Package 'sparklyr'

May 26, 2017

Type Package

Title R Interface to Apache Spark

Version 0.5.5

Maintainer Javier Luraschi <javier@rstudio.com>

Description R interface to Apache Spark, a fast and general engine for big data processing, see <http://spark.apache.org>. This package supports connecting to local and remote Apache Spark clusters, provides a 'dplyr' compatible back-end, and provides an interface to Spark's built-in machine learning algorithms.

License Apache License 2.0 | file LICENSE

URL http://spark.rstudio.com

BugReports https://github.com/rstudio/sparklyr/issues

LazyData TRUE

RoxygenNote 6.0.0

Depends R (>= 3.1.2)

Imports methods, lazyeval (>= 0.2.0), dplyr (>= 0.5.0), DBI (>= 0.6), readr (>= 0.2.0), digest, config, rappdirs, assertthat, rprojroot, withr, httr, jsonlite, base64enc, rstudioapi, shiny (>= 1.0.1)

Suggests testthat, RCurl, janeaustenr

NeedsCompilation no

Author Javier Luraschi [aut, cre], Kevin Ushey [aut], JJ Allaire [aut], RStudio [cph], The Apache Software Foundation [aut, cph]

Repository CRAN

Date/Publication 2017-05-26 06:19:21 UTC

R topics documented:

`compile_package_jars` *Compile Scala sources into a Java Archive (jar)*

Description

Compile the `scala` source files contained within an R package into a Java Archive (`jar`) file that can be loaded and used within a Spark environment.

Usage

```
compile_package_jars(..., spec = NULL)
```

Arguments

...	Optional compilation specifications, as generated by spark_compilation_spec. When no arguments are passed, spark_default_compilation_spec is used instead.
spec	An optional list of compilation specifications. When set, this option takes precedence over arguments passed to ....

---

connection_config *Read configuration values for a connection*

Description

Read configuration values for a connection

Usage

```
connection_config(sc, prefix, not_prefix = list())
```

Arguments

sc	spark_connection
prefix	Prefix to read parameters for (e.g. spark.context., spark.sql., etc.)
not_prefix	Prefix to not include.

Value

Named list of config parameters (note that if a prefix was specified then the names will not include the prefix)

---

copy_to.spark_connection *Copy an R Data Frame to Spark*

Description

Copy an R data.frame to Spark, and return a reference to the generated Spark DataFrame as a tbl_spark. The returned object will act as a dplyr-compatible interface to the underlying Spark table.

Usage

```
## S3 method for class 'spark_connection'
copy_to(dest, df, name = deparse(substitute(df)),
  overwrite = FALSE, memory = TRUE, repartition = 0L, ...)
```

Arguments

dest	A spark_connection.
df	An R data.frame.
name	The name to assign to the copied table in Spark.
overwrite	Boolean; overwrite a pre-existing table with the name name if one already exists?
memory	Boolean; should the table be cached into memory?
repartition	The number of partitions to use when distributing the table across the Spark cluster. The default (0) can be used to avoid partitioning.
...	Optional arguments; currently unused.

Value

A tbl_spark, representing a dplyr-compatible interface to a Spark DataFrame.

---

ensure *Enforce Specific Structure for R Objects*

Description

These routines are useful when preparing to pass objects to a Spark routine, as it is often necessary to ensure certain parameters are scalar integers, or scalar doubles, and so on.

Usage

```
ensure_scalar_integer(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)

ensure_scalar_double(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)

ensure_scalar_boolean(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)

ensure_scalar_character(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)
```

Arguments

object	An R object.
allow.na	Are NA values permitted for this object?
allow.null	Are NULL values permitted for this object?
default	If object is NULL, what value should be used in its place? If default is specified, allow.null is ignored (and assumed to be TRUE).

---

find_scalac *Discover the Scala Compiler*

Description

Find the scalac compiler for a particular version of scala, by scanning some common directories containing scala installations.

Usage

```
find_scalac(version, locations = NULL)
```

Arguments

version	The scala version to search for. Versions of the form major.minor will be matched against the scalac installation with version major.minor.patch; if multiple compilers are discovered the most recent one will be used.
locations	Additional locations to scan. By default, the directories /opt/scala and /usr/local/scala will be scanned.

---

ft_binarizer *Feature Transformation – Binarizer*

Description

Apply thresholding to a column, such that values less than or equal to the threshold are assigned the value 0.0, and values greater than the threshold are assigned the value 1.0.

Usage

```
ft_binarizer(x, input.col = NULL, output.col = NULL, threshold = 0.5, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
threshold	The numeric threshold.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

---

ft_bucketizer *Feature Transformation – Bucketizer*

Description

Similar to R's cut function, this transforms a numeric column into a discretized column, with breaks specified through the splits parameter.

Usage

```
ft_bucketizer(x, input.col = NULL, output.col = NULL, splits, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
splits	A numeric vector of cutpoints, indicating the bucket boundaries.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

---

`ft_discrete_cosine_transform`

Feature Transformation – Discrete Cosine Transform (DCT)

Description

Transform a column in the time domain into another column in the frequency domain.

Usage

```
ft_discrete_cosine_transform(x, input.col = NULL, output.col = NULL,
  inverse = FALSE, ...)
```

Arguments

`x`	An object (usually a `spark_tbl`) coercable to a Spark DataFrame.
`input.col`	The name of the input column(s).
`output.col`	The name of the output column.
`inverse`	Perform inverse DCT?
`...`	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: `ft_binarizer`, `ft_bucketizer`, `ft_elementwise_product`, `ft_index_to_string`, `ft_one_hot_encoder`, `ft_quantile_discretizer`, `ft_regex_tokenizer`, `ft_sql_transformer`, `ft_string_indexer`, `ft_tokenizer`, `ft_vector_assembler`, `sdf_mutate`

`ft_elementwise_product`

Feature Transformation – ElementwiseProduct

Description

Computes the element-wise product between two columns. Generally, this is intended as a scaling transformation, where an input vector is scaled by another vector, but this should apply for all element-wise product transformations.

Usage

```
ft_elementwise_product(x, input.col = NULL, output.col = NULL, scaling.col,
  ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
scaling.col	The column used to scale input.col.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

---

ft_index_to_string *Feature Transformation – IndexToString*

Description

Symmetrically to ft_string_indexer, ft_index_to_string maps a column of label indices back to a column containing the original labels as strings.

Usage

```
ft_index_to_string(x, input.col = NULL, output.col = NULL, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

ft_one_hot_encoder *Feature Transformation – OneHotEncoder*

Description

One-hot encoding maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms which expect continuous features, such as Logistic Regression, to use categorical features.

Usage

```
ft_one_hot_encoder(x, input.col = NULL, output.col = NULL, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

ft_quantile_discretizer *Feature Transformation – QuantileDiscretizer*

Description

Takes a column with continuous features and outputs a column with binned categorical features. The bin ranges are chosen by taking a sample of the data and dividing it into roughly equal parts. The lower and upper bin bounds will be -Infinity and +Infinity, covering all real values. This attempts to find numBuckets partitions based on a sample of the given input data, but it may find fewer depending on the data sample values.

Usage

```
ft_quantile_discretizer(x, input.col = NULL, output.col = NULL,
  n.buckets = 5L, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
n.buckets	The number of buckets to use.
...	Optional arguments; currently unused.

Details

Note that the result may be different every time you run it, since the sample strategy behind it is non-deterministic.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

---

ft_regex_tokenizer *Feature Tranformation – RegexTokenizer*

Description

A regex based tokenizer that extracts tokens either by using the provided regex pattern to split the text (default) or repeatedly matching the regex (if gaps is false). Optional parameters also allow filtering tokens using a minimal length. It returns an array of strings that can be empty.

Usage

```
ft_regex_tokenizer(x, input.col = NULL, output.col = NULL, pattern, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
pattern	The regular expression pattern to be used.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

---

ft_sql_transformer *Feature Transformation – SQLTransformer*

Description

Transform a data set using SQL. Use the __THIS__ placeholder as a proxy for the active table.

Usage

```
ft_sql_transformer(x, input.col = NULL, output.col = NULL, sql, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
sql	A SQL statement.
...	Optional arguments; currently unused.

Details

Although this function accepts the input.col and output.col arguments, they are ignored – this interface is done purely for compatibility with sdf_mutate.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_string_indexer, ft_tokenizer, ft_vector_assembler, sdf_mutate

ft_string_indexer *Feature Transformation – StringIndexer*

Description

Encode a column of labels into a column of label indices. The indices are in [0, numLabels), ordered by label frequencies, with the most frequent label assigned index 0. The transformation can be reversed with ft_index_to_string.

Usage

```
ft_string_indexer(x, input.col = NULL, output.col = NULL, params = NULL,
  ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
params	An (optional) R environment – when available, the index <-> label mapping generated by the string indexer will be injected into this environment under the labels key.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_tokenizer, ft_vector_assembler, sdf_mutate

ft_tokenizer *Feature Tranformation – Tokenizer*

Description

A tokenizer that converts the input string to lowercase and then splits it by white spaces.

Usage

```
ft_tokenizer(x, input.col = NULL, output.col = NULL, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_vector_assembler, sdf_mutate

---

ft_vector_assembler *Feature Transformation – VectorAssembler*

Description

Combine multiple vectors into a single row-vector; that is, where each row element of the newly generated column is a vector formed by concatenating each row element from the specified input columns.

Usage

```
ft_vector_assembler(x, input.col = NULL, output.col = NULL, ...)
```

Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

See Also

See http://spark.apache.org/docs/latest/ml-features.html for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: ft_binarizer, ft_bucketizer, ft_discrete_cosine_transform, ft_elementwise_product, ft_index_to_string, ft_one_hot_encoder, ft_quantile_discretizer, ft_regex_tokenizer, ft_sql_transformer, ft_string_indexer, ft_tokenizer, sdf_mutate

invoke *Invoke a Method on a JVM Object*

Description

Invoke methods on Java object references. These functions provide a mechanism for invoking various Java object methods directly from R.

Usage

```
invoke(jobj, method, ...)

invoke_static(sc, class, method, ...)

invoke_new(sc, class, ...)
```

Arguments

jobj	An R object acting as a Java object reference (typically, a spark_jobj).
method	The name of the method to be invoked.
...	Optional arguments, currently unused.
sc	A spark_connection.
class	The name of the Java class whose methods should be invoked.

Details

Use each of these functions in the following scenarios:

invoke	Execute a method on a Java object reference (typically, a spark_jobj).
invoke_static	Execute a static method associated with a Java class.
invoke_new	Invoke a constructor associated with a Java class.

Examples

```
sc <- spark_connect(master = "spark://HOST:PORT")
spark_context(sc) %>%
  invoke("textFile", "file.csv", 1L) %>%
    invoke("count")
```

livy_config *Create a Spark Configuration for Livy*

Description

Create a Spark Configuration for Livy

Usage

```
livy_config(config = spark_config(), username, password)
```

Arguments

config	Optional base configuration
username	The username to use in the Authorization header
password	The password to use in the Authorization header

Details

Extends a Spark "spark_config" configuration with settings for Livy. For instance, "username" and "password" define the basic authentication settings for a Livy session.

Value

Named list with configuration data

---

livy_service_start *Start Livy*

Description

Starts the livy service.

Stops the running instances of the livy service.

Usage

```
livy_service_start(version = NULL, spark_version = NULL)

livy_service_stop()
```

Arguments

version	The version of 'livy' to use.
spark_version	The version of 'spark' to connect to.

`ml_als_factorization` *Spark ML – Alternating Least Squares (ALS) matrix factorization.*

Description

Perform alternating least squares matrix factorization on a Spark DataFrame.

Usage

```
ml_als_factorization(x, rating.column = "rating", user.column = "user",
  item.column = "item", rank = 10L, regularization.parameter = 0.1,
  iter.max = 10L, ml.options = ml_options(), ...)
```

Arguments

`x`	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
`rating.column`	The name of the column containing ratings.
`user.column`	The name of the column containing user IDs.
`item.column`	The name of the column containing item IDs.
`rank`	Rank of the factorization.
`regularization.parameter`	The regularization parameter.
`iter.max`	The maximum number of iterations to use.
`ml.options`	Optional arguments, used to affect the model generated. See `ml_options` for more details.
`...`	Optional arguments. The `data` argument can be used to specify the data to be used when `x` is a formula; this allows calls of the form `ml_linear_regression(y ~ x, data = tbl)`, and is especially useful in conjunction with `do`.

See Also

Other Spark ML routines: `ml_decision_tree`, `ml_generalized_linear_regression`, `ml_gradient_boosted_trees`, `ml_kmeans`, `ml_lda`, `ml_linear_regression`, `ml_logistic_regression`, `ml_multilayer_perceptron`, `ml_naive_bayes`, `ml_one_vs_rest`, `ml_pca`, `ml_random_forest`, `ml_survival_regression`

ml_binary_classification_eval

Spark ML - Binary Classification Evaluator

Description

See the Spark ML Documentation BinaryClassificationEvaluator

Usage

```
ml_binary_classification_eval(predicted_tbl_spark, label, score,
  metric = "areaUnderROC")
```

Arguments

predicted_tbl_spark	The result of running sdf_predict
label	Name of column string specifying which column contains the true, indexed labels (ie 0 / 1)
score	Name of column contains the scored probability of a success (ie 1)
metric	The classification metric - one of: areaUnderRoc (default) or areaUnderPR

Value

area under the specified curve

ml_classification_eval

Spark ML - Classification Evaluator

Description

See the Spark ML Documentation MulticlassClassificationEvaluator

Usage

```
ml_classification_eval(predicted_tbl_spark, label, predicted_lbl,
  metric = "f1")
```

Arguments

predicted_tbl_spark	A tbl_spark object that contains a columns with predicted labels
label	Name of the column that contains the true, indexed label. Support for binary and multi-class labels, column should be of double type (use as.double)
predicted_lbl	Name of the column that contains the predicted label NOT the scored probability. Support for binary and multi-class labels, column should be of double type (use as.double)
metric	A classification metric, one of: f1 (default), precision, recall, weightedPrecision, weightedRecall, accuracy

Value

see metric

---

ml_create_dummy_variables *Create Dummy Variables*

Description

Given a column in a Spark DataFrame, generate a new Spark DataFrame containing dummy variable columns.

Usage

```
ml_create_dummy_variables(x, input, reference = NULL, levels = NULL,
  labels = NULL, envir = new.env(parent = emptyenv()))
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
input	The name of the input column.
reference	The reference label. This variable is omitted when generating dummy variables (to avoid perfect multi-collinearity if all dummy variables were to be used in the model fit); to generate dummy variables for all columns this can be explicitly set as NULL.
levels	The set of levels for which dummy variables should be generated. By default, constructs one variable for each unique value occurring in the column specified by input.
labels	An optional R list, mapping values in the input column to column names to be assigned to the associated dummy variable.
envir	An optional R environment; when provided, it will be filled with useful auxiliary information. See **Auxiliary Information** for more information.

Details

The dummy variables are generated in a similar mechanism to `model.matrix`, where categorical variables are expanded into a set of binary (dummy) variables. These dummy variables can be used for regression of categorical variables within the various regression routines provided by `sparklyr`.

Auxiliary Information

The `envir` argument can be used as a mechanism for returning optional information. Currently, the following pieces are returned:

`levels`:	The set of unique values discovered within the input column.
`columns`:	The column names generated.

If the `envir` argument is supplied, the names of any dummy variables generated will be included, under the `labels` key.

`ml_decision_tree` *Spark ML – Decision Trees*

Description

Perform regression or classification using decision trees.

Usage

```
ml_decision_tree(x, response, features, max.bins = 32L, max.depth = 5L,
  type = c("auto", "regression", "classification"),
  ml.options = ml_options(), ...)
```

Arguments

`x`	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
`response`	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When `response` is a formula, it is used in preference to other parameters to set the `response`, `features`, and `intercept` parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. `response ~ feature1 + feature2 + ...`. The intercept term can be omitted by using `- 1` in the model fit.
`features`	The name of features (terms) to use for the model fit.
`max.bins`	The 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.
`max.depth`	Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.

type	The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_generalized_linear_regression, ml_gradient_boosted_trees
ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron,
ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_generalized_linear_regression

Spark ML – Generalized Linear Regression

Description

Perform generalized linear regression on a Spark DataFrame.

Usage

```
ml_generalized_linear_regression(x, response, features, intercept = TRUE,
  family = gaussian(link = "identity"), iter.max = 100L,
  ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
family	The family / link function to use; analogous to those normally passed in to calls to R's own glm.
iter.max	The maximum number of iterations to use.

ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

Details

In contrast to ml_linear_regression() and ml_logistic_regression(), these routines do not allow you to tweak the loss function (e.g. for elastic net regression); however, the model fits returned by this routine are generally richer in regards to information provided for assessing the quality of fit.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_gradient_boosted_trees

Spark ML – Gradient-Boosted Tree

Description

Perform regression or classification using gradient-boosted trees.

Usage

```
ml_gradient_boosted_trees(x, response, features, max.bins = 32L,
  max.depth = 5L, type = c("auto", "regression", "classification"),
  ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
max.bins	The 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.

max.depth	Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.
type	The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_kmeans *Spark ML – K-Means Clustering*

Description

Perform k-means clustering on a Spark DataFrame.

Usage

```
ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x),
  compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
centers	The number of cluster centers to compute.
iter.max	The maximum number of iterations to use.
features	The name of features (terms) to use for the model fit.
compute.cost	Whether to compute cost for k-means model using Spark's computeCost.
tolerance	Param for the convergence tolerance for iterative algorithms.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

Value

ml_model object of class kmeans with overloaded print, fitted and predict functions.

References

Bahmani et al., Scalable K-Means++, VLDB 2012

See Also

For information on how Spark k-means clustering is implemented, please see http://spark.apache.org/docs/latest/mllib-clustering.html#k-means.

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptro ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_lda *Spark ML – Latent Dirichlet Allocation*

Description

Fit a Latent Dirichlet Allocation (LDA) model to a Spark DataFrame.

Usage

```
ml_lda(x, features = dplyr::tbl_vars(x), k = length(features),
  alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
features	The name of features (terms) to use for the model fit.
k	The number of topics to estimate.
alpha	Concentration parameter for the prior placed on documents' distributions over topics. This is a singleton which is replicated to a vector of length k in fitting (as currently EM optimizer only supports symmetric distributions, so all values in the vector should be the same). For Expectation-Maximization optimizer values should be > 1.0. By default alpha = (50 / k) + 1, where 50/k is common in LDA libraries and +1 follows from Asuncion et al. (2009), who recommend a +1 adjustment for EM.
beta	Concentration parameter for the prior placed on topics' distributions over terms. For Expectation-Maximization optimizer value should be > 1.0 and by default beta = 0.1 + 1, where 0.1 gives a small amount of smoothing and +1 follows Asuncion et al. (2009), who recommend a +1 adjustment for EM.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.

...	Optional arguments. The `data` argument can be used to specify the data to be used when `x` is a formula; this allows calls of the form `ml_linear_regression(y ~ x, data = tbl)`, and is especially useful in conjunction with `do`.

Note

The topics' distributions over terms are called "beta" in the original LDA paper by Blei et al., but are called "phi" in many later papers such as Asuncion et al., 2009.

For terminology used in LDA model see Spark LDA documentation.

References

Original LDA paper (journal version): Blei, Ng, and Jordan. "Latent Dirichlet Allocation." JMLR, 2003.

Asuncion et al. (2009)

See Also

Other Spark ML routines: `ml_als_factorization`, `ml_decision_tree`, `ml_generalized_linear_regression`, `ml_gradient_boosted_trees`, `ml_kmeans`, `ml_linear_regression`, `ml_logistic_regression`, `ml_multilayer_perceptron`, `ml_naive_bayes`, `ml_one_vs_rest`, `ml_pca`, `ml_random_forest`, `ml_survival_regression`

---

ml_linear_regression *Spark ML – Linear Regression*

Description

Perform linear regression on a Spark DataFrame.

Usage

```
ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0,
  lambda = 0, iter.max = 100L, ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When `response` is a formula, it is used in preference to other parameters to set the `response`, `features`, and `intercept` parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. `response ~ feature1 + feature2 + ...`. The intercept term can be omitted by using `- 1` in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?

alpha, lambda	Parameters controlling loss function penalization (for e.g. lasso, elastic net, and ridge regression). See **Details** for more information.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

Details

Spark implements for both $L1$ and $L2$ regularization in linear regression models. See the preamble in the Spark Classification and Regression documentation for more details on how the loss function is parameterized.

In particular, with alpha set to 1, the parameterization is equivalent to a lasso model; if alpha is set to 0, the parameterization is equivalent to a ridge regression model.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_logistic_regression

Spark ML – Logistic Regression

Description

Perform logistic regression on a Spark DataFrame.

Usage

```
ml_logistic_regression(x, response, features, intercept = TRUE, alpha = 0,
  lambda = 0, iter.max = 100L, ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.

intercept	Boolean; should the model be fit with an intercept term?
alpha, lambda	Parameters controlling loss function penalization (for e.g. lasso, elastic net, and ridge regression). See **Details** for more information.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

Details

Spark implements for both $L1$ and $L2$ regularization in linear regression models. See the preamble in the Spark Classification and Regression documentation for more details on how the loss function is parameterized.

In particular, with alpha set to 1, the parameterization is equivalent to a lasso model; if alpha is set to 0, the parameterization is equivalent to a ridge regression model.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

---

ml_model *Create an ML Model Object*

Description

Create an ML model object, wrapping the result of a Spark ML routine call. The generated object will be an R list with S3 classes c("ml_model_<class>", "ml_model").

Usage

```
ml_model(class, model, ..., .call = sys.call(sys.parent()))
```

Arguments

class	The name of the machine learning routine used in the encompassing model. Note that the model name generated will be generated as ml_model_<class>; that is, ml_model will be prefixed.
model	The underlying Spark model object.
...	Additional model information; typically supplied as named values.
.call	The R call used in generating this model object (ie, the top-level R routine that wraps over the associated Spark ML routine). Typically used for print output in e.g. print and summary methods.

`ml_multilayer_perceptron` *Spark ML – Multilayer Perceptron*

Description

Creates and trains multilayer perceptron on a Spark DataFrame.

Usage

```
ml_multilayer_perceptron(x, response, features, layers, iter.max = 100,
  seed = sample(.Machine$integer.max, 1), ml.options = ml_options(), ...)
```

Arguments

`x`	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
`response`	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When `response` is a formula, it is used in preference to other parameters to set the `response`, `features`, and `intercept` parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. `response ~ feature1 + feature2 + ...`. The intercept term can be omitted by using `- 1` in the model fit.
`features`	The name of features (terms) to use for the model fit.
`layers`	A numeric vector describing the layers – each element in the vector gives the size of a layer. For example, `c(4, 5, 2)` would imply three layers, with an input (feature) layer of size 4, an intermediate layer of size 5, and an output (class) layer of size 2.
`iter.max`	The maximum number of iterations to use.
`seed`	A random seed. Set this value if you need your results to be reproducible across repeated calls.
`ml.options`	Optional arguments, used to affect the model generated. See `ml_options` for more details.
`...`	Optional arguments. The `data` argument can be used to specify the data to be used when `x` is a formula; this allows calls of the form `ml_linear_regression(y ~ x, data = tbl)`, and is especially useful in conjunction with `do`.

See Also

Other Spark ML routines: `ml_als_factorization`, `ml_decision_tree`, `ml_generalized_linear_regression`, `ml_gradient_boosted_trees`, `ml_kmeans`, `ml_lda`, `ml_linear_regression`, `ml_logistic_regression`, `ml_naive_bayes`, `ml_one_vs_rest`, `ml_pca`, `ml_random_forest`, `ml_survival_regression`

ml_naive_bayes *Spark ML – Naive-Bayes*

Description

Perform regression or classification using naive bayes.

Usage

```
ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options(),
  ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
lambda	The (Laplace) smoothing parameter. Defaults to zero.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_one_vs_rest, ml_pca, ml_random_forest, ml_survival_regression

ml_one_vs_rest *Spark ML – One vs Rest*

Description

Perform regression or classification using one vs rest.

Usage

```
ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options(),
  ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
classifier	The classifier model. These model objects can be obtained through the use of the only.model parameter supplied with ml_options.
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_pca, ml_random_forest, ml_survival_regression

---

ml_options *Options for Spark ML Routines*

Description

Provide this object to the various Spark ML methods, to control certain facets of the model outputs produced.

Usage

```
ml_options(id.column = random_string("id"),
  response.column = random_string("response"),
  features.column = random_string("features"), model.transform = NULL,
  only.model = FALSE, na.action = getOption("na.action", "na.omit"), ...)
```

Arguments

id.column	The name to assign to the generated id column.
response.column	The name to assign to the generated response column.
features.column	The name to assign to the generated features column.
model.transform	An optional R function that accepts a Spark model and returns a Spark model. This can be used to supply optional Spark model fitting parameters not made available in the sparklyr APIs.
only.model	Boolean; should the Spark model object itself be returned without fitting the actual model? Useful for ml_one_vs_rest.
na.action	An R function, or the name of an R function, indicating how missing values should be handled.
...	Optional arguments, reserved for future expansion.

---

ml_pca *Spark ML – Principal Components Analysis*

Description

Perform principal components analysis on a Spark DataFrame.

Usage

```
ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
features	The columns to use in the principal components analysis. Defaults to all columns in x.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_random_forest, ml_survival_regression

`ml_prepare_dataframe` *Prepare a Spark DataFrame for Spark ML Routines*

Description

This routine prepares a Spark DataFrame for use by Spark ML routines.

Usage

```
ml_prepare_dataframe(x, features, response = NULL, ...,
  ml.options = ml_options(), envir = new.env(parent = emptyenv()))
```

Arguments

`x`	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
`features`	The name of features (terms) to use for the model fit.
`response`	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When `response` is a formula, it is used in preference to other parameters to set the `response`, `features`, and `intercept` parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. `response ~ feature1 + feature2 + ...`. The intercept term can be omitted by using `- 1` in the model fit.
`...`	Optional arguments. The `data` argument can be used to specify the data to be used when `x` is a formula; this allows calls of the form `ml_linear_regression(y ~ x, data = tbl)`, and is especially useful in conjunction with `do`.
`ml.options`	Optional arguments, used to affect the model generated. See `ml_options` for more details.
`envir`	An R environment – when supplied, it will be filled with metadata describing the transformations that have taken place.

Details

Spark DataFrames are prepared through the following transformations:

1. All specified columns are transformed into a numeric data type (using a simple cast for integer / logical columns, and `ft_string_indexer` for strings),
2. The `ft_vector_assembler` is used to combine the specified features into a single 'feature' vector, suitable for use with Spark ML routines.

After calling this function, the `envir` environment (when supplied) will be populated with a set of variables:

`features`:	The name of the generated `features` vector.
`response`:	The name of the generated `response` vector.
`labels`:	When the `response` column is a string vector, the `ft_string_indexer` is used to transform the vector into a [0:n

Examples

```
## Not run:
# example of how 'ml_prepare_dataframe' might be used to invoke
# Spark's LinearRegression routine from the 'ml' package
envir <- new.env(parent = emptyenv())
tdf <- ml_prepare_dataframe(df, features, response, envir = envir)

lr <- invoke_new(
  sc,
  "org.apache.spark.ml.regression.LinearRegression"
)

# use generated 'features', 'response' vector names in model fit
model <- lr %>%
  invoke("setFeaturesCol", envir$features) %>%
  invoke("setLabelCol", envir$response)

## End(Not run)
```

---

ml_prepare_response_features_intercept

Pre-process the Inputs to a Spark ML Routine

Description

Pre-process / normalize the inputs typically passed to a Spark ML routine.

Usage

```
ml_prepare_response_features_intercept(x = NULL, response, features,
  intercept, envir = parent.frame(),
  categorical.transformations = new.env(parent = emptyenv()),
  ml.options = ml_options())

ml_prepare_features(x, features, envir = parent.frame(),
  ml.options = ml_options())
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When `response` is a formula, it is used in preference to other parameters to set the `response`, `features`, and `intercept` parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. `response ~ feature1 + feature2 + ...`. The intercept term can be omitted by using `- 1` in the model fit.
features	The name of features (terms) to use for the model fit.

intercept	Boolean; should the model be fit with an intercept term?
envir	The R environment in which the response, features and intercept bindings should be mutated. (Typically, the parent frame).
categorical.transformations	An R environment used to record what categorical variables were binarized in this procedure. Categorical variables that included in the model formula will be transformed into binary variables, and the generated mappings will be stored in this environment.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.

Details

Pre-processing of these inputs typically involves:

1. Handling the case where response is itself a formula describing the model to be fit, thereby extracting the names of the response and features to be used,
2. Splitting categorical features into dummy variables (so they can easily be accommodated + specified in the underlying Spark ML model fit),
3. Mutating the associated variables *in the specified environment*.

Please take heed of the last point, as while this is useful in practice, the behavior will be very surprising if you are not expecting it.

Examples

```
## Not run:
# note that ml_prepare_features, by default, mutates the 'features'
# binding in the same environment in which the function was called
local({
   ml_prepare_features(features = ~ x1 + x2 + x3)
   print(features) # c("x1", "x2", "x3")
})

## End(Not run)
```

---

ml_random_forest *Spark ML – Random Forests*

Description

Perform regression or classification using random forests with a Spark DataFrame.

Usage

```
ml_random_forest(x, response, features, max.bins = 32L, max.depth = 5L,
  num.trees = 20L, type = c("auto", "regression", "classification"),
  ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ...The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
max.bins	The 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.
max.depth	Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.
num.trees	Number of trees to train (>= 1).
type	The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_survival_regression

---

ml_saveload *Save / Load a Spark ML Model Fit*

Description

Save / load a ml_model fit.

Usage

```
ml_load(sc, file)

ml_save(model, file)
```

Arguments

sc	A spark_connection.
file	The filepath used for model save / load. Currently, only local filepaths are supported.
model	A ml_model fit.

Details

These functions are currently experimental and not yet ready for production use. Unfortunately, the training summary information for regression fits (linear, logistic, generalized) are currently not serialized as part of the model fit, and so model fits recovered through ml_load will not work with e.g. fitted, residuals, and so on. Such fits should still be suitable for generating predictions with new data, however.

---

ml_survival_regression

Spark ML – Survival Regression

Description

Perform survival regression on a Spark DataFrame, using an Accelerated failure time (AFT) model with potentially right-censored data.

Usage

```
ml_survival_regression(x, response, features, intercept = TRUE,
  censor = "censor", iter.max = 100L, ml.options = ml_options(), ...)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + ...The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
censor	The name of the vector that provides censoring information. This should be a numeric vector, with 0 marking uncensored data, and 1 marking right-censored data.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See ml_options for more details.

...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form ml_linear_regression(y ~ x, data = tbl), and is especially useful in conjunction with do.

See Also

Other Spark ML routines: ml_als_factorization, ml_decision_tree, ml_generalized_linear_regression, ml_gradient_boosted_trees, ml_kmeans, ml_lda, ml_linear_regression, ml_logistic_regression, ml_multilayer_perceptron, ml_naive_bayes, ml_one_vs_rest, ml_pca, ml_random_forest

---

ml_tree_feature_importance *Spark ML - Feature Importance for Tree Models*

Description

Spark ML - Feature Importance for Tree Models

Usage

```
ml_tree_feature_importance(sc, model)
```

Arguments

sc	A spark_connection.
model	An ml_model encapsulating the output from a decision tree.

Value

A sorted data frame with feature labels and their relative importance.

---

na.replace *Replace Missing Values in Objects*

Description

This S3 generic provides an interface for replacing NA values within an object.

Usage

```
na.replace(object, ...)
```

Arguments

object	An R object.
...	Arguments passed along to implementing methods.

register_extension *Register a Package that Implements a Spark Extension*

Description

Registering an extension package will result in the package being automatically scanned for spark dependencies when a connection to Spark is created.

Usage

```
register_extension(package)

registered_extensions()
```

Arguments

package	The package(s) to register.

Note

Packages should typically register their extensions in their `.onLoad` hook – this ensures that their extensions are registered when their namespaces are loaded.

sdf-saveload *Save / Load a Spark DataFrame*

Description

Routines for saving and loading Spark DataFrames.

Usage

```
sdf_save_table(x, name, overwrite = FALSE, append = FALSE)

sdf_load_table(sc, name)

sdf_save_parquet(x, path, overwrite = FALSE, append = FALSE)

sdf_load_parquet(sc, path)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
name	The table name to assign to the saved Spark DataFrame.
overwrite	Boolean; overwrite a pre-existing table of the same name?
append	Boolean; append to a pre-existing table of the same name?
sc	A spark_connection object.
path	The path where the Spark DataFrame should be saved.

---

sdf_copy_to *Copy an Object into Spark*

Description

Copy an object into Spark, and return an R object wrapping the copied object (typically, a Spark DataFrame).

Usage

```
sdf_copy_to(sc, x, name, memory, repartition, overwrite, ...)

sdf_import(x, sc, name, memory, repartition, overwrite, ...)
```

Arguments

sc	The associated Spark connection.
x	An R object from which a Spark DataFrame can be generated.
name	The name to assign to the copied table in Spark.
memory	Boolean; should the table be cached into memory?
repartition	The number of partitions to use when distributing the table across the Spark cluster. The default (0) can be used to avoid partitioning.
overwrite	Boolean; overwrite a pre-existing table with the name name if one already exists?
...	Optional arguments, passed to implementing methods.

Advanced Usage

sdf_copy_to is an S3 generic that, by default, dispatches to sdf_import. Package authors that would like to implement sdf_copy_to for a custom object type can accomplish this by implementing the associated method on sdf_import.

See Also

Other Spark data frames: sdf_partition, sdf_predict, sdf_register, sdf_sample, sdf_sort

Examples

```
sc <- spark_connect(master = "spark://HOST:PORT")
sdf_copy_to(sc, iris)
```

`sdf_mutate` *Mutate a Spark DataFrame*

Description

Use Spark's feature transformers to mutate a Spark DataFrame.

Usage

```
sdf_mutate(.data, ...)

sdf_mutate_(.data, ..., .dots)
```

Arguments

`.data`	A `spark_tbl`.
`...`	Named arguments, mapping new column names to the transformation to be applied.
`.dots`	A named list, mapping output names to transformations.

Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

See Also

Other feature transformation routines: `ft_binarizer`, `ft_bucketizer`, `ft_discrete_cosine_transform`, `ft_elementwise_product`, `ft_index_to_string`, `ft_one_hot_encoder`, `ft_quantile_discretizer`, `ft_regex_tokenizer`, `ft_sql_transformer`, `ft_string_indexer`, `ft_tokenizer`, `ft_vector_assembler`

Examples

```
## Not run:
# using the 'beaver1' dataset, binarize the 'temp' column
data(beavers, package = "datasets")
beaver_tbl <- copy_to(sc, beaver1, "beaver")
beaver_tbl %>%
  mutate(squared = temp ^ 2) %>%
  sdf_mutate(warm = ft_binarizer(squared, 1000)) %>%
  sdf_register("mutated")

# view our newly constructed tbl
head(beaver_tbl)

# note that we have two separate tbls registered
dplyr::src_tbls(sc)

## End(Not run)
```

`sdf_partition` *Partition a Spark Dataframe*

Description

Partition a Spark DataFrame into multiple groups. This routine is useful for splitting a DataFrame into, for example, training and test datasets.

Usage

```
sdf_partition(x, ..., weights = NULL, seed = sample(.Machine$integer.max,
  1))
```

Arguments

`x`	An object coercable to a Spark DataFrame.
`...`	Named parameters, mapping table names to weights. The weights will be normalized such that they sum to 1.
`weights`	An alternate mechanism for supplying weights – when specified, this takes precedence over the `...` arguments.
`seed`	Random seed to use for randomly partitioning the dataset. Set this if you want your partitioning to be reproducible on repeated runs.

Details

The sampling weights define the probability that a particular observation will be assigned to a particular partition, not the resulting size of the partition. This implies that partitioning a DataFrame with, for example,

```
sdf_partition(x, training = 0.5, test = 0.5)
```

is not guaranteed to produce `training` and `test` partitions of equal size.

Value

An R `list` of `tbl_sparks`.

Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

See Also

Other Spark data frames: `sdf_copy_to`, `sdf_predict`, `sdf_register`, `sdf_sample`, `sdf_sort`

Examples

```
## Not run:
# randomly partition data into a 'training' and 'test'
# dataset, with 60% of the observations assigned to the
# 'training' dataset, and 40% assigned to the 'test' dataset
data(diamonds, package = "ggplot2")
diamonds_tbl <- copy_to(sc, diamonds, "diamonds")
partitions <- diamonds_tbl %>%
  sdf_partition(training = 0.6, test = 0.4)
print(partitions)

# alternate way of specifying weights
weights <- c(training = 0.6, test = 0.4)
diamonds_tbl %>% sdf_partition(weights = weights)

## End(Not run)
```

`sdf_persist` *Persist a Spark DataFrame*

Description

Persist a Spark DataFrame, forcing any pending computations and (optionally) serializing the results to disk.

Usage

```
sdf_persist(x, storage.level = "MEMORY_AND_DISK")
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
storage.level	The storage level to be used. Please view the Spark Documentation for information on what storage levels are accepted.

Details

Spark DataFrames invoke their operations lazily – pending operations are deferred until their results are actually needed. Persisting a Spark DataFrame effectively 'forces' any pending computations, and then persists the generated Spark DataFrame as requested (to memory, to disk, or otherwise).

Users of Spark should be careful to persist the results of any computations which are non-deterministic – otherwise, one might see that the values within a column seem to 'change' as new operations are performed on that data set.

---

sdf_predict *Model Predictions with Spark DataFrames*

Description

Given a ml_model fit alongside a new data set, produce a new Spark DataFrame with predicted values encoded in the "prediction" column.

Usage

```
sdf_predict(object, newdata, ...)
```

Arguments

object, newdata	An object coercable to a Spark DataFrame.
...	Optional arguments; currently unused.

See Also

Other Spark data frames: sdf_copy_to, sdf_partition, sdf_register, sdf_sample, sdf_sort

sdf_quantile *Compute (Approximate) Quantiles with a Spark DataFrame*

Description

Given a numeric column within a Spark DataFrame, compute approximate quantiles (to some relative error).

Usage

```
sdf_quantile(x, column, probabilities = c(0, 0.25, 0.5, 0.75, 1),
  relative.error = 1e-05)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
column	The column for which quantiles should be computed.
probabilities	A numeric vector of probabilities, for which quantiles should be computed.
relative.error	The relative error – lower values imply more precision in the computed quantiles.

sdf_read_column *Read a Column from a Spark DataFrame*

Description

Read a single column from a Spark DataFrame, and return the contents of that column back to R.

Usage

```
sdf_read_column(x, column)
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
column	The name of a column within x.

sdf_register *Register a Spark DataFrame*

Description

Registers a Spark DataFrame (giving it a table name for the Spark SQL context), and returns a `tbl_spark`.

Usage

```
sdf_register(x, name = NULL)
```

Arguments

`x`	A Spark DataFrame.
`name`	A name to assign this table.

Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

See Also

Other Spark data frames: `sdf_copy_to`, `sdf_partition`, `sdf_predict`, `sdf_sample`, `sdf_sort`

sdf_sample *Randomly Sample Rows from a Spark DataFrame*

Description

Draw a random sample of rows (with or without replacement) from a Spark DataFrame.

Usage

```
sdf_sample(x, fraction = 1, replacement = TRUE, seed = NULL)
```

Arguments

`x`	An object coercable to a Spark DataFrame.
`fraction`	The fraction to sample.
`replacement`	Boolean; sample with replacement?
`seed`	An (optional) integer seed.

Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

See Also

Other Spark data frames: `sdf_copy_to`, `sdf_partition`, `sdf_predict`, `sdf_register`, `sdf_sort`

`sdf_schema` *Read the Schema of a Spark DataFrame*

Description

Read the schema of a Spark DataFrame.

Usage

```
sdf_schema(x)
```

Arguments

`x`	An object coercable to a Spark DataFrame (typically, a `tbl_spark`).

Details

The `type` column returned gives the string representation of the underlying Spark type for that column; for example, a vector of numeric values would be returned with the type `"DoubleType"`. Please see the Spark Scala API Documentation for information on what types are available and exposed by Spark.

Value

An R `list`, with each `list` element describing the `name` and `type` of a column.

sdf_sort *Sort a Spark DataFrame*

Description

Sort a Spark DataFrame by one or more columns, with each column sorted in ascending order.

Usage

```
sdf_sort(x, columns)
```

Arguments

x	An object coercable to a Spark DataFrame.
columns	The column(s) to sort by.

Transforming Spark DataFrames

The family of functions prefixed with sdf_ generally access the Scala Spark DataFrame API directly, as opposed to the dplyr interface which uses Spark SQL. These functions will 'force' any pending SQL in a dplyr pipeline, such that the resulting tbl_spark object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly collect() the table.

See Also

Other Spark data frames: sdf_copy_to, sdf_partition, sdf_predict, sdf_register, sdf_sample

sdf_with_unique_id *Add a Unique ID Column to a Spark DataFrame*

Description

Add a unique ID column to a Spark DataFrame. The Spark monotonicallyIncreasingId function is used to produce these and is guaranteed to produce unique, monotonically increasing ids; however, there is no guarantee that these IDs will be sequential. The table is persisted immediately after the column is generated, to ensure that the column is stable – otherwise, it can differ across new computations.

Usage

```
sdf_with_unique_id(x, id = "id")
```

Arguments

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
id	The name of the column to host the generated IDs.

---

spark-api *Access the Spark API*

Description

Access the commonly-used Spark objects associated with a Spark instance. These objects provide access to different facets of the Spark API.

Usage

```
spark_context(sc)

java_context(sc)

hive_context(sc)

spark_session(sc)
```

Arguments

sc	A spark_connection.

Details

The Scala API documentation is useful for discovering what methods are available for each of these objects. Use invoke to call methods on these objects.

Spark Context

The main entry point for Spark functionality. The **Spark Context** represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

Java Spark Context

A Java-friendly version of the aforementioned **Spark Context**.

Hive Context

An instance of the Spark SQL execution engine that integrates with data stored in Hive. Configuration for Hive is read from hive-site.xml on the classpath.

Starting with Spark >= 2.0.0, the **Hive Context** class has been deprecated – it is superceded by the **Spark Session** class, and hive_context will return a **Spark Session** object instead. Note that both classes share a SQL interface, and therefore one can invoke SQL through these objects.

Spark Session

Available since Spark 2.0.0, the **Spark Session** unifies the **Spark Context** and **Hive Context** classes into a single interface. Its use is recommended over the older APIs for code targeting Spark 2.0.0 and above.

`spark-connections` *Manage Spark Connections*

Description

These routines allow you to manage your connections to Spark.

Usage

```
spark_connect(master = "local", spark_home = Sys.getenv("SPARK_HOME"),
  method = c("shell", "livy", "databricks", "test"), app_name = "sparklyr",
  version = NULL, hadoop_version = NULL, config = spark_config(),
  extensions = sparklyr::registered_extensions())

spark_connection_is_open(sc)

spark_disconnect(sc, ...)

spark_disconnect_all()
```

Arguments

`master`	Spark cluster url to connect to. Use `"local"` to connect to a local instance of Spark installed via `spark_install`.
`spark_home`	The path to a Spark installation. Defaults to the path provided by the `SPARK_HOME` environment variable. If `SPARK_HOME` is defined, it will be always be used unless the `version` parameter is specified to force the use of a locally installed version.
`method`	The method used to connect to Spark. Currently, only `"shell"` is supported.
`app_name`	The application name to be used while running in the Spark cluster.
`version`	The version of Spark to use. Only applicable to `"local"` Spark connections.
`hadoop_version`	The version of Hadoop to use. Only applicable to `"local"` Spark connections.
`config`	Custom configuration for the generated Spark connection. See `spark_config` for details.
`extensions`	Extension packages to enable for this connection. By default, all packages enabled through the use of `sparklyr::register_extension` will be passed here.
`sc`	A `spark_connection`.
`...`	Optional arguments; currently unused.

Examples

```
sc <- spark_connect(master = "spark://HOST:PORT")
connection_is_open(sc)

spark_disconnect(sc)
```

`spark_compilation_spec` *Define a Spark Compilation Specification*

Description

For use with `compile_package_jars`. The Spark compilation specification is used when compiling Spark extension Java Archives, and defines which versions of Spark, as well as which versions of Scala, should be used for compilation.

Usage

```
spark_compilation_spec(spark_version = NULL, spark_home = NULL,
  scalac_path = NULL, scala_filter = NULL, jar_name = NULL)
```

Arguments

`spark_version`	The Spark version to build against. This can be left unset if the path to a suitable Spark home is supplied.
`spark_home`	The path to a Spark home installation. This can be left unset if `spark_version` is supplied; in such a case, `sparklyr` will attempt to discover the associated Spark installation using `spark_home_dir`.
`scalac_path`	The path to the `scalac` compiler to be used during compilation of your Spark extension. Note that you should ensure the version of `scalac` selected matches the version of `scalac` used with the version of Spark you are compiling against.
`scala_filter`	An optional R function that can be used to filter which `scala` files are used during compilation. This can be useful if you have auxiliary files that should only be included with certain versions of Spark.
`jar_name`	The name to be assigned to the generated `jar`.

Details

Most Spark extensions won't need to define their own compilation specification, and can instead rely on the default behavior of `compile_package_jars`.

spark_config *Read Spark Configuration*

Description

Read Spark Configuration

Usage

```
spark_config(file = "config.yml", use_default = TRUE)
```

Arguments

file	Name of the configuration file
use_default	TRUE to use the built-in detaults provided in this package

Details

Read Spark configuration using the **config** package.

Value

Named list with configuration data

spark_connection *Retrieve the Spark Connection Associated with an R Object*

Description

Retrieve the spark_connection associated with an R object.

Usage

```
spark_connection(x, ...)
```

Arguments

x	An R object from which a spark_connection can be obtained.
...	Optional arguments; currently unused.

spark_dataframe *Retrieve a Spark DataFrame*

Description

This S3 generic is used to access a Spark DataFrame object (as a Java object reference) from an R object.

Usage

```
spark_dataframe(x, ...)
```

Arguments

x	An R object wrapping, or containing, a Spark DataFrame.
...	Optional arguments; currently unused.

Value

A spark_jobj representing a Java object reference to a Spark DataFrame.

spark_default_compilation_spec *Default Compilation Specification for Spark Extensions*

Description

This is the default compilation specification used for Spark extensions, when used with compile_package_jars.

Usage

```
spark_default_compilation_spec(pkg = infer_active_package_name())
```

Arguments

pkg	The package containing Spark extensions to be compiled.

spark_dependency *Define a Spark dependency*

Description

Define a Spark dependency consisting of a set of custom JARs and Spark packages.

Usage

```
spark_dependency(jars = NULL, packages = NULL)
```

Arguments

jars	Character vector of full paths to JAR files
packages	Character vector of Spark packages names

Value

An object of type ‘spark_dependency‘

spark_install *Download and install various versions of Spark*

Description

Install versions of Spark for use with local Spark connections (i.e. spark_connect(master = "local")

Usage

```
spark_install(version = NULL, hadoop_version = NULL, reset = TRUE,
  logging = "INFO", verbose = interactive())

spark_uninstall(version, hadoop_version)

spark_install_dir()

spark_install_tar(tarfile)

spark_installed_versions()

spark_available_versions()
```

Arguments

version	Version of Spark to install. See spark_available_versions for a list of supported versions
hadoop_version	Version of Hadoop to install. See spark_available_versions for a list of supported versions
reset	Attempts to reset settings to defaults.
logging	Logging level to configure install. Supported options: "WARN", "INFO"
verbose	Report information as Spark is downloaded / installed
tarfile	Path to TAR file conforming to the pattern spark-###-bin-(hadoop)?### where ### reference spark and hadoop versions respectively.

Value

List with information about the installed version.

---

spark_jobj *Retrieve a Spark JVM Object Reference*

Description

This S3 generic is used for accessing the underlying Java Virtual Machine (JVM) Spark objects associated with R objects. These objects act as references to Spark objects living in the JVM. Methods on these objects can be called with the invoke family of functions.

Usage

```
spark_jobj(x, ...)
```

Arguments

x	An R object containing, or wrapping, a spark_jobj.
...	Optional arguments; currently unused.

See Also

invoke, for calling methods on Java object references.

spark_load_table *Load a Spark Table into a Spark DataFrame.*

Description

Load a Spark Table into a Spark DataFrame.

Usage

```
spark_load_table(sc, name, options = list(), repartition = 0,
  memory = TRUE, overwrite = TRUE)
```

Arguments

sc	A spark_connection.
name	The name to assign to the newly generated table.
options	A list of strings with additional options. See http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration.
repartition	The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.
memory	Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)
overwrite	Boolean; overwrite the table with the given name if it already exists?

See Also

Other Spark serialization routines: spark_read_csv, spark_read_json, spark_read_parquet, spark_save_table, spark_write_csv, spark_write_json, spark_write_parquet

spark_log *View Entries in the Spark Log*

Description

View the most recent entries in the Spark log. This can be useful when inspecting output / errors produced by Spark during the invocation of various commands.

Usage

```
spark_log(sc, n = 100, filter = NULL, ...)
```

Arguments

sc	A spark_connection.
n	The max number of log entries to retrieve. Use NULL to retrieve all entries within the log.
filter	Character string to filter log entries.
...	Optional arguments; currently unused.

---

spark_read_csv *Read a CSV file into a Spark DataFrame*

Description

Read a tabular data file into a Spark DataFrame.

Usage

```
spark_read_csv(sc, name, path, header = TRUE, columns = NULL,
  infer_schema = TRUE, delimiter = ",", quote = "\"", escape = "\\",
  charset = "UTF-8", null_value = NULL, options = list(),
  repartition = 0, memory = TRUE, overwrite = TRUE)
```

Arguments

sc	A spark_connection.
name	The name to assign to the newly generated table.
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
header	Boolean; should the first row of data be used as a header? Defaults to TRUE.
columns	A named vector specifying column types.
infer_schema	Boolean; should column types be automatically inferred? Requires one extra pass over the data. Defaults to TRUE.
delimiter	The character used to delimit each column. Defaults to ',' .
quote	The character used as a quote. Defaults to '"'.
escape	The character used to escape other characters. Defaults to '\'.
charset	The character set. Defaults to '"UTF-8"'.
null_value	The character to use for null, or missing, values. Defaults to NULL.
options	A list of strings with additional options.
repartition	The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.
memory	Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)
overwrite	Boolean; overwrite the table with the given name if it already exists?

Details

You can read data from HDFS (`hdfs://`), S3 (`s3n://`), as well as the local file system (`file://`).

If you are reading from a secure S3 bucket be sure that the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables are both defined.

When `header` is `FALSE`, the column names are generated with a `V` prefix; e.g. `V1, V2, ...`.

See Also

Other Spark serialization routines: `spark_load_table`, `spark_read_json`, `spark_read_parquet`, `spark_save_table`, `spark_write_csv`, `spark_write_json`, `spark_write_parquet`

---

spark_read_json *Read a JSON file into a Spark DataFrame*

Description

Read a table serialized in the JavaScript Object Notation format into a Spark DataFrame.

Usage

```
spark_read_json(sc, name, path, options = list(), repartition = 0,
  memory = TRUE, overwrite = TRUE)
```

Arguments

`sc`	A `spark_connection`.
`name`	The name to assign to the newly generated table.
`path`	The path to the file. Needs to be accessible from the cluster. Supports the '`"hdfs://"`', '`"s3n://"`' and '`"file://"`' protocols.
`options`	A list of strings with additional options.
`repartition`	The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.
`memory`	Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)
`overwrite`	Boolean; overwrite the table with the given name if it already exists?

Details

You can read data from HDFS (`hdfs://`), S3 (`s3n://`), as well as the local file system (`file://`).

If you are reading from a secure S3 bucket be sure that the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables are both defined.

See Also

Other Spark serialization routines: `spark_load_table`, `spark_read_csv`, `spark_read_parquet`, `spark_save_table`, `spark_write_csv`, `spark_write_json`, `spark_write_parquet`

spark_read_parquet *Read a Parquet file into a Spark DataFrame*

Description

Read a Parquet file into a Spark DataFrame.

Usage

```
spark_read_parquet(sc, name, path, options = list(), repartition = 0,
  memory = TRUE, overwrite = TRUE)
```

Arguments

sc	A spark_connection.
name	The name to assign to the newly generated table.
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
options	A list of strings with additional options. See http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration.
repartition	The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.
memory	Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)
overwrite	Boolean; overwrite the table with the given name if it already exists?

Details

You can read data from HDFS (hdfs://), S3 (s3n://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure that the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables are both defined.

See Also

Other Spark serialization routines: spark_load_table, spark_read_csv, spark_read_json, spark_save_table, spark_write_csv, spark_write_json, spark_write_parquet

spark_save_table *Saves a Spark DataFrame as a Spark table*

Description

Saves a Spark DataFrame and as a Spark table.

Usage

```
spark_save_table(x, path, mode = NULL)
```

Arguments

x	A Spark DataFrame or dplyr operation
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
mode	Specifies the behavior when data or table already exists.

See Also

Other Spark serialization routines: spark_load_table, spark_read_csv, spark_read_json, spark_read_parquet, spark_write_csv, spark_write_json, spark_write_parquet

spark_version *Get the Spark Version Associated with a Spark Connection*

Description

Retrieve the version of Spark associated with a Spark connection.

Usage

```
spark_version(sc)
```

Arguments

sc	A spark_connection.

Details

Suffixes for e.g. preview versions, or snapshotted versions, are trimmed – if you require the full Spark version, you can retrieve it with invoke(spark_context(sc), "version").

Value

The Spark version as a numeric_version.

spark_version_from_home *Get the Spark Version Associated with a Spark Installation*

Description

Retrieve the version of Spark associated with a Spark installation.

Usage

```
spark_version_from_home(spark_home, default = NULL)
```

Arguments

spark_home	The path to a Spark installation.
default	The default version to be inferred, in case version lookup failed, e.g. no Spark installation was found at spark_home.

spark_web *Open the Spark web interface*

Description

Open the Spark web interface

Usage

```
spark_web(sc, ...)
```

Arguments

sc	A spark_connection.
...	Optional arguments; currently unused.

spark_write_csv *Write a Spark DataFrame to a CSV*

Description

Write a Spark DataFrame to a tabular (typically, comma-separated) file.

Usage

```
spark_write_csv(x, path, header = TRUE, delimiter = ",", quote = "\"",
  escape = "\\", charset = "UTF-8", null_value = NULL,
  options = list())
```

Arguments

x	A Spark DataFrame or dplyr operation
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
header	Should the first row of data be used as a header? Defaults to TRUE.
delimiter	The character used to delimit each column, defaults to ,.
quote	The character used as a quote, defaults to "hdfs://".
escape	The chatacter used to escape other characters, defaults to \.
charset	The character set, defaults to "UTF-8".
null_value	The character to use for default values, defaults to NULL.
options	A list of strings with additional options.

See Also

Other Spark serialization routines: spark_load_table, spark_read_csv, spark_read_json, spark_read_parquet, spark_save_table, spark_write_json, spark_write_parquet

spark_write_json *Write a Spark DataFrame to a JSON file*

Description

Serialize a Spark DataFrame to the JavaScript Object Notation format.

Usage

```
spark_write_json(x, path, mode = NULL, options = list())
```

Arguments

x	A Spark DataFrame or dplyr operation
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.

See Also

Other Spark serialization routines: spark_load_table, spark_read_csv, spark_read_json, spark_read_parquet, spark_save_table, spark_write_csv, spark_write_parquet

---

spark_write_parquet *Write a Spark DataFrame to a Parquet file*

Description

Serialize a Spark DataFrame to the Parquet format.

Usage

```
spark_write_parquet(x, path, mode = NULL, options = list())
```

Arguments

x	A Spark DataFrame or dplyr operation
path	The path to the file. Needs to be accessible from the cluster. Supports the '"hdfs://"', '"s3n://"' and '"file://"' protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options. See http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration.

See Also

Other Spark serialization routines: spark_load_table, spark_read_csv, spark_read_json, spark_read_parquet, spark_save_table, spark_write_csv, spark_write_json

tbl_cache *Cache a Spark Table*

Description

Force a Spark table with name `name` to be loaded into memory. Operations on cached tables should normally (although not always) be more performant than the same operation performed on an uncached table.

Usage

```
tbl_cache(sc, name, force = TRUE)
```

Arguments

`sc`	A `spark_connection`.
`name`	The table name.
`force`	Force the data to be loaded into memory? This is accomplished by calling the `count` API on the associated Spark DataFrame.

tbl_uncache *Uncache a Spark Table*

Description

Force a Spark table with name `name` to be unloaded from memory.

Usage

```
tbl_uncache(sc, name)
```

Arguments

`sc`	A `spark_connection`.
`name`	The table name.

Index