RapidMiner Radoop 7

Operator Reference Manual

rapidminer

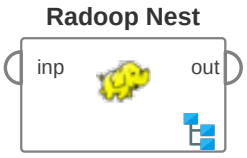
RapidMiner Radoop 7
Operator Reference Manual

August 15, 2016

RapidMiner GmbH
www.rapidminer.com

Contents

Radoop Nest



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 Ports

input 1 *(inp)*

Output Ports

output 1 *(out)*

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.

change sample size Override default output sample size for this subprocess.

sample size Sample size for Hadoop data sets on the Nest output, zero means full sample.

hive file format Default file format for the created Hive tables

impala file format Default file format for the created Impala tables

reload impala metadata 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.

tables to reload 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.

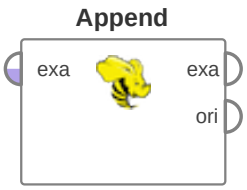
cleaning Clean temporary tables after finish

auto convert Push example set input data to the cluster automatically

1 Data Access

1.1 Hive

Append into Hive



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

use default database Use the database specified in the connection of the Radoop Nest.

database Name of the database being used.

tablename Target Hive table.

create Create table if it does not exist

insert nulls Insert NULL values for missing columns if the target table exists already with further columns.

partitioning Enable insert into partitioned table. Dynamic partitioning is performed based on the target Hive tables' partitioning columns

max partitions 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).

Retrieve from Hive

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 Ports

output ***(out)***

Parameters

use default database Use the database specified in the connection of the Radoop Nest.

database Name of the database being used.

table Input table.

filter clause 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.

Store in Hive

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 on the Hadoop Data view), 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 Ports

input *(inp)*

Output Ports

output ***(out)***

Parameters

use default database Use the database specified in the connection of the Radoop Nest.

database Name of the database being used.

tablename Hive table:

dropfirst Forced table creation. For external tables the target directory will be cleaned.

external table Store in external table (specify the location explicitly).

location Location of the external table data: a directory on the HDFS, or S3 (use s3n:// prefix), etc.

partition by Ordered list of partitioning columns.

max partitions 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).

custom storage Use custom storage format. You may specify the target table's storage format explicitly.

custom storage handler Use a custom storage handler. You specify storage handler class name explicitly.

storage handler Custom storage handler. It must exist in the CLASSPATH of the Hive server.

row format Target table row format. Please note that older Hive versions may not support all row format settings.

fields delimiter Custom field delimiter character.

fields escape char Escape character that can be used to escape the field delimiter character. Leave empty for no escape character. Use '\\' for the '\' character.

collection delimiter Custom delimiter character that separates collection items (COLLECTION data type).

map keys delimiter Custom delimiter character that separates map keys (MAP data type).

lines delimiter Custom delimiter character that separates records (lines).

null character Character for storing a NULL value.

serde class name Custom SerDe class name. It must exist in the CLASSPATH of the Hive server.

serde properties User defined SerDe parameter. These case sensitive key-value pairs are passed to the table's SerDe.

hive file format Target table file format. Please note that older Hive versions may not support all file format types.

impala file format Target table file format. Please note that older Impala versions may not support all file format types.

input format Custom input format class name. It must exist in the CLASSPATH of the Hive server. Example: 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'

output format Custom output format class name. It must exist in the CLASSPATH of the Hive server. Example: 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

1.2 Read

Read CSV

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 Ports

output *(out)*

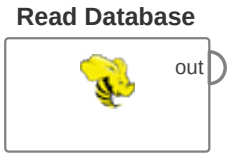
Parameters

Configuration Configure the operator with wizard

override file 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.

override location Source location.

Read Database



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 Ports

output *(out)*

Parameters

define connection Indicates how the database connection should be specified.

connection A predefined database connection.

database system The used database system.

database url The URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database'

username The database username.

password The password for the database.

jndi name JNDI name for a data source.

define query Specifies whether the database query should be defined directly, through a file or implicitely by a given table name.

query An SQL query.

query file A file containing an SQL query.

use default schema If checked, the user's default schema will be used.

schema name The schema name to use, unless use_default_schema is true.

table name A database table.

prepare statement If checked, the statement is prepared, and '?'-parameters can be filled in using the parameter 'parameters'.

parameters Parameters to insert into '?' placeholders when statement is prepared.

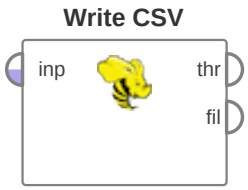
datamanagement Determines, how the data is represented internally.

temporary table Temporary table

saved table name Table name

1.3 Write

Write CSV



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 Ports

input *(inp)*

Output Ports

through *(thr)*

file *(fil)*

Parameters

csv file Name of the file to write the data in.

column separator The column separator.

write attribute names Indicates if the attribute names should be written as first row.

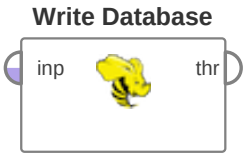
quote nominal values Indicates if nominal values should be quoted with double quotes.

format date attributes Indicates if date attributes are written as a formated string or as milliseconds past since January 1, 1970, 00:00:00 GMT

append to file Indicates if new content should be appended to the file or if the pre-existing file content should be overwritten.

encoding The encoding used for reading or writing files.

Write Database



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 Ports

input *(inp)*

Output Ports

through *(thr)*

Parameters

define connection Indicates how the database connection should be specified.

connection A predefined database connection.

database system The used database system.

database url The URL connection string for the database, e.g. 'jdbc:mysql://foo.bar:portnr/database'

username The database username.

password The password for the database.

jndi name JNDI name for a data source.

use default schema If checked, the user's default schema will be used.

schema name The schema name to use, unless use_default_schema is true.

table name A database table.

overwrite mode Indicates if an existing table should be overwritten or if data should be appended.

set default varchar length Set varchar columns to default length.

default varchar length Default length of varchar columns.

add generated primary keys Indicates whether a new attribute holding the auto generated primary keys is added to the result set.

db key attribute name The name of the attribute for the auto generated primary keys

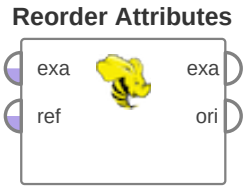
batch size The number of examples which are written at once with one single query to the database. Larger values can greatly improve the speed - too large values however can drastically decrease the performance. Additionally, some databases have restrictions on the maximum number of values written at once.

Varchar size Varchar length

2 Blending

2.1 Attributes

Reorder Attributes



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 Ports

example set input ***(exa)***

reference data ***(ref)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

sort mode Ordering method that should be applied.

sort direction Sort direction for attribute names.

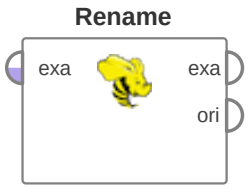
attribute ordering Rules to order attributes.

handle unmatched Defines the behavior for unmatched attributes.

use regular expressions If checked attribute orders will be evaluated as regular expressions.

2.1.1 Names and Roles

Rename



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

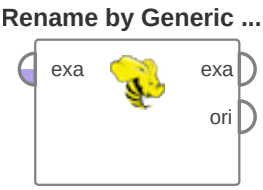
Parameters

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.

Rename by Generic Names



This operator can be used to rename attributes generically with an incremental index.

Description

This operator can be used to rename attributes of the input table generically. It requires a name stem which will be followed by an incrementally generated index. 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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

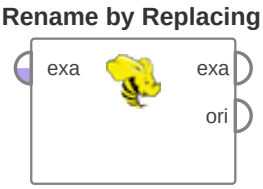
numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

generic name stem The selected attributes will be named by this parameter, followed by an incremental index.

Rename by Replacing



This operator can be used to rename attributes. It uses a regular expression to choose involved part(s) of the attribute name.

Description

This operator can be used to replace parts of the chosen attributes' names. These parts are selected by a regular expression. The replacing text part comes from the *replace by* parameter, which might include capturing groups of the defined regular expression as well. These can be accessed with syntax $1, $2, $3... 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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

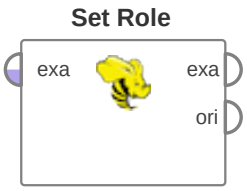
invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

replace what A regular expression defining what should be replaced in the attribute names.

replace by A replacing text for regular expression matches.

Set Role



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

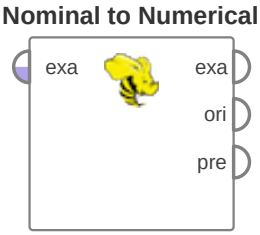
name The name of the attribute whose role should be changed.

target role The target role of the attribute (only changed if parameter change_attribute_type is true).

set additional roles This parameter defines additional attribute role combinations.

2.1.2 Types

Nominal to Numerical



Converts a nominal Hive table attribute and its values to numerical.

Description

This operator converts the type of one or more nominal attributes in the data set to numerical type. The value of the attribute(s) will also be transformed by the selected *coding type* method. Further information about coding types can be found at parameter description.

If *dummy coding* or *effect coding* is selected, you can set up a list of comparison groups. The attribute created from a comparison group will not appear in the output example set.

You can set the maximal number of distinct nominal values. This can be useful in case you want to avoid creating vast amount of attributes, or long processing time. If the limitation is exceeded, an error message will arise and process will be stopped.

Input Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

preprocessing model *(pre)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

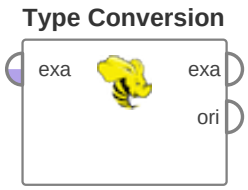
coding type The coding of the numerical attributes. Unique integers coding simply assigns an integer value to each distinct nominal values. Dummy coding creates a new attribute for each distinct value. Within these attributes, it indicates value match with value 1 and mismatch with value 0. Comparison groups might be set for all attributes. These will not appear in the output example set. Effect coding works just like dummy coding, but it always requires a filled up comparison group list, and it sets the value to -1 if the nominal value corresponds to the comparison group.

use comparison groups If checked, for each selected attribute in the input set a value has to be specified as comparison group, which will not appear in the final result set.

comparison groups The value which becomes the comparison group.

distinct values limit Maximum number of distinct nominal values in any attribute.

Type Conversion



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

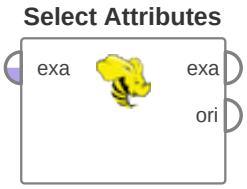
attribute Attribute

new type Type

type conversions List of type conversions.

2.1.3 Selection

Select Attributes



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

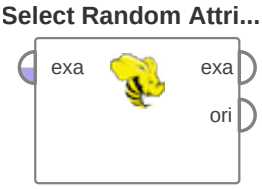
except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

Select Random Attributes



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

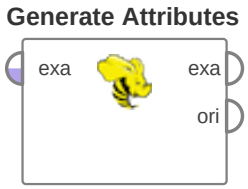
filter ration Relative size of the attribute set

use local random seed Indicates if a local random seed should be used.

local random seed Specifies the local random seed

2.1.4 Generation

Generate Attributes



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 might be used as variables in the expression. 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 view, 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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

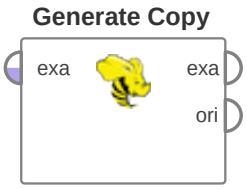
attribute name Attribute name

attribute expression Expression for the new attribute

new attributes List of generated attributes.

auto validate Validate the attribute expression automatically using the remote Hive connection. This is required for appropriate meta data generation during design-time.

Generate Copy



Copies a single attribute.

Description

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

Input Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

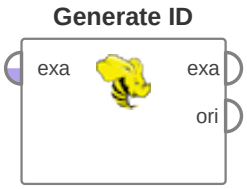
original ***(ori)***

Parameters

attribute name Attribute to copy

new attribute name New attribute name

Generate ID



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 Ports

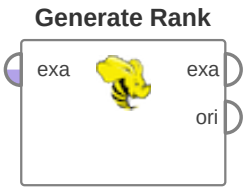
example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Generate Rank



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

attribute name Attribute name

partition by Ordered list of the partitioning attributes.

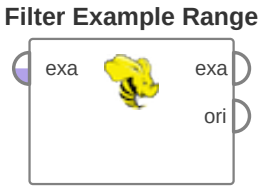
order by The attributes and sorting directions which should be used to determine the order of the data before the ranking is applied.

dense rank Dense Rank returns the rank of rows within the partition of a result set, without any gaps in the ranking.

2.2 Examples

2.2.1 Filter

Filter Example Range



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 Ports

example set input ***(exa)***

Output Ports

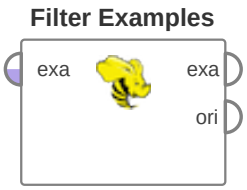
example set output ***(exa)***

original ***(ori)***

Parameters

row limit Row limit

Filter Examples



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 panel.

Input Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

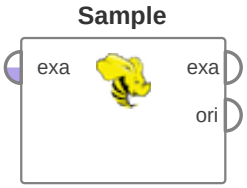
Parameters

condition class Implementation of the condition.

parameter string Parameter string for the condition, e.g. 'attribute=value' for the AttributeValueFilter.

2.2.2 Sampling

Sample



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

sample Determines how the amount of data is specified.

balance data If you need to sample differently for examples of a certain class, you might check this.

sample size The estimated number of examples which should be sampled. A sample probabilty for each example is calculated based on this value.

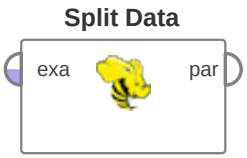
sample probability The sample probability for each example.

sample size per class The estimated sample size per class.

sample probability per class The fraction per class.

case sensitive Indicates whether the specified class names should be considered case sensitive or not.

Split Data



Splits a data set into partitions.

Description

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

Input Ports

example set *(exa)*

Output Ports

partition 1 *(par)*

Parameters

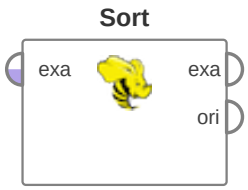
equal sized partitions Indicates that the data rows should be uniformly distributed among partitions, you only specify the number of partitions.

number of partitions Number of partitions. Data rows are uniformly distributed among them.

partitions The partitions that should be created.

2.2.3 Sort

Sort



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

sort attribute Indicates the attribute which should be used for determining the sorting.

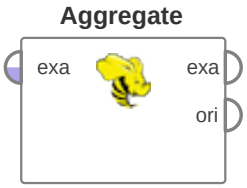
sort direction Indicates the direction of the sorting.

additional attributes List of the additional sorting attributes and the corresponding sorting directions

2.3 Table

2.3.1 Grouping

Aggregate



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

use default aggregation If checked you can select a default aggregation function for a subset of the attributes.

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

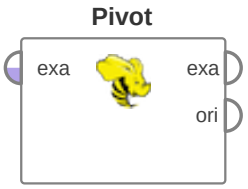
default aggregation function The type of the used aggregation function.

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.

2.3.2 Rotation

Pivot



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

index attribute Attribute which differentiates examples inside a group.

aggregation attribute Specifies the attribute which should be aggregated.

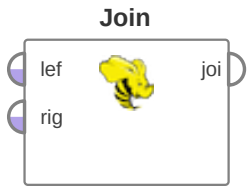
aggregation function The type of the used aggregation function.

group attributes Attributes that group the examples which form one example after pivoting.

max unique indexes The maximum number of distinct indexes that the operator should process.

2.3.3 Joins

Join



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 Ports

left ***(lef)***

right ***(rig)***

Output Ports

join ***(joi)***

Parameters

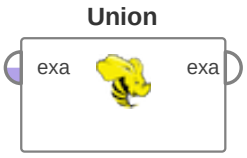
remove double attributes Indicates if double attributes should be removed or renamed

join type Specifies which join should be executed.

use id attribute as key Indicates if the id attribute is used for join.

key attributes The attributes which shall be used for join. Attributes which shall be matched must be of the same type.

Union



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 Ports

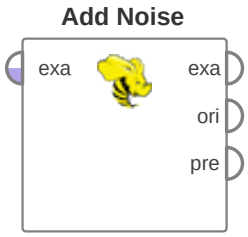
example set 1 ***(exa)***

Output Ports

example set output ***(exa)***

2.4 Values

Add Noise



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

preprocessing model *(pre)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

random attributes Adds this number of random attributes.

label noise Add this percentage of a numerical label range as a normal distributed noise or probability for a nominal label change.

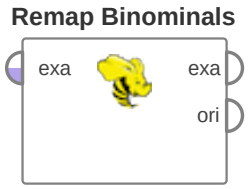
default attribute noise The standard deviation of the default attribute noise.

additional noise List of noises for each attribute.

offset Offset added to the values of each random attribute

linear factor Linear factor multiplicated with the values of each random attribute

Remap Binominals



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

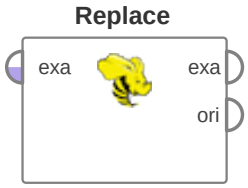
discover mapping automatically Automatically discover the mapping for the selected attributes.

negative value The first/negative/false value.

positive value The second/positive/true value.

validate values 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.

Replace



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

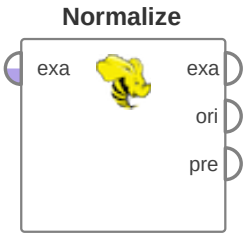
replace what A regular expression specifying what should be replaced.

replace by The replacement for the region matched by the regular expression. Possibly including capturing groups.

3 Cleansing

3.1 Normalization

Normalize



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

preprocessing model *(pre)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

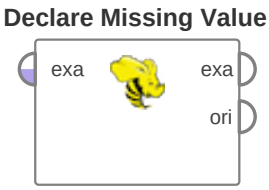
normalize method Transformation method

min value Min value

max value Max value

3.2 Missing

Declare Missing Value



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

attribute value This parameter defines the missing value

mode Select the value type of the missing value

numeric value Defines the missing numerical value

nominal value Defines the missing nominal value

expression value Defines the logical expression for the missing value

auto validate Validate the attribute expression automatically using the remote Hive connection. This is required for appropriate meta data generation during design-time.

Replace Missing Values

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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

original ***(ori)***

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

invert selection Indicates if only attributes should be accepted which would normally filtered.

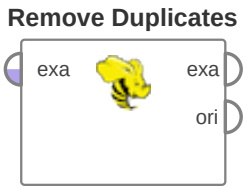
include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

replace method Replace method

replace value Value

3.3 Duplicates

Remove Duplicates



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 Ports

example set input *(exa)*

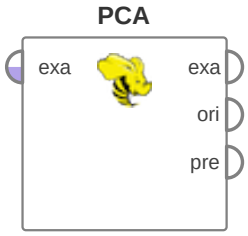
Output Ports

example set output *(exa)*

original *(ori)*

3.4 Dimensionality Reduction

Principal Component Analysis



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

original *(ori)*

preprocessing model *(pre)*

Parameters

dimensionality reduction Indicates which type of dimensionality reduction should be applied

variance threshold Keep all components with a cumulative variance smaller than the given threshold.

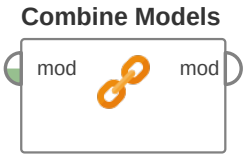
number of components Keep this number of components.

improved algorithm Indicates if the improved calculating method should be used.

4 Modeling

4.1 Predictive

Combine Models



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).

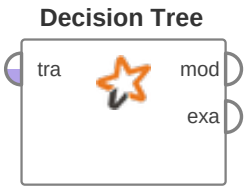
Input Ports

model input 1 ***(mod)***

Output Ports

model output ***(mod)***

Decision Tree



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

Description

Information about the algorithm can be found here: http://spark.apache.org/docs/latest/ml-decision-tree.html

Requirements

The operator has the following dependencies on the cluster:

- Spark assembly version 1.5.0 or later. You can set the Spark version for your connection on the Advanced Connection Panel.

Input Ports

input *(inp)* This port can have a HadoopExampleSet input. It must have label attribute that can be either binominal or polynominal.

Output Ports

model *(mod)* This port delivers the model that is built by the operator.

output *(out)* This port delivers the original input.

Parameters

file format *(selection)* 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Default PARQUET.

- **TEXTFILE** Materializing in Textfile format is supported on all Hive versions.
- **PARQUET** Materializing in Parquet format requires less storage but requires a later Hive version.

criterion *(selection)* Criterion used for information gain calculation. Default Gini.

- **Gini** Gini
- **Entopy** Entropy

minimal gain *(real)* For a node to be split further, the split must improve at least this much (in terms of information gain). Should be in range [0, 1], default 0.1.

maximal depth ***(integer)*** 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. Should be >= 0, default 20. (Depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.)

maximum bins ***(integer)*** Maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node. More bins give higher granularity. Should be >= 2 and >= number of categories in any categorical feature, default 32.

minimal size for split ***(integer)*** For a node to be split further, each of its children must receive at least this number of training instances. Should be >= 1, default 4.

maximum memory in MB ***(integer)*** 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 maximum memory can lead to faster training (if the memory is available) by allowing fewer passes over the data. However, there may be decreasing returns as maximum memory grows since the amount of communication on each iteration can be proportional to maximum memory. Default 256.

use node id cache ***(boolean)*** If this is set to true, the algorithm will avoid passing the current model (tree or trees) to executors on each iteration. Default false.

use binominal mappings ***(boolean)*** If this is set to true, the algorithm will try to avoid discovering the nominal values. This can decrease the execution time noticeably. Enable this checkbox if you want to train a tree on only numerical and binominal features and you have provided a correct mapping for every binominal feature in the training data set. Please note that in this case your input data must not contain missing values. Default false.

driver memory (MB) ***(integer)*** 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 2048.

Decision Tree (MLlib binominal)

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 at *https://spark.apache.org/docs/latest/mllib-decision-tree.html* . 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 Ports

training set *(tra)*

Output Ports

model *(mod)*

exampleSet *(exa)*

Parameters

file format 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in Parquet format requires less storage but requires a later Hive version.

criterion Selects the criterion on which attributes will be selected for splitting.

minimal gain For a node to be split further, the split must improve at least this much (in terms of information gain).

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.

maximum bins Number of bins used when discretizing continuous features.

minimal size for split For a node to be split further, each of its children must receive at least this number of training instances.

maximum memory in MB 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.

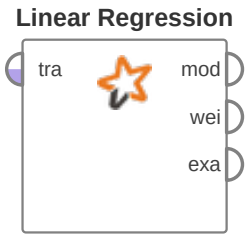
subsampling rate Fraction of the training data used for learning the decision tree.

use node id cache If this is set to true, the algorithm will avoid passing the current model (tree or trees) to executors on each iteration.

use binominal mappings If this is set to true, the algorithm will try to avoid discovering the nominal values. This can decrease the execution time noticeably.Enable this checkbox if you want to train a Tree on only numerical and binominal features and you have provided a correct mapping for every binominal feature in the training data set. Please note that in this case your input data must not contain missing values.

driver memory (MB) 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.

Linear Regression



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 at *https://spark.apache.org/docs/latest/mllib-linear-methods.html#linear-least-squares-lasso-and-ridge-regression* .

Input Ports

training set *(tra)*

Output Ports

model *(mod)*

weights *(wei)*

exampleSet *(exa)*

Parameters

file format 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in Parquet format requires less storage but requires a later Hive version.

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.

number of iterations Number of iterations of gradient descent to run.

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.

convergence to L Set the convergence tolerance of iterations. Default 1E-3. Smaller value will lead to higher accuracy with the cost of more iterations. This parameter is only available in Spark 1.5 or later. For earlier Spark versions it is skipped.

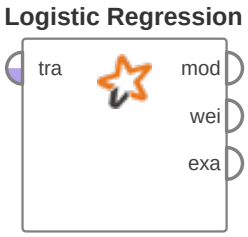
minibatch fractions Fraction of the input data set that should be used for one iteration of SGD. Default 1.0 (corresponding to deterministic/classical gradient descent)

add intercept Set if the algorithm should add an intercept.

use feature scaling 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.

regularization parameter The regularization parameter.

Logistic Regression



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 at *https://spark.apache.org/docs/latest/mllib-linear-methods.html#logistic-regression* .

The operator supports both Stochastic Gradient Descent (SGD) and Limited-memory BFGS (LBFGS) optimizers. Information on the optimizers can be found at *https://spark.apache.org/docs/latest/mllib-optimization.html* .

Input Ports

training set *(tra)*

Output Ports

model *(mod)*

weights *(wei)*

exampleSet *(exa)*

Parameters

file format 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Materializing in Parquet format requires less storage but requires a later Hive version.

optimizer The optimizer to solve the problem. Possible values are SGD (Stochastic Gradient Descent) and LBFGS (Limited-memory BFGS).

number of iterations Number of iterations of gradient descent to run.

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).

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)

convergence to L (L-BFGS) 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.

convergence to L (SGD) Set the convergence tolerance of iterations for SGD. Default 1E-3. Smaller value will lead to higher accuracy with the cost of more iterations. This parameter is only available in Spark 1.5 or later and its value must be lower than 1.0. For earlier Spark versions it is skipped.

number of corrections Set the number of corrections used in the LBFGS update. The value must be greater than 0, by default it is 10. Values of numCorrections less than 3 are not recommended; large values of numCorrections will result in excessive computing time. It is recommended to set this parameter between 3 and 10.

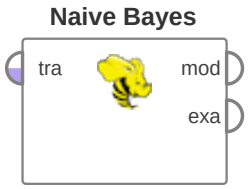
add intercept Set if the algorithm should add an intercept.

use feature scaling 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.

updater 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.

regularization parameter The regularization parameter.

Naive Bayes



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 Ports

training set *(tra)*

Output Ports

model *(mod)*

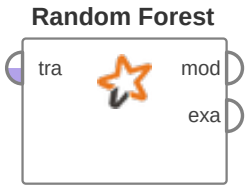
exampleSet *(exa)*

Parameters

laplace correction Use Laplace correction to prevent high influence of zero probabilities.

nominal group size This parameter affects only the performance, not the output. Statistics for a group of nominal attributes are calculated together in a single scan. This is the number of nominal attributes in each group. Increase it for faster learning (fewer scans), decrease if nodes run out of memory (more scans).

Random Forest



Generates a Random Forest for classification of both nominal and numerical data. It is based on the random forest implementation in Spark ML.

Description

Information about the algorithm can be found here: http://spark.apache.org/docs/latest/ml-ensembles.html#random-forests

Requirements

The operator has the following dependencies on the cluster:

- Spark assembly version 1.5.0 or later. You can set the Spark version for your connection on the Advanced Connection Panel.

Input Ports

input *(inp)* This port can have a HadoopExampleSet input. It must have label attribute that can be either binominal or polynominal.

Output Ports

model *(mod)* This port delivers the model that is built by the operator.

output *(out)* This port delivers the original input.

Parameters

file format *(selection)* 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Default PARQUET.

- **TEXTFILE** Materializing in Textfile format is supported on all Hive versions.
- **PARQUET** Materializing in Parquet format requires less storage but requires a later Hive version.

feature subset strategy *(selection)* Selects the feature selection strategy. Default Auto.

- **Auto** Choose automatically for task: If number of trees == 1, set to All If number of trees $>$ 1 (forest), set to Sqrt.
- **All** Use all features
- **Onethird** Use 1/3 ot the features
- **Sqrt** Use sqrt(number of features)

- **Log2** Use log2(number of features)

criterion ***(selection)*** Criterion used for information gain calculation. Default Gini.

- **Gini** Gini
- **Entopy** Entropy

minimal gain ***(real)*** For a node to be split further, the split must improve at least this much (in terms of information gain). Should be in range [0, 1], default 0.1.

maximal depth ***(integer)*** 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. Should be >= 0, default 20. (Depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.)

maximum bins ***(integer)*** Maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node. More bins give higher granularity. Should be >= 2 and >= number of categories in any categorical feature, default 32.

minimal size for split ***(integer)*** For a node to be split further, each of its children must receive at least this number of training instances. Should be >= 1, default 4.

number of trees ***(integer)*** Number of trees to train. Should be >= 1, default 10. (If 1, then no bootstrapping is used, if > 1, then bootstrapping is done.)

maximum memory in MB ***(integer)*** 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 maximum memory can lead to faster training (if the memory is available) by allowing fewer passes over the data. However, there may be decreasing returns as maximum memory grows since the amount of communication on each iteration can be proportional to maximum memory. Default 256.

subsampling rate ***(real)*** Fraction of the training data used for learning each decision tree. Should be in range (0, 1], default 1.

use node id cache ***(boolean)*** If this is set to true, the algorithm will avoid passing the current model (tree or trees) to executors on each iteration. Default false.

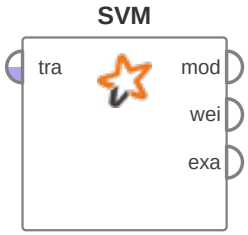
use binominal mappings ***(boolean)*** If this is set to true, the algorithm will try to avoid discovering the nominal values. This can decrease the execution time noticeably. Enable this checkbox if you want to train a tree on only numerical and binominal features and you have provided a correct mapping for every binominal feature in the training data set. Please note that in this case your input data must not contain missing values. Default false.

driver memory (MB) ***(integer)*** 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 2048.

use local random seed ***(boolean)*** Indicates if a local random seed should be used. Default false.

local random seed ***(integer)*** Specifies the local random seed. Should be >= 1, default 1992.

Support Vector Machine



This operator is a Support Vector Machine Learner. It is based on the support vector machine implementation in Spark MLlib.

Description

Support Vector Machine is used to predict a binary response. Detailed information can be found here: http://spark.apache.org/docs/latest/mllib-linear-methods.html#linear-support-vector-machines-svms

The operator supports Stochastic Gradient Descent (SGD) optimizer. Information on SGD optimizer can be found here: http://spark.apache.org/docs/latest/mllib-optimization.html#stochastic-gradient-descent-sgd

Input Ports

input *(inp)* This port can have a HadoopExampleSet input. It must have binominal label.

Output Ports

model *(mod)* This port delivers the model that is built by the operator.

weights *(wei)* This port delivers the weights computed by the operator.

output *(out)* This port delivers the original input.

Parameters

file format *(selection)* 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 Parquet format - in this case no materalization is performed. Please note that you can force materializing in Text/Parquet by using the Store operator or by setting the File Format parameter of the Radoop Nest. Default PARQUET.

- **TEXTFILE** Materializing in Textfile format is supported on all Hive versions.
- **PARQUET** Materializing in Parquet format requires less storage but requires a later Hive version.

number of iterations *(integer)* Number of iterations of gradient descent to run. Default 100.

step size *(real)* The initial step size of SGD for the first step. In subsequent steps, the step size will decrease with stepSize/sqrt(current_iteration_number). Default 1.0.

minibatch fractions *(real)* Fraction of the input data set that should be used for one iteration of SGD. Should be in range [0, 1], default 1.0.

convergence to L *(real)* Set the convergence tolerance of iterations. Smaller value will lead to higher accuracy with the cost of more iterations. This parameter is only available in Spark 1.5 or later. For earlier Spark versions it is skipped. Should be in range [0, 1], default 1.0E-4.

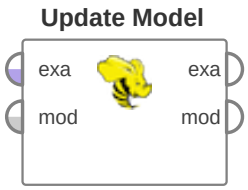
add intercept *(boolean)* Set if the algorithm should add an intercept. Default true.

use feature scaling *(boolean)* 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 feature scaling 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 true.

updater *(selection)* 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 Simple Updater.

- **Simple Updater** Simple Updater
- **L1 Updater** L1 Updater
- **Squared L2 Updater** Squared L2 Updater

Update Model



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 Ports

example set *(exa)*

model *(mod)*

Output Ports

example set *(exa)*

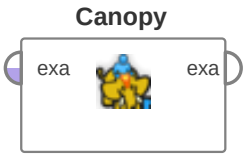
model *(mod)*

Parameters

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

4.2 Segmentation

Canopy



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 Ports

example set input ***(exa)***

Output Ports

example set output ***(exa)***

Parameters

distance function Distance function

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 If set to true, different distance thresholds may be specified for the reducer phase.

reducer t1 distance The reducer's T1 distance metric. If not specified, T1 is used by the reducer.

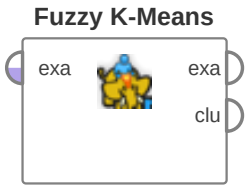
reducer t2 distance The reducer's T2 distance metric. If not specified, T2 is used by the reducer.

canopies min number The minimum size of canopies produced (can be zero)

cluster classification threshold 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.

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.

Fuzzy K-Means



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

cluster model *(clu)*

Parameters

number of clusters Number of clusters

maximum iterations The maximum number of iterations to run, independent of the convergence specified

delta convergence 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)

distance function Distance function

the fuzzification factor 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.

emit most likely cluster A boolean indicating, if true, that the clustering step should only emit the most likely cluster for each clustered point.

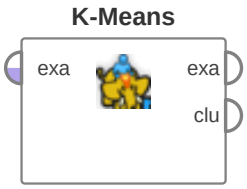
cluster classification threshold Is a clustering strictness / outlier removal parameter. Its value should be between 0 and 1. Vectors having pdf below this value will not be clustered.

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.

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.

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.

K-Means



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 Ports

example set input *(exa)*

Output Ports

example set output *(exa)*

cluster model *(clu)*

Parameters

number of clusters Number of clusters

maximum iterations Maximum number of iterations

delta convergence 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)

distance function Distance function

cluster classification threshold 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.

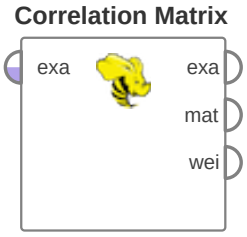
only result If set, clustering returns only (ID, ClusterID) pairs, and removes other attributes. This option removes some overhead, thus, should decrease the processing time.

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.

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.

4.3 Correlations

Correlation Matrix



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 Ports

example set ***(exa)***

Output Ports

example set ***(exa)***

matrix ***(mat)***

weights ***(wei)***

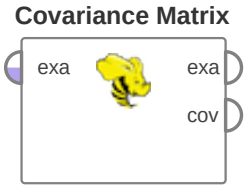
Parameters

normalize weights Indicates if the attributes weights should be normalized.

squared correlation Indicates if the squared correlation should be calculated.

improved algorithm Indicates if the improved calculating method should be used.

Covariance Matrix



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 Ports

example set *(exa)*

Output Ports

example set *(exa)*

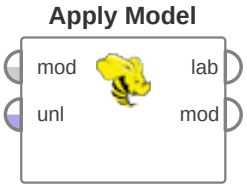
covariance *(cov)*

Parameters

improved algorithm Indicates if the improved calculating method should be used.

5 Scoring

Apply Model



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* panel)
- *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 Ports

model *(mod)*

unlabelled data *(unl)*

Output Ports

labelled data *(lab)*

model *(mod)*

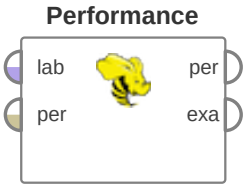
Parameters

application parameters List of model type specific parameters for the application (usually not needed).

use general applier 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.

6 Validation

Performance (Binominal Classification)



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 Ports

labelled data *(lab)*

performance *(per)*

Output Ports

performance *(per)*

example set *(exa)*

Parameters

main criterion The criterion used for comparing performance vectors.

accuracy Relative number of correctly classified examples

classification error Relative number of misclassified examples

kappa The kappa statistics for the classification

precision Relative number of correctly as positive classified examples among all examples classified as positive

recall Relative number of correctly as positive classified examples among all positive examples

lift The lift of the positive class

fallout Relative number of incorrectly as positive classified examples among all negative examples

f measure Combination of precision and recall: f=2pr/(p+r)

false positive Absolute number of incorrectly as positive classified examples

false negative Absolute number of incorrectly as negative classified examples

true positive Absolute number of correctly as positive classified examples

true negative Absolute number of correctly as negative classified examples

sensitivity Relative number of correctly as positive classified examples among all positive examples (like recall)

specificity Relative number of correctly as negative classified examples among all negative examples

youden The sum of sensitivity and specificity minus 1

positive predictive value Relative number of correctly as positive classified examples among all examples classified as positive (same as precision)

negative predictive value Relative number of correctly as negative classified examples among all examples classified as negative

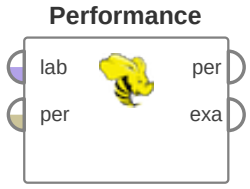
psep The sum of the positive predicitve value and the negative predictive value minus 1

skip undefined labels If set to true, examples with undefined labels are skipped.

use example weights Indicated if example weights should be used for performance calculations if possible.

positive class Specify the positive nominal value for the label attribute (case sensitive). Only for operators below version 2.1.001.

Performance (Classification)



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 Ports

labelled data *(lab)*

performance *(per)*

Output Ports

performance *(per)*

example set *(exa)*

Parameters

main criterion The criterion used for comparing performance vectors.

accuracy Relative number of correctly classified examples

classification error Relative number of misclassified examples

kappa The kappa statistics for the classification

absolute error Average absolute deviation of the prediction from the actual value

relative error Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)

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)

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)

root mean squared error Averaged root-mean-squared error

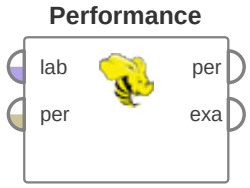
squared error Averaged squared error

skip undefined labels If set to true, examples with undefined labels are skipped.

use example weights Indicated if example weights should be used for performance calculations if possible.

class weights The weights for all classes (first column: class name, second column: weight), empty: using 1 for all classes.

Performance (Regression)



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 numerical 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 Ports

labelled data *(lab)*

performance *(per)*

Output Ports

performance *(per)*

example set *(exa)*

Parameters

main criterion The criterion used for comparing performance vectors.

root mean squared error Averaged root-mean-squared error

absolute error Average absolute deviation of the prediction from the actual value

relative error Average relative error (average of absolute deviation of the prediction from the actual value divided by actual value)

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)

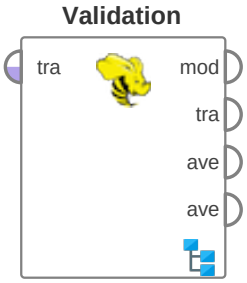
relative error strict Average strict relative error (average of absolute deviation of the prediction from the actual value divided by minimum of the actual value and the prediction)

squared error Averaged squared error

skip undefined labels If set to true, examples with undefined labels are skipped.

use example weights Indicated if example weights should be used for performance calculations if possible.

Split Validation



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 Ports

training *(tra)*

Output Ports

model *(mod)*

training *(tra)*

averagable 1 *(ave)*

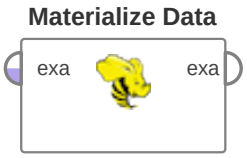
averagable 2 *(ave)*

Parameters

split ratio Relative size of the training set

7 Utility

Materialize Data



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 Ports

example set input *(exa)*

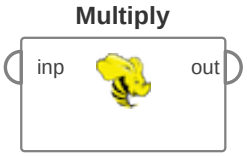
Output Ports

example set output *(exa)*

Parameters

force Force materialization and ignore cost-based optimization.

Multiply



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 Ports

input *(inp)*

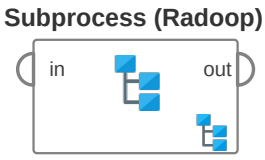
Output Ports

output 1 *(out)*

Parameters

do not materialize 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.

Subprocess (Radoop)



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 Ports

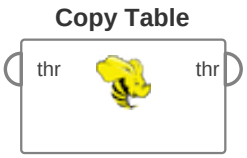
in 1 ***(in)***

Output Ports

out 1 ***(out)***

7.1 Hive

Copy Hive Table



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 Ports

through 1 ***(thr)***

Output Ports

through 1 ***(thr)***

Parameters

use default database for old table Use the database specified in the connection of the Radoop Nest.

database for old table Name of the database being used.

old table Table to copy.

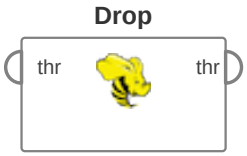
use default database for new table Use the database specified in the connection of the Radoop Nest.

database for new table Name of the database being used.

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.

Drop Hive Table



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 Ports

through 1 *(thr)*

Output Ports

through 1 *(thr)*

Parameters

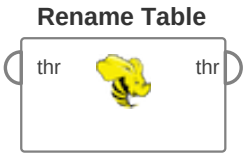
use default database for table Use the database specified in the connection of the Radoop Nest.

database for table Name of the database being used.

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.

Rename Hive Table



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 Ports

through 1 *(thr)*

Output Ports

through 1 *(thr)*

Parameters

use default database for old table Use the database specified in the connection of the Radoop Nest.

database for old table Name of the database being used.

old table Table to rename.

use default database for new table Use the database specified in the connection of the Radoop Nest.

database for new table Name of the database being used.

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.

7.2 Scripting

Hive Script

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 view, 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 Ports

example set input 1 ***(exa)***

Output Ports

example set output ***(exa)***

Parameters

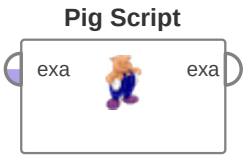
hivescript The Hive script to execute.

auto validate Validate the script automatically using the remote Hive connection. This is required for appropriate meta data generation during design-time.

user defined functions Add User-Defined Functions (UDFs) that can be used in the script. The 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 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.

Pig Script



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 panel), if the software 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.

Please note that the schema of all output relations must be known to the Pig engine. This means that, for example, if you use a STREAM command, you must explicitly specify the schema in the script.

The operator integrates Pig 0.11.2 release.

Input Ports

example set 1 *(exa)*

Output Ports

example set 1 *(exa)*

Parameters

pigscript The Pig script to execute.

preserve binominal mappings 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.

Spark Script

Executes an arbitrary Spark script written in R or Python.

Description

This operator executes the script specified as parameter. The arguments of the script correspond to the input ports, where Hadoop example sets are converted to Spark DataFrames. Analogously, the values returned by the script are delivered at the output ports of the operator, where Spark DataFrames are converted to Hadoop example sets.

Requirements

The Spark Script operator has the following dependencies on the cluster:

- Spark assembly version 1.5.0 or later. You can set the Spark version for your connection on the Advanced Connection Panel.
- *For executing Spark script in Python:* Python 2.6+ or Python 3.4+ installed on the cluster nodes. For using Spark MLlib or spark.ml classes: numpy package installed on the cluster nodes.
- *For executing Spark script in R:* R 3.1+ installed on the cluster nodes.

Meta data delivery

The Spark Script operator does not provide special meta data handling. It gives the meta data of the Nth input port on the Nth output port. If there are more output ports than input ports, the last input port's meta data is delivered on the additional output ports.

YARN log collection

RapidMiner Radoop is able to collect and process the aggregated YARN logs to give a decent error message about a failed Spark script execution. This log is shown in the Log View (View ->Show View ->Log). Please note that for this service the YARN log aggregation has to be enabled on the cluster. You can set the timeout for the log collection in the Preferences/Radoop menu. To turn off the feature, set the timeout to 0.

Input Ports

input *(inp)* The Script operator can have an arbitrary number of HadoopExampleSet inputs. Radoop automatically converts them to Spark DataFrame objects by materializing them in Parquet format. They can be used in the rm_main function as input arguments. If you use multiple input ports, modify the rm_main function to accept more arguments, respectively. Please note that for Parquet materialization you need to have Hive 0.13 or later on the cluster.

Output Ports

result *(res)* The Script operator can have an arbitrary number of HadoopExampleSet outputs. The script must return a DataFrame for every connected output port. The execution will fail if you return less DataFrames than the number of the connected output ports. You can return multiple outputs by returning a list or tuple in Python (e.g. return [result1, result2]), or a vector in R (e.g. return(c(result1, result2));).

Parameters

language *(selection)* The language for Spark scripting. The possible values are R or Python.

- **Python** Spark Python (pyspark) API Docs: http://spark.apache.org/docs/latest/api/python
- **R** Spark R (sparkr) API Docs: http://spark.apache.org/docs/latest/api/R/index.html

R/Python script *(text)* The Spark script to execute. A method (function) with the name

rm_main with one input argument and one return value is defined in the default script. Please do not change the name of the function. It can have as many arguments as the number of the connected input ports and as many returned values as the number of the connected output ports. When using R, please return the output DataFrames as a list (see the default script or the examples).

Other functions and imports can be defined in the script. If you want to import and use third-party sources, please add them as additional sources.

In Spark 1.5 it is possible to reference DataFrame columns by name. This is the preferred way when using the operator over the column indexing as the attribute order of the underlying Hive table is not guaranteed and may be different from the attribute order in the meta data. If you need to use indexing (e.g. because the DataFrame is converted into an RDD), please insert a Reorder Attributes operator before the Spark Script to guarantee the attribute order.

additional local sources *(enumeration)* You can specify additional local sources by adding entries to this parameter. The supported file extensions are .zip for compressed formats and .py/.R for source files. RapidMiner Radoop will automatically upload the specified local sources to the HDFS every time you execute the process. If you want to avoid this behaviour please upload the sources to the HDFS and specify them as *additional HDFS sources* . In Python, the additional sources can be imported with the

from ... import ... syntax. In R, source files can be referenced with

source(...) and the packages with

library(...) . The packages installed on the cluster nodes can also be imported and used in the script (e.g. the numpy package).

additional hdfs sources *(enumeration)* If you want to avoid uploading large source files or packages to the HDFS you can specify their absolute HDFS path in this parameter. Please use this parameter instead of the *additional local sources* if you use RapidMiner Server to execute the process.

preserve binominal mapping *(boolean)* Keep the mappings of the binominal attributes in the first input HadoopExampleSet. If set to false, they will be converted to nominals. If true, you should not introduce new values to the binominal attribute other than the positive value, the negative value and missings.

Tutorial Processes

Running SparkSQL using the Python and R Spark API

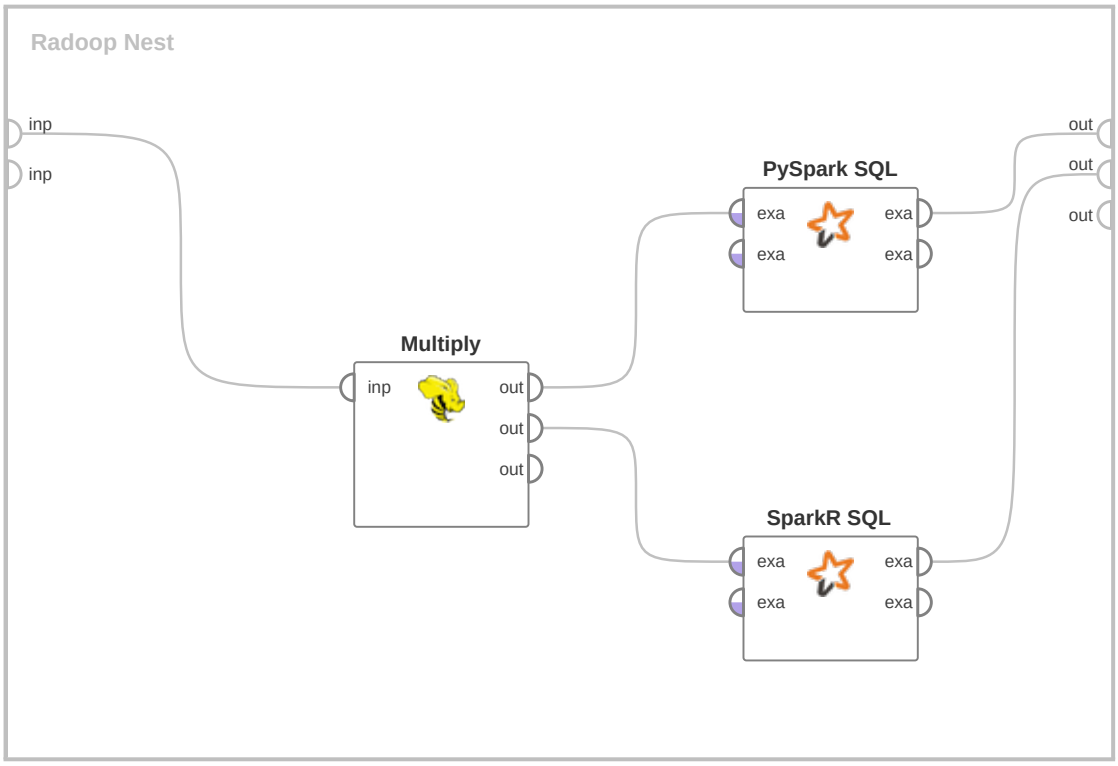


Figure 7.1: Tutorial process ‘Running SparkSQL using the Python and R Spark API’.

This tutorial process uses the Spark Script operator to execute SparkSQL commands. First we generate our example data with the ‘Generate Sales Data’ operator and pass this data to the ’Radoop Nest’. In the Radoop Nest we use the ‘Multiply’ operator, which channels the data into two separate Spark Script operators. These two operators do the same: The first (PySpark SQL) uses the Spark Python API, while the second (SparkR SQL) uses the Spark R API to execute a SparkSQL query on the input data. The output of both Spark Script operators appear as the output of the RapidMiner process.

Please note that to execute the tutorial process, you need to set a properly configured Connection parameter for the Radoop Nest.

Running K-Means using Spark ML in Python

This tutorial process uses the Spark Script operator to execute a K-Means clustering. First we generate our example data with the ‘Generate Data’ operator, remove the ‘label’ attribute and pass this data to the ‘Radoop Nest’. In the Radoop Nest we use the Spark Script operator to conduct a K-Means clustering. The parameters for the K-Means algorithm are set in the Python code of the Spark Script operator. The output of the operator will contain the result of the clustering in the ‘cluster_index’ attribute.

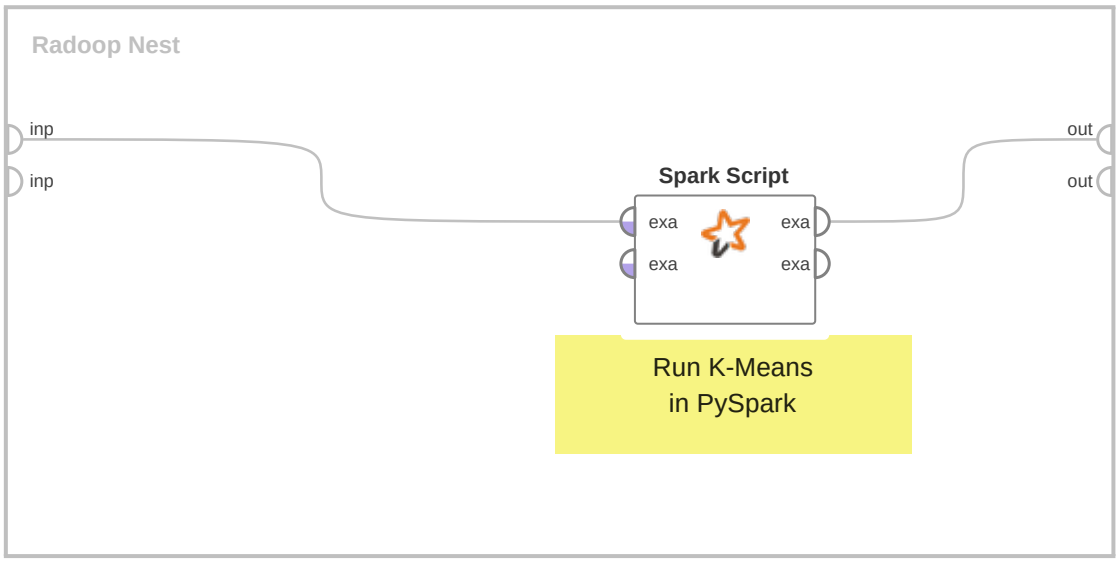


Figure 7.2: Tutorial process ‘Running K-Means using Spark ML in Python’.

Logistic Regression in Python and R

This tutorial process uses the Spark Script operator to train a Logistic Regression model and apply it on a test dataset. First we generate our example data with the ‘Generate Team Profit Data’ operator, convert the label values to binary values, drop all non-numeric attributes and pass this data to the ‘Radoop Nest’. In the Radoop Nest we generate a training and a test dataset by splitting the input data into two parts. We remove the label from the test data and connect these datasets as the inputs of the Python and R Spark Script operators. The output of these operators will hold the results of the classification in the attribute called ‘prediction’.

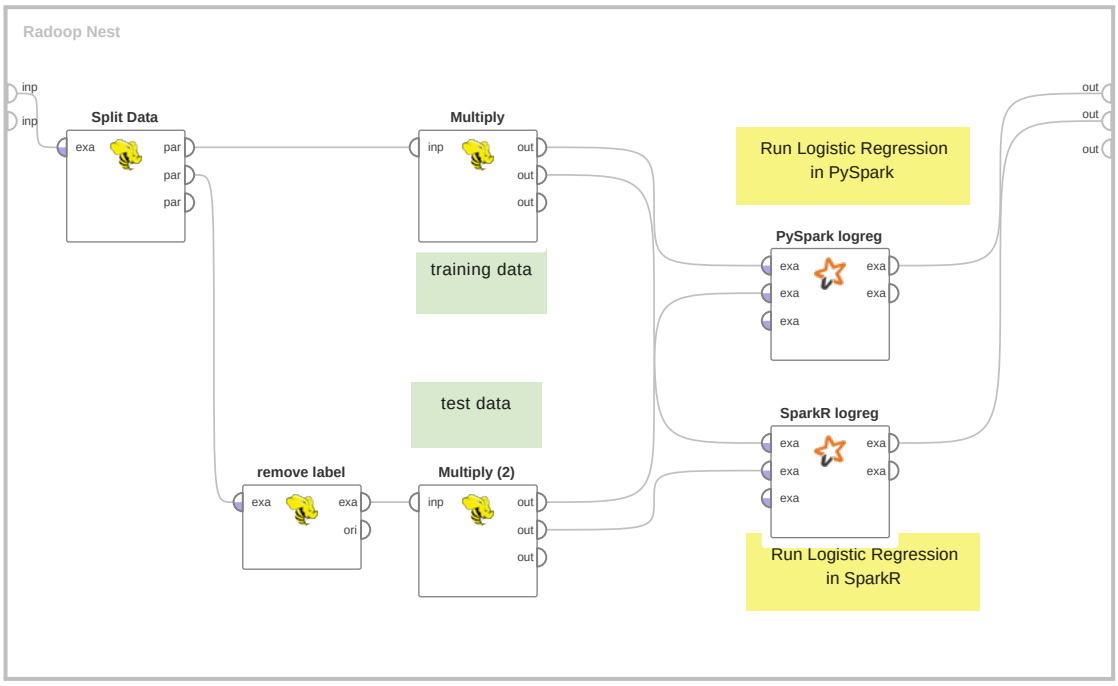
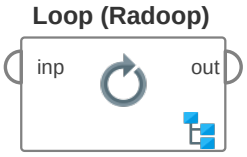


Figure 7.3: Tutorial process 'Logistic Regression in Python and R'.

7.3 Process Control

Loop (Radoop)



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 Ports

input 1 ***(inp)***

Output Ports

output 1 ***(out)***

Parameters

set iteration macro Selects if in each iteration a macro with the current iteration number is set.

macro name The name of the iteration macro.

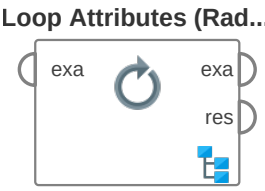
macro start value The number which is set for the macro in the first iteration.

iterations Number of iterations

limit time If checked, the loop will be aborted at last after a specified time.

timeout Timeout in minutes

Loop Attributes (Radoop)



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 Ports

example set *(exa)*

Output Ports

example set *(exa)*

result 1 *(res)*

Parameters

attribute filter type The condition specifies which attributes are selected or affected by this operator.

attribute The attribute which should be chosen.

attributes The attribute which should be chosen.

regular expression A regular expression for the names of the attributes which should be kept.

use except expression 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.

except regular expression A regular expression for the names of the attributes which should be filtered out although matching the above regular expression.

value type The value type of the attributes.

use value type exception 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.

except value type Except this value type.

block type The block type of the attributes.

use block type exception If enabled, an exception to the specified block type might be specified.

except block type Except this block type.

numeric condition Parameter string for the condition, e.g. '>= 5'

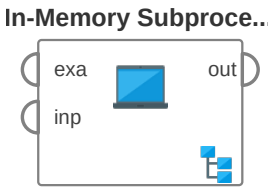
invert selection Indicates if only attributes should be accepted which would normally filtered.

include special attributes Indicate if this operator should also be applied on the special attributes. Otherwise they are always kept.

iteration macro The name of the macro which holds the name of the current feature in each iteration.

7.4 Local In-Memory Computation

In-Memory Subprocess (Full)



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 Ports

example set input *(exa)*

input 1 *(inp)*

Output Ports

output 1 ***(out)***

Parameters

set iteration macro Selects if in each iteration a macro with the current iteration number is set.

macro name The name of the iteration macro.

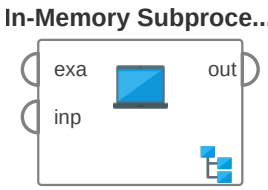
macro start value The number which is set for the macro in the first iteration.

partitioning method Select a method for partitioning the data set to chunks that fit into the memory.

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.

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.

In-Memory Subprocess (Sample)



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 Ports

example set input ***(exa)***

input 1 ***(inp)***

Output Ports

output 1 ***(out)***

Parameters

sample Determines how the amount of data is specified.

balance data If you need to sample differently for examples of a certain class, you might check this.

sample size The estimated number of examples which should be sampled. A sample probabilty for each example is calculated based on this value.

sample probability The sample probability for each example.

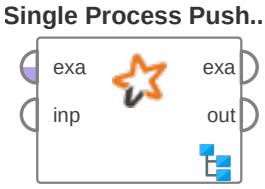
sample size per class The estimated sample size per class.

sample probability per class The fraction per class.

case sensitive Indicates whether the specified class names should be considered case sensitive or not.

7.5 Process Pushdown

Single Process Pushdown



Pushes the process to Hadoop and executes it on a single node, using the node's memory and computation resources.

Description

The subprocess in this meta operator can contain almost any operator from RapidMiner. Furthermore, external extensions (e.g. Weka, Text Processing) can be used as well. A few operators, such as operators related to database handling are not supported (see warnings in the Problems panel).

The subprocess is executed on a single cluster node using the node's available memory. Please note that this operator uses Spark, and a Spark job needs significantly more memory to execute the same process than what a standalone RapidMiner execution would need. Memory consumption overhead varies from process to process; a general rule of thumb is to dedicate 4x more memory to the Spark job than what you would expect to be needed in RapidMiner Studio. If the *Spark Resource Allocation Policy* in your connection is set to *Static, Heuristic Configuration* , the operator automatically calculates the memory for the Spark job based on your cluster setup. If *Static, Manual Configuration* is set, the *driver memory (MB)* and the *executor max memory %* parameters are ignored and the Advanced Spark Parameters from the *Radoop Connection* are used. Please note that the *Static, Default Configuration* and the *Dynamic Resource Allocation* settings are not recommended to be used with this operator in real-life use cases.

The first input port can handle an arbitrarily large example set. The other input ports can handle any *IOObject* (model, performance vector, example set, etc.). Please note that if you connect an example set to any other than the first input port, the data flowing through these ports will be temporarily stored in RapidMiner Studio's memory, so you should only use this for relatively small example sets. The same stands for the output ports: the first output port can handle an arbitrarily large example set, whereas the others work for other kinds of *IOObjects* , or example sets that fit in RapidMiner Studio's memory.

Disclaimer: When the Single Process Pushdown operator is used, all extensions that are installed on the executing RapidMiner Studio instance will be shipped to Hadoop.

Input Ports

example set input *(exa)* This port can have an arbitrarily large *HadoopExampleSet* input. The input will be materialized in Parquet format.

input *(inp)* This operator can have an arbitrary number of *IOObject* inputs. The inputs are serialized and sent to the cluster. Since example sets for this input are collected in memory, it is recommended that you only use this input for relatively small example sets.

Output Ports

example set output *(exa)* This port delivers an arbitrarily large *HadoopExampleSet* output. The output will be materialized in text format.

output *(out)* The operator can have an arbitrary number of additional *IOObject* outputs. The outputs are serialized and sent back to RapidMiner Studio. If an example set is delivered here, it is collected in memory, therefore it is recommended that you only use this output relatively small example sets.

Parameters

sample data *(boolean)* Use a sample of the input data. The sampling can be absolute of probabilistic.

sample *(selection)* Determines how the sampling is conducted:

- **absolute** Absolute sampling. The size of the desired sample needs to be provided.
- **probability** Probabilistic sampling. The sample probability needs to be provided.

balance data *(boolean)* Check this if you need to sample differently for examples of a certain class.

sample probability per class This parameter specifies the probability of examples per class. This parameter is only available when the *sample* parameter is set to 'probability' and the *balance data* parameter is set to true.

sample size per class This parameter specifies the absolute sample size per class. This parameter is only available when the *sample* parameter is set to 'absolute' and the *balance data* parameter is set to true.

balance data *(boolean)* Indicates whether the specified class names should be considered case sensitive or not. This parameter is only available when the *balance data* parameter is set to true.

sample size *(integer)* This parameter specifies the exact number of examples to be included in the sample. This parameter is only available when the *sample* parameter is set to 'absolute' and the *balance data* parameter is not set to true.

sample probability *(real)* This parameter specifies the sample probability for each example. This parameter is only available when the *sample* parameter is set to 'probability' and the *balance data* parameter is not set to true.

driver memory (MB) *(integer)* Amount of memory to be used by the driver process (in MB). This parameter is only considered if *Spark Resource Allocation Policy* is set to *Static, Heuristic Configuration* in the current Radoop Connection.

executor max memory *(integer)* Percentage of the memory on the largest node of the cluster that can be used by the operator. This parameter is only considered if *Spark Resource Allocation Policy* is set to *Static, Heuristic Configuration* in the current Radoop Connection.

use memory monitor *(boolean)* Enables the continuous monitoring of the pushdown process that may terminate the job if it seems that it will run out of memory.

configuration parameters List of configuration parameters that will be set in the pushed down process, just as if they were set in the Preferences menu.

Tutorial Processes

Calculating total transaction values for each store

Figure 7.4: Tutorial process ‘Calculating total transaction values for each store’.

In this simple tutorial process the data is generated using the Generate Sales Data operator. In the Radoop Nest there is a Single Process Pushdown operator that receives the input example set on the first input port and calculates the total sales for each store using the RapidMiner core Generate Attributes, Aggregate and Rename operators. The output is connected to the first (example set) output.

Please note that in order to execute this tutorial process, a properly configured Connection parameter for the Radoop Nest needs to be selected.

Build a Neural Network on the cluster node and apply it with Hive

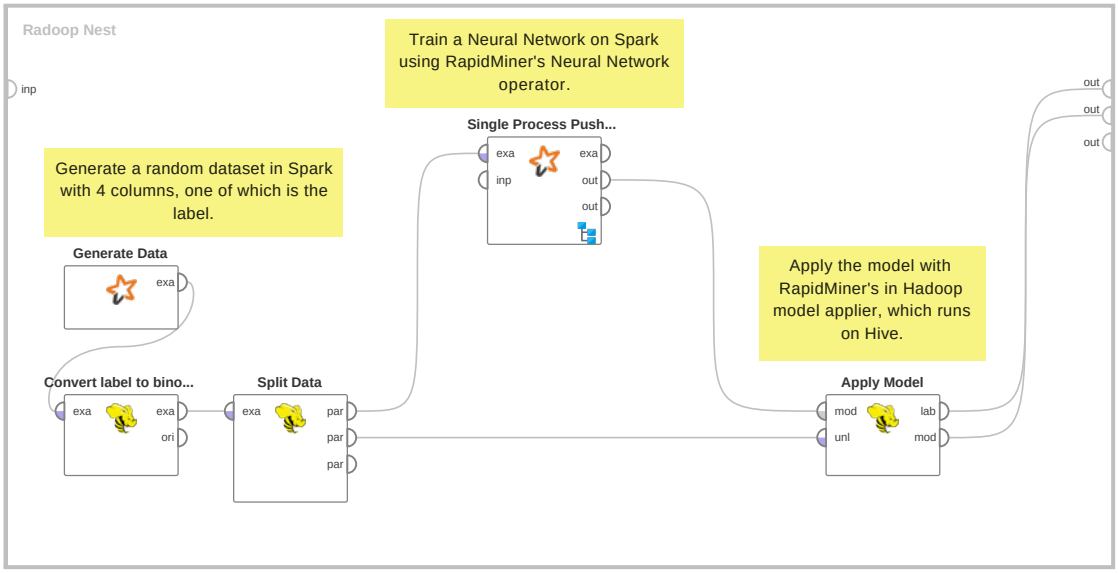


Figure 7.5: Tutorial process ‘Build a Neural Network on the cluster node and apply it with Hive’.

This tutorial process uses the Single Process Pushdown operator to train a Neural Net model on the cluster node using the core Neural Net operator. The input data is generated using Radoop's Generate Data operator. The target function is "binomclasswithnoise" that generates -1.0 and 1.0 (real) label values. It is converted to binominal using the Type Conversion operator, then Split Data is used to create the training and testing Example sets. The first input of the Single Process Pushdown is used for pushing the example set to the cluster. The subprocess contains the core Neural Net operator and connects the model to the second output port, which is applied in-Hadoop on the testing example set using Radoop's Apply Model operator. At the end, RapidMiner's Performance (Binominal Classification) operator is used to evaluate the model's performance.

Please note that in order to execute this tutorial process, a properly configured Connection parameter for the Radoop Nest needs to be selected.

Basket Association Rules

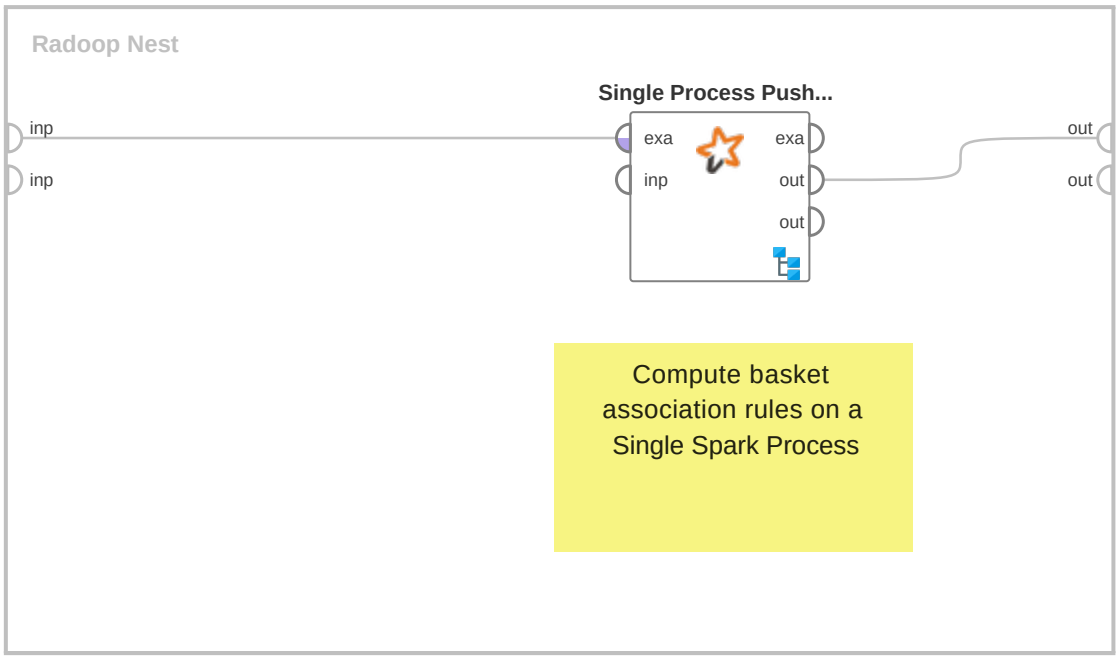


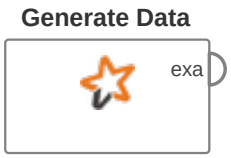
Figure 7.6: Tutorial process 'Basket Association Rules'.

This tutorial process uses the Single Process Pushdown operator to find a frequent pattern association rule in a basket data. For the sake of simplicity, a randomized basket data is generated in memory. The dataset consists of records of grocery purchases. Healthy groceries are likely to be bought together. The basket data is pushed down to Hadoop through a Radoop Nest. On Hadoop a Single Process Pushdown operator is executed, in which an FP-Growth modeling is conducted. Finally, the results are taken back to RapidMiner Studio.

Please note that in order to execute this tutorial process, a properly configured Connection parameter for the Radoop Nest needs to be selected.

7.6 Random Data Generation

Generate Data



This operator generates a numerical example set with a numerical label constructed from the attribute values.

Description

Generate data creates a numerical example set where the values are generated with uniform distribution. The example set consists of the given amount of examples and attributes. The label value is calculated from the attribute values by a predefined function. The distribution is then transformated: the values are multiplied by the given range and an offset value is added to them.

The label target functions are calculated as follows (assuming *n* generated attributes):

- total: att1 + att2 + ... + att[*n*]
- average: (att1 + att2 + ... + att[*n*]) / *n*
- polynomial: att1^3 + att2^2 + att3
- nonlinear: att1 * att2 * att3 + att1 * att2 + att2 * att2
- complicated: att1 * att1 * att2 + att2 * att3 - *e* ^att3
- complicated2: att1 * att1 * att1 + att2 * att2 + att1 * att2 + att1 / *abs(* att3 *)* - 1 / (att3 * att3)
- sinus: *sin(* att1 *)*
- sinus2: *sin(* att1 * att2 *)* + *sin(* att1 + att2 *)*
- superposition: 5 * *sin(* att1 *)* + *sin(* 30 * att1 *)*
- sinusfreq: 10 * *sin(* 3 * att1 *)* + 12 * *sin(* 7 * att1 *)* + 11 * *sin(* 5 * att2 *)* + 9 * *sin(* 10 * att2 *)* + 10 * *sin(* 8 * att1 + att2 *)*
- sinuswithtrend: *sin(* att1 *)* + 0.1 * att1
- binomclass: *signum(* (att1 - *offset*) / *range* - 0.5 *)*
- binomclasswithnoise: *signum(* (a1 + a2 + ... + a[*n*] + *noise*) / (*n* + 1) - 0.5 *)* , where a[*i*] = (att[*i*] - *offset*) / *range* and *noise* is a randomly generated number between 0 and 1.

Output Ports

example set output *(exa)*

Parameters

target function Specifies the target function of this example set.

number examples The number of generated examples.

number of attributes The number of attributes.

attributes lower bound The minimum value for the attributes.

attributes upper bound The maximum value for the attributes.