RapidMiner Radoop Documentation

Release 2.3.0

RapidMiner

April 30, 2015

CHAPTER

ONE

INTRODUCTION

1.1 Preface

RapidMiner Radoop provides an easy-to-use graphical interface for analyzing data on a Hadoop cluster. It requires a properly configured Hadoop cluster with a running Hive server. This documentation describes the requirements to use the Radoop extension, provides a quick introduction to the software and contains the detailed specification of the operators and capabilities of the solution to process and analyze big data.

1.2 Basic Architecture

RapidMiner Radoop is a client software that connects to a Hadoop cluster and executes processes created with an intuitive graphical user interface. Nothing must be installed on the cluster itself. The *Architecture diagram of the RapidMiner Radoop client* figure shows the basic architecture of the complete solution.

RapidMiner Server is also capable of using the Radoop extension. It executes, schedules, manages processes created with the client and provides additional collaboration features and a web reporting interface. The *Architecture diagram of RapidMiner Server with Radoop* figure shows the basic architecture of the complete solution.

1.3 Documentation Overview

Here follows a short summary of the documentation chapters. You may want to jump to the part that mostly interests you, however, the chapters are arranged in a way to support a guided introduction to the software, so at any time you can just simply proceed with the next one.

RapidMiner Radoop requires a Hadoop cluster accessible from the client machine that runs the software. The Installation Guide is available on the RapidMiner Radoop Documentation site. It describes basic steps to setup or configure your cluster if it is not ready yet or does not meet some of the requirements. If your cluster is ready, the documentation guides you through the simple installation process of the Radoop client. It also describes how to configure the connection to your cluster(s) in the software. Security issues about your Hadoop cluster and connection are covered in the section about the cluster security. Installation instructions for the RapidMiner Server are also covered in the installation guide.

Your data may already be on the Hadoop cluster in a Hive structure, otherwise, Radoop provides tools to import your data to the cluster. Please refer to the *Data Loading Guide* for details. Knowing how to use RapidMiner's graphical interface, process designer, etc. is essential to work with Radoop effectively. If you are yet unfamiliar with it, start to learn it with the *RapidMiner Data Flow* chapter, which may eventually guide you to the RapidMiner Documentation.

Equipped with basic RapidMiner experience, *Radoop Basics* introduces you to Radoop. Here you will learn how to design processes that run on your Hadoop cluster and perform distributed data processing. *Advanced Radoop Pro-*

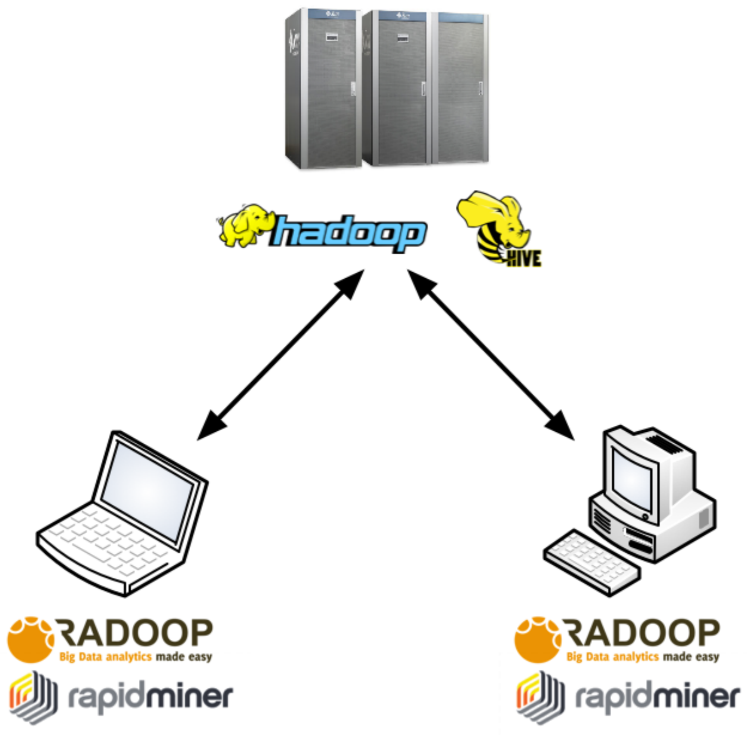


Figure 1.1: Architecture diagram of the RapidMiner Radoop client

cesses takes you further in the introduction to the software by covering overview of designing data mining processes, scoring, evaluation and advanced data flow design with Radoop.

Appendices are also important part of this documentation. *Appendix A - Radoop Properties* lists all the settings that influence Radoop. Default settings are probably ok, but you should definitely take a look at this table. *Appendix B - Hive Management Perspective* describes the interface Radoop provides you to manage your data on the cluster easily. You can use this handy perspective to do smaller tasks with your data that is processed in your data flows. *Appendix C - Radoop Operator Reference* is the place where you can find all the operators that you can use to build Radoop processes in addition to the basic RapidMiner operators (for the latter, please consult the RapidMiner Studio Documentation). This appendix describes the operators together with their inputs, outputs and parameters and may give you advices to their usage as well.

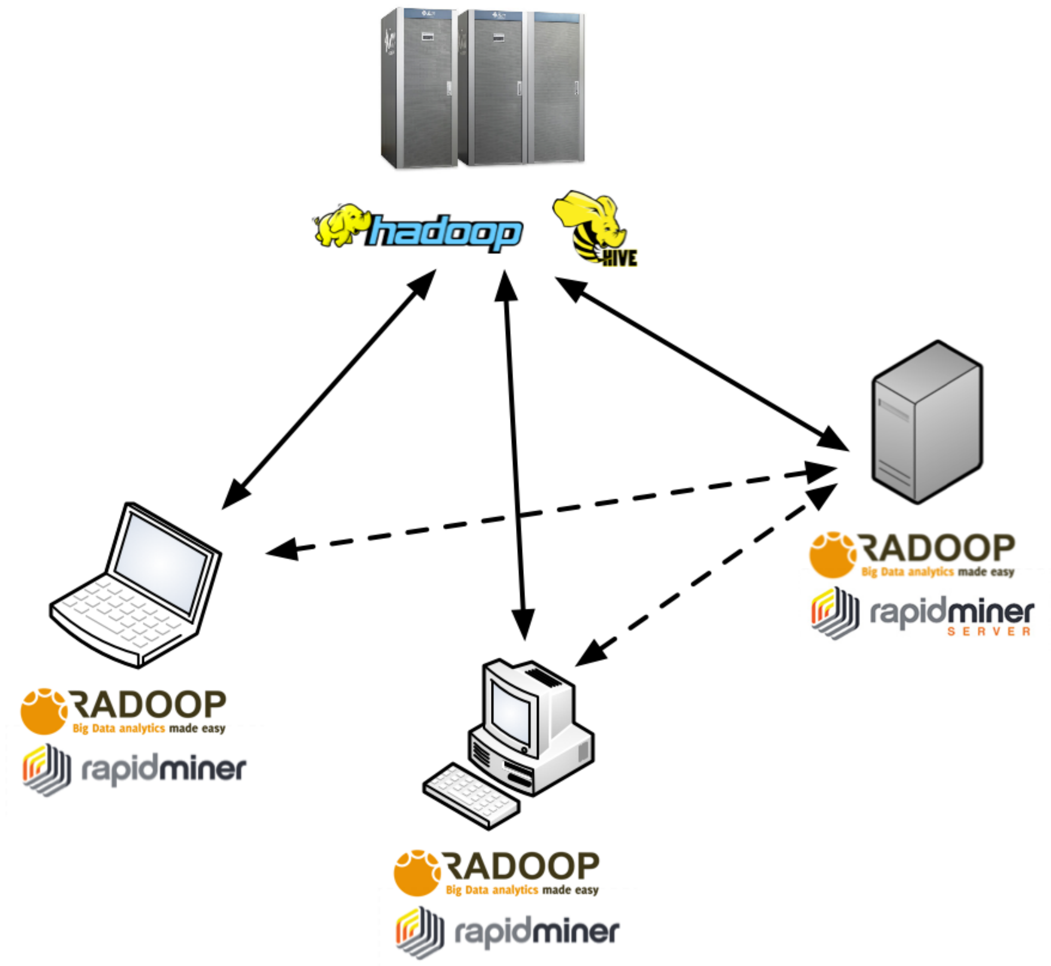


Figure 1.2: Architecture diagram of RapidMiner Server with Radoop

CHAPTER

TWO

RAPIDMINER DATA FLOW

2.1 Preface

Radoop is the big data analysis extension for RapidMiner, the powerful ETL and data mining software. This section is a short introduction to RapidMiner. However, if you are not familiar with the software, you should consult the RapidMiner Documentation (for the user manual and video tutorials). The purpose of this section is only to provide a glimpse into this software and to explain how the solution works with your Hadoop cluster.

2.2 RapidMiner

RapidMiner is a popular open source ETL, data mining and reporting solution. It is both a stand-alone application and an engine to be integrated into other appications. This chapter highlights and shortly describes the main features of the solution. Radoop brings all of these features to the big data level and provides a full analytics package for your big data problem.

- **Data integration**, **ETL**, **analysis** and **reporting** in a single solution.
- Design all analysis processes using a powerful and intuitive **graphical user interface**.
- **Meta data transformation** during design time. You can inspect meta data (attributes, data types, statistics) of your data sets at any step during the design of your process.
- **Errors** are **recognized** early, mostly during design time. **Quick fixes** help solving these issues, thus dramatically decreasing the learning curve.
- **Hundreds of operators** for loading, transformation, modeling and visualization.
- **Repositories** for handling processes, data and meta data.

The following sections shortly introduce the most important basic concepts of the solution.

2.3 User Interface

RapidMiner's user interface is basically made up of *perspectives*. These perspectives all consist of interface elements called *views*. After installing the Radoop extension, you will find the following five perspectives to choose from (from left to right on the toolbar on the top of your screen):

- *Home*. This is the welcome screen with the most frequent starting points for your work. You can access your recent processes, create a new one, open a template or the RapidMiner Tutorial.
- *Design*. This perspective is where you design your process from basic building blocks or steps called *operators*. You can manage processes and data in your repositories in the *Repositories* view. Configure your operators

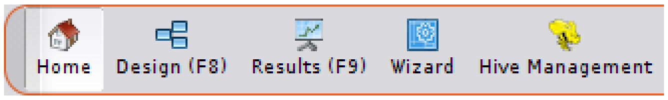


Figure 2.1: Perspectives: Home, Design, Results, Wizard, Hive Management

using the *Parameters* view, after you have made yourself familiar with them by reading the description in the *Help* view. In the *Process* view you implement the data flow by connecting the input and output ports of the operators to each other. You can create subprocesses and continue the design inside them. Examine the meta data on the ports or on the connection at design time. The *Problems* view shows warnings and errors related to the process, which should be addressed before you start the process by using the *Run* button. The so-called quick fixes may help you to solve these problems. You can add breakpoints to the process to help the design or the debugging of your process. You can pause the run and examine the data at these points. The *Log* view shows information, warning and error messages during the process run. The statusbar on the bottom of the screen shows the active operator and the elapsed time since the start. The *Overview* view helps in navigating around a more complex process. The *XML* view shows the xml code that describes the process you created in the Process view. Users of the earlier versions of RapidMiner may be more familiar with the *Tree* view, which is another representation of the process having the advantage that the operators inside a subprocess or an *operator chain* are visible in a single view.

- *Results*. This is where you can examine the data sets, models, evaluation results or any other output of your process. When a process completes, the Result perspective shows all the objects that are connected to any of the process output ports. You can also view intermediate results if you pause the process with a breakpoint. The *Result Overview* may show all previous processes' results, each of them can be examined in a new tab on this perspective. This is where you can access the rich visualization capabilities of the software. You can visualize models, plot data and create advanced data charts. You can store the results in the repository, print or export them to files. The Repositories and the Log views should be familiar from the Design perspective. The Results perspective shows a fourth view by default, the *System Monitor*, which is used to keep an eye on the allocated and the available operative memory for the program.
- *Wizard*. Application Wizard helps you application templates for churn reduction, sentiment analysis, predictive maintenance and direct marketing.
- *Hive Management*. This perspective is a powerful tool to manage your data on the Hadoop cluster. Read more about this perpective in the section *Appendix B - Hive Management Perspective*.

The following sections provide a brief overview on the process, the data flow and related concepts.

2.4 RapidMiner Process

A RapidMiner process consists of a set of possibly nested operators and connections between them, which control the input and output objects (*I/O objects*) between these blocks. More than 500 operators are ready to implement professional data analysis.

The process can be created using the flow design or the operator tree representation. The operators have configurable parameters, some of which may be mandatory. Other parameters may be so-called expert parameters, that are only visible if you enable them. The operators also have input and output ports, through which they are connected to each other, thus implementing the process data flow. An operator consumes objects on its input ports, and deliver objects on its output ports. These connections may carry several types of objects other then data sets, like different types of models, files, performance vectors, etc.

The operators may be nested to implement common programming techniques, like loops, or to encapsule reusable subprocesses. But most of the time you deal with one of the powerful *meta operators*, operators using other operators for performing analytics tasks like validation, parameter optimization or meta modeling. The flexibility and reusability of this kind of process design makes RapidMiner an exceptionally powerful and intuitive analytics solution with unbeatable modeling capabilities.

RapidMiner constantly validates your process during the design, revealing errors as soon as possible. This prevents you or other users from spending much time on understanding or debugging a process. You can examine the meta data of your results as soon as you connect your operators, as RapidMiner performs the transformations on the meta data automatically while it validates the process.

2.5 In-memory Analysis

The most common object you deal with in RapidMiner is the so-called *ExampleSet* object. These objects represent data sets that are stored in the operative memory, and are very much similar to tables in a relational database. The *attributes* are columns in these tables. They have their declared *data types*, and they may have so-called *roles* that define special roles they play in some operators.

Multi-layered data view concept ensures that RapidMiner handles data efficiently. However, generally the size of the operative memory limits the size of the data sets you may deal with in a RapidMiner process. The main purpose of the Radoop extension is to overcome this limitation while keeping all original operators and the exceptionally powerful process design interface of the software. The operators in the Radoop operator group all operate on your Hadoop cluster while requiring only a minimal memory footprint on the client. This way, Radoop enables you to scale without limits and analyse even terabytes or petabytes of data. To learn more about creating processes that combine the powerful in-memory analysis with working on large amount of data on the cluster, please refer to the next chapter, *Radoop Basics*. The following subsection gives you an overview of the rich set of RapidMiner operators by shortly describing the main operator groups.

2.6 Core Operator Groups

Excluding Radoop operators, RapidMiner contains the following core operator groups:

- *Repository Access*: Operators related to accessing and managing objects in a repository.
- *Import*: Read objects (data, models, results, etc.) from external formats into the memory.
- *Export*: Large number of operators for writing objects to external formats.
- *Data Transformation*: Contains all preprocessing and trailing
- *Process Control*: Operators like loops and branches to control the process flow.
- *Utility*: Operators for grouping subprocesses, using macros or logging.
- *Modeling*: Data mining algorithms, methods for clustering, classification, association rules, regression, correlation and similarity analysis.
- *Evaluation*: Methods for evaluating the quality of models.

CHAPTER

THREE

RADOOP BASICS

3.1 Preface

This chapter covers the main concepts that are necessary to be able to understand and design processes that run on your Hadoop cluster. Since Apache Hive is part of the Radoop solution, this documentation often refers to basic Hive concepts like tables or columns. If you are unfamiliar with Hive, you can just think of it as a (distributed) relational database or data warehouse infrastructure over Hadoop. It understands SQL and translates it to MapReduce jobs.

3.2 Radoop Nest

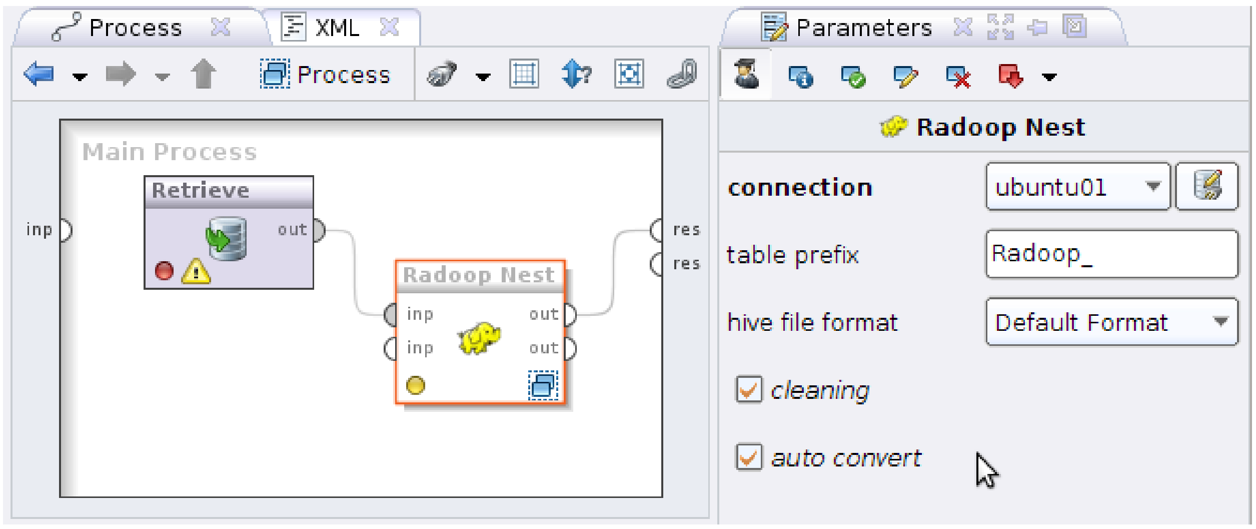
Radoop Nest is the most important building block. Every Radoop process must contain at least one Radoop Nest (meta) operator. It specifies the connection to the Hadoop cluster. The subprocess you put inside the Radoop Nest describes the process that runs on this Hadoop cluster. All other parts of the process outside the Nest process the data in the memory.

Data processed by basic RapidMiner operators always resides in the memory. On the contrary, Radoop operators - operators inside Radoop Nest - process the data on the Hadoop cluster. When you design a process, you will not notice any difference between these two sets of operators considering inputs, outputs or parameters. You can easily connect arbitrary operators to each other assuming they get their mandatory input delivered. The Problems view helps you address issues in the process you are designing.

Radoop Nest is an operator chain that runs a subprocess inside, constructed of arbitrary Radoop operators. This subprocess should implement every operation on the cluster. Most of these operators has a RapidMiner counterpart, the main difference is that Radoop operators have minimal memory footprint and always keep and process the data on the cluster. Radoop Nest switches the context: its subprocess runs on the cluster. The cluster connection is a parameter of the Radoop Nest operator. If you change this parameter, the client immediately tests the connection (see the progress bar in the lower right corner) and raises a warning if the connection test fails.

There are three more parameters for this operator:

- *table prefix*: During a process run, Radoop creates temporary objects in Hive. The name of these objects start with the prefix that you define in this parameter. It may be a good practice to use user specific prefixes during the design. You can also use *rapidminer.radoop.table.prefix* global property to set a default value for this parameter (the property applies on new Radoop Nest operators).
- *cleaning*: This parameter decides whether Radoop should delete the temporary objects after a process has been finished (default behaviour). It is highly recommended that you do not uncheck this parameter, since your cluster may soon be full of these temporary objects. However, you may want to uncheck this for a short period of time to debug your process. In this case, please refer to *Appendix B - Hive Management Perspective* for deleting these objects easily afterwards.



connection: ubuntu01
table prefix: Radoop_
hive file format: Default Format
[✓] cleaning
[✓] auto convert

Figure 3.1: Cluster connection is a parameter of the Radoop Nest.

- *auto convert*: This is an expert parameter for experienced users. If set to true (default and recommended state), then the data sets (stored in the operative memory) consumed at Radoop Nest input ports are immediately pushed to the cluster. That means the data is written to a temporary Hive table, and is ready to serve as the input of operators inside the Nest. If this parameter is set to false, then only the Radoop operators inside the Nest will write the data to the cluster usually when they consume it on their input ports. In the latter case you can also operate on in-memory data sets inside the Nest, but you would rarely need to do this. Below, you can read more about the data sets and the main difference between operators inside the Nest (that run on the cluster) and basic RapidMiner operators.
- *hive file format / impala file format*: This parameter defines what storage format should Radoop use inside this Nest for Hive tables. This setting applies both to temporary and permanent Hive tables, although the storage format of the latter may be overriden both in the *Store in Hive* and the *Read CSV* operators. The default option is to omit specifying the storage format, thus leaving it to Hive server / Impala default settings, which usually results in TEXTFILE format. TEXTFILE format can be also explicitly defined in this parameter. This format has the advantage, that the data is in a human-readable format on the distributed file system. However, if you want to use more advanced file formats for optimized performance (lower size and faster processing), you can use other formats as well.

You can easily combine memory-based and cluster-based operators in the same process. Radoop Nest may have arbitrary number of input ports, which accept ExampleSet objects. Radoop Nest will automatically import the data on its input port from the client's operative memory to the cluster. The operators inside the Nest consume and produce so-called HadoopExampleSet objects. This object type is the cluster-based variant of the ExampleSet object. The difference between an ExampleSet and a HadoopExampleSet is that the latter stores the data in Hive, in a temporary or permanent table or view, and have minimal footprint on the client's operative memory. You can, of course, also process data that resides already on the cluster or import data directly to the cluster. That means, you do not have to use any input ports of the Radoop Nest operator, just access or import the data in its subprocess. See the next section for such import scenarios.

Radoop Nest may have arbitrary number of output ports that deliver memory-based ExampleSet objects directly to a process output port or to an input port of the next RapidMiner operator outside the Nest. You can connect an operator inside Radoop Nest to the Nest's output port. The data or a data sample from the HadoopExampleSet output will be fetched to the operative memory of the client and propagate further as a memory-based ExampleSet on the process flow. The data sample must fit into the operative memory. Typically, you may want to work on aggregated data that fits into the memory after the aggregation took place on the cluster. You can limit the number of rows for the fetched data sample using a property (*rapidminer.radoop.sample_size.overall*, see *Appendix A - Radoop Properties*).

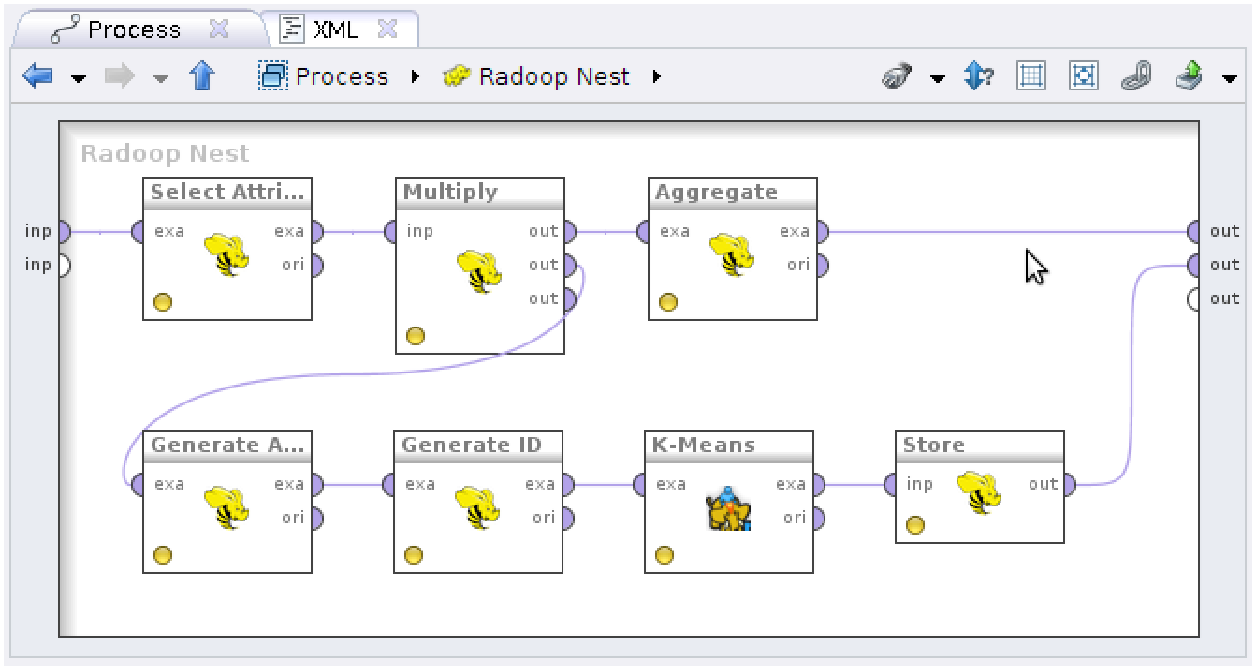


Figure 3.2: The subprocess inside Radoop Nest runs on the cluster.

The process can be started using the Run button on the Process Perspective. You can follow the run with the help of the status icons on the operators and the statusbar on the bottom of the screen. However, in case of Radoop operators, this can be misleading at first, because the operator that seems to be active may process previous operations also, which were deferred only. The reason is that Radoop operators usually create only Hive views, and postpone the calculations. The intensive computation, the MapReduce jobs only takes place when the HadoopExampleSet is materialized, which means that a Hive table is generated for it, meaning that the - sometimes temporary - data is written to the HDFS. This is only done when it is necessary or when the optimizer algorithm decides to do so.

3.3 Data Model

The terms attribute and columns are interchangeable in this document, as an attribute in RapidMiner can be a column of a Hive table or view on the cluster and vice versa. The following two tables shows the matching of data types between RapidMiner and Hive: *Conversion to Hive data type*, *Conversion to Hive data type*. The first one shows the conversion that takes place during an ExampleSet import and the second shows the conversion that takes place when the data is fetched from the cluster to the operative memory. Please note that table and attribute names may change slightly inside a Radoop Nest: these identifiers are automatically converted to lowercase, special characters are replaced by underscores and the collision with certain reserved words in Hive may be avoided by an underscore suffix. E.g., an attribute with the name “Column” in RapidMiner becomes “column_” inside the Radop Nest, since “COLUMN” is a keyword in the Hive Query Language. You will notice and easily track these changes during design time by checking the meta data propagation.

Table 3.1: Conversion to Hive data type

RapidMiner data type	Hive data type
integer	bigint
real	double
numerical	double
binominal	string
polynominal	string
nominal	string
date	string
other	string

Table 3.2: Conversion to RapidMiner data type

Hive data type	RapidMiner data type
tinyint	integer
smallint	integer
int	integer
bigint	integer
float	real
double	real
string	nominal
other	nominal

RapidMiner maintains a *nominal mapping* for nominal attributes. This internal data structure maps nominal (string) values to double values for efficient memory usage. As the process inside the Nest runs on the cluster and must have minimal operative memory footprint, Radoop does not maintain this structure, only for *binominal attributes*. However, when Radoop Nest delivers a data set on its output port fetched from the cluster, this nominal mapping may be rebuilt by subsequent operators of the process. Please pay attention to notes about the usage of nominal attributes in core operator help texts. The nominal mapping for binominal attributes also tells us which string value is considered as the positive value out of the two possible string values.

3.4 Radoop Operators

Besides Radoop Nest, which is the container for the subprocess on the Hadoop cluster, there are a lot of Radoop operators that can run inside the Radoop Nest. These operators are categorized into the following groups.

3.4.1 Hive Access

The group contains three operators and a subgroup: Retrieve, Store, Append and the group Table Management. This subgroup also contains three operators: Drop, Rename and Copy.

Data that resides already on the cluster in a Hive table can be accessed by using *Retrieve* operator. The operator only loads references and meta data into the memory, the data remains on the cluster to be processed with further operators. A data sample will only be fetched to the memory, if the output port is connected directly to the Radoop Nest output.

Store and *Append* writes the data on their input ports into a Hive table. Usually, this may take some time, as the cluster has to carry out all the tasks defined by previous data processing operators before writing the data to the HDFS. Append verifies that the data on its input fits into the specified Hive table.

Table Management operators are able to do typical management operations on these Hive tables. E.g. *Drop* may be used before an Append operator in a loop to make sure that a new process run starts with a new table. Both *Rename* and *Copy* may help in more complex processes that may include loops, or they may manage tables of other processes that run before or after them.

3.4.2 Import/Export

The data must reside on the cluster to be processed and analysed by Radoop operators. Smaller data sets may fit into the memory. Radoop imports any ExampleSet automatically that you connect to one of the Radoop Nest's input port. Larger data sets may be imported with an operator from the Import group. The group contains two operators: Read CSV and Read Database.

Read CSV can create a Hive table from a file that resides on the client or from a file that resides on the HDFS or on Amazon S3. The import can be configured easily with the help of a wizard. You can define the separator, the name of the attributes (or set to read them from the file), and modify the type or set the roles of the attributes.

Read Database queries arbitrary data from a database and writes the result to the cluster. Radoop Client executes this process, so the database has to be accessable from the client, which writes the data to the HDFS using only a minimal memory footprint on the client. Supported databases are MySQL, PostgreSQL, Sybase, Oracle, HISQLDB, Ingres, Microsoft SQL Server or any other database using an ODBC Bridge. If you want to import from a database in direct or parallel manner (leaving the client out of the route), look for a solution like Sqoop.

Export operators (*Write CSV*, *Write Database*) write the data referenced by the HadoopExampleSet on their input to a flat file on the client or on the HDFS or on Amazon S3, or write the data into a database through the specified database connection.

Read more about data import in the chapter *Data Loading Guide*.

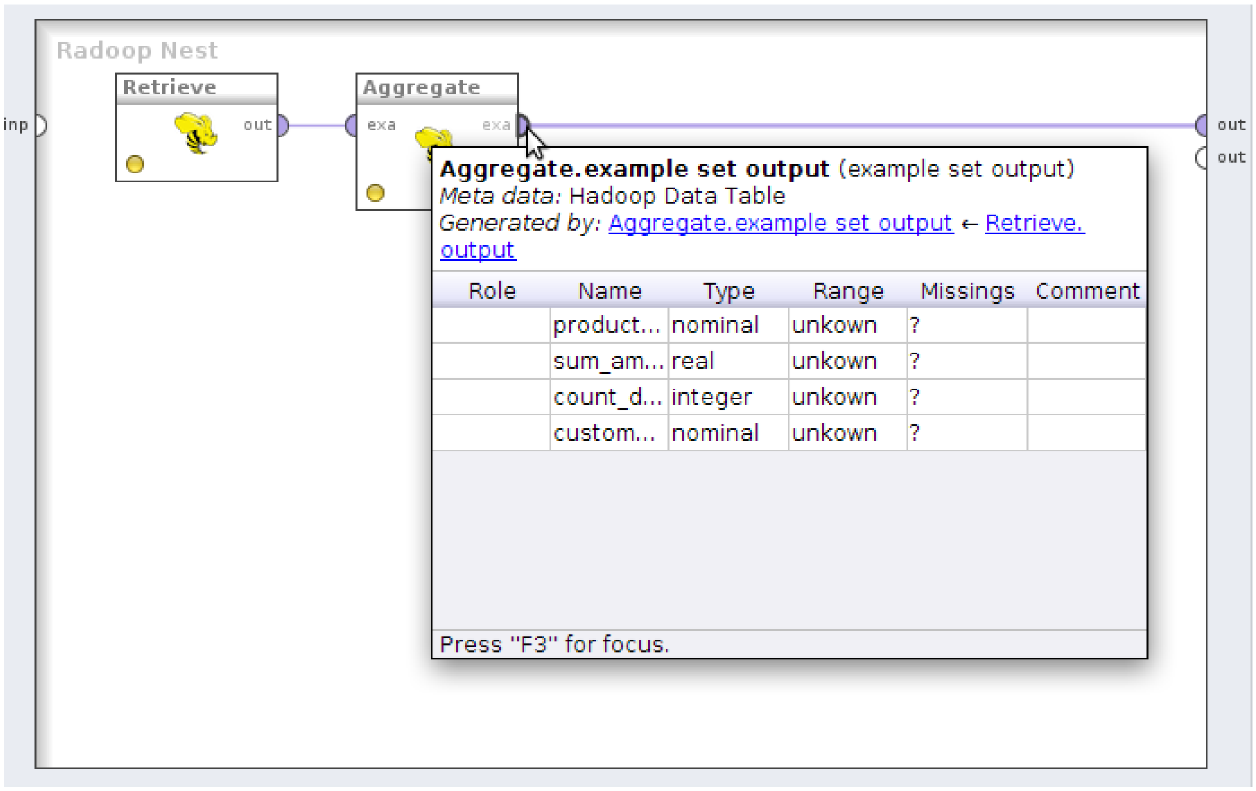
3.4.3 Transformation

This is the largest group which contains all transformation operators organized into subgroups. The operators usually take a HadoopExampleSet as input, and have two output ports: one for the transformed HadoopExampleSet and one for the original one. Bring the mouse over the first output port and you can examine the structure or meta data after the transformation during design time: *Examine the meta data during design time.*. These similarly structured operators are (subgroup in parenthesis): *Set Role* and *Rename Attributes* (*Name and Role Modification*), *Type Conversion* and *Remap Binominals* (*Type Conversion*), *Select Attributes*, *Select Random Attributes*, *Add Noise*, *Generate Attributes*, *Generate Copy*, *Generate ID*, *Generate Rank*, *Reorder Attibutes*, *Principal Component Analysis* (*Attribute Set Reduction and Transformation*), *Normalize*, *Replace*, *Declare Missing Values* (*Value Modification*), *Replace Missing Values* (*Data Cleansing*), *Sample*, *Remove Duplicates*, *Filter Example Range*, *Filter Examples* (*Filtering*), *Sort* (*Sorting*), *Pivot Table* (*Rotation*), *Aggregate* (*Aggregation*).

Custom transformations may be implemented using the two of the most popular scripting languages on top of Hadoop: *Hive* and *Pig*. *Hive Script* and *Pig Script* (*Custom Script*) operators let you write your own code. Both can take multiple input data sets and Pig Script can deliver multiple output data sets. Both operators determine the meta data during design time, and also check whether the scripts contain syntax errors.

For combining multiple data sets (HadoopExampleSet objects) there are two operators: *Join* and *Union* (*Set Operators*). The Join operator implements the four type of join operations common in relational databases and may be familiar to you from SQL. The operator has two input ports and one output port, and can perform inner, left, right or full outer join. The Union operator takes arbitrary number of input data sets that have the same structure. The output data set is the union of these input data sets (duplicates are not removed).

You can read the description of every operator and their parameters on the Help View in the Design Perspective or you can browse the description at the end of this documentation, see *Appendix C - Radoop Operator Reference*.



Aggregate.example set output (example set output)
Meta data: Hadoop Data Table
Generated by: Aggregate.example set output ← Retrieve.output

Role	Name	Type	Range	Missings	Comment
	product...	nominal	unkown	?	
	sum_am...	real	unkown	?	
	count_d...	integer	unkown	?	
	custom...	nominal	unkown	?	

Press "F3" for focus.

Figure 3.3: Examine the meta data during design time.

3.4.4 Modeling

Besides the ETL operators, Radoop also contains operators for predictive analytics. All the modeling algorithms in this group use either the MapReduce paradigm or Spark to take full advantage of the parallelism of the distributed system. They are ready to scale with your large volume of data on the cluster. This section only lists the operators, Radoop's predictive analytics features are covered in *Advanced Radoop Processes*.

The operators in this group deal with exactly the same type of model objects as core RapidMiner operators do. Thus, a model output of a Radoop operator can be connected to the input port of a core RapidMiner operator and vice versa. Models trained on the cluster can be visualized exactly the same way as models trained on a data set in the operative memory. The same compatibility is true for *Performance Vector* objects that hold performance criteria values calculated to review and compare model performance. These objects can also be easily shared easily between operators that operate on the cluster and those that use the operative memory.

Clustering

This group contains three different clustering operators. All of them expect a HadoopExampleSet object on their input, and deliver the same HadoopExampleSet on their output with a new column added which contains the result of the clustering algorithms: clusters represented by nominal identifiers (*cluster_0*, *cluster_1*, *cluster_2*, etc.). The values indentify the rows that belong to the same cluster.

The input data must have an id attribute which uniquely identifies the rows. The attribute must have the "*Id*" role (use *Set Role* operator). If there is no id column on the data set, you can use the Radoop operator *Generate ID* for creating one.

Classification and Regression

Naive Bayes, *Decision Tree*, *Linear Regression* and *Logistic Regression* operators all implement distributed machine learning algorithms for classification or regression. Their prediction model is built on the input HadoopExampleSet on the cluster. The model can be applied both on data on the cluster (see *Application* operator group), or on data in the operative memory by the core RapidMiner operator, Apply Model.

Combine Models operator combines the trained prediction models on its input ports into a simple voting model - a Bagging Model.

Correlation and Dependency Computation

Correlation Matrix and *Covariance Matrix* operators calculate correlation and covariance between attributes in the input data set. Correlation shows how strongly pairs of attributes are related. Covariance measure how much two attributes change together.

Application

This group contains two operators: *Apply Model* and *Update Model*. Apply Model implements scoring: it applies a model on the data on its input port. This operator supports a wide range of RapidMiner prediction models. The model may have been trained by a RapidMiner modeling operator on an ExampleSet in the memory, or it may have been trained by a Radoop operator on the Hadoop cluster. The operator is able to apply both prediction and clustering models.

Update Model implements incremental learning. Radoop 2.3 supports this feature for Naive Bayes models. Incremental learning means that an existing model is modified - its attributes are updated - based on new data, new observations. Thus, the machine learning algorithm does not have to build a new model on the whole data set, it only updates the model by training on the new records. The operator's expected inputs are: the previously built model and a data set having the same structure as the one that the model was built on.

3.4.5 Evaluation

This group contains two operators for evaluating prediction results. Both *Performance (Binominal Classification)* and *Performance (Classification)* operators require HadoopExampleSet objects as input having both label and predicted label attributes. The operators compare these results and calculate perfomance criterion values. Several performance criteria can be selected (for parameters, see *Appendix C - Radoop Operator Reference*). The generated Performance Vector object that contains these values are, again, fully compatible with the I/O objects that core RapidMiner operators deal with.

3.4.6 Process Control

This group contains operators related to organizing the steps of the process. One subgroup is the *Loop* operator group. *Loop Attributes* iterates through the columns of a data set and performs an arbitrary Radoop subprocess in each iteration. General *Loop* operator performs its inner subprocess arbitrary number of times.

The *In-Memory Processing* subgroup contains operators that helps you combine the processing in the cluster with the processing in the memory. *In-Memory Subprocess (Sample)* fetches a sample from a data set in the cluster to the memory. It performs its inner subprocess on this sample data set in the memory. This subprocess can use any of the hundreds of RapidMiner original operators, as well as the ones from an extension. *In-Memory Processing (Full)* is similar, but it reads and processes the whole data set by performing iterations similar to the Sample version. I.e., it partitions the input data set into *n* chunks, then runs its in-memory subprocess on each of these chunks in a loop with *n* iterations. It may append the result data sets of these subprocesses to create an output data set on the cluster. *Advanced Radoop Processes* covers this in more details.

The *Subprocess* operator lets you run a process inside the process. This operator is a great help in designing clear and modular processes. The *Multiply* operator is required if you want to use the same HadoopExampleSet object in different branches of the process. The operator simply delivers the object on its input port to every connected output port.

3.5 Breakpoints

During process design you may want to examine intermediate results. You can define breakpoints before or after any operator and examine the objects on the input or output ports. For operators inside Radoop Nest, as they store data on the cluster, this means that a data sample will be fetched to the operative memory for each HadoopExampleSet object. The data can be examined the same way as for any other ExampleSet object. Meta data and data are shown in a tabular format and the Charts and Advanced Charts views provide highly configurable tools to visualize the data. You can control the maximum size of these data samples by setting a property (*rapidminer.radoop.sample_size.breakpoint*, see *Appendix A - Radoop Properties*).

Please note that using a breakpoint means that the data will be materialized on the cluster at the point where the breakpoint pauses the process. That means that the computation may take place earlier than without the breakpoint, and may use more space on the HDFS. For the same reason the total run time with a breakpoint may also be larger than the time required for the process to complete without breakpoints. This may be considered a standard debugging overhead.

CHAPTER

FOUR

DATA LOADING GUIDE

This section describes data loading scenarios for importing your data to your cluster. Ideally, your data already resides on the cluster. The import in this case does not necessary mean any movement of the data, only the creation of the objects through which Radoop may accesss them. But import may involve copying your data that resides on the cluster or import from your client machine. The data import can be part of the Radoop process you create.

The Radoop operators mentioned and shortly described in this section are: *Read CSV*, *Read Database*, *Store in Hive*, *Append into Hive*, *Retrieve from Hive*. If you are yet unfamiliar with the concept of operators and processes, please start with sections *RapidMiner Data Flow* and *Radoop Basics*.

Radoop primarily uses Hive as a data warehouse layer over Hadoop. (Since Radoop 2.0 it also supports Impala.) Radoop provides operators and wizards to define the structure of your data to be able to use them as inputs in your processes. It also provides an easy-to-use interface to manage the database objects, and quickly explore them using visualization tools.

4.1 Import Scenarios

When you design your import process, you may choose from the following basic scenarios:

- Define a Hive external table over your data on HDFS or on Amazon S3. No data movement is involved.
- Define a Hive managed table over your data on HDFS. Source data will be copied to the HDFS directory structure managed by Hive. You can specify custom storage format for the target table.
- Define a Hive managed table for your local data on your client machine. The data from the local machine will be copied to HDFS into the directory structure managed by Hive. This can by far be the slowest operation out of these options.

Regarding the execution of the import, you have two options:

- Launch the import immediately. Open the Import Configuration Wizard, describe the source and the target object, then launch the import immediately. This is only is recommended for external tables (created instantly), or smaller HDFS sources, or even smaller local sources. The import configuration will not be saved and the Radoop client will wait until the operation finishes. A progress bar will show the progress of the operation.
- Create an import process. Use the Design perspective to create a process that includes one or more data import operation. Use an operator from the *Radoop / Import operator* group. Read CSV operator opens the same Import Configuration Wizard as the stand-alone import. Read Database operator opens a similar import wizard for the configuration of a data import from an external database that the client accesses (the data will stream through the client machine.) The import operation will be performed when you run the process you designed. Choose this option if you may want to reuse the configuration, do periodic imports, schedule or share the import process with others.

There exists another scenario of pushing data to the cluster besides these explicit import operations. That is, when you load a data set (an ExampleSet) of a RapidMiner process to the cluster (from the client's operative memory). This is done automatically by the *Radoop Nest* operator chain: every ExampleSet connected to any of its input ports is pushed to the cluster, and becomes automatically available to any operator inside the Nest that expects a data set as input.

The following section guides through the steps of the import wizard. The wizard and the steps are basically the same for a local source and for a file or directory on the cluster.

4.1.1 Import Wizard

The easiest way to access the import wizard is using the Hive Management Perspective. After the completion of the steps (see details below), Radoop will immediately import the data to your cluster. If you have more than one cluster connections defined, Radoop may ask, which connection you want to use for the import. The import may take a lot of time depending on the size of your data and the connection to the cluster.

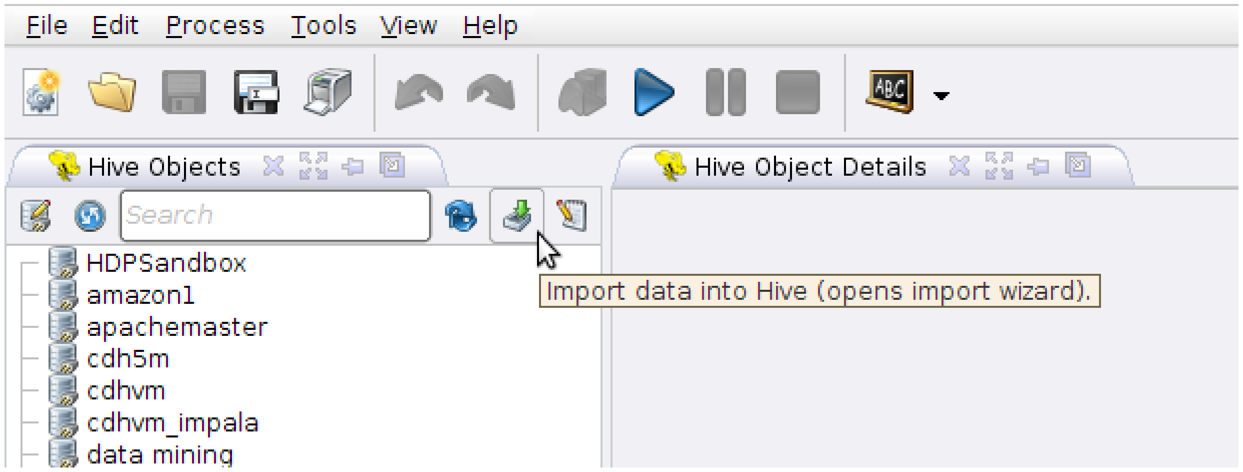


Figure 4.1: Open the Import Wizard from the Hive Management Perspective

You can access the same wizard if you create a new process that has a *Read CSV* operator (from the *Radoop* operator group) inside a *Radoop Nest* operator. Clicking on the *Import Configuration Wizard* in the Parameters view opens the same wizard. In this case, completing the steps will only define the import parameters to the operator, the import itself will only be performed when you run the process. The configuration that you defined using the wizard will be saved in the process file, when you save your process. Thus, you are encouraged to create a process, if the import procedure may have to be repeated later.

The wizard guides you through the following steps to describe the source file and define the destination structure on Hive (and in RapidMiner).

Step 1 of 4. You may choose between three kinds of source location: import from the client's local filesystem, import from HDFS directly or import from Amazon S3. The *Choose local file* / *Choose hdfs file/directory* button opens a file browser to select the input file or directory. For an HDFS source, you may want to create an external table instead, which you can accomplish by checking the *Create external table (do not move the data)* checkbox. For an external table, you have to select a source directory. The content of the files in this directory will be the content of the external table. This is also currently the only option when importing from Amazon S3. For a non-external (managed) tables, you can select a single file or a complete directory as a source.

Step 2 of 4. Configure the format of the input data. Select the encoding, the column separator (use a regular expression if needed), and set other options, such as reading attribute/column names from the first line. Based on the settings you provide, you can check a preview of the parsed data in the *Data preview* pane. If you are satisfied with the result, please continue to the next step.

Step 3 of 4. RapidMiner uses strongly typed attributes. This step helps you defining the attributes of the table you will create on Hive. The table corresponds to an ExampleSet in RapidMiner (a so-called HadoopExampleSet, to be precise). The columns of the table are attributes in this ExampleSet. The wizard will automatically guess the attribute types based on the values it finds in the beginning of the file. The allowed data types are: *real*, *integer* and *nominal*. Real and integer attributes will be stored in Hive as DOUBLE and BIGINT columns, while nominal attributes are stored as STRING columns. Radoop does not explicitly support other types, however, you can load, for example, date values into a nominal attribute without any problem. You can later process them with Radoop operators like Generate Attributes, and also use any of Hive's numerous date functions on them.

You can also assign *roles* to the attributes, defining what they can be used for by individual operators. You can set these roles for the attributes later in the process using the *Set Role* operator. If you are unfamiliar with any concept mentioned in this step and want to know more before continuing, please read about them in sections *RapidMiner Data Flow* and *Radoop Basics*.

Step 4 of 4. As a last step, you choose a name for the destination Hive table, and optionally change the storage format. If you opened the wizard from the Parameters view of a Read CSV operator, you can also select whether you want to store the data in a temporary or a permanent table. The first option is a good choice for you, if you want to immediately process the data using other Radoop operators, and do not need the original data in this format later (nonetheless you can store the processed data with *Store in Hive* operator).

You can optionally change the default storage format (the default is defined by Hive, usually it is *textfile*). You can use partitioning and have several options for the file format. Partitioning may enhance the performance of queries that filter for the partioning column. Special storage formats may decrease the required space and increase the performance. Please note that choosing a special format may increase the import running time as the conversion may require additional jobs to run. Please refer to the Hive Documentation for the details of the storage types.

After you have imported your data to the cluster, you can easily access, query, visualize it using the Hive Management Perspective. Read more about the interface for managing your data on Hive in *Appendix B - Hive Management Perspective*. You can access the imported data in a process using the *Retrieve from Hive* operator. Right click on the object in this perspective and click on *Create Process: Retrieve* to instantly create a new Radoop process that uses this data as an input.

If you opened the wizard for defining the parameters of a Read CSV operator on the Design Perspective, then you can immediately explore the meta data that appear on the output port of the Read CSV operator, just as with the *Retrieve from Hive* operator after selecting the input table you created.

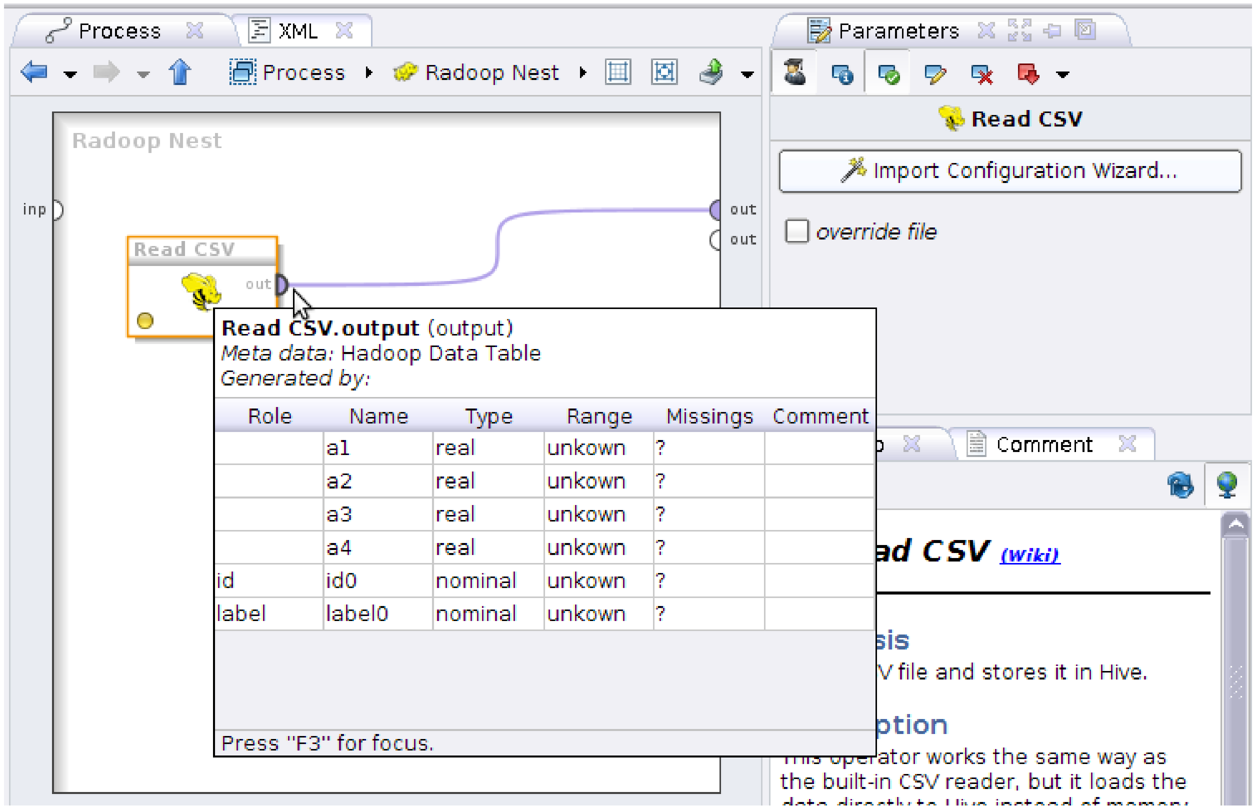
4.1.2 Read Database

Radoop can also stream data to the cluster from a database. The client reads from the defined database connection, and pushes the data instantly to the cluster. The source can be a database object, but also the result of a query that you define in a parameter. The operator supports connections to MySQL, PostgreSQL, Sybase, Oracle, HISQLDB, Ingres, Microsoft SQL Server or any other database using an ODBC Bridge.

Please note that the software may miss the closed source JDBC driver for certain database systems. This only means that you have to download the driver file yourself and configure it by following the instructions on the Rapid-I-Wiki.

4.1.3 Load From the Memory

You can also load data from the client machine's operative memory in a process by connecting a RapidMiner operator that delivers an ExampleSet object on its output to a Radoop Nest operator input. The data set on the wire will be pushed to the cluster. The processing continues within the Nest. These concepts are explained in greater detail in the section *Radoop Basics*.


Process
XML
Parameters
Process
Radoop Nest
Read CSV
Import Configuration Wizard...
override file
Radoop Nest
inp
out
out
Read CSV
out
Comment
ad CSV (wiki)
sis
V file and stores it in Hive.
ption
This operator works the same way as the built-in CSV reader, but it loads the data directly to Hive instead of memory

Read CSV.output (output)
Meta data: Hadoop Data Table
Generated by:

Role	Name	Type	Range	Missings	Comment
	a1	real	unkown	?	
	a2	real	unkown	?	
	a3	real	unkown	?	
	a4	real	unkown	?	
id	id0	nominal	unkown	?	
label	label0	nominal	unkown	?	

Press "F3" for focus.

Figure 4.2: Meta data on the output port.

4.2 Access Data on Hive

You can save the current data set in your process with the help of the *Store in Hive* operator. The data inside a Radoop Nest resides always on the cluster, hence, storing effectively means only saving the structure and the data itself in a Hive table for later use, i.e. Radoop does not delete it after the process completes. The storing itself does not last for a noticeable time, however, deferred calculations may have to be completed before it can complete.

Retrieve from Hive is the operator for accessing Hive tables. It delivers the data from the selected table on its output port. The data resides on the cluster until you connect an output port of a Radoop operator to the output of the Radoop Nest. In this case, a sample from the data will be fetched to the client machine's operative memory.

Append into Hive is similar to Store in Hive operator, if the target table does not yet exist. If it exists, the operator first checks if its structure is the same as the stucture of the data set on its input port. If they match, then the data will be inserted into the destination table. The current content of the destination table will also be kept.

For further details on processes and operators, see sections *RapidMiner Data Flow* and *Radoop Basics*.

CHAPTER

FIVE

ADVANCED RADOOP PROCESSES

5.1 Preface

This chapter discusses advanced Radoop process design. You should be familiar with chapters *RapidMiner Data Flow*, *Radoop Basics* and *Data Loading Guide*. The following sections introduce predictive analytics capabilities of the software. You will see how smoothly Radoop integrates into the RapidMiner client interface by using the same I/O objects, meta data and the same process design concepts (learn, evaluate, etc). However, Radoop implements the operations on the Hadoop cluster, so it is able to deploy your process (or the intended part of it) to the cluster and deal with enormous data volumes using scalable distributed algorithms.

5.2 Model Objects

Data mining methods, primarily supervised learning algorithms create prediction models. Other operators may generate other types of models, but in this chapter we mainly consider prediction and clustering models. These models describe the information explored from a training data set. A RapidMiner model is an I/O object - *Prediction Model*, *Cluster Model* or *Preprocessing Model* - just like an ExampleSet object. The visualization of the model may present the explored information to the user.

We can apply the model on another data set that has a similar schema as the training data. This is also called *scoring*, while the input data set is sometimes simply called the test data set. The target attribute has the role called the *label*. If it is a numerical attribute, we the procedure is a regression. If it is nominal, then it is a classification, and further attributes are generated besides the prediction, namely, confidence values for the possible classes. A confidence for a class is a value between 0 and 1 that indicate the probabilitiy that the record belongs to this class.

Radoop and core RapidMiner operators use - train and apply - the same type of model objects. More precisely, Radoop implements the scoring and learning of some popular RapidMiner models over the distributed infrastructure. This is true for prediction models, but also true for the preprocessing models that a preprocessing operator, for example, Add Noise generates (Noise Model).

Another type of I/O object related to modeling is the *Performance Vector* object. Prediction, hence, models are evaluated by comparing the expected target attribute values to the result of the scoring. Then performance criteria create measures to describe the validity of the prediction (model) from the viewpoint of the goals of the data mining process. Radoop performs these calculations on the cluster as well.

5.3 Data mining scenarios

This section reviews the possible scenarios for your predictive modeling process.

- Unsupervised learning on the cluster. Clustering of the data in the cluster using scalable distributed algorithms. Some clustering algorithms build clustering models that you can apply on other similar data sets either on your cluster or in the memory (look for the *cluster model* output).
- Supervised learning in the memory, scoring on the cluster. Core RapidMiner modeling (classification and regression) operators build predictive models that you can apply on your data in the cluster. Most popular prediction models are supported.
- Supervised learning and scoring on the cluster. The learning and the scoring are both implemented on the distributed platform. Radoop supports the following algorithms: Naive Bayes, Linear Regression, Logistic Regression, Decision Tree. Iterative building of a Naive Bayes model is also supported, which means that a model (built on the cluster or in the memory) can be updated by new data (on the cluster or in the memory).

5.3.1 Clustering

Radoop clustering operators are built on the distributed machine learning algorithms of the Apache Mahout project. The input of a clustering operator is a HadoopExampleSet object. The algorithm creates the specified number of segments in the data. It adds a new attribute to the data set named *clusterid* that has the *cluster* role. The attribute contains the cluster identifier: *cluster_1*, *cluster_2*, etc.

The clustering operators may also indicate that a record is so distant from the rest of the records, that it should be considered an outlier. In this case, the cluster attribute contains the value: *outlier*.

KMeans and *FuzzyKMeans* algorithms also build a so-called centroid clustering model. You can use this model to cluster similarly structured data sets that resides either on your cluster, or in the memory.

5.3.2 Classification and Regression

Prediction and cluster models trained by core RapidMiner operators can be connected to the input port of a Radoop Nest and used inside its subprocess. The *Apply Model* Radoop operator takes such a model and a test data set as inputs. The operator applies the prediction model on the HadoopExampleSet input and delivers a HadoopExampleSet with new prediction and confidence columns. In case of a cluster model a cluster column is generated.

Radoop (since version 2.2) supports the scoring on the cluster for all RapidMiner prediction and cluster models. These models produce the same prediction inside and outside the Nest on the same data sets, but the scoring inside the Nest is not limited to the operative memory size, but scalable over the distributed platform.

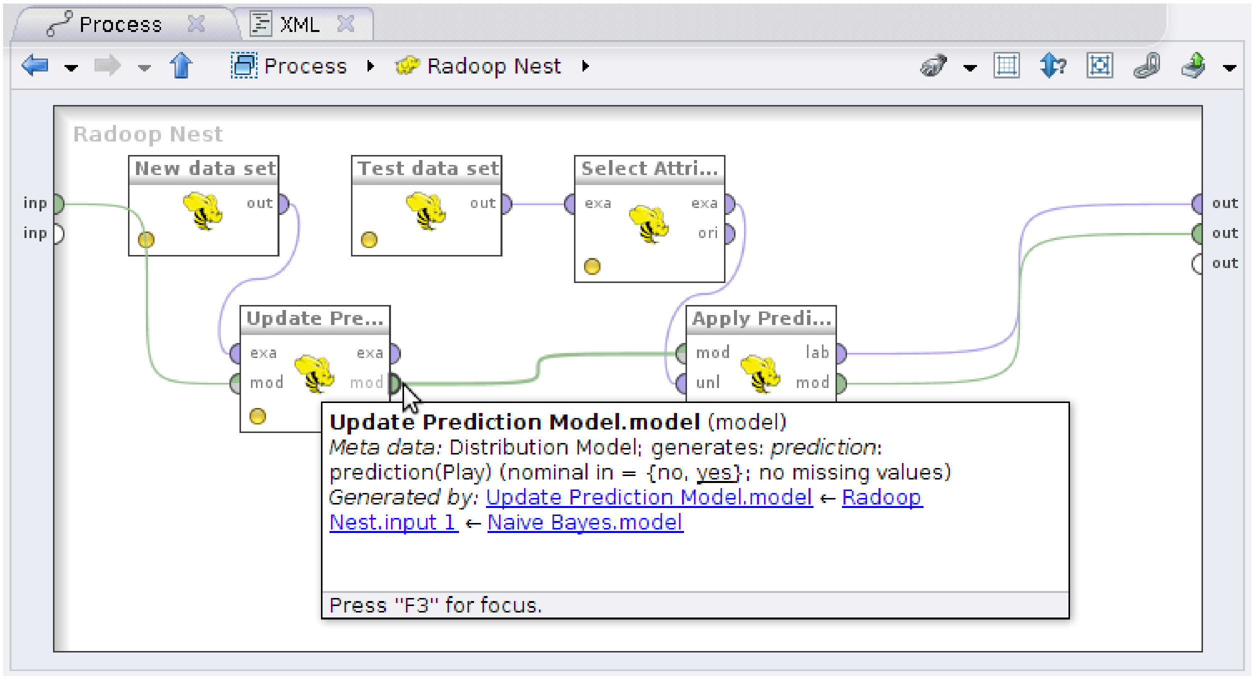
You can also train a prediction model on the distributed platform. Radoop 2.3 supports Naive Bayes, Linear Regression, Logistic Regression and Decision Tree learning algorithms. Naive Bayes and Decision Tree classification can handle both numerical and nominal attributes, Linear Regression and Logistic Regression handles numerical attributes. The target attribute (class) can be polynominal for Naive Bayes, binominal for Logistic Regression and Decision Tree, and numerical for Linear Regression. Radoop contains a unique, linearly scalable implementation for Naive Bayes, and integrates the Spark MLlib implementations for Linear Regression, Logistic Regression and Decision Tree. The built model can be applied exactly the same way as a model built using a core RapidMiner learning operator. The Naive Bayes algorithm is also ideal for iterative learning.

5.3.3 Iterative Learning

Iterative learning is a special type of learning for classification or regression tasks. The model is *updated* with a new training data set in each iteration. We do not have to rebuild the model from scratch on the whole data set, because the much faster update with the new records gives exactly the same model, as if we had built it on the whole data set. This is a very common scenario for big data analytics, e.g. new log records may update a prediction model periodically.

Radoop implements this concept with the *Update Model* operator. This operator takes a model as input (a *Distribution Model* that was built with Naive Bayes) and updates it by training on the HadoopExampleSet on its other input port.

The input model may have been trained either by a Radoop learner operator or by a core RapidMiner learner. The new data set must have exactly the same schema as the original training data set had. That means, the name and type of the attributes must match, but their ordering is irrelevant. As the learning algorithm of the Update Model operator may have parameters like optimization parameters, this operator has a special parameter, where you can easily specify model type specific parameters.



Update Prediction Model.model (model)
Meta data: Distribution Model; generates: *prediction*:
prediction(Play) (nominal in = {no, yes}; no missing values)
Generated by: Update Prediction Model.model ← Radoop Nest.input 1 ← Naive Bayes.model

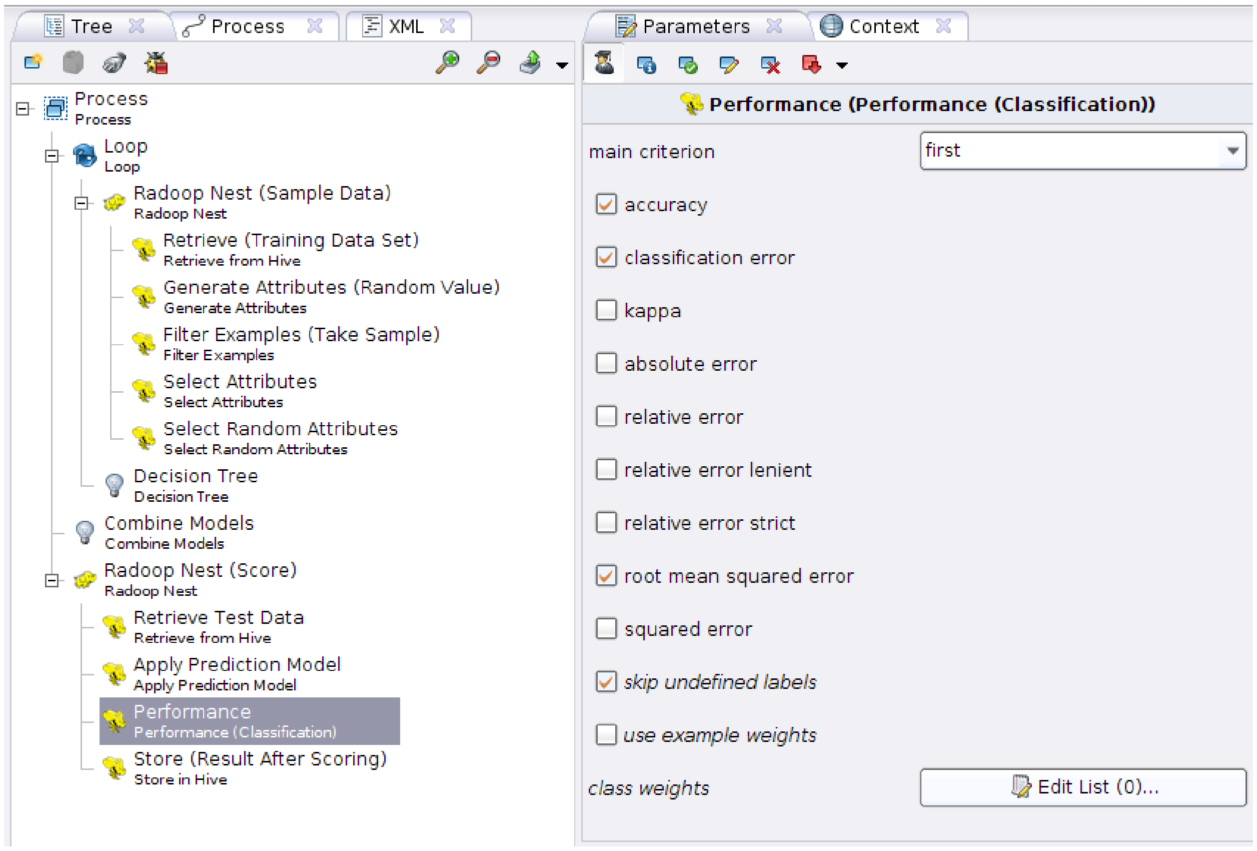
Press "F3" for focus.

Figure 5.1: Model update and apply in a single subprocess

5.3.4 Ensemble Modeling

Radoop contains a meta operator called *Combine models*. This operator can be used both outside the Radoop Nest or inside it. It simply creates a *Bagging Model* - a simple voting model - out of the trained models on its input (all input models have to be trained on data with similar schema). The input may be a *Collection* of models or arbitrary number of models on the input ports. The ensemble model on the output port will be a voting model. If this model is used for scoring, it applies all inner models, then takes the average of confidence values in case of classification to decide the predicted class. In case of regression (numerical label), it calculates the average of the inner models' prediction.

You may want to use Combine models together with a training loop if you have too much training data on the cluster and for some reasons, the provided learning algorithm does not fit your use case. You can fetch random samples from your data to the memory and train different models on these samples in the memory, then combine these models into a Bagging Model. You can then apply this model directly on your data on the cluster. This way you can not only utilize RapidMiner's wide range of learning algorithms, but you can build a very precise and robust ensemble model in limited time using only samples of data. You may even consider using only a subset of the regular attributes in each training iteration to build a more precise voting model. *Random Attribute Filter* operator is ideal for this task. One of RapidMiner's *Decision Tree* learning algorithms may be a good choice for the training algorithm: this way, you can implement a *Random Forest* learning algorithm, where you control the sample size, the number of iterations and the randomization method based on the resources available (memory, bandwidth, amount of data).



- Process (Process)
  - Loop (Loop)
    - Radoop Nest (Sample Data) (Radoop Nest)
      - Retrieve (Training Data Set) (Retrieve from Hive)
      - Generate Attributes (Random Value) (Generate Attributes)
      - Filter Examples (Take Sample) (Filter Examples)
      - Select Attributes (Select Attributes)
      - Select Random Attributes (Select Random Attributes)
    - Decision Tree (Decision Tree)
  - Combine Models (Combine Models)
  - Radoop Nest (Score) (Radoop Nest)
    - Retrieve Test Data (Retrieve from Hive)
    - Apply Prediction Model (Apply Prediction Model)
    - Performance (Performance (Classification))
    - Store (Result After Scoring) (Store in Hive)

Performance (Performance (Classification))

main criterion	first
accuracy	✓
classification error	✓
kappa	☐
absolute error	☐
relative error	☐
relative error lenient	☐
relative error strict	☐
root mean squared error	✓
squared error	☐
skip undefined labels	✓
use example weights	☐
class weights	Edit List (0)...

Figure 5.2: Process implementing a Random Forest training algorithm on data samples (Loop with 100 iterations)

5.3.5 In-Memory Subprocess

In-Memory Subprocess (Sample) and *In-Memory Subprocess (Full)* implements the concept described in the previous section in a more general way. These operators are very useful for training ensemble models, but the In-Memory Subprocess (Full) operator is also capable of doing arbitrary preprocessing tasks.

In-Memory Subprocess (Sample) is a meta operator inside the Radoop Nest that runs its subprocess on a data sample in the operative memory. A sample is fetched from the HadoopExampleSet on its input port to the memory, and the subprocess - consisting of arbitrary RapidMiner operators - is applied on this data. Then the data is written back to the HDFS to be further processed by the rest of the Radoop process. Other I/O objects, like a model trained on this sample data, can be connected to the output port and used on the cluster later.

In-Memory Subprocess (Full) differs in the following way: it processes all of the data on its input port in small partitions that fit into the operative memory. You must either define the number of these partitions that the input data must be splitted into, or the size of one such partition. The meta operator performs a loop on these partitions, and in each iteration it reads the data in the current partition to the memory, runs the subprocess on it, then writes the data back to the HDFS (appends it to the output table).

There are basically two different scenarios for using this meta operator. The first one is when you want to do some kind of special preprocessing on the data that you can not implement with Radoop operators, but you can do it with core RapidMiner operators or operators from another extension. In this case with this meta operator you can stream your data through the client's operative memory while preprocessing it. As the operative memory and the network bandwidth may put a too strong limit on doing this, you should probably only do this via the RapidMiner Server for larger data sets. Your server may have enough memory and a good connection to process larger input data in this way (larger partitions mean fewer iterations).

The second use case is when you build some kind of object or objects using these splits, and you do not need to write the data back to the cluster. An example for this is a similar process to the Random Forest example in the previous section: you can train as many decision tree models as many partitions you choose to have, then you can combine these model to a voting model with the Combine Models operator. Just connect the model built inside the In-Memory Subprocess (Full) meta operator to the output, then connect the *Collection of Decision Tree* output of the meta operator to the Combine Models input. This way, you train a model on data that is much larger that the operative memory size, but you can still expect this voting model to have quite good precision.

5.4 Evaluation

The purpose of performance evaluator operators is to provide simple measures for evaluating the current prediction and the prediction model itself. Analysts can define the target measures that they want to optimize for. Models can be compared using these performance criteria.

Radoop implements measures for evaluating binominal and measures for polynominal classification. These form a subset of the set of criteria that core RapidMiner evaluation operators implement. *List of performance criteria* describes these measures.

Table 5.1: List of performance criteria

Binominal or polynominal	Criterion name	Criterion description
both	accuracy	Relative number of correctly classified examples
both	classification_error	Relative number of misclassified examples
both	kappa	The kappa statistics for the classification
binominal	precision	Relative number of correctly as positive classified examples among all examples classified as positive
binominal	recall	Relative number of correctly as positive classified examples among all positive examples
binominal	lift	The lift of the positive class
binominal	fallout	Relative number of incorrectly as positive classified examples among all negative examples
binominal	f_measure	Combination of precision and recall: f=2pr/(p+r)
binominal	false_positive	Absolute number of incorrectly as positive classified examples
binominal	false_negative	Absolute number of incorrectly as negative classified examples
binominal	true_positive	Absolute number of correctly as positive classified examples
binominal	true_negative	Absolute number of correctly as negative classified examples
binominal	sensitivity	Relative number of correctly as positive classified examples among all positive examples (like recall)
binominal	specificity	Relative number of correctly as negative classified examples among all negative examples
binominal	youden	The sum of sensitivity and specificity minus 1
binominal	positive_predictive_value	Relative number of correctly as positive classified examples among all examples classified as positive (same as precision)
binominal	negative_predictive_value	Relative number of correctly as negative classified examples among all examples classified as negative
binominal	psep	The sum of the positive predictive value and the negative predictive value minus 1
polynominal	absolute_error	Average absolute deviation of the prediction from the actual value
polynominal	relative_error	Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
polynominal	relative_error_lenient	Average lenient relative error (average of absolute deviation of the prediction from the actual value divided by maximum of the actual value and the prediction)
polynominal	relative_error_strict	Average strict relative error (average of absolute deviation of the prediction from the actual value divided by minimum of the actual value and the prediction)
polynominal	root_mean_squared_error	Averaged root-mean-squared error
polynominal	squared_error	Averaged squared error

CHAPTER

SIX

APPENDIX A - RADOOP PROPERTIES

This appendix explains the properties on the Preferences dialog that influences Radoop's operation. You can find these properties on the *Radoop* tab. All of the keys start with *rapidminer.radoop*.

Key	Default value	Description
sample_size.breakpoint	1000	Sample size for Hadoop data sets after breakpoints, zero means full sample. When you pause a Radoop process using a breakpoint, a sample of the processed data will be fetched into the client machine's memory to be manually examined. You can define the size of the sample (number of rows) with this value.
sample_size.overall	200000	Sample size for Hadoop data sets on Nest output, zero means full sample. When data arrives onto the output of the Radoop Nest, it will be fetched into the client machine's memory. You can limit the size of the data (sample) with this parameter.
hdfs_directory	/tmp/radoop/	HDFS directory for temporary files. Defines the path in which Radoop stores temporary files on the cluster. The client user that runs Radoop must have permission to create this directory if it does not exist, and/or must have read and write permission on the directory.
connection.timeout	11	Timeout, in seconds, for the Radoop connection, zero means default value (11). This setting defines the time after which Radoop may cancel a connection test (and consider it failed). You may increase this value, if the connection latency is high, or may vary in a larger interval.
hive_command.timeout	17	Timeout, in seconds, for simple Hive commands that should return in seconds, zero means default value (17). This setting defines the time after which Radoop may cancel an atomic operation on the cluster, since the command should finish in a few seconds at most. You may increase this value, if the connection latency is high or may vary in a larger interval.
table.prefix	Radoop_	Default Hive temporary table prefix for new processes. You can specify this prefix as a parameter of a Radoop Nest operator, this property only defaults this parameter to the specified value, so different clients or users may easily differentiate their temporary objects on the cluster.
describe.max_errors	5	Hive Management Perspective considers a connection failed if it encounters more errors during describing Hive objects than this limit. You may have to increase this value, if you have many objects in Hive having errors when described (e.g. missing custom input/output format classes).
fileformat.hive	Default format	Storage format default for Hive connections. The storage format is generally defined by the Radoop Nest parameter *hive_file_format*, but this property sets a default for this parameter in new Radoop Nests. It also defines the default settings for new table import on the Hive Management Perspective. 'Default format' means use Hive server default (usually TEXTFILE).
fileformat.impala	Default format	Storage format default for Impala connections. The storage format is generally defined by the Radoop Nest parameter *impala_file_format*, but this property sets a default for this parameter in new Radoop Nests. It also defines the default settings for new table import on the Hive Management Perspective. 'Default format' means use Impala default (usually TEXTFILE).
auto_describe	false	Hive Management Perspective - Automatically describe all Hive objects after connection or refresh if this is enabled. This property saves the state of the toggle button on the Hive Management Perspective. If enabled, all metadata of your Hive objects are immediately fetched, which may be slow if you have a lot of objects.

CHAPTER

SEVEN

APPENDIX B - HIVE MANAGEMENT PERSPECTIVE

Hive Management perspective is part of Radoop, it is an easy-to-use client for managing your data on the Hive server. You can browse, manage database objects, execute SQL statements, fetch data samples from objects or query results, and plot the data using advanced plot charts.

Please note that this perspective, as well as a Radoop process, can also connect to and work with Impala exactly the same way as with the Hive server. An Impala connection is expected to be much faster than Hive.

7.1 Object Browser

Hive Object Browser is a browser for your data on the cluster. Hive Management perspective may connect to multiple clusters at the same time. Active connections are displayed with a bold font. You can open the Connection Setup dialog anytime to modify the connection settings or add new connections.

After you connect to a cluster and Radoop succeeds in testing it, the Hive object list will be retrieved. If the **Auto describe** mode is enabled, then the client immediately retrieves the details of all objects. This may take some time if you have a lot of objects in Hive. This automation is disabled by default, but you can enable it with the button that is left to the filter text field. When disabled, however, the fetching of the object list is very fast, but the type and the attributes of a Hive object will only be visible if you expand the object or open the action menu by a right click on the object.

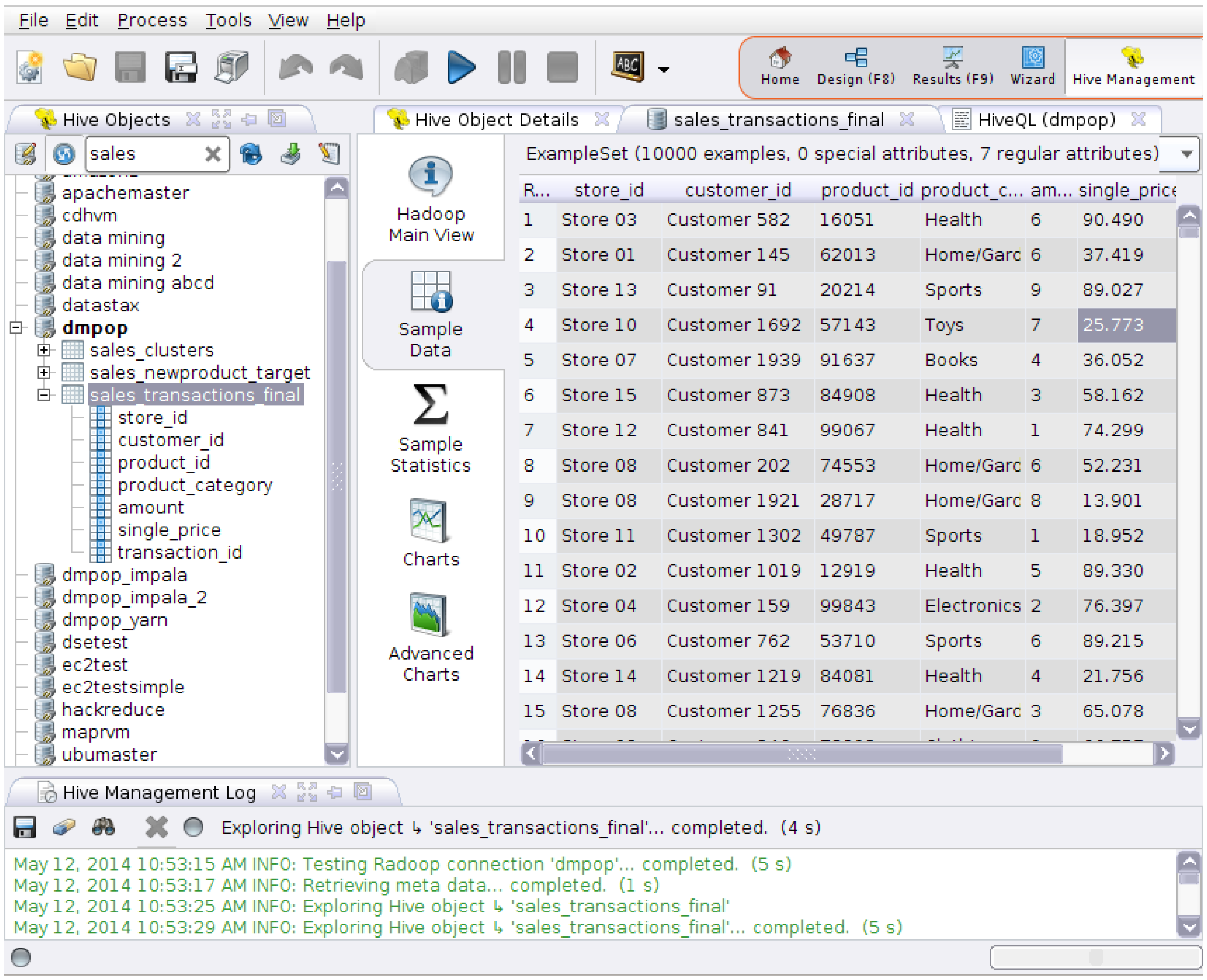
Hive tables and views are shown together with their attributes (you can expand and collapse the connection entries as well as the Hive objects). Use the filter field to show only the objects that contain the search word you enter. The icon appearing on the right during typing clears the filter. The filter applies to all connections.

The remaining three buttons are used to refresh objects, import data and open up a query window, respectively. You can read more about these actions in the following subsections. These operations, as well as others, are accessible also in the popup menu visible after a right-click on a connection or on a Hive object or on the empty space in the Object Browser. You can select multiple objects or connections by holding down the SHIFT key or the CTRL key while using the mouse buttons or the arrow keys for selection.

Connect. To browse the database objects of your Hive instance, double click on the selected connection name, or select *Connect* from the popup menu after a right-click on the connection. Radoop first tests the connection, and after a successful test, it starts to retrieve the metadata (object list) from Hive. The tables and views will appear on the tree. You can explore, query, rename or delete any of these objects using the tree panel.

Refresh Objects. Clears, then refreshes the object list and the meta data of the objects from the Hive server of the selected connection(s). When no connection selected, all active connections are refreshed. You can use the *Refresh* button on a single object as well.

Invalidate Metadata (*Only available for Impala connections*). In contrast to the single Hive server, there are usually multiple Impala Daemons. Every change you do to objects using the Impala connection will be immediately reflected in Hive. However, changes through the Hive connection (the Hive server) are not immediately visible through the



ExampleSet (10000 examples, 0 special attributes, 7 regular attributes)

R...	store_id	customer_id	product_id	product_c...	am...	single_price
1	Store 03	Customer 582	16051	Health	6	90.490
2	Store 01	Customer 145	62013	Home/Garc	6	37.419
3	Store 13	Customer 91	20214	Sports	9	89.027
4	Store 10	Customer 1692	57143	Toys	7	25.773
5	Store 07	Customer 1939	91637	Books	4	36.052
6	Store 15	Customer 873	84908	Health	3	58.162
7	Store 12	Customer 841	99067	Health	1	74.299
8	Store 08	Customer 202	74553	Home/Garc	6	52.231
9	Store 08	Customer 1921	28717	Home/Garc	8	13.901
10	Store 11	Customer 1302	49787	Sports	1	18.952
11	Store 02	Customer 1019	12919	Health	5	89.330
12	Store 04	Customer 159	99843	Electronics	2	76.397
13	Store 06	Customer 762	53710	Sports	6	89.215
14	Store 14	Customer 1219	84081	Health	4	21.756
15	Store 08	Customer 1255	76836	Home/Garc	3	65.078

Exploring Hive object ↳ 'sales_transactions_final'... completed. (4 s)

```
May 12, 2014 10:53:15 AM INFO: Testing Radoop connection 'dmpop'... completed. (5 s)
May 12, 2014 10:53:17 AM INFO: Retrieving meta data... completed. (1 s)
May 12, 2014 10:53:25 AM INFO: Exploring Hive object ↳ 'sales_transactions_final'
May 12, 2014 10:53:29 AM INFO: Exploring Hive object ↳ 'sales_transactions_final'... completed. (5 s)
```

Figure 7.1: The three main panels on Hive Management Perspective: the Object Browser, the Management Log and the Object Details view

Impala connection. You must explicitly call an Invalidate Metadata command to update Impala with the metadata in the Hive Metastore Server. After the completion of this action, every Hive object will be available in Impala.

7.2 Explore Objects

Explore. Exploring a Hive table means that Radoop fetches a data sample from the table to the client machine's operative memory, and displays it in a tabular format. This format may be familiar to you from the Results Perspective, where you can explore an ExampleSet process output. It also allows you to plot the data and create advanced charts out of the sample. You can control the data sample (maximum) size with the help of a property (see *Appendix A - Radoop Properties*), or use **Explore first N rows** action, and define the number of rows explicitly.

Show query. A Hive view is a stored SQL query based on other tables or views. You can examine this query using this action. *Exploring* a Hive view is similar to fetching data from a Hive table with the exception that the server will first execute the query of the view - the required time depends on the complexity of this task -, before reading the data sample to the client machine. The result can be examined the same way as a data sample from an ordinary Hive table.

Count Rows. Counting the number of rows may take some time. The result will be shown to you in a small popup

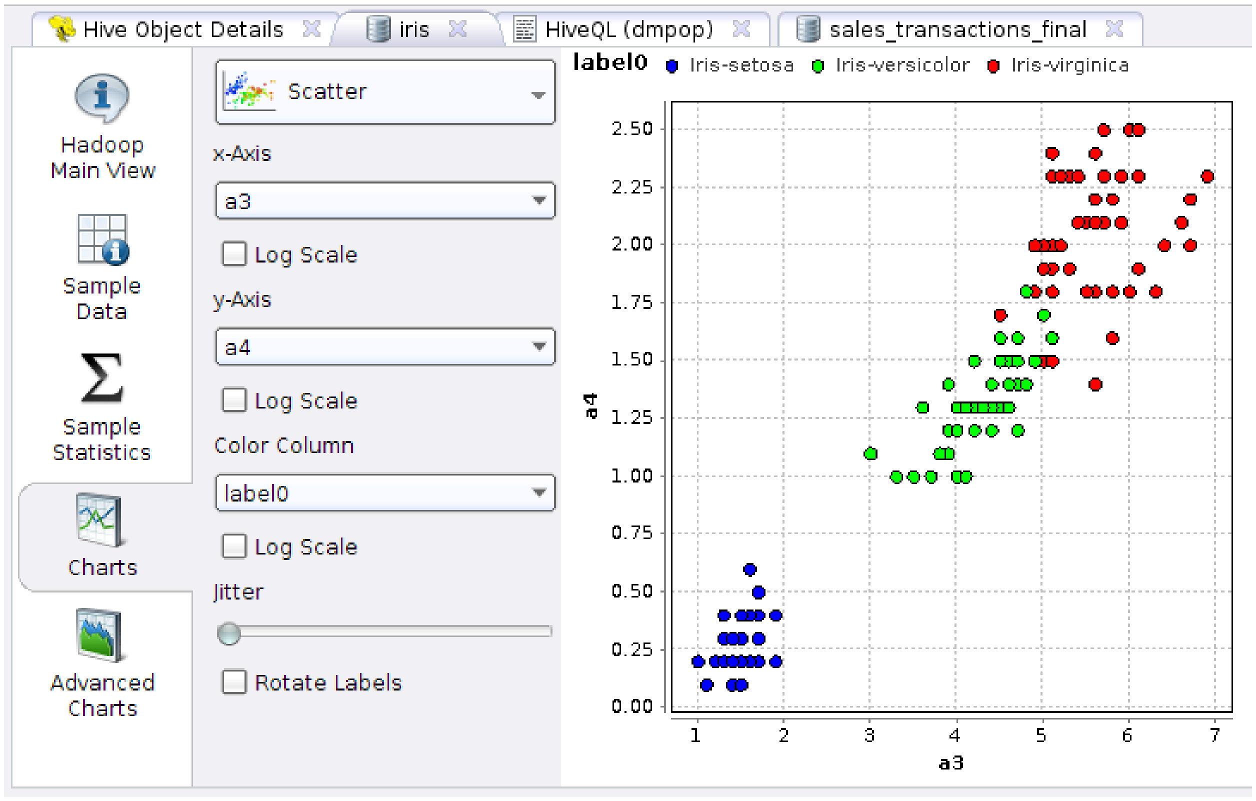


Figure 7.2: Visualize your data (sample) with a few clicks

window.

Drop and **Rename**. A Hive table or view can be dropped or renamed easily. Please note, that dropping a Hive object cannot be undone. You can also rename an attribute of a Hive table.

During the run of a process, Radoop creates temporary Hive tables and views. The name of these temporary objects has the prefix that you define in the Radoop Nest *table prefix* parameter (`Radoop_` by default). These objects are deleted by the end of the process if you set the Radoop Nest parameter *cleaning* to true (default value). However, due to breakpoints or errors, some temporary objects may remain on the cluster even when *cleaning* is set to true. You can easily delete them using the Object Browser. Use the filter field to show only these temporary objects by entering the prefix, then use the SHIFT key to select all of them. Remove the selected objects with the DEL button or the Drop Objects action in the right-click popup menu.

7.3 Import and Create Process

Import. Import data to the cluster with the Data Import Wizard. You can select a text file on your local file system, on the HDFS or on Amazon S3 and import its contents into a Hive table on the cluster. Define the column separator, encoding and other settings as well as the target attribute types and table name. The wizard is basically the same as the wizard for the *Read CSV* Radoop operator, but with this standalone importer you do not create a process for this operation. However, if the import is a recurring task, then creating a process for it might be a better practice.

Create Process: Retrieve: This action leads to a good starting point for process design. It creates a simple Radoop process which starts with a *Retrieve* operator inside a Radoop Nest. You can then go on with the design of your process that processes the data in this table or view.

7.4 Execute SQL

Execute query.... You can execute arbitrary valid SQL (HiveQL) statement against the selected Hive instance. If the statement is a query, then Radoop will fetch a data sample from the result to the client machine's memory. You can examine and plot the data using the graphical interface. You can change the default data sample size (limit) before executing the query. You can also execute valid DDL or DML statements the same way.

You can open the *Hive Expression Editor* dialog to create an expression for a column in a SELECT statement with an easy-to-use expression builder. The editor contains the numerous Hive functions and operators with their argument list and short description. It is recommended to validate your more complex queries with the *Check Expression* button before sending it to the Hive instance with the *Run Query...* button. (Of course, a successful check does not guarantee that the query will succeed.)

You can write multiple SQL statements to the query text field. Separate them by a semicolon and Radoop will automatically select (validate or execute) the one that is under the cursor. Both the Run Query... and the Check Expression actions will apply on this single statement under the cursor. If you want to run multiple statements (separated by semicolons), use the *Run All Statements...* button. This action assumes that the last statement may be a query, hence, if that statement returns a result set, it will be displayed.

During query execution, you may choose to cancel the run of the query using the *Cancel Query* button. This will send a kill command to the cluster to stop all jobs that this query has initiated.

For a complete documentation of the SQL-like Hive Query Language please refer to the Hive Language Manual.

7.5 Management Log view

Management Log view shows information about ongoing operations. You can search and save the log text the same way as the process Log.

You can cancel any action using the *Cancel* button. Hitting this button will try to stop (kill) all running remote operations on the cluster. This may take a few seconds to complete.

CHAPTER

EIGHT

APPENDIX C - RADOOP OPERATOR REFERENCE

8.1 Evaluation - Performance Measurement

8.1.1 Performance (Binominal Classification)

Synopsis

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 *Performance (Classification)* operator. This operator expects a test *HadoopExampleSet* 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.

Input:

- **labelled data**:
  - *expects:* ExampleSet, *expects:* ExampleSetMetaData: #examples: = 0; #attributes: 0
- **performance**:
  - *optional:* PerformanceVector

Output:

- **performance**
- **example set**

Parameters:

- *positive class*: Specify the positive nominal value for the label attribute (case sensitive).
  - *Depends on*: any binominal criterion is selected, operator version is at most 2.1.1
- *main criterion*: The criterion used for comparing performance vectors.
  - *Default value*: first
- *accuracy*: Relative number of correctly classified examples
  - *Default value*: true
- *classification error*: Relative number of misclassified examples
  - *Default value*: false
- *kappa*: The kappa statistics for the classification
  - *Default value*: false
- *precision*: Relative number of correctly as positive classified examples among all examples classified as positive
  - *Default value*: false
- *recall*: Relative number of correctly as positive classified examples among all positive examples
  - *Default value*: false
- *lift*: The lift of the positive class
  - *Default value*: false
- *fallout*: Relative number of incorrectly as positive classified examples among all negative examples
  - *Default value*: false
- *f measure*: Combination of precision and recall: f=2pr/(p+r)
  - *Default value*: false
- *false positive*: Absolute number of incorrectly as positive classified examples
  - *Default value*: false
- *false negative*: Absolute number of incorrectly as negative classified examples
  - *Default value*: false
- *true positive*: Absolute number of correctly as positive classified examples
  - *Default value*: false
- *true negative*: Absolute number of correctly as negative classified examples
  - *Default value*: false
- *sensitivity*: Relative number of correctly as positive classified examples among all positive examples (like recall)
  - *Default value*: false
- *specificity*: Relative number of correctly as negative classified examples among all negative examples
  - *Default value*: false
- *youden*: The sum of sensitivity and specificity minus 1
  - *Default value*: false
- *positive predictive value*: Relative number of correctly as positive classified examples among all examples classified as positive (same as precision)

  - *Default value*: false
- *negative predictive value*: Relative number of correctly as negative classified examples among all examples classified as negative
  - *Default value*: false
- *psep*: The sum of the positive predicitve value and the negative predictive value minus 1
  - *Default value*: false
- *skip undefined labels* (expert): If set to true, examples with undefined labels are skipped.
  - *Default value*: true
- *use example weights* (expert): Indicated if example weights should be used for performance calculations if possible.
  - *Default value*: true

8.1.2 Performance (Classification)

Synopsis

This operator calculates a *PerformanceVector* containing performance values according to a list of selected performance criteria applicable for multi-class 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.

This operator expects a test *HadoopExampleSet* as input, containing one attribute with the role *label* and one with the role *prediction*. See the *Set Role* operator for more details. On the basis of this two attributes a *PerformanceVector* is calculated, containing the values of the performance criteria. If a *PerformanceVector* was fed into *performance* input, it's values are kept if it does not already contain the new criteria. Otherwise the values are averaged over the old and the new values. The output is compatible and can be combined with the output of the similar RapidMiner operator.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a *Log* operator 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. attribute selection or other meta optimization process setups. If no main criterion was selected, the first criterion in the resulting performance vector will be assumed to be the main criterion.

Input:

- **labelled data**:
  - *expects:* ExampleSet, *expects:* ExampleSetMetaData: #examples: = 0; #attributes: 0
- **performance**:
  - *optional:* PerformanceVector

Output:

- **performance**
- **example set**

Parameters:

- *main criterion*: The criterion used for comparing performance vectors.

  - *Default value*: first
- *accuracy*: Relative number of correctly classified examples
  - *Default value*: true
- *classification error*: Relative number of misclassified examples
  - *Default value*: false
- *kappa*: The kappa statistics for the classification
  - *Default value*: false
- *absolute error*: Average absolute deviation of the prediction from the actual value
  - *Default value*: false
- *relative error*: Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
  - *Default value*: false
- *relative error lenient*: Average lenient relative error (average of absolute deviation of the prediction from the actual value divided by maximum of the actual value and the prediction)
  - *Default value*: false
- *relative error strict*: Average strict relative error (average of absolute deviation of the prediction from the actual value divided by minimum of the actual value and the prediction)
  - *Default value*: false
- *root mean squared error*: Averaged root-mean-squared error
  - *Default value*: false
- *squared error*: Averaged squared error
  - *Default value*: false
- *skip undefined labels* (expert): If set to true, examples with undefined labels are skipped.
  - *Default value*: true
- *use example weights* (expert): Indicated if example weights should be used for performance calculations if possible.
  - *Default value*: true
- *class weights* (expert): The weights for all classes (first column: class name, second column: weight), empty: using 1 for all classes.

8.1.3 Performance (Regression)

Synopsis

This operator calculates a *PerformanceVector* containing performance values according to a list of selected performance criteria applicable for regression tasks.

Description

This performance evaluator operator should be used for regression tasks, i.e. in cases where the label attribute has a numberical value type.

This operator expects a test *HadoopExampleSet* as input, containing one attribute with the role *label* and one with the role *prediction*. See the *Set Role* operator for more details. On the basis of this two attributes a *PerformanceVector* is calculated, containing the values of the performance criteria. If a *PerformanceVector* was fed into *performance* input, it's values are kept if it does not already contain the new criteria. Otherwise the values are averaged over the old and the new values. The output is compatible and can be combined with the output of the similar RapidMiner operator.

All of the performance criteria can be switched on using boolean parameters. Their values can be queried by a *Log* operator 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. attribute selection or other meta optimization process setups. If no main criterion was selected, the first criterion in the resulting performance vector will be assumed to be the main criterion.

Input:

- **labelled data**:
  - *expects:* ExampleSet, *expects:* ExampleSetMetaData: #examples: = 0; #attributes: 0
- **performance**:
  - *optional:* PerformanceVector

Output:

- **performance**
- **example set**

Parameters:

- *main criterion*: The criterion used for comparing performance vectors.
  - *Default value*: first
- *root mean squared error*: Averaged root-mean-squared error
  - *Default value*: true
- *absolute error*: Average absolute deviation of the prediction from the actual value
  - *Default value*: false
- *relative error*: Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)
  - *Default value*: false
- *relative error lenient*: Average lenient relative error (average of absolute deviation of the prediction from the actual value divided by maximum of the actual value and the prediction)
  - *Default value*: false
- *relative error strict*: Average strict relative error (average of absolute deviation of the prediction from the actual value divided by minimum of the actual value and the prediction)
  - *Default value*: false
- *squared error*: Averaged squared error
  - *Default value*: false
- *skip undefined labels* (expert): If set to true, examples with undefined labels are skipped.

  - *Default value*: true

- *use example weights* (expert): Indicated if example weights should be used for performance calculations if possible.
  - *Default value*: true

8.2 Evaluation - Validation

8.2.1 Split Validation

Synopsis

Randomly splits up the example set into a training and test set and evaluates a model.

Description

Splits up the data set into training and test data sets. Using these data sets, this meta operator evaluates a model to estimate the performance of a learning operator.

The meta operator has two subprocesses. The training subprocess should be used to build a prediction model. The built model is then applied in the testing subprocess. The testing subprocess calculates performance of the model. The inputs of the two subprocesses are the training set and the test set, respectively. These are two partitions of the original data set. You can specify the ratio of the training partition.

The Split Validation operator can be used to predict the performance of a model on unseen data when no explicit test data set is available.

Input:

- **training**:
  - *expects:* HadoopExampleSet, *expects:* HadoopExampleSet

Output:

- **model**
- **training**
- **averagable 1**
- **averagable 2**

Parameters:

- *split ratio*: Relative size of the training set
  - *Default value*: 0.7

8.3 Export

8.3.1 Write CSV

Synopsis

Writes CSV file from a Hive table.

Description

This operator exports an example set on the cluster directly to a CSV file on the client's local file system. The data is read and written as a stream, so even huge files can be written safely with a small memory footprint (as long as there is enough disk space).

Input:

- **input**:
  - *expects:* HadoopExampleSet

Output:

- **through**
- **file**

Parameters:

- *csv file*: Name of the file to write the data in.
  - *Depends on*: file output port is not connected
- *column separator*: The column separator.
  - *Default value*: ;
- *write attribute names*: Indicates if the attribute names should be written as first row.
  - *Default value*: true
- *quote nominal values*: Indicates if nominal values should be quoted with double quotes.
  - *Default value*: true
- *format date attributes* (expert): Indicates if date attributes are written as a formated string or as milliseconds past since January 1, 1970, 00:00:00 GMT
  - *Default value*: true
- *append to file*: Indicates if new content should be appended to the file or if the pre-existing file content should be overwritten.
  - *Default value*: false
  - *Depends on*: file output port is not connected
- *encoding* (expert): The encoding used for reading or writing files.
  - *Default value*: SYSTEM

8.3.2 Write Database

Synopsis

Writes Database from Hive table.

Description

This operator writes a Hive table directly to database. The data is read and written as a stream, so even huge files can be written safely with a small memory footprint (as long as there is enough disk space).

Input:

- **input**:
  - *expects:* HadoopExampleSet

Output:

- **through**

Parameters:

- *define connection*: Indicates how the database connection should be specified.
  - *Default value*: predefined
- *connection*: A predefined database connection.
  - *Depends on*: define connection = predefined
- *database system*: The used database system.
  - *Default value*: MySQL
  - *Depends on*: define connection = url
- *database url*: The URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database'
  - *Depends on*: define connection = url
- *username*: The database username.
  - *Depends on*: define connection = url
- *password*: The password for the database.
  - *Depends on*: define connection = url
- *jndi name*: JNDI name for a data source.
  - *Depends on*: define connection = jndi
- *use default schema* (expert): If checked, the user's default schema will be used.
  - *Default value*: true
- *schema name* (expert): The schema name to use, unless use_default_schema is true.
  - *Depends on*: use default schema = false
- *table name*: A database table.
- *overwrite mode* (expert): Indicates if an existing table should be overwritten or if data should be appended.
  - *Default value*: none
- *set default varchar length* (expert): Set varchar columns to default length.
  - *Default value*: false
- *default varchar length* (expert): Default length of varchar columns.
  - *Default value*: 128
  - *Depends on*: set default varchar length = true
- *add generated primary keys* (expert): Indicates whether a new attribute holding the auto generated primary keys is added to the result set.
  - *Default value*: false
- *db key attribute name* (expert): The name of the attribute for the auto generated primary keys

   - *Default value*: generated_primary_key
   - *Depends on*: add generated primary keys = true

- *batch size* (expert): The number of examples which are written at once with one single query to the database. Larger values can greatly improve the speed - too large values however can drastically *decrease* the performance. Additionally, some databases have restrictions on the maximum number of values written at once.
   - *Default value*: 1
- *Varchar size* (expert): Varchar length
   - *Default value*: 100

8.4 Hive Access

8.4.1 Append into Hive

Synopsis

Appends rows of the input data set to a permanent Hive table.

Description

This operator appends the content of the current result set permanently to a Hive table of the same structure on the cluster. This operation might take a lot of time to run, because it needs to materialize the input data to append it to a Hive table.

The operator tries to match the attributes of the input data set with the columns of the target Hive table by their name (names are case-insensitive). The data must fit into the target table, so the type of the matched attributes/columns must be appropriate (see the RapidMiner and Hive data type conversion section in the documentation), or the target column must have a string data type. Otherwise, the operator gives a design time error and throws an error during execution.

The input data set may have further attributes, the operator only deals with those that exists in the target Hive table. However, the input data set must have all attributes that the target Hive table has, otherwise an error is thrown. You may change this rule if you set the *insert_nulls* parameter to true. This tells the operator to insert NULL values into the columns that are missing from the input data set. This decreases the strictness of the schema validation, but it allows you to add columns to the table later without causing earlier append processes to fail.

You may also insert into a partitioned Hive table. The input data set must contain the partitioning columns (they can not be set to NULL). Dynamic partitioning is performed, so the target partitions is determined during execution time. You must explicitly enable inserting into a partitioned table with the *partitioning* parameter. If this is set to true, you may set the *max_partitions* parameter which is an upper limit for the number of partitions that this operation inserts into. The purpose of this setting is to protect against inserting into a large number of partitions, as it may lead to a large overhead. Your Hive server has a default limitation for this. If you set the operator's parameter to -1, this default value will be the upper limit. Otherwise, only the operator's parameter limits the number of partitions affected during this insert operation. This parameter has no effect on other operations.

Please note that append by more than one processes at the same time into the same destination table is not supported, and may lead to unpredictable results.

Input:

- **example set input**:
   - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *tablename*: Target Hive table.
- *create* (expert): Create table if it does not exist
  - *Default value*: true
- *insert nulls* (expert): Insert NULL values for missing columns if the target table exists already with further columns.
  - *Default value*: false
- *partitioning* (expert): Enable insert into partitioned table. Dynamic partitioning is performed based on the target Hive tables' partitioning columns
  - *Default value*: false
- *max partitions* (expert): Upper limit for the number of partitions (dynamic partitioning); use -1 to use Hive's settings. This is a limit for the different values of the partitioning columns (combined).
  - *Default value*: -1
  - *Depends on*: partitioning = true

8.4.2 Retrieve from Hive

Synopsis

Retrieves a Hive table for analysis.

Description

Retrieves the Hive table for further analysis. The data remains on the cluster and Radoop only loads references, metadata and statistics about the table. It takes the same amount of time to retrieve a huge table and a small table. Output:

- **output**

Parameters:

- *table*: Input table.
- *filter clause* (expert): Here you can specify the WHERE clause of the initial query. It is especially useful if you are querying partitioned tables. Only use this if you know exactly what you are doing.
  - *Default value*: WHERE 1 = 1

8.4.3 Store in Hive

Synopsis

Stores current Hive view as a permanent Hive table.

Description

This operator stores a result table permanently on the cluster. It might take a lot of time to run, because it needs to materialize the data to store as a Hive table, i.e. complete all deferred calculations to get the data.

You can choose to store the data in a so-called *external table*. This means that you control the location (directory) where the files are stored on the distributed file system. When you drop an external table (e.g. use the Drop Table command in the Hive Management Perspective), the data is not removed. However, if you check the *dropfirst* parameter in this operator and the target table already exists, the operator cleans the target directory. Hence, this flag parameter's behavior is consistent between normal and external table.

Using the external table option, you can save your data on a different storage system, like Amazon S3. Use the *s3://<bucket>/<path>* or *s3n://<bucket>/<path>* format to specify the destination directory (it will be created if it does not exist). Please note that in this case the target directory can not be checked or emptied beforehand, since it can not be accessed directly without AWS credentials.

Another useful feature is *partitioning*. You may choose one or more so-called partitioning columns. Rows with different values in these columns are handled separately by Hive. This feature is important for enhancing manageability and performance. Data in the different partitions are kept separately. Performance may be radically increased if you filter on the partitioning columns (use *Retrieve* operator's *filter_clause* parameter).

This operator allows *dynamic* partitioning. This means that the target partition for a row is determined during execution by the value of the partitioning columns. Dynamic partitioning is traditionally restricted by Hive, as the user may misuse this easily by using a column for partitioning that has a lot of different values. This causes a large overhead in processing the table and misses the point of partitioning. Please choose partitioning columns so, that they do not cause extremely large number partitions because of the large number of different values. You may explicitly modify the limit for the maximum number of partitions that this store operation allows using the *max_partitions* parameter. (This parameter value only limits this dynamic partitioning operation. For other Hive commands, Hive's configuration applies). Typical partitioning columns are log dates (day or month) or larger area codes (hundreds of partitions at most). Please note that you should avoid the NULL values in the partitioning columns, as they may lead to errors in querying the Hive table later. Use *Replace Missing Values* operator to change NULL values in any attribute.

The target table is created with the default storage settings defined in you Hive server configuration. You may alter this behavior by setting the *custom_storage* parameter to true and changing the storage parameters. You should consult the Hive documentation for the details and the advantages/disadvantages of these settings.

Input:

- **input**:
  - *expects:* HadoopExampleSet

Output:

- **output**

Parameters:

- *tablename*: Hive table:
- *dropfirst* (expert): Forced table creation. For external tables the target directory will be cleaned.
  - *Default value*: true
- *external table* (expert): Store in external table (specify the location explicitly).
  - *Default value*: false
- *location* (expert): Location of the external table data: a directory on the HDFS, or S3 (use s3n:// prefix), etc.
  - *Depends on*: external table = true
- *partition by* (expert): Ordered list of partitioning columns.

- *max partitions* (expert): Upper limit for the number of partitions (dynamic partitioning); use -1 to use Hive's settings. This is a limit for the different values of the partitioning columns (combined).
  - *Default value*: -1
  - *Depends on*: partition by parameter selects a valid partitioning column
- *custom storage* (expert): Use custom storage format. You may specify the target table's storage format explicitly.
  - *Default value*: false
- *custom storage handler* (expert): Use a custom storage handler. You specify storage handler class name explicitly.
  - *Default value*: false
  - *Depends on*: custom storage = true
- *storage handler* (expert): Custom storage handler. It must exist in the CLASSPATH of the Hive server.
  - *Depends on*: custom storage handler = true
- *row format* (expert): Target table row format. Please note that older Hive versions may not support all row format settings.
  - *Default value*: Default format
  - *Depends on*: custom storage = true, custom storage handler = false
- *fields delimiter* (expert): Custom field delimiter character.
  - *Default value*: 001
  - *Depends on*: row format = DELIMITED
- *fields escape char* (expert): Escape character that can be used to escape the field delimiter character. Leave empty for no escape character. Use '\\' for the '\' character.
  - *Depends on*: row format = DELIMITED
- *collection delimiter* (expert): Custom delimiter character that separates collection items (COLLECTION data type).
  - *Default value*: 002
  - *Depends on*: row format = DELIMITED
- *map keys delimiter* (expert): Custom delimiter character that separates map keys (MAP data type).
  - *Default value*: 003
  - *Depends on*: row format = DELIMITED
- *lines delimiter* (expert): Custom delimiter character that separates records (lines).
  - *Default value*: n
  - *Depends on*: row format = DELIMITED
- *null character* (expert): Character for storing a NULL value.
  - *Default value*: N
  - *Depends on*: row format = DELIMITED
- *serde class name* (expert): Custom SerDe class name. It must exist in the CLASSPATH of the Hive server.
  - *Depends on*: row format = Custom SerDe

- *serde properties* (expert): User defined SerDe parameter. These case sensitive key-value pairs are passed to the table's SerDe.
  - *Depends on*: custom storage = true, any binominal criterion is selected
- *hive file format* (expert): Target table file format. Please note that older Hive versions may not support all file format types.
  - *Default value*: Default Format
  - *Depends on*: custom storage = true, custom storage handler = false, Hive version of the connection is Hive server
- *impala file format* (expert): Target table file format. Please note that older Impala versions may not support all file format types.
  - *Default value*: Default format
  - *Depends on*: custom storage = true, custom storage handler = false, Hive version of the connection is Impala
- *input format* (expert): Custom input format class name. It must exist in the CLASSPATH of the Hive server. Example: 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'
  - *Depends on*: hive file format = Custom Format
- *output format* (expert): Custom output format class name. It must exist in the CLASSPATH of the Hive server. Example: 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
  - *Depends on*: hive file format = Custom Format

8.5 Hive Access - Table Management

8.5.1 Copy Hive Table

Synopsis

Copies a Hive table.

Description

Copies the specified Hive table. If the *overwrite* parameter is true the operator drops the possibly existing table with the given name. If set to false, the operator will generate an error in case of a conflict. Partitioned tables cannot be copied. For this purpose use the Retrieve and Store operators and specify the partitioning attributes explicitly. Please note that the execution may fail if you overwrite a table used by the current process.

Input:

- **through 1**

Output:

- **through 1**

Parameters:

- *old table*: Table to copy.
- *new table*: The name of the copied table.

- *overwrite*: Determines whether a possibly existing table with the same table name should be overwritten. If set to false an exception is thrown in case of a conflict.
  - *Default value*: false

8.5.2 Drop Hive Table

Synopsis

Drops a Hive table.

Description

Drops (deletes) the specified Hive table or view. If the *fail if missing* parameter is true the operator generates an error in case of a missing table or view. Please note that the execution may fail if you drop a table used by the current process.

Input:

- **through 1**

Output:

- **through 1**

Parameters:

- *table*: Table to drop.
- *fail if missing*: Determines whether an exception should be generated if the table is missing, e. g. because it already got deleted in the last run. If set to false nothing happens if this error occurs.
  - *Default value*: false

8.5.3 Rename Hive Table

Synopsis

Renames a Hive table.

Description

Renames the specified Hive table. If the *overwrite* parameter is true the operator drops the possibly existing table with the given name. If set to false, the operator will generate an error in case of a conflict. Please note that the execution may fail if you overwrite a table used by the current process.

Input:

- **through 1**

Output:

- **through 1**

Parameters:

- *old table*: Table to rename.

- *new table*: The new table name.
- *overwrite*: Determines whether a possibly existing table with the same table name should be overwritten. If set to false an exception is thrown in case of a conflict.
  - *Default value*: false

8.6 Import

8.6.1 Read CSV

Synopsis

Reads CSV file and stores it in Hive.

Description

This operator works the same way as the built-in CSV reader, but it loads the data directly to Hive instead of the memory. Even huge data files can be loaded safely as it has a low memory footprint. The CSV file may reside on the client's local file system or on the Hadoop Distributed File System (HDFS).

Currently, the operator supports three types of import scenarios:

- Import from a local flat file to the Hadoop cluster into a Hive table.
- Import from the distributed file system to a Hive table.
- Import from the distributed file system without copying any data.

In the first scenario a local flat file on the client's file system is the source. You can define the columns, specify the column separator, the usage of quote characters etc. After you successfully configured these settings, you can specify the target Hive table properties. If you want to further process the data immediately after this operator, you can use a temporary table to store the data. If you want to permanently store the data in Hive, then you must choose a name for the table and you may also specify advanced storage settings (partitioning, storage format) for this table. In case of a permanent table, you can easily access the data later with a *Retrieve* operator.

The second scenario is when the data already resides on your Hadoop cluster. This is the preferred scenario if you have a large input data, as streaming a local large file may take a lot of time. In this case, you must specify the distributed file system (usually HDFS) location of your source data. This may be a directory, in which case, all non-empty files in it will be imported, or it can be a single file. You can specify the fields and the separators similarly to the local file scenario. You also can specify the target storage settings similarly. During process execution, the operator will start an import job that reads the input directory or file, and writes the content into the specified Hive table.

The third method is the fastest way to process a data that already resides in the distributed file system. In this case, you only create a so-called *external table* in Hive. This means that you create a table for which you specify the location of the data. When you query this table, Hive will look up the content in the path you have specified. In this case there is no need for an import job to be performed, as the data is never copied to another path, it is always read from its current location. However, you have some limitations in the settings compared to the second scenario. You can not specify a single file as the source, it must be a directory. You also have fewer options for defining separators. If you are fine with these limitations, this is the fastest way to access the content of a flat file on the distributed file system by your process.

Output:

- **output**

Parameters:

- *Configuration*: Configure the operator with wizard
- *override file* (expert): This location string overrides the source path defined using the import wizard. This is useful e.g. for testing purposes or for using macros in the path.
  - *Default value*: false
- *override location* (expert): Source location.
  - *Depends on*: override file = true

8.6.2 Read Database

Synopsis

Reads Database table and stores it in Hive.

Description

This operator works the same way as the built-in database reader, but it loads the data directly to Hive instead of the memory. Even huge data files can be loaded safely as it has a low memory footprint. Output:

- **output**

Parameters:

- *define connection*: Indicates how the database connection should be specified.
  - *Default value*: predefined
- *connection*: A predefined database connection.
  - *Depends on*: define connection = predefined
- *database system*: The used database system.
  - *Default value*: MySQL
  - *Depends on*: define connection = url
- *database url*: The URL connection string for the database, e.g. ‘jdbc:mysql://foo.bar:portnr/database‘
  - *Depends on*: define connection = url
- *username*: The database username.
  - *Depends on*: define connection = url
- *password*: The password for the database.
  - *Depends on*: define connection = url
- *jndi name*: JNDI name for a data source.
  - *Depends on*: define connection = jndi
- *define query*: Specifies whether the database query should be defined directly, through a file or implicitely by a given table name.
  - *Default value*: query
- *query*: An SQL query.
  - *Depends on*: define query = query

- *query file*: A file containing an SQL query.
  - *Depends on*: define query = query file
- *use default schema* (expert): If checked, the user's default schema will be used.
  - *Default value*: true
  - *Depends on*: define query = table name
- *schema name* (expert): The schema name to use, unless use_default_schema is true.
  - *Depends on*: use default schema = false
- *table name*: A database table.
  - *Depends on*: define query = table name
- *prepare statement* (expert): If checked, the statement is prepared, and '?'-parameters can be filled in using the parameter 'parameters'.
  - *Default value*: false
- *parameters* (expert): Parameters to insert into '?' placeholders when statement is prepared.
  - *Depends on*: prepare statement = true
- *datamanagement*: Determines, how the data is represented internally.
  - *Default value*: double_array
- *temporary table* (expert): Temporary table
  - *Default value*: true
- *saved table name* (expert): Table name
  - *Depends on*: temporary table = false

8.7 Modeling - Application

8.7.1 Apply Model

Synopsis

Applies a model on the cluster

Description

This operator applies a model on the data in Hive. This means, that you can apply a RapidMiner model on your data in the cluster. The application of every supported model is performed by a distributed, scalable algorithm. The operator supports all core RapidMiner prediction and cluster model types. The model type is verified during design time, if possible, and you will get an error for unsupported models. Please note that the application of several models may require Java 7 to be used on the Hadoop nodes as well, as that is a requirement of RapidMiner.

You may specify some model specific parameters using the *application_parameters*.

- **BaggingModel** - *materialization_limit* - Forces Hive table materialization after the specified number of iterations (integer; set to zero to turn off; default value: 5). Applying a BaggingModel on your data in Hive may result in creating a lot of Hive views. You should set this value if you experience that the Apply Prediction

Model operator hangs or takes too much time, or even notice that a lot of time elapses between two submitted Hive statements (set the *rapidminer.gui.log_level* property to *FINE* and check the *Log* view)

- **DistributionModel** - *split_statements* - If set to true (boolean: use “true” or “false” literals; default value: false), longer HiveQL statements will be splitted into several statements, each materializing the data. The code for Naive Bayes scoring may be quite large if the training data set contains a lot of attributes and/or if the label attribute has several possible class value. Please note that if you set this to true, the scoring will probably take much longer time. Use this only if the model application seems to hang or takes too much time.
- **DistributionModel** - *use_udf* - If set to true (boolean: use “true” or “false” literals; default value: false), the model scoring will be performed by an UDF written in Java. The presence of a lot of regular attributes or class values may cause that the HiveQL that implements the scoring algorithm becomes too large for the HiveQL parser to handle. In this case this option may help you and prevent such errors.
- **PCAModel** - *number_of_components* - Specify a lower number of components
- **PCAModel** - *variance_threshold* - Specify a new threshold for the cumulative variance of the principal components.
- **PCAModel** - *keep_attributes* - If true, the original features are not removed.

You may also force the usage of the so-called general model applier implementation by setting *use_general_applier* to true. In this case the model apply operation is preferred to be performed by the same code as when the model is applied in-memory. I.e. the core RapidMiner code is used instead of translating the operation into custom Hadoop code. In this case, however, model application parameters are not available. If the parameter is set to false, then custom Hadoop code may be used for several model types to achieve better performance.

Input:

- **model**:
  - *expects:* Model
- **unlabelled data**:
  - *expects:* HadoopExampleSet

Output:

- **labelled data**
- **model**

Parameters:

- *application parameters* (expert): List of model type specific parameters for the application (usually not needed).
- *use general applier* (expert): Set it to true to force the usage of the general model applier implementation. In this case the model apply operation is preferred to be performed by the same code as when the model is applied in-memory. I.e. the core RapidMiner code is used instead of translating the operation into custom Hadoop code. If the parameter is set to false, then custom Hadoop code may be used for several model types to achieve better performance.
  - *Default value*: false

8.7.2 Update Model

Synopsis

Updates a model using data on the cluster

Description

This operator updates a prediction model using the data in Hive. Radoop currently only supports updating a Naive Bayes model. The model on the model input port is trained using the data on the example set input port. The model you update with this operator may have been initially trained in RapidMiner or on the cluster (Naive Bayes learner operator). Please note that the data on the example set input port must have exactly the same structure as the training data which the model was built on. Nominal attributes may have new values, which will update the model accordingly. For notes on the learning algorithm, see the Naive Bayes operator. This operator has a parameter where you can specify model type specific parameters for the learning algorithms.

- **DistributionModel** - *nominal_group_size* - This expert tuning parameter does not affect the model output, only the performace of the learning procedure. This parameter is an integer value (default: 100) that defines how many nominal attributes will be calculated in a single Map/Reduce job. This is a trade/off between processing time and operative memory usage on the cluster nodes. Hence, you should increase the value for performance and decrease it if you encounter with any heap space error on the nodes. However, the latter case is highly unlikely and rather indicates an incorrect usage of nominal attributes (see *Naive Bayes* Radoop learner operator).

Input:

- **example set**:
  - *expects:* HadoopExampleSet
- **model**:
  - *expects:* Model

Output:

- **example set**
- **model**

Parameters:

- *training parameters*: List of model type specific parameters for the learning algorithms.

8.8 Modeling - Classification and Regression

8.8.1 Combine Models

Synopsis

Combines arbitrary number of models into a voting (bagging) model.

Description

This operator can be used outside the Radoop Nest.

Combines the prediction models on its input ports into a BaggingModel. This model can then be applied on both data in the memory or data on the cluster. The operator merges the nominal mappings of the nominal attributes (including the label's mapping in case of classification). This operator allows the combination of models that have been trained on different subsets of an attribute set (ideal for a Random Forest algorithm), but the inner models must be able to score a data set that has more regular attributes than the model was trained on (warnings in the log may warn you about this during scoring).

Input:

- **model input 1**:
  - requires at least 1 HadoopExampleSet (collection)

Output:

- **model output**

8.8.2 Decision Tree

Synopsis

Generates a Decision Tree for classification of both nominal and numerical data. It is based on the decision tree implementation in Spark MLlib.

Description

Information about the algorithm can be found in the MLlib documentation about Decision Tree Please note that unlike RapidMiner's Decision Tree, this can handle only binominal label. The displayed count of label classes in each node in the model is only a scaled probability, not the exact count.

Input:

- **training set**:
  - *expects:* HadoopExampleSet

Output:

- **model**
- **exampleSet**

Parameters:

- *file format* (expert): The input ExampleSet will be materialized in the specified format. This setting is ignored if the input is already a table in Text or in ORC format - in this case no materalization is performed. Please note that you can force materializing in Text/ORC by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in ORC format requires less storage but the execution may be slower.
  - *Default value*: TEXTFILE
- *criterion*: Selects the criterion on which attributes will be selected for splitting.
  - *Default value*: Gini
- *minimal gain*: For a node to be split further, the split must improve at least this much (in terms of information gain).
  - *Default value*: 0.1
- *maximal depth*: Maximum depth of a tree. Deeper trees are more expressive (potentially allowing higher accuracy), but they are also more costly to train and are more likely to overfit.
  - *Default value*: 20
- *maximum bins*: Number of bins used when discretizing continuous features.
  - *Default value*: 32
- *minimal size for split* (expert): For a node to be split further, each of its children must receive at least this number of training instances.

  - *Default value*: 4

- *maximum memory in MB* (expert): Amount of memory to be used for collecting sufficient statistics. The default value is conservatively chosen to be 256 MB to allow the decision algorithm to work in most scenarios. Increasing maxMemoryInMB can lead to faster training (if the memory is available) by allowing fewer passes over the data. However, there may be decreasing returns as maxMemoryInMB grows since the amount of communication on each iteration can be proportional to maxMemoryInMB.
  - *Default value*: 256
- *subsampling rate* (expert): Fraction of the training data used for learning the decision tree.
  - *Default value*: 1.0
- *use node id cache* (expert): If this is set to true, the algorithm will avoid passing the current model (tree or trees) to executors on each iteration.
  - *Default value*: false
- *driver memory (MB)* (expert): Amount of memory to use for the driver process in MB. You should consider setting this higher if you train on features with many distinct categorical values. Set it to 0 to use the configured default value.
  - *Default value*: 2048

8.8.3 Linear Regression

Synopsis

This operator is a Linear Regression Learner. It is based on the linear regression implementation in Spark MLlib.

Description

Regression is a technique used for numerical prediction. Regression is a statistical measure that attempts to determine the strength of the relationship between one dependent variable (i.e. the label attribute) and a series of other changing variables known as independent variables (regular attributes). Just like Classification is used for predicting categorical labels, Regression is used for predicting a continuous value. For example, we may wish to predict the salary of university graduates with 5 years of work experience, or the potential sales of a new product given its price. Regression is often used to determine how much specific factors such as the price of a commodity, interest rates, particular industries or sectors influence the price movement of an asset.

Linear regression attempts to model the relationship between a scalar variable and one or more explanatory variables by fitting a linear equation to observed data. For example, one might want to relate the weights of individuals to their heights using a linear regression model.

Detailed information about the algorithm can be found in the MLlib documentation about Linear Regression.

Input:

- **training set**:
  - *expects:* HadoopExampleSet

Output:

- **model**
- **weights**
- **exampleSet**

Parameters:

- *file format* (expert): The input ExampleSet will be materialized in the specified format. This setting is ignored if the input is already a table in Text or in ORC format - in this case no materalization is performed. Please note that you can force materializing in Text/ORC by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in ORC format requires less storage but the execution may be slower.
  - *Default value*: TEXTFILE
- *regression method*: Various regression methods are derived by using different types of regularization: ordinary least squares or linear least squares uses no regularization; ridge regression uses L2 regularization; and Lasso uses L1 regularization.
  - *Default value*: Linear Regression
- *number of iterations*: Number of iterations of gradient descent to run.
  - *Default value*: 100
- *step size*: The initial step size of SGD for the first step. Default 0.1. In subsequent steps, the step size will decrease with stepSize/sqrt(current_iteration_number). This parameter should be < 1.0. Lower step size requires higher number of iterations. In this case the algorithm will generally converge slower but results in a better model.
  - *Default value*: 0.1
- *minibatch fractions*: Fraction of the input data set that should be used for one iteration of SGD. Default 1.0 (corresponding to deterministic/classical gradient descent)
  - *Default value*: 1.0
- *add intercept* (expert): Set if the algorithm should add an intercept.
  - *Default value*: true
- *use feature scaling* (expert): Scaling columns to unit variance as a heuristic to reduce the condition number: During the optimization process, the convergence (rate) depends on the condition number of the training dataset. Scaling the variables often reduces this condition number heuristically, thus improving the convergence rate. Without reducing the condition number, some training datasets mixing the columns with different scales may not be able to converge. Here, if useFeatureScaling is enabled, Spark will standardize the training features by dividing the variance of each column (without subtracting the mean), and train the model in the scaled space.
  - *Default value*: true
- *regularization parameter* (expert): The regularization parameter.
  - *Default value*: 0.01
  - *Depends on*: regression method = Linear Regression

8.8.4 Logistic Regression

Synopsis

This operator is a Logistic Regression Learner. It is based on the logistic regression implementation in Spark MLlib.

Description

Logistic regression is used to predict a binary response. Detailed information can be found in the MLlib documentation about Logistic Regression.

The operator supports both Stochastic Gradient Descent (SGD) and Limited-memory BFGS (LBFGS) optimizers. Information on the optimizers can be found in the MLlib documentation about optimizers.

Input:

- **training set**:
  - *expects:* HadoopExampleSet

Output:

- **model**
- **weights**
- **exampleSet**

Parameters:

- *file format* (expert): The input ExampleSet will be materialized in the specified format. This setting is ignored if the input is already a table in Text or in ORC format - in this case no materalization is performed. Please note that you can force materializing in Text/ORC by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in ORC format requires less storage but the execution may be slower.
  - *Default value*: TEXTFILE
- *optimizer*: The optimizer to solve the problem. Possible values are SGD (Stochastic Gradient Descent) and LBFGS (Limited-memory BFGS).
  - *Default value*: Stochastic Gradient Descent
- *number of iterations*: Number of iterations of gradient descent to run.
  - *Default value*: 100
- *step size*: The initial step size of SGD for the first step. Default 1.0. In subsequent steps, the step size will decrease with stepSize/sqrt(current_iteration_number).
  - *Default value*: 1.0
  - *Depends on*: optimizer = Stochastic Gradient Descent
- *minibatch fractions*: Fraction of the input data set that should be used for one iteration of SGD. Default 1.0 (corresponding to deterministic/classical gradient descent)
  - *Default value*: 1.0
  - *Depends on*: optimizer = Stochastic Gradient Descent
- *convergence to L*: Set the convergence tolerance of iterations for L-BFGS. Default 1E-4. Smaller value will lead to higher accuracy with the cost of more iterations.
  - *Default value*: 1.0E-4
  - *Depends on*: optimizer = Limited-memory BFGS
- *number of corrections*: Set the number of corrections used in the LBFGS update. Default 10. Values of numCorrections less than 3 are not recommended; large values of numCorrections will result in excessive computing time. 3 < numCorrections < 10 is recommended. Restriction: numCorrections > 0.
  - *Default value*: 10
  - *Depends on*: optimizer = Limited-memory BFGS
- *add intercept* (expert): Set if the algorithm should add an intercept.
  - *Default value*: true

- *use feature scaling* (expert): Scaling columns to unit variance as a heuristic to reduce the condition number: During the optimization process, the convergence (rate) depends on the condition number of the training dataset. Scaling the variables often reduces this condition number heuristically, thus improving the convergence rate. Without reducing the condition number, some training datasets mixing the columns with different scales may not be able to converge. Here, if useFeatureScaling is enabled, Spark will standardize the training features by dividing the variance of each column (without subtracting the mean), and train the model in the scaled space.
  - *Default value*: true
- *updater* (expert): Set the updater function to actually perform a gradient step in a given direction. The updater is responsible to perform the update from the regularization term as well, and therefore determines what kind or regularization is used, if any.
  - *Default value*: Simple Updater
- *regularization parameter* (expert): The regularization parameter.
  - *Default value*: 0.0
  - *Depends on*: updater = Simple Updater

8.8.5 Naive Bayes

Synopsis

Returns a classification model using estimated normal distributions.

Description

Naive Bayes learner on the cluster. Trains a Naive Bayes model on your data on the cluster. The trained model may be applied both on the cluster (Apply Prediction Model operator) or in the memory (RapidMiner's Apply Model operator). You can also update a trained Naive Bayes model with additional data. With Update Prediction Model operator you do not have to train a new model on the whole data set, just update it with additional examples on the cluster. This classifier can be used on weighted examples, where the weights are determined by the attribute having the weight role.

The Naive Bayes classifier algorithm applies Bayes' theorem with strong independence assumptions. The algorithm assumes normal distribution for numerical attributes. For nominal attributes the model will be based on the relative frequencies. Please note that nominal attributes having thousands or more unique values should never have a nominal type when applying a Naive Bayes learner. If a nominal attribute in the training set has too many values, the operator will throw an error. You should either group these nominals into fewer values or convert them to numericals. You can also generate numerical or nominal attributes with fewer distinct values. E.g. date attributes should be converted to numericals, while other information, like a flag attribute for weekdays/weekends may be extracted from them to create a proper training data set.

The algorithm has an expert tuning parameter that does not affect the model output only the performace of the learning procedure. This parameter is an integer value that defines how many nominal attributes will be calculated in a single Map/Reduce job. This is a trade/off between processing time and operative memory usage on the cluster nodes. Hence, you should increase the value for performance and decrease it if you encounter with any heap space error on the nodes. However, the latter case is highly unlikely and rather indicates an incorrect usage of nominal attributes (see above). This parameter can also be set when the model is trained using the *Update Prediction Model* operator.

Input:

- **training set**:
  - *expects:* HadoopExampleSet

Output:

- **model**
- **exampleSet**

Parameters:

- *laplace correction* (expert): Use Laplace correction to prevent high influence of zero probabilities.
  - *Default value*: true
- *nominal group size* (expert): This parameter affects only the performance, not the output. Statistics for a group of nominals are calculated together. This is the size of these groups. Increase it for faster learning, decrease if nodes run out of memory.
  - *Default value*: 100

8.9 Modeling - Clustering

8.9.1 Canopy

Synopsis

Clustering with Canopy on Mahout

Description

This operator represents an implementation of Canopy clustering. This operator will create a cluster attribute if not present yet.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, *roles:* id

Output:

- **example set output**

Parameters:

- *distance function*: Distance function
  - *Default value*: SquaredEuclidean
- *t1 distance metric*: The T1 distance metric (distance threshold for adding a point to a cluster)
- *t2 distance metric*: The T2 distance metric (distance threshold for keeping the point for further processing, T1 > T2)
- *reducer distance metrics* (expert): If set to true, different distance thresholds may be specified for the reducer phase.
  - *Default value*: false
- *reducer t1 distance* (expert): The reducer's T1 distance metric. If not specified, T1 is used by the reducer.
  - *Depends on*: reducer distance metrics = true
- *reducer t2 distance* (expert): The reducer's T2 distance metric. If not specified, T2 is used by the reducer.

  - *Depends on*: reducer distance metrics = true
- *canopies min number* (expert): The minimum size of canopies produced (can be zero)
  - *Default value*: 0
- *cluster classification threshold* (expert): Is a clustering strictness / outlier removal parameter. Its value should be between 0 and 1. Vectors having pdf below this value will not be clustered.
  - *Default value*: 0.0
- *only result*: If set, clustering returns only (ID, ClusterID) pairs, and removes other attributes. This option removes some overhead, thus, should decrease the processing time.
  - *Default value*: false

8.9.2 Fuzzy K-Means

Synopsis

Clustering with Fuzzy K-Means on Mahout

Description

This operator represents an implementation of Fuzzy K-Means. This operator will create a cluster attribute if not present yet.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, *roles:* id

Output:

- **example set output**
- **cluster model**

Parameters:

- *number of clusters*: Number of clusters
  - *Default value*: 4
- *maximum iterations* (expert): The maximum number of iterations to run, independent of the convergence specified
  - *Default value*: 4
- *delta convergence* (expert): Converge delta: a double value used to determine if the algorithm has converged (clusters have not moved more than the value in the last iteration)
  - *Default value*: 0.5
- *distance function* (expert): Distance function
  - *Default value*: SquaredEuclidean
- *the fuzzification factor* (expert): The "fuzzyness" argument, a double >= 1. If equal to 2, this is equivalent to normalising the coefficient linearly to make their sum 1. If it is close to 1, then the cluster center closest to the point is given much more weight than the others, and the algorithm is similar to k-means.
  - *Default value*: 2.0

- *emit most likely cluster* (expert): A boolean indicating, if true, that the clustering step should only emit the most likely cluster for each clustered point.
  - *Default value*: true
- *cluster classification threshold* (expert): Is a clustering strictness / outlier removal parameter. Its value should be between 0 and 1. Vectors having pdf below this value will not be clustered.
  - *Default value*: 0.0
- *only result*: If set, clustering returns only (ID, ClusterID) pairs, and removes other attributes. This option removes some overhead, thus, should decrease the processing time.
  - *Default value*: false
- *use local random seed*: Indicates if a local random seed should be used for randomization. Randomization may be used for selecting k different points at the start of the algorithm as potential centroids.
  - *Default value*: false
- *local random seed*: This parameter specifies the local random seed. This parameter is only available if the use local random seed parameter is set to true.
  - *Default value*: 1992
  - *Depends on*: use local random seed = true

8.9.3 K-Means

Synopsis

Clustering with K-Means on Mahout

Description

This operator represents an implementation of K-Means. This operator will create a cluster attribute if not present yet.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, *roles:* id

Output:

- **example set output**
- **cluster model**

Parameters:

- *number of clusters*: Number of clusters
  - *Default value*: 4
- *maximum iterations* (expert): Maximum number of iterations
  - *Default value*: 4
- *delta convergence* (expert): Converge delta: a double value used to determine if the algorithm has converged (clusters have not moved more than the value in the last iteration)
  - *Default value*: 0.5

- *distance function* (expert): Distance function
  - *Default value*: SquaredEuclidean
- *cluster classification threshold* (expert): Is a clustering strictness / outlier removal parameter. Its value should be between 0 and 1. Vectors having pdf below this value will not be clustered.
  - *Default value*: 0.0
- *only result*: If set, clustering returns only (ID, ClusterID) pairs, and removes other attributes. This option removes some overhead, thus, should decrease the processing time.
  - *Default value*: false
- *use local random seed*: Indicates if a local random seed should be used for randomization. Randomization may be used for selecting k different points at the start of the algorithm as potential centroids.
  - *Default value*: false
- *local random seed*: This parameter specifies the local random seed. This parameter is only available if the use local random seed parameter is set to true.
  - *Default value*: 1992
  - *Depends on*: use local random seed = true

8.10 Modeling - Correlation and Dependency Computation

8.10.1 Correlation Matrix

Synopsis

This operator determines correlation between all numerical attributes and it can produce a weights vector based on these correlations. Correlation is a statistical technique that can show whether and how strongly pairs of attributes are related.

Description

A correlation is a number between -1 and +1 that measures the degree of association between two attributes (call them X and Y). A positive value for the correlation implies a positive association. In this case large values of X tend to be associated with large values of Y and small values of X tend to be associated with small values of Y. A negative value for the correlation implies a negative or inverse association. In this case large values of X tend to be associated with small values of Y and vice versa.

This operator can be used for creating a correlation matrix that shows correlations of all the numeric attributes of the input ExampleSet. Please note that the operator skips the nominal attributes in the input Example Set. Furthermore, if an attribute contains a null value in any of the examples, the correlation matrix will contain nulls in the attribute's row and column. If you want to avoid this behaviour, please use the Replace Missing Values operator or set the improved correlation parameter to false - in this case the execution will be much slower, but the correlation matrix will be the same as RapidMiner's.

The attribute weights vector; based on the correlations is also returned by this operator.

Input:

- **example set**:
  - *expects:* HadoopExampleSet, *expects:* ExampleSet

Output:

- **example set**
- **matrix**
- **weights**

Parameters:

- *normalize weights*: Indicates if the attributes weights should be normalized.
  - *Default value*: true
- *squared correlation*: Indicates if the squared correlation should be calculated.
  - *Default value*: false
- *improved algorithm* (expert): Indicates if the improved calculating method should be used.
  - *Default value*: true

8.10.2 Covariance Matrix

Synopsis

This operator calculates the covariance between all attributes of the input HadoopExampleSet and returns a covariance matrix giving a measure of how much two attributes change together.

Description

Covariance is a measure of how much two attributes change together. If the greater values of one attribute mainly correspond with the greater values of the other attribute, and the same holds for the smaller values, i.e. the attributes tend to show similar behavior, the covariance is a positive number. In the opposite case, when the greater values of one attribute mainly correspond to the smaller values of the other, i.e. the attributes tend to show opposite behavior, the covariance is negative. The sign of the covariance therefore shows the tendency in the linear relationship between the variables.

This operator can be used for creating a covariance matrix that shows the covariances of all the numeric attributes of the input ExampleSet. Please note that the operator skips the nominal attributes in the input Example Set. Furthermore, if an attribute contains a null value in any of the examples, the covariance matrix will contain nulls in the attribute's row and column. If you want to avoid this behaviour, please use the Replace Missing Values operator.

The improved algorithm parameter indicates if the operator should use Hive UDAF for the execution. If you set this parameter to false the the execution will be much slower and the covariance matrix will not be the same as RapidMiner's - it calculates the covariance even if there were nulls in the input example set.

Input:

- **example set**:
  - *expects:* ExampleSet

Output:

- **example set**
- **covariance**

Parameters:

- *improved algorithm* (expert): Indicates if the improved calculating method should be used.

   - *Default value*: true

8.11 Process Control

8.11.1 Multiply

Synopsis

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.

If the input object is a *HadoopExampleSet* (data set), then this operator first performs all complex deferred calculations on the data set and writes the output to the distributed storage before passing it through on its output ports. This way, the operator prevents redundant calculations later after forking. Without this materialization step, all branches may re-execute the same calculations.

Materialization generally means that the operator performs all deferred calculations on the input data set and writes the data to the distributed file system (into a temporal table). It creates a fresh, clean copy of the data. Generally speaking, you should trust Radoop on handling the materialization of the data sets. The software optimizes operations by accumulating calculations of consecutive operators into the minimum number of distributed jobs. The cost-based optimizer only writes the data to the disk, if it is necessary or if it prevents multiple execution of the same operation. This feature dramatically increases performance and decreases storage requirements. In rare cases should you override this behaviour by explicitly telling Radoop when to write the data to the disks.

One use case for setting the *do_not_materialize* parameter to true, is when you are low on free disk space and you want to minimize disk space usage. This is a trade-off between disk space usage and execution time.

Input:

- **input**

Output:

- **output 1**

Parameters:

- *do not materialize* (expert): If this expert parameter is set to true, the operator does not materialize the input data set before branching. Please read the operator help about this option.
   - *Default value*: false

8.11.2 Subprocess (Radoop)

Synopsis

This operator contains a process within a process.

Description

This is a simple operator chain which can have an arbitrary number of inner operators. The main purpose of this operator is to reduce process complexity and introduce a structure to the whole process.

Input:

- **in 1**

Output:

- **out 1**

8.12 Process Control - In-Memory Processing

8.12.1 In-Memory Subprocess (Full)

Synopsis

Runs in-memory subprocess iterations on data partitions.

Description

This meta operator can be used to create a RapidMiner subprocess inside the Radoop Nest. The operator splits its input data set into partitions (chunks) that fit into the memory. In each iteration it fetches the data of one partition into the memory and executes its subprocess on this ExampleSet. Rows are randomly and uniformly distributed among the partitions, hence, ExampleSets in each iterations should roughly require the same amount of memory. After the required number of iterations, the whole data set will be processed by the operator chain.

If you connect an ExampleSet to one of the output ports, it will generate a HadoopExampleSet. The data in each iteration will be appended to the underlying Hive table. In a typical use case, you perform a complex preprocessing subprocess on a large dataset - processing one partition in each iteration -, then write the rows back to the distributed file system. Other type of IOObjects are delivered in a Collection on the chain's output port.

You control the partitioning by choosing from the following two methods:

- *Fixed number of iterations*. The data is randomly splitted into the specified number of partitions. In this case you can explicitly control the number of iterations. However, if you have constantly growing data, you have to keep an eye on the partitions' size, as they should always fit into the memory.
- *Fixed partition size*. You specify the estimated number of rows in a partition. This is the preferred method if you expect the data set to grow constantly, as you can explicitly control the size of the data that should fit into the operative memory. This method first counts the number of rows to get the required number of partitions.

Optionally, a macro can be generated for the loop that increments after every iteration. The *set iteration macro* parameter should be set to true to define the iteration macro. The name and the start value of the macro can be specified by the *macro name* and *macro start value* parameters respectively.

Please note that you cannot specify a seed value for the random generator that the sampling uses. This means that you may get different result each time you run this operator. Generating deterministic pseudo-random values in a distributed environment is far from a trivial task. You can always build a custom, deterministic sampling process with the help of a unique ID attribute, Generate Attributes and Filter Examples operators.

Input:

- **example set input**
- **input 1**

Output:

- **output 1**

Parameters:

- *set iteration macro* (expert): Selects if in each iteration a macro with the current iteration number is set.
  - *Default value*: false
- *macro name* (expert): The name of the iteration macro.
  - *Default value*: iteration
  - *Depends on*: set iteration macro = true
- *macro start value* (expert): The number which is set for the macro in the first iteration.
  - *Default value*: 1
  - *Depends on*: set iteration macro = true
- *partitioning method* (expert): Select a method for partitioning the data set to chunks that fit into the memory.
  - *Default value*: fixed_chunk_size
- *number of iterations*: The data will be partitioned into the specified number of roughly equal sized partitions. Each iteration processes one partition that should fit into the memory.
  - *Default value*: 30
  - *Depends on*: partitioning method = fixed_number_of_iteration
- *chunk size*: The data will be partitioned into chunks with roughly the specified number of rows.Each iteration processes one partition that should fit into the memory.
  - *Default value*: 50000
  - *Depends on*: partitioning method = fixed_chunk_size

8.12.2 In-Memory Subprocess (Sample)

Synopsis

Runs an in-memory subprocess on sampled data.

Description

This meta operator can be used to create a RapidMiner subprocess inside the Radoop Nest. The subprocess works on data that resides in the client's operative memory. This means that the operator chain takes a random sample of the data set input (extracts an ExampleSet object). After the subprocess completes its operation, the meta operator pushes data on its output back to the cluster. The sample method and the sample size can be controlled by the parameters.

The typical use case for this operator is to learn a prediction model on a sampled training data set. You can use any of RapidMiner's hundreds of operators to achieve this task. Every core operator or extension operator (except Radoop operators) is allowed to use, as data sets reside in the operative memory and no task is pushed to the cluster.

You can select from the following sampling methods for the data set inputs:

- *Sample probability*. You specify a sample probability value between 0 and 1. Each example has equal probability to be included in the sample data set. This is a fast and simple method, but you have to be careful when you are dealing with constantly growing data. Your data sample in this case will also grow and you may end up running out of memory.

- *Absolute sample size*. You specify the number of examples for the sample data set. Please note that this is only a close estimate of the sample. The sample probability for each example will be the ratio of this number and the data set size. This method is slower than directly specifying the sample probability, but is much safer if your large data set is growing constantly.
- *Balanced data - sample probability per class*. You specify a separate probability value for each class. This method requires an attribute with the 'label' role. Examples of a class that is missing from the list are not included in the sample data set (sample probability is considered 0 for them).
- *Balanced data - absolute sample size per class*. You specify a separate sample size estimate for each class. This method requires an attribute with the 'label' role. Examples of a class that is missing from the list are not included in the sample data set (sample size is considered 0 for them). The sample probability for a class will be the ratio of the specified size and the number of rows for this class in the full data set.

Please note that you cannot specify a seed value for the random generator that the sampling uses. This means that you may get different result each time you run this operator. Generating deterministic pseudo-random values in a distributed environment is far from a trivial task. You can always build a custom, deterministic sampling process with the help of a unique ID attribute, Generate Attributes and Filter Examples operators.

Input:

- **example set input**
- **input 1**

Output:

- **output 1**

Parameters:

- *sample*: Determines how the amount of data is specified.
  - *Default value*: absolute
- *balance data* (expert): If you need to sample differently for examples of a certain class, you might check this.
  - *Default value*: false
- *sample size*: The estimated number of examples which should be sampled. A sample probabilty for each example is calculated based on this value.
  - *Default value*: 50000
  - *Depends on*: sample = absolute, balance data = false
- *sample probability*: The sample probability for each example.
  - *Default value*: 0.05
  - *Depends on*: sample = probability, balance data = false
- *sample size per class*: The estimated sample size per class.
  - *Depends on*: sample = absolute, balance data = true
- *sample probability per class*: The fraction per class.
  - *Depends on*: sample = probability, balance data = true
- *case sensitive* (expert): Indicates whether the specified class names should be considered case sensitive or not.
  - *Default value*: true
  - *Depends on*: balance data = true

8.13 Process Control - Loop

8.13.1 Loop (Radoop)

Synopsis

Performs its inner operators k times.

Description

Performs its inner operators for the defined number of times. Optionally, a macro can be defined that increments after every iteration. To use such a macro, set the *set_iteration_macro* parameter to true and choose a name for the iteration macro. You can access the current value of this macro in any operators inside the subprocess. (Please note that during design-time validation, macros cannot be substituted, hence, using them may lead to design-time errors, but that does not mean that the process will fail.)

The results of the subprocess runs are collected and returned as a *Collection* of objects.

This operator is a general looping operator. For some specific tasks, there are special looping operators, like the *Loop Attributes* operator that loops through the specified subset of the attributes of the input data set.

Input:

- **input 1**

Output:

- **output 1**

Parameters:

- *set iteration macro* (expert): Selects if in each iteration a macro with the current iteration number is set.
  - *Default value*: false
- *macro name* (expert): The name of the iteration macro.
  - *Default value*: iteration
  - *Depends on*: set iteration macro = true
- *macro start value* (expert): The number which is set for the macro in the first iteration.
  - *Default value*: 1
  - *Depends on*: set iteration macro = true
- *iterations*: Number of iterations
  - *Default value*: 1
- *limit time* (expert): If checked, the loop will be aborted at last after a specified time.
  - *Default value*: false
- *timeout* (expert): Timeout in minutes
  - *Default value*: 1
  - *Depends on*: limit time = true

8.13.2 Loop Attributes (Radoop)

Synopsis

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 matching the filter expression.

Input:

- **example set**

Output:

- **example set**
- **result 1**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.

  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *iteration macro*: The name of the macro which holds the name of the current feature in each iteration.
  - *Default value*: loop_attribute

8.14 Radoop

8.14.1 Radoop Nest

Synopsis

This is the main operator for running processes on Hadoop.

Description

The cluster settings should be provided here and all further Radoop operators can only be used inside this super-operator.

The subprocess you build inside the nest runs on your Hadoop cluster. You can connect IOObjects to the input ports, which will be available inside the nest. *ExampleSet* objects are converted into *HadoopExampleSet* objects. The data that is stored in the memory for an ExampleSet is pushed to the hard disks of the cluster. Hence, data inside the nest is not stored in the memory, but on the distributed file system. Other IOObjects, like *Models* are propagated the same way inside and outside the nest.

You can process the data on your cluster with the Radoop process you build inside the nest. During execution the process usually starts MapReduce jobs that perform the desired operations on the data. The output data is also written to the distributed file system. A single job may complete several operators' work. Radoop automatically optimizes the process and tries to use the minimum number of jobs and I/O operations.

The output ports of the Radoop Nest delivers the IOObjects that you connect to them inside the nest. HadoopExampleSet objects are converted back to ExampleSet objects. This means, that the underlying data from the distributed file system is fetched into the client machine's operative memory. The ExampleSet in the memory may then be further processed by the remaining RapidMiner process. You can control the size of the data that is fetched into the memory from the distributed file system, since you do not want to run out of memory. Hence, you either fetch a sample of a data set to the memory, or you only connect relatively small data sets to an output port of the Radoop Nest, like aggregated results that fit into the memory.

Input:

- **input 1**

Output:

- **output 1**

Parameters:

- *connection*: Radoop connection
- *table prefix*: Table prefix for temporary objects on the cluster to be easily distinguishable from permanent objects. These objects are automatically deleted after the process completes if cleaning is set to true. Default value can be changed by a global property.
  - *Default value*: Radoop
- *hive file format*: Default file format for the created Hive tables
  - *Default value*: Default Format
  - *Depends on*: Hive version of the connection is Hive server
- *impala file format*: Default file format for the created Impala tables
  - *Default value*: Default format
  - *Depends on*: Hive version of the connection is Impala
- *reload impala metadata* (expert): Call invalidate metadata statement on the selected tables or the whole database if table are not specified. This reloads the metadata in Impala from the Hive metastore so you can use all Hive tables and views in your process.
  - *Default value*: false
  - *Depends on*: Hive version of the connection is Impala
- *tables to reload* (expert): Call invalidate metadata on certain tables or the whole database if tables are not specified. You should consider setting this parameter if your database contains a large number of tables.
  - *Depends on*: reload impala metadata = true
- *cleaning* (expert): Clean temporary tables after finish
  - *Default value*: true
- *auto convert* (expert): Push example set input data to the cluster automatically
  - *Default value*: true

8.15 Transformation - Aggregation

8.15.1 Aggregate

Synopsis

Performs one of the aggregation functions (count, sum...) known from SQL on the data set (with optional grouping).

Description

This operator creates a new data set from the input 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 multiple attributes (similar to the group-by clause known from SQL). In this case a new example will be created for each group. Please note that the HAVING clause known from SQL can be simulated by an additional *Filter Examples* operator following this one.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *use default aggregation*: If checked you can select a default aggregation function for a subset of the attributes.
  - *Default value*: false
- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
  - *Depends on*: use default aggregation = true
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single, use default aggregation = true
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset, use default aggregation = true
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression, use default aggregation = true
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression, use default aggregation = true
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression, use default aggregation = true

- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type, use default aggregation = true
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type, use default aggregation = true
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type, use default aggregation = true
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block
  - *Depends on*: attribute filter type = block_type, use default aggregation = true
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type, use default aggregation = true
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type, use default aggregation = true
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter, use default aggregation = true
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
  - *Depends on*: use default aggregation = true
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
  - *Depends on*: use default aggregation = true
- *default aggregation function*: The type of the used aggregation function.
  - *Default value*: average
  - *Depends on*: use default aggregation = true
- *aggregation attributes*: The attributes which should be aggregated.
- *group by attributes*: Performs a grouping by the values of the attributes whose names match the given regular expression.

8.16 Transformation - Attribute Set Reduction and Transformation

8.16.1 Add Noise

Synopsis

This operator adds noise to the given HadoopExampleSet by adding random attributes to it and by adding noise to the existing attributes. The operator also creates a NoiseModel.

Description

With the Add Noise operator you can choose the attributes for which customized noise should be added. This operator can add noise to the label attribute or to the regular attributes separately. In case of a numerical label the given label noise (specified by the *label_noise* parameter) 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 label noise parameter defines the probability to randomly change the nominal label value. In case of adding noise to regular attributes the default attribute noise parameter simply defines the standard deviation of normal distributed noise without using the attribute value range. Using the parameter list is also possible for setting different noise levels for different attributes (by using the *noise* parameter). However, it is not possible to add noise to nominal attributes.

The Add Noise operator can add random attributes to the ExampleSet. The number of random attributes is specified by the *random_attributes* parameter. New random attributes are simply filled with random data which is not correlated to the label at all. The offset and linear factor parameters are available for adjusting the values of new random attributes.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, Example set matching at least one selected attribute., specified attribute

Output:

- **example set output**
- **original**
- **preprocessing model**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression

- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *random attributes*: Adds this number of random attributes.
  - *Default value*: 0
- *label noise*: Add this percentage of a numerical label range as a normal distributed noise or probability for a nominal label change.
  - *Default value*: 0.05
- *default attribute noise*: The standard deviation of the default attribute noise.
  - *Default value*: 0.0
- *additional noise* (expert): List of noises for each attribute.

- *offset*: Offset added to the values of each random attribute
  - *Default value*: 0.0
- *linear factor*: Linear factor multiplicated with the values of each random attribute
  - *Default value*: 1.0

8.16.2 Generate Attributes

Synopsis

This operator constructs new user defined attributes from expressions.

Description

This operator constructs new attributes from the attributes of the input table and arbitrary constants. The attribute names mig

The expression can be any HiveQL (SQL-like) expression. The Hive Expression Editor dialog helps you to construct a valid expression. Just click on the small calculator icon next to the *attribute_expression* parameter textfield. The dialog will open and you can build the expression easily. Below, you find a list of the operators and functions you can use to build the HiveQL expression.

By default, the operator automatically validates the attribute expressions using the remote connection to the Hive server during design-time. The meta data on the output port shows precisely the expected output data types. However, due to the remote connection, this adds some latency to the design perspective, as a change in any operator before the Generate Attributes operator in the process causes some remote calls to revalidate the attribute expressions. If this latency is unacceptable for you, uncheck the *auto_validate* parameter to prevent these automatic remote calls. In this case, however, this operator cannot predict the types of the generated attributes, hence, it assumes them to be *nominals* during design-time. The types can be explicitly defined easily with the help of a *Type Conversion* operator that follows this operator and sets the data types of the generated attributes. The *auto_validate* parameter has no effect during the process execution.

Supported Expressions

The following operators and functions are supported:

- ! a - Logical not
- a != b - Returns TRUE if a is not equal to b
- a % b - Returns the remainder when dividing a by b
- a & b - Bitwise and
- a | b - Bitwise or
- ~ n - Bitwise not
- a * b - Multiplies a by b
- a + b - Returns a+b
- a - b - Returns the difference a-b
- a / b - Divide a by b
- a < b - Returns TRUE if a is less than b
- a <= b - Returns TRUE if a is not greater than b

- a <> b - Returns TRUE if a is not equal to b
- a = b - Returns TRUE if a equals b and false otherwise
- a == b - Returns TRUE if a equals b and false otherwise
- a > b - Returns TRUE if a is greater than b
- a >= b - Returns TRUE if a is not smaller than b
- a ^ b - Bitwise exclusive or
- abs(x) - returns the absolute value of x
- acos(x) - returns the arc cosine of x if -1<=x<=1 or NULL otherwise
- a and b - Logical and
- array(n0, n1...) - Creates an array with the given elements
- array_contains(array, value) - Returns TRUE if the array contains value.
- ascii(str) - returns the numeric value of the first character of str
- asin(x) - returns the arc sine of x if -1<=x<=1 or NULL otherwise
- assert_true(condition) - Throw an exception if ‘condition’ is not true.
- atan(x) - returns the atan (arctan) of x (x is in radians)
- avg(x) - Returns the mean of a set of numbers
- bin(n) - returns n in binary
- binary(a) - cast a to binary
- ceil(x) - Find the smallest integer not smaller than x
- ceiling(x) - Find the smallest integer not smaller than x
- coalesce(a1, a2, ...) - Returns the first non-null argument
- collect_set(x) - Returns a set of objects with duplicate elements eliminated
- concat(str1, str2, ... strN) - returns the concatenation of str1, str2, ... strN or concat(bin1, bin2, ... binN) - returns the concatenation of bytes in binary data bin1, bin2, ... binN
- concat_ws(separator, str1, str2, ...) - returns the concatenation of the strings separated by the separator.
- context_ngrams(expr, array<string1, string2, ...>, k, pf) estimates the top-k most frequent n-grams that fit into the specified context. The second parameter specifies a string of words that specify the positions of the n-gram elements, with a null value standing in for a ‘blank’ that must be filled by an n-gram element.
- conv(num, from_base, to_base) - convert num from from_base to to_base
- corr(x,y) - Returns the Pearson coefficient of correlation between a set of number pairs
- cos(x) - returns the cosine of x (x is in radians)
- count(*) - Returns the total number of retrieved rows, including rows containing NULL values. count(expr) - Returns the number of rows for which the supplied expression is non-NULL. count(DISTINCT expr[, expr...]) - Returns the number of rows for which the supplied expression(s) are unique and non-NULL.
- covar_pop(x,y) - Returns the population covariance of a set of number pairs
- covar_samp(x,y) - Returns the sample covariance of a set of number pairs
- create_union(tag, obj1, obj2, obj3, ...) - Creates a union with the object for given tag
- date_add(start_date, num_days) - Returns the date that is num_days after start_date.

- date_sub(start_date, num_days) - Returns the date that is num_days before start_date.
- datediff(date1, date2) - Returns the number of days between date1 and date2
- day(date) - Returns the date of the month of date
- dayofmonth(date) - Returns the date of the month of date
- degrees(x) - Converts radians to degrees
- a div b - Divide a by b rounded to the long integer
- e() - returns E
- elt(n, str1, str2, ...) - returns the n-th string
- ewah_bitmap(expr) - Returns an EWAH-compressed bitmap representation of a column.
- ewah_bitmap_and(b1, b2) - Return an EWAH-compressed bitmap that is the bitwise AND of two bitmaps.
- ewah_bitmap_empty(bitmap) - Predicate that tests whether an EWAH-compressed bitmap is all zeros
- ewah_bitmap_or(b1, b2) - Return an EWAH-compressed bitmap that is the bitwise OR of two bitmaps.
- exp(x) - Returns e to the power of x
- explode(a) - separates the elements of array a into multiple rows, or the elements of a map into multiple rows and columns
- field(str, str1, str2, ...) - returns the index of str in the str1,str2,... list or 0 if not found
- find_in_set(str,str_array) - Returns the first occurrence of str in str_array where str_array is a comma-delimited string. Returns null if either argument is null. Returns 0 if the first argument has any commas.
- floor(x) - Find the largest integer not greater than x
- from_unixtime(unix_time, format) - returns unix_time in the specified format
- get_json_object(json_txt, path) - Extract a json object from path
- hash(a1, a2, ...) - Returns a hash value of the arguments
- hex(n or str) - Convert the argument to hexadecimal
- histogram_numeric(expr, nb) - Computes a histogram on numeric 'expr' using nb bins.
- hour(date) - Returns the hour of date
- test in(val1, val2...) - returns true if test equals any valN
- in_file(str, filename) - Returns true if str appears in the file
- instr(str, substr) - Returns the index of the first occurance of substr in str
- isnotnull a - Returns true if a is not NULL and false otherwise
- isnull a - Returns true if a is NULL and false otherwise
- json_tuple(jsonStr, p1, p2, ..., pn) - like get_json_object, but it takes multiple names and return a tuple. All the input parameters and output column types are string.
- lcase(str) - Returns str with all characters changed to lowercase
- length(str | binary) - Returns the length of str or number of bytes in binary data
- like(str, pattern) - Checks if str matches pattern
- ln(x) - Returns the natural logarithm of x
- locate(substr, str[, pos]) - Returns the position of the first occurance of substr in str after position pos

- log([b], x) - Returns the logarithm of x with base b
- log10(x) - Returns the logarithm of x with base 10
- log2(x) - Returns the logarithm of x with base 2
- lower(str) - Returns str with all characters changed to lowercase
- lpad(str, len, pad) - Returns str, left-padded with pad to a length of len
- ltrim(str) - Removes the leading space characters from str
- map(key0, value0, key1, value1...) - Creates a map with the given key/value pairs
- map_keys(map) - Returns an unordered array containing the keys of the input map.
- map_values(map) - Returns an unordered array containing the values of the input map.
- max(expr) - Returns the maximum value of expr
- min(expr) - Returns the minimum value of expr
- minute(date) - Returns the minute of date
- month(date) - Returns the month of date
- named_struct(name1, val1, name2, val2, ...) - Creates a struct with the given field names and values
- negative a - Returns -a
- ngrams(expr, n, k, pf) - Estimates the top-k n-grams in rows that consist of sequences of strings, represented as arrays of strings, or arrays of arrays of strings. 'pf' is an optional precision factor that controls memory usage.
- not a - Logical not
- a or b - Logical or
- parse_url(url, partToExtract[, key]) - extracts a part from a URL
- parse_url_tuple(url, partname1, partname2, ..., partnameN) - extracts N (N>=1) parts from a URL. It takes a URL and one or multiple partnames, and returns a tuple. All the input parameters and output column types are string.
- percentile(expr, pc) - Returns the percentile(s) of expr at pc (range: [0,1]).pc can be a double or double array
- percentile_approx(expr, pc, [nb]) - For very large data, computes an approximate percentile value from a histogram, using the optional argument [nb] as the number of histogram bins to use. A higher value of nb results in a more accurate approximation, at the cost of higher memory usage.
- pi() - returns pi
- a pmod b - Compute the positive modulo
- positive a - Returns a
- pow(x1, x2) - raise x1 to the power of x2
- power(x1, x2) - raise x1 to the power of x2
- radians(x) - Converts degrees to radians
- rand([seed]) - Returns a pseudorandom number between 0 and 1
- reflect(class,method[,arg1[,arg2..]]) calls method with reflection
- str regexp regexp - Returns true if str matches regexp and false otherwise
- regexp_extract(str, regexp[, idx]) - extracts a group that matches regexp
- regexp_replace(str, regexp, rep) - replace all substrings of str that match regexp with rep

- repeat(str, n) - repeat str n times
- reverse(str) - reverse str
- str rlike regexp - Returns true if str matches regexp and false otherwise
- round(x[, d]) - round x to d decimal places
- rpad(str, len, pad) - Returns str, right-padded with pad to a length of len
- rtrim(str) - Removes the trailing space characters from str
- second(date) - Returns the second of date
- sentences(str, lang, country) - Splits str into arrays of sentences, where each sentence is an array of words. The 'lang' and'country' arguments are optional, and if omitted, the default locale is used.
- sign(x) - returns the sign of x)
- sin(x) - returns the sine of x (x is in radians)
- size(a) - Returns the size of a
- space(n) - returns n spaces
- split(str, regex) - Splits str around occurances that match regex
- sqrt(x) - returns the square root of x
- stack(n, cols...) - turns k columns into n rows of size k/n each
- std(x) - Returns the standard deviation of a set of numbers
- stddev(x) - Returns the standard deviation of a set of numbers
- stddev_pop(x) - Returns the standard deviation of a set of numbers
- stddev_samp(x) - Returns the sample standard deviation of a set of numbers
- str_to_map(text, delimiter1, delimiter2) - Creates a map by parsing text
- struct(col1, col2, col3, ...) - Creates a struct with the given field values
- substr(str, pos[, len]) - returns the substring of str that starts at pos and is of length len or substr(bin, pos[, len]) - returns the slice of byte array that starts at pos and is of length len
- substring(str, pos[, len]) - returns the substring of str that starts at pos and is of length len or substring(bin, pos[, len]) - returns the slice of byte array that starts at pos and is of length len
- sum(x) - Returns the sum of a set of numbers
- tan(x) - returns the tangent of x (x is in radians)
- to_date(expr) - Extracts the date part of the date or datetime expression expr
- trim(str) - Removes the leading and trailing space characters from str
- ucase(str) - Returns str with all characters changed to uppercase
- unhex(str) - Converts hexadecimal argument to string
- union_map(col) - aggregate given maps into a single map
- unix_timestamp([date[, pattern]]) - Returns the UNIX timestamp
- upper(str) - Returns str with all characters changed to uppercase
- var_pop(x) - Returns the variance of a set of numbers
- var_samp(x) - Returns the sample variance of a set of numbers

- variance(x) - Returns the variance of a set of numbers
- weekofyear(date) - Returns the week of the year of the given date. A week is considered to start on a Monday and week 1 is the first week with >3 days.
- xpath(xml, xpath) - Returns a string array of values within xml nodes that match the xpath expression
- xpath_boolean(xml, xpath) - Evaluates a boolean xpath expression
- xpath_double(xml, xpath) - Returns a double value that matches the xpath expression
- xpath_float(xml, xpath) - Returns a float value that matches the xpath expression
- xpath_int(xml, xpath) - Returns an integer value that matches the xpath expression
- xpath_long(xml, xpath) - Returns a long value that matches the xpath expression
- xpath_number(xml, xpath) - Returns a double value that matches the xpath expression
- xpath_short(xml, xpath) - Returns a short value that matches the xpath expression
- xpath_string(xml, xpath) - Returns the text contents of the first xml node that matches the xpath expression
- year(date) - Returns the year of date

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *attribute name*: Attribute name
- *attribute expression*: Expression for the new attribute
- *new attributes*: List of generated attributes.
- *auto validate* (expert): Validate the attribute expression automatically using the remote Hive connection. This is required for appropriate meta data generation during design-time.
  - *Default value*: true

8.16.3 Generate Copy

Synopsis

Copies a single attribute.

Description

Adds a copy of a single attribute in the input data set.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *attribute name*: Attribute to copy
- *new attribute name*: New attribute name

8.16.4 Generate ID

Synopsis

Adds a new id attribute to the example set, each example is tagged with a random double number.

Description

This operator adds an ID attribute to the given example set. Each example is tagged with a random double number.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

8.16.5 Generate Rank

Synopsis

This operator generates the (dense) rank of each row within the given partition.

Description

The rank of a row is one plus the count of ranks before the given row. The dense rank of a row is one plus the count of distinct ranks before the given row.

The operator adds a design-time warning, if the *partition_by* parameter list is empty. The reason is that if no grouping (partitioning) is defined with this parameter, the operator will generate a global rank attribute after sorting the whole data set. This can be a very slow operation for a large data set and is probably not what you want to do. If you wan to add a unique ID variable to the data set, use the *Generate ID* operator.

Please note that this operator is only supported starting with Hive 0.11. If you use an older server release, please update, if you want to use this operator.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *attribute name*: Attribute name
- *partition by*: Ordered list of the partitioning attributes.
- *order by*: The attributes and sorting directions which should be used to determine the order of the data before the ranking is applied.
- *dense rank*: Dense Rank returns the rank of rows within the partition of a result set, without any gaps in the ranking.
  - *Default value*: false

8.16.6 Principal Component Analysis

Synopsis

This operator performs a Principal Component Analysis (PCA) using the covariance matrix. The user can specify the amount of variance to cover in the original data while retaining the best number of principal components. The user can also specify manually the number of principal components.

Description

Principal component analysis (PCA) is an attribute reduction procedure. It is useful when you have obtained data on a number of attributes (possibly a large number of attributes), and believe that there is some redundancy in those attributes. In this case, redundancy means that some of the attributes are correlated with one another, possibly because they are measuring the same construct. Because of this redundancy, you believe that it should be possible to reduce the observed attributes into a smaller number of principal components (artificial attributes) that will account for most of the variance in the observed attributes.

Principal Component Analysis is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated attributes into a set of values of uncorrelated attributes called principal components. The number of principal components is less than or equal to the number of original attributes. This transformation is defined in such a way that the first principal component's variance is as high as possible (accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it should be orthogonal to (uncorrelated with) the preceding components.

Please note that PCA is sensitive to the relative scaling of the original attributes. This means that whenever different attributes have different units (like temperature and mass); PCA is a somewhat arbitrary method of analysis. Different results would be obtained if one used Fahrenheit rather than Celsius for example.

The improved algorithm parameter indicates if the operator should use Hive UDAF for the execution. Set this parameter to false if you want to avoid this behaviour (in this case the execution will be much slower).

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, *expects:* HadoopExampleSet

Output:

- **example set output**

- **original**
- **preprocessing model**

Parameters:

- *dimensionality reduction*: Indicates which type of dimensionality reduction should be applied
  - *Default value*: keep variance
- *variance threshold*: Keep all components with a cumulative variance smaller than the given threshold.
  - *Default value*: 0.95
  - *Depends on*: dimensionality reduction = keep variance
- *number of components*: Keep this number of components.
  - *Default value*: 1
  - *Depends on*: dimensionality reduction = fixed number
- *improved algorithm* (expert): Indicates if the improved calculating method should be used.
  - *Default value*: true

8.16.7 Reorder Attributes

Synopsis

This operator allows to reorder regular attributes of a HadoopExampleSet. Reordering can be done alphabetically, by user specification (including Regular Expressions) or with a reference ExampleSet.

Description

> This operator allows to change the ordering of regular attributes of an *ExampleSet*. Therefore different order modes may be selected in the parameter *sort_mode*.

If sort mode alphabetically is chosen attributes are sorted alphabetically according to the selected *sort_direction*. If sort mode user specified is chosen the user can specify rules that define how attributes should be ordered. If sort mode reference data is chosen the input *HadoopExampleSet* will be sorted according to the order of reference ExampleSet. Note that special attributes will not be considered by this operator. If they should be considered set them to regular with *Set Role* operator.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet
- **reference_data**:
  - *optional:* ExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *sort mode*: Ordering method that should be applied.

  - *Default value*: user specified

- *sort direction*: Sort direction for attribute names.
  - *Default value*: ascending
- *attribute ordering*: Rules to order attributes.
  - *Depends on*: sort mode = user specified
- *handle unmatched*: Defines the behavior for unmatched attributes.
  - *Default value*: append
  - *Depends on*: sort mode in {user specified, reference data}
- *use regular expressions* (expert): If checked attribute orders will be evaluated as regular expressions.
  - *Default value*: false
  - *Depends on*: sort mode = user specified

8.16.8 Select Attributes

Synopsis

This operator allows to select which attributes should be part of the resulting table.

Description

This operator selects which attributes of a Hive table should be kept and which are removed. Therefore, different filter types may be selected in the parameter attribute filter type and only attributes fulfilling this condition type are selected. The rest will be removed from the table. There's a global switch to invert the outcome, so that all attributes which would have been originally discarded will be kept and vice versa. To invert the decision, use the invert selection parameter.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression

- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false

8.16.9 Select Random Attributes

Synopsis

This operator allows to select a random subset of attributes that should be part of the resulting table.

Description

This operator selects a random subset of the regular attributes that should be kept. The double parameter defines the expected ratio of the selected attributes, it specifies the probability that an attribute is included. If a low probability value would cause that no regular attribute would be selected, the operator still adds a randomly selected one to the result data set (if there is any). You can specify a random seed to get deterministic result.

Special attributes are all kept.

Please note that as the operator cannot predict the result attribute set during design-time, it simply propagates the metadata on its input port to its output port.

The operator can be of great use inside loops, e.g. for training an ensemble model on different attribute subsets (like a Random Forest algorithm). For determistic result inside a loop, you should use the iteration macro as the random seed.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *filter ration*: Relative size of the attribute set
  - *Default value*: 0.6
- *use local random seed* (expert): Indicates if a local random seed should be used.
  - *Default value*: false
- *local random seed* (expert): Specifies the local random seed
  - *Default value*: 1992
  - *Depends on*: use local random seed = true

8.17 Transformation - Custom Script

8.17.1 Hive Script

Synopsis

Runs an arbitrary Hive QL script.

Description

This operator is for advanced users who want to write their own Hive QL scripts for data manipulation. The script can refer to the example sets on its input ports as *##inputtable1##*, *##inputtable2##*, etc. The script should start with the following clause (do not change this line): *CREATE VIEW ##outputtable## AS*.

By default, the operator automatically validates the script using the remote connection to the Hive server during design-time. The meta data on the output port shows precisely the expected output data set structure. However, due to the remote connection, this adds some latency to the design perspective, as the change in any operator before the Hive Script operator in the process causes some remote calls to revalidate the user-defined Hive script and generate the output meta data. If this latency is unacceptable for you, uncheck the *auto_validate* parameter to prevent these automatic remote calls. In this case, however, this operator cannot predict the output data set structure, hence, it simply propagates its input meta data to its output port. The *auto_validate* parameter has no effect during the process execution.

The operator automatically copies the attribute roles of the first input data set to the output. An attribute of the output data set that exists in the first input data set keeps its role.

Input:

- **example set input 1**:
  - requires at least 1 HadoopExampleSet (collection)

Output:

- **example set output**

Parameters:

- *hivescript*: Script:
  - *Default value*:

```
--
-- Please do not change the first line of the script. The default script does
-- nothing, just delivers the input data set on the output port. You may
-- refer to other input data sets in a similar way to the last line.
--
CREATE VIEW ##outputtable## AS
SELECT *
FROM ##inputtable1##
```

- *auto validate* (expert): Validate the script automatically using the remote Hive connection. This is required for appropriate meta data generation during design-time.
  - *Default value*: true
- *user defined functions* (expert): Add User-Defined Functions (UDFs) that can be used in the script. The temporary functions are defined by their name and the class name that implements it. Please note that the class must exist both in Hadoop's classpath and Hive's classpath.
- *preserve binominal mappings* (expert): Keep the mapping for the binominal attributes. If set to false, they will be converted to nominals. If true, you you should not introduce new values to the binominal attribute other than the positive value, the negative value and missings.
  - *Default value*: false

8.17.2 Pig Script

Synopsis

Runs an arbitrary Pig script.

Description

This operator is for advanced users who want to write their Pig scripts to manipulate their data directly in the process data flow. This operator also enables Pig experts to integrate their existing Pig Latin code into a Radoop process. To be able to do this, please note the following instructions about handling input and output data in your Pig script.

As a Pig Latin script may work on multiple inputs and produce multiple outputs, the operator may have arbitrary number of inputs and outputs. Just connect an input example set to the free input port if you want to use it in you Pig script. Similarly, you can connect an output port if you want to produce another output with this operator. Your Pig script should specify the data on these output ports.

The first input data set should be referred in the Pig script using the following keywords: *##inputfile1##*, *##inputstorage1##*, *##inputcolumns1##*. Before running the operator, Radoop will replace these keywords with the appropriate values to produce a valid Pig script. The *##inputfile1##* keyword refers to the directory that contains the data of the first input example set. The *##inputstorage1##* keyword will be replaced by the appropriate Pig storage handler class (with their arguments like the field separator) that the software determines automatically for this input data set. The *##inputcolumns1##* keyword refers to the list of column name and column type pairs of the input example table. The conversion of RapidMiner (and Hive) column types to Pig data types is done automatically. The default Pig script of the operator shows a simple line that loads an input example set using these keywords. The relation name here can be any arbitrary name.

operator_input1 = LOAD '##inputfile1##' USING ##inputstorage1## AS (##inputcolumns1##);

You can load all input example sets the same way, just use the next integer number in the keywords instead of 1. Only in very rare cases should you consider changing this template for loading your input data.

You can later insert a column list of the your first input example set into the script with the keyword *##inputcolumnaliases1##*. E.g. this may be used in a *FOREACH* expression, like in the following default script code line.

operator_output1 = FOREACH operator_input1 GENERATE ##inputcolumnaliases1##;

Otherwise, you may refer to the columns of an example set by their RapidMiner attribute name (this is true if you load your data with the default template (*##inputcolumns1##*)).

Generating output data sets is somewhat similar to handling input data sets. You should use the *STORE* Pig expression to produce an output. Here, the relation name is not arbitrary, you should use *operator_output1* alias for the first output, *operator_output2* for the second, etc. The keywords that handle these output connections are similar to the input keywords: *##outputfile1##*, *##outputstorage1##*. The *##outputfile1##* keyword will be replaced by a name (and path) for the connected first output. The *##outputstorage1##* keyword refers to the Pig storage class for the first output. The default Pig script produces the first output example set with the following line:

STORE operator_output1 INTO '##outputfile1##' USING ##outputstorage1##;

You probably should never change this template for producing output data sets. The alias (relation name: *operator_output1*) for the output is important because Radoop will look for this name to describe the schema of the first output. It will use this schema to create the output data set after converting the Pig data types back to RapidMiner attribute types (the log may contain warnings for possibly unintended data type conversions). You will get a process setup error during design time (see the Problems view), if the sofware is not able to analyze such an output relation. The reason for this can be that the script has errors, or you have not defined a relation for a connected output port.

The full script will be processed by the Pig engine when the process reaches this operator. However, for generating output metadata and validating the script, Radoop will execute part of the script during design time. To be explicit, the lines before the first *STORE* expression will be processed and validated. You may get an error message for an invalid script or for invalid or missing output relations.

The operator integrates Pig 0.11.2 release.

Input:

- **example set 1**:
  - requires at least 1 HadoopExampleSet (collection)

Output:

- **example set 1**

Parameters:

- *pigscript*: Script:
  - *Default value*:

```
--
-- This line will load your data from the input Hive table. Modify only
-- if you want to explicitly change the input column aliases or their types.
--
operator_input1 = LOAD '##inputfile1##' USING ##inputstorage1## AS (##inputcolumns1##);

--
-- Write your own script here.
--
-- The following line is an example on how to refer to the input table
-- columns. The operator will substitute the column list of the input here.
--
operator_output1 = FOREACH operator_input1 GENERATE ##inputcolumnaliases1##;

--
-- This line stores the output. You can access it on the output port of
-- the operator. You have to use the operator_output<n> aliases to
-- produce outputs for the operator.
--
STORE operator_output1 INTO '##outputfile1##' USING ##outputstorage1##;
```

- *preserve binominal mappings* (expert): Keep the mappings of the binominal attributes. If set to false, they will be converted to nominals. If true, you you should not introduce new values to the binominal attribute other than the positive value, the negative value and missings.
  - *Default value*: false

8.18 Transformation - Data Cleansing

8.18.1 Replace Missing Values

Synopsis

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", and "average" which is applied to the non missing attribute values of the example set. The replenishment "value" indicates that the user defined parameter should be used for the replacement. If you explicitly specify a value, do not use any quotes in it. If you want to use a quote inside a nominal string value, please use an escape character before it (\).

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**

- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'

  - *Depends on*: attribute filter type = numeric_value_filter

- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *replace method*: Replace method
  - *Default value*: Value
- *replace value*: Value
  - *Depends on*: replace method = Value

8.19 Transformation - Filtering

8.19.1 Filter Example Range

Synopsis

This only allows the first N examples to pass.

Description

This operator selects the first N rows of the input table. The other examples will be removed from the input example set.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *row limit*: Row limit
  - *Default value*: 1000

8.19.2 Filter Examples

Synopsis

This operator only allows examples to pass if they fulfill a specified condition.

Description

This operator takes a data set as input and returns a data set including only the rows that fulfill a condition. For *attribute_value_condition* the parameter string can be any condition that you would write in SQL after a WHERE statement. For a HiveQL function reference you can check the Hive Expression Editor of the *Generate Attributes* operator.

Various predefined conditions are available for filtering examples. Users can select any of them by setting the *condition class* parameter. Examples satisfying the selected condition are passed to the output port, others are removed. Following conditions are available:

- **all**: if this option is selected, no examples are removed.
- **correct_predictions**: if this option is selected, only those examples make it to the output port that have correct predictions i.e. the value of the label attribute and prediction attribute are the same.
- **wrong_predictions**: if this option is selected, only those examples make to the output port that have wrong predictions i.e. the value of the label attribute and prediction attribute are not the same.
- **no_missing_attributes**: if this option is selected, only those examples make it to the output port that have no missing values in any of their attribute values. Missing values or null values are usually shown by '?' in RapidMiner.
- **missing_attributes**: if this option is selected, only those examples make it to the output port that have some missing values in their attribute values.
- **no_missing_labels**: if this option is selected, only those examples make it to the output port that do not have a missing value in their label attribute value. Missing values or null values are usually shown by '?' in RapidMiner.
- **missing_label**: if this option is selected, only those examples make to the output port that have missing value in their label attribute value.
- **attribute_value_filter**: if this option is selected, another parameter (parameter string)is enabled in the Parameter View.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *condition class*: Implementation of the condition.
  - *Default value*: all
- *parameter string*: Parameter string for the condition, e.g. 'attribute=value' for the AttributeValueFilter.
  - *Depends on*: condition class = attribute_value_filter

8.19.3 Remove Duplicates

Synopsis

This operator removes duplicates from a data set.

Description

The Remove Duplicates operator keeps only one row of the row sets where all column values are the same. The NULL value is considered a unique value, hence, it is only considered equal to another NULL value.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

8.19.4 Sample

Synopsis

Creates a random sample from a data set by drawing a fraction.

Description

Takes a random sample from a data set.

You can choose from the following sampling methods:

- *Sample probability*. You specify a sample probability value between 0 and 1. Each example has equal probability to be included in the sample data set. This is a fast and simple method, but you should note that with a constantly growing input data set, the output will also grow over time.
- *Absolute sample size*. You specify the number of examples for the sample data set. Please note that this is only a close estimate of the sample. The sample probability for each example will be the ratio of this number and the data set size. This method is slower than directly specifying the sample probability, but you explicitly limit the size of your sample.
- *Balanced data - sample probability per class*. You specify a separate probability value for each class. This method requires an attribute with the 'label' role. Examples of a class that is missing from the list are not included in the sample data set (sample probability is considered 0 for them).
- *Balanced data - absolute sample size per class*. You specify a separate sample size estimate for each class. This method requires an attribute with the 'label' role. Examples of a class that is missing from the list are not included in the sample data set (sample size is considered 0 for them). The sample probability for a class will be the ratio of the specified size and the number of rows for this class in the full data set.

Please note that you cannot specify a seed value for the random generator that the sampling uses. This means that you may get different result each time you run this operator. Generating deterministic pseudo-random values in a distributed environment is far from a trivial task. You can always build a custom, deterministic sampling process with the help of a unique ID attribute, Generate Attributes and Filter Examples operators.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**

- **original**

Parameters:

- *sample*: Determines how the amount of data is specified.
  - *Default value*: absolute
- *balance data* (expert): If you need to sample differently for examples of a certain class, you might check this.
  - *Default value*: false
- *sample size*: The estimated number of examples which should be sampled. A sample probabilty for each example is calculated based on this value.
  - *Default value*: 50000
  - *Depends on*: sample = absolute, balance data = false
- *sample probability*: The sample probability for each example.
  - *Default value*: 0.05
  - *Depends on*: sample = probability, balance data = false
- *sample size per class*: The estimated sample size per class.
  - *Depends on*: sample = absolute, balance data = true
- *sample probability per class*: The fraction per class.
  - *Depends on*: sample = probability, balance data = true
- *case sensitive* (expert): Indicates whether the specified class names should be considered case sensitive or not.
  - *Default value*: true
  - *Depends on*: balance data = true

8.19.5 Split Data

Synopsis

Splits a data set into partitions.

Description

This operators splits the input data set into the specified number of random partitions.

Input:

- **example set**:
  - *expects:* HadoopExampleSet

Output:

- **partition 1**

Parameters:

- *equal sized partitions*: Indicates that the data rows should be uniformly distributed among partitions, you only specify the number of partitions.
  - *Default value*: false

- *number of partitions*: Number of partitions. Data rows are uniformly distributed among them.
  - *Default value*: 3
  - *Depends on*: equal sized partitions = true
- *partitions*: The partitions that should be created.
  - *Depends on*: equal sized partitions = false

8.20 Transformation - Miscellaneous

8.20.1 Materialize Data

Synopsis

This operators materializes its input data set before passing it onto its output port.

Description

Materialization means that the operator performs all deferred calculations on the input data set and writes the data to the distributed file system (into a temporal table). It creates a fresh, clean copy of the data. Generally speaking, you should trust Radoop on handling the materialization of the data sets. The software optimizes operations by accumulating calculations of consecutive operators into the minimum number of distributed jobs. The cost-based optimizer only writes the data to the disk, if it is necessary or if the materialization prevents multiple execution of the same operation. This feature dramatically increases performance and decreases storage requirements. In rare cases should you override this behaviour by using an explicit materialization operator. If you want to write your data into a permanent table, please use the *Store* operator.

If the *force* parameter is set to true, then the cost-based estimates and optimization treshold is ignored and the data is written to the disk for sure. If it is set to false, then the operator considers the cost estimates of the deferred calculations of previous operators and decides whether to write the data to the disk or not.

One case for using this operator may be that there is some kind of randomization in the process and multiple runs may result in different result. With a materialization step, you can be 100% sure that the preceding operations will not be performed multiple times (hence, you avoid possibly delivering different results on different branches). However, Radoop, by itself, knows which operators may be undeterministic. If the process forks after such an operator, then the software materializes the data, before proceeding with the execution of the two or more branches (see also *Multiply* operator).

Another justifiable reason for using this operator may be troubleshooting. You may encounter with a rare, strange error, e.g. a Hive error that occurs in an operator of your process. In this case you should use the Breakpoint feature to localize the error. This may be tricky, because you usually cannot be sure that the error lies in the operator, in which the process fails, as the cause may be in one of the deferred calculations of earlier operators. However, if you have managed to find the cause of the error and you are sure that your process should otherwise succeed - so the error is indeed a Hive error caused by complex calculations -, you may try to create a workaround by putting a Materialize Data operator right before the operator in which you think the process fails. This way, you may be able to create a workaround, if the process succeeds with the Materialize Data operator added. If it still fails you should continue the breakpoint method, or test the operation in another way.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**

Parameters:

- *force* (expert): Force materialization and ignore cost-based optimization.
  - *Default value*: true

8.21 Transformation - Name and Role Modification

8.21.1 Rename Attributes

Synopsis

This operator can be used to rename an attribute.

Description

> This operator can be used to rename an attribute of the input table. Please keep in mind, that attribute names have to be unique. Please note that all attribute names inside the Radoop Nest

are automatically converted to lowercase, special characters are replaced by underscores and collision with certain reserved keywords may be avoided by an underscore suffix. You will notice and easily track these changes during design time by checking the meta data on the output port.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *old name*: The old name of the attribute.
- *new name*: The new name of the attribute.
- *rename additional attributes*: A list that can be used to define additional attributes that should be renamed.

8.21.2 Set Role

Synopsis

This operator can be used to change the attribute role (regular, special, label, id...).

Description

This operator can be used to change the role of an attribute of the input. If you want to change the attribute name you should use the Rename operator. The target role 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 membership 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.

Please be aware that roles have to be unique! Assigning a non regular role the second time will cause the first attribute to be dropped from the example set. If you want to keep this attribute, you have to change it's role first.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *name*: The name of the attribute whose role should be changed.
- *target role*: The target role of the attribute (only changed if parameter change_attribute_type is true).
  - *Default value*: regular
- *set additional roles*: This parameter defines additional attribute role combinations.

8.22 Transformation - Rotation

8.22.1 Pivot Table

Synopsis

This operator rotates a HadoopExampleSet by aggregating and grouping multiple examples of same groups to single examples.

Description

> This operator performs the pivoting operation on the input data set. The index attribute parameter specifies the attribute whose values are used to identify the examples inside the groups.

The values of this attribute are used to name the group attributes which are created during the pivoting. The group attributes parameter specifies the grouping attributes (i.e. the attributes which identify examples belonging to the groups).

The rows of the output table contain the aggregated values of the aggregation attributes, calculated with the given aggregation function.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *index attribute*: Attribute which differentiates examples inside a group.
- *aggregation attribute*: Specifies the attribute which should be aggregated.
- *aggregation function*: The type of the used aggregation function.
  - *Default value*: sum
- *group attributes*: Attributes that group the examples which form one example after pivoting.
- *max unique indexes* (expert): The maximum number of distinct indexes that the operator should process.
  - *Default value*: 10000

8.23 Transformation - Set Operations

8.23.1 Join

Synopsis

Builds the join of two data sets using the id or any other key attributes of the sets in order to identify the same examples.

Description

Builds the join of two example sets using the id or any other key attributes of the sets. The attributes of the result example set will consist of the union set or the union list (depending on parameter settings, duplicate attributes will be removed or renamed) of both feature sets. In case of removing duplicate attributes, the one from the left example set will be taken. The attribute from the right set will be discarded. Special attributes of the second input example set which do exist in the first example set will simply be skipped.

Input:

- **left**:
  - *expects:* HadoopExampleSet, specified attribute
- **right**:
  - *expects:* HadoopExampleSet, specified attribute

Output:

- **join**

Parameters:

- *remove double attributes* (expert): Indicates if double attributes should be removed or renamed
  - *Default value*: false

- *join type*: Specifies which join should be executed.
  - *Default value*: inner
- *use id attribute as key*: Indicates if the id attribute is used for join.
  - *Default value*: false
- *key attributes*: The attributes which shall be used for join. Attributes which shall be matched must be of the same type.
  - *Depends on*: use id attribute as key = false

8.23.2 Union

Synopsis

Union combines the data from multiple data sets into a single data set.

Description

Union appends the data of the second, third etc. input data set to the first input data set. All input data set must have the same attributes (their number and names and types should match). The output data set contains all rows from the input data sets (duplicates are not eliminated).

Input:

- **example set 1**:
  - requires at least 1 HadoopExampleSet (collection)

Output:

- **example set output**

8.24 Transformation - Sorting

8.24.1 Sort

Synopsis

This operator sorts the given data set.

Description

This operator sorts the given data set by one or more attributes specified by the parameters. The examples are sorted according to the natural order of the values of these attribute either in increasing or in decreasing direction, depending on the setting of sorting direction.

Please note that sorting a large data set with this operator may take very long time. You should usually use it only on smaller data sets, like one that has limited number of rows after a *Filter Example Range* operator.

Input:

- **example set input**:

  - *expects:* HadoopExampleSet, specified attribute, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *sort attribute*: Indicates the attribute which should be used for determining the sorting.
- *sort direction*: Indicates the direction of the sorting.
  - *Default value*: increasing
- *additional attributes*: List of the additional sorting attributes and the corresponding sorting directions

8.25 Transformation - Type Conversion

8.25.1 Remap Binominals

Synopsis

This operator modifies the internal value mapping of binominal attributes according to the specified negative and positive values or discovers the values automatically.

Description

The Remap Binominals operator modifies the internal mapping of binominal attributes according to the specified positive and negative values or discovers the values automatically. The positive and negative values are specified by the positive value and negative value parameters respectively.

Please note that Radoop is not generally aware of the internal mapping of the binominal attributes as RapidMiner does. If the mapping is unknown, the specified values will be the considered as the mapping without any error checking. If you are not sure about the concrete values in the ExampleSet, you can force the checking by selecting the *validate_values* expert parameter. If this is set to true, the process will throw an error when a row violates the specified mapping, i.e. it contains another value. If the internal mapping is already known, then it is replaced by the specified one.

By selecting the "Discover mapping automatically" option Radoop will discover and set the mapping for the attribute automatically (this takes time). This is useful if you don't know the exact values in the Example Set.

Please note that this operator changes the internal mapping so the changes are not explicitly visible in the ExampleSet. This operator can be applied only on binominal attributes. Please note that if there is a nominal attribute in the ExampleSet with only two possible values, this operator will still not be applicable on it. This operator requires the attribute to be explicitly defined as binominal in the meta data by using the *Type Conversion operator*.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, Example set matching at least one selected attribute.

Output:

- **example set output**
- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: binominal
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: binominal
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: value_matrix_start
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter

- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *discover mapping automatically*: Automatically discover the mapping for the selected attributes.
  - *Default value*: false
- *negative value*: The first/negative/false value.
  - *Depends on*: discover mapping automatically = false
- *positive value*: The second/positive/true value.
  - *Depends on*: discover mapping automatically = false
- *validate values* (expert): Validate the specified positive and negative values. If false, the specified values are considered correct and your process may fail if they are not. If true, Radoop will validate them but this takes extra processing time.
  - *Default value*: false
  - *Depends on*: discover mapping automatically = false

8.25.2 Type Conversion

Synopsis

Converts the type of a Hive table attribute.

Description

This operator converts the type of one or more attributes in the data set. Currently it only supports conversion between integer, double and string.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet, specified attribute, specified attribute, specified attribute

Output:

- **example set output**
- **original**

Parameters:

- *attribute*: Attribute
- *new type (2.1.0)*: Type
  - *Default value*: BIGINT
  - *Depends on*: operator version is at most 2.1.1
- *type conversions (2.1.0)*: List of type conversions.
  - *Depends on*: operator version is at most 2.1.1

- *new type (since 2.2.0)*: Type
  - *Default value*: integer
  - *Depends on*: operator version is at least 2.2.0
- *type conversions (since 2.2.0)*: List of type conversions.
  - *Depends on*: operator version is at least 2.2.0

8.26 Transformation - Value Modification

8.26.1 Declare Missing Value

Synopsis

Declares a missing numeric or nominal value on a selected subset, which will be replaced by NULL.

Description

The given value will be replaced with NULL throughout the specified subset, so it will be treated as a missing value by subsequent operators.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

   - *Depends on*: use except expression = true, attribute filter type = regular_expression

- *value type*: The value type of the attributes.
   - *Default value*: attribute_value
   - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
   - *Default value*: false
   - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
   - *Default value*: time
   - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
   - *Default value*: attribute_block
   - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
   - *Default value*: false
   - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
   - *Default value*: value_matrix_row_start
   - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
   - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
   - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
   - *Default value*: false
- *attribute value*: This parameter defines the missing value

8.26.2 Normalize

Synopsis

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.

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: attribute_value
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type
- *except value type* (expert): Except this value type.
  - *Default value*: time
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: attribute_block

  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: value_matrix_row_start
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *normalize method* (expert): Transformation method
  - *Default value*: Z-transformation
- *min value* (expert): Min value
  - *Default value*: 0.0
  - *Depends on*: normalize method = Range transformation
- *max value* (expert): Max value
  - *Default value*: 1.0
  - *Depends on*: normalize method = Range transformation

8.26.3 Replace

Synopsis

This operator replaces parts of the values of nominal attributes.

Description

This operator replaces parts of the string values of all nominal attributes it is applied on. The **attribute filter type** gives the possibility to restrict them. For each value of each attribute it is checked if the regular expression of **replace what** matches the string. Each matching part of the string will be replaced by the value of the **replace_what** parameter. The replacement might be empty and can contain capturing groups.

Please keep in mind that although regular expressions are much more powerful than simple strings, you might simply enter characters to search for.

Examples

The attribute contains the values “color red”, “color green” and “color blue”.

- replacing “color” by “” yields: ” red”, ” green”, ” blue”
- replacing “color” by “colour” yields: “colour red”, “colour green”, “colour blue”
- replacing “color\s” by “” yields: “red”, “green”, “blue”
- replacing “\s+” by “_” yields: “color_red”, “color_green”, “color_blue”
- replacing “color\s(.*)” by “$1” yields: “red”, “green”, “blue”
- replacing ”.*\s(.*)” by “$1” yields: “red”, “green”, “blue”

Input:

- **example set input**:
  - *expects:* HadoopExampleSet

Output:

- **example set output**
- **original**

Parameters:

- *attribute filter type*: The condition specifies which attributes are selected or affected by this operator.
  - *Default value*: all
- *attribute*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = single
- *attributes*: The attribute which should be chosen.
  - *Depends on*: attribute filter type = subset
- *regular expression*: A regular expression for the names of the attributes which should be kept.
  - *Depends on*: attribute filter type = regular_expression
- *use except expression* (expert): If enabled, an exception to the specified regular expression might be specified. Attributes of matching this will be filtered out, although matching the first expression.
  - *Default value*: false
  - *Depends on*: attribute filter type = regular_expression
- *except regular expression* (expert): A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.
  - *Depends on*: use except expression = true, attribute filter type = regular_expression
- *value type*: The value type of the attributes.
  - *Default value*: nominal
  - *Depends on*: attribute filter type = value_type
- *use value type exception* (expert): If enabled, an exception to the specified value type might be specified. Attributes of this type will be filtered out, although matching the first specified type.
  - *Default value*: false
  - *Depends on*: attribute filter type = value_type

- *except value type* (expert): Except this value type.
  - *Default value*: file_path
  - *Depends on*: use value type exception = true, attribute filter type = value_type
- *block type*: The block type of the attributes.
  - *Default value*: single_value
  - *Depends on*: attribute filter type = block_type
- *use block type exception* (expert): If enabled, an exception to the specified block type might be specified.
  - *Default value*: false
  - *Depends on*: attribute filter type = block_type
- *except block type* (expert): Except this block type.
  - *Default value*: single_value
  - *Depends on*: use block type exception = true, attribute filter type = block_type
- *numeric condition*: Parameter string for the condition, e.g. '>= 5'
  - *Depends on*: attribute filter type = numeric_value_filter
- *invert selection*: Indicates if only attributes should be accepted which would normally filtered.
  - *Default value*: false
- *include special attributes*: Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.
  - *Default value*: false
- *replace what*: A regular expression specifying what should be replaced.
- *replace by*: The replacement for the region matched by the regular expression. Possibly including capturing groups.