MAX_VALUE) (real; -∞-+∞)
- ○ **H:** Shrinkage parameter. (default 1) (real; -∞-+∞)
- ○ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ○ **I:** Number of iterations. (default 10) (real; -∞-+∞)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for performing additive logistic regression.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: J. Friedman, T. Hastie, R. Tibshirani (1998). Additive Logistic Regression: a Statistical View of Boosting. Stanford University.

5.4.126 W-M5P

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **N:** Use unpruned tree/rules (boolean; default: false)
- ○ **U:** Use unsmoothed predictions (boolean; default: false)
- ○ **R:** Build regression tree/rule rather than a model tree/rule (boolean; default: false)
- ○ **M:** Set minimum number of instances per leaf (default 4) (real; $-\infty$-$+\infty$)
- ○ **L:** Save instances at the nodes in the tree (for visualization purposes) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label

Short description: The weka learner W-M5P

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ross J. Quinlan: Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial Intelligence, Singapore, 343-348, 1992.

Y. Wang, I. H. Witten: Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European Conference on Machine Learning, 1997.

5.4.127 W-M5Rules

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **N:** Use unpruned tree/rules (boolean; default: false)
- ◦ **U:** Use unsmoothed predictions (boolean; default: false)
- ◦ **R:** Build regression tree/rule rather than a model tree/rule (boolean; default: false)
- ◦ **M:** Set minimum number of instances per leaf (default 4) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label

Short description: Generates a decision list for regression problems using separate-and-conquer.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Geoffrey Holmes, Mark Hall, Eibe Frank: Generating Rule Sets from Model Trees. In: Twelfth Australian Joint Conference on Artificial Intelligence, 1-12, 1999.

Ross J. Quinlan: Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial Intelligence, Singapore, 343-348, 1992.

Y. Wang, I. H. Witten: Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European Conference on Machine Learning, 1997.

5.4.128 W-MDD

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** Turn on debugging output. (boolean; default: false)
- ○ **N:** Whether to 0=normalize/1=standardize/2=neither. (default 1=standardize) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: Modified Diverse Density algorithm, with collective assumption.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Oded Maron (1998). Learning from ambiguity.

O. Maron, T. Lozano-Perez (1998). A Framework for Multiple Instance Learning. Neural Information Processing Systems. 10.

5.4.129 W-MIBoost

Group: Learner.Supervised.Weka.Mi

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** Turn on debugging output. (boolean; default: false)
- ○ **B:** The number of bins in discretization (default 0, no discretization) (real; -∞-+∞)
- ○ **R:** Maximum number of boost iterations. (default 10) (real; -∞-+∞)
- ○ **W:** Full name of classifier to boost. eg: weka.classifiers.bayes.NaiveBayes (string; default: 'weka.classifiers.rules.ZeroR')
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label

Short description: MI AdaBoost method, considers the geometric mean of posterior of instances inside a bag (arithmatic mean of log-posterior) and the expectation for a bag is taken inside the loss function.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.

5.4.130 W-MIDD

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** Turn on debugging output. (boolean; default: false)
- ○ **N:** Whether to 0=normalize/1=standardize/2=neither. (default 1=standardize) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: Re-implement the Diverse Density algorithm, changes the testing procedure.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Oded Maron (1998). Learning from ambiguity.

O. Maron, T. Lozano-Perez (1998). A Framework for Multiple Instance Learning. Neural Information Processing Systems. 10.

5.4.131 W-MIEMDD

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 1=standardize) (real; -∞-+∞)
- **S:** Random number seed. (default 1) (real; -∞-+∞)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: EMDD model builds heavily upon Dietterich's Diverse Density (DD) algorithm.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Qi Zhang, Sally A. Goldman: EM-DD: An Improved Multiple-Instance Learning Technique. In: Advances in Neural Information Processing Systems 14, 1073-108, 2001.

5.4.132 W-MILR

Group: Learner.Supervised.Weka.Mi

Required input:	Generated output:
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Turn on debugging output. (boolean; default: false)
- **R:** Set the ridge in the log-likelihood. (real; $-\infty$-$+\infty$)
- **A:** Defines the type of algorithm: 0. standard MI assumption 1. collective MI assumption, arithmetic mean for posteriors 2. collective MI assumption, geometric mean for posteriors (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: Uses either standard or collective multi-instance assumption, but within linear regression.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.133 W-MINND

Group: Learner.Supervised.Weka.Mi

Required input:	Generated output:
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **K:** Set number of nearest neighbour for prediction (default 1) (real; -∞-+∞)
- **S:** Set number of nearest neighbour for cleansing the training data (default 1) (real; -∞-+∞)
- **E:** Set number of nearest neighbour for cleansing the testing data (default 1) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: Multiple-Instance Nearest Neighbour with Distribution learner.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Xin Xu (2001). A nearest distribution approach to multiple-instance learning. Hamilton, NZ.

5.4.134 W-MIOptimalBall

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label, weighted examples

Short description: This classifier tries to find a suitable ball in the multiple-instance space, with a certain data point in the instance space as a ball center.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Peter Auer, Ronald Ortner: A Boosting Approach to Multiple Instance Learning. In: 15th European Conference on Machine Learning, 63-74, 2004.

5.4.135 W-MISMO

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

- **no-checks:** Turns off all checks - use with caution! Turning them off assumes that data is purely numeric, doesn't contain any missing values, and has a nominal class. Turning them off also means that no header information will be stored if the machine is linear. Finally, it also assumes that no instance has a weight equal to 0. (default: checks on) (boolean; default: false)
- **C:** The complexity constant C. (default 1) (real; -∞-+∞)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; -∞-+∞)
- **I:** Use MIminimax feature space. (boolean; default: false)
- **L:** The tolerance parameter. (default 1.0e-3) (real; -∞-+∞)
- **P:** The epsilon for round-off error. (default 1.0e-12) (real; -∞-+∞)
- **M:** Fit logistic models to SVM outputs. (boolean; default: false)
- **V:** The number of folds for the internal cross-validation. (default -1, use training data) (real; -∞-+∞)
- **W:** The random number seed. (default 1) (real; -∞-+∞)
- **K:** The Kernel to use. (default: weka.classifiers.functions.supportVector.PolyKernel) (string; default: 'weka.classifiers.mi.supportVector.MIPolyKernel -C 250007 -E 1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: J. Platt: Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, 1998.

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy (2001). Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation. 13(3):637-649.

5.4.136 W-MIWrapper

Group: Learner.Supervised.Weka.Mi

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **P:** The method used in testing: 1.arithmetic average 2.geometric average 3.max probability of positive bag. (default: 1) (real; $-\infty$-$+\infty$)
- ○ **A:** The type of weight setting for each single-instance: 0.keep the weight to be the same as the original value; 1.weight = 1.0 2.weight = 1.0/Total number of single-instance in the corresponding bag 3. weight = Total number of single-instance / (Total number of bags * Total number of single-instance in the corresponding bag). (default: 3) (real; $-\infty$-$+\infty$)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- ○ **W:** Full name of base classifier. (default: weka.classifiers.rules.ZeroR) (string; default: 'weka.classifiers.rules.ZeroR')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: A simple Wrapper method for applying standard propositional learners to multi-instance data.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: E. T. Frank, X. Xu (2003). Applying propositional learning algorithms to multi-instance data. Department of Computer Science, University of Waikato, Hamilton, NZ.

5.4.137 W-MetaCost

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **I:** Number of bagging iterations. (default 10) (real; $-\infty$-$+\infty$)
- **C:** File name of a cost matrix to use. If this is not supplied, a cost matrix will be loaded on demand. The name of the on-demand file is the relation name of the training data plus ".cost", and the path to the on-demand file is specified with the -N option. (string)
- **N:** Name of a directory to search for cost files when loading costs on demand (default current directory). (string; default: '/home/ingo/workspace/yale')
- **cost-matrix:** The cost matrix in Matlab single line format. (string)
- **P:** Size of each bag, as a percentage of the training set size. (default 100) (real; $-\infty$-$+\infty$)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: This metaclassifier makes its base classifier cost-sensitive using the method specified in

Pedro Domingos: MetaCost: A general method for making classifiers cost-sensitive.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Pedro Domingos: MetaCost: A general method for making classifiers cost-sensitive. In: Fifth International Conference on Knowledge Discovery and Data Mining, 155-164, 1999.

5.4.138 W-MinMaxExtension

Group: Learner.Supervised.Weka.Misc

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **M:** Use maximal extension (default: minimal extension) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: This class is an implementation of the minimal and maximal extension.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: S. Lievens, B. De Baets, K. Cao-Van (2006). A Probabilistic Framework for the Design of Instance-Based Supervised Ranking Algorithms in an Ordinal Setting. Annals of Operations Research..

Kim Cao-Van (2003). Supervised ranking: from semantics to algorithms.

Stijn Lievens (2004). Studie en implementatie van instantie-gebaseerde algoritmen voor gesuperviseerd rangschikken.

5.4.139 W-MultiBoostAB

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **C:** Number of sub-committees. (Default 3) (real; $-\infty$-$+\infty$)
- ○ **P:** Percentage of weight mass to base training on. (default 100, reduce to around 90 speed up) (real; $-\infty$-$+\infty$)
- ○ **Q:** Use resampling for boosting. (boolean; default: false)

- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **I:** Number of iterations. (default 10) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for boosting a classifier using the MultiBoosting method.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Geoffrey I. Webb (2000). MultiBoosting: A Technique for Combining Boosting and Wagging. Machine Learning. Vol.40(No.2).

5.4.140 W-MultiClassClassifier

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **M:** Sets the method to use. Valid values are 0 (1-against-all), 1 (random codes), 2 (exhaustive code), and 3 (1-against-1). (default 0) (real; -∞-+∞)
- **R:** Sets the multiplier when using random codes. (default 2.0) (real; -∞-+∞)
- **P:** Use pairwise coupling (only has an effect for 1-against1) (boolean; default: false)
- **S:** Random number seed. (default 1) (real; -∞-+∞)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A metaclassifier for handling multi-class datasets with 2-class classifiers.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.141 W-MultiScheme

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **X:** Use cross validation for model selection using the given number of folds. (default 0, is to use training error) (real; -∞-+∞)
- ○ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for selecting a classifier from among several using cross validation on the training data or the performance on the training data.

Description: Performs the ensemble learning scheme of Weka with the same name. An arbitrary number of other Weka learning schemes must be embedded as inner operators. See the Weka javadoc for further classifier and parameter descriptions.

5.4.142 W-MultilayerPerceptron

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- ∘ **L:** Learning Rate for the backpropagation algorithm. (Value should be between 0 - 1, Default = 0.3). (real; -∞-+∞)
- ∘ **M:** Momentum Rate for the backpropagation algorithm. (Value should be between 0 - 1, Default = 0.2). (real; -∞-+∞)
- ∘ **N:** Number of epochs to train through. (Default = 500). (real; -∞-+∞)
- ∘ **V:** Percentage size of validation set to use to terminate training (if this is non zero it can pre-empt num of epochs. (Value should be between 0 - 100, Default = 0). (real; -∞-+∞)
- ∘ **S:** The value used to seed the random number generator (Value should be ¿= 0 and and a long, Default = 0). (real; -∞-+∞)
- ∘ **E:** The consequetive number of errors allowed for validation testing before the netwrok terminates. (Value should be ¿ 0, Default = 20). (real; -∞-+∞)
- ∘ **G:** GUI will be opened. (Use this to bring up a GUI). (boolean; default: false)
- ∘ **A:** Autocreation of the network connections will NOT be done. (This will be ignored if -G is NOT set) (boolean; default: false)
- ∘ **B:** A NominalToBinary filter will NOT automatically be used. (Set this to not use a NominalToBinary filter). (boolean; default: false)
- ∘ **H:** The hidden layers to be created for the network. (Value should be a list of comma separated Natural numbers or the letters 'a' = (attribs + classes) / 2, 'i' = attribs, 'o' = classes, 't' = attribs .+ classes) for wildcard values, Default = a). (string; default: 'a')
- ∘ **C:** Normalizing a numeric class will NOT be done. (Set this to not normalize the class if it's numeric). (boolean; default: false)
- ∘ **I:** Normalizing the attributes will NOT be done. (Set this to not normalize the attributes). (boolean; default: false)
- ∘ **R:** Reseting the network will NOT be allowed. (Set this to not allow the network to reset). (boolean; default: false)
- ∘ **D:** Learning rate decay will occur. (Set this to cause the learning rate to decay). (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: A Classifier that uses backpropagation to classify instances.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.143 W-NBTree

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for generating a decision tree with naive Bayes classifiers at the leaves.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ron Kohavi: Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In: Second International Conference on Knoledge Discovery and Data Mining, 202-207, 1996.

5.4.144 W-ND

Group: Learner.Supervised.Weka.Nesteddichotomies

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ◦ **I:** Number of iterations. (default 10) (real; -∞-+∞)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A meta classifier for handling multi-class datasets with 2-class classifiers by building a random tree structure.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Lin Dong, Eibe Frank, Stefan Kramer: Ensembles of Balanced Nested Dichotomies for Multi-class Problems. In: PKDD, 84-95, 2005.

Eibe Frank, Stefan Kramer: Ensembles of nested dichotomies for multi-class problems. In: Twenty-first International Conference on Machine Learning, 2004.

5.4.145 W-NNge

Group: Learner.Supervised.Weka.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **G:** Number of attempts of generalisation. (real; $-\infty$-$+\infty$)
- **I:** Number of folder for computing the mutual information. (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, updatable

Short description: Nearest-neighbor-like algorithm using non-nested generalized exemplars (which are hyperrectangles that can be viewed as if-then rules).

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Brent Martin (1995). Instance-Based learning: Nearest Neighbor With Generalization. Hamilton, New Zealand.

Sylvain Roy (2002). Nearest Neighbor With Generalization. Christchurch, New Zealand.

5.4.146 W-NaiveBayes

Group: Learner.Supervised.Weka.Bayes

Deprecated: please use NaiveBayes instead.

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **N:** Normalize the word weights for each class (boolean; default: false)
- ◦ **S:** Smoothing value to avoid zero WordGivenClass probabilities (default=1.0). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for a Naive Bayes classifier using estimator classes.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Jason D. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger: Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In: ICML, 616-623, 2003.

5.4.147 W-NaiveBayesMultinomial

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for building and using a multinomial Naive Bayes classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Andrew Mccallum, Kamal Nigam: A Comparison of Event Models for Naive Bayes Text Classification. In: AAAI-98 Workshop on 'Learning for Text Categorization', 1998.

5.4.148 W-NaiveBayesMultinomialUpdateable

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, polynominal label, binominal label, updatable, weighted examples

Short description: Class for building and using a multinomial Naive Bayes classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Andrew Mccallum, Kamal Nigam: A Comparison of Event Models for Naive Bayes Text Classification. In: AAAI-98 Workshop on 'Learning for Text Categorization', 1998.

5.4.149 W-NaiveBayesSimple

Group: Learner.Supervised.Weka.Bayes

Deprecated: please use NaiveBayes instead.

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Class for building and using a simple Naive Bayes classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Richard Duda, Peter Hart (1973). Pattern Classification and Scene Analysis. Wiley, New York.

5.4.150 W-NaiveBayesUpdateable

Group: Learner.Supervised.Weka.Bayes

Deprecated: please use NaiveBayes instead.

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **K:** Use kernel density estimator rather than normal distribution for numeric attributes (boolean; default: false)
- **D:** Use supervised discretization to process numeric attributes (boolean; default: false)
- **O:** Display model in old format (good when there are many classes) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, updatable, weighted examples

Short description: Class for a Naive Bayes classifier using estimator classes.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995.

5.4.151 W-OLM

Group: Learner.Supervised.Weka.Misc

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **C:** Sets the classification type to be used. (Default: REG) (string; default: 'REG')
- **A:** Sets the averaging type used in phase 1 of the classifier. (Default: MEAN) (string; default: 'MEAN')
- **N:** If different from NONE, a nearest neighbour rule is fired when the rule base doesn't contain an example smaller than the instance to be classified (Default: NONE). (string; default: 'EUCL')

- ◦ **E:** Sets the extension type, i.e. the rule base to use. (Default: MIN) (string; default: 'MIN')
- ◦ **sort:** If set, the instances are also sorted within the same class before building the rule bases (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: This class is an implementation of the Ordinal Learning Method Further information regarding the algorithm and variants can be found in:

Arie Ben-David (1992).

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Arie Ben-David (1992). Automatic Generation of Symbolic Multiattribute Ordinal Knowledge-Based DSSs: methodology and Applications. Decision Sciences. 23:1357-1372.

Lievens, Stijn (2003-2004). Studie en implementatie van instantie-gebaseerde algoritmen voor gesuperviseerd rangschikken..

5.4.152 W-OSDL

Group: Learner.Supervised.Weka.Misc

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- ○ **C:** Sets the classification type to be used. (Default: MED) (string; default: 'MED')
- ○ **B:** Use the balanced version of the Ordinal Stochastic Dominance Learner (string)
- ○ **W:** Use the weighted version of the Ordinal Stochastic Dominance Learner (string)
- ○ **S:** Sets the value of the interpolation parameter (not with -W/T/P/L/U) (default: 0.5). (real; -∞-+∞)
- ○ **T:** Tune the interpolation parameter (not with -W/S) (default: off) (boolean; default: false)
- ○ **L:** Lower bound for the interpolation parameter (not with -W/S) (default: 0) (string)
- ○ **U:** Upper bound for the interpolation parameter (not with -W/S) (default: 1) (string)
- ○ **P:** Determines the step size for tuning the interpolation parameter, nl. (U-L)/P (not with -W/S) (default: 10) (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: This class is an implementation of the Ordinal Stochastic Dominance Learner.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: S. Lievens, B. De Baets, K. Cao-Van (2006). A Probabilistic Framework for the Design of Instance-Based Supervised Ranking Algorithms in an Ordinal Setting. Annals of Operations Research..

Kim Cao-Van (2003). Supervised ranking: from semantics to algorithms.

Stijn Lievens (2004). Studie en implementatie van instantie-gebaseerde algoritmen voor gesuperviseerd rangschikken.

5.4.153 W-OneR

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **B:** The minimum number of objects in a bucket (default: 6). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Class for building and using a 1R classifier; in other words, uses the minimum-error attribute for prediction, discretizing numeric attributes.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: R.C. Holte (1993). Very simple classification rules perform well on most commonly used datasets. Machine Learning. 11:63-91.

5.4.154 W-OrdinalClassClassifier

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Meta classifier that allows standard classification algorithms to be applied to ordinal class problems.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Eibe Frank, Mark Hall: A Simple Approach to Ordinal Classification. In: 12th European Conference on Machine Learning, 145-156, 2001.

5.4.155 W-PART

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **C:** Set confidence threshold for pruning. (default 0.25) (real; $-\infty$-$+\infty$)
- **M:** Set minimum number of objects per leaf. (default 2) (real; $-\infty$-$+\infty$)
- **R:** Use reduced error pruning. (boolean; default: false)
- **N:** Set number of folds for reduced error pruning. One fold is used as pruning set. (default 3) (string)
- **B:** Use binary splits only. (boolean; default: false)
- **U:** Generate unpruned decision list. (boolean; default: false)
- **Q:** Seed for random data shuffling (default 1). (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for generating a PART decision list.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Eibe Frank, Ian H. Witten: Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, 144-151, 1998.

5.4.156 W-PLSClassifier

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **filter:** The PLS filter to use. Full classname of filter to include, followed by scheme options. (default: weka.filters.supervised.attribute.PLSFilter) (string; default: 'weka.filters.supervised.attribute.PLSFilter -C 20 -M -A PLS1 -P center')
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, numerical label

Short description: A wrapper classifier for the PLSFilter, utilizing the PLSFilter's ability to perform predictions.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.157 W-PaceRegression

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** Produce debugging output. (default no debugging output) (boolean; default: false)
- **E:** The estimator can be one of the following: eb – Empirical Bayes estimator for noraml mixture (default) nested – Optimal nested model selector for normal mixture subset – Optimal subset selector for normal mixture pace2 – PACE2 for Chi-square mixture pace4 – PACE4 for Chi-square mixture pace6 – PACE6 for Chi-square mixture

  ols – Ordinary least squares estimator aic – AIC estimator bic – BIC estimator ric – RIC estimator olsc – Ordinary least squares subset selector with a threshold (string; default: 'eb')
- **S:** Threshold value for the OLSC estimator (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: binominal attributes, numerical attributes, numerical label, weighted examples

Short description: Class for building pace regression linear models and using them for prediction.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Wang, Y (2000). A new approach to fitting linear models in high dimensional spaces. Hamilton, New Zealand.

Wang, Y., Witten, I. H.: Modeling for optimal probability prediction. In: Proceedings of the Nineteenth International Conference in Machine Learning, Sydney, Australia, 650-657, 2002.

5.4.158 W-PredictiveApriori

Group: Learner.Unsupervised.Itemsets.Weka

Required input:

- ExampleSet

Generated output:

- WekaAssociator

Parameters:

- **N:** The required number of rules. (default = 2147483642) (real; $-\infty$-$+\infty$)
- **A:** If set class association rules are mined. (default = no) (boolean; default: false)
- **c:** The class index. (default = last) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Class implementing the predictive apriori algorithm to mine association rules.

Description: Performs the Weka association rule learner with the same name. The operator returns a result object containing the rules found by the association learner. In contrast to models generated by normal learners, the association rules cannot be applied to an example set. Hence, there is no way to evaluate the performance of association rules yet. See the Weka javadoc for further operator and parameter descriptions.

Further information: Tobias Scheffer: Finding Association Rules That Trade Support Optimally against Confidence. In: 5th European Conference on Principles of Data Mining and Knowledge Discovery, 424-435, 2001.

5.4.159 W-Prism

Group: Learner.Supervised.Weka.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: Class for building and using a PRISM rule set for classification.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: J. Cendrowska (1987). PRISM: An algorithm for inducing modular rules. International Journal of Man-Machine Studies. 27(4):349-370.

5.4.160 W-RBFNetwork

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Set the number of clusters (basis functions) to generate. (default = 2). (real; -∞-+∞)
- **S:** Set the random seed to be used by K-means. (default = 1). (real; -∞-+∞)
- **R:** Set the ridge value for the logistic or linear regression. (real; -∞-+∞)
- **M:** Set the maximum number of iterations for the logistic regression. (default -1, until convergence). (real; -∞-+∞)
- **W:** Set the minimum standard deviation for the clusters. (default 0.1). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Short description: Class that implements a normalized Gaussian radial basisbasis function network.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.161 W-REPTree

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **M:** Set minimum number of instances per leaf (default 2). (real; $-\infty$-$+\infty$)
- **V:** Set minimum numeric class variance proportion of train variance for split (default 1e-3). (real; $-\infty$-$+\infty$)
- **N:** Number of folds for reduced error pruning (default 3). (real; $-\infty$-$+\infty$)
- **S:** Seed for random data shuffling (default 1). (real; $-\infty$-$+\infty$)
- **P:** No pruning. (boolean; default: false)
- **L:** Maximum tree depth (default -1, no maximum) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: Fast decision tree learner.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.162 W-RacedIncrementalLogitBoost

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **C:** Minimum size of chunks. (default 500) (real; $-\infty$-$+\infty$)
- **M:** Maximum size of chunks. (default 2000) (real; $-\infty$-$+\infty$)
- **V:** Size of validation set. (default 1000) (real; $-\infty$-$+\infty$)

- ○ **P:** Committee pruning to perform. 0=none, 1=log likelihood (default) (real; $-\infty$-$+\infty$)
- ○ **Q:** Use resampling for boosting. (boolean; default: false)
- ○ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, updatable

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Classifier for incremental learning of large datasets by way of racing logit-boosted committees.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.163 W-RandomCommittee

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)

- **I:** Number of iterations. (default 10) (real; -∞-+∞)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for building an ensemble of randomizable base classifiers.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.164 W-RandomForest

Group: Learner.Supervised.Weka.Trees

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **I:** Number of trees to build. (real; -∞-+∞)
- **K:** Number of features to consider (¡1=int(logM+1)). (real; -∞-+∞)
- **S:** Seed for random number generator. (default 1) (real; -∞-+∞)

- **depth:** The maximum depth of the trees, 0 for unlimited. (default 0) (string)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for constructing a forest of random trees.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5-32.

5.4.165 W-RandomSubSpace

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **P:** Size of each subspace: ¡ 1: percentage of the number of attributes ¿=1: absolute number of attributes (real; -∞-+∞)
- **S:** Random number seed. (default 1) (real; -∞-+∞)
- **I:** Number of iterations. (default 10) (real; -∞-+∞)

- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: This method constructs a decision tree based classifier that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Tin Kam Ho (1998). The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence. 20(8):832-844. URL http://citeseer.ist.psu.edu/ho98random.html.

5.4.166 W-RandomTree

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **K:** Number of attributes to randomly investigate (¡1 = int(log(#attributes)+1)). (real; -∞-+∞)
- **M:** Set minimum number of instances per leaf. (real; -∞-+∞)
- **S:** Seed for random number generator. (default 1) (real; -∞-+∞)
- **depth:** The maximum depth of the tree, 0 for unlimited. (default 0) (string)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Class for constructing a tree that considers K randomly chosen attributes at each node.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.167 W-RegressionByDiscretization

Group: Learner.Supervised.Weka.Meta

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **B:** Number of bins for equal-width discretization (default 10). (real; -∞-+∞)

- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A regression scheme that employs any classifier on a copy of the data that has the class attribute (equal-width) discretized.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.168 W-Ridor

Group: Learner.Supervised.Weka.Rules

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **F:** Set number of folds for IREP One fold is used as pruning set. (default 3) (real; -∞-+∞)
- **S:** Set number of shuffles to randomize the data in order to get better rule. (default 10) (real; -∞-+∞)

- **A:** Set flag of whether use the error rate of all the data to select the default class in each step. If not set, the learner will only use the error rate in the pruning data (boolean; default: false)
- **M:** Set flag of whether use the majority class as the default class in each step instead of choosing default class based on the error rate (if the flag is not set) (boolean; default: false)
- **N:** Set the minimal weights of instances within a split. (default 2.0) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: The implementation of a RIpple-DOwn Rule learner.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Brian R. Gaines, Paul Compton (1995). Induction of Ripple-Down Rules Applied to Modeling Large Databases. J. Intell. Inf. Syst.. 5(3):211-228.

5.4.169 W-SMO

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **no-checks:** Turns off all checks - use with caution! Turning them off assumes that data is purely numeric, doesn't contain any missing values, and has a nominal class. Turning them off also means that no header information will be stored if the machine is linear. Finally, it also assumes that no instance has a weight equal to 0. (default: checks on) (boolean; default: false)
- **C:** The complexity constant C. (default 1) (real; -∞-+∞)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; -∞-+∞)
- **L:** The tolerance parameter. (default 1.0e-3) (real; -∞-+∞)
- **P:** The epsilon for round-off error. (default 1.0e-12) (real; -∞-+∞)
- **M:** Fit logistic models to SVM outputs. (boolean; default: false)
- **V:** The number of folds for the internal cross-validation. (default -1, use training data) (real; -∞-+∞)
- **W:** The random number seed. (default 1) (real; -∞-+∞)
- **K:** The Kernel to use. (default: weka.classifiers.functions.supportVector.PolyKernel) (string; default: 'weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: J. Platt: Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, 1998.

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy (2001). Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation. 13(3):637-649.

Trevor Hastie, Robert Tibshirani: Classification by Pairwise Coupling. In: Advances in Neural Information Processing Systems, 1998.

5.4.170 W-SMOreg

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **no-checks:** Turns off all checks - use with caution! Turning them off assumes that data is purely numeric, doesn't contain any missing values, and has a nominal class. Turning them off also means that no header information will be stored if the machine is linear. Finally, it also assumes that no instance has a weight equal to 0. (default: checks on) (boolean; default: false)
- **S:** The amount up to which deviations are tolerated (epsilon). (default 1e-3) (real; -∞-+∞)
- **C:** The complexity constant C. (default 1) (real; -∞-+∞)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; -∞-+∞)
- **T:** The tolerance parameter. (default 1.0e-3) (real; -∞-+∞)
- **P:** The epsilon for round-off error. (default 1.0e-12) (real; -∞-+∞)
- **K:** The Kernel to use. (default: weka.classifiers.functions.supportVector.PolyKernel) (string; default: 'weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label, weighted examples

Short description: Implements Alex Smola and Bernhard Scholkopf's sequential minimal optimization algorithm for training a support vector regression model.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Alex J. Smola, Bernhard Schoelkopf: A Tutorial on Support Vector Regression. In NeuroCOLT2 Technical Report Series, 1998.

S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy (1999). Improvements to SMO Algorithm for SVM Regression. Control Division Dept of Mechanical and Production Engineering, National University of Singapore.

5.4.171 W-SVMreg

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **C:** The complexity constant C. (default 1) (real; $-\infty$-$+\infty$)
- **N:** Whether to 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; $-\infty$-$+\infty$)

- **I:** Optimizer class used for solving quadratic optimization problem (default weka.classifiers.functions.supportVector.RegSMOImproved) (string; default: 'weka.classifiers.functions.supportVector.RegSMOImproved -L 0.0010 -W 1 -P 1.0E-12 -T 0.0010 -V')
- **K:** The Kernel to use. (default: weka.classifiers.functions.supportVector.PolyKernel) (string; default: 'weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, numerical label, weighted examples

Short description: SVMreg implements the support vector machine for regression.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy: Improvements to the SMO Algorithm for SVM Regression. In: IEEE Transactions on Neural Networks, 1999.

A.J. Smola, B. Schoelkopf (1998). A tutorial on support vector regression.

5.4.172 W-SerializedClassifier

Group: Learner.Supervised.Weka.Misc

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

○ **model:** The file containing the serialized model. (required) (string; default: '/home/ingo/workspace/yale')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: A wrapper around a serialized classifier model.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.173 W-SimpleCart

Group: Learner.Supervised.Weka.Trees

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)

○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

○ **M:** The minimal number of instances at the terminal nodes. (default 2) (real; $-\infty$-$+\infty$)

○ **N:** The number of folds used in the minimal cost-complexity pruning. (default 5) (real; $-\infty$-$+\infty$)

○ **U:** Don't use the minimal cost-complexity pruning. (default yes). (boolean; default: false)

○ **H:** Don't use the heuristic method for binary split. (default true). (boolean; default: false)

- ◦ **A:** Use 1 SE rule to make pruning decision. (default no). (boolean; default: false)
- ◦ **C:** Percentage of training data size (0-1]. (default 1). (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Class implementing minimal cost-complexity pruning.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone (1984). Classification and Regression Trees. Wadsworth International Group, Belmont, California.

5.4.174 W-SimpleKMeans

Group: Learner.Unsupervised.Clustering.Weka

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ◦ **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- ◦ **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- ◦ **N:** number of clusters. (default 2). (real; $-\infty$-$+\infty$)

- ○ **V:** Display std. deviations for centroids. (boolean; default: false)
- ○ **M:** Replace missing values with mean/mode. (boolean; default: false)
- ○ **S:** Random number seed. (default 10) (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: The weka clusterer W-SimpleKMeans

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

5.4.175 W-SimpleLinearRegression

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: numerical attributes, numerical label, weighted examples

Short description: Learns a simple linear regression model.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.176 W-SimpleLogistic

Group: Learner.Supervised.Weka.Functions

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **I:** Set fixed number of iterations for LogitBoost (real; $-\infty$-$+\infty$)
- ○ **S:** Use stopping criterion on training set (instead of cross-validation) (boolean; default: false)
- ○ **P:** Use error on probabilities (rmse) instead of misclassification error for stopping criterion (boolean; default: false)
- ○ **M:** Set maximum number of boosting iterations (real; $-\infty$-$+\infty$)
- ○ **H:** Set parameter for heuristic for early stopping of LogitBoost. If enabled, the minimum is selected greedily, stopping if the current minimum has not changed for iter iterations. By default, heuristic is enabled with value 50. Set to zero to disable heuristic. (real; $-\infty$-$+\infty$)
- ○ **W:** Set beta for weight trimming for LogitBoost. Set to 0 for no weight trimming. (real; $-\infty$-$+\infty$)
- ○ **A:** The AIC is used to choose the best iteration (instead of CV or training error). (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Classifier for building linear logistic regression models.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Niels Landwehr, Mark Hall, Eibe Frank (2005). Logistic Model Trees.

Marc Sumner, Eibe Frank, Mark Hall: Speeding up Logistic Model Tree Induction. In: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, 675-683, 2005.

5.4.177 W-SimpleMI

Group: Learner.Supervised.Weka.Mi

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **M:** The method used in transformation: 1.arithmatic average; 2.geometric centor; 3.using minimax combined features of a bag (default: 1)

  Method 3: Define s to be the vector of the coordinate-wise maxima and minima of X, ie., s(X)=(minx1, ..., minxm, maxx1, ...,maxxm), transform the exemplars into mono-instance which contains attributes s(X) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **W:** Full name of base classifier. (default: weka.classifiers.rules.ZeroR) (string; default: 'weka.classifiers.rules.ZeroR')

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label

Short description: Reduces MI data into mono-instance data.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.178 W-Stacking

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **M:** Full name of meta classifier, followed by options. (default: ”weka.classifiers.rules.Zero”) (string; default: 'weka.classifiers.rules.ZeroR ')
- **X:** Sets the number of cross-validation folds. (real; $-\infty$-$+\infty$)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Combines several classifiers using the stacking method.

Description: Performs the ensemble learning scheme of Weka with the same name. An arbitrary number of other Weka learning schemes must be embedded as inner operators. See the Weka javadoc for further classifier and parameter descriptions.

Further information: David H. Wolpert (1992). Stacked generalization. Neural Networks. 5:241-259.

5.4.179 W-StackingC

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **M:** Full name of meta classifier, followed by options. Must be a numeric prediction scheme. Default: Linear Regression. (string; default: 'weka.classifiers.functions.LinearRegression -S 1 -R 1.0E-8')
- **X:** Sets the number of cross-validation folds. (real; $-\infty$-$+\infty$)
- **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Implements StackingC (more efficient version of stacking).

Description: Performs the ensemble learning scheme of Weka with the same name. An arbitrary number of other Weka learning schemes must be embedded as inner operators. See the Weka javadoc for further classifier and parameter descriptions.

Further information: A.K. Seewald: How to Make Stacking Better and Faster While Also Taking Care of an Unknown Weakness. In: Nineteenth International Conference on Machine Learning, 554-561, 2002.

5.4.180 W-TLD

Group: Learner.Supervised.Weka.Mi

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **C:** Set whether or not use empirical log-odds cut-off instead of 0 (boolean; default: false)
- ◦ **R:** Set the number of multiple runs needed for searching the MLE. (real; -∞-+∞)
- ◦ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: Two-Level Distribution approach, changes the starting value of the searching algorithm, supplement the cut-off modification and check missing values.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Xin Xu (2003). Statistical learning in multiple instance problem. Hamilton, NZ.

5.4.181 W-TLDSimple

Group: Learner.Supervised.Weka.Mi

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **C:** Set whether or not use empirical log-odds cut-off instead of 0 (boolean; default: false)
- ○ **R:** Set the number of multiple runs needed for searching the MLE. (real; $-\infty$-$+\infty$)
- ○ **S:** Random number seed. (default 1) (real; $-\infty$-$+\infty$)
- ○ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label

Short description: The weka learner W-TLDSimple

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Xin Xu (2003). Statistical learning in multiple instance problem. Hamilton, NZ.

5.4.182 W-Tertius

Group: Learner.Unsupervised.Itemsets.Weka

Required input:

- ExampleSet

Generated output:

- WekaAssociator

Parameters:

- ◦ **K:** Set maximum number of confirmation values in the result. (default: 10) (real; $-\infty$-$+\infty$)
- ◦ **F:** Set frequency threshold for pruning. (default: 0) (real; $-\infty$-$+\infty$)
- ◦ **C:** Set confirmation threshold. (default: 0) (string)
- ◦ **N:** Set noise threshold : maximum frequency of counter-examples. 0 gives only satisfied rules. (default: 1) (real; $-\infty$-$+\infty$)
- ◦ **R:** Allow attributes to be repeated in a same rule. (boolean; default: false)
- ◦ **L:** Set maximum number of literals in a rule. (default: 4) (real; $-\infty$-$+\infty$)
- ◦ **G:** Set the negations in the rule. (default: 0) (real; $-\infty$-$+\infty$)
- ◦ **S:** Consider only classification rules. (boolean; default: false)
- ◦ **c:** Set index of class attribute. (default: last). (real; $-\infty$-$+\infty$)
- ◦ **H:** Consider only horn clauses. (boolean; default: false)

- **E:** Keep equivalent rules. (boolean; default: false)
- **M:** Keep same clauses. (boolean; default: false)
- **T:** Keep subsumed rules. (boolean; default: false)
- **I:** Set the way to handle missing values. (default: 0) (real; -∞-+∞)
- **O:** Use ROC analysis. (boolean; default: false)
- **p:** Set the file containing the parts of the individual for individual-based learning. (string; default: ”)
- **P:** Set output of current values. (default: 0) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Finds rules according to confirmation measure (Tertius-type algorithm).

Description: Performs the Weka association rule learner with the same name. The operator returns a result object containing the rules found by the association learner. In contrast to models generated by normal learners, the association rules cannot be applied to an example set. Hence, there is no way to evaluate the performance of association rules yet. See the Weka javadoc for further operator and parameter descriptions.

Further information: P. A. Flach, N. Lachiche (1999). Confirmation-Guided Discovery of first-order rules with Tertius. Machine Learning. 42:61-95.

5.4.183 W-ThresholdSelector

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **C:** The class for which threshold is determined. Valid values are: 1, 2 (for first and second classes, respectively), 3 (for whichever class is least frequent), and 4 (for whichever class value is most frequent), and 5 (for the first class named any of "yes","pos(itive)" "1", or method 3 if no matches). (default 5). (real; -∞-+∞)
- ◦ **X:** Number of folds used for cross validation. If just a hold-out set is used, this determines the size of the hold-out set (default 3). (real; -∞-+∞)
- ◦ **R:** Sets whether confidence range correction is applied. This can be used to ensure the confidences range from 0 to 1. Use 0 for no range correction, 1 for correction based on the min/max values seen during threshold selection (default 0). (real; -∞-+∞)
- ◦ **E:** Sets the evaluation mode. Use 0 for evaluation using cross-validation, 1 for evaluation using hold-out set, and 2 for evaluation on the training data (default 1). (real; -∞-+∞)
- ◦ **M:** Measure used for evaluation (default is FMEASURE). (string; default: 'FMEASURE')
- ◦ **manual:** Set a manual threshold to use. This option overrides automatic selection and options pertaining to automatic selection will be ignored. (default -1, i.e. do not use a manual threshold). (string)
- ◦ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: A metaclassifier that selecting a mid-point threshold on the probability output by a Classifier.

Description: Performs the meta learning scheme of Weka with the same name. Another non-meta learning scheme of Weka must be embedded as inner operator. See the Weka javadoc for further classifier and parameter descriptions.

5.4.184 W-VFI

Group: Learner.Supervised.Weka.Misc

Required input:	**Generated output:**
• ExampleSet	• Model

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **C:** Don't weight voting intervals by confidence (boolean; default: false)
- ○ **B:** Set exponential bias towards confident intervals (default = 1.0) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, weighted examples

Short description: Classification by voting feature intervals.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: G. Demiroz, A. Guvenir: Classification by voting feature intervals. In: 9th European Conference on Machine Learning, 85-92, 1997.

5.4.185 W-Vote

Group: Learner.Supervised.Weka.Meta

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **S:** Random number seed. (default 1) (real; -∞-+∞)
- ◦ **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- ◦ **R:** The combination rule to use (default: AVG) (string; default: 'AVG')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Class for combining classifiers.

Description: Performs the ensemble learning scheme of Weka with the same name. An arbitrary number of other Weka learning schemes must be embedded as inner operators. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Ludmila I. Kuncheva (2004). Combining Pattern Classifiers: Methods and Algorithms. John Wiley and Sons, Inc..

J. Kittler, M. Hatef, Robert P.W. Duin, J. Matas (1998). On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence. 20(3):226-239.

5.4.186 W-VotedPerceptron

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **I:** The number of iterations to be performed. (default 1) (real; -∞-+∞)
- **E:** The exponent for the polynomial kernel. (default 1) (real; -∞-+∞)
- **S:** The seed for the random number generation. (default 1) (real; -∞-+∞)
- **M:** The maximum number of alterations allowed. (default 10000) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, binominal label

Short description: Implementation of the voted perceptron algorithm by Freund and Schapire.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: Y. Freund, R. E. Schapire: Large margin classification using the perceptron algorithm. In: 11th Annual Conference on Computational Learning Theory, New York, NY, 209-217, 1998.

5.4.187 W-WAODE

Group: Learner.Supervised.Weka.Bayes

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)
- **I:** Whether to print some more internals. (default: no) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, polynominal label, binominal label

Short description: WAODE contructs the model called Weightily Averaged One-Dependence Estimators.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: L. Jiang, H. Zhang: Weightily Averaged One-Dependence Estimators. In: Proceedings of the 9th Biennial Pacific Rim International Conference on Artificial Intelligence, PRICAI 2006, 970-974, 2006.

5.4.188 W-Winnow

Group: Learner.Supervised.Weka.Functions

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **L:** Use the baLanced version (default false) (boolean; default: false)
- **I:** The number of iterations to be performed. (default 1) (real; -∞-+∞)
- **A:** Promotion coefficient alpha. (default 2.0) (real; -∞-+∞)
- **B:** Demotion coefficient beta. (default 0.5) (real; -∞-+∞)
- **H:** Prediction threshold. (default -1.0 == number of attributes) (real; -∞-+∞)
- **W:** Starting weights. (default 2.0) (real; -∞-+∞)
- **S:** Default random seed. (default 1) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, binominal label, updatable

Short description: Implements Winnow and Balanced Winnow algorithms by Littlestone.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

Further information: N. Littlestone (1988). Learning quickly when irrelevant attributes are abound: A new linear threshold algorithm. Machine Learning. 2:285-318.

N. Littlestone (1989). Mistake bounds and logarithmic linear-threshold learning algorithms. University of California, Santa Cruz.

5.4.189 W-XMeans

Group: Learner.Unsupervised.Clustering.Weka

Required input:

- ExampleSet

Generated output:

- ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **I:** maximum number of overall iterations (default 1). (real; -∞-+∞)
- **M:** maximum number of iterations in the kMeans loop in the Improve-Parameter part (default 1000). (real; -∞-+∞)
- **J:** maximum number of iterations in the kMeans loop for the splitted centroids in the Improve-Structure part (default 1000). (real; -∞-+∞)
- **L:** minimum number of clusters (default 2). (real; -∞-+∞)
- **H:** maximum number of clusters (default 4). (real; -∞-+∞)
- **B:** distance value for binary attributes (default 1.0). (real; -∞-+∞)
- **use-kdtree:** Uses the KDTree internally (default no). (boolean; default: false)
- **K:** Full class name of KDTree class to use, followed by scheme options. eg: "weka.core.neighboursearch.kdtrees.KDTree -P" (default no KDTree class used). (string)
- **C:** cutoff factor, takes the given percentage of the splitted centroids if none of the children win (default 0.0). (real; -∞-+∞)

- **D:** Full class name of Distance function class to use, followed by scheme options. (default weka.core.EuclideanDistance). (string; default: 'weka.core.EuclideanDistance -R first-last')
- **N:** file to read starting centers from (ARFF format). (string)
- **O:** file to write centers to (ARFF format). (string)
- **U:** The debug level. (default 0) (string)
- **Y:** The debug vectors file. (string)
- **S:** Random number seed. (default 10) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Cluster data using the X-means algorithm.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

Further information: Dan Pelleg, Andrew W. Moore: X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In: Seventeenth International Conference on Machine Learning, 727-734, 2000.

5.4.190 W-ZeroR

Group: Learner.Supervised.Weka.Rules

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **D:** If set, classifier is run in debug mode and may output additional info to the console (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Learner capabilities: polynominal attributes, binominal attributes, numerical attributes, polynominal label, binominal label, numerical label, weighted examples

Short description: Class for building and using a 0-R classifier.

Description: Performs the Weka learning scheme with the same name. See the Weka javadoc for further classifier and parameter descriptions.

5.4.191 W-sIB

Group: Learner.Unsupervised.Clustering.Weka

Required input:	**Generated output:**
• ExampleSet	• ClusterModel

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **add_cluster_attribute:** Indicates if a cluster id is generated as new special attribute. (boolean; default: true)
- **add_characterization:** Indicates if a characterization of each cluster is created by a simple classification learner. (boolean; default: false)
- **I:** maximum number of iterations (default 100). (real; $-\infty$-$+\infty$)
- **M:** minimum number of changes in a single iteration (default 0). (real; $-\infty$-$+\infty$)
- **N:** number of clusters. (default 2). (real; $-\infty$-$+\infty$)

- **R:** number of restarts. (default 5). (real; -∞-+∞)
- **U:** set not to normalize the data (default true). (boolean; default: false)
- **V:** set to output debug info (default false). (boolean; default: false)
- **S:** Random number seed. (default 1) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Cluster data using the sequential information bottleneck algorithm.

Description: This operator performs the Weka clustering scheme with the same name. The operator expects an example set containing ids and returns a FlatClusterModel or directly annotates the examples with a cluster attribute. Please note: Currently only clusterers that produce a partition of items are supported.

Further information: Noam Slonim, Nir Friedman, Naftali Tishby: Unsupervised document classification using sequential information maximization. In: Proceedings of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval, 129-136, 2002.

5.5 Meta optimization schemes

This group of operators iterate several times through an sub-process in order to optimize some parameters with respect to target functions like performance criteria.

5.5.1 AbsoluteSplitChain

Group: Meta.Control

Required input:

- ExampleSet

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **number_training_examples:** Absolute size of the training set. -1 equal to not defined (integer; -1-+∞; default: -1)
- **number_test_examples:** Absolute size of the test set. -1 equal to not defined (integer; -1-+∞; default: -1)
- **sampling_type:** Defines the sampling type of this operator.
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Operator 1 (First Part) must be able to handle [ExampleSet].
- Operator 2 (Second Part) must be able to handle [ExampleSet].

Short description: Splits an example set in two parts based on user defined set sizes and uses the output of the first child and the second part as input for the second child.

Description: An operator chain that split an ExampleSet into two disjunct parts and applies the first child operator on the first part and applies the second child on the second part and the result of the first child. The total result is the result of the second operator.

The input example set will be splitted based on a user defined absolute numbers.

5.5.2 AverageBuilder

Group: Meta.Other

Required input:

- AverageVector

Generated output:

- AverageVector

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Builds the average of input average vectors (e.g. performance) of the same type.

Description: Collects all average vectors (e.g. PerformanceVectors) from the input and average those of the same type.

5.5.3 BatchProcessing

Group: Meta

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **batch_size:** This number of examples is processed batch-wise by the inner operators of this operator. (integer; 1-+∞; default: 1000)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet].

Short description: Creates batches from the input examples and performs its inner operators on each of these batches which might be useful for applying methods on very large data sets directly in databases.

Description: This operator groups the input examples into batches of the specified size and performs the inner operators on all batches subsequently. This might be useful for very large data sets which cannot be load into memory but must be handled in a database. In these cases, preprocessing methods or model applications and other tasks can be performed on each batch and the result might be again written into a database table (by using the DatabaseExampleSetWriter in its append mode).

5.5.4 ClusterIteration

Group: Meta.Control

Required input:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: Applies all inner operators to all clusters.

Description: This operator splits up the input example set according to the clusters and applies its inner operators *number_of_clusters* time. This requires the example set to have a special cluster attribute which can be either created by a `Clusterer` or might be declared in the attribute description file that was used when the data was loaded.

5.5.5 EvolutionaryParameterOptimization

Group: Meta.Parameter

Generated output:

- ParameterSet
- PerformanceVector

Parameters:

- **configure_operator:** Configure this operator by means of a Wizard.
- **parameters:** The parameters. (list)
- **max_generations:** Stop after this many evaluations (integer; 1-+∞; default: 50)
- **generations_without_improval:** Stop after this number of generations without improvement (-1: optimize until max_iterations). (integer; -1-+∞; default: 1)
- **population_size:** The population size (-1: number of examples) (integer; -1-+∞; default: 5)
- **tournament_fraction:** The fraction of the population used for tournament selection. (real; 0.0-+∞)
- **keep_best:** Indicates if the best individual should survive (elititst selection). (boolean; default: true)
- **mutation_type:** The type of the mutation operator.
- **selection_type:** The type of the selection operator.
- **crossover_prob:** The probability for crossover. (real; 0.0-1.0)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **show_convergence_plot:** Indicates if a dialog with a convergence plot should be drawn. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **best:** best performance ever
- **looptime:** The time elapsed since the current loop started.
- **performance:** currently best performance
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: This operator finds the optimal values for parameters using an evolutionary computation approach.

Description: This operator finds the optimal values for a set of parameters using an evolutionary strategies approach which is often more appropriate than a grid search or a greedy search like the quadratic programming approach and leads to better results. The parameter *parameters* is a list of key value pairs where the keys are of the form `operator_name.parameter_name` and the value for each parameter must be a semicolon separated pair of a minimum and a maximum value in squared parantheses, e.g. [10;100] for a range of 10 until 100.

The operator returns an optimal ParameterSet which can as well be written to a file with a ParameterSetWriter (see section 5.3.35). This parameter set can be read in another process using a ParameterSetLoader (see section 5.3.34).

The file format of the parameter set file is straightforward and can easily be generated by external applications. Each line is of the form

```
operator_name.parameter_name = value
```

Please refer to section 4.3 for an example application.

5.5.6 ExampleSetIterator

Group: Meta.Control

Required input:

- ExampleSet

Parameters:

- ○ **only_best:** Return only best result? (Requires a PerformanceVector in the inner result). (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [].

Short description: Performs its inner operators for each example set found in input.

Description: For each example set the ExampleSetIterator finds in its input, the inner operators are applied as if it was an OperatorChain. This operator can be used to conduct a process consecutively on a number of different data sets.

5.5.7 ExperimentEmbedder

Group: Meta

Please use the operator 'ProcessEmbedder' instead (and note the change of the parameter name!)

Parameters:

- **process_file:** The process file which should be encapsulated by this operator (filename)
- ○ **use_input:** Indicates if the operator input should be used as input of the process (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator embeds a complete experiment previously written into a file.

Description: This operator can be used to embed a complete process definition into the current process definition. The process must have been written into a file before and will be loaded and executed when the current process reaches this operator. Optionally, the input of this operator can be used as input for the embedded process. In both cases, the output of the process will be delivered as output of this operator. Please note that validation checks will not work for process containing an operator of this type since the check cannot be performed without actually loading the process.

5.5.8 FeatureIterator

Group: Meta.Control

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **filter:** A regular expression which can be used to filter the features in this loop, i.e. the inner operators are only applied to features which name fulfills the filter expression. (string)
- **type_filter:** Indicates if a value type filter should be applied for this loop.
- **invert_selection:** Indicates if the filter settings should be inverted, i.e. the loop will run over all features not fulfilling the specified criteria. (boolean; default: false)
- **iteration_macro:** The name of the macro which holds the name of the current feature in each iteration. (string; default: 'loop_feature')

Values:

- **applycount:** The number of times the operator was applied.
- **feature_name:** The number of the current feature.
- **iteration:** The number of the current iteration / loop.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [].

Short description: Iterates over the given features and applies the inner operators for each feature where the inner operators can access the current feature name by a macro.

Description: This operator takes an input data set and applies its inner operators as often as the number of features of the input data is. Inner operators can access the current feature name by a macro, whose name can be specified via the parameter `iteration_macro`.

The user can specify with a parameter if this loop should iterate over all features or only over features with a specific value type, i.e. only over numerical or over nominal features. A regular expression can also be specified which is used as a filter, i.e. the inner operators are only applied for feature names fulfilling the filter expression.

5.5.9 FeatureSubsetIteration

Group: Meta.Control

Parameters:

- **min_number_of_attributes:** Determines the minimum number of features used for the combinations. (integer; 1-+∞; default: 1)
- **max_number_of_attributes:** Determines the maximum number of features used for the combinations (-1: try all combinations up to possible maximum) (integer; -1-+∞; default: -1)
- **exact_number_of_attributes:** Determines the exact number of features used for the combinations (-1: use the feature range defined by min and max). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **feature_names:** The names of the used features in the current iteration.
- **feature_number:** The number of used features in the current iteration.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [].

Short description: Performs its inner operator for all specified feature subsets (useful for brute force evaluations in combination with the ProcessLog operator).

Description: This meta operator iterates through all possible feature subsets within the specified range and applies the inner operators on the feature subsets. This might be useful in combination with the ProcessLog operator and, for example, a performance evaluation. In contrast to the BruteForce feature selection, which performs a similar task, this iterative approach needs much less memory and can be performed on larger data sets.

5.5.10 GridParameterOptimization

Group: Meta.Parameter

Generated output:

- ParameterSet
- PerformanceVector

Parameters:

- **configure_operator:** Configure this operator by means of a Wizard.
- **parameters:** The parameters. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **performance:** currently best performance
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: This operator finds the optimal values for parameters.

Description: This operator finds the optimal values for a set of parameters using a grid search. The parameter *parameters* is a list of key value pairs where the keys are of the form `operator_name.parameter_name` and the value is either a comma separated list of values (e.g. 10,15,20,25) or an interval definition in the format [start;end;stepsize] (e.g. [10;25;5]). Alternatively a value grid pattern may be used by [e.g. [start;end;no_steps;scale], where scale identifies the type of the pattern.

The operator returns an optimal ParameterSet which can as well be written to a file with a ParameterSetWriter (see section 5.3.35). This parameter set can be read in another process using a ParameterSetLoader (see section 5.3.34).

The file format of the parameter set file is straightforward and can easily be generated by external applications. Each line is of the form

```
operator_name.parameter_name = value
```

Please refer to section 4.3 for an example application. Another parameter optimization schems like the EvolutionaryParameterOptimization (see section 5.5.5) might also be useful if the best ranges and dependencies are not known at all. Another operator which works similar to this parameter optimization operator is the operator ParameterIteration (see section 5.5.17). In contrast to the optimization operator, this operator simply iterates through all parameter combinations. This might be especially useful for plotting purposes.

5.5.11 IteratingOperatorChain

Group: Meta.Control

Parameters:

- **iterations:** Number of iterations (integer; 0-+∞; default: 1)
- **timeout:** Timeout in minutes (-1: no timeout) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The iteration currently performed by this looping operator.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: Performs its inner operators k times.

Description: Performs its inner operators for the defined number of times. The input of this operator will be the input of the first operator in the first iteration. The output of each children operator is the input for the following one, the output of the last inner operator will be the input for the first child in the next iteration. The output of the last operator in the last iteration will be the output of this operator.

5.5.12 LearningCurve

Group: Meta.Other

Required input:

- ExampleSet

Parameters:

- **training_ratio:** The fraction of examples which shall be maximal used for training (dynamically growing), the rest is used for testing (fixed) (real; 0.0-1.0)
- **step_fraction:** The fraction of examples which would be additionally used in each step. (real; 0.0-1.0)
- **start_fraction:** Starts with this fraction of the training data and iteratively add step_fraction examples from the training data (-1: use step_fraction). (real; -1.0-1.0)
- **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- **local_random_seed:** The local random seed for random number generation (-1: use global random generator). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The variance of the last performance (main criterion).

- **fraction:** The used fraction of data.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: Iterates its inner operator for an increasing number of samples and collects the performances.

Description: This operator first divides the input example set into two parts, a training set and a test set according to the parameter "training_ratio". It then uses iteratively bigger subsets from the fixed training set for learning (the first operator) and calculates the corresponding performance values on the fixed test set (with the second operator).

5.5.13 MultipleLabelIterator

Group: Meta.Other

Required input:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: Performs its inner operators for each label found in input example set.

Description: Performs the inner operator for all label attributes, i.e. special attributes whose name starts with "label". In each iteration one of the multiple labels is used as label. The results of the inner operators are collected and returned. The example set will be consumed during the iteration.

5.5.14 OperatorEnabler

Group: Meta.Control

Parameters:

- **operator_name:** The name of the operator which should be disabled or enabled (inner operator names)
- ◦ **enable:** Indicates if the operator should be enabled (true) or disabled (false) (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: This operator can be used to automatically enable or disable inner operators.

Description: This operator can be used to enable and disable other operators. The operator which should be enabled or disabled must be a child operator of this one. Together with one of the parameter optimizing or iterating operators this operator can be used to dynamically change the process setup which might be useful in order to test different layouts, e.g. the gain by using different preprocessing steps.

5.5.15 OperatorSelector

Group: Meta.Control

Parameters:

- ○ **select_which:** Indicates which inner operator should be currently employed by this operator on the input objects. (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [] and must deliver [].

Short description: This operator can be used to select a single inner operator which should be performed, e.g. by means of parameter iteration or optimization operators.

Description: This operator can be used to employ a single inner operator or operator chain. Which operator should be used can be defined by the parameter "select_which". Together with one of the parameter optimizing or iterating operators this operator can be used to dynamically change the process setup which might be useful in order to test different layouts, e.g. the gain by using different preprocessing steps or chains or the quality of a certain learner.

5.5.16 ParameterCloner

Group: Meta.Parameter

Parameters:

- ○ **name_map:** A list mapping operator parameters from the set to other operator parameters in the process setup. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Applies a set of parameters of a source operator on a target operator.

Description: Sets a list of parameters using existing parameter values.

The operator is similar to PARAMETERSETTER (see section 5.5.18), but differs from that in not requiring a ParameterSet input. It simply reads a parameter value from a source and uses it to set the parameter value of a target parameter. Both, source and target, are given in the format 'operator'.'parameter'.

This operator is more general than ParameterSetter and could completely replace it. It is most useful, if you need a parameter which is optimized more than once within the optimization loop - ParameterSetter cannot be used here.

These parameters can either be generated by a PARAMETEROPTIMIZATIONOPERATOR or read by a PARAMETERSETLOADER (see section 5.3.34). This operator is useful, e.g. in the following scenario. If one wants to find the best parameters for a certain learning scheme, one usually is also interested in the model generated with this parameters. While the first is easily possible using a PARAMETEROPTIMIZATIONOPERATOR, the latter is not possible because the PARAMETEROPTIMIZATIONOPERATOR does not return the IOObjects produced within, but only a parameter set. This is, because the parameter optimization operator knows nothing about models, but only about the performance vectors produced within. Producing performance vectors does not necessarily require a model.

To solve this problem, one can use a `ParameterSetter`. Usually, a process definition with a `ParameterSetter` contains at least two operators of the same type, typically a learner. One learner may be an inner operator of the PARAMETEROPTIMIZATIONOPERATOR and may be named "Learner", whereas a second learner of the same type named "OptimalLearner" follows the parameter optimization and should use the optimal parameter set found by the optimization. In order to make the `ParameterSetter` set the optimal parameters of the right operator, one must specify its name. Therefore, the parameter list *name_map* was introduced. Each parameter in this list maps the name of an operator that was used during optimization (in our case this is "Learner") to an operator that should now use these parameters (in our case this is "OptimalLearner").

5.5.17 ParameterIteration

Group: Meta.Parameter

Parameters:

- **configure_operator:** Configure this operator by means of a Wizard.
- **parameters:** The parameters. (list)
- **synchronize:** Synchronize parameter iteration (boolean; default: false)
- **keep_output:** Delivers the merged output of the last operator of all the iterations, delivers the original input otherwise. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: This operator just iterates through all defined parameter combinations.

Description: In contrast to the GridParameterOptimization (see section 5.5.10) operator this operators simply uses the defined parameters and perform the inner operators for all possible combinations. This can be especially usefull for plotting or logging purposes and sometimes also for simply configuring the parameters for the inner operators as a sort of meta step (e.g. learning curve generation).

This operator iterates through a set of parameters by using all possible parameter combinations. The parameter *parameters* is a list of key value pairs where the keys are of the form `operator_name.parameter_name` and the value is either a comma separated list of values (e.g. 10,15,20,25) or an interval definition in the format [start;end;stepsize] (e.g. [10;25;5]). Additionally, the format [start;end;steps;scale] is allowed.

Please note that this operator has two modes: synchronized and non-synchronized. In the latter, all parameter combinations are generated and the inner operators are applied for each combination. In the synchronized mode, no combinations are generated but the set of all pairs of the increasing number of parameters are used. For the iteration over a single parameter there is no difference between both modes. Please note that the number of parameter possibilities must be the same for all parameters in the synchronized mode.

5.5.18 ParameterSetter

Group: Meta.Parameter

Required input:

- ParameterSet

Parameters:

- ○ **name_map:** A list mapping operator names from the set to operator names in the process setup. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Applies a set of parameters. Operator names may be remapped.

Description: Sets a set of parameters. These parameters can either be generated by a ParameterOptimizationOperator or read by a ParameterSetLoader (see section 5.3.34). This operator is useful, e.g. in the following scenario. If one wants to find the best parameters for a certain learning scheme, one usually is also interested in the model generated with this parameters. While the first is easily possible using a ParameterOptimizationOperator, the latter is not possible because the ParameterOptimizationOperator does not return the IOObjects produced within, but only a parameter set. This is, because the parameter optimization operator knows nothing about models, but only about the performance vectors produced within. Producing performance vectors does not necessarily require a model.

To solve this problem, one can use a `ParameterSetter`. Usually, a process with a `ParameterSetter` contains at least two operators of the same type, typically a learner. One learner may be an inner operator of the ParameterOptimizationOperator and may be named "Learner", whereas a second learner of the same type named "OptimalLearner" follows the parameter optimization and should use the optimal parameter set found by the optimization. In order to make the `ParameterSetter` set the optimal parameters of the right operator, one must specify its name. Therefore, the parameter list *name_map* was introduced. Each parameter in this list maps the name of an operator that was used during optimization (in our case this is "Learner") to an operator that should now use these parameters (in our case this is "OptimalLearner").

5.5.19 PartialExampleSetLearner

Group: Meta.Other

Required input:

- ExampleSet

Generated output:

- Model

Parameters:

- **fraction:** The fraction of examples which shall be used. (real; 0.0-1.0)
- **sampling_type:** Defines the sampling type (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [Model].

Short description: Uses only a fraction of the data to apply the inner operator on it.

Description: This operator works similar to the LearningCurve (see section 5.5.12). In contrast to this, it just splits the ExampleSet according to the parameter ”fraction” and learns a model only on the subset. It can be used, for example, in conjunction with GridParameterOptimization (see section 5.5.10) which sets the fraction parameter to values between 0 and 1. The advantage is, that this operator can then be used inside of a XValidation (see section 5.9.32), which delivers more stable result estimations.

5.5.20 ProcessBranch

Group: Meta.Control

Parameters:

- **condition_type:** The condition which is used for the condition check.
- ○ **condition_value:** A condition parameter which might be desired for some condition checks. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: This operator provides a conditional execution of operators.

Description: This operator provides a conditional execution of parts of processes. It has to have two OperatorChains as childs. The first chain is processed if the specified condition is true, the second one is processed if it is false (if-then-else). The second chain may be omitted (if-then). In this case, this operator has only one inner operator.

If the condition “attribute_value_filter” is used, the same attribute value conditions already known from the ExampleFilter (see section 5.8.32) operator can be used. In addition to the known attribute value relation format (e.g. “att1¿=0.7”), this operator expects an additional definition for the used example which cam be added in “[“ and “]“ after the attribute value condition. The following values are possible:

- a fixed number, e.g. “att1¿0.7 [7]“ meaning that the value for attribute “att1” for the example 7 must be greater than 0.7
- the wildcard “*“ meaning that the attribute value condition must be fulfilled for all examples, e.g. “att4¡=5 [*]“
- no example definition, meaning the same as the wildcard definition [*]

5.5.21 ProcessEmbedder

Group: Meta

Parameters:

- **process_file:** The process file which should be encapsulated by this operator (filename)
- ○ **use_input:** Indicates if the operator input should be used as input of the process (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator embeds a complete process previously written into a file.

Description: This operator can be used to embed a complete process definition into the current process definition. The process must have been written into a file before and will be loaded and executed when the current process reaches this operator. Optionally, the input of this operator can be used as input for the embedded process. In both cases, the output of the process will be delivered as output of this operator. Please note that validation checks will not work for process containing an operator of this type since the check cannot be performed without actually loading the process.

5.5.22 QuadraticParameterOptimization

Group: Meta.Parameter

Generated output:

- ParameterSet
- PerformanceVector

Parameters:

- ○ **configure_operator:** Configure this operator by means of a Wizard.
- ○ **parameters:** The parameters. (list)
- ○ **if_exceeds_region:** What to do if range is exceeded.
- ○ **if_exceeds_range:** What to do if range is exceeded.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **performance:** currently best performance
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: This operator finds the optimal values for parameters using a quadratic interaction model.

Description: This operator finds the optimal values for a set of parameters using a quadratic interaction model. The parameter *parameters* is a list of key value pairs where the keys are of the form `OperatorName.parameter_name` and the value is a comma separated list of values (as for the GridParameterOptimization operator).

The operator returns an optimal ParameterSet which can as well be written to a file with a ParameterSetLoader (see section 5.3.34). This parameter set can be read in another process using an ParameterSetLoader (see section 5.3.34).

The file format of the parameter set file is straightforward and can also easily be generated by external applications. Each line is of the form

```
operator_name.parameter_name = value
```

.

5.5.23 RandomOptimizer

Group: Meta.Other

Parameters:

- **iterations:** The number of iterations to perform (integer; 1-$+\infty$)
- ○ **timeout:** Timeout in minutes (-1 = no timeout) (integer; 1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **avg_performance:** The average performance
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The current best performance
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: Performs its inner operators k times and returns the best results.

Description: This operator iterates several times through the inner operators and in each cycle evaluates a performance measure. The IOObjects that are produced as output of the inner operators in the best cycle are then returned. The target of this operator are methods that involve some non-deterministic elements such that the performance in each cycle may vary. An example is k-means with random intialization.

5.5.24 RepeatUntilOperatorChain

Group: Meta.Control

Parameters:

- ○ **min_attributes:** Minimal number of attributes in first example set (integer; 0-$+\infty$; default: 0)

- **max_attributes:** Maximal number of attributes in first example set (integer; 0-+∞; default: 0)
- **min_examples:** Minimal number of examples in first example set (integer; 0-+∞; default: 0)
- **max_examples:** Maximal number of examples in first example set (integer; 0-+∞; default: +∞)
- **min_criterion:** Minimal main criterion in first performance vector (real; -∞-+∞)
- **max_criterion:** Maximal main criterion in first performance vector (real; -∞-+∞)
- **max_iterations:** Maximum number of iterations (integer; 0-+∞; default: +∞)
- **timeout:** Timeout in minutes (-1 = no timeout) (integer; 1-+∞; default: -1)
- **performance_change:** Stop when performance of inner chain behaves like this.
- **condition_before:** Evaluate condition before inner chain is applied (true) or after? (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: All inner operators must be able to handle the output of their predecessor.

Short description: Performs its inner operators until some condition is met.

Description: Performs its inner operators until all given criteria are met or a timeout occurs.

5.5.25 SeriesPrediction

Group: Meta.Other

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ○ **window_width:** The number of values used as indicators for predicting the target value. (integer; 1-+∞; default: 10)
- ○ **horizon:** The gap size used between training windows and prediction value. (integer; 1-+∞; default: 1)
- ○ **max_training_set_size:** The maximum number of examples (windows) used for training the prediction model. (integer; 1-+∞; default: 10)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Creates predictions for an ordered time series given as the label attribute of an example set.

Description: This operator can be used for some basic series prediction operations. The given series must be univariate and must be encoded by examples, i.e. each point of time is encoded by the values in one single example. The values which should be predicted must be defined by the label attribute. Other attributes will be ignored.

The operator creates time windows and learns a model from these windows to predict the value of the label column after a certain amount of values (horizon). After predicting a value, the window is moved with step size 1 and the next value is predicted. All predictions are kept and can be compared afterwards to the actual values in a series plot or with a performance evaluation operator.

If you want predictions for different horizons, you have to restart this operator with different settings for horizon. This might be useful to get a prediction for 1 to horizon future time steps.

The inner learner must be able to work on numerical regression problems.

5.5.26 SplitChain

Group: Meta.Control

Required input:

- ExampleSet

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **split_ratio:** Relative size of the training set. (real; 0.0-1.0)
- ◦ **sampling_type:** Defines the sampling type of this operator.
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Operator 1 (First Part) must be able to handle [ExampleSet].
- Operator 2 (Second Part) must be able to handle [ExampleSet].

Short description: Splits an example set in two parts based on a user defined ratio and uses the output of the first child and the second part as input for the second child.

Description: An operator chain that split an ExampleSet into two disjunct parts and applies the first child operator on the first part and applies the second child on the second part and the result of the first child. The total result is the result of the second operator.

The input example set will be splitted based on a defined ratio between 0 and 1.

5.5.27 ValueSubgroupIterator

Group: Meta.Control

Required input:

- ExampleSet

Parameters:

- ○ **attributes:** The attributes. (list)
- ○ **p:** Threshold of value occurance. (real; 0.0-1.0)
- ○ **filter_attribute:** Filter subgroup defining attribute. (boolean; default: true)
- ○ **apply_on_complete_set:** Apply inner operators also on complete set. (boolean; default: false)
- ○ **iteration_macro:** Name of macro which is set in each iteration. (string; default: 'loop_value')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [].

Short description: Iterates over the values of the specified attributes and applies the inner operators on the subgroups which exhibit the current attribute value.

Description: In each iteration step, this meta operator applies its inner operators to a subset of the input example set. The subsets represent subgroups which are defined by the values of the specified attributes. If an attribute is specified which has 'male' or 'female' as possible values the first iteration subset will consist of all males, the second of all females, respectively. Please note that no attribute value combinations are supported and hence only subgroups defined by exactly one attribute are considered at a time.

A subset is build (and an inner operator application is executed) for each possible attribute value of the specified attributes if `all` is selected for the `values` parameter. If `above p` is selected, a subset is only build for that values which exhibit an occurance ratio of at least p. This may be helpful, if only large subgroups should be considered.

The parameter `filter_attribute` specifies, if the subgroup defining attribute should be filtered from the subsets.

The parameter `apply_on_complete_set` specifies, if the inner operators should be applied on the completed example set in addition to the subset iterations.

5.5.28 XVPrediction

Group: Meta.Control

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **number_of_validations:** Number of subsets for the crossvalidation. (integer; 2-+∞; default: 10)
- **leave_one_out:** Set the number of validations to the number of examples. If set to true, number_of_validations is ignored. (boolean; default: false)
- **sampling_type:** Defines the sampling type of the cross validation.
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [ExampleSet].

Short description: Predicts the examples in a cross-validation-like fashion.

Description: Operator chain that splits an ExampleSet into a training and test sets similar to XValidation, but returns the test set predictions instead of a performance vector. The inner two operators must be a learner returning a Model and an operator or operator chain that can apply this model (usually a model applier)

5.6 OLAP operators

OLAP (Online Analytical Processing) is an approach to quickly providing answers to analytical queries that are multidimensional in nature. Usually, the basics of OLAP is a set of SQL queries which will typically result in a matrix (or pivot) format. The dimensions form the row and column of the matrix. RapidMiner supports basic OLAP functionality like grouping and aggregations.

5.6.1 ANOVAMatrix

Group: OLAP

Required input:

- ExampleSet

Generated output:

- ANOVAMatrix

Parameters:

- ○ **significance_level:** The significance level for the ANOVA calculation. (real; 0.0-1.0)
- ○ **only_distinct:** Indicates if only rows with distinct values for the aggregation attribute should be used for the calculation of the aggregation function. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs an ANOVA significance test for a all numerical attribute based on the groups defined by all other nominal attributes.

Description: This operator calculates the significance of difference for the values for all numerical attributes depending on the groups defined by all nominal attributes. Please refer to the operator GroupedANOVA (see section 5.6.3) for details of the calculation.

5.6.2 Aggregation

Group: OLAP

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: true)
- ○ **aggregation_attributes:** The attributes which should be aggregated. (list)
- ○ **group_by_attributes:** Performs a grouping by the values of the attributes whose names match the given regular expression. (string)
- ○ **only_distinct:** Indicates if only rows with distinct values for the aggregation attribute should be used for the calculation of the aggregation function. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs one of the aggregation functions (count, sum...) known from SQL (allows also grouping).

Description: This operator creates a new example set from the input example set showing the results of arbitrary aggregation functions (as SUM, COUNT etc. known from SQL). Before the values of different rows are aggregated into a new row the rows might be grouped by the values of a multiple attributes (similar to the group-by clause known from SQL). In this case a new line will be created for each group.

Please note that the known HAVING clause from SQL can be simulated by an additional ExampleFilter (see section 5.8.32) operator following this one.

5.6.3 GroupedANOVA

Group: OLAP

Required input:

- ExampleSet

Generated output:

- SignificanceTestResult

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **anova_attribute:** Calculate the ANOVA for this attribute based on the groups defines by group_by_attribute. (string)
- **group_by_attribute:** Performs a grouping by the values of the attribute with this name. (string)
- ◦ **significance_level:** The significance level for the ANOVA calculation. (real; 0.0-1.0)
- ◦ **only_distinct:** Indicates if only rows with distinct values for the aggregation attribute should be used for the calculation of the aggregation function. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs an ANOVA significance test for a single numerical attribute based on the groups defined by another (nominal) attribute.

Description: This operator creates groups of the input example set based on the defined grouping attribute. For each of the groups the mean and variance of another attribute (the anova attribute) is calculated and an ANalysis Of VAriance (ANOVA) is performed. The result will be a significance test result for the specified significance level indicating if the values for the attribute are significantly different between the groups defined by the grouping attribute.

5.7 Postprocessing

Postprocessing operators can usually be applied on models in order to perform some postprocessing steps like cost-sensitive threshold selection or scaling schemes like Platt scaling.

5.7.1 FrequentItemSetUnificator

Group: Postprocessing

Required input:

- FrequentItemSets
- ExampleSet

Generated output:

- FrequentItemSets

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Compares sets of frequent item sets and removes common not unique sets.

Description: This operator compares a number of FrequentItemSet sets and removes every not unique FrequentItemSet.

5.7.2 PlattScaling

Group: Postprocessing

Required input:

- ExampleSet
- Model

Generated output:

- Model

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Turns confidence scores of boolean classifiers into probability estimates.

Description: A scaling operator, applying the original algorithm by Platt (1999) to turn confidence scores of boolean classifiers into probability estimates.

Unlike the original version this operator assumes that the confidence scores are already in the interval of [0,1], as e.g. given for the RapidMiner boosting operators. The crude estimates are then transformed into log odds, and scaled by the original transformation of Platt.

The operator assumes a model and an example set for scaling. It outputs a PlattScalingModel, that contains both, the supplied model and the scaling step. If the example set contains a weight attribute, then this operator is able to fit a model to the weighted examples.

5.7.3 ThresholdApplier

Group: Postprocessing

Required input:

- ExampleSet
- Threshold

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Applies a threshold on soft classified data.

Description: This operator applies the given threshold to an example set and maps a soft prediction to crisp values. If the confidence for the second class (usually positive for RapidMiner) is greater than the given threshold the prediction is set to this class.

5.7.4 ThresholdCreator

Group: Postprocessing

Generated output:

- Threshold

Parameters:

- ○ **threshold:** The confidence threshold to determine if the prediction should be positive. (real; 0.0-1.0)
- **first_class:** The class which should be considered as the first one (confidence 0). (string)
- **second_class:** The class which should be considered as the second one (confidence 1). (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a user defined threshold for given prediction confidences (soft predictions) in order to turn it into a crisp classifier.

Description: This operator creates a user defined threshold for crisp classifying based on prediction confidences.

5.7.5 ThresholdFinder

Group: Postprocessing

Required input:

- ExampleSet

Generated output:

- ExampleSet
- Threshold

Parameters:

- **misclassification_costs_first:** The costs assigned when an example of the first class is classified as one of the second. (real; 0.0-$+\infty$)
- **misclassification_costs_second:** The costs assigned when an example of the second class is classified as one of the first. (real; 0.0-$+\infty$)
- **show_roc_plot:** Display a plot of the ROC curve. (boolean; default: false)
- **use_example_weights:** Indicates if example weights should be used. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Finds a threshold for given prediction confidences (soft predictions) , costs and distributional information in order to turn it into a crisp classification. The optimization step is based on ROC analysis.

Description: This operator finds the best threshold for crisp classifying based on user defined costs.

5.7.6 UncertainPredictionsTransformation

Group: Postprocessing

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **min_confidence:** The minimal confidence necessary for not setting the prediction to 'unknown'. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Sets all predictions to 'unknown' (missing value) if the corresponding confidence is smaller than the specified value.

Description: This operator sets all predictions which do not have a higher confidence than the specified one to "unknown" (missing value). This operator is a quite simple version of the CostBasedThresholdLearner which might be useful in simple binominal classification settings (although it does also work for polynominal classifications).

5.8 Data preprocessing

Preprocessing operators can be used to generate new features by applying functions on the existing features or by automatically cleaning up the data replacing missing values by, for instance, average values of this attribute.

5.8.1 AGA

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)
- **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-$+\infty$; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-$+\infty$)
- **population_size:** Number of individuals per generation. (integer; 1-$+\infty$; default: 5)

- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **selection_scheme:** The selection scheme of this EA.
- **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)
- **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- **p_initialize:** Initial probability for an attribute to be switched on. (real; 0.0-1.0)
- **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- **crossover_type:** Type of the crossover.
- **use_plus:** Generate sums. (boolean; default: true)
- **use_diff:** Generate differences. (boolean; default: false)
- **use_mult:** Generate products. (boolean; default: true)
- **use_div:** Generate quotients. (boolean; default: false)
- **reciprocal_value:** Generate reciprocal values. (boolean; default: true)
- **max_number_of_new_attributes:** Max number of attributes to generate for an individual in one generation. (integer; 0-+∞; default: 1)
- **max_total_number_of_attributes:** Max total number of attributes in all generations (-1: no maximum). (integer; -1-+∞; default: -1)
- **p_generate:** Probability for an individual to be selected for generation. (real; 0.0-1.0)
- **p_mutation:** Probability for an attribute to be changed (-1: 1 / numberOfAtts). (real; -1.0-1.0)
- **use_square_roots:** Generate square root values. (boolean; default: false)
- **use_power_functions:** Generate the power of one attribute and another. (boolean; default: false)

- ◦ **use_sin:** Generate sinus. (boolean; default: false)
- ◦ **use_cos:** Generate cosinus. (boolean; default: false)
- ◦ **use_tan:** Generate tangens. (boolean; default: false)
- ◦ **use_atan:** Generate arc tangens. (boolean; default: false)
- ◦ **use_exp:** Generate exponential functions. (boolean; default: false)
- ◦ **use_log:** Generate logarithmic functions. (boolean; default: false)
- ◦ **use_absolute_values:** Generate absolute values. (boolean; default: false)
- ◦ **use_min:** Generate minimum values. (boolean; default: false)
- ◦ **use_max:** Generate maximum values. (boolean; default: false)
- ◦ **use_sgn:** Generate signum values. (boolean; default: false)
- ◦ **use_floor_ceil_functions:** Generate floor, ceil, and rounded values. (boolean; default: false)
- ◦ **restrictive_selection:** Use restrictive generator selection (faster). (boolean; default: true)
- ◦ **remove_useless:** Remove useless attributes. (boolean; default: true)
- ◦ **remove_equivalent:** Remove equivalent attributes. (boolean; default: true)
- ◦ **equivalence_samples:** Check this number of samples to prove equivalency. (integer; 1-+∞; default: 5)
- ◦ **equivalence_epsilon:** Consider two attributes equivalent if their difference is not bigger than epsilon. (real; 0.0-+∞)
- ◦ **equivalence_use_statistics:** Recalculates attribute statistics before equivalence check. (boolean; default: true)
- ◦ **search_fourier_peaks:** Use this number of highest frequency peaks for sinus generation. (integer; 0-+∞; default: 0)
- ◦ **attributes_per_peak:** Use this number of additional peaks for each found peak. (integer; 1-+∞; default: 1)
- ◦ **epsilon:** Use this range for additional peaks for each found peak. (real; 0.0-+∞)
- ◦ **adaption_type:** Use this adaption type for additional peaks.

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.

- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Another (improved) genetic algorithm for feature selection and feature generation (AGA).

Description: Basically the same operator as the GeneratingGeneticAlgorithm (see section 5.8.53) operator. This version adds additional generators and improves the simple GGA approach by providing some basic intron prevention techniques. In general, this operator seems to work better than the original approach but frequently deliver inferior results compared to the operator YAGGA2 (see section 5.8.126).

5.8.2 AbsoluteSampling

Group: Preprocessing.Data.Sampling

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ◦ **sample_size:** The number of examples which should be sampled (integer; 1-$+\infty$; default: 100)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a sample from an example set by drawing an exact number of examples.

Description: Absolute sampling operator. This operator takes a random sample with the given size. For example, if the sample size is set to 50, the result will have exactly 50 examples randomly drawn from the complete data set. Please note that this operator does not sample during a data scan but jumps to the rows. It should therefore only be used in case of memory data management and not, for example, for database or file management.

5.8.3 AbsoluteStratifiedSampling

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **sample_size:** The number of examples which should be sampled (integer; 1-+∞; default: 100)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a stratified sample from an example set by drawing a given number of examples.

Description: Stratified sampling operator. This operator performs a random sampling of a given size. In contrast to the simple sampling operator, this operator performs a stratified sampling for data sets with nominal label attributes, i.e. the class distributions remains (almost) the same after sampling. Hence, this operator cannot be applied on data sets without a label or with a numerical label. In these cases a simple sampling without stratification is performed. In some cases it might happen that not the exact desired number of examples is sampled, e.g. if the desired number is 100 from three qually distributed classes the resulting number will be 99 (33 of each class).

5.8.4 AddNominalValue

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attribute_name:** The name of the nominal attribute to which values should be added. (string)
- **new_value:** The value which should be added. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator adds an additional value to a specified nominal attribute which is from then mapped to a specific index.

Description: Adds a value to a nominal attribute definition.

5.8.5 AttributeCopy

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attribute_name:** The name of the nominal attribute to which values should be added. (string)

○ **new_name:** The name of the new (copied) attribute. If this parameter is missing, simply the same name with an appended number is used. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Copies a single attribute (only the view on the data column, not the data itself).

Description: Adds a copy of a single attribute to the given example set.

5.8.6 AttributeFilter

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **condition_class:** Implementation of the condition.

○ **parameter_string:** Parameter string for the condition, e.g. 'attribute=value' for the ValueFilter. (string)

○ **invert_filter:** Indicates if only attributes should be accepted which would normally filtered. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator removes attributes which fulfill a specified condition.

Description: This operator filters the attributes of an exampleSet. Therefore, different conditions may be selected as parameter and only attributes fulfilling this condition are kept. The rest will be removed from the exampleSet The conditions may be inverted. The conditions are tested over all attributes and for every attribute over all examples. For example the numeric_value_filter with the parameter string "¿ 6" will keep all nominal attributes and all numeric attributes having a value of greater 6 in every example. A combination of conditions is possible: "¿ 6 ANDAND ¡ 11" or "¡= 5 || ¡ 0". But ANDAND and || must not be mixed. Please note that ANDAND has to be replaced by two ampers ands.

The attribute_name_filter keeps all attributes which names match the given regular expression. The nominal_value_filter keeps all numeric attribute and all nominal attributes containing at least one of specified nominal values. "rainy ANDAND cloudy" would keep all attributes containing at least one time "rainy" and one time "cloudy". "rainy || sunny" would keep all attributes containing at least one time "rainy" or one time "sunny". ANDAND and || are not allowed to be mixed. And again, ANDAND has to be replaced by two ampers ands.

5.8.7 AttributeMerge

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **first_attribute:** The first attribute of this merger. (string)
- **second_attribute:** The second attribute of this merger. (string)
- ○ **separator:** Indicated a string which is used as separation of both values. (string; default: '_')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Merges two attributes into a single new attribute by concatenating the values.

Description: This operator merges two attributes by simply concatenating the values and store those new values in a new attribute which will be nominal. If the resulting values are actually numerical, you could simply change the value type afterwards with the corresponding operators.

5.8.8 AttributeSubsetPreprocessing

Group: Preprocessing.Attributes

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

◦ **condition_class:** Implementation of the condition.

◦ **parameter_string:** Parameter string for the condition, e.g. 'attribute=value' for the nominal value filter. (string)

- **attribute_name_regex:** A regular expression which matches against all attribute names (including special attributes). (string)

◦ **invert_selection:** Indicates if the specified attribute selection should be inverted. (boolean; default: false)

◦ **process_special_attributes:** Indicates if special attributes like labels etc. should also be processed. (boolean; default: false)

◦ **keep_subset_only:** Indicates if the attributes which did not match the regular expression should be removed by this operator. (boolean; default: false)

◦ **deliver_inner_results:** Indicates if the additional results (other than example set) of the inner operator should also be returned. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [ExampleSet].

Short description: Selects one attribute (or a subset) via a regular expression and applies its inner operators to the resulting subset.

Description: This operator can be used to select one attribute (or a subset) by defining a regular expression for the attribute name and applies its inner operators to the resulting subset. Please note that this operator will also use special attributes which makes it necessary for all preprocessing steps which should be performed on special attributes (and are normally not performed on special attributes).

This operator is also able to deliver the additional results of the inner operator if desired.

Afterwards, the remaining original attributes are added to the resulting example set if the parameter "keep_subset_only" is set to false (default).

Please note that this operator is very powerful and can be used to create new preprocessing schemes by combinating it with other preprocessing operators. Hoewever, there are two major restrictions (among some others): first, since the inner result will be combined with the rest of the input example set, the number of examples (data points) is not allowed to be changed inside of the subset preprocessing. Second, attribute role changes will not be delivered to the outside since internally all special attributes will be changed to regular for the inner operators and role changes can afterwards not be delivered.

5.8.9 AttributeValueMapper

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attributes:** Mapping of values will be applied to the attributes that match the given regular expression. (string)
- ○ **apply_to_special_features:** Filter also special attributes (label, id...) (boolean; default: false)
- **replace_what:** All occurrences of this value will be replaced. (string)
- **replace_by:** The new attribute value to use. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps certain values of an attribute to other values.

Description: This operator takes an `ExampleSet` as input and maps the values of certain attributes to other values. For example, it can replace all occurrences of the String "unknown" in a nominal Attribute by a default String, for all examples in the ExampleSet.

This operator can replace nominal values (e.g. replace the value "green" by the value "green_color") as well as numerical values (e.g. replace the all values "3" by "-1").

This operator supports regular expressions for the attribute names, i.e. the value mapping is applied on all attributes for which the name fulfills the pattern defined by the name expression.

5.8.10 AttributeValueSubstring

Group: Preprocessing.Attributes.Filter

Required input:	Generated output:
• ExampleSet	• ExampleSet

Parameters:

- **attributes:** Substring creation of values will be applied to the attributes that match the given regular expression. (string)

∘ **apply_to_special_features:** Filter also special attributes (label, id...) (boolean; default: false)

∘ **first:** The index of the first character of the substring which should be kept (counting starts with 1, 0: start with beginning of value). (integer; 1-+∞; default: 1)

∘ **last:** The index of the last character of the substring which should be kept (counting starts with 1, 0: end with end of value). (integer; 1-+∞; default: +∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates new attributes from nominal attributes which only contain substrings of the original attributes.

Description: This operator creates new attributes from nominal attributes where the new attributes contain only of substrings of the original values. Please note that the counting starts with 1 and that the first and the last character will be included in the resulting substring. For example, the value is "RapidMiner" and the first index is set to 6 and the last index is set to 9 the result will be "Mine". If the last index is larger than the length of the word, the resulting substrings will end with the last character.

5.8.11 AttributeWeightSelection

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet
- AttributeWeights

Generated output:

- ExampleSet

Parameters:

- ◦ **keep_attribute_weights:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **weight:** Use this weight for the selection relation. (real; -∞-+∞)
- ◦ **weight_relation:** Selects only weights which fulfill this relation.
- ◦ **k:** Number k of attributes to be selected for weight-relations 'top k' or 'bottom k'. (integer; 1-+∞; default: 10)
- ◦ **p:** Percentage of attributes to be selected for weight-relations 'top p
- ◦ **deselect_unknown:** Indicates if attributes which weight is unknown should be deselected. (boolean; default: true)
- ◦ **use_absolute_weights:** Indicates if the absolute values of the weights should be used for comparison. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Selects only attributes which weights fulfill a given relation with respect to the input attribute weights.

Description: This operator selects all attributes which have a weight fulfilling a given condition. For example, only attributes with a weight greater than `min_weight` should be selected. This operator is also able to select the k attributes with the highest weight.

5.8.12 AttributeWeightsApplier

Group: Preprocessing.Attributes

Required input:

- ExampleSet
- AttributeWeights

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Deselects attributes with weight 0 and calculates new values for numeric attributes.

Description: This operator deselects attributes with a weight value of 0.0. The values of the other numeric attributes will be recalculated based on the weights delivered as AttributeWeights object in the input.

This operator can hardly be used to select a subset of features according to weights determined by a former weighting scheme. For this purpose the operator AttributeWeightSelection (see section 5.8.11) should be used which will select only those attribute fulfilling a specified weight relation.

5.8.13 Attributes2RealValues

Group: Preprocessing.Attributes.Filter

Please use the operator Nominal2Numerical instead

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- ○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all values to real values.

Description: This operator maps all non numeric attributes to real valued attributes. Nothing is done for numeric attributes, binary attributes are mapped to 0 and 1.

For nominal attributes one of the following calculations will be done:

- Dichotomization, i.e. one new attribute for each value of the nominal attribute. The new attribute which corresponds to the actual nominal value gets value 1 and all other attributes gets value 0.
- Alternatively the values of nominal attributes can be seen as equally ranked, therefore the nominal attribute will simply be turned into a real valued attribute, the old values results in equidistant real values.

At this moment the same applies for ordinal attributes, in a future release more appropriate values based on the ranking between the ordinal values may be included.

5.8.14 BackwardWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **keep_best:** Keep the best n individuals in each generation. (integer; 1-+∞; default: 1)
- **generations_without_improval:** Stop after n generations without improval of the performance. (integer; 1-+∞; default: 1)
- **weights:** Use these weights for the creation of individuals in each generation. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Assumes that features are independent and optimizes the weights of the attributes with a linear search.

Description: Uses the backward selection idea for the weighting of features.

5.8.15 BinDiscretization

Group: Preprocessing.Data.Discretization

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)

○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)

○ **number_of_bins:** Defines the number of bins which should be used for each attribute. (integer; 2-$+\infty$; default: 2)

○ **use_long_range_names:** Indicates if long range names including the limits should be used. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Discretize numerical attributes into a user defined number of bins.

Description: This operator discretizes all numeric attributes in the dataset into nominal attributes. This discretization is performed by simple binning, i.e. the specified number of equally sized bins is created and the numerical values are simply sorted into those bins. Skips all special attributes including the label.

5.8.16 Bootstrapping

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ◦ **sample_ratio:** This ratio determines the size of the new example set. (real; 0.0-$+\infty$)
- ◦ **local_random_seed:** Local random seed for this operator (-1: use global random seed). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a bootstrapped sample by sampling with replacement.

Description: This operator constructs a bootstrapped sample from the given example set. That means that a sampling with replacement will be performed. The usual sample size is the number of original examples. This operator also offers the possibility to create the inverse example set, i.e. an example set containing all examples which are not part of the bootstrapped example set. This inverse example set might be used for a bootstrapped validation (together with an IteratingPerformanceAverage (see section 5.9.19) operator.

5.8.17 BruteForce

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ◦ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ◦ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ◦ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ◦ **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- ◦ **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- ◦ **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- ◦ **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- ◦ **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)

- **min_number_of_attributes:** Determines the minimum number of features used for the combinations. (integer; 1-+∞; default: 1)
- **max_number_of_attributes:** Determines the maximum number of features used for the combinations (-1: try all combinations up to possible maximum) (integer; -1-+∞; default: -1)
- **exact_number_of_attributes:** Determines the exact number of features used for the combinations (-1: use the feature range defined by min and max). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Selects the best features for an example set by trying all possible combinations of attribute selections.

Description: This feature selection operator selects the best attribute set by trying all possible combinations of attribute selections. It returns the example set containing the subset of attributes which produced the best performance. As this operator works on the powerset of the attributes set it has exponential runtime.

5.8.18 ChangeAttributeName

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **old_name:** The old name of the attribute. (string)
- **new_name:** The new name of the attribute. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to rename an attribute.

Description: This operator can be used to rename an attribute of the input example set. If you want to change the attribute type (e.g. from regular to id attribute or from label to regular etc.), you should use the ChangeAttributeType (see section 5.8.20) operator.

5.8.19 ChangeAttributeRole

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **name:** The name of the attribute of which the type should be changed. (string)
- ○ **target_role:** The target role of the attribute (only changed if parameter change_attribute_type is true).

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to change the attribute type (regular, special, label, id...).

Description: This operator can be used to change the attribute type of an attribute of the input example set. If you want to change the attribute name you should use the ChangeAttributeName (see section 5.8.18) operator.

The target type indicates if the attribute is a regular attribute (used by learning operators) or a special attribute (e.g. a label or id attribute). The following target attribute types are possible:

- regular: only regular attributes are used as input variables for learning tasks
- id: the id attribute for the example set
- label: target attribute for learning
- prediction: predicted attribute, i.e. the predictions of a learning scheme
- cluster: indicates the memebership to a cluster
- weight: indicates the weight of the example
- batch: indicates the membership to an example batch

Users can also define own attribute types by simply using the desired name.

5.8.20 ChangeAttributeType

Group: Preprocessing.Attributes.Filter

Please use the operator ChangeAttributeRole instead

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **name:** The name of the attribute of which the type should be changed. (string)

○ **target_type:** The target type of the attribute (only changed if parameter change_attribute_type is true).

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to change the attribute type (regular, special, label, id...).

Description: This operator can be used to change the attribute type of an attribute of the input example set. If you want to change the attribute name you should use the ChangeAttributeName (see section 5.8.18) operator.

The target type indicates if the attribute is a regular attribute (used by learning operators) or a special attribute (e.g. a label or id attribute). The following target attribute types are possible:

- regular: only regular attributes are used as input variables for learning tasks
- id: the id attribute for the example set
- label: target attribute for learning
- prediction: predicted attribute, i.e. the predictions of a learning scheme
- cluster: indicates the memebership to a cluster
- weight: indicates the weight of the example
- batch: indicates the membership to an example batch

Users can also define own attribute types by simply using the desired name.

5.8.21 ChiSquaredWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **number_of_bins:** The number of bins used for discretization of numerical attributes before the chi squared test can be performed. (integer; 2-+∞; default: 10)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator calculates the relevance of a feature by computing for each attribute of the input example set the value of the chi-squared statistic with respect to the class attribute.

Description: This operator calculates the relevance of a feature by computing for each attribute of the input example set the value of the chi-squared statistic with respect to the class attribute.

5.8.22 CompleteFeatureGeneration

Group: Preprocessing.Attributes.Generation

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ○ **keep_all:** If set to true, all the original attributes are kept, otherwise they are removed from the example set. (boolean; default: true)
- ○ **use_plus:** Generate sums. (boolean; default: false)
- ○ **use_diff:** Generate differences. (boolean; default: false)
- ○ **use_mult:** Generate products. (boolean; default: false)
- ○ **use_div:** Generate quotients. (boolean; default: false)
- ○ **use_reciprocals:** Generate reciprocal values. (boolean; default: false)
- ○ **use_square_roots:** Generate square root values. (boolean; default: false)

- **use_power_functions:** Generate the power of one attribute and another. (boolean; default: false)
- **use_sin:** Generate sinus. (boolean; default: false)
- **use_cos:** Generate cosinus. (boolean; default: false)
- **use_tan:** Generate tangens. (boolean; default: false)
- **use_atan:** Generate arc tangens. (boolean; default: false)
- **use_exp:** Generate exponential functions. (boolean; default: false)
- **use_log:** Generate logarithmic functions. (boolean; default: false)
- **use_absolute_values:** Generate absolute values. (boolean; default: false)
- **use_min:** Generate minimum values. (boolean; default: false)
- **use_max:** Generate maximum values. (boolean; default: false)
- **use_ceil:** Generate ceil values. (boolean; default: false)
- **use_floor:** Generate floor values. (boolean; default: false)
- **use_rounded:** Generate rounded values. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: The feature generation operator generates new features via applying a set of functions on all features.

Description: This operator applies a set of functions on all features of the input example set. Applicable functions include +, -, *, /, norm, sin, cos, tan, atan, exp, log, min, max, floor, ceil, round, sqrt, abs, and pow. Features with two arguments will be applied on all pairs. Non commutative functions will also be applied on all permutations.

5.8.23 ComponentWeights

Group: Preprocessing.Attributes.Weighting

Required input:

- Model

Generated output:

- Model
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: false)
- ○ **component_number:** Create the weights of this component. (integer; 1-$+\infty$; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates the AttributeWeights of models containing components like PCA, GHA or FastICA.

Description: For models creating components like `PCA`, `GHA` and `FastICA` you can create the `AttributeWeights` from a component.

5.8.24 CorpusBasedWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **class_to_characterize:** The target class for which to find characteristic feature weights. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator uses a corpus of examples to characterize a single class by setting feature weights.

Description: This operator uses a corpus of examples to characterize a single class by setting feature weights. Characteristic features receive higher weights than less characteristic features. The weight for a feature is determined by calculating the average value of this feature for all examples of the target class. This operator assumes that the feature values characterize the importance of this feature for an example (e.g. TFIDF or others). Therefore, this operator is mainly used on textual data based on TFIDF weighting schemes. To extract such feature values from text collections you can use the Word Vector Tool plugin.

5.8.25 Date2Nominal

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **attribute_name:** The attribute which should be parsed. (string)
- **date_format:** The output format of the date values, for example "yyyy/MM/dd". (string)

○ **locale:** The used locale for date texts, for example "Wed" (English) in contrast to "Mi" (German).

○ **keep_old_attribute:** Indicates if the original date attribute should be kept. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Parses the date values for the specified date attribute with respect to the given date format string and transforms the values into nominal values.

Description: This operator transforms the specified date attribute and writes a new nominal attribute in a user specified format. This might be useful for time base OLAP to change the granularity of the time stamps from day to week or month.

The date format can be specified by the date_format parameter like described in the following.

Date and Time Patterns Date and time formats are specified by *date and time pattern* strings in the date_format parameter. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

- *G*: era designator; Text; example: AD
- *y*: year; Year; example: 1996; 96
- *M*: month in year; Month; example: July; Jul; 07
- *w*: week in year; Number; example: 27
- *W*: week in month; Number; example: 2
- *D*: day in year; Number; example: 189
- *d*: day in month; Number; example: 10
- *F*: day of week in month; Number; example: 2
- *E*: day in week; Text; example: Tuesday; Tue
- *a*: am/pm marker; Text; example: PM
- *H*: hour in day (0-23); Number; example: 0
- *k*: hour in day (1-24); Number; example: 24
- *K*: hour in am / pm (0-11); Number; example: 0
- *h*: hour in am / pm (1-12); Number; example: 12
- *m*: minute in hour; Number; example: 30

- *s*: second in minute; Number; example: 55
- *S*: millisecond; Number; example: 978
- *z*: time zone; General Time Zone; example: Pacific Standard Time; PST; GMT-08:00
- *Z*: time zone; RFC 822 Time Zone; example: -0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text:* For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
- *Number:* For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
- *Year:* If the underlying calendar is the Gregorian calendar, the following rules are applied.
  - For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a *number*.
  - For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.
  - For parsing with the abbreviated year pattern ("y" or "yy"), this operator must interpret the abbreviated year relative to some century. It does this by adjusting dates to be within 80 years before and 20 years after the time the operator is created. For example, using a pattern of "MM/dd/yy" and the operator created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, "-1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

Otherwise, calendar system specific forms are applied. If the number of pattern letters is 4 or more, a calendar specific long form is used. Otherwise, a calendar short or abbreviated form is used.

- *Month:* If the number of pattern letters is 3 or more, the month is interpreted as *text*; otherwise, it is interpreted as a *number*.
- *General time zone:* Time zones are interpreted as *text* if they have names. It is possible to define time zones by representing a GMT offset value. RFC 822 time zones are also accepted.
- *RFC 822 time zone:* For formatting, the RFC 822 4-digit time zone format is used. General time zones are also accepted.

This operator also supports *localized date and time pattern* strings by defining the locale parameter. In these strings, the pattern letters described above may be replaced with other, locale dependent, pattern letters.

Examples The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

- *“yyyy.MM.dd G ’at’ HH:mm:ss z”*: 2001.07.04 AD at 12:08:56 PDT
- *“EEE, MMM d, ”yy”*: Wed, Jul 4, ’01
- *“h:mm a”*: 12:08 PM
- *“hh ’o”clock’ a, zzzz”*: 12 o’clock PM, Pacific Daylight Time
- *“K:mm a, z”*: 0:08 PM, PDT
- *“yyyyy.MMMMM.dd GGG hh:mm aaa”*: 02001.July.04 AD 12:08 PM
- *“EEE, d MMM yyyy HH:mm:ss Z”*: Wed, 4 Jul 2001 12:08:56 -0700
- *“yyMMddHHmmssZ”*: 010704120856-0700
- *“yyyy-MM-dd’T’HH:mm:ss.SSSZ”*: 2001-07-04T12:08:56.235-0700

5.8.26 DeObfuscator

Group: Preprocessing.Other

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **obfuscation_map_file:** File where the obfuscator map was written to. (filename)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Replaces all obfuscated values and attribute names by the ones given in a file.

Description: This operator takes an `ExampleSet` as input and maps all nominal values to randomly created strings. The names and the construction descriptions of all attributes will also replaced by random strings. This operator can be used to anonymize your data. It is possible to save the obfuscating map into a file which can be used to remap the old values and names. Please use the operator `Deobfuscator` for this purpose. The new example set can be written with an `ExampleSetWriter`.

5.8.27 DensityBasedOutlierDetection

Group: Preprocessing.Data.Outlier

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **distance:** The distance for objects. (real; 0.0-$+\infty$)
- **proportion:** The proportion of objects related to D. (real; 0.0-1.0)

◦ **distance_function:** Indicates which distance function will be used for calculating the distance between two objects

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Identifies outliers in the given ExampleSet based on the data density.

Description: This operator is a DB outlier detection algorithm which calculates the DB(p,D)-outliers for an ExampleSet passed to the operator. DB(p,D)-outliers are Distance based outliers according to Knorr and Ng. A DB(p,D)-outlier is an object to which at least a proportion of p of all objects are farer away than distance D. It implements a global homogenous outlier search.

Currently, the operator supports cosine, sine or squared distances in addition to the usual euclidian distance which can be specified by the corresponding parameter. The operator takes two other real-valued parameters p and D. Depending on these parameters, search objects will be created from the examples in the ExampleSet passed to the operator. These search objects will be added to a search space which will perform the outlier search according to the DB(p,D) scheme.

The Outlier status (boolean in its nature) is written to a new special attribute "Outlier" and is passed on with the example set.

5.8.28 DistanceBasedOutlierDetection

Group: Preprocessing.Data.Outlier

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **number_of_neighbors:** Specifies the k value for the k-th nearest neighbours to be the analyzed.(default value is 10, minimum 1 and max is set to 1 million) (integer; 1-+∞; default: 10)

○ **number_of_outliers:** The number of top-n Outliers to be looked for.(default value is 10, minimum 2 (internal reasons) and max is set to 1 million) (integer; 1-+∞; default: 10)

- **distance_function:** choose which distance function will be used for calculating the distance between two objects

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Identifies n outliers in the given ExampleSet based on the distance to their k nearest neighbors.

Description: This operator performs a Dk_n Outlier Search according to the outlier detection approach recommended by Ramaswamy, Rastogi and Shim in ”Efficient Algorithms for Mining Outliers from Large Data Sets”. It is primarily a statistical outlier search based on a distance measure similar to the DB(p,D)-Outlier Search from Knorr and Ng. But it utilizes a distance search through the k-th nearest neighbourhood, so it implements some sort of locality as well.

The method states, that those objects with the largest distance to their k-th nearest neighbours are likely to be outliers respective to the data set, because it can be assumed, that those objects have a more sparse neighbourhood than the average objects. As this effectively provides a simple ranking over all the objects in the data set according to the distance to their k-th nearest neighbours, the user can specify a number of n objects to be the top-n outliers in the data set.

The operator supports cosine, sine or squared distances in addition to the euclidian distance which can be specified by a distance parameter. The Operator takes an example set and passes it on with an boolean top-n Dk outlier status in a new boolean-valued special outlier attribute indicating true (outlier) and false (no outlier).

5.8.29 EqualLabelWeighting

Group: Preprocessing.Data.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **total_weight:** The total weight distributed over all examples. (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Distributes weight over examples, so that per label weights sum up equally.

Description: This operator distributes example weights so that all example weights of labels sum up equally.

5.8.30 EvolutionaryFeatureAggregation

Group: Preprocessing.Attributes.Aggregation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- PerformanceVector

Parameters:

- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **aggregation_function:** The aggregation function which is used for feature aggregations.
- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 10)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 100)
- **selection_type:** The type of selection.
- **tournament_fraction:** The fraction of the population which will participate in each tournament. (real; 0.0-1.0)
- **crossover_type:** The type of crossover.

- ○ **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: A generating genetic algorithm for unsupervised learning (experimental).

Description: Performs an evolutionary feature aggregation. Each base feature is only allowed to be used as base feature, in one merged feature, or it may not be used at all.

5.8.31 EvolutionaryWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)

- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **selection_scheme:** The selection scheme of this EA.
- **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)
- **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- **mutation_variance:** The (initial) variance for each mutation. (real; 0.0-+∞)
- **1_5_rule:** If set to true, the 1/5 rule for variance adaption is used. (boolean; default: true)
- **bounded_mutation:** If set to true, the weights are bounded between 0 and 1. (boolean; default: false)

- **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- **crossover_type:** Type of the crossover.

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Weight the features with an evolutionary approach.

Description: This operator performs the weighting of features with an evolutionary strategies approach. The variance of the gaussian additive mutation can be adapted by a 1/5-rule.

5.8.32 ExampleFilter

Group: Preprocessing.Data.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **condition_class:** Implementation of the condition.
- **parameter_string:** Parameter string for the condition, e.g. 'attribute=value' for the AttributeValueFilter. (string)

- ∘ **invert_filter:** Indicates if only examples should be accepted which would normally filtered. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator only allows examples which fulfill a specified condition.

Description: This operator takes an ExampleSet as input and returns a new ExampleSet including only the `Examples` that fulfill a condition.

By specifying an implementation of `Condition` and a parameter string, arbitrary filters can be applied. Users can implement their own conditions by writing a subclass of the above class and implementing a two argument constructor taking an ExampleSet and a parameter string. This parameter string is specified by the parameter `parameter_string`. Instead of using one of the predefined conditions users can define their own implementation with the fully qualified class name.

For "attribute_value_condition" the parameter string must have the form `attribute op value`, where attribute is a name of an attribute, value is a value the attribute can take and op is one of the binary logical operators similar to the ones known from Java, e.g. greater than or equals.

For "unknown_attributes" the parameter string must be empty. This filter removes all examples containing attributes that have missing or illegal values. For "unknown_label" the parameter string must also be empty. This filter removes all examples with an unknown label value.

5.8.33 ExampleRangeFilter

Group: Preprocessing.Data.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **first_example:** The first example of the resulting example set. (integer; 1-+∞)
- **last_example:** The last example of the resulting example set. (integer; 1-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This only allows examples in the specified index range.

Description: This operator keeps only the examples of a given range (including the borders). The other examples will be removed from the input example set.

5.8.34 ExampleSet2AttributeWeights

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator simply creates new attribute weights of 1 for each input attribute.

Description: This operator creates a new attribute weights IOObject from a given example set. The result is a vector of attribute weights containing the weight 1.0 for each of the input attributes.

5.8.35 ExampleSetCartesian

Group: Preprocessing.Other

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **remove_double_attributes:** Indicates if double attributes should be removed or renamed (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Build the cartesian product of two example sets. In contrast to the ExampleSetJoin operator Id attributes are not needes.

Description: Build the cartesian product of two example sets. In contrast to the ExampleSetJoin (see section 5.8.36) operator, this operator does not depend on Id attributes. The result example set will consist of the union set or the union list (depending on parameter setting double attributes will be removed or renamed) of both feature sets. In case of removing double attribute the attribute values must be the same for the examples of both example set, otherwise an exception will be thrown.

Please note that this check for double attributes will only be applied for regular attributes. Special attributes of the second input example set which do not exist in the first example set will simply be added. If they already exist they are simply skipped.

5.8.36 ExampleSetJoin

Group: Preprocessing.Other

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **remove_double_attributes:** Indicates if double attributes should be removed or renamed (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Build the join of two example sets using the id attributes of the sets in order to identify the same examples.

Description: Build the join of two example sets using the id attributes of the sets, i.e. both example sets must have an id attribute where the same id indicate the same examples. If examples are missing an exception will be thrown. The result example set will consist of the same number of examples but the union set or the union list (depending on parameter setting double attributes will be removed or renamed) of both feature sets. In case of removing double attribute the attribute values must be the same for the examples of both example set, otherwise an exception will be thrown.

Please note that this check for double attributes will only be applied for regular attributes. Special attributes of the second input example set which do not exist in the first example set will simply be added. If they already exist they are simply skipped.

5.8.37 ExampleSetMerge

Group: Preprocessing.Other

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **datamanagement:** Determines, how the data is represented internally.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Build a merged example set from two or more compatible example sets by adding all examples into a combined set.

Description: This operator merges two or more given example sets by adding all examples in one example table containing all data rows. Please note that the new example table is built in memory and this operator might therefore not be applicable for merging huge data set tables from a database. In that case other preprocessing tools should be used which aggregates, joins, and merges tables into one table which is then used by RapidMiner.

All input example sets must provide the same attribute signature. That means that all examples sets must have the same number of (special) attributes and attribute names. If this is true this operator simply merges all example sets by adding all examples of all table into a new set which is then returned.

5.8.38 ExampleSetTranspose

Group: Preprocessing.Other

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Transposes the input example set similar to the matrix operator transpose.

Description: This operator transposes an example set, i.e. the columns with become the new rows and the old rows will become the columns. Hence, this operator works very similar to the well know transpose operation for matrices.

If an Id attribute is part of the given example set, the ids will become the names of the new attributes. The names of the old attributes will be transformed into the id values of a new special Id attribute. Since no other "special" examples or data rows exist, all other new attributes will be regular after the transformation. You can use the ChangeAttributeType (see section 5.8.20) operator in order to change one of these into a special type afterwards.

If all old attribute have the same value type, all new attributes will have this value type. Otherwise, the new value types will all be "nominal" if at least one nominal attribute was part of the given example set and "real" if the types contained mixed numbers.

This operator produces a copy of the data in the main memory and it therefore not suggested to use it on very large data sets.

5.8.39 ExchangeAttributeRoles

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **first_attribute:** The name of the first attribute for the attribute role exchange. (string)
- **second_attribute:** The name of the first attribute for the attribute role exchange. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator can be used to exchange the attribute roles of two attributes (e.g. a label with a regular attribute).

Description: This operator changes the attribute roles of two input attributes. This could for example be useful to exchange the roles of a label with a regular attribute (and vice versa), or a label with a batch attribute, a label with a cluster etc.

5.8.40 FastICA

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- Model

Parameters:

- **number_of_components:** Number components to be extracted (-1 number of attributes is used). (integer; -1-+∞; default: -1)
- **algorithm_type:** If 'parallel' the components are extracted simultaneously, 'deflation' the components are extracted one at a time
- **function:** The functional form of the G function used in the approximation to neg-entropy
- **alpha:** constant in range [1, 2] used in approximation to neg-entropy when fun="logcosh" (real; 1.0-2.0)
- **row_norm:** Indicates whether rows of the data matrix should be standardized beforehand. (boolean; default: false)
- **max_iteration:** maximum number of iterations to perform (integer; 0-+∞; default: 200)
- **tolerance:** A positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged. (real; 0.0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs an independent component analysis (ICA).

Description: This operator performs the independent componente analysis (ICA). Implementation of the FastICA-algorithm of Hyvaerinen und Oja. The operator outputs a `FastICAModel`. With the `ModelApplier` you can transform the features.

5.8.41 FeatureBlockTypeFilter

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **filter_special_features:** Filter also special attributes (label, id...) (boolean; default: false)
- ○ **skip_features_of_type:** All features of this type will be deselected off.
- ○ **except_features_of_type:** All features of this type will not be deselected.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator switches off those features whose block type matches the given one.

Description: This operator switches off all features whose block type matches the one given in the parameter `skip_features_of_type`. This can be useful e.g. for preprocessing operators that can handle only series attributes.

5.8.42 FeatureGeneration

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **filename:** Create the attributes listed in this file (written by an AttributeConstructionsWriter). (filename)

- ◦ **functions:** List of functions to generate. (list)
- ◦ **keep_all:** If set to true, all the original attributes are kept, otherwise they are removed from the example set. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: The feature generation operator generates new user defined features.

Description: This operator generates new user specified features. The new features are specified by their function names (prefix notation) and their arguments using the names of existing features.

Legal function names include +, -, etc. and the functions norm, sin, cos, tan, atan, exp, log, min, max, floor, ceil, round, sqrt, abs, and pow. Constant values can be defined by "const[value]()" where value is the desired value. Do not forget the empty round brackets. Example: `+(a1, -(a2, a3))` will calculate the sum of the attribute `a1` and the difference of the attributes `a2` and `a3`.

Features are generated in the following order:

1. Features specified by the file referenced by the parameter "filename" are generated
2. Features specified by the parameter list "functions" are generated
3. If "keep_all" is false, all of the old attributes are removed now

The list of supported functions include +, -, etc. and the functions sin, cos, tan, atan, exp, log, min, max, floor, ceil, round, sqrt, abs, sgn, pow.

5.8.43 FeatureNameFilter

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **filter_special_features:** Filter also special attributes (label, id...) (boolean; default: false)
- **skip_features_with_name:** Remove attributes with a matching name (accepts regular expressions) (string)
- ○ **except_features_with_name:** Does not remove attributes if their name fulfills this matching criterion (accepts regular expressions) (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator switches off those features whose name matches the given one (regular expressions are also allowed).

Description: This operator switches off all features whose name matches the one given in the parameter `skip_features_with_name`. The name can be defined as a regular expression.

5.8.44 FeatureRangeRemoval

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **first_attribute:** The first attribute of the attribute range which should be removed (integer; 1-+∞)
- **last_attribute:** The last attribute of the attribute range which should be removed (integer; 1-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator removes a range of features.

Description: This operator removes the attributes of a given range. The first and last attribute of the range will be removed, too. Counting starts with 1.

5.8.45 FeatureSelection

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ○ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ○ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ○ **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- ○ **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- ○ **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- ○ **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- ○ **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- ○ **selection_direction:** Forward selection or backward elimination.

- **keep_best:** Keep the best n individuals in each generation. (integer; 1-$+\infty$; default: 1)
- **generations_without_improval:** Stop after n generations without improval of the performance (-1: stops if the maximum_number_of_generations is reached). (integer; -1-$+\infty$; default: 1)
- **maximum_number_of_generations:** Delivers the maximum amount of generations (-1: might use or deselect all features). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: This operator realizes feature selection by forward selection and backward elimination, respectively.

Description: This operator realizes the two deterministic greedy feature selection algorithms forward selection and backward elimination. However, we added some enhancements to the standard algorithms which are described below:

Forward Selection

1. Create an initial population with n individuals where n is the input example set's number of attributes. Each individual will use exactly one of the features.

2. Evaluate the attribute sets and select only the best k.

3. For each of the k attribute sets do: If there are j unused attributes, make j copies of the attribute set and add exactly one of the previously unused attributes to the attribute set.

4. As long as the performance improved in the last p iterations go to 2

Backward Elimination

1. Start with an attribute set which uses all features.

2. Evaluate all attribute sets and select the best k.

3. For each of the k attribute sets do: If there are j attributes used, make j copies of the attribute set and remove exactly one of the previously used attributes from the attribute set.

4. As long as the performance improved in the last p iterations go to 2

The parameter k can be specified by the parameter `keep_best`, the parameter p can be specified by the parameter `generations_without_improval`. These parameters have default values 1 which means that the standard selection algorithms are used. Using other values increase the runtime but might help to avoid local extrema in the search for the global optimum.

Another unusual parameter is `maximum_number_of_generations`. This parameter bounds the number of iterations to this maximum of feature selections / deselections. In combination with `generations_without_improval` this allows several different selection schemes (which are described for forward selection, backward elimination works analogous):

- `maximum_number_of_generations` $= m$ and `generations_without_improval` $= p$: Selects maximal m features. The selection stops if not performance improvement was measured in the last p generations.

- `maximum_number_of_generations` $= -1$ and `generations_without_improval` $= p$: Tries to selects new features until no performance improvement was measured in the last p generations.

- `maximum_number_of_generations` $= m$ and `generations_without_improval` $= -1$: Selects maximal m features. The selection stops is not stopped until all combinations with maximal m were tried. However, the result might contain less features than these.

- `maximum_number_of_generations` $= -1$ and `generations_without_improval` $= -1$: Test all combinations of attributes (brute force, this might take a very long time and should only be applied to small attribute sets).

5.8.46 FeatureValueTypeFilter

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **filter_special_features:** Filter also special attributes (label, id...) (boolean; default: false)
- ○ **skip_features_of_type:** All features of this type will be deselected.
- ○ **except_features_of_type:** All features of this type will not be deselected.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator switches off those features whose value type matches the given one.

Description: This operator switches off all features whose value type matches the one given in the parameter `skip_features_of_type`. This can be useful e.g. for learning schemes that can handle only nominal attributes.

5.8.47 ForwardWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ○ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ○ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ○ **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- ○ **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- ○ **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- ○ **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- ○ **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- ○ **keep_best:** Keep the best n individuals in each generation. (integer; 1-+∞; default: 1)
- ○ **generations_without_improval:** Stop after n generations without improval of the performance. (integer; 1-+∞; default: 1)
- ○ **weights:** Use these weights for the creation of individuals in each generation. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Assumes that features are independent and optimizes the weights of the attributes with a linear search.

Description: This operator performs the weighting under the naive assumption that the features are independent from each other. Each attribute is weighted with a linear search. This approach may deliver good results after short time if the features indeed are not highly correlated.

5.8.48 FourierTransform

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Uses the label as function of each attribute and calculates the fourier transformations as new attributes.

Description: Creates a new example set consisting of the result of a fourier transformation for each attribute of the input example set.

5.8.49 FrequencyDiscretization

Group: Preprocessing.Data.Discretization

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- ○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)
- ○ **number_of_bins:** Defines the number of bins which should be used for each attribute. (integer; 2-+∞; default: 2)
- ○ **use_sqrt_of_examples:** If true, the number of bins is instead determined by the square root of the number of non-missing values. (boolean; default: false)
- ○ **use_long_range_names:** Indicates if long range names including the limits should be used. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Discretize numerical attributes into a user defined number of bins with equal frequency.

Description: This operator discretizes all numeric attributes in the dataset into nominal attributes. This discretization is performed by equal frequency binning, i.e. the thresholds of all bins is selected in a way that all bins contain the same number of numerical values. The number of bins is specified by a parameter, or, alternatively, is calculated as the square root of the number of examples with non-missing values (calculated for every single attribute). Skips all special attributes including the label.

5.8.50 FrequentItemSetAttributeCreator

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet
- FrequentItemSets

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates attributes from frequent item sets.

Description: This operator takes all FrequentItemSet sets within IOObjects and creates attributes for every frequent item set. This attributes indicate if the examples contains all values of this frequent item set. The attributes will contain values 0 or 1 and are numerical.

5.8.51 GHA

Group: Preprocessing.Attributes.Transformation

Required input:	**Generated output:**
• ExampleSet	• ExampleSet
	• Model

Parameters:

- ○ **number_of_components:** Number of components to compute. If '-1' nr of attributes is taken.' (integer; -1-+∞; default: -1)
- ○ **number_of_iterations:** Number of Iterations to apply the update rule. (integer; 0-+∞; default: 10)
- ○ **learning_rate:** The learning rate for GHA (small) (real; 0.0-+∞)
- ○ **local_random_seed:** The local random seed for this operator, uses global random number generator if -1. (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generalized Hebbian Algorithm (GHA). Performs an iterative principal components analysis.

Description: Generalized Hebbian Algorithm (GHA) is an iterative method to compute principal components. From a computational point of view, it can be advantageous to solve the eigenvalue problem by iterative methods which do not need to compute the covariance matrix directly. This is useful when the ExampleSet contains many Attributes (hundreds, thousands). The operator outputs a `GHAModel`. With the `ModelApplier` you can transform the features.

5.8.52 GeneratingForwardSelection

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)
- **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-$+\infty$; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-$+\infty$)
- **selection_direction:** Forward selection or backward elimination.

- ○ **keep_best:** Keep the best n individuals in each generation. (integer; 1-+∞; default: 1)
- ○ **generations_without_improval:** Stop after n generations without improval of the performance (-1: stops if the maximum_number_of_generations is reached). (integer; -1-+∞; default: 1)
- ○ **maximum_number_of_generations:** Delivers the maximum amount of generations (-1: might use or deselect all features). (integer; -1-+∞; default: -1)
- ○ **reciprocal_value:** Generate reciprocal values. (boolean; default: true)
- ○ **use_plus:** Generate sums. (boolean; default: true)
- ○ **use_diff:** Generate differences. (boolean; default: true)
- ○ **use_mult:** Generate products. (boolean; default: true)
- ○ **use_div:** Generate quotients. (boolean; default: true)
- ○ **use_max:** Generate maximum. (boolean; default: true)
- ○ **restrictive_selection:** Use restrictive generator selection (faster). (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.
- **turn:** The number of the current turn.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: This operator is a kind of nested forward selection and thus is (in contrast to a genetic algorithm) a directed search.

Description: This operator is a kind of nested forward selection and thus is (in contrast to a genetic algorithm) a directed search.

1. use forward selection in order to determine the best attributes
2. Create a new attribute by multiplying any of the original attributes with any of the attributes selected by the forward selection in the last turn
3. loop as long as performance increases

5.8.53 GeneratingGeneticAlgorithm

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)
- ○ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ○ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ○ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ○ **plot_generations:** Update the population plotter in these generations. (integer; 1-$+\infty$; default: 10)
- ○ **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- ○ **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- ○ **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)

- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **selection_scheme:** The selection scheme of this EA.
- **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)
- **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- **p_initialize:** Initial probability for an attribute to be switched on. (real; 0.0-1.0)
- **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- **crossover_type:** Type of the crossover.
- **use_plus:** Generate sums. (boolean; default: true)
- **use_diff:** Generate differences. (boolean; default: false)
- **use_mult:** Generate products. (boolean; default: true)
- **use_div:** Generate quotients. (boolean; default: false)
- **reciprocal_value:** Generate reciprocal values. (boolean; default: true)
- **max_number_of_new_attributes:** Max number of attributes to generate for an individual in one generation. (integer; 0-+∞; default: 1)
- **max_total_number_of_attributes:** Max total number of attributes in all generations (-1: no maximum). (integer; -1-+∞; default: -1)
- **p_generate:** Probability for an individual to be selected for generation. (real; 0.0-1.0)
- **p_mutation:** Probability for an attribute to be changed (-1: 1 / numberOfAtts). (real; -1.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: A genetic algorithm for feature selection and feature generation (GGA).

Description: In contrast to the class GeneticAlgorithm (see section 5.8.54), the GeneratingGeneticAlgorithm (see section 5.8.53) generates new attributes and thus can change the length of an individual. Therfore specialized mutation and crossover operators are being applied. Generators are chosen at random from a list of generators specified by boolean parameters.

Since this operator does not contain algorithms to extract features from value series, it is restricted to example sets with only single attributes. For automatic feature extraction from values series the value series plugin for RapidMiner written by Ingo Mierswa should be used. It is available at http://rapid-i.com[2]

5.8.54 GeneticAlgorithm

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

[2] http://rapid-i.com

Parameters:

- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **selection_scheme:** The selection scheme of this EA.
- **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)

- ○ **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- ○ **min_number_of_attributes:** Determines the minimum number of features used for the combinations. (integer; 1-+∞; default: 1)
- ○ **max_number_of_attributes:** Determines the maximum number of features used for the combinations (-1: try all combinations up to possible maximum) (integer; -1-+∞; default: -1)
- ○ **exact_number_of_attributes:** Determines the exact number of features used for the combinations (-1: use the feature range defined by min and max). (integer; -1-+∞; default: -1)
- ○ **p_initialize:** Initial probability for an attribute to be switched on. (real; 0.0-1.0)
- ○ **p_mutation:** Probability for an attribute to be changed (-1: 1 / numberOfAtt). (real; -1.0-1.0)
- ○ **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- ○ **crossover_type:** Type of the crossover.

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: A genetic algorithm for feature selection.

Description: A genetic algorithm for feature selection (mutation=switch features on and off, crossover=interchange used features). Selection is done by roulette wheel. Genetic algorithms are general purpose optimization / search algorithms that are suitable in case of no or little problem knowledge.

A genetic algorithm works as follows

1. Generate an initial population consisting of `population_size` individuals. Each attribute is switched on with probability `p_initialize`
2. For all individuals in the population
   - Perform mutation, i.e. set used attributes to unused with probability `p_mutation` and vice versa.
   - Choose two individuals from the population and perform crossover with probability `p_crossover`. The type of crossover can be selected by `crossover_type`.
3. Perform selection, map all individuals to sections on a roulette wheel whose size is proportional to the individual's fitness and draw `population_size` individuals at random according to their probability.
4. As long as the fitness improves, go to 2

If the example set contains value series attributes with blocknumbers, the whole block will be switched on and off.

5.8.55 GiniIndexWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator calculates the relevance of the attributes based on the Gini impurity index.

Description: This operator calculates the relevance of a feature by computing the Gini index of the class distribution, if the given example set would have been splitted according to the feature.

5.8.56 GroupBy

Group: Preprocessing.Other

Required input:	**Generated output:**
• ExampleSet	• SplittedExampleSet

Parameters:

- **attribute_name:** Name of the attribute which is used to create partitions. If no such attribute is found in the input-exampleset or the attribute is not nominal or not an integer, execution will fail. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Partitions an example set according to the values of a single nominal or integer attributes.

Description: This operator creates a SplittedExampleSet from an arbitrary example set. The partitions of the resulting example set are created according to the values of the specified attribute. This works similar to the `GROUP BY` clause in SQL.

Please note that the resulting example set is simply a splitted example set where no subset is selected. Following operators might decide to select one or several of the subsets, e.g. one of the aggregation operators.

5.8.57 GuessValueTypes

Group: Preprocessing.Attributes

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: (Re-)guesses all value types and changes them accordingly.

Description: This operator can be used to (re-)guess the value types of all attributes. This might be useful after some preprocessing transformations and "purifying" some of the columns, especially if columns which were nominal before can be handled as numerical columns. With this operator, the value types of all attributes do not have to be transformed manually with operators like NominalNumbers2Numerical (see section 5.8.82).

5.8.58 IdTagging

Group: Preprocessing

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **create_nominal_ids:** True if nominal ids (instead of integer ids) should be created (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Adds a new id attribute to the example set, each example is tagged with an incremented number.

Description: This operator adds an ID attribute to the given example set. Each example is tagged with an incremental integer number. If the example set already contains an id attribute, the old attribute will be removed before the new one is added.

5.8.59 InfiniteValueReplenishment

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **default:** Function to apply to all columns that are not explicitly specified by parameter 'columns'.

○ **columns:** List of replacement functions for each column. (list)

○ **replenishment_value:** This value is used for some of the replenishment types. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Replaces infinite values in examples.

Description: Replaces positive and negative infinite values in examples by one of the functions "none", "zero", "max_byte", "max_int", "max_double", and "missing". "none" means, that the value is not replaced. The max_xxx functions replace plus infinity by the upper bound and minus infinity by the lower bound of the range of the Java type xxx. "missing" means, that the value is replaced by nan (not a number), which is internally used to represent missing values. A MissingValueReplenishment (see section 5.8.72) operator can be used to replace missing values by average (or the mode for nominal attributes), maximum, minimum etc. afterwards.

For each attribute, the function can be selected using the parameter list `columns`. If an attribute's name appears in this list as a key, the value is used as the function name. If the attribute's name is not in the list, the function specified by the `default` parameter is used.

5.8.60 InfoGainRatioWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator calculates the relevance of the attributes based on the information gain ratio.

Description: This operator calculates the relevance of a feature by computing the information gain ratio for the class distribution (if exampleSet would have been splitted according to each of the given features).

5.8.61 InfoGainWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator calculates the relevance of the attributes based on the information gain.

Description: This operator calculates the relevance of a feature by computing the information gain in class distribution, if exampleSet would be splitted after the feature.

5.8.62 InteractiveAttributeWeighting

Group: Preprocessing.Attributes.Weighting

Generated output:

- AttributeWeights

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Shows a window with feature weights and allows users to change them.

Description: This operator shows a window with the currently used attribute weights and allows users to change the weight interactively.

5.8.63 IterativeWeightOptimization

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet
- AttributeWeights

Generated output:

- AttributeWeights
- PerformanceVector

Parameters:

- **parameter:** The parameter to set the weight value (string)
- **min_diff:** The minimum difference between two weights. (real; 0.0-+∞)
- **iterations_without_improvement:** Number iterations without performance improvement. (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **best_performance:** best performance
- **looptime:** The time elapsed since the current loop started.
- **performance:** performance of the last evaluated weight
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector, AttributeWeights].

Short description: Feature selection (forward, backward) guided by weights. Weights have to be updated after each iteration

Description: Performs an iterative feature selection guided by the AttributeWeights. Its an backward feature elemination where the feature with the smallest weight value is removed. After each iteration the weight values are updated (e.g. by a learner like JMySVMLearner).

5.8.64 KennardStoneSampling

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ◦ **sample_ratio:** The fraction of examples which should be sampled (real; 0.0-1.0)
- ◦ **absolute_sample:** If checked, the absolute number of examples will be used. Otherwise the ratio. (boolean; default: false)
- ◦ **sample_size:** The number of examples which should be sampled (integer; 1-+∞; default: 1000)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a sample from an example set using the Kennard-Stone algorithm.

Description: This operator performs a Kennard-Stone Sampling. This sampling Algorithm works as follows: First find the two points most separated in the training set. For each candidate point, find the smallest distance to any object already selected. Select that point for the training set which has the largest of these smallest distances As described above, this algorithm always gives the same result, due to the two starting points which are always the same. This implementation reduces number of iterations by holding a list with candidates of the largest smallest distances. The parameters controll the number of examples in the sample

5.8.65 KernelPCA

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- Model

Parameters:

- ○ **kernel_type:** The SVM kernel type
- ○ **kernel_gamma:** The SVM kernel parameter gamma (RBF, anova). (real; 0.0-+∞)
- ○ **kernel_sigma1:** The SVM kernel parameter sigma1 (Epanechnikov, Gaussian Combination, Multiquadric). (real; 0.0-+∞)
- ○ **kernel_sigma2:** The SVM kernel parameter sigma2 (Gaussian Combination). (real; 0.0-+∞)
- ○ **kernel_sigma3:** The SVM kernel parameter sigma3 (Gaussian Combination). (real; 0.0-+∞)
- ○ **kernel_degree:** The SVM kernel parameter degree (polynomial, anova, Epanechnikov). (real; 0.0-+∞)
- ○ **kernel_shift:** The SVM kernel parameter shift (polynomial, Multiquadric). (real; -∞-+∞)
- ○ **kernel_a:** The SVM kernel parameter a (neural). (real; -∞-+∞)
- ○ **kernel_b:** The SVM kernel parameter b (neural). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a kernel principal component analysis (PCA).

Description: This operator performs a kernel-based principal components analysis (PCA). Hence, the result will be the set of data points in a non-linearly transformed space. Please note that in contrast to the usual linear PCA the kernel variant does also works for large numbers of attributes but will become slow for large number of examples.

5.8.66 LOFOutlierDetection

Group: Preprocessing.Data.Outlier

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **minimal_points_lower_bound:** The lower bound for MinPts for the Outlier test (default value is 10) (integer; 0-+∞; default: 10)
- ○ **minimal_points_upper_bound:** The upper bound for MinPts for the Outlier test (default value is 20) (integer; 0-+∞; default: 20)
- ○ **distance_function:** choose which distance function will be used for calculating the distance between two objects

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Identifies outliers in the given ExampleSet based on local outlier factors.

Description: This operator performs a LOF outlier search. LOF outliers or outliers with a local outlier factor per object are density based outliers according to Breuning, Kriegel, et al.

The approach to find those outliers is based on measuring the density of objects and its relation to each other (referred to as local reachability density). Based on the average ratio of the local reachability density of an object and its k-nearest neighbours (e.g. the objects in its k-distance neighbourhood), a local outlier factor (LOF) is computed. The approach takes a parameter MinPts (actually specifying the "k") and it uses the maximum LOFs for objects in a MinPts range (lower bound and upper bound to MinPts).

Currently, the operator supports cosine, sine or squared distances in addition to the usual euclidian distance which can be specified by the corresponding parameter. In the first step, the objects are grouped into containers. For each object, using a radius screening of all other objects, all the available distances between that object and another object (or group of objects) on the (same) radius given by the distance are associated with a container. That container than has the distance information as well as the list of objects within that distance (usually only a few) and the information, how many objects are in the container.

In the second step, three things are done: (1) The containers for each object are counted in acending order according to the cardinality of the object list within the container (= that distance) to find the k-distances for each object and the objects in that k-distance (all objects in all the subsequent containers with a smaller distance). (2) Using this information, the local reachability densities are computed by using the maximum of the actual distance and the k-distance for each object pair (object and objects in k-distance) and averaging it by the cardinality of the k-neighbourhood and than taking the reciprocal value. (3) The LOF is computed for each MinPts value in the range (actually for all up to upper bound) by averaging the ratio between the MinPts-local reachability-density of all objects in the k-neighbourhood and the object itself. The maximum LOF in the MinPts range is passed as final LOF to each object.

Afterwards LOFs are added as values for a special real-valued outlier attribute in the example set which the operator will return.

5.8.67 LabelTrend2Classification

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator iterates over an example set with numeric label and converts the label values to either the class 'up' or the class 'down' based on whether the change from the previous label is positive or negative.

Description: This operator iterates over an example set with numeric label and converts the label values to either the class 'up' or the class 'down' based on whether the change from the previous label is positive or negative. Please note that this does not make sense on example sets where the examples are not ordered in some sense (like, e.g. ordered according to time). This operator might become useful in the context of a Series2WindowExamples operator.

5.8.68 LinearCombination

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **keep_all:** Indicates if the all old attributes should be kept. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator created a new example set containing only one feature: the linear combination of all input attributes.

Description: This operator applies a linear combination for each vector of the input ExampleSet, i.e. it creates a new feature containing the sum of all values of each row.

5.8.69 MergeNominalValues

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **attribute_name:** The name of the nominal attribute which values should be merged. (string)
- **first_value:** The first value which should be merged. (string)
- **second_value:** The second value which should be merged. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Merges two nominal values of a specified attribute.

Description: Merges two nominal values of a given attribute.

5.8.70 MinimalEntropyPartitioning

Group: Preprocessing.Data.Discretization

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)

○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)

○ **use_long_range_names:** Indicates if long range names including the limits should be used. (boolean; default: true)

○ **remove_useless:** Indicates if useless attributes, i.e. those containing only one single range, should be removed. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Discretizes numerical attributes. Bin boundaries are chosen as to minimize the entropy in the induced partitions.

Description: This operator discretizes all numeric attributes in the dataset into nominal attributes. The discretization is performed by selecting a bin boundary minimizing the entropy in the induced partitions. The method is then applied recursively for both new partitions until the stopping criterion is reached. For Details see a) Multi-interval discretization of continued-values attributes for classification learning (Fayyad,Irani) and b) Supervised and Unsupervised Discretization (Dougherty,Kohavi,Sahami). Skips all special attributes including the label.

Please note that this operator automatically removes all attributes with only one range (i.e. those attributes which are not actually discretized since the entropy criterion is not fulfilled). This behavior can be controlled by the remove_useless parameter.

5.8.71 MissingValueImputation

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **order:** Order of attributes in which missing values are estimated.
- ○ **sort:** Sort direction which is used in order strategy.
- ○ **iterate:** Impute missing values immediately after having learned the corresponding concept and iterate. (boolean; default: true)
- ○ **filter_learning_set:** Apply filter to learning set in addition to determination which missing values should be substituted. (boolean; default: false)
- ○ **learn_on_complete_cases:** Learn concepts to impute missing values only on the basis of complete cases (should be used in case learning approach can not handle missing values). (boolean; default: true)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators:

- Operator 1 (Filter) must be able to handle [ExampleSet] and must deliver [ExampleSet].
- Operator 2 (Learner) must be able to handle [ExampleSet] and must deliver [Model].

Short description: Replaces missing values in examples by applying a model learned for missing values.

Description: The operator MissingValueImpution imputes missing values by learning models for each attribute (except the label) and applying those models to the data set. The learner which is to be applied has to be given as inner operator. In order to specify a subset of the example set in which the missing values should be imputed (e.g. to limit the imputation to only numerical attributes) an arbitrary filter can be used as the first inner operator. In the case that such a filter is used, the learner has to be the second inner operator.

Please be aware that depending on the ability of the inner operator to handle missing values this operator might not be able to impute all missing values in some cases. This behaviour leads to a warning. It might hence be useful to combine this operator with a subsequent MissingValueReplenishment.

ATTENTION: This operator is currently under development and does not properly work in all cases. We do not recommend the usage of this operator in production systems.

5.8.72 MissingValueReplenishment

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

◦ **default:** Function to apply to all columns that are not explicitly specified by parameter 'columns'.

- **columns:** List of replacement functions for each column. (list)
- **replenishment_value:** This value is used for some of the replenishment types. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Replaces missing values in examples.

Description: Replaces missing values in examples. If a value is missing, it is replaced by one of the functions "minimum", "maximum", "average", and "none", which is applied to the non missing attribute values of the example set. "none" means, that the value is not replaced. The function can be selected using the parameter list `columns`. If an attribute's name appears in this list as a key, the value is used as the function name. If the attribute's name is not in the list, the function specified by the `default` parameter is used. For nominal attributes the mode is used for the average, i.e. the nominal value which occurs most often in the data. For nominal attributes and replacement type zero the first nominal value defined for this attribute is used. The replenishment "value" indicates that the user defined parameter should be used for the replacement.

5.8.73 MissingValueReplenishmentView

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Replaces missing values in examples. In contrast to the usual missing value replenishment, this operator does not change the underlying data but replaces the missing values on the fly by using a new view on the data.

Description: This operator simply creates a new view on the input data without changing the actual data or creating a new data table. The new view will not contain any missing values for regular attributes but the mean value (or mode) of the non-missing values instead.

5.8.74 ModelBasedSampling

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a sample from an example set. The sampling is based on a model and is constructed to focus on examples not yet explained.

Description: Sampling based on a learned model.

5.8.75 MultivariateSeries2WindowExamples

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **series_representation:** This parameter defines how the series values will be represented.

- ∘ **horizon:** The prediction horizon, i.e. the distance between the last window value and the value to predict. (integer; 1-+∞; default: 1)
- ∘ **window_size:** The width of the used windows. (integer; 1-+∞; default: 100)
- ∘ **step_size:** The step size of the used windows, i.e. the distance between the first values (integer; 1-+∞; default: 1)
- ∘ **create_single_attributes:** Indicates if the result example set should use single attributes instead of series attributes. (boolean; default: true)
- **label_dimension:** The dimension which should be used for creating the label values (counting starts with 0). (integer; 0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates examples from a multivariate value series data by windowing and using a label value with respect to a user defined prediction horizon in one of the dimensions.

Description: This operator transforms a given example set containing series data into a new example set containing single valued examples. For this purpose, windows with a specified window and step size are moved across the series and the attribute value lying horizon values after the window end is used as label which should be predicted. In contrast to the Series2WindowExamples operator, this operator can also handle multivariate series data. In order to specify the dimension which should be predicted, one must use the parameter "label_dimension" (counting starts at 0). If you want to predict all dimensions of your multivariate series you must setup several process definitions with different label dimensions, one for each dimension.

The series data must be given as ExampleSet. The parameter "series_representation" defines how the series data is represented by the ExampleSet:

- encode_series_by_examples: the series index variable (e.g. time) is encoded by the examples, i.e. there is a set of attributes (one for each dimension of the multivariate series) and a set of examples. Each example encodes the value vector for a new time point, each attribute value represents another dimension of the multivariate series.

- encode_series_by_attributes: the series index variable (e.g. time) is encoded by the attributes, i.e. there is a set of examples (one for each dimension of the multivariate series) and a set of attributes. The set of attribute values for all examples encodes the value vector for a new time point, each example represents another dimension of the multivariate series.

Please note that the encoding as examples is usually more efficient with respect to the memory usage.

5.8.76 NameBasedWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **name_regex_to_weights:** This list maps different regular expressions for the feature names to the specified weights. (list)
- ○ **default_weight:** This default weight is used for all features not covered by any of the regular expressions given in the list. (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator defines the weights for all features based on a list of regular expressions for the feature names which can be used to set a specified weight to features with a name fulfilling these expressions.

Description: This operator is able to create feature weights based on regular expressions defined for the feature names. For example, the user can map all features with a name starting with "Att" to the weight 0.5 by using the regular expression "Att.*". All other feature weights whose feature names are not covered by one of the regular expressions are set to the default weight.

Please note that the weights defined in the regular expression list are set in the order as they are defined in the list, i.e. weights can overwrite weights set before.

5.8.77 NoiseGenerator

Group: Preprocessing.Other

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **random_attributes:** Adds this number of random attributes. (integer; 0-+∞; default: 0)
- **label_noise:** Add this percentage of a numerical label range as a normal distributed noise or probability for a nominal label change. (real; 0.0-+∞)
- **default_attribute_noise:** The standard deviation of the default attribute noise. (real; 0.0-+∞)
- **noise:** List of noises for each attributes. (list)
- **offset:** Offset added to the values of each random attribute (real; -∞-+∞)
- **linear_factor:** Linear factor multiplicated with the values of each random attribute (real; 0.0-+∞)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Adds noise to existing attributes or add random attributes.

Description: This operator adds random attributes and white noise to the data. New random attributes are simply filled with random data which is not correlated to the label at all. Additionally, this operator might add noise to the label attribute or to the regular attributes. In case of a numerical label the given `label_noise` is the percentage of the label range which defines the standard deviation of normal distributed noise which is added to the label attribute. For nominal labels the parameter `label_noise` defines the probability to randomly change the nominal label value. In case of adding noise to regular attributes the parameter `default_attribute_noise` simply defines the standard deviation of normal distributed noise without using the attribute value range. Using the parameter list it is possible to set different noise levels for different attributes. However, it is not possible to add noise to nominal attributes.

5.8.78 Nominal2Binary

Group: Preprocessing.Attributes.Filter

Please use the operator 'Nominal2Binominal' instead.

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- ○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)
- ○ **transform_binominal:** Indicates if attributes which are already binominal should be transformed. (boolean; default: true)
- ○ **use_underscore_in_name:** Indicates if underscores should be used in the new attribute names instead of empty spaces and '='. Although the resulting names are harder to read for humans it might be more appropriate to use these if the data should be written into a database system. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all nominal values to binary attributes.

Description: This operator maps the values of all nominal values to binary attributes. For example, if a nominal attribute with name "costs" and possible nominal values "low", "moderate", and "high" is transformed, the result is a set of three binominal attributes "costs = low", "costs = moderate", and "costs = high". Only one of the values of each attribute is true for a specific example, the other values are false.

5.8.79 Nominal2Binominal

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)
- **transform_binominal:** Indicates if attributes which are already binominal should be transformed. (boolean; default: true)
- **use_underscore_in_name:** Indicates if underscores should be used in the new attribute names instead of empty spaces and '='. Although the resulting names are harder to read for humans it might be more appropriate to use these if the data should be written into a database system. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all nominal values to binominal (binary) attributes.

Description: This operator maps the values of all nominal values to binary attributes. For example, if a nominal attribute with name "costs" and possible nominal values "low", "moderate", and "high" is transformed, the result is a set of three binominal attributes "costs = low", "costs = moderate", and "costs = high". Only one of the values of each attribute is true for a specific example, the other values are false.

5.8.80 Nominal2Date

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **attribute_name:** The attribute which should be parsed. (string)
- ○ **date_type:** The desired value type for the parsed attribute.
- **date_format:** The parse format of the date values, for example "yyyy/MM/dd". (string)
- ○ **locale:** The used locale for date texts, for example "Wed" (English) in contrast to "Mi" (German).
- ○ **keep_old_attribute:** Indicates if the original date attribute should be kept. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Parses the nominal values for the specified attribute with respect to the given date format string and transforms the values into date values.

Description: This operator parses given nominal attributes in order to create date and / or time attributes. The date format can be specified by the date_format parameter. The old nominal attribute will be removed and replaced by a new date attribute if the corresponding parameter is not set (default).

Date and Time Patterns Date and time formats are specified by *date and time pattern* strings in the date_format parameter. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

- *G*: era designator; Text; example: AD
- *y*: year; Year; example: 1996; 96
- *M*: month in year; Month; example: July; Jul; 07
- *w*: week in year; Number; example: 27
- *W*: week in month; Number; example: 2
- *D*: day in year; Number; example: 189
- *d*: day in month; Number; example: 10
- *F*: day of week in month; Number; example: 2
- *E*: day in week; Text; example: Tuesday; Tue
- *a*: am/pm marker; Text; example: PM
- *H*: hour in day (0-23); Number; example: 0
- *k*: hour in day (1-24); Number; example: 24
- *K*: hour in am / pm (0-11); Number; example: 0
- *h*: hour in am / pm (1-12); Number; example: 12
- *m*: minute in hour; Number; example: 30
- *s*: second in minute; Number; example: 55
- *S*: millisecond; Number; example: 978
- *z*: time zone; General Time Zone; example: Pacific Standard Time; PST; GMT-08:00
- *Z*: time zone; RFC 822 Time Zone; example: -0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- *Text:* For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
- *Number:* For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
- *Year:* If the underlying calendar is the Gregorian calendar, the following rules are applied.
  - For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a *number*.
  - For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.
  - For parsing with the abbreviated year pattern ("y" or "yy"), this operator must interpret the abbreviated year relative to some century. It does this by adjusting dates to be within 80 years before and 20 years after the time the operator is created. For example, using a pattern of "MM/dd/yy" and the operator created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, "-1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

  Otherwise, calendar system specific forms are applied. If the number of pattern letters is 4 or more, a calendar specific long form is used. Otherwise, a calendar short or abbreviated form is used.
- *Month:* If the number of pattern letters is 3 or more, the month is interpreted as *text*; otherwise, it is interpreted as a *number*.
- *General time zone:* Time zones are interpreted as *text* if they have names. It is possible to define time zones by representing a GMT offset value. RFC 822 time zones are also accepted.

- *RFC 822 time zone:* For formatting, the RFC 822 4-digit time zone format is used. General time zones are also accepted.

This operator also supports *localized date and time pattern* strings by defining the locale parameter. In these strings, the pattern letters described above may be replaced with other, locale dependent, pattern letters.

Examples The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

- *“yyyy.MM.dd G ’at’ HH:mm:ss z”*: 2001.07.04 AD at 12:08:56 PDT
- *“EEE, MMM d, ”yy”*: Wed, Jul 4, ’01
- *“h:mm a”*: 12:08 PM
- *“hh ’o”clock’ a, zzzz”*: 12 o’clock PM, Pacific Daylight Time
- *“K:mm a, z”*: 0:08 PM, PDT
- *“yyyyy.MMMMM.dd GGG hh:mm aaa”*: 02001.July.04 AD 12:08 PM
- *“EEE, d MMM yyyy HH:mm:ss Z”*: Wed, 4 Jul 2001 12:08:56 -0700
- *“yyMMddHHmmssZ”*: 010704120856-0700
- *“yyyy-MM-dd’T’HH:mm:ss.SSSZ”*: 2001-07-04T12:08:56.235-0700

5.8.81 Nominal2Numeric

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)

○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all values to real values (usually simply using the internal indices).

Description: This operator maps all non numeric attributes to real valued attributes. Nothing is done for numeric attributes, binary attributes are mapped to 0 and 1.

For nominal attributes one of the following calculations will be done:

- Dichotomization, i.e. one new attribute for each value of the nominal attribute. The new attribute which corresponds to the actual nominal value gets value 1 and all other attributes gets value 0.
- Alternatively the values of nominal attributes can be seen as equally ranked, therefore the nominal attribute will simply be turned into a real valued attribute, the old values results in equidistant real values.

At this moment the same applies for ordinal attributes, in a future release more appropriate values based on the ranking between the ordinal values may be included.

5.8.82 NominalNumbers2Numerical

Group: Preprocessing.Attributes.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

○ **decimal_point_character:** Character that is used as decimal point. (string; default: '.')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all nominal values to numerical values by parsing the numbers if possible.

Description: This operator transforms nominal attributes into numerical ones. In contrast to the NominalToNumeric operator, this operator directly parses numbers from the wrongly as nominal values encoded values. Please note that this operator will first check the stored nominal mappings for all attributes. If (old) mappings are still stored which actually are nominal (without the corresponding data being part of the example set), the attribute will not be converted. Please use the operator GuessValueTypes (see section 5.8.57) in these cases.

5.8.83 Normalization

Group: Preprocessing

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ○ **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- ○ **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)
- ○ **z_transform:** Determines whether to perform a z-transformation (mean 0 and standard deviation 1) or not; this scaling ignores min- and max-setings (boolean; default: true)
- ○ **min:** The minimum value after normalization (real; $-\infty$-$+\infty$)
- ○ **max:** The maximum value after normalization (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Normalizes the attribute values for a specified range.

Description: This operator performs a normalization. This can be done between a user defined minimum and maximum value or by a z-transformation, i.e. on mean 0 and variance 1.

5.8.84 Numeric2Binary

Group: Preprocessing.Attributes.Filter

Please use the operator Numeric2Binominal instead

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **min:** The minimal value which is mapped to false (included). (real; -∞-+∞)
- ○ **max:** The maximal value which is mapped to false (included). (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all numeric values to 'false' if they are in the specified range (typical: equal 0.0) and to 'true' otherwise.

Description: Converts all numerical attributes to binary ones. If the value of an attribute is between the specified minimal and maximal value, it becomes *false*, otherwise *true*. If the value is missing, the new value will be missing. The default boundaries are both set to 0, thus only 0.0 is mapped to false and all other values are mapped to true.

5.8.85 Numeric2Binominal

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **min:** The minimal value which is mapped to false (included). (real; $-\infty$-$+\infty$)
- **max:** The maximal value which is mapped to false (included). (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all numeric values to 'false' if they are in the specified range (typical: equal 0.0) and to 'true' otherwise.

Description: Converts all numerical attributes to binary ones. If the value of an attribute is between the specified minimal and maximal value, it becomes *false*, otherwise *true*. If the value is missing, the new value will be missing. The default boundaries are both set to 0, thus only 0.0 is mapped to false and all other values are mapped to true.

5.8.86 Numeric2Polynominal

Group: Preprocessing.Attributes.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Maps all numeric values simply to the corresponding nominal values. Please use one of the discretization operators if you need more sophisticated nominalization methods.

Description: Converts all numerical attributes to nominal ones. Each numerical value is simply used as nominal value of the new attribute. If the value is missing, the new value will be missing. Please note that this operator might drastically increase memory usage if many different numerical values are used. Please use the available discretization operators then.

5.8.87 Obfuscator

Group: Preprocessing.Other

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- ◦ **obfuscation_map_file:** File where the obfuscator map should be written to. (filename)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Replaces all nominal values and attribute names by random strings.

Description: This operator takes an `ExampleSet` as input and maps all nominal values to randomly created strings. The names and the construction descriptions of all attributes will also replaced by random strings. This operator can be used to anonymize your data. It is possible to save the obfuscating map into a file which can be used to remap the old values and names. Please use the operator `Deobfuscator` for this purpose. The new example set can be written with an `ExampleSetWriter`.

5.8.88 PCA

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- Model

Parameters:

- ○ **variance_threshold:** Keep the all components with a cumulative variance smaller than the given threshold. (real; 0.0-1.0)
- ○ **dimensionality_reduction:** Indicates which type of dimensionality reduction should be applied
- ○ **number_of_components:** Keep this number of components. If '-1' then keep all components.' (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a principal component analysis (PCA) using the covariance matrix.

Description: This operator performs a principal components analysis (PCA) using the covariance matrix. The user can specify the amount of variance to cover in the original data when retaining the best number of principal components. The user can also specify manually the number of principal components. The operator outputs a `PCAModel`. With the `ModelApplier` you can transform the features.

5.8.89 PCAWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **component_number:** Indicates the number of the component from which the weights should be calculated. (integer; 1-$+\infty$; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator uses the factors of a PCA component (usually the first) as feature weights.

Description: Uses the factors of one of the principal components (default is the first) as feature weights. Please note that the PCA weighting operator is currently the only one which also works on data sets without a label, i.e. for unsupervised learning.

5.8.90 PSOWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: false)

- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **inertia_weight:** The (initial) weight for the old weighting. (real; 0.0-+∞)
- **local_best_weight:** The weight for the individual's best position during run. (real; 0.0-+∞)
- **global_best_weight:** The weight for the population's best position during run. (real; 0.0-+∞)
- **dynamic_inertia_weight:** If set to true the inertia weight is improved during run. (boolean; default: true)
- **min_weight:** The lower bound for the weights. (real; -∞-+∞)
- **max_weight:** The upper bound for the weights. (real; -∞-+∞)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **best:** The performance of the best individual ever (main criterion).
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Weight the features with a particle swarm optimization approach.

Description: This operator performs the weighting of features with a particle swarm approach.

5.8.91 Permutation

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Permutates the examples in the table. Caution: will increase memory usage!

Description: This operator creates a new, shuffled ExampleSet by making *a new copy* of the exampletable in main memory! Caution! System may run out of memory, if exampletable is too large.

5.8.92 PrincipalComponentsGenerator

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **min_variance_coverage:** The minimum variance to cover in the original data to determine the number of principal components. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Build the principal components of the given data.

Description: Builds the principal components of the given data. The user can specify the amount of variance to cover in the original data when retaining the best number of principal components. This operator makes use of the Weka implementation `PrincipalComponent`.

5.8.93 RandomSelection

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **number_of_features:** The number of features which should randomly selected (-1: use a random number). (integer; -1-+∞; default: -1)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator simply selects a random or a predefined number of random features.

Description: This operator selects a randomly chosen number of features randomly from the input example set. This can be useful in combination with a ParameterIteration operator or can be used as a baseline for significance test comparisons for feature selection techniques.

5.8.94 Relief

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **number_of_neighbors:** Number of nearest neigbors for relevance calculation. (integer; 1-+∞; default: 10)
- ○ **sample_ratio:** Number of examples used for determining the weights. (real; 0.0-1.0)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Relief measures the relevance of features by sampling examples and comparing the value of the current feature for the nearest example of the same and of a different class.

Description: Relief measures the relevance of features by sampling examples and comparing the value of the current feature for the nearest example of the same and of a different class. This version also works for multiple classes and regression data sets. The resulting weights are normalized into the interval between 0 and 1.

5.8.95 RemoveCorrelatedFeatures

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **correlation:** Use this correlation for the filter relation. (real; -1.0-1.0)
- **filter_relation:** Removes one of two features if their correlation fulfill this relation.
- **attribute_order:** The algorithm takes this attribute order to calculate correlation and filter.
- **use_absolute_correlation:** Indicates if the absolute value of the correlations should be used for comparison. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **features_removed:** Number of removed features
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Removes correlated features.

Description: Removes (un-) correlated features due to the selected filter relation. The procedure is quadratic in number of attributes. In order to get more stable results, the original, random, and reverse order of attributes is available.

Please note that this operator might fail in some cases when the attributes should be filtered out, e.g. it might not be able to remove for example all negative correlated features. The reason for this behaviour seems to be that for the complete m x m - matrix of correlations (for m attributes) the correlations will not be recalculated and hence not checked if one of the attributes of the current pair was already marked for removal. That means for three attributes a1, a2, and a3 that it might be that a2 was already ruled out by the negative correlation with a1 and is now not able to rule out a3 any longer.

5.8.96 RemoveUselessAttributes

Group: Preprocessing.Attributes.Selection

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **numerical_min_deviation:** Removes all numerical attributes with standard deviation less or equal to this threshold. (real; 0.0-+∞)
- **nominal_single_value_upper:** Removes all nominal attributes which provides more than the given amount of only one value. (real; 0.0-1.0)
- **nominal_single_value_lower:** Removes all nominal attributes which provides less than the given amount of at least one value (-1: remove attributes with values occuring only once). (real; -1.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Remove all useless attributes from an example set.

Description: Removes useless attribute from the example set. Useless attributes are

- nominal attributes which has the same value for more than `p` percent of all examples.
- numerical attributes which standard deviation is less or equal to a given deviation threshold `t`.

5.8.97 SOMDimensionalityReduction

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- **number_of_dimensions:** Defines the number of dimensions, the data shall be reduced. (integer; 1-+∞; default: 2)

- **net_size:** Defines the size of the SOM net, by setting the length of every edge of the net. (integer; 1-+∞; default: 30)
- **training_rounds:** Defines the number of trainnig rounds (integer; 1-+∞; default: 30)
- **learning_rate_start:** Defines the strength of an adaption in the first round. The strength will decrease every round until it reaches the learning_rate_end in the last round. (real; 0.0-+∞)
- **learning_rate_end:** Defines the strength of an adaption in the last round. The strength will decrease to this value in last round, beginning with learning_rate_start in the first round. (real; 0.0-+∞)
- **adaption_radius_start:** Defines the radius of the sphere around an stimulus, within an adaption occoures. This radius decreases every round, starting by adaption_radius_start in first round, to adaption_radius_end in last round. (real; 0.0-+∞)
- **adaption_radius_end:** Defines the radius of the sphere around an stimulus, within an adaption occoures. This radius decreases every round, starting by adaption_radius_start in first round, to adaption_radius_end in last round. (real; 0.0-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Trains a self-organizing map and applyes the examples on the map. The resulting coordinates are used as new attributes.

Description: This operator performs a dimensionality reduction based on a SOM (Self Organizing Map, aka Kohonen net).

5.8.98 SVDReduction

Group: Preprocessing.Attributes.Transformation

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- **dimensions:** the number of dimensions in the result representation (integer; 1-$+\infty$; default: 2)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a dimensionality reduction based on Singular Value Decomposition (SVD).

Description: A dimensionality reduction method based on Singular Value Decomposition. TODO: see super class

5.8.99 SVMWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **C:** The SVM complexity weighting factor. (real; $-\infty$-$+\infty$)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator uses the coefficients of a hyperplance calculated by an SVM as feature weights.

Description: Uses the coefficients of the normal vector of a linear SVM as feature weights. In contrast to most of the SVM based operators available in RapidMiner, this one works for multiple classes, too.

5.8.100 Sampling

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

◦ **sample_ratio:** The fraction of examples which should be sampled (real; 0.0-1.0)

◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a sample from an example set by drawing a fraction.

Description: Simple sampling operator. This operator performs a random sampling of a given fraction. For example, if the input example set contains 5000 examples and the sample ratio is set to 0.1, the result will have approximately 500 examples.

5.8.101 Series2WindowExamples

Group: Preprocessing.Data.Filter

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Parameters:

- **series_representation:** This parameter defines how the series values will be represented.
- **horizon:** The prediction horizon, i.e. the distance between the last window value and the value to predict. (integer; 1-$+\infty$; default: 1)
- **window_size:** The width of the used windows. (integer; 1-$+\infty$; default: 100)
- **step_size:** The step size of the used windows, i.e. the distance between the first values (integer; 1-$+\infty$; default: 1)
- **create_single_attributes:** Indicates if the result example set should use single attributes instead of series attributes. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates examples from an univariate value series data by windowing and using a label value with respect to a user defined prediction horizon.

Description: This operator transforms a given example set containing series data into a new example set containing single valued examples. For this purpose, windows with a specified window and step size are moved across the series and the series value lying horizon values after the window end is used as label which should be predicted. This operator can only be used for univariate series prediction. For the multivariate case, please use the operator MULTIVARIATESERIES2WINDOWEXAMPLES (see section 5.8.75).

The series data must be given as ExampleSet. The parameter "series_representation" defines how the series data is represented by the ExampleSet:

- encode_series_by_examples: the series index variable (e.g. time) is encoded by the examples, i.e. there is a *single* attribute and a set of examples. Each example encodes the value for a new time point.

- encode_series_by_attributes: the series index variable (e.g. time) is encoded by the attributes, i.e. there is a (set of) examples and a set of attributes. Each attribute value encodes the value for a new time point. If there is more than one example, the windowing is performed for each example independently and all resulting window examples are merged into a complete example set.

Please note that the encoding as examples is usually more efficient with respect to the memory usage. To ensure backward compatibility, the default representation is, however, set to time_as_attributes.

5.8.102 Single2Series

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Changes the value type of all single valued attributes and forms a value series from all attributes.

Description: Transforms all regular attributes of a given example set into a value series. All attributes must have the same value type. Attributes with block type value series can be used by special feature extraction operators or by the operators from the value series plugin.

5.8.103 SingleRuleWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator measures the relevance of features by constructing a single rule for each attribute and calculating the errors.

Description: This operator calculates the relevance of a feature by computing the error rate of a OneR Model on the exampleSet without this feature.

5.8.104 Sorting

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **attribute_name:** Indicates the attribute which should be used for determining the sorting. (string)
- ○ **sorting_direction:** Indicates the direction of the sorting.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator sorts the given example set according to a single attribute.

Description: This operator sorts the given example set according to a single attribute. The example set is sorted according to the natural order of the values of this attribute either in increasing or in decreasing direction.

5.8.105 StandardDeviationWeighting

Group: Preprocessing.Attributes.Weighting

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **normalize:** Indicates if the standard deviation should be divided by the minimum, maximum, or average of the attribute.

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Computes weights based on the (normalized) standard deviation of the attributes.

Description: Creates weights from the standard deviations of all attributes. The values can be normalized by the average, the minimum, or the maximum of the attribute.

5.8.106 StratifiedSampling

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ○ **sample_ratio:** The fraction of examples which should be sampled (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a stratified sample from an example set by drawing a fraction.

Description: Stratified sampling operator. This operator performs a random sampling of a given fraction. In contrast to the simple sampling operator, this operator performs a stratified sampling for data sets with nominal label attributes, i.e. the class distributions remains (almost) the same after sampling. Hence, this operator cannot be applied on data sets without a label or with a numerical label. In these cases a simple sampling without stratification is performed.

5.8.107 SymmetricalUncertaintyWeighting

Group: Preprocessing.Attributes.Weighting

Required input:	**Generated output:**
• ExampleSet	• ExampleSet • AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **number_of_bins:** The number of bins used for discretization of numerical attributes before the chi squared test can be performed. (integer; 2-+∞; default: 10)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator calculates the relevance of an attribute by measuring the symmetrical uncertainty with respect to the class.

Description: This operator calculates the relevance of an attribute by measuring the symmetrical uncertainty with respect to the class. The formulaization for this is:

```
relevance = 2 * (P(Class) - P(Class | Attribute)) / P(Class) + P(Attribute)
```

5.8.108 TFIDFFilter

Group: Preprocessing.Data.Filter

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **calculate_term_frequencies:** Indicates if term frequency values should be generated (must be done if input data is given as simple occurence counts). (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a TF-IDF filtering to the input data set.

Description: This operator generates TF-IDF values from the input data. The input example set must contain either simple counts, which will be normalized during calculation of the term frequency TF, or it already contains the calculated term frequency values (in this case no normalization will be done).

5.8.109 UseRowAsAttributeNames

Group: Preprocessing.Other

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

○ **row_number:** Indicates which row should be used as attribute names. Counting starts with 1. (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Uses the specified row as new attibute names and deletes the row from the data set.

Description: This operators uses the values of the specified row of the data set as new attribute names (including both regular and special columns). This might be useful for example after a transpose operation. The row will be deleted from the data set. Please note, however, that an internally used nominal mapping will not be removed and following operators like NominalNumbers2Numerical (see section 5.8.82) could possibly not work as expected. In order to correct the value types and nominal value mappings, one could use the operator GuessValueTypes (see section 5.8.57) after this operator.

5.8.110 UserBasedDiscretization

Group: Preprocessing.Data.Discretization

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- **return_preprocessing_model:** Indicates if the preprocessing model should also be returned (boolean; default: false)
- **create_view:** Create View to apply preprocessing instead of changing the data (boolean; default: false)
- **attribute_type:** Attribute type of the discretized attribute.
- **classes:** Defines the classes and the upper limits of each class. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Discretize numerical attributes into user defined bins.

Description: This operator discretizes a numerical attribute to either a nominal or an ordinal attribute. The numerical values are mapped to the classes according to the thresholds specified by the user. The user can define the classes by specifying the upper limits of each class. The lower limit of the next class is automatically specified as the upper limit of the previous one. A parameter defines to which adjacent class values that are equal to the given limits should be mapped. If the upper limit in the last list entry is not equal to Infinity, an additional class which is automatically named is added. If a '?' is given as class value the according numerical values are mapped to unknown values in the resulting attribute.

5.8.111 W-ChiSquaredAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)

- **M:** treat missing values as a seperate value. (boolean; default: false)
- **B:** just binarize numeric attributes instead of properly discretizing them. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: ChiSquaredAttributeEval :

Evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.112 W-CostSensitiveAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **C:** File name of a cost matrix to use. If this is not supplied, a cost matrix will be loaded on demand. The name of the on-demand file is the relation name of the training data plus ".cost", and the path to the on-demand file is specified with the -N option. (string)
- **N:** Name of a directory to search for cost files when loading costs on demand (default current directory). (string; default: '/home/ingo/workspace/yale')

- **cost-matrix:** The cost matrix in Matlab single line format. (string)
- **S:** The seed to use for random number generation. (real; $-\infty$-$+\infty$)
- **W:** Full name of base evaluator. Options after – are passed to the evaluator. (default: weka.attributeSelection.ReliefFAttributeEval) (string; default: 'weka.attributeSelection.ReliefFAttributeEval')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: A meta subset evaluator that makes its base subset evaluator cost-sensitive.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.113 W-FilteredAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **W:** Full name of base evaluator to use, followed by evaluator options. eg: "weka.attributeSelection.InfoGainAttributeEval -M" (string; default: 'weka.attributeSelection.InfoGainAttributeEval ')

- **F:** Full class name of filter to use, followed by filter options. eg: "weka.filters.supervised.instance. -M 1" (string; default: 'weka.filters.supervised.instance.SpreadSubsample -M 0.0 -X 0.0 -S 1')

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Class for running an arbitrary attribute evaluator on data that has been passed through an arbitrary filter (note: filters that alter the order or number of attributes are not allowed).

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.114 W-GainRatioAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **M:** treat missing values as a seperate value. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: GainRatioAttributeEval :

Evaluates the worth of an attribute by measuring the gain ratio with respect to the class.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.115 W-InfoGainAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **M:** treat missing values as a seperate value. (boolean; default: false)
- **B:** just binarize numeric attributes instead of properly discretizing them. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: InfoGainAttributeEval :

Evaluates the worth of an attribute by measuring the information gain with respect to the class.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.116 W-LatentSemanticAnalysis

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ◦ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ◦ **N:** Normalize input data. (boolean; default: false)
- ◦ **R:** Rank approximation used in LSA. May be actual number of LSA attributes to include (if greater than 1) or a proportion of total singular values to account for (if between 0 and 1). A value less than or equal to zero means use all latent variables.(default = 0.95) (real; -∞-+∞)
- ◦ **A:** Maximum number of attributes to include in transformed attribute names. (-1 = include all) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs latent semantic analysis and transformation of the data.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.117 W-OneRAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **S:** Random number seed for cross validation (default = 1) (real; -∞-+∞)
- ○ **F:** Number of folds for cross validation (default = 10) (real; -∞-+∞)
- ○ **D:** Use training data for evaluation rather than cross validaton (boolean; default: false)
- ○ **B:** Minimum number of objects in a bucket (passed on to OneR, default = 6) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: OneRAttributeEval :

Evaluates the worth of an attribute by using the OneR classifier.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.118 W-PrincipalComponents

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ◦ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ◦ **D:** Don't normalize input data. (boolean; default: false)
- ◦ **R:** Retain enough PC attributes to account for this proportion of variance in the original data. (default = 0.95) (real; -∞-+∞)
- ◦ **O:** Transform through the PC space and back to the original space. (boolean; default: false)
- ◦ **A:** Maximum number of attributes to include in transformed attribute names. (-1 = include all) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a principal components analysis and transformation of the data.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.119 W-ReliefFAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:	**Generated output:**
• ExampleSet	• ExampleSet • AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **M:** Specify the number of instances to sample when estimating attributes. If not specified, then all instances will be used. (real; -∞-+∞)
- ○ **D:** Seed for randomly sampling instances. (Default = 1) (real; -∞-+∞)
- ○ **K:** Number of nearest neighbours (k) used to estimate attribute relevances (Default = 10). (real; -∞-+∞)
- ○ **W:** Weight nearest neighbours by distance (boolean; default: false)
- ○ **A:** Specify sigma value (used in an exp function to control how quickly weights for more distant instances decrease. Use in conjunction with -W. Sensible value=1/5 to 1/10 of the number of nearest neighbours. (Default = 2) (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: ReliefFAttributeEval :

Evaluates the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

Further information: Kenji Kira, Larry A. Rendell: A Practical Approach to Feature Selection. In: Ninth International Workshop on Machine Learning, 249-256, 1992.

Igor Kononenko: Estimating Attributes: Analysis and Extensions of RELIEF. In: European Conference on Machine Learning, 171-182, 1994.

Marko Robnik-Sikonja, Igor Kononenko: An adaptation of Relief for attribute estimation in regression. In: Fourteenth International Conference on Machine Learning, 296-304, 1997.

5.8.120 W-SVMAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- **X:** Specify the constant rate of attribute elimination per invocation of the support vector machine. Default = 1. (real; -∞-+∞)
- **Y:** Specify the percentage rate of attributes to elimination per invocation of the support vector machine. Trumps constant rate (above threshold). Default = 0. (real; -∞-+∞)
- **Z:** Specify the threshold below which percentage attribute elimination reverts to the constant method. (real; -∞-+∞)
- **P:** Specify the value of P (epsilon parameter) to pass on to the support vector machine. Default = 1.0e-25 (real; -∞-+∞)
- **T:** Specify the value of T (tolerance parameter) to pass on to the support vector machine. Default = 1.0e-10 (real; -∞-+∞)
- **C:** Specify the value of C (complexity parameter) to pass on to the support vector machine. Default = 1.0 (real; -∞-+∞)
- **N:** Whether the SVM should 0=normalize/1=standardize/2=neither. (default 0=normalize) (real; -∞-+∞)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: SVMAttributeEval :

Evaluates the worth of an attribute by using an SVM classifier.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

Further information: I. Guyon, J. Weston, S. Barnhill, V. Vapnik (2002). Gene selection for cancer classification using support vector machines. Machine Learning. 46:389-422.

5.8.121 W-SymmetricalUncertAttributeEval

Group: Preprocessing.Attributes.Weighting.Weka

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights

Parameters:

- ○ **normalize_weights:** Activates the normalization of all weights. (boolean; default: true)
- ○ **M:** treat missing values as a seperate value. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: SymmetricalUncertAttributeEval :

Evaluates the worth of an attribute by measuring the symmetrical uncertainty with respect to the class.

Description: Performs the AttributeEvaluator of Weka with the same name to determine a sort of attribute relevance. These relevance values build an instance of AttributeWeights. Therefore, they can be used by other operators which make use of such weights, like weight based selection or search heuristics which use attribute weights to speed up the search. See the Weka javadoc for further operator and parameter descriptions.

5.8.122 WeightGuidedFeatureSelection

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)

- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **generations_without_improval:** Stop after n generations without improval of the performance (-1: stops if the number of features is reached). (integer; -1-+∞; default: 1)
- **use_absolute_weights:** Indicates that the absolute values of the input weights should be used to determine the feature adding order. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Adds iteratively features according to input attribute weights

Description: This operator uses input attribute weights to determine the order of features added to the feature set starting with the feature set containing only the feature with highest weight. The inner operators must provide a performance vector to determine the fitness of the current feature set, e.g. a cross validation of a learning scheme for a wrapper evaluation. Stops if adding the last `k` features does not increase the performance or if all features were added. The value of `k` can be set with the parameter `generations_without_improval`.

5.8.123 WeightOptimization

Group: Preprocessing.Attributes.Selection

Required input:

- ExampleSet
- AttributeWeights

Generated output:

- ParameterSet
- PerformanceVector
- AttributeWeights

Parameters:

- ○ **parameter:** The parameter to set the weight value (string)
- ○ **selection_direction:** Forward selection or backward elimination.
- ○ **min_diff:** The minimum difference between two weights. (real; 0.0-+∞)
- ○ **iterations_without_improvement:** Number iterations without performance improvement. (integer; 1-+∞; default: 1)

Values:

- **applycount:** The number of times the operator was applied.
- **best_performance:** best performance
- **looptime:** The time elapsed since the current loop started.
- **performance:** performance of the last evaluated weight
- **time:** The time elapsed since this operator started.
- **weight:** currently used weight

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: Feature selection (forward, backward) guided by weights.

Description: Performs a feature selection guided by the AttributeWeights. Forward selection means that features with the highest weight-value are selected first (starting with an empty selection). Backward elemination means that features with the smallest weight value are eleminated first (starting with the full feature set).

5.8.124 WeightedBootstrapping

Group: Preprocessing.Data.Sampling

Required input:

- ExampleSet

Generated output:

- ExampleSet

Parameters:

- ○ **sample_ratio:** This ratio determines the size of the new example set. (real; 0.0-+∞)
- ○ **local_random_seed:** Local random seed for this operator (-1: use global random seed). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a bootstrapped sample by weighted sampling with replacement.

Description: This operator constructs a bootstrapped sample from the given example set which must provide a weight attribute. If no weight attribute was provided this operator will stop the process with an error message. See the operator Bootstrapping (see section 5.8.16) for more information.

5.8.125 YAGGA

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

- ◦ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ◦ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ◦ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ◦ **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)
- ◦ **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- ◦ **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- ◦ **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- ◦ **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- ◦ **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- ◦ **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- ◦ **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- ◦ **selection_scheme:** The selection scheme of this EA.
- ◦ **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- ◦ **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- ◦ **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)
- ◦ **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- ◦ **p_initialize:** Initial probability for an attribute to be switched on. (real; 0.0-1.0)

- **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- **crossover_type:** Type of the crossover.
- **use_plus:** Generate sums. (boolean; default: true)
- **use_diff:** Generate differences. (boolean; default: false)
- **use_mult:** Generate products. (boolean; default: true)
- **use_div:** Generate quotients. (boolean; default: false)
- **reciprocal_value:** Generate reciprocal values. (boolean; default: true)
- **p_mutation:** Probability for mutation (-1: 1/n). (real; 0.0-1.0)
- **max_total_number_of_attributes:** Max total number of attributes in all generations (-1: no maximum). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Yet Another GGA (Generating Geneting Algorithm). On average individuals (= selected attribute sets) will keep their original length, unless longer or shorther ones prove to have a better fitness.

Description: YAGGA is an acronym for Yet Another Generating Genetic Algorithm. Its approach to generating new attributes differs from the original one. The (generating) mutation can do one of the following things with different probabilities:

- Probability $p/4$: Add a newly generated attribute to the feature vector

- Probability $p/4$: Add a randomly chosen original attribute to the feature vector
- Probability $p/2$: Remove a randomly chosen attribute from the feature vector

Thus it is guaranteed that the length of the feature vector can both grow and shrink. On average it will keep its original length, unless longer or shorter individuals prove to have a better fitness.

Since this operator does not contain algorithms to extract features from value series, it is restricted to example sets with only single attributes. For (automatic) feature extraction from values series the value series plugin for RapidMiner written by Ingo Mierswa should be used. It is available at http://rapid-i.com[3].

5.8.126 YAGGA2

Group: Preprocessing.Attributes.Generation

Required input:

- ExampleSet

Generated output:

- ExampleSet
- AttributeWeights
- PerformanceVector

Parameters:

- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)
- ◦ **show_stop_dialog:** Determines if a dialog with a button should be displayed which stops the run: the best individual is returned. (boolean; default: false)
- ◦ **user_result_individual_selection:** Determines if the user wants to select the final result individual from the last population. (boolean; default: false)
- ◦ **show_population_plotter:** Determines if the current population should be displayed in performance space. (boolean; default: false)
- ◦ **plot_generations:** Update the population plotter in these generations. (integer; 1-+∞; default: 10)

[3] http://rapid-i.com

- **constraint_draw_range:** Determines if the draw range of the population plotter should be constrained between 0 and 1. (boolean; default: false)
- **draw_dominated_points:** Determines if only points which are not Pareto dominated should be painted. (boolean; default: true)
- **population_criteria_data_file:** The path to the file in which the criteria data of the final population should be saved. (filename)
- **maximal_fitness:** The optimization will stop if the fitness reaches the defined maximum. (real; 0.0-+∞)
- **population_size:** Number of individuals per generation. (integer; 1-+∞; default: 5)
- **maximum_number_of_generations:** Number of generations after which to terminate the algorithm. (integer; 1-+∞; default: 30)
- **generations_without_improval:** Stop criterion: Stop after n generations without improval of the performance (-1: perform all generations). (integer; -1-+∞; default: -1)
- **selection_scheme:** The selection scheme of this EA.
- **tournament_size:** The fraction of the current population which should be used as tournament members (only tournament selection). (real; 0.0-1.0)
- **start_temperature:** The scaling temperature (only Boltzmann selection). (real; 0.0-+∞)
- **dynamic_selection_pressure:** If set to true the selection pressure is increased to maximum during the complete optimization run (only Boltzmann and tournament selection). (boolean; default: true)
- **keep_best_individual:** If set to true, the best individual of each generations is guaranteed to be selected for the next generation (elitist selection). (boolean; default: false)
- **p_initialize:** Initial probability for an attribute to be switched on. (real; 0.0-1.0)
- **p_crossover:** Probability for an individual to be selected for crossover. (real; 0.0-1.0)
- **crossover_type:** Type of the crossover.
- **use_plus:** Generate sums. (boolean; default: true)
- **use_diff:** Generate differences. (boolean; default: false)
- **use_mult:** Generate products. (boolean; default: true)
- **use_div:** Generate quotients. (boolean; default: false)
- **reciprocal_value:** Generate reciprocal values. (boolean; default: true)
- **p_mutation:** Probability for mutation (-1: 1/n). (real; 0.0-1.0)

- **max_total_number_of_attributes:** Max total number of attributes in all generations (-1: no maximum). (integer; -1-+∞; default: -1)
- **use_square_roots:** Generate square root values. (boolean; default: false)
- **use_power_functions:** Generate the power of one attribute and another. (boolean; default: true)
- **use_sin:** Generate sinus. (boolean; default: true)
- **use_cos:** Generate cosinus. (boolean; default: false)
- **use_tan:** Generate tangens. (boolean; default: false)
- **use_atan:** Generate arc tangens. (boolean; default: false)
- **use_exp:** Generate exponential functions. (boolean; default: true)
- **use_log:** Generate logarithmic functions. (boolean; default: false)
- **use_absolute_values:** Generate absolute values. (boolean; default: true)
- **use_min:** Generate minimum values. (boolean; default: false)
- **use_max:** Generate maximum values. (boolean; default: false)
- **use_sgn:** Generate signum values. (boolean; default: false)
- **use_floor_ceil_functions:** Generate floor, ceil, and rounded values. (boolean; default: false)
- **restrictive_selection:** Use restrictive generator selection (faster). (boolean; default: true)
- **remove_useless:** Remove useless attributes. (boolean; default: true)
- **remove_equivalent:** Remove equivalent attributes. (boolean; default: true)
- **equivalence_samples:** Check this number of samples to prove equivalency. (integer; 1-+∞; default: 5)
- **equivalence_epsilon:** Consider two attributes equivalent if their difference is not bigger than epsilon. (real; 0.0-+∞)
- **equivalence_use_statistics:** Recalculates attribute statistics before equivalence check. (boolean; default: true)
- **max_construction_depth:** The maximum depth for the argument attributes used for attribute construction (-1: allow all depths). (integer; -1-+∞; default: -1)
- **unused_functions:** Space separated list of functions which are not allowed in arguments for attribute construction. (string)
- **constant_generation_prob:** Generate random constant attributes with this probability. (real; 0.0-1.0)
- **associative_attribute_merging:** Post processing after crossover (only possible for runs with only one generator). (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **average_length:** The average number of attributes.
- **best:** The performance of the best individual ever (main criterion).
- **best_length:** The number of attributes of the best example set.
- **generation:** The number of the current generation.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The performance of the current generation (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must be able to handle [ExampleSet] and must deliver [PerformanceVector].

Short description: Improved version of Yet Another GGA (Generating Geneting Algorithm).

Description: YAGGA is an acronym for Yet Another Generating Genetic Algorithm. Its approach to generating new attributes differs from the original one. The (generating) mutation can do one of the following things with different probabilities:

- Probability $p/4$: Add a newly generated attribute to the feature vector
- Probability $p/4$: Add a randomly chosen original attribute to the feature vector
- Probability $p/2$: Remove a randomly chosen attribute from the feature vector

Thus it is guaranteed that the length of the feature vector can both grow and shrink. On average it will keep its original length, unless longer or shorter individuals prove to have a better fitness.

In addition to the usual YAGGA operator, this operator allows more feature generators and provides several techniques for intron prevention. This leads to smaller example sets containing less redundant features.

Since this operator does not contain algorithms to extract features from value series, it is restricted to example sets with only single attributes. For (automatic) feature extraction from values series the value series plugin for RapidMiner should be used.

For more information please refer to

Mierswa, Ingo (2007): *RobustGP: Intron-Free Multi-Objective Feature Construction* (to appear)

5.9 Performance Validation

When applying a model to a real-world problem, one usually wants to rely on a statistically significant estimation of its performance. There are several ways to measure this performance by comparing predicted label and true label. This can of course only be done if the latter is known. The usual way to estimate performance is therefore, to split the labelled dataset into a training set and a test set, which can be used for performance estimation. The operators in this section realise different ways of evaluating the performance of a model and splitting the dataset into training and test set.

5.9.1 Anova

Group: Validation.Significance

Required input:

- PerformanceVector

Generated output:

- PerformanceVector
- SignificanceTestResult

Parameters:

- ○ **alpha:** The probability threshold which determines if differences are considered as significant. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs ANalysis Of VAriances to determine the probability for the null hypothesis 'the actual means are the same'.

Description: Determines if the null hypothesis (all actual mean values are the same) holds for the input performance vectors. This operator uses an ANalysis Of VAriances approach to determine probability that the null hypothesis is wrong.

5.9.2 AttributeCounter

Group: Validation.Performance

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

○ **optimization_direction:** Indicates if the fitness should for maximal or minimal number of features.

Values:

- **applycount:** The number of times the operator was applied.
- **attributes:** The currently selected number of attributes.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: This operator created a performance vector containing the number of features of the input example set.

Description: Returns a performance vector just counting the number of attributes currently used for the given example set.

5.9.3 BatchSlidingWindowValidation

Group: Validation.Other

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- **cumulative_training:** Indicates if each training batch should be added to the old one or should replace the old one. (boolean; default: false)
- **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: Performs a sliding window validation on predefined example batches.

Description: The `BatchSlidingWindowValidation` is similar to the usual SlidingWindowValidation (see section 5.9.26). This operator, however, does not split the data itself in windows of predefined widths but uses the partition defined by the special attribute "batch". This can be an arbitrary nominal or integer attribute where each possible value occurs at least once (since many learning schemes depend on this minimum number of examples).

In each iteration, the next training batch is used for learning and the batch after this for prediction. It is also possible to perform a cumulative batch creation where each test batch will simply be added to the current training batch for the training in the next generation.

The first inner operator must accept an ExampleSet while the second must accept an ExampleSet and the output of the first (which is in most cases a Model) and must produce a PerformanceVector.

5.9.4 BatchXValidation

Group: Validation.Other

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- ◦ **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: A batched cross-validation in order to estimate the performance of a learning operator according to predefined example batches.

Description: `BatchXValidation` encapsulates a cross-validation process. The example set S is split up into *number_of_validations* subsets S_i. The inner operators are applied *number_of_validations* times using S_i as the test set (input of the second inner operator) and $S \backslash S_i$ training set (input of the first inner operator).

In contrast to the usual cross validation operator (see XValidation (see section 5.9.32)) this operator does not (randomly) split the data itself but uses the partition defined by the special attribute "batch". This can be an arbitrary nominal or integer attribute where each possible value occurs at least once (since many learning schemes depend on this minimum number of examples).

The first inner operator must accept an ExampleSet while the second must accept an ExampleSet and the output of the first (which is in most cases a Model) and must produce a PerformanceVector.

5.9.5 BinominalClassificationPerformance

Group: Validation

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **main_criterion:** The criterion used for comparing performance vectors.
- **AUC:** The area under a ROC curve. Given example weights are also considered. Please note that the second class is considered to be positive. (boolean; default: false)

- **precision:** Relative number of correctly as positive classified examples among all examples classified as positive (boolean; default: false)
- **recall:** Relative number of correctly as positive classified examples among all positive examples (boolean; default: false)
- **lift:** The lift of the positive class (boolean; default: false)
- **fallout:** Relative number of incorrectly as positive classified examples among all negative examples (boolean; default: false)
- **f_measure:** Combination of precision and recall: f=2pr/(p+r) (boolean; default: false)
- **false_positive:** Absolute number of incorrectly as positive classified examples (boolean; default: false)
- **false_negative:** Absolute number of incorrectly as negative classified examples (boolean; default: false)
- **true_positive:** Absolute number of correctly as positive classified examples (boolean; default: false)
- **true_negative:** Absolute number of correctly as negative classified examples (boolean; default: false)
- **skip_undefined_labels:** If set to true, examples with undefined labels are skipped. (boolean; default: true)
- **comparator_class:** Fully qualified classname of the PerformanceComparator implementation. (string)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)

Values:

- **AUC:** The area under a ROC curve. Given example weights are also considered. Please note that the second class is considered to be positive.
- **applycount:** The number of times the operator was applied.
- **f_measure:** Combination of precision and recall: f=2pr/(p+r)
- **fallout:** Relative number of incorrectly as positive classified examples among all negative examples
- **false_negative:** Absolute number of incorrectly as negative classified examples
- **false_positive:** Absolute number of incorrectly as positive classified examples
- **lift:** The lift of the positive class
- **looptime:** The time elapsed since the current loop started.

- **performance:** The last performance (main criterion).
- **precision:** Relative number of correctly as positive classified examples among all examples classified as positive
- **recall:** Relative number of correctly as positive classified examples among all positive examples
- **time:** The time elapsed since this operator started.
- **true_negative:** Absolute number of correctly as negative classified examples
- **true_positive:** Absolute number of correctly as positive classified examples

Short description: This operator delivers as output a list of performance values according to a list of selected performance criteria (for binominal classification tasks).

Description: This performance evaluator operator should be used for classification tasks, i.e. in cases where the label attribute has a binominal value type. Other polynominal classification tasks, i.e. tasks with more than two classes can be handled by the ClassificationPerformance (see section 5.9.8) operator. This operator expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of performance values according to a list of performance criteria that it calculates. If an input performance vector was already given, this is used for keeping the performance values.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a ProcessLogOperator using the same names. The main criterion is used for comparisons and need to be specified only for processes where performance vectors are compared, e.g. feature selection or other meta optimization process setups. If no other main criterion was selected, the first criterion in the resulting performance vector will be assumed to be the main criterion.

The resulting performance vectors are usually compared with a standard performance comparator which only compares the fitness values of the main criterion. Other implementations than this simple comparator can be specified using the parameter *comparator_class*. This may for instance be useful if you want to compare performance vectors according to the weighted sum of the individual criteria. In order to implement your own comparator, simply subclass `PerformanceComparator`. Please note that for true multi-objective optimization usually another selection scheme is used instead of simply replacing the performance comparator.

5.9.6 BootstrappingValidation

Group: Validation.Other

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- ◦ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ◦ **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- ◦ **number_of_validations:** Number of subsets for the crossvalidation. (integer; 2-$+\infty$; default: 10)
- ◦ **sample_ratio:** This ratio of examples will be sampled (with replacement) in each iteration. (real; 0.0-$+\infty$)
- ◦ **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors. (boolean; default: true)
- ◦ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-$+\infty$; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: This operator encapsulates an iterated bootstrapping sampling with performance evaluation on the remaining examples.

Description: This validation operator performs several bootstrapped samplings (sampling with replacement) on the input set and trains a model on these samples. The remaining samples, i.e. those which were not sampled, build a test set on which the model is evaluated. This process is repeated for the specified number of iterations after which the average performance is calculated.

The basic setup is the same as for the usual cross validation operator. The first inner operator must provide a model and the second a performance vector. Please note that this operator does not regard example weights, i.e. weights specified in a weight column.

5.9.7 CFSFeatureSetEvaluator

Group: Validation.Performance

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Calculates a performance measure based on the Correlation (filter evaluation).

Description: CFS attribute subset evaluator. For more information see:

Hall, M. A. (1998). Correlation-based Feature Subset Selection for Machine Learning. Thesis submitted in partial fulfilment of the requirements of the degree of Doctor of Philosophy at the University of Waikato.

This operator creates a filter based performance measure for a feature subset. It evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class while having low intercorrelation are preferred.

This operator can be applied on both numerical and nominal data sets.

5.9.8 ClassificationPerformance

Group: Validation

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **main_criterion:** The criterion used for comparing performance vectors.
- **accuracy:** Relative number of correctly classified examples (boolean; default: false)
- **classification_error:** Relative number of misclassified examples (boolean; default: false)
- **kappa:** The kappa statistics for the classification (boolean; default: false)
- **weighted_mean_recall:** The weighted mean of all per class recall measurements. (boolean; default: false)
- **weighted_mean_precision:** The weighted mean of all per class precision measurements. (boolean; default: false)
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho. (boolean; default: false)

- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b. (boolean; default: false)
- **absolute_error:** Average absolute deviation of the prediction from the actual value (boolean; default: false)
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value) (boolean; default: false)
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted. (boolean; default: false)
- **root_mean_squared_error:** Averaged root-mean-squared error (boolean; default: false)
- **root_relative_squared_error:** Averaged root-relative-squared error (boolean; default: false)
- **squared_error:** Averaged squared error (boolean; default: false)
- **correlation:** Returns the correlation coefficient between the label and predicted label. (boolean; default: false)
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label. (boolean; default: false)
- **cross-entropy:** The cross-entropy of a classifier, defined as the sum over the logarithms of the true label's confidences divided by the number of examples (boolean; default: false)
- **margin:** The margin of a classifier, defined as the minimal confidence for the correct label. (boolean; default: false)
- **soft_margin_loss:** The average soft margin loss of a classifier, defined as the average of all 1 - confidences for the correct label. (boolean; default: false)
- **logistic_loss:** The logistic loss of a classifier, defined as the average of ln(1 + exp(- [confidence of the correct class])) (boolean; default: false)
- **skip_undefined_labels:** If set to true, examples with undefined labels are skipped. (boolean; default: true)
- **comparator_class:** Fully qualified classname of the PerformanceComparator implementation. (string)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)
- **class_weights:** The weights for all classes (first column: class name, second column: weight), empty: using 1 for all classes. (list)

Values:

- **absolute_error:** Average absolute deviation of the prediction from the actual value
- **accuracy:** Relative number of correctly classified examples
- **applycount:** The number of times the operator was applied.
- **classification_error:** Relative number of misclassified examples
- **correlation:** Returns the correlation coefficient between the label and predicted label.
- **cross-entropy:** The cross-entropy of a classifier, defined as the sum over the logarithms of the true label's confidences divided by the number of examples
- **kappa:** The kappa statistics for the classification
- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b.
- **logistic_loss:** The logistic loss of a classifier, defined as the average of ln(1 + exp(- [confidence of the correct class]))
- **looptime:** The time elapsed since the current loop started.
- **margin:** The margin of a classifier, defined as the minimal confidence for the correct label.
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted.
- **performance:** The last performance (main criterion).
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
- **root_mean_squared_error:** Averaged root-mean-squared error
- **root_relative_squared_error:** Averaged root-relative-squared error
- **soft_margin_loss:** The average soft margin loss of a classifier, defined as the average of all 1 - confidences for the correct label.
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho.
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label.
- **squared_error:** Averaged squared error
- **time:** The time elapsed since this operator started.
- **weighted_mean_precision:** The weighted mean of all per class precision measurements.
- **weighted_mean_recall:** The weighted mean of all per class recall measurements.

Short description: This operator delivers as output a list of performance values according to a list of selected performance criteria (for all classification tasks).

Description: This performance evaluator operator should be used for classification tasks, i.e. in cases where the label attribute has a (poly-)nominal value type. The operator expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of performance values according to a list of performance criteria that it calculates. If an input performance vector was already given, this is used for keeping the performance values.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a ProcessLogOperator using the same names. The main criterion is used for comparisons and need to be specified only for processes where performance vectors are compared, e.g. feature selection or other meta optimization process setups. If no other main criterion was selected, the first criterion in the resulting performance vector will be assumed to be the main criterion.

The resulting performance vectors are usually compared with a standard performance comparator which only compares the fitness values of the main criterion. Other implementations than this simple comparator can be specified using the parameter *comparator_class*. This may for instance be useful if you want to compare performance vectors according to the weighted sum of the individual criteria. In order to implement your own comparator, simply subclass `PerformanceComparator`. Please note that for true multi-objective optimization usually another selection scheme is used instead of simply replacing the performance comparator.

5.9.9 ClusterCentroidEvaluator

Group: Validation.Performance.Clustering

Required input:

- ExampleSet
- ClusterModel

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

- **keep_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **main_criterion:** The main criterion to use
- **main_criterion_only:** return the main criterion only (boolean; default: false)
- **normalize:** divide the criterion by the number of features (boolean; default: false)
- **maximize:** do not multiply the result by minus one (boolean; default: false)

Values:

- **AVD:** avg_within_distance
- **DB:** DaviesBouldin
- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Delivers a performance based on cluster centroids.

Description: An evaluator for centroid based clustering methods.

5.9.10 ClusterDensityEvaluator

Group: Validation.Performance.Clustering

Required input:

- ClusterModel
- SimilarityMeasure

Generated output:

- PerformanceVector

Parameters:

- **keep_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **keep_similarity_measure:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **clusterdensity:** Avg. within cluster similarity/distance
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Delivers a performance based on cluster densities.

Description: This operator is used to evaluate a flat cluster model based on diverse density measures. Currently, only the avg. within cluster similarity/distance (depending on the type of SimilarityMeasure input object used) is supported.

5.9.11 ClusterModelFScore

Group: Validation.Performance.Clustering

Required input:	**Generated output:**
• HierarchicalClusterModel	• PerformanceVector

Parameters:

○ **weight_clusters:** should the result clusters be weighted by the fraction of items they contain (boolean; default: true)

○ **switch:** switch the both cluster models (boolean; default: false)

○ **symmetric:** build the average of a two-way comparison (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Compares a result cluster model to a reference model by averaging the f-measures of the best matching concepts.

Description: Compares two cluster models by searching for each concept a best matching one in the compared cluster model in terms of f-measure. The average f-measure of the best matches is then the overall cluster model similarity.

5.9.12 ClusterModelLabelComparator

Group: Validation.Performance.Clustering

Required input:

- ClusterModel

Generated output:

- PerformanceVector

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Determines the performance by comparing the labels in hierarchical cluster models.

Description: Compares two hierarchical clustering models according to the label of their root node. If this label is equal, 1 is returned, 0 otherwise.

5.9.13 ClusterNumberEvaluator

Group: Validation.Performance.Clustering

Required input:

- FlatClusterModel

Generated output:

- PerformanceVector

Parameters:

○ **keep_flat_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **clusternumber:** The number of clusters.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Delivers a performance based on the number of clusters.

Description: This operator does actually not compute a performance criterion but simply provides the number of cluster as a value.

5.9.14 ConsistencyFeatureSetEvaluator

Group: Validation.Performance

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Calculates a performance measure based on the consistency (filter evaluation).

Description: Consistency attribute subset evaluator. For more information see:

Liu, H., and Setiono, R., (1996). A probabilistic approach to feature selection - A filter solution. In 13th International Conference on Machine Learning (ICML'96), July 1996, pp. 319-327. Bari, Italy.

This operator evaluates the worth of a subset of attributes by the level of consistency in the class values when the training instances are projected onto the subset of attributes. Consistency of any subset can never be lower than that of the full set of attributes, hence the usual practice is to use this subset evaluator in conjunction with a Random or Exhaustive search which looks for the smallest subset with consistency equal to that of the full set of attributes.

This operator can only be applied for classification data sets, i.e. where the label attribute is nominal.

5.9.15 ConstraintClusterValidation

Group: Validation.Performance.Clustering

Required input:

- ClusterConstraintList
- ClusterModel

Generated output:

- PerformanceVector

Parameters:

- ○ **keep_cluster_constraint_list:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **keep_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Validate constrained k-means

Description: Evaluates a ClusterModel with regard to a given ClusterConstraintList and takes the weight of the violated constraints as performance value.

5.9.16 CostEvaluator

Group: Validation.Performance

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- ○ **keep_exampleSet:** Indicates if the example set should be kept. (boolean; default: false)
- • **cost_matrix:** The cost matrix in Matlab single line format (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: A cost evaluator delivers as output the costs for given classification results.

Description: This operator provides the ability to evaluate classification costs. Therefore a cost matrix might be specified, denoting the costs for every possible classification outcome: predicted label x real label. Costs will be minimized during optimization.

5.9.17 FixedSplitValidation

Group: Validation.Other

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- **training_set_size:** Absolute size required for the training set (-1: use rest for training) (integer; -1-+∞; default: 100)
- **test_set_size:** Absolute size required for the test set (-1: use rest for testing) (integer; -1-+∞; default: -1)
- **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: A FixedSplitValidation splits up the example set at a fixed point into a training and test set and evaluates the model.

Description: A FixedSplitValidationChain splits up the example set at a fixed point into a training and test set and evaluates the model (linear sampling). For non-linear sampling methods, i.e. the data is shuffled, the specified amounts of data are used as training and test set. The sum of both must be smaller than the input example set size.

At least either the training set size must be specified (rest is used for testing) or the test set size must be specified (rest is used for training). If both are specified, the rest is not used at all.

The first inner operator must accept an ExampleSet while the second must accept an ExampleSet and the output of the first (which in most cases is a Model) and must produce a PerformanceVector.

5.9.18 ItemDistributionEvaluator

Group: Validation.Performance.Clustering

Required input:

- FlatClusterModel

Generated output:

- PerformanceVector

Parameters:

- ◦ **keep_flat_cluster_model:** Indicates if this input object should also be returned as output. (boolean; default: true)
- **measure:** the item distribution measure to apply

Values:

- **applycount:** The number of times the operator was applied.
- **item_distribution:** The distribution of items over clusters.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Delivers a performance of a cluster model based on the distribution of items.

Description: Evaluates flat cluster models on how well the items are distributed over the clusters.

5.9.19 IteratingPerformanceAverage

Group: Validation.Other

Generated output:

- PerformanceVector

Parameters:

- ◦ **iterations:** The number of iterations. (integer; 1-+∞; default: 10)
- ◦ **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.

- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **time:** The time elapsed since this operator started.

Inner operators: The inner operators must deliver [PerformanceVector].

Short description: Iterates the inner operators and builds the average of the results.

Description: This operator chain performs the inner operators the given number of times. The inner operators must provide a PerformanceVector. These are averaged and returned as result.

5.9.20 MinMaxWrapper

Group: Validation.Performance

Required input:

- PerformanceVector

Generated output:

- PerformanceVector

Parameters:

○ **minimum_weight:** Defines the weight for the minimum fitness agains the average fitness (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Puts all input criteria into a min-max criterion which delivers the minimum instead of the average or arbitrary weighted combinations.

Description: Wraps a MinMaxCriterion around each performance criterion of type MeasuredPerformance. This criterion uses the minimum fitness achieved instead of the average fitness or arbitrary weightings of both. Please note that the average values stay the same and only the fitness values change.

5.9.21 Performance

Group: Validation

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance (main criterion).
- **time:** The time elapsed since this operator started.

Short description: This operator delivers as output a list of performance values automatically determined in order to fit the learning task type.

Description: In contrast to the other performance evaluation methods, this performance evaluator operator can be used for all types of learning tasks. It will automatically determine the learning task type and will calculate the most common criteria for this type. For more sophisticated performance calculations, you should check the operators RegressionPerformance (see section 5.9.23), ClassificationPerformance (see section 5.9.8), or BinominalClassificationPerformance (see section 5.9.5). You can even simply write your own performance measure and calculate it with the operator UserBasedPerformance (see section 5.9.28).

The operator expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of most commmon performance values for the provided learning task type (regression of (binominal) classification. If an input performance vector was already given, this is used for keeping the performance values.

5.9.22 PerformanceEvaluator

Group: Validation.Performance

Please use the operators BasicPerformance, RegressionPerformance, ClassificationPerformance, or BinominalClassificationPerformance instead.

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **main_criterion:** The criterion used for comparing performance vectors.
- **root_mean_squared_error:** Averaged root-mean-squared error (boolean; default: false)
- **absolute_error:** Average absolute deviation of the prediction from the actual value (boolean; default: false)
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value) (boolean; default: false)
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted. (boolean; default: false)
- **root_relative_squared_error:** Averaged root-relative-squared error (boolean; default: false)
- **squared_error:** Averaged squared error (boolean; default: false)
- **correlation:** Returns the correlation coefficient between the label and predicted label. (boolean; default: false)
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label. (boolean; default: false)
- **prediction_average:** This is not a real performance measure, but merely the average of the predicted labels. (boolean; default: false)
- **prediction_trend_accuracy:** Measures the average of times a regression prediction was able to correctly predict the trend of the regression. (boolean; default: false)
- **AUC:** The area under a ROC curve. Given example weights are also considered. Please note that the second class is considered to be positive. (boolean; default: false)

- **cross-entropy:** The cross-entropy of a classifier, defined as the sum over the logarithms of the true label's confidences divided by the number of examples (boolean; default: false)
- **margin:** The margin of a classifier, defined as the minimal confidence for the correct label. (boolean; default: false)
- **soft_margin_loss:** The average soft margin loss of a classifier, defined as the average of all 1 - confidences for the correct label. (boolean; default: false)
- **logistic_loss:** The logistic loss of a classifier, defined as the average of ln(1 + exp(- [confidence of the correct class])) (boolean; default: false)
- **accuracy:** Relative number of correctly classified examples (boolean; default: false)
- **classification_error:** Relative number of misclassified examples (boolean; default: false)
- **kappa:** The kappa statistics for the classification (boolean; default: false)
- **weighted_mean_recall:** The weighted mean of all per class recall measurements. (boolean; default: false)
- **weighted_mean_precision:** The weighted mean of all per class precision measurements. (boolean; default: false)
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho. (boolean; default: false)
- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b. (boolean; default: false)
- **skip_undefined_labels:** If set to true, examples with undefined labels are skipped. (boolean; default: true)
- **comparator_class:** Fully qualified classname of the PerformanceComparator implementation. (string)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)
- **class_weights:** The weights for all classes (first column: class name, second column: weight), empty: using 1 for all classes. (list)

Values:

- **AUC:** The area under a ROC curve. Given example weights are also considered. Please note that the second class is considered to be positive.
- **absolute_error:** Average absolute deviation of the prediction from the actual value
- **accuracy:** Relative number of correctly classified examples

- **applycount:** The number of times the operator was applied.
- **classification_error:** Relative number of misclassified examples
- **correlation:** Returns the correlation coefficient between the label and predicted label.
- **cross-entropy:** The cross-entropy of a classifier, defined as the sum over the logarithms of the true label's confidences divided by the number of examples
- **kappa:** The kappa statistics for the classification
- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b.
- **logistic_loss:** The logistic loss of a classifier, defined as the average of ln(1 + exp(- [confidence of the correct class]))
- **looptime:** The time elapsed since the current loop started.
- **margin:** The margin of a classifier, defined as the minimal confidence for the correct label.
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted.
- **performance:** The last performance (main criterion).
- **prediction_average:** This is not a real performance measure, but merely the average of the predicted labels.
- **prediction_trend_accuracy:** Measures the average of times a regression prediction was able to correctly predict the trend of the regression.
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
- **root_mean_squared_error:** Averaged root-mean-squared error
- **root_relative_squared_error:** Averaged root-relative-squared error
- **soft_margin_loss:** The average soft margin loss of a classifier, defined as the average of all 1 - confidences for the correct label.
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho.
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label.
- **squared_error:** Averaged squared error
- **time:** The time elapsed since this operator started.
- **weighted_mean_precision:** The weighted mean of all per class precision measurements.
- **weighted_mean_recall:** The weighted mean of all per class recall measurements.

Short description: A performance evaluator delivers as output a list of performance values according to a list of performance criteria.

Description: A performance evaluator is an operator that expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of performance values according to a list of performance criteria that it calculates. If an input performance vector was already given, this is used for keeping the performance values.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a ProcessLogOperator using the same names. The main criterion is used for comparisons and need to be specified only for processes where performance vectors are compared, e.g. feature selection processes. If no other main criterion was selected the first criterion in the resulting performance vector will be assumed to be the main criterion.

The resulting performance vectors are usually compared with a standard performance comparator which only compares the fitness values of the main criterion. Other implementations than this simple comparator can be specified using the parameter *comparator_class*. This may for instance be useful if you want to compare performance vectors according to the weighted sum of the individual criteria. In order to implement your own comparator, simply subclass `PerformanceComparator`. Please note that for true multi-objective optimization usually another selection scheme is used instead of simply replacing the performance comparator.

Additional user-defined implementations of PerformanceCriterion can be specified by using the parameter list *additional_performance_criteria*. Each key/value pair in this list must specify a fully qualified classname (as the key), and a string parameter (as value) that is passed to the constructor. Please make sure that the class files are in the classpath (this is the case if the implementations are supplied by a plugin) and that they implement a one-argument constructor taking a string parameter. It must also be ensured that these classes extend MeasuredPerformance since the PerformanceEvaluator operator will only support these criteria. Please note that only the first three user defined criteria can be used as logging value with names "user1", ... , "user3".

5.9.23 RegressionPerformance

Group: Validation

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **main_criterion:** The criterion used for comparing performance vectors.
- **root_mean_squared_error:** Averaged root-mean-squared error (boolean; default: false)
- **absolute_error:** Average absolute deviation of the prediction from the actual value (boolean; default: false)
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value) (boolean; default: false)
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted. (boolean; default: false)
- **root_relative_squared_error:** Averaged root-relative-squared error (boolean; default: false)
- **squared_error:** Averaged squared error (boolean; default: false)
- **correlation:** Returns the correlation coefficient between the label and predicted label. (boolean; default: false)
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label. (boolean; default: false)
- **prediction_average:** This is not a real performance measure, but merely the average of the predicted labels. (boolean; default: false)
- **prediction_trend_accuracy:** Measures the average of times a regression prediction was able to correctly predict the trend of the regression. (boolean; default: false)
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho. (boolean; default: false)
- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b. (boolean; default: false)
- **skip_undefined_labels:** If set to true, examples with undefined labels are skipped. (boolean; default: true)
- **comparator_class:** Fully qualified classname of the PerformanceComparator implementation. (string)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)

Values:

- **absolute_error:** Average absolute deviation of the prediction from the actual value
- **applycount:** The number of times the operator was applied.
- **correlation:** Returns the correlation coefficient between the label and predicted label.
- **kendall_tau:** The rank correlation between the actual and predicted labels, using Kendall's tau-b.
- **looptime:** The time elapsed since the current loop started.
- **normalized_absolute_error:** The absolute error divided by the error made if the average would have been predicted.
- **performance:** The last performance (main criterion).
- **prediction_average:** This is not a real performance measure, but merely the average of the predicted labels.
- **prediction_trend_accuracy:** Measures the average of times a regression prediction was able to correctly predict the trend of the regression.
- **relative_error:** Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
- **root_mean_squared_error:** Averaged root-mean-squared error
- **root_relative_squared_error:** Averaged root-relative-squared error
- **spearman_rho:** The rank correlation between the actual and predicted labels, using Spearman's rho.
- **squared_correlation:** Returns the squared correlation coefficient between the label and predicted label.
- **squared_error:** Averaged squared error
- **time:** The time elapsed since this operator started.

Short description: This operator delivers as output a list of performance values according to a list of selected performance criteria (for regression tasks).

Description: This performance evaluator operator should be used for regression tasks, i.e. in cases where the label attribute has a numerical value type. The operator expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of performance values according to a list of performance criteria that it calculates. If an input performance vector was already given, this is used for keeping the performance values.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a ProcessLogOperator using the same names.

The main criterion is used for comparisons and need to be specified only for processes where performance vectors are compared, e.g. feature selection or other meta optimization process setups. If no other main criterion was selected, the first criterion in the resulting performance vector will be assumed to be the main criterion.

The resulting performance vectors are usually compared with a standard performance comparator which only compares the fitness values of the main criterion. Other implementations than this simple comparator can be specified using the parameter *comparator_class*. This may for instance be useful if you want to compare performance vectors according to the weighted sum of the individual criteria. In order to implement your own comparator, simply subclass `PerformanceComparator`. Please note that for true multi-objective optimization usually another selection scheme is used instead of simply replacing the performance comparator.

5.9.24 SimpleValidation

Group: Validation

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- ○ **split_ratio:** Relative size of the training set (real; 0.0-1.0)
- ○ **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **looptime:** The time elapsed since the current loop started.

- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: A SimpleValidation randomly splits up the example set into a training and test set and evaluates the model.

Description: A `RandomSplitValidationChain` splits up the example set into a training and test set and evaluates the model. The first inner operator must accept an ExampleSet while the second must accept an ExampleSet and the output of the first (which is in most cases a Model) and must produce a PerformanceVector.

5.9.25 SimpleWrapperValidation

Group: Validation.Other

Required input:

- ExampleSet

Generated output:

- PerformanceVector
- AttributeWeights

Parameters:

- ○ **split_ratio:** Relative size of the training set (real; 0.0-1.0)
- ○ **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)

- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance (main criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Wrapper) must be able to handle [ExampleSet] and must deliver [AttributeWeights].
- Operator 2 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 3 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: A simple validation method to check the performance of a feature weighting or selection wrapper.

Description: This operator evaluates the performance of feature weighting algorithms including feature selection. The first inner operator is the weighting algorithm to be evaluated itself. It must return an attribute weights vector which is applied on the data. Then a new model is created using the second inner operator and a performance is retrieved using the third inner operator. This performance vector serves as a performance indicator for the actual algorithm.

This implementation is described for the SimpleValidation (see section 5.9.24).

5.9.26 SlidingWindowValidation

Group: Validation.Other

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- **training_window_width:** Number of examples in the window which is used for training (integer; 1-+∞; default: 100)
- **training_window_step_size:** Number of examples the window is moved after each iteration (-1: same as test window width) (integer; -1-+∞; default: -1)
- **test_window_width:** Number of examples which are used for testing (following after 'horizon' examples after the training window end) (integer; 1-+∞; default: 100)
- **horizon:** Increment from last training to first testing example (1 = next example). (integer; 1-+∞; default: 1)
- **cumulative_training:** Indicates if each training window should be added to the old one or should replace the old one. (boolean; default: false)
- **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].

- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: SlidingWindoValidation encapsulates sliding windows of training and tests in order to estimate the performance of a prediction operator.

Description: This is a special validation chain which can only be used for series predictions where the time points are encoded as examples. It uses a certain window of examples for training and uses another window (after horizon examples, i.e. time points) for testing. The window is moved across the example set and all performance measurements are averaged afterwards. The parameter "cumulative_training" indicates if all former examples should be used for training (instead of only the current window).

5.9.27 T-Test

Group: Validation.Significance

Required input:

- PerformanceVector

Generated output:

- PerformanceVector
- SignificanceTestResult

Parameters:

○ **alpha:** The probability threshold which determines if differences are considered as significant. (real; 0.0-1.0)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Performs a t-test to determine the probability for the null hypothesis 'the actual means are the same'.

Description: Determines if the null hypothesis (all actual mean values are the same) holds for the input performance vectors. This operator uses a simple (pairwise) t-test to determine the probability that the null hypothesis is wrong. Since a t-test can only be applied on two performance vectors this test will be applied to all possible pairs. The result is a significance matrix. However, pairwise t-test may introduce a larger type I error. It is recommended to apply an additional ANOVA test to determine if the null hypothesis is wrong at all.

5.9.28 UserBasedPerformance

Group: Validation.Performance

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **skip_undefined_labels:** If set to true, examples with undefined labels are skipped. (boolean; default: true)
- **comparator_class:** Fully qualified classname of the PerformanceComparator implementation. (string)
- **use_example_weights:** Indicated if example weights should be used for performance calculations if possible. (boolean; default: true)
- **main_criterion:** The criterion used for comparing performance vectors.
- **additional_performance_criteria:** List of classes that implement com.rapidminer..operator.performance.Pe (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance (main criterion).
- **time:** The time elapsed since this operator started.
- **user1:** The user defined performance criterion 0
- **user2:** The user defined performance criterion 1
- **user3:** The user defined performance criterion 2

Short description: This operator delivers as output a list of performance values according to a list of user defined performance criteria.

Description: This performance evaluator operator should be used for regression tasks, i.e. in cases where the label attribute has a numerical value type. The operator expects a test ExampleSet as input, whose elements have both true and predicted labels, and delivers as output a list of performance values according to a list of performance criteria that it calculates. If an input performance vector was already given, this is used for keeping the performance values.

Additional user-defined implementations of PerformanceCriterion can be specified by using the parameter list *additional_performance_criteria*. Each key/value pair in this list must specify a fully qualified classname (as the key), and a string parameter (as value) that is passed to the constructor. Please make sure that the class files are in the classpath (this is the case if the implementations are supplied by a plugin) and that they implement a one-argument constructor taking a string parameter. It must also be ensured that these classes extend MeasuredPerformance since the PerformanceEvaluator operator will only support these criteria. Please note that only the first three user defined criteria can be used as logging value with names "user1", ... , "user3".

The resulting performance vectors are usually compared with a standard performance comparator which only compares the fitness values of the main criterion. Other implementations than this simple comparator can be specified using the parameter *comparator_class*. This may for instance be useful if you want to compare performance vectors according to the weighted sum of the individual criteria. In order to implement your own comparator, simply subclass `PerformanceComparator`. Please note that for true multi-objective optimization usually another selection scheme is used instead of simply replacing the performance comparator.

5.9.29 WeightedBootstrappingValidation

Group: Validation.Other

Required input:	**Generated output:**
• ExampleSet	• PerformanceVector

Parameters:

- ○ **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- ○ **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)

- ○ **number_of_validations:** Number of subsets for the crossvalidation. (integer; 2-+∞; default: 10)
- ○ **sample_ratio:** This ratio of examples will be sampled (with replacement) in each iteration. (real; 0.0-+∞)
- ○ **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors. (boolean; default: true)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global). (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: This operator encapsulates an iterated weighted bootstrapping sampling with performance evaluation on the remaining examples.

Description: This validation operator performs several bootstrapped samplings (sampling with replacement) on the input set and trains a model on these samples. The remaining samples, i.e. those which were not sampled, build a test set on which the model is evaluated. This process is repeated for the specified number of iterations after which the average performance is calculated.

The basic setup is the same as for the usual cross validation operator. The first inner operator must provide a model and the second a performance vector. Please note that this operator does not regard example weights, i.e. weights specified in a weight column.

5.9.30 WeightedPerformanceCreator

Group: Validation.Performance

Required input:

- PerformanceVector

Generated output:

- PerformanceVector

Parameters:

- ○ **default_weight:** The default weight for all criteria not specified in the list 'criteria_weights'. (real; 0.0-$+\infty$)
- ○ **criteria_weights:** The weights for several performance criteria. Criteria weights not defined in this list are set to 'default_weight'. (list)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Returns a performance vector containing the weighted fitness value of the input criteria.

Description: Returns a performance vector containing the weighted fitness value of the input criteria.

5.9.31 WrapperXValidation

Group: Validation

Required input:

- ExampleSet

Generated output:

- PerformanceVector
- AttributeWeights

Parameters:

- ○ **number_of_validations:** Number of subsets for the crossvalidation (integer; 2-+∞; default: 10)
- ○ **leave_one_out:** Set the number of validations to the number of examples. If set to true, number_of_validations is ignored (boolean; default: false)
- ○ **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance (main criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Wrapper) must be able to handle [ExampleSet] and must deliver [AttributeWeights].
- Operator 2 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 3 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: Encapsulates a cross-validation to evaluate a feature weighting or selection method (wrapper).

Description: This operator evaluates the performance of feature weighting and selection algorithms. The first inner operator is the algorithm to be evaluated itself. It must return an attribute weights vector which is applied on the test data. This fold is used to create a new model using the second inner operator and retrieve a performance vector using the third inner operator. This performance vector serves as a performance indicator for the actual algorithm. This implementation of a MethodValidationChain works similar to the XValidation (see section 5.9.32).

5.9.32 XValidation

Group: Validation

Required input:

- ExampleSet

Generated output:

- PerformanceVector

Parameters:

- **keep_example_set:** Indicates if this input object should also be returned as output. (boolean; default: false)
- **create_complete_model:** Indicates if a model of the complete data set should be additionally build after estimation. (boolean; default: false)
- **average_performances_only:** Indicates if only performance vectors should be averaged or all types of averagable result vectors (boolean; default: true)
- **leave_one_out:** Set the number of validations to the number of examples. If set to true, number_of_validations is ignored (boolean; default: false)
- **number_of_validations:** Number of subsets for the crossvalidation. (integer; 2-+∞; default: 10)
- **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)

Values:

- **applycount:** The number of times the operator was applied.
- **deviation:** The standard deviation of the last performance (main criterion).

- **iteration:** The number of the current iteration.
- **looptime:** The time elapsed since the current loop started.
- **performance:** The last performance average (main criterion).
- **performance1:** The last performance average (first criterion).
- **performance2:** The last performance average (second criterion).
- **performance3:** The last performance average (third criterion).
- **time:** The time elapsed since this operator started.
- **variance:** The variance of the last performance (main criterion).

Inner operators:

- Operator 1 (Training) must be able to handle [ExampleSet] and must deliver [Model].
- Operator 2 (Testing) must be able to handle [ExampleSet, Model] and must deliver [PerformanceVector].

Short description: XValidation encapsulates a cross-validation in order to estimate the performance of a learning operator.

Description: `XValidation` encapsulates a cross-validation process. The example set S is split up into *number_of_validations* subsets S_i. The inner operators are applied *number_of_validations* times using S_i as the test set (input of the second inner operator) and $S\backslash S_i$ training set (input of the first inner operator).

The first inner operator must accept an ExampleSet while the second must accept an ExampleSet and the output of the first (which is in most cases a Model) and must produce a PerformanceVector.

Like other validation schemes the RapidMiner cross validation can use several types of sampling for building the subsets. Linear sampling simply divides the example set into partitions without changing the order of the examples. Shuffled sampling build random subsets from the data. Stratifed sampling builds random subsets and ensures that the class distribution in the subsets is the same as in the whole example set.

5.10 Visualization

These operators provide visualization techniques for data and other RapidMiner objects. Visualization is probably the most important tool for getting insight in your data and the nature of underlying patterns.

5.10.1 CorrelationMatrix

Group: Visualization

Required input:

- ExampleSet

Generated output:

- ExampleSet
- SymmetricalMatrix

Parameters:

- ◦ **create_weights:** Indicates if attribute weights based on correlation should be calculated or if the complete matrix should be returned. (boolean; default: false)
- ◦ **normalize_weights:** Indicates if the attributes weights should be normalized. (boolean; default: true)
- ◦ **squared_correlation:** Indicates if the squared correlation should be calculated. (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Determines the correlation between all attributes and can produce a weight vector based on correlations.

Description: This operator calculates the correlation matrix between all attributes of the input example set. Furthermore, attribute weights based on the correlations can be returned. This allows the deselection of highly correlated attributes with the help of an AttributeWeightSelection (see section 5.8.11) operator. If no weights should be created, this operator produces simply a correlation matrix which up to now cannot be used by other operators but can be displayed to the user in the result tab.

Please note that this simple implementation performs a data scan for each attribute combination and might therefore take some time for non-memory example tables.

5.10.2 CovarianceMatrix

Group: Visualization

Required input:

- ExampleSet

Generated output:

- ExampleSet
- SymmetricalMatrix

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Determines the covariance between all attributes.

Description: This operator calculates the covariances between all attributes of the input example set and returns a covariance matrix object which can be visualized.

5.10.3 DataStatistics

Group: Visualization

Required input:

- ExampleSet

Generated output:

- ExampleSet
- DataStatistics

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Calculates some simple data statistics usually displayed by the GUI (only necessary for command line processes).

Description: This operators calculates some very simple statistics about the given example set. These are the ranges of the attributes and the average or mode values for numerical or nominal attributes respectively. These informations are automatically calculated and displayed by the graphical user interface of RapidMiner. Since they cannot be displayed with the command line version of RapidMiner this operator can be used as a workaround in cases where the graphical user interface cannot be used.

5.10.4 ExampleVisualizer

Group: Visualization

Required input:	**Generated output:**
• ExampleSet	• ExampleSet

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Allows the visualization of examples (attribute values) in the plot view of an example set (double click on data point).

Description: Remembers the given example set and uses the ids provided by this set for the query for the corresponding example and the creation of a generic example visualizer. This visualizer simply displays the attribute values of the example. Adding this operator is often necessary to enable the visualization of single examples in the provided plotter components.

5.10.5 ExperimentLog

Group: Visualization

Please use the operator 'ProcessLog' instead.

Parameters:

- **filename:** File to save the data to. (filename)
- **log:** List of key value pairs where the key is the column name and the value specifies the process value to log. (list)
- **persistent:** Indicates if results should be written to file immediately (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Saves almost arbitrary data to a log file and create statistics for online plotting of values/parameters provided by operators.

Description: This operator records almost arbitrary data. It can written to a file which can be read e.g. by gnuplot. Alternatively, the collected data can be plotted by the GUI. This is even possible during process runtime (i.e. online plotting).

Parameters in the list `log` are interpreted as follows: The *key* gives the name for the column name (e.g. for use in the plotter). The *value* specifies where to retrieve the value from. This is best explained by an example:

- If the value is `operator.Evaluator.value.absolute`, the ProcessLogOperator looks up the operator with the name `Evaluator`. If this operator is a PERFORMANCEEVALUATOR (see section 5.9.22), it has a value named *absolute* which gives the absolute error of the last evaluation. This value is queried by the ProcessLogOperator
- If the value is `operator.SVMLearner.parameter.C`, the ProcessLogOperator looks up the parameter *C* of the operator named `SVMLearner`.

Each time the ProcessLogOperator is applied, all the values and parameters specified by the list *log* are collected and stored in a data row. When the process finishes, the operator writes the collected data rows to a file (if specified). In GUI mode, 2D or 3D plots are automatically generated and displayed in the result viewer.

Please refer to section 4.3 for an example application.

5.10.6 LiftChart

Group: Visualization

Required input:

- ExampleSet
- Model

Generated output:

- ExampleSet
- Model

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates a lift chart for the given binominal model and input data set.

Description: This operator creates a Lift chart for the given example set and model. The model will be applied on the example set and a lift chart will be produced afterwards.

Please note that a predicted label of the given example set will be removed during the application of this operator.

5.10.7 ModelVisualizer

Group: Visualization

Required input:

- ExampleSet
- Model

Generated output:

- ExampleSet
- Model

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates a SOM plot (transforming arbitrary number of dimensions to two) of the given data set and colorizes the landscape with the predictions of the given model.

Description: This class provides an operator for the visualization of arbitrary models with help of the dimensionality reduction via a SOM of both the data set and the given model.

5.10.8 ProcessLog

Group: Visualization

Parameters:

- **filename:** File to save the data to. (filename)
- **log:** List of key value pairs where the key is the column name and the value specifies the process value to log. (list)
- **persistent:** Indicates if results should be written to file immediately (boolean; default: false)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Saves almost arbitrary data to a log file and create statistics for online plotting of values/parameters provided by operators.

Description: This operator records almost arbitrary data. It can written to a file which can be read e.g. by gnuplot. Alternatively, the collected data can be plotted by the GUI. This is even possible during process runtime (i.e. online plotting).

Parameters in the list `log` are interpreted as follows: The *key* gives the name for the column name (e.g. for use in the plotter). The *value* specifies where to retrieve the value from. This is best explained by an example:

- If the value is `operator.Evaluator.value.absolute`, the ProcessLogOperator looks up the operator with the name `Evaluator`. If this operator is a PerformanceEvaluator (see section 5.9.22), it has a value named *absolute* which gives the absolute error of the last evaluation. This value is queried by the ProcessLogOperator

- If the value is `operator.SVMLearner.parameter.C`, the ProcessLogOperator looks up the parameter C of the operator named `SVMLearner`.

Each time the ProcessLogOperator is applied, all the values and parameters specified by the list *log* are collected and stored in a data row. When the process finishes, the operator writes the collected data rows to a file (if specified). In GUI mode, 2D or 3D plots are automatically generated and displayed in the result viewer.

Please refer to section 4.3 for an example application.

5.10.9 ROCChart

Group: Visualization

Required input:

- ExampleSet
- Model

Generated output:

- ExampleSet
- Model

Parameters:

- ◦ **use_example_weights:** Indicates if example weights should be used for calculations (use 1 as weights for each example otherwise). (boolean; default: true)
- ◦ **use_model:** If checked a given model will be applied for generating ROC-Chart. If not the examples set must have a predicted label. (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Generates a ROC chart for the given binominal model and input data set.

Description: This operator creates a ROC chart for the given example set and model. The model will be applied on the example set and a ROC chart will be produced afterwards. If you are interested in finding an optimal threshold, the operator THRESHOLDFINDER (see section 5.7.5) should be used. If you are interested in the performance criterion Area-Under-Curve (AUC) the usual PERFORMANCEEVALUATOR (see section 5.9.22) can be used. This operator just presents a ROC plot for a given model and data set.

Please note that a predicted label of the given example set will be removed during the application of this operator.

5.10.10 ROCComparator

Group: Visualization

Required input:

- ExampleSet

Generated output:

- ExampleSet
- ROCComparison

Parameters:

- ○ **number_of_folds:** The number of folds used for a cross validation evaluation (-1: use simple split ratio). (integer; -1-+∞; default: 10)
- ○ **split_ratio:** Relative size of the training set (real; 0.0-1.0)
- ○ **sampling_type:** Defines the sampling type of the cross validation (linear = consecutive subsets, shuffled = random subsets, stratified = random subsets with class distribution kept constant)
- ○ **local_random_seed:** Use the given random seed instead of global random numbers (-1: use global) (integer; -1-+∞; default: -1)
- ○ **use_example_weights:** Indicates if example weights should be regarded (use weight 1 for each example otherwise). (boolean; default: true)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Inner operators: Each inner operator must be able to handle [ExampleSet] and must deliver [Model].

Short description: Generates a ROC chart for the models created by each of the inner learners and plot all charts in the same plotter.

Description: This operator uses its inner operators (each of those must produce a model) and calculates the ROC curve for each of them. All ROC curves together are plotted in the same plotter. The comparison is based on the average values of a k-fold cross validation. Alternatively, this operator can use an internal split into a test and a training set from the given data set.

Please note that a former predicted label of the given example set will be removed during the application of this operator.

5.10.11 TransitionMatrix

Group: Visualization

Required input:

- ExampleSet

Generated output:

- Matrix

Parameters:

- **attribute:** the name of a nominal attribute for which the transition matrix should be created. (string)
- ○ **group_attribute:** provides a groups for examples, transitions between groups are not considered. (string)

Values:

- **applycount:** The number of times the operator was applied.
- **looptime:** The time elapsed since the current loop started.
- **time:** The time elapsed since this operator started.

Short description: Creates a transition matrix for nominal values.

Description: Creates a transition matrix for a given nominal attribute. An entry v(x,y) in the matrix denotes the conditional probability that value y occurs, after value x occurred.

Chapter 6

Extending RapidMiner

The core RapidMiner operators provide solutions for a large amount of usual data mining applications. However, it is quite simple to write your own operators in order to extend RapidMiner. The platform provides the data management, the nesting of operators, and the handling of optional and mandatory parameters.

This chapter describes how to implement your own RapidMiner operator in Java. At least you should know the basic concepts of this language to understand what we are doing here. All necessary information about the RapidMiner classes can be found in the RapidMiner API documentation which should be available on the RapidMiner homepage `http://www.rapidminer.com/`.

6.1 Project structure

In order to compile your own operators against RapidMiner, you must add the file `rapidminer.jar` and eventually some other `jar` files in the `lib` directory of RapidMiner to your `CLASSPATH`. If you downloaded the source version of RapidMiner, you should add the `build` directory (instead of `rapidminer.jar`) to the `CLASSPATH`.

Using the source version of RapidMiner has the advantage that you can use the `ant` buildfile. `ant` is a make-like open source build tool for Java you can download from `http://ant.apache.org`. The buildfile defines several useful targets, among which is `build` which, as one may easily guess, compiles all sources.

An Emacs JDE project file is also part of the source distribution. JDE is the Java Development Environment for Emacs and turns Emacs into a Java IDE. It can be downloaded from `http://jdee.sunsite.dk`. On Unix platforms, Emacs is a widespread editor but it also runs on Windows. It can be downloaded from http://www.gnu.org/software/emacs.

There are also project files for Eclipse in the project folders. Eclipse is a powerful open-source IDE for Java which can be downloaded at http://www.eclipse.org. It is also very easy to integrate the latest CVS version into Eclipse which is described in detail at our web page[1].

6.2 Operator skeleton

The first step to do when implementing a new operator is to decide which class must be extended. If your operator simply performs some action on its input and delivers some output it should extend the class

`com.rapidminer.operator.Operator`.

If the operator shall be able to contain inner operators, it must inherit from

`com.rapidminer.operator.OperatorChain`,

which itself extends `Operator`. Please refer to the API documentation if there is a more specific subclass of `Operator` which may serve your purpose. If your operator shall be a learning scheme it might be useful to implement

`com.rapidminer.operator.learner.Learner`

or extend

`com.rapidminer.operator.learner.AbstractLearner`

though it does not need to. Similar interfaces and abstract operator classes exists for other purposes too.

Now, there are some important things to specify about your operator. These specifications will automatically be used for sanity checks, parameter value checks, creation of GUI elements, and documentation. The following methods must or should be overridden (if the operator does not inherit from `Operator` directly, this may not be necessary):

1. One argument constructor: this constructor gets an object of the class `OperatorDescription` which must be passed to the superclass by invoking `super(description)`.

[1] http://www.rapidminer.com/

2. Class[] getInputClasses(): Specifies the number and type of objects that are expected as input classes. Only classes that implement com.rapidminer.operator.IOObject may be passed between operators. Typical objects passed between operators include example sets, models, and performance vectors (see section 6.3.4).

3. Class[] getOutputClasses(): Specifies the number and type of objects that are generated by this operator as output classes.

4. List<ParameterType> getParameterTypes(): Specifies the names and types of parameters that may be queried by this operator. Please make sure to add the parameter types to a java.util.List retrieved by a call to super.getParameterTypes(). The usage of subclasses of ParameterType for this purpose is described in sections 6.3.1 and the retrieval of parameter values is described in section 6.3.2.

5. IOObject[] apply(): This is the main method that is invoked whenever the operator should perform its work. This method can query parameter values, input objects, and maybe call the apply methods of inner operators (in case of an operator chain). It returns an array of IOObjects as a result. Please note that this method might throw an exception of type OperatorException.

If your operator extends OperatorChain is must additionally implement the following methods:

1. int getMaxNumberOfInnerOperators() and getMinNumberOfInnerOperators(): these methods specify the maximum and minimum number of inner operators allowed for the operator chain. Return 0 and Integer.MAX_VALUE, respectively for an unlimited number of inner operators.

2. InnerOperatorCondition getInnerOperatorCondition(): Operator chains have to implement this method. The delivered condition is used to perform checks if all inner operators can handle their input and deliver the necessary output. Several implementations for InnerOperatorCondition are available, please refer to the API documentation for details.

Please have a look at the simple operator skeleton showed in figure 6.1. As described above, the operator skeleton extends the class Operator.

The methods getInputClasses() and getInputClasses() do not declare any input and output objects yet and so does the method getParameterTypes(), which simply returns the parameters declared by its superclass. According to these declarations, the apply() method does nothing, but quietly returns an

```
package my.new.operators;

import com.rapidminer.operator.Operator;
import com.rapidminer.operator.OperatorDescription;
import com.rapidminer.operator.OperatorException;
import com.rapidminer.operator.IOObject;
import com.rapidminer.parameter.ParameterType;

import java.util.List;

public class OperatorSkeleton extends Operator {

    /** Must pass the given object to the superclass. */
    public OperatorSkeleton(OperatorDescription description) {
        super(description);
    }

    /** Perform the operators action here. */
    public IOObject[] apply() throws OperatorException {
        // describe the core function of this operator
        return new IOObject[0];
    }

    /** Add your parameters to the list. */
    public List<ParameterType> getParameterTypes() {
        List<ParameterType> types = super.getParameterTypes();
        // add your parameter types here
        return types;
    }

    /** Return the required input classes. */
    public Class[] getInputClasses() { return new Class[0]; }

    /** Return the delivered output classes. */
    public Class[] getOutputClasses() { return new Class[0]; }
}
```

Figure 6.1: Operator skeleton

empty array. The following sections describe, how you can fill these methods with code.

Note: Since version 3.0 of RapidMiner each operator must have an one-argument constructor which must at least pass the given operator description object to the superclass constructor. Please note that during operator construction the method getParameterTypes() will be invoked and must be fully functional, i. e. not depending on uninitialized fields of the operator.

Finally, if your operator is implemented and you want to use it from RapidMiner, you must declare the new operator class by adding a short entry to an XML file. This is described in section 6.8.

6.3 Useful methods for operator design

Before we discuss an easy example for a self-written operator, the required methods are described in detail. These methods enable you to declare a parameter, query a parameter value, adding a `Value` which can be plotted by the PocessLog operator and handling the in- and output of your operator.

6.3.1 Defining parameters

As we have seen above, the method `getParameterTypes()` can be used to add parameters to your operator. Each parameter is described by a `ParameterType` object, i.e. an object which contains the name, a small description, and in case of a numerical parameter the range and default value of this parameter. A new parameter type has to extend the class

```
com.rapidminer.parameter.ParameterType
```

In RapidMiner, for each simple data type a parameter type is provided, e.g. for boolean values the type `ParameterTypeBoolean` or `ParameterTypeInteger` for integers. Table 6.1 shows all possible parameter types. Please refer to the API documentation for details on constructing the different parameter types.

Since the method `getParameterTypes()` returns a list of `ParameterTypes`, your operator should first invoke `super.getParameterTypes()` and add its parameter types to the list which is returned by this method. In this way it is ensured that the parameters of super classes can also be set by the user. Figure 6.2 shows how a new integer parameter is added.

As you can see you create a new `ParameterTypeInteger` and add it to the list. The first argument of the constructor is the name of the parameter which will be used in the XML description files or in the GUI parameter table. The

```
public List<ParameterType> getParameterTypes() {
    List<ParameterType> types = super.getParameterTypes();
    types.add(new ParameterTypeInteger("number",
                                       "This␣is␣important.",
                                       1, 10, 5));
    return types;
}
```

Figure 6.2: Adding a parameter

second argument is a short description. This is used as tool tip text when the mouse pointer stays on the parameter in the GUI for some seconds. For numerical parameter types a range can be specified. The first number defines the minimum value of this parameter, the second the maximum value. The last number is the default value which is used when the user do not change this parameter in the process setup.

Not every operator needs parameters. This is the reason why the method `getParameterTypes()` is not abstract in Operator. You can simply ommit the implementation of this method if your operator does not use any parameters. However, you should notice that the method `getParameterTypes()` is invoked by the super-constructor. You should therefore not use global variables which are not initialized yet.

6.3.2 Getting parameters

Now you can add different parameter types to your operator. For each type

ParameterTypeXXX

a method `getParameterAsXXX()` is provided by the superclass `Operator` unless another method is described in table 6.1. All these methods return an appropriate Java type, e.g. `double` for `getParameterAsDouble()`. Table 6.2 shows the parameter getter methods of the class `Operator` in detail.

The methods `getParameterAsXXX()` will throw an UndefinedParameterError if the user has not defined a value for a non-optional parameter without default value. Since this Error extends UserError which extends OperatorException you should just throw these error out of your apply method. A proper GUI message will be automatically created.

The List returned by `getParameterList(String)` contains Object arrays of length 2. The first object is a key (String) and the second the parameter value object, e.g. a Double for ParameterTypeDouble.

Type	**Description**
ParameterTypeBoolean	A boolean parameter. The defined value can be queried by getParameterAsBoolean(key).
ParameterTypeCategory	A category parameter which allows defined strings. The index of the chosen string can be queried by getParameterAsInt(key).
ParameterTypeColor	A parameter for colors. This is currently only used for user interface settings. The specified color can be queried by getParameterAsColor(key).
ParameterTypeDirectory	A directory. The path to the chosen directory can be queried by getParameterAsString(key).
ParameterTypeDouble	A real valued parameter. The defined value can be queried by getParameterAsDouble(key).
ParameterTypeFile	A file. The path to the chosen file can be queried by getParameterAsString(key).
ParameterTypeInt	A integer parameter. The defined value can be queried by getParameterAsInt(key).
ParameterTypeList	A list of parameters of another parameter type. The defined list can be queried by getParameterList(key).
ParameterTypePassword	A password parameter. Passwords are masked with * in the GUI and queried by the system if the user has not specified the password in the process setup. The defined string can be queried by getParameterAsString(key).
ParameterTypeString	A simple string parameter. The defined value can be queried by getParameterAsString(key).
ParameterTypeStringCategory	A category parameter which allows defined strings. Additionally the user can specify another string. The chosen string can be queried by getParameterAsString(key).

Table 6.1: These parameter types can be added to your operator. Please refer to the API documentation for details on creation.

Method	**Description**
`getParameterAsBoolean(String key)`	Returns a parameter and casts it to boolean.
`getParameterAsColor(String key)`	Returns a parameter and casts it to Java Color.
`getParameterAsDouble(String key)`	Returns a parameter and casts it to double.
`getParameterAsFile(String key)`	Returns a parameter and casts it to a Java File.
`getParameterAsInt(String key)`	Returns a parameter and casts it to int.
`getParameterAsString(String key)`	Returns a parameter and casts it to String.
`getParameterList(String key)`	Returns a parameter and casts it to a Java List.

Table 6.2: Methods for obtaining parameters from `Operator`

6.3.3 Providing `Values` for logging

As you can see, the operator skeleton contains a one-argument constructor which must pass the given description object to the super-constructor. This is necessary for the automatic operator creation with help of factory methods (see section 7). These constructors can also be used to declare `Values` which can be queried by an ProcessLog operator (see 5.10.8). Each value you want to add must extend

`com.rapidminer.operator.Value`

and override the abstract method `getValue()`. Figure 6.3 shows how you can add some values in the constructor of your operator. Note that usually non-static inner classes are used to extend `Value`. These classes have access to private fields of the operator and may, e.g. return the number of the current run, the current performance or similar values.

Note: Please make sure that the only purpose of an operator's constructor should be to add values and *not* querying parameters or perform other actions. Since the operator description and parameters will be initialized after operator construction these type of actions will probably not work and might cause exceptions.

6.3.4 Input and output

As default, operators consume their input by using it. This is often a useful behavior, especially in complex process definitions. For example, a learning

```
public MyOperator(OperatorDescription description) {
  // invoke super-constructor
  super( description );
  // add values for process logging
  addValue(new Value("number", "The␣current␣number.") {
      public double getValue() {
        return currentNumber;
      }
    });
  addValue(new Value("performance", "The␣best␣performance.") {
      public double getValue() {
        return bestPerformance;
      }
    });
}
```

Figure 6.3: Adding `Values` to your Operator which can be queried by ProcessLog.

operator consumes an example set to produce a model and so does a cross validation to produce a performance value of the learning method. To receive the input `IOObject` of a certain class simply use

```
<T extends IOObject> T getInput(Class<T> class)
```

This method delivers the first object of the desired class which is in the input of this operator. By using generics it is already ensured that the delivered object has the correct type and no cast is necessary. The delivered object is consumed afterwards and thus is removed from input. If the operator alters this object, it should return the altered object as output again. Therefore, you have to add the object to the output array which is delivered by the `apply()` method of the operator. You also have to declare it in `getOutputClasses()`. All input objects which are not used by your operator will be automatically passed to the next operators.

Note: In versions before 3.4 it was necessary to cast the delivered object to the correct type. This cast is no longer necessary.

In some cases it would be useful if the user can define if the input object should be consumed or not. For example, a validation chain like cross validation should estimate the performance but should also be able to return the example set which is then used to learn the overall model. Operators can change the default behavior for input consumation and a parameter will be automatically defined and queried. The default behavior is defined in the method `getInputDescription(Class cls)` of operator and should be overriden in these cases. Please note that input objects with a changed input description must not be defined in `getOutputClasses()` and must not be returned at the end of apply. Both is automatically done with respect to the value of the automatically created parameter. Figure 6.4 shows how this could be done. Please refer to the Javadoc comments of this method for further explanations.

```
import com.rapidminer.example.ExampleSet;
import com.rapidminer.operator. InputDescription ;

 ...

/** Change the default behavior for input handling. */
public InputDescription getInputDescription(Class cls) {
    // returns a changed input description for example sets
    if (ExampleSet.class.isAssignableFrom(cls)) {
        // consume default: false, create parameter: true
        return new InputDescription(cls, false, true);
    } else {
        // other input types should be handled by super class
        return super.getInputDescription(cls);
    }
}

 ...
```

Figure 6.4: Changing the input handling behavior of your operator. In this case, example sets should be consumed per default but a parameter named `keep_example_set` will be automatically defined.

6.3.5 Generic Operators

Sometimes a generic operator class should be implemented which should provide more than one operator. In this case several operators can be declared to RapidMiner (see section 6.8) with the same class but different names. The subtype or name of an operator can be requested by `getOperatorClassName()` which is a method of operator. Although this is very similar to define the operator type with help of a parameter, subtypes can be used in more subtile way: they can already be used to define the parameters and therefore each subtype of an operator class may have other parameters. This feature is used to provide a normal RapidMiner operator with different parameter types for each Weka operator with the help of only one (short) class. Please check the source code of the Weka learners for an example of a generic operator.

6.4 Example: Implementation of a simple operator

After these technical preliminary remarks we set an example which performs a very elementary task: It should write all examples of an ExampleSet into a file.

First we consider that all we need as input for this operator is an example set. Since we will not manipulate it, we deliver the same example set as output. Therefore the methods `getInputClasses()` and `getOutputClasses()` will only contain one class: `com.rapidminer.example.ExampleSet`. If ExampleSet is not contained in the output of the former operator, your operator can not work and RapidMiner will terminate at the beginning of the process.

Your operator uses one parameter: A file where it should store the examples. Therefore a `ParameterTypeFile` is added to the parameter list. The third argument in the constructor indicates that this parameter is mandatory. Let us presume that your own operators are in the package `my.new.operators`. Please have a look at the operator in figure 6.5, then we will explain the apply() function in detail.

The first line in `apply()` fetches the name of the file to write the example set to. This method returns the value of the parameter "example_set_file", if it is declared in the operator section of the process configuration file. Since this parameter is mandatory the process ends immediately with an error message if this parameter is not given.

We need the input example set to iterate through the examples and write them to a file. We simply use the `getInput(ExampleSet.class)` method in order to get the desired input object (the example set).

Note: Please note that a cast to `ExampleSet` is not necessary. For RapidMiner versions before 3.4 a cast to the actual type has to be performed.

Then a stream to the specified file is opened and an iterator over the examples is created. With this `Iterator<Example>` you can pass through the examples of an example set like the way you do with a `List` iterator. For each example the values are written into the file and afterwards the stream to the file is closed. Each operator can throw an `OperatorException` to the calling operator, which would be done if any exception occurred during writing the file. In this case the thrown exception is an `UserError` which is used because writing presumably fails because the file is not writable. We will discuss the error handling in section 6.4.2.

Note: In versions before 3.4 the iterator was called `ExampleReader`. Changing to the generic `Iterator<Example>` also allows for the nice new for-loop introduced in Java 5.0: `for (Example e :  exampleSet) ....`

The last thing to do is creating a new array of `IOObjects` which contains only the used ExampleSet since no additional output was produced. The next section describes the iterating through an example set in detail, then the exception concept will be explained.

```
package my.new.operators;

import com.rapidminer.example.*;
import com.rapidminer.operator.*;
import com.rapidminer.parameter.*;
import java.io.*;
import java.util.List;

public class ExampleSetWriter extends Operator {

  public ExampleSetWriter(OperatorDescription description) {
    super(description);
  }

  public IOObject[] apply() throws OperatorException {
    File file = getParameterAsFile("example_set_file");
    ExampleSet eSet = getInput(ExampleSet.class);
    try {
      PrintWriter out = new PrintWriter(new FileWriter(file));
      for (Example example : eSet) {
        out.println(example);
      }
      out.close();
    } catch (IOException e) {
      throw new UserError(this, 303, file, e.getMessage());
    }
    return new IOObject[] { eSet };
  }

  public List<ParameterType> getParameterTypes() {
    List<ParameterType> types = super.getParameterTypes();
    types.add(new ParameterTypeFile("example_set_file",
                                    "The␣file␣for␣the␣examples.",
                                    "txt", // default file extension
                                    false)); // non-optional
    return types;
  }

  public Class[] getInputClasses() {
    return new Class[] { ExampleSet.class };
  }

  public Class[] getOutputClasses() {
    return new Class[] { ExampleSet.class };
  }
}
```

Figure 6.5: Implementation of an example set writer

6.4.1 Iterating over an ExampleSet

RapidMiner is about data mining and one of the most frequent applications of data mining operators is to iterate over a set of examples. This can be done for preprocessing purposes, for learning, for applying a model to predict labels of examples, and for many other tasks. We have seen the mechanism in our example above and we will describe it below.

The way you iterate over an example set is very similar to the concept of iterators, e.g. in terms of `Lists`. The methods which are provided have the same signature as the methods of a Java `Iterator`. The first thing you have to do is to create such an iterator. The following code snippet will show you how:

```
Iterator <Example> reader = exampleSet.iterator();
while (reader.hasNext()) {
  Example example = reader.next();
  // ... do something with the example ...
}
```

Figure 6.6: Creating and using an example iterator

Assume `exampleSet` is a set of examples which we get from the input of the operator. First of all, an iterator is created and then we traverse through the examples in a loop. These iterators are backed up by different implementations of the interface `ExampleReader`. The classes ExampleSet, ExampleReader, and Example are provided within the package `com.rapidminer.example`. Please check the RapidMiner API documentation to see what else can be done with example sets and examples.

6.4.2 Log messages and throw Exceptions

If you write your operator, you should make some logging messages so that users can understand what your operator is currently doing. It is especially reasonable to log error messages as soon as possible. RapidMiner provides some methods to log the messages of an operator. We distinguish between *log messages* and *results*. Of course you can write your results into the normal log file specified in the process configuration file. But the intended way to announce results of the process is to use a ResultWriter (see section 5.3.38) which writes each currently available result residing in his input. For this purpose two classes exist, a class `LogService` and a class `ResultService`. The latter can be used by invoking the static method

```
logResult(String result)
```

or by simply using a ResultWriter as described above.

The class com.rapidminer.tools.LogService provides the static method

```
logMessage(String message, int verbosityLevel)
```

to log text messages. Possible verbosity levels are MINIMUM, IO, STATUS, INIT, WARNING, EXCEPTION, ERROR, FATAL, and MAXIMUM which are all public constant static fields of LogService. The verbosity levels IO and INIT should *not* be used by operator developers. Normal log messages should be logged with verbosity level STATUS.

6.4.3 Operator exceptions and user errors

The best way to abort the process because of an error is throwing an OperatorException. If the error occurred due to an unforseen situation, an instance of OperatorException should be thrown. To ease bug tracking, it is useful to pass RuntimeExceptions to the OperatorException constructor as the cause parameter. If the error was caused by wrong usage of the operator, e.g. missing files, or wrong parameter values, an instance of UserError should be thrown. An error code referencing an error message in the file resources/UserErrorMessages.properties must be passed to the constructor of a UserError. These messages are formatted using an instance of MessageFormat. Index numbers in curly braces are replaced by the arguments passed to the UserError constructor. Please refer to the API documentation for construction details.

6.5 Building operator chains

Now you can extend RapidMiner by writing operators which perform tasks on a given input and deliver the input or additional output to a surrounding operator. We have discussed the specifications to create the operator in such a way that it can be nested into other operators. But what we have not seen is the possibility to write your own operator chain, i.e. operators which contain inner operators to which input will be given and whose output is used by your operator. What transmutes a simple operator to an operator chain is the possibility to contain other inner operators.

The way you create an operator chain is straightforward: First your operator does not directly extend Operator any longer, but OperatorChain instead. Since OperatorChain extends Operator itself you still have to implement the methods discussed above.

The second thing you have to do is to declare how many inner operators your operator can cope with. Therefore every operator chain has to overwrite two abstract methods from `OperatorChain`:

```
int getMinNumberOfInnerOperators()
```

and

```
int getMaxNumberOfInnerOperators()
```

which returns the minimum and maximum number of inner operators. If these numbers are equal, your operator chain must include exactly this number of inner operators or RapidMiner will terminate at the beginning of an process.

There is another method which you have to implement:

```
InnerOperatorCondition getInnerOperatorCondition()
```

This method delivers a condition about the inner operators. This condition should ensure that the current process setup is appropriate and all inner operators can be executed. Several implementations of `InnerOperatorCondition` are available, please check the API for further details. We will explain both methods in detail when we discuss the example in section 6.6.

6.5.1 Using inner operators

You can simply use inner operators via the method

```
getOperator(int index)
```

which delivers the inner operator with the given index. You can invoke the `apply()` method of this operator yourself. The `apply()` method of the superclass automatically performs the actions of the inner operators. RapidMiner takes care of the sequential execution.

6.5.2 Additional input

But what if you want to add additional `IOObjects` to the input of an inner operator? A cross-validation operator for example, divides an example set into subsets and adds certain subsets to the input of a learning operator and others to the input of an operator chain which includes a model applier and a performance evaluator. In this case your operator has to consume the original `IOObject` and add others to the input of the inner operators.

In section 6.3.4 we have seen how an operator can get the input. This is consumed per default. If your operator should add a certain IOObject to the input of an inner operator it simply has to call the apply() method of the inner operator in a way like

```
apply(getInput().append(new IOObject[] { additionalIO }))
```

or

```
apply(new InputContainer(new IOObject[] { additionalIO })).
```

The method getInput() delivers the RapidMiner container which provides the input and output objects of the operators[2]. You can add an array of additional IOObjects using the append() method. The latter variant ignores the input for the current operator chain and produces a new input container for the child operators.

You should also use this method if you want to use the same IOObject as input for an inner operator several times, e.g. in a loop, or if you want to add more than one IOObject to the input of an inner operator.

6.5.3 Using output

Inner operators can produce output which your surrounding operator must handle. The call of the apply(IOContainer) method of an inner operator delivers a container like the one described above. You can get the IOObjects out of this container with some getter-methods provided by this class. Figure 6.7 shows the methods to append additional input for the inner operator and getting specified output from the result of the apply() method. The example set is split before into training and test set.

Mostly you do not need to do anything about adding additional input or getting the output and RapidMiner will manage the in- and output for your operator. Pay attention to the fact that you do not need to care about the learned model: RapidMiner copes with the learned model for your model applier.

6.6 Example 2: Implementation of an operator chain

The following example does not make much sense for data mining purposes, but it demonstrates the implementation of an operator chain. Figure 6.8 shows the complete code.

[2] com.rapidminer.operator.IOContainer

```
 [...]
// use first inner operator for learning on training set
Learner learner = (Learner)getOperator(0);
IOContainer container =
   learner .apply(getInput (). append(new IOObject[] {trainingSet }));

// apply model on test set
ModelApplier applier = (ModelApplier)getOperator(1);
container =
   applier .apply( container .append(new IOObject[] {testSet}));

// retrieve the example set with predictions
ExampleSet withPredictions =
   container . get(ExampleSet.class);
 [...]
```

Figure 6.7: In- and output of an inner operator

All methods inherited of `Operator` are implemented as described above. Since this operator chain uses no parameters, the method `getParameterTypes()` is not overwritten. This operator chain must have at least one inner operator, the maximum number of inner operators is the biggest integer which Java can handle. The method which returns the estimated number of steps of this operator chain makes use of a method of the superclass `OperatorChain`: `getNumberOfChildrenSteps()` returns the sum of all children's steps.

The purpose of this operator chain is described in the `apply()` method. This operator expects an example set as input and clones it before it uses the clone as input for each of the inner operators. The inner operators must produce a performance vector. These vectors are averaged and then returned.

The desired in- and output behaviours of inner operators must be described with a condition object returned by the method `getInnerOperatorCondition()`. In this example each inner operator should be able to handle an example set and deliver a performance vector.

6.7 Overview: the data core classes

It will be sufficient for many data mining purposes to iterate through the example set. However, in some cases one must perform some more complex changes of data or meta data. RapidMiner tries to hide the internal data transformations for typical data mining purposes. It uses a view mechanism to make a trade-off between efficient data mining data transformations and the usage of memory. In this section we discuss some basic concepts of the data handling in RapidMiner to support users who want to write more complex operators.

```
package my.new.operators;

import com.rapidminer.operator.*;
import com.rapidminer.operator.performance.*;
import com.rapidminer.example.*;

public class MyOperatorChain extends OperatorChain {

  public MyOperatorChain(OperatorDescription description) {
    super( description );
  }

  public IOObject[] apply() throws OperatorException {
    ExampleSet exampleSet = getInput(ExampleSet.class);
    ExampleSet clone = null;
    PerformanceVector result = new PerformanceVector();
    for (int i = 0; i < getNumberOfOperators(); i++) {
      clone = (ExampleSet)exampleSet.clone();
      IOContainer input = getInput().append(new IOObject[] { clone });
      IOContainer applyResult = getOperator(i).apply(input);
      PerformanceVector vector =
          applyResult.getInput(PerformanceVector.class);
      result.buildAverages(vector);
    }
    return new IOObject[] { result };
  }

  public Class[] getInputClasses() {
    return new Class[] { ExampleSet.class };
  }

  public Class[] getOutputClasses() {
    return new Class[] { PerformanceVector.class };
  }

  public InnerOperatorCondition getInnerOperatorCondition() {
    return new AllInnerOperatorCondition(new Class[] { ExampleSet.class},
                                         new Class[] { PerformanceVector.class} );
  }

  public int getMinNumberOfInnerOperators() { return 1; }
  public int getMaxNumberOfInnerOperators() { return Integer.MAX_VALUE; }
  public int getNumberOfSteps() { return super.getNumberOfChildrensSteps(); }
}
```

Figure 6.8: Example implementation of an operator chain.

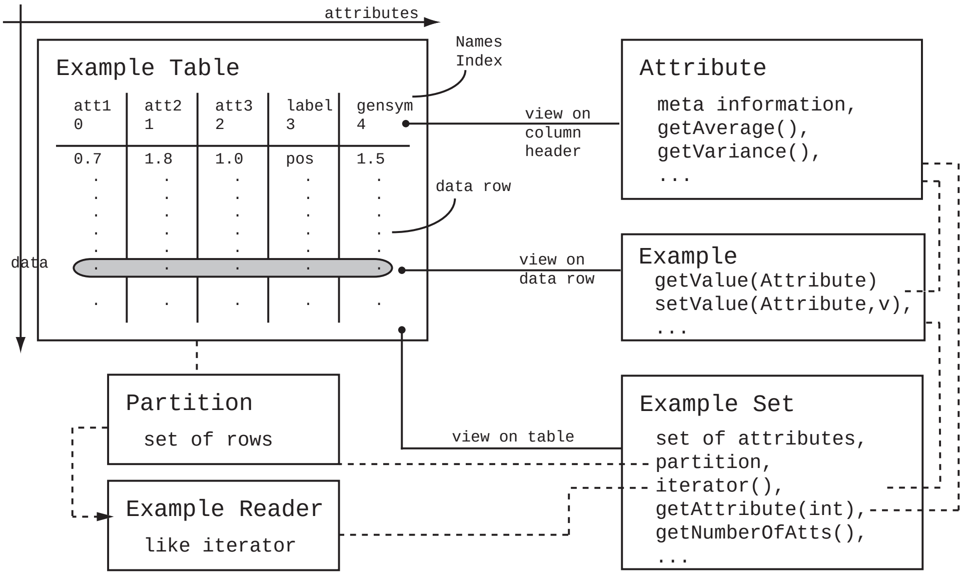


Figure 6.9: The main classes used for data handling in RapidMiner. The central class is ExampleTable which keeps all data loaded and generated during processes. However, it is almost never directly used by operators.

Figure 6.9 shows the main classes and interfaces which are used for data handling in RapidMiner. The class ExampleTable keeps all data which is loaded or generated during processes and stores it in one table. The columns are defined by Attributes, which are used for two purposes: managing meta data about table columns and referring to columns when one asks for the data in one cell. One might say, that Attribute is a view on the header of a column in the data table. Each row of the table is given by a DataRow. Although the table is the central class for data management, RapidMiner developers almost never use it directly. The other classes shown in Figure 6.9 are used to encapsulate the functionality and provide more convenient and secure ways to alter your data.

Since RapidMiner is a data mining environment we often work on data. This data is, as you know, given as ExampleSets. Example sets consist of a set of attributes and a partition. It is important to understand, that example sets not keeps the data itself. That means that you can copy an example set without copying the data. An example set is merely a view on the example table[3].

An important method of example sets is the creation of an example reader to iterate over the data. Depending whether the example set is splitted with a partition, a particular instance of an example reader is returned by the method iterator(). If only a partition of examples is used, the returned example reader

[3] This is the reason why the index of an attribute in the example set is not in general equal to the index in the example table. To ask for the value of an attribute the Attribute object should always be used instead of the index.

skips the deselected examples. Applying weights for attributes also requires a particular example reader, which construct examples based on the weights. RapidMiner provides interfaces and an adaptor concept for example sets to ensure the nestability of the operators and the used example sets and readers.

The last important class in the data management of RapidMiner is `Example` which is a view on a data row. Examples are constructed on the fly by the current example reader of the used example set. The current data row from the example table is used to back up the example and weights are applied if a weighting or selection should be applied. Since the indices of attributes in example sets and tables must not be equal, the query for an attribute's value of an example should always be performed with help of the attribute and not of it's index.

Several subclasses exist for example set, example table, and example reader. These subclasses provide different forms of data management (main memory, database,. . .) and views on your data. This concept ensures data transparency for all operators and the nesting of operators. In addition, new classes can be easily written to extend RapidMiner for special purposes.

6.8 Declaring your operators to RapidMiner

At this point you know all the tricks to write your own operators and the tool kit which is provided by RapidMiner for this purpose. The last thing you have to do is to declare your operator to RapidMiner. Every operator must comply with the following terms:

name A meaningful name to specify the operator in a process configuration file is required. The name must be unique.

class The fully classified classname of your operator (must be in your java `CLASSPATH` variable).

description A short description of your operator and its task.

deprecation A short description why your operator is deprecated and a short description of a workaround.

group A name of a group. This may be your own group or one of the predefined RapidMiner group names.

icon This is also optional but can be used to ease identification of the operator.

The definition of deprecation and icon are optional. If deprecation is omitted, the operator is simply not regarded as deprecated - which is pretty much the default. If icon is missing, the default icon for the operator group is used. To link these description parts to one another you have to specify them in a operator description file. Every entry holds for one operator and they are written like the ones in figure 6.10. We assume that you save these descriptions in a file named 'operators.xml'.

```
<operators>

  <!-- Your own operator factory -->
  <factory class = "my.new.operators.OperatorFactory" />

  <!-- Your own Operators -->
  <operator
    name        = "MyExampleSetWriter"
    class       = "my.new.operators.ExampleSetWriter"
    description = "Writes example set into file."
    group       = "MyOps"/>

  <operator
    name        = "MyPreprocessing"
    class       = "my.new.operators.GenericPreprocessing"
    description = "Best preprocessing for my purpose."
    deprecation = "Please use the default preprocessing instead."
    group       = "MyOps"/>

</operators>
```

Figure 6.10: Declaring operators to RapidMiner

Additionally to simple operator entries you can specify one or more operator factory classes which must implement the interface

`com.rapidminer.tools.GenericOperatorFactory`.

This is especially useful if you want to provide more than one operator for each class by working with operator subtypes. This is the preferred way to add generic operators with one class but more than subtype or operator name.

In order to use your operators with RapidMiner you have to add them to your `CLASSPATH` variable. Then you can start RapidMiner with the option

```
-Drapidminer.operators.additional=path/to/your/operators.xml
```

Please edit your start scripts and add the parameter to the line which starts RapidMiner or start RapidMiner manually with a call like

```
java
-cp $RAPIDMINER_HOME/lib/rapidminer.jar:your/class/path
```

```
-Drapidminer.home=$RAPIDMINER_HOME
-Drapidminer.operators.additional=path/to/your/operators.xml
com.rapidminer.gui.RapidMinerGUI
```

Your new operators should be available now and can be chosen in the GUI. More than one additional operator description file can be specified by making use of the system dependent path separator, for unix systems for example with

```
-Drapidminer.operators.additional=my_operators.xml:other_ops.xml
```

6.9 Packaging plugins

If you want to make your operators available for download, you can easily create plugins.

1. Compile your Java sources.
2. Create a file named `operators.xml` as described above
3. Create a jar archive using the `jar` command that comes with the JDK. The archive must contain your operator class files and all classes they depend on. The file `operators.xml` must go into the `META-INF` directory of the archive.
4. If desired, create a file named `ABOUT.NFO` and add it to the `META-INF` directory of the jar file.
5. If you use a `Manifest` file, the entries Implementation-Title, Implementation-Version, Implementation-Vendor, Implementation-URL, RapidMiner-Version, and Plugin-Dependencies will be evaluated by RapidMiner. RapidMiner-Version defines the minimum RapidMiner version which is needed for this plugin. Plugin-Dependencies must have the form
   plugin_name1 [plugin_version1] # ... # plugin_nameM [plugin_versionM]
6. You can include GUI icons for your operators within the jar file. If you set the `icon` attribute of the `<operator>` tag in the `operators.xml` file to "foo", RapidMiner will look for a file named `op_foo.gif` in the directory `com/rapidminer/resources/icons/groups/24` or in the directory `com/rapidminer/resources/icons/operators/24` of the jar file.
7. Copy the archive into `lib/plugins` directory. If you like, also put it on your website or send it to us. Since RapidMiner is licensed under the GNU General Public License you have to develop your plugins as open-source software and you have to make it available for the RapidMiner

community. If this is not possible for any reasons, e.g. because the development of the plugin was done for commercial purposes, you should contact us for a special commercial version of RapidMiner.

Hint: If your plugin depends on external libraries, you do not have to package these into one archive. You can reference the libraries using a `Class-Path` entry in your jar `Manifest` file. For information on installation of plugins, please refer to section 2.5.

6.10 Documentation

The operator reference chapter of the LaTeX RapidMiner tutorial is generated using the Javadoc class comments of the operator source code. Therefore some additional Javadoc tags can be used.

@rapidminer.xmlclass The classname given in the `operators.xml` if different from the classname.

@rapidminer.index For LaTeXoutput generate an index entry for the tutorial referring to the description of this operator.

@rapidminer.reference A BibTeX key that will be used to generate a HTML bibliography entry. Ignored for LaTeX output.

@rapidminer.cite Inline tag. For LaTeX output generate a citation. For HTML output, simply print the key.

@rapidminer.ref Inline tag. For LaTeX output generate a reference to a tutorial section, for HTML simply print the reference name.

@rapidminer.xmlinput The text of this tag must consist of three strings separated by a pipe symbol ("|"). The first string must be a filename, the second must be a label, and the third must be a caption for a figure. The file specified will be input both in HTML and LaTeX documentation.

Please refer to the API documentation of the class

com.rapidminer.docs.DocumentationGenerator

to learn how the documentation for your operators can automatically created from your Java source code and Javadoc comments.

6.11 Non-Operator classes

Some operators, like PerformanceEvaluator and the ExampleFilter, have parameters that let you specify implementations of certain interfaces that will solve simple subtasks, e.g. determining which of two performance vectors is preferable, or which examples to remove from an example set. Those classes must be specified with the fully qualified classname. If you want to implement such an interface, you simply have to add the implementation to your classpath and declare it to the operator. Of course it is also possible to add these implementations to your plugin.

6.12 Line Breaks

In order to ensure platform compatibility you should never use $\backslash n$ in your code. Line breaks should always be created with help of the methods

```
com.rapidminer.tools.Tools.getLineSeparator()
```

for a single line break and

```
com.rapidminer.tools.Tools.getLineSeparators(int)
```

for multiple line breaks.

6.13 GUI Programming

If you want to create visualizations of models or other output types of your operators you might be interested to stick to the RapidMiner look and feel guidelines. There are several things which should be considered for GUI programming:

- Use `ExtendedJTable` instead of `JTable`
- Use `ExtendedJScrollBar` instead of `JScrollBar`
- Use only the colors defined as constants in `SwingTools`

Chapter 7

Integrating RapidMiner into your application

RapidMiner can easily be invoked from other Java applications. You can both read process configurations from xml `Files` or `Readers`, or you can construct `Processes` by starting with an empty process and adding `Operators` to the created `Process` in a tree-like manner. Of course you can also create single operators and apply them to some input objects, e.g. learning a model or performing a single preprocessing step. However, the creation of processes allows RapidMiner to handle the data management and process traversal. If the operators are created without beeing part of an process, the developer must ensure the correct usage of the single operators himself.

7.1 Initializing RapidMiner

Before RapidMiner can be used (especially before any operator can be created), RapidMiner has to be properly initialized. The method

```
RapidMiner.init()
```

must be invoked before the `OperatorService` can be used to create operators. Several other initialization methods for RapidMiner exist, please make sure that you invoke at least one of these. If you want to configure the initialization of RapidMiner you might want to use the method

```
RapidMiner.init(InputStream operatorsXMLStream,
    File pluginDir,
    boolean addWekaOperators,
    boolean searchJDBCInLibDir,
    boolean searchJDBCInClasspath,
    boolean addPlugins)
```

Setting some of the properties to false (e.g. the loading of database drivers or of the Weka operators might drastically improve the needed runtime during start-up. If you even want to use only a subset of all available operators you can provide a stream to a reduced operator description (operators.xml). If the parameter `operatorsXMLStream` is null, just all core operators are used. Please refer to the API documentation for more details on the initialization of RapidMiner.

You can also use the simple method `RapidMiner.init()` and configure the settings via this list of environment variables:

- `rapidminer.init.operators` (file name)
- `rapidminer.init.plugins.location` (directory name)
- `rapidminer.init.weka` (boolean)
- `rapidminer.init.jdbc.lib` (boolean)
- `rapidminer.init.jdbc.classpath` (boolean)
- `rapidminer.init.plugins` (boolean)

7.2 Creating Operators

It is important that operators are created using one of the `createOperator(...)` methods of

```
com.rapidminer.tools.OperatorService
```

Table 7.1 shows the different factory methods for operators which are provided by OperatorService. Please note that few operators have to be added to a process in order to properly work. Please refer to section 7.4 for more details on using single operators and adding them to a process.

7.3 Creating a complete process

Figure 7.1 shows a detailed example for the RapidMiner API to create operators and setting its parameters.

```
import com.rapidminer.tools.OperatorService;
import com.rapidminer.RapidMiner;
import com.rapidminer.Process;
import com.rapidminer.operator.Operator;
import com.rapidminer.operator.OperatorException;
import java.io.IOException;

public class ProcessCreator {

    public static Process createProcess() {
        try {
        // invoke init before using the OperatorService
            RapidMiner.init();
        } catch (IOException e) { e.printStackTrace(); }

        // create process
        Process process = new Process();
        try {
            // create operator
            Operator inputOperator =
                OperatorService.createOperator(ExampleSetGenerator.class);

            // set parameters
            inputOperator.setParameter("target_function", "sum␣classification");

            // add operator to process
            process.getRootOperator().addOperator(inputOperator);

            // add other operators and set parameters
            // [...]
        } catch (Exception e) { e.printStackTrace(); }
        return process;
    }

    public static void main(String[] argv) {
        // create process
        Process process = createProcess();
        // print process setup
        System.out.println(process.getRootOperator().createProcessTree(0));

        try {
            // perform process
            process.run();
            // to run the process with input created by your application use
        // process.run(new IOContainer(new IOObject[] { ... your objects ... });
        } catch (OperatorException e) { e.printStackTrace(); }
    }
}
```

Figure 7.1: Creation of new operators and setting up an process from your application

Method	**Description**
`createOperator(String name)`	Use this method for the creation of an operator from its name. The name is the name which is defined in the `operators.xml` file and displayed in the GUI.
`createOperator(Operator-Description description)`	Use this method for the creation of an operator whose OperatorDescription is already known. Please refer to the RapidMiner API.
`createOperator(Class clazz)`	Use this method for the creation of an operator whose Class is known. This is the recommended method for the creation of operators since it can be ensured during compile time that everything is correct. However, some operators exist which do not depend on a particular class (e.g. the learners derivced from the Weka library) and in these cases one of the other methods must be used.

Table 7.1: These methods should be used to create operators. In this way it is ensured that the operators can be added to processes and properly used.

We can simply create a new process setup via `new Process()` and add operators to the created process. The root of the process' operator tree is queried by `process.getRootOperator()`. Operators are added like children to a parent tree. For each operator you have to

1. create the operator with help of the OperatorService,
2. set the necessary parameters,
3. add the operator at the correct position of the operator tree of the process.

After the process was created you can start the process via

```
process.run().
```

If you want to provide some initial input you can also use the method

```
process.run(IOContainer).
```

If you want to use a log file you should set the parameter `logfile` of the process root operator like this

```
process.getRootOperator().setParameter(
ProcessRootOperator.PARAMETER_LOGFILE, filename )
```

before the run method is invoked. If you want also to keep the global logging messages in a file, i.e. those logging messages which are not associated to a single process, you should also invoke the method

```
LogService.initGlobalLogging( OutputStream out, int
logVerbosity )
```

before the run method is invoked.

If you have already defined a process configuration file, for example with help of the graphical user interface, another very simple way of creating a process setup exists. Figure 7.2 shows how a process can be read from a process configuration file. Just creating a process from a file (or stream) is a very simple way to perform processes which were created with the graphical user interface beforehand.

```
public static IOContainer createInput() {
    // create a wrapper that implements the ExampleSet interface and
    // encapsulates your data
    // ...
    return new IOContainer(IOObject[] { myExampleSet });
}

public static void main(String[] argv) throws Exception {
    // MUST BE INVOKED BEFORE ANYTHING ELSE !!!
    RapidMiner.init();

    // create the process from the command line argument file
    Process process = new Process(new File(argv[0]));

    // create some input from your application, e.g. an example set
    IOContainer input = createInput();

    // run the process on the input
    process.run(input);
}
```

Figure 7.2: Using complete RapidMiner processes from external programs

As it was said before, please ensure that RapidMiner was properly initialized by one of the init methods presented above.

7.4 Using single operators

The creation of a Process object is the intended way of performing a complete data mining process within your application. For small processes like a single learning or preprocessing step, the creation of a complete process object might include a lot of overhead. In these cases you can easily manage the data flow yourself and create and use single operators.

The data flow is managed via the class IOContainer (see section 6.5.2). Just create the operators you want to use, set necessary parameters and invoke the method apply(IOContainer). The result is again an IOContainer which can deliver the desired output object. Figure 7.3 shows a small programm which loads some training data, learns a model, and applies it to an unseen data set.

Please note that using an operator without an surrounding process is only supported for operators not directly depending on others in an process configuration. This is true for almost all operators available in RapidMiner. There are, however, some exceptions: some of the meta optimization operators (e.g. the parameter optimization operators) and the ProcessLog operator only work if they are part of the same process of which the operators should be optimized or logged respectively. The same applies for the MacroDefinition operator which also can only be properly used if it is embedded in a Process. Hence, those operators cannot be used without a Process and an error will occur.

Please note also that the method

```
RapidMiner.init()
```

or any other init() taking some parameters must be invoked before the OperatorService can be used to create operators (see above).

7.5 RapidMiner as a library

If RapidMiner is separately installed and your program uses the RapidMiner classes you can just adapt the examples given above. However, you might also want to integrate RapidMiner into your application so that users do not have to download and install RapidMiner themself. In that case you have to consider that

1. RapidMiner needs a rapidminerrc file in rapidminer.home/etc directory

2. RapidMiner might search for some library files located in the directory rapidminer.home/lib.

```
public static void main(String[] args) {
    try {
        RapidMiner.init();

        // learn
        Operator exampleSource =
            OperatorService.createOperator(ExampleSource.class);
        exampleSource.setParameter("attributes",
                                   "/path/to/your/training_data.xml");
        IOContainer container = exampleSource.apply(new IOContainer());
        ExampleSet exampleSet = container.get(ExampleSet.class);

    // here the string based creation must be used since the J48 operator
    // do not have an own class (derived from the Weka library).
        Learner learner = (Learner)OperatorService.createOperator("J48");
        Model model = learner.learn(exampleSet);

        // loading the test set (plus adding the model to result container)
        Operator testSource =
            OperatorService.createOperator(ExampleSource.class);
        testSource.setParameter("attributes", "/path/to/your/test_data.xml");
        container = testSource.apply(new IOContainer());
        container = container.append(model);

        // applying the model
    Operator modelApp =
        OperatorService.createOperator(ModelApplier.class);
        container = modelApp.apply(container);

        // print results
        ExampleSet resultSet = container.get(ExampleSet.class);
        Attribute predictedLabel = resultSet.getPredictedLabel();
        ExampleReader reader = resultSet.getExampleReader();
        while (reader.hasNext()) {
            System.out.println(reader.next().getValueAsString(predictedLabel));
        }
    } catch (IOException e) {
        System.err.println("Cannot␣ initialize ␣RapidMiner:" + e.getMessage());
    } catch (OperatorCreationException e) {
        System.err.println("Cannot␣create␣operator:" + e.getMessage());
    } catch (OperatorException e) {
        System.err.println("Cannot␣create␣model:␣" + e.getMessage());
    }
}
```

Figure 7.3: Using single RapidMiner operators from external programs

For the Weka jar file, you can define a system property named `rapidminer.weka.jar` which defines where the Weka jar file is located. This is especially useful if your application already contains Weka. However, you can also just omit all of the library jar files, if you do not need their functionality in your application. RapidMiner will then just work without this additional functionality, for example, it simply does not provide the Weka learners if the weka.jar library was omitted.

7.6 Transform data for RapidMiner

Often it is the case that you already have some data in your application on which some operators should be applied. In this case, it would be very annoying to write your data into a file, load it into RapidMiner with an ExampleSource operator and apply other operators to the resulting ExampleSet. It would therefore be a nice feature if it would be possible to directly use your own application data as input. This section describes the basic ideas for this approach.

As we have seen in Section 6.7, all data is stored in a central data table (called `ExampleTable`) and one or more views on this table (called `ExampleSets`) can be created and will be used by operators. Figure 7.4 shows how this central `ExampleTable` can be created.

First of all, a list containing all attributes must be created. Each `Attribute` represents a column in the final example table. We assume that the method `getMyNumOfAttributes()` returns the number of regular attributes. We also assume that all regular attribute have numerical type. We create all attributes with help of the class `AttributeFactory` and add them to the attribute list.

For example tables, it does not matter if a specific column (attribute) is a special attribute like a classification label or just a regular attribute which is used for learning. We therefore just create a nominal classification label and add it to the attribute list, too.

After all attributes were added, the example table can be created. In this example we create a `MemoryExampleTable` which will keep all data in the main memory. The attribute list is given to the constructor of the example table. One can think of this list as a description of the column meta data or column headers. At this point of time, the complete table is empty, i.e. it does not contain any data rows.

The next step will be to fill the created table with data. Therefore, we create a DataRow object for each of the `getMyNumOfRows()` data rows and add it to the table. We create a simple double array and fill it with the values from your application. In this example, we assume that the method `getMyValue(d,a)` will deliver the value for the a-th attribute of the d-th data row. Please note that the order of values and the order of attributes added to the attribute list

```
import com.rapidminer.example.*;
import com.rapidminer.example.table .*;
import com.rapidminer.example.set .*;
import com.rapidminer.tools . Ontology;
import java. util .*;

public class CreatingExampleTables {

  public static void main(String [] argv) {
    // create attribute list
    List<Attribute> attributes = new LinkedList<Attribute>();
    for (int a = 0; a < getMyNumOfAttributes(); a++) {
      attributes .add( AttributeFactory . createAttribute ("att" + a,
                                                  Ontology.REAL));
    }
    Attribute label = AttributeFactory. createAttribute (" label ",
                                                  Ontology.NOMINAL));
    attributes .add(label );

    // create table
    MemoryExampleTable table = new MemoryExampleTable(attributes);

    // fill table (here: only real values)
    for (int d = 0; d < getMyNumOfDataRows(); d++) {
      double[] data = new double[attributes. size ()];
      for (int a = 0; a < getMyNumOfAttributes(); a++) {
        // fill with proper data here
        data[a] = getMyValue(d, a);
      }

      // maps the nominal classification to a double value
      data[data. length - 1] =
          label .getMapping().mapString(getMyClassification (d ));

      // add data row
      table . addDataRow(new DoubleArrayDataRow(data));
    }

    // create example set
    ExampleSet exampleSet = table.createExampleSet(label);
  }
}
```

Figure 7.4: The complete code for creating a memory based ExampleTable

must be the same!

For the label attribute, which is a nominal classification value, we have to map the `String` delivered by `getMyClassification(d)` to a proper double value. This is done with the method `mapString(String)` of `Attribute`. This method will ensure that following mappings will always produce the same double indices for equal strings.

The last thing in the loop is to add a newly created `DoubleArrayDataRow` to the example table. Please note that only MemoryExampleTable provide a method `addDataRow(DataRow)`, other example tables might have to initialized in other ways.

The last thing which must be done is to produce a view on this example table. Such views are called `ExampleSet` in RapidMiner. The creation of these views is done by the method `createCompleteExampleSet(label, null, null, null)`. The resulting example set can be encapsulated in a `IOContainer` and given to operators.

Remark: Since Attribute, DataRow, ExampleTable, and ExampleSet are all interfaces, you can of course implement one or several of these interfaces in order to directly support RapidMiner with data even without creating a MemoryExampleTable.

Chapter 8

Acknowledgements

We thank SourceForge[1] for providing a great platform for open-source development.

We are grateful to the developers of Eclipse[2], Ant[3], and JUnit[4] for making these great open-source development environments available.

We highly appreciate the operators and extensions written by several external contributors. Please check our website for a complete list of authors.

We thank the Weka[5] developers for providing an open source Java archive with lots of great machine learning operators.

We are grateful to Stefan Rüping for providing his implementation of a support vector machine[6].

We thank Chih-Chung Chang and Chih-Jen Lin for their SVM implementation LibSVM[7].

We would like to thank Stefan Haustein for providing his library `kdb`[8], which we use for several input formats like dBase and BibTeX.

Thanks to the users of RapidMiner. Your comments help to improve RapidMiner for both end users and data mining developers.

[1] http://sourceforge.net/
[2] http://www.eclipse.org
[3] http://ant.apache.org
[4] http://www.junit.org
[5] http://www.cs.waikato.ac.nz/ml/weka/
[6] http://www-ai.informatik.uni-dortmund.de/SOFTWARE/MYSVM/
[7] http://www.csie.ntu.edu.tw/cjlin/libsvm/
[8] http://www.me.objectweb.org/

Appendix A

Regular expressions

Regular expressions are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used as a tool to search, edit or manipulate text or data. Regular expressions range from being simple to quite complex, but once you understand the basics of how they're constructed, you'll be able to understand any regular expression.

In RapidMiner several parameters use regular expressions, e.g. for the definition of the column separators for the ExampleSource operator or for the feature names of the FeatureNameFilter. This chapter gives an overview of all regular expression constructs available in RapidMiner. These are the same as the usual regular expressions available in Java. Further information can be found at

http://java.sun.com/docs/books/tutorial/extra/regex/index.html.

A.1 Summary of regular-expression constructs

Construct	**Matches**
Characters	
x	The character x
\\	The backslash character
\0n	The character with octal value 0n (0 <= n <= 7)
\0nn	The character with octal value 0nn (0 <= n <= 7)
\0mnn	The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)

Construct	**Matches**
\xhh	The character with hexadecimal value 0xhh
\uhhhh	The character with hexadecimal value 0xhhhh
\t	The tab character (’\u0009’)
\n	The newline (line feed) character (’\u000A’)
\r	The carriage-return character (’\u000D’)
\f	The form-feed character (’\u000C’)
\a	The alert (bell) character (’\u0007’)
\e	The escape character (’\u001B’)
\cx	The control character corresponding to x
Character classes	
[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)
Predefined character classes	
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]
POSIX character classes (US-ASCII only)	
\p{Lower}	A lower-case alphabetic character: [a-z]
\p{Upper}	An upper-case alphabetic character: [A-Z]
\p{ASCII}	All ASCII: [\x00-\x7F]
\p{Alpha}	An alphabetic character:[\p{Lower}\p{Upper}]
\p{Digit}	A decimal digit: [0-9]
\p{Alnum}	An alphanumeric character:[\p{Alpha}\p{Digit}]
\p{Punct}	Punctuation: One of !”#$%&’()*+,-./:;<=>?@[\]^_‘{—}∼

Construct	**Matches**
\p{Graph}	A visible character: [\p{Alnum}\p{Punct}]
\p{Print}	A printable character: [\p{Graph}]
\p{Blank}	A space or a tab: [\t]
\p{Cntrl}	A control character: [\x00-\x1F\x7F]
\p{XDigit}	A hexadecimal digit: [0-9a-fA-F]
\p{Space}	A whitespace character: [\t\n\x0B\f\r]
Classes for Unicode blocks and categories	
\p{InGreek}	A character in the Greek block (simple block)
\p{Lu}	An uppercase letter (simple category)
\p{Sc}	A currency symbol
\P{InGreek}	Any character except one in the Greek block (negation)
[\p{L}&&[^\p{Lu}]]	Any letter except an uppercase letter (subtraction)
Boundary matchers	
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input
Greedy quantifiers	
X?	X, once or not at all
X*	X, zero or more times
X+	X, one or more times
X{n}	X, exactly n times
X{n,}	X, at least n times
X{n,m}	X, at least n but not more than m times
Reluctant quantifiers	
X??	X, once or not at all
X*?	X, zero or more times
X+?	X, one or more times
X{n}?	X, exactly n times
X{n,}?	X, at least n times

Construct	Matches
X{n,m}?	X, at least n but not more than m times
Logical operators	
XY	X followed by Y
X—Y	Either X or Y
(X)	X, as a capturing group
Back references	
\n	Whatever the n-th capturing group matched
Quotation	
\	Nothing, but quotes the following character
\Q	Nothing, but quotes all characters until \E
\E	Nothing, but ends quoting started by \Q
Special constructs (non-capturing)	
(?:X)	X, as a non-capturing group
(?idmsux-idmsux)	Nothing, but turns match flags on - off
(?idmsux-idmsux:X)	X, as a non-capturing group with the given flags on - off
(?=X)	X, via zero-width positive lookahead
(?!X)	X, via zero-width negative lookahead
(?<=X)	X, via zero-width positive lookbehind
(?<!X)	X, via zero-width negative lookbehind
(?>X)	X, as an independent, non-capturing group

Bibliography

[1] Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2001.

[2] G. Daniel, J. Dienstuhl, S. Engell, S. Felske, K. Goser, R. Klinkenberg, K. Morik, O. Ritthoff, and H. Schmidt-Traub. *Advances in Computational Intelligence – Theory and Practice*, chapter Novel Learning Tasks, Optimization, and Their Application, pages 245–318. Springer, 2002.

[3] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the KDD 2004*, 2004.

[4] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the KDD 1996*, 1996.

[5] Sven Felske, Oliver Ritthoff, and Ralf Klinkenberg. Bestimmung von isothermenparametern mit hilfe des maschinellen lernens. Technical Report CI-149/03, Collaborative Research Center 531, University of Dortmund, 2003.

[6] Ralf Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In *Workshop notes of the IJCAI-01 Workshop on* Learning from Temporal and Spatial Data, pages 16–24, 2001.

[7] Ralf Klinkenberg. Predicting phases in business cycles under concept drift. In *Proc. of LLWA 2003*, pages 3–10, 2003.

[8] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, 8(3), 2004.

[9] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 487–494, San Francisco, CA, USA, 2000. Morgan Kaufmann.

[10] Ralf Klinkenberg, Oliver Ritthoff, and Katharina Morik. Novel learning tasks from practical applications. In *Proceedings of the workshop of the special interest groups Machine Learning (FGML)*, pages 46–59, 2002.

[11] Ralf Klinkenberg and Stefan Rüping. Concept drift and the importance of examples. In Jürgen Franke, Gholamreza Nakhaeizadeh, and Ingrid Renz, editors, *Text Mining – Theoretical Aspects and Applications*, pages 55–77. Physica-Verlag, Heidelberg, Germany, 2003.

[12] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, 2006.

[13] Ingo Mierswa. Automatic feature extraction from large time series. In *Proc. of LWA 2004*, 2004.

[14] Ingo Mierswa. Incorporating fuzzy knowledge into fitness: Multiobjective evolutionary 3d design of process plants. In *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2005*, 2005.

[15] Ingo Mierswa and Thorsten Geisbe. Multikriterielle evolutionre aufstellungsoptimierung von chemieanlagen unter beachtung gewichteter designregeln. Technical Report CI-188/04, Collaborative Research Center 531, University of Dortmund, 2004.

[16] Ingo Mierswa, Ralf Klinkberg, Simon Fischer, and Oliver Ritthoff. A flexible platform for knowledge discovery experiments: Yale – yet another learning environment. In *Proc. of LLWA 2003*, 2003.

[17] Ingo Mierswa and Katharina Morik. Automatic feature extraction for classifying audio data. *Machine Learning Journal*, 58:127–149, 2005.

[18] Ingo Mierswa and Katharina Morik. Method trees: Building blocks for self-organizable representations of value series. In *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2005, Workshop on Self-Organization In Representations For Evolutionary Algorithms: Building complexity from simplicity*, 2005.

[19] Ingo Mierswa and Michael Wurst. Efficient case based feature construction for heterogeneous learning tasks. Technical Report CI-194/05, Collaborative Research Center 531, University of Dortmund, 2005.

[20] Ingo Mierswa and Michael Wurst. Efficient feature construction by meta learning – guiding the search in meta hypothesis space. In *Proc. of the Internation Conference on Machine Learning, Workshop on Meta Learning*, 2005.

[21] O. Ritthoff, R. Klinkenberg, S. Fischer, I. Mierswa, and S. Felske. YALE: Yet Another Machine Learning Environment. In *Proc. of LLWA 01*, pages 84–92. Department of Computer Science, University of Dortmund, 2001.

[22] Oliver Ritthoff and Ralf Klinkenberg. Evolutionary feature space transformation using type-restricted generators. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 1606–1607, 2003.

[23] Oliver Ritthoff, Ralf Klinkenberg, Simon Fischer, and Ingo Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. In *Proc. of the 2002 U.K. Workshop on Computational Intelligence (UKCI-02)*, pages 147–154, 2002.

[24] Martin Scholz. Knowledge-Based Sampling for Subgroup Discovery. In Katharina Morik, Jean-Francois Boulicaut, and Arno Siebes, editors, *Proc. of the Workshop on Detecting Local Patterns*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[25] Martin Scholz. Sampling-Based Sequential Subgroup Mining. In *Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Databases (KDD'05)*, 2005. Accepted for publication.

Index