Package ‘sparklyr’

April 20, 2020

Type Package
Title R Interface to Apache Spark
Version 1.2.0
Maintainer Yitao Li <yitao@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
SystemRequirements Spark: 1.6.x or 2.x
URL http://spark.rstudio.com
BugReports https://github.com/sparklyr/sparklyr/issues
LazyData TRUE
RoxygenNote 7.1.0
Depends R (>= 3.2)
Imports assertthat, base64enc, config (>= 0.2), DBI (>= 0.6-1), dplyr (>= 0.7.2), dbplyr (>= 1.1.0), digest, forge, generics, globals, httr (>= 1.2.1), jsonlite (>= 1.4), methods,openssl (>= 0.8), purrr, r2d3, rappdirs, rjson (>= 0.2.20), rlang (>= 0.1.4), rprojroot, rstudioapi (>= 0.10), tibble, tidyr, withr, xml2, ellipsis (>= 0.1.0)
Suggests broom, foreach, ggplot2, iterators, jamaeustenr, Lahman, mlbench, mnet, nycflights13, R6, RCurl, reshape2, shiny (>= 1.0.1), testthat
Collate 'arrow_data.R' 'config_settings.R' 'config_spark.R'
 'connection_instances.R' 'connection_progress.R'
 'connection_shinyapp.R' 'connection_spark.R'
 'connection_viewer.R' 'core_arrow.R' 'core_config.R'
 'core_connection.R' 'core_deserialize.R' 'core_gateway.R'
 'core_invoke.R' 'core_job.R' 'core_serialize.R' 'core_utils.R'
 'core_worker_config.R' 'data_copy.R' 'data_csv.R'
'ml_model_isotonic_regression.R' 'ml_model_kmeans.R'
'ml_model_lda.R' 'ml_model_linear_regression.R'
'ml_model_linear_svc.R' 'ml_model_logistic_regression.R'
'ml_model_naive_bayes.R' 'ml_model_one_vs_rest.R'
'ml_model_random_forest.R' 'ml_model_utils.R'
'ml_param_utils.R' 'ml_persistence.R' 'ml_pipeline.R'
'ml_pipeline_utils.R' 'ml_print_utils.R'
'ml_recommendation_als.R'
'ml_regression_aft_survival_regression.R'
'ml_regression_decision_tree_regressor.R'
'ml_regression_gbt_regressor.R'
'ml_regression_generalized_linear_regression.R'
'ml_regression_isotonic_regression.R'
'ml_regression_linear_regression.R'
'ml_regression_random_forest_regressor.R' 'ml_stat.R'
'ml_summary.R' 'ml_transformation_methods.R'
'ml_transformer_and_estimator.R' 'ml_tuning.R'
'ml_tuning_cross_validator.R'
'ml_tuning_train_validation_split.R' 'ml_utils.R'
'ml_validator_utils.R' 'mutation.R' 'na_actions.R'
'new_model_multilayer_perceptron.R' 'precondition.R'
'project_template.R' 'qubole_connection.R' 'reexports.R'
'sdf_dim.R' 'sdf_interface.R' 'sdf_ml.R' 'sdf_saveload.R'
'sdf_sequence.R' 'sdf_sql.R' 'sdf_stat.R' 'sdf_streaming.R'
'sdf_utils.R' 'sdf_wrapper.R' 'spark_apply.R'
'spark_apply_bundle.R' 'spark.apply_legacy.R' 'spark.compile.R'
'spark.compile_embedded.R' 'spark.connection.R'
'spark.context_config.R' 'spark.dataframe.R'
'spark.extensions.R' 'spark_gateway.R' 'spark_globals.R'
'spark_hive.R' 'spark_home.R' 'spark_invoke.R' 'spark_submit.R'
'spark.utils.R' 'spark_version.R' 'stream_data.R'
'stream_job.R' 'stream_operations.R' 'stream_shiny.R'
'stream_view.R' 'tables_spark.R' 'tbl_spark.R'
'test_connection.R' 'tidiers_ml_aft_survival_regression.R'
'tidiers_ml_als.R' 'tidiers_ml_isotonic_regression.R'
'tidiers_ml_lda.R' 'tidiers_ml_linear_models.R'
'tidiers_ml_logistic_regression.R'
'tidiers_ml_multilayer_perceptron.R' 'tidiers_ml_naive_bayes.R'
'tidiers_ml_svc_models.R' 'tidiers_ml_tree_models.R'
'tidiers_ml_unsupervised_models.R' 'tidiers_pca.R'
'tidiers_utils.R' 'utils.R' 'worker_apply.R' 'worker_connect.R'
'worker_connection.R' 'worker_invoke.R' 'worker_log.R'
'worker_main.R' 'yarn_cluster.R' 'yarn_config.R' 'yarn_ui.R'
'zzz.R'

**NeedsCompilation** no

**Author** Javier Luraschi [aut],
Kevin Kuo [aut] (<https://orcid.org/0000-0001-7803-7901>),
Kevin Ushey [aut],
R topics documented:

R topics documented:

- checkpoint_directory
- compile_package_jars
- connection_config
- copy_to.spark_connection
- download_scalac
- ensure
- find_scalac
- ft_binarizer
- ft_bucketizer
- ft_chisq_selector
- ft_count_vectorizer
- ft_dct
- ft_elementwise_product
- ft_feature_hasher
- ft_hashing_tf
- ft_idf
- ft_imputer
- ft_index_to_string
- ft_interaction
- ft_lsh
- ft_lsh_utils
- ft_max_abs_scaler
- ft_min_max_scaler
- ft_ngram
- ft_normalizer
- ft_one_hot_encoder
- ft_one_hot_encoder_estimator
- ft_pca
- ft_polynomial_expansion
- ft_quantile_discretizer
- ft_regex_tokenizer
- ft_r_formula
- ft_sql_transformer
- ft_standard_scaler
R topics documented:

ft_stop_words_remover .................................................. 52
ft_string_indexer ............................................................ 53
ft_tokenizer ................................................................. 55
ft_vector_assembler .......................................................... 56
ft_vector_indexer ............................................................... 57
ft_vector_slicer ................................................................. 58
ft_word2vec ................................................................. 59
hive_context_config ........................................................... 61
invoke ............................................................................ 62
livy_config ................................................................. 62
livy_service_start ............................................................. 64
ml-params ................................................................. 65
ml-persistence ............................................................... 65
ml-transform-methods ........................................................ 66
ml-tuning ................................................................. 67
ml_aft_survival_regression .................................................. 70
ml_als ................................................................. 73
ml_als_tidiers ............................................................... 76
ml_bisecting_kmeans ........................................................ 76
ml_chisquare_test .............................................................. 78
ml_clustering_evaluator ....................................................... 79
ml_corr ................................................................. 80
ml_decision_tree_classifier .................................................. 81
ml_default_stop_words ....................................................... 85
ml_evaluate ................................................................. 86
ml_evaluator ................................................................. 87
ml_feature_importances ....................................................... 89
ml_fpgrowth ................................................................. 90
ml_gaussian_mixture .......................................................... 91
ml_gbt_classifier ............................................................. 93
ml_generalized_linear_regression ........................................... 97
ml_glm_tidiers ............................................................... 101
ml_isotonic_regression ....................................................... 102
ml_isotonic_regression_tidiers ................................................ 104
ml_kmeans ................................................................. 104
ml_lda ................................................................. 106
ml_lda_tidiers ............................................................... 111
ml_linear_regression .......................................................... 112
ml_linear_svc ............................................................... 114
ml_linear_svc_tidiers ........................................................ 116
ml_logistic_regression ....................................................... 117
ml_logistic_regression_tidiers ................................................ 120
ml_model_data ............................................................... 121
ml_multilayer_perceptron_classifier ..................................... 121
ml_multilayer_perceptron_tidiers .......................................... 125
ml_naive_bayes ............................................................. 125
ml_naive_bayes_tidiers ......................................................... 128
ml_one_vs_rest .............................................................. 128
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml_pca_tidiers ........... 130</td>
</tr>
<tr>
<td>ml_pipeline ............. 130</td>
</tr>
<tr>
<td>ml_random_forest_classifier .... 131</td>
</tr>
<tr>
<td>ml_stage ............... 136</td>
</tr>
<tr>
<td>ml_summary ............. 136</td>
</tr>
<tr>
<td>ml_survival_regression_tidiers .... 137</td>
</tr>
<tr>
<td>ml_tree_tidiers ........ 137</td>
</tr>
<tr>
<td>ml_uid ................. 139</td>
</tr>
<tr>
<td>ml_unsupervised_tidiers .... 139</td>
</tr>
<tr>
<td>na.replace ............. 140</td>
</tr>
<tr>
<td>random_string .......... 140</td>
</tr>
<tr>
<td>reactiveSpark .......... 141</td>
</tr>
<tr>
<td>registerDoSpark ....... 141</td>
</tr>
<tr>
<td>register_extension ...... 142</td>
</tr>
<tr>
<td>sdf-saveload ........... 142</td>
</tr>
<tr>
<td>sdf-transform-methods .... 143</td>
</tr>
<tr>
<td>sdf_along ............. 144</td>
</tr>
<tr>
<td>sdf_bind ............... 144</td>
</tr>
<tr>
<td>sdf_broadcast ......... 145</td>
</tr>
<tr>
<td>sdf_checkpoint ....... 145</td>
</tr>
<tr>
<td>sdf_coalesce ........... 146</td>
</tr>
<tr>
<td>sdf_collect ............ 146</td>
</tr>
<tr>
<td>sdf_copy_to ........... 147</td>
</tr>
<tr>
<td>sdf_crosstab .......... 148</td>
</tr>
<tr>
<td>sdf_debug_string ....... 148</td>
</tr>
<tr>
<td>sdf_describe ........... 149</td>
</tr>
<tr>
<td>sdf_dim ............... 149</td>
</tr>
<tr>
<td>sdf_drop_duplicates .... 150</td>
</tr>
<tr>
<td>sdf_is_streaming ...... 150</td>
</tr>
<tr>
<td>sdf_last_index ........ 150</td>
</tr>
<tr>
<td>sdf_len ............... 151</td>
</tr>
<tr>
<td>sdf_num_partitions ..... 151</td>
</tr>
<tr>
<td>sdfPersist .......... 152</td>
</tr>
<tr>
<td>sdf_pivot ............ 152</td>
</tr>
<tr>
<td>sdf_project ........... 153</td>
</tr>
<tr>
<td>sdf_quantile .......... 154</td>
</tr>
<tr>
<td>sdf_random_split ..... 155</td>
</tr>
<tr>
<td>sdf_read_column ....... 156</td>
</tr>
<tr>
<td>sdf_register .......... 157</td>
</tr>
<tr>
<td>sdf_repartition .. 157</td>
</tr>
<tr>
<td>sdf_residuals.ml_model_generalized_linear_regression .... 158</td>
</tr>
<tr>
<td>sdf_sample ............ 158</td>
</tr>
<tr>
<td>sdf_schema ........... 159</td>
</tr>
<tr>
<td>sdf_separate_column ... 160</td>
</tr>
<tr>
<td>sdf_seq ............... 160</td>
</tr>
<tr>
<td>sdf_sort ............ 161</td>
</tr>
<tr>
<td>sdf_sql ............... 161</td>
</tr>
<tr>
<td>sdf_with_sequential_id ... 162</td>
</tr>
</tbody>
</table>
### Topics Documented

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdf_with_unique_id</td>
<td>162</td>
</tr>
<tr>
<td>spark-api</td>
<td>163</td>
</tr>
<tr>
<td>spark-connections</td>
<td>164</td>
</tr>
<tr>
<td>spark_apply</td>
<td>165</td>
</tr>
<tr>
<td>spark_apply_bundle</td>
<td>167</td>
</tr>
<tr>
<td>spark_apply_log</td>
<td>168</td>
</tr>
<tr>
<td>spark_compilation_spec</td>
<td>168</td>
</tr>
<tr>
<td>spark_config</td>
<td>169</td>
</tr>
<tr>
<td>spark_config_kubernetes</td>
<td>170</td>
</tr>
<tr>
<td>spark_config_packages</td>
<td>171</td>
</tr>
<tr>
<td>spark_config_settings</td>
<td>171</td>
</tr>
<tr>
<td>spark_connection</td>
<td>172</td>
</tr>
<tr>
<td>spark_connection_class</td>
<td>172</td>
</tr>
<tr>
<td>spark_connection_find</td>
<td>172</td>
</tr>
<tr>
<td>spark_context_config</td>
<td>173</td>
</tr>
<tr>
<td>spark_dataframe</td>
<td>173</td>
</tr>
<tr>
<td>spark_default_compilation_spec</td>
<td>174</td>
</tr>
<tr>
<td>spark_dependency</td>
<td>174</td>
</tr>
<tr>
<td>spark_dependency_fallback</td>
<td>175</td>
</tr>
<tr>
<td>spark_extension</td>
<td>175</td>
</tr>
<tr>
<td>spark_home_set</td>
<td>176</td>
</tr>
<tr>
<td>spark_job</td>
<td>176</td>
</tr>
<tr>
<td>spark_job_class</td>
<td>177</td>
</tr>
<tr>
<td>spark_load_table</td>
<td>177</td>
</tr>
<tr>
<td>spark_log</td>
<td>178</td>
</tr>
<tr>
<td>spark_read_csv</td>
<td>178</td>
</tr>
<tr>
<td>spark_read_delta</td>
<td>180</td>
</tr>
<tr>
<td>spark_read_jdbc</td>
<td>181</td>
</tr>
<tr>
<td>spark_read_json</td>
<td>182</td>
</tr>
<tr>
<td>spark_read_libsvm</td>
<td>183</td>
</tr>
<tr>
<td>spark_read_orc</td>
<td>184</td>
</tr>
<tr>
<td>spark_read_parquet</td>
<td>185</td>
</tr>
<tr>
<td>spark_read_source</td>
<td>187</td>
</tr>
<tr>
<td>spark_read_table</td>
<td>188</td>
</tr>
<tr>
<td>spark_read_text</td>
<td>189</td>
</tr>
<tr>
<td>spark_save_table</td>
<td>190</td>
</tr>
<tr>
<td>spark_session_config</td>
<td>191</td>
</tr>
<tr>
<td>spark_table_name</td>
<td>191</td>
</tr>
<tr>
<td>spark_version</td>
<td>192</td>
</tr>
<tr>
<td>spark_version_from_home</td>
<td>192</td>
</tr>
<tr>
<td>spark_web</td>
<td>193</td>
</tr>
<tr>
<td>spark_write_csv</td>
<td>193</td>
</tr>
<tr>
<td>spark_write_delta</td>
<td>194</td>
</tr>
<tr>
<td>spark_write_jdbc</td>
<td>195</td>
</tr>
<tr>
<td>spark_write_json</td>
<td>196</td>
</tr>
<tr>
<td>spark_write_orc</td>
<td>197</td>
</tr>
<tr>
<td>spark_write_parquet</td>
<td>198</td>
</tr>
<tr>
<td>spark_write_source</td>
<td>199</td>
</tr>
</tbody>
</table>
checkpoint_directory

Set/Get Spark checkpoint directory

Description

Set/Get Spark checkpoint directory

Usage

spark_set_checkpoint_dir(sc, dir)

spark_get_checkpoint_dir(sc)
**compile_package_jars**

**Arguments**

- **sc**: A `spark_connection`.
- **dir**: Checkpoint directory, must be HDFS path of running on cluster.

**Description**

Compile 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**

```r
closest_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**

```r
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**

```r
## S3 method for class 'spark_connection'
copy_to(
  dest,
  df,
  name = spark_table_name(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.
download_scalac

Downloads default Scala Compilers

Description

compile_package_jars requires several versions of the scala compiler to work, this is to match Spark scala versions. To help setup your environment, this function will download the required compilers under the default search path.

Usage

download_scalac(dest_path = NULL)

Arguments

dest_path The destination path where scalac will be downloaded to.

Details

See find_scalac for a list of paths searched and used by this function to install the required compilers.

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.

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 (Transformer)

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. Column output is numeric for compatibility with other modeling functions.

Usage

ft_binarizer(
  x,
  input_col,
  output_col,
  threshold = 0,
  uid = random_string("binarizer_"),
  ...
)
Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **threshold**: Threshold used to binarize continuous features.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()

Examples

```r
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
iris_tbl %>%
  ft_binarizer(input_col = "Sepal.Length",
               output_col = "Sepal.Length_bin",
               threshold = 5) %>%
  select(Sepal.Length, Sepal.Length_bin, Species)
## End(Not run)
```
Description

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

Usage

```r
ft_bucketizer(
  x,
  input_col = NULL,
  output_col = NULL,
  splits = NULL,
  input_cols = NULL,
  output_cols = NULL,
  splits_array = NULL,
  handle_invalid = "error",
  uid = random_string("bucketizer_"),
  ...
)
```

Arguments

- `x` A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_col` The name of the input column.
- `output_col` The name of the output column.
- `splits` A numeric vector of cutpoints, indicating the bucket boundaries.
- `input_cols` Names of input columns.
- `output_cols` Names of output columns.
- `splits_array` Parameter for specifying multiple splits parameters. Each element in this array can be used to map continuous features into buckets.
- `handle_invalid` (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"
- `uid` A character string used to uniquely identify the feature transformer.
- `...` Optional arguments; currently unused.

Value

The object returned depends on the class of `x`. 
ft_chisq_selector

**Description**

Chi-Squared feature selection, which selects categorical features to use for predicting a categorical label.

- **spark_connection**: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.

- **tbl_spark**: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

**See Also**

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_chisq_selector()`, `ft_countVectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_maxAbsScaler()`, `ft_minMaxScaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_oneHotEncoder_estimator()`, `ft_oneHotEncoder()`, `ft_pca()`, `ft_polynomialExpansion()`, `ft_quantileDiscretizer()`, `ft_rFormula()`, `ft_regexTokenizer()`, `ft_sqlTransformer()`, `ft_standardScaler()`, `ft_stopWordsRemover()`, `ft_stringIndexer()`, `ft_tokenizer()`, `ft_vectorAssembler()`, `ft_vectorIndexer()`, `ft_vectorSlicer()`, `ft_word2vec()`

**Examples**

```r
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  ft_bucketizer(input_col = "Sepal_Length",
               output_col = "Sepal_Length_bucket",
               splits = c(0, 4.5, 5, 8)) %>%
  select(Sepal_Length, Sepal_Length_bucket, Species)

## End(Not run)
```
Usage

```r
ft_chisq_selector(
  x,
  features_col = "features",
  output_col = NULL,
  label_col = "label",
  selector_type = "numTopFeatures",
  fdr = 0.05,
  fpr = 0.05,
  fwe = 0.05,
  num_top_features = 50,
  percentile = 0.1,
  uid = random_string("chisq_selector_"),
  ...
)
```

Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **features_col**: Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- **output_col**: The name of the output column.
- **label_col**: Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`.
- **selector_type**: (Spark 2.1.0+) The selector type of the ChisqSelector. Supported options: "numTopFeatures" (default), "percentile", "fpr", "fdr", "fwe".
- **fdr**: (Spark 2.2.0+) The upper bound of the expected false discovery rate. Only applicable when selector_type = "fdr". Default value is 0.05.
- **fpr**: (Spark 2.1.0+) The highest p-value for features to be kept. Only applicable when selector_type = "fpr". Default value is 0.05.
- **fwe**: (Spark 2.2.0+) The upper bound of the expected family-wise error rate. Only applicable when selector_type = "fwe". Default value is 0.05.
- **num_top_features**: Number of features that selector will select, ordered by ascending p-value. If the number of features is less than `num_top_features`, then this will select all features. Only applicable when selector_type = "numTopFeatures". The default value of `num_top_features` is 50.
- **percentile**: (Spark 2.1.0+) Percentile of features that selector will select, ordered by statistics value descending. Only applicable when selector_type = "percentile". Default value is 0.1.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.
**Details**

In the case where \( x \) is a `tbl_spark`, the estimator fits against \( x \) to obtain a transformer, which is then immediately used to transform \( x \), returning a `tbl_spark`.

**Value**

The object returned depends on the class of \( x \).

- **spark_connection**: When \( x \) is a `spark_connection`, the function returns a `ml_transformer`, `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When \( x \) is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When \( x \) is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

**See Also**

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tfidf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standardScaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

**ft_count_vectorizer**  
**Feature Transformation – CountVectorizer (Estimator)**

**Description**

Extracts a vocabulary from document collections.

**Usage**

```r
ft_count_vectorizer(  
x,  
input_col = NULL,  
output_col = NULL,  
binary = FALSE,  
min_df = 1,  
min_tf = 1,  
vocab_size = 2^18,  
uid = random_string("count_vectorizer_"),
```
...)
}

ml_vocabulary(model)

**Arguments**

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **binary**: Binary toggle to control the output vector values. If True, all nonzero counts (after min_tf filter applied) are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default: FALSE
- **min_df**: Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer greater than or equal to 1, this specifies the number of documents the term must appear in; if this is a double in [0,1), then this specifies the fraction of documents. Default: 1.
- **min_tf**: Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer greater than or equal to 1, then this specifies a count (of times the term must appear in the document); if this is a double in [0,1), then this specifies a fraction (out of the document's token count). Default: 1.
- **vocab_size**: Build a vocabulary that only considers the top vocab_size terms ordered by term frequency across the corpus. Default: $2^{18}$.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.
- **model**: A ml_count_vectorizer_model.

**Details**

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

**Value**

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

ml_vocabulary() returns a vector of vocabulary built.
ft_dct

Feature Transformation – Discrete Cosine Transform (DCT) (Transformer)

Description

A feature transformer that takes the 1D discrete cosine transform of a real vector. No zero padding is performed on the input vector. It returns a real vector of the same length representing the DCT. The return vector is scaled such that the transform matrix is unitary (aka scaled DCT-II).

Usage

```r
ft_dct(
  x,
  input_col = NULL,
  output_col = NULL,
  inverse = FALSE,
  uid = random_string("dct_"),
  ...)
```

```r
ft_discrete_cosine_transform(
  x,
  input_col,
  output_col,
  inverse = FALSE,
  uid = random_string("dct_"),
  ...)
```

Arguments

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_col`: The name of the input column.
- `output_col`: The name of the output column.
ft_elementwise_product

inverse Indicates whether to perform the inverse DCT (TRUE) or forward DCT (FALSE).
uid A character string used to uniquely identify the feature transformer.
... Optional arguments; currently unused.

Details

ft_discrete_cosine_transform() is an alias for ft_dct for backwards compatibility.

Value

The object returned depends on the class of x.

• spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
• ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
• tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
ft_elementwise_product

Usage

ft_elementwise_product(
    x,
    input_col = NULL,
    output_col = NULL,
    scaling_vec = NULL,
    uid = random_string("elementwise_product_"),
    ...
)

Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **scaling_vec**: The vector to multiply with input vectors.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When `x` is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When `x` is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark.

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`
ft_feature_hasher  
Feature Transformation – FeatureHasher (Transformer)

Description
Feature Transformation – FeatureHasher (Transformer)

Usage

ft_feature_hasher(
  x,  
  input_cols = NULL,  
  output_col = NULL,  
  num_features = 2^18,  
  categorical_cols = NULL,  
  uid = random_string("feature_hasher_"),  
  ...  
)

Arguments

x  A spark_connection, ml_pipeline, or a tbl_spark.
input_cols  Names of input columns.
output_col  Name of output column.
um_features  Number of features. Defaults to 2^18.
categorical_cols  Numeric columns to treat as categorical features. By default only string and boolean columns are treated as categorical, so this param can be used to explicitly specify the numerical columns to treat as categorical.
uid  A character string used to uniquely identify the feature transformer.
...  Optional arguments; currently unused.

Details
Feature hashing projects a set of categorical or numerical features into a feature vector of specified dimension (typically substantially smaller than that of the original feature space). This is done using the hashing trick [https://en.wikipedia.org/wiki/Feature_hashing](https://en.wikipedia.org/wiki/Feature_hashing) to map features to indices in the feature vector.

The FeatureHasher transformer operates on multiple columns. Each column may contain either numeric or categorical features. Behavior and handling of column data types is as follows:

- Numeric columns: For numeric features, the hash value of the column name is used to map the feature value to its index in the feature vector. By default, numeric features are not treated as categorical (even when they are integers). To treat them as categorical, specify the relevant columns in categorical_cols.

- String columns: For categorical features, the hash value of the string "column_name=value" is used to map to the vector index, with an indicator value of 1.0. Thus, categorical features are
"one-hot" encoded (similarly to using OneHotEncoder with drop_last=FALSE). Boolean columns: Boolean values are treated in the same way as string columns. That is, boolean features are represented as "column_name=true" or "column_name=false", with an indicator value of 1.0.

Null (missing) values are ignored (implicitly zero in the resulting feature vector).

The hash function used here is also the MurmurHash 3 used in HashingTF. Since a simple modulo on the hashed value is used to determine the vector index, it is advisable to use a power of two as the num_features parameter; otherwise the features will not be mapped evenly to the vector indices.

Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh()`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`
num_features = 2**18,
uid = random_string("hashing_tf_"),
...)

Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.
input_col The name of the input column.
output_col The name of the output column.
binary Binary toggle to control term frequency counts. If true, all non-zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. (default = FALSE)
num_features Number of features. Should be greater than 0. (default = 2**18)
uid A character string used to uniquely identify the feature transformer.
... Optional arguments; currently unused.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
Description

Compute the Inverse Document Frequency (IDF) given a collection of documents.

Usage

```r
ft_idf(
  x,
  input_col = NULL,
  output_col = NULL,
  min_doc_freq = 0,
  uid = random_string("idf_"),
  ...
)
```

Arguments

- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **min_doc_freq**: The minimum number of documents in which a term should appear. Default: 0
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Details

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, which is then immediately used to transform `x`, returning a `tbl_spark`.

Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose `Pipeline` objects.
- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`
See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

**ft_imputer**  
*Feature Transformation – Imputer (Estimator)*

**Description**

Imputation estimator for completing missing values, either using the mean or the median of the columns in which the missing values are located. The input columns should be of numeric type. This function requires Spark 2.2.0+.

**Usage**

```r
ft_imputer(
  x,
  input_cols = NULL,
  output_cols = NULL,
  missing_value = NULL,
  strategy = "mean",
  uid = random_string("imputer_"),
  ...
)
```

**Arguments**

- `x`  
  A spark_connection, ml_pipeline, or a tbl_spark.

- `input_cols`  
  The names of the input columns.

- `output_cols`  
  The names of the output columns.

- `missing_value`  
  The placeholder for the missing values. All occurrences of `missing_value` will be imputed. Note that null values are always treated as missing.

- `strategy`  
  The imputation strategy. Currently only "mean" and "median" are supported. If "mean", then replace missing values using the mean value of the feature. If "median", then replace missing values using the approximate median value of the feature. Default: mean

- `uid`  
  A character string used to uniquely identify the feature transformer.

- `...`  
  Optional arguments; currently unused.
Details

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

Value

The object returned depends on the class of x.

• spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
• ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
• tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
ft_interaction

Feature Transformation – Interaction (Transformer)

Description

Implements the feature interaction transform. This transformer takes in Double and Vector type columns and outputs a flattened vector of their feature interactions. To handle interaction, we first one-hot encode any nominal features. Then, a vector of the feature cross-products is produced.
Usage

```r
ft_interaction(
  x, 
  input_cols = NULL, 
  output_col = NULL, 
  uid = random_string("interaction_"), 
  ... 
)
```

Arguments

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_cols`: The names of the input columns.
- `output_col`: The name of the output column.
- `uid`: A character string used to uniquely identify the feature transformer.
- `...`: Optional arguments; currently unused.

Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`
Feature Transformation – LSH (Estimator)

Description
Locality Sensitive Hashing functions for Euclidean distance (Bucketed Random Projection) and Jaccard distance (MinHash).

Usage

```r
ft_bucketed_random_projection_lsh(
  x,
  input_col = NULL,
  output_col = NULL,
  bucket_length = NULL,
  num_hash_tables = 1,
  seed = NULL,
  uid = random_string("bucketed_random_projection_lsh_"),
  ...
)
```

```r
ft_minhash_lsh(
  x,
  input_col = NULL,
  output_col = NULL,
  num_hash_tables = 1L,
  seed = NULL,
  uid = random_string("minhash_lsh_"),
  ...
)
```

Arguments

- `x` A `spark_connection, ml_pipeline, or a tbl_spark`.
- `input_col` The name of the input column.
- `output_col` The name of the output column.
- `bucket_length` The length of each hash bucket, a larger bucket lowers the false negative rate. The number of buckets will be \((\text{max L2 norm of input vectors}) / \text{bucketLength}\).
- `num_hash_tables` Number of hash tables used in LSH OR-amplification. LSH OR-amplification can be used to reduce the false negative rate. Higher values for this param lead to a reduced false negative rate, at the expense of added computational complexity.
- `seed` A random seed. Set this value if you need your results to be reproducible across repeated calls.
- `uid` A character string used to uniquely identify the feature transformer.
- `...` Optional arguments; currently unused.
In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

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

ft_lsh_utils

Other feature transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
ml_approx_similarity_join(
    model,
    dataset_a,
    dataset_b,
    threshold,
    dist_col = "distCol"
)

Arguments

model  | A fitted LSH model, returned by either `ft_minhash_lsh()` or `ft_bucketed_random_projection_lsh()`.
dataset | The dataset to search for nearest neighbors of the key.
key | Feature vector representing the item to search for.
um_nearest_neighbors | The maximum number of nearest neighbors.
dist_col | Output column for storing the distance between each result row and the key.
dataset_a | One of the datasets to join.
dataset_b | Another dataset to join.
threshold | The threshold for the distance of row pairs.

---

`ft_max_abs_scaler` | Feature Transformation – MaxAbsScaler (Estimator)

Description

Rescale each feature individually to range [-1, 1] by dividing through the largest maximum absolute value in each feature. It does not shift/center the data, and thus does not destroy any sparsity.

Usage

`ft_max_abs_scaler(
    x,
    input_col = NULL,
    output_col = NULL,
    uid = random_string("max_abs_scaler_"),
    ...
)

Arguments

x | A spark_connection, ml_pipeline, or a tbl_spark.
input_col | The name of the input column.
output_col | The name of the output column.
uid | A character string used to uniquely identify the feature transformer.
... | Optional arguments; currently unused.
Details

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.

- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

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

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width")

iris_tbl %>%
  ft_vector_assembler(input_col = features, output_col = "features_temp") %>%
  ft_max_abs_scaler(input_col = "features_temp", output_col = "features")

## End(Not run)
```
Feature Transformation – MinMaxScaler (Estimator)

Description
Rescale each feature individually to a common range [min, max] linearly using column summary statistics, which is also known as min-max normalization or Rescaling.

Usage
```
ft_min_max_scaler(
  x,
  input_col = NULL,
  output_col = NULL,
  min = 0,
  max = 1,
  uid = random_string("min_max_scaler_"),
  ...
)
```

Arguments
- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **min**: Lower bound after transformation, shared by all features. Default: 0.0
- **max**: Upper bound after transformation, shared by all features. Default: 1.0
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Details
In the case where x is a `tbl_spark`, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a `tbl_spark`.

Value
The object returned depends on the class of x.

- **spark_connection**: When x is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`
ft_ngram

Feature Transformation – NGram (Transformer)

Description
A feature transformer that converts the input array of strings into an array of n-grams. Null values in the input array are ignored. It returns an array of n-grams where each n-gram is represented by a space-separated string of words.

Usage
ft_ngram(
    x,
    input_col = NULL,
    output_col = NULL,
    n = 2,
    uid = random_string("ngram_"),
    ...
)

See Also
See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh(), ft_max_abs_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder(), ft_one_hot_encoder_estimator(), ft polynomial expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.

input_col The name of the input column.

output_col The name of the output column.

n Minimum n-gram length, greater than or equal to 1. Default: 2, bigram features

uid A character string used to uniquely identify the feature transformer.

Details

When the input is empty, an empty array is returned. When the input array length is less than n (number of elements per n-gram), no n-grams are returned.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an ml_transformer, an ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.

- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomialExpansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
ft_normalizer

Feature Transformation – Normalizer (Transformer)

Description

Normalize a vector to have unit norm using the given p-norm.

Usage

```r
ft_normalizer(
  x,  
  input_col = NULL, 
  output_col = NULL, 
  p = 2, 
  uid = random_string("normalizer_"), 
  ... 
)
```

Arguments

- `x` A spark_connection, ml_pipeline, or a tbl_spark.
- `input_col` The name of the input column.
- `output_col` The name of the output column.
- `p` Normalization in L^p space. Must be >= 1. Defaults to 2.
- `uid` A character string used to uniquely identify the feature transformer.
- `...` Optional arguments; currently unused.

Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- `ml_pipeline`: When `x` is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- `tbl_spark`: When `x` is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(),
ft_one_hot_encoder

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. Typically, used with `ft_string_indexer()` to index a column first.

Usage

```r
ft_one_hot_encoder(
  x,
  input_cols = NULL,
  output_cols = NULL,
  handle_invalid = NULL,
  drop_last = TRUE,
  uid = random_string("one_hot_encoder_"),
  ...
)
```

Arguments

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_cols`: The name of the input columns.
- `output_cols`: The name of the output columns.
- `handle_invalid`: (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"
- `drop_last`: Whether to drop the last category. Defaults to TRUE.
- `uid`: A character string used to uniquely identify the feature transformer.
- `...`: Optional arguments; currently unused.

Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
• ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.

• tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
handle_invalid (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"

drop_last Whether to drop the last category. Defaults to TRUE.

uid A character string used to uniquely identify the feature transformer.

... Optional arguments; currently unused.

Details

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_countVectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
Usage

ft_pca(
    x,
    input_col = NULL,
    output_col = NULL,
    k = NULL,
    uid = random_string("pca_"),
    ...
)

ml_pca(x, features = tbl_vars(x), k = length(features), pc_prefix = "PC", ...)

Arguments

x
A spark_connection, ml_pipeline, or a tbl_spark.

input_col
The name of the input column.

output_col
The name of the output column.

k
The number of principal components

uid
A character string used to uniquely identify the feature transformer.

features
The columns to use in the principal components analysis. Defaults to all columns in x.

pc_prefix
Length-one character vector used to prepend names of components.

Details

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

ml_pca() is a wrapper around ft_pca() that returns a ml_model.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark
See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh()`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder()`, `ft_one_hot_encoder_estimator()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

Examples

```r
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  select(-Species) %>%
  ml_pca(k = 2)

## End(Not run)
```

---

**ft_polynomial_expansion**

*Feature Transformation – PolynomialExpansion (Transformer)*

**Description**

Perform feature expansion in a polynomial space. E.g. take a 2-variable feature vector as an example: \((x, y)\), if we want to expand it with degree 2, then we get \((x, x \times x, y, x \times y, y \times y)\).

**Usage**

```r
ft_polynomial_expansion(
  x,
  input_col = NULL,
  output_col = NULL,
  degree = 2,
  uid = random_string("polynomial_expansion_"),
  ...)
```
ft_quantile_discretizer

Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **degree**: The polynomial degree to expand, which should be greater than equal to 1. A value of 1 means no expansion. Default: 2
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()

___

ft_quantile_discretizer

Feature Transformation – QuantileDiscretizer (Estimator)

___

Description

ft_quantile_discretizer takes a column with continuous features and outputs a column with binned categorical features. The number of bins can be set using the num_buckets parameter. It is possible that the number of buckets used will be smaller than this value, for example, if there are too few distinct values of the input to create enough distinct quantiles.
Usage

```r
ft_quantile_discretizer(
  x,
  input_col = NULL,
  output_col = NULL,
  num_buckets = 2,
  input_cols = NULL,
  output_cols = NULL,
  num_buckets_array = NULL,
  handle_invalid = "error",
  relative_error = 0.001,
  uid = random_string("quantile_discretizer_"),
  ...
)
```

Arguments

- `x` A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_col` The name of the input column.
- `output_col` The name of the output column.
- `num_buckets` Number of buckets (quantiles, or categories) into which data points are grouped. Must be greater than or equal to 2.
- `input_cols` Names of input columns.
- `output_cols` Names of output columns.
- `num_buckets_array` Array of number of buckets (quantiles, or categories) into which data points are grouped. Each value must be greater than or equal to 2.
- `handle_invalid` (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"
- `relative_error` (Spark 2.0.0+) Relative error (see documentation for org.apache.spark.sql.DataFrameStatFunctions.approxQuantile here for description). Must be in the range [0, 1]. default: 0.001
- `uid` A character string used to uniquely identify the feature transformer.
- `...` Optional arguments; currently unused.

Details

NaN handling: null and NaN values will be ignored from the column during QuantileDiscretizer fitting. This will produce a Bucketizer model for making predictions. During the transformation, Bucketizer will raise an error when it finds NaN values in the dataset, but the user can also choose to either keep or remove NaN values within the dataset by setting handle_invalid. If the user chooses to keep NaN values, they will be handled specially and placed into their own bucket, for example, if 4 buckets are used, then non-NaN data will be put into buckets[0-3], but NaNs will be counted in a special bucket[4].

Algorithm: The bin ranges are chosen using an approximate algorithm (see the documentation for org.apache.spark.sql.DataFrameStatFunctions.approxQuantile here for a detailed description). The
precision of the approximation can be controlled with the relative_error parameter. The lower and upper bin bounds will be -Infinity and +Infinity, covering all real values.

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

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.

- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh(), ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()

---

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

```r
ft_regex_tokenizer(
  x,
  input_col = NULL,
  output_col = NULL,
  gaps = TRUE,
  min_token_length = 1,
  pattern = "\s+",
  to_lower_case = TRUE,
  uid = random_string("regex_tokenizer_"),
  ...)
```

Arguments

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `input_col`: The name of the input column.
- `output_col`: The name of the output column.
- `gaps`: Indicates whether regex splits on gaps (TRUE) or matches tokens (FALSE).
- `min_token_length`: Minimum token length, greater than or equal to 0.
- `pattern`: The regular expression pattern to be used.
- `to_lower_case`: Indicates whether to convert all characters to lowercase before tokenizing.
- `uid`: A character string used to uniquely identify the feature transformer.
- `...`: Optional arguments; currently unused.

Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a `spark_connection`, the function returns a `ml_transformer`, `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- `ml_pipeline`: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- `tbl_spark`: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`,
ft_r_formula

Description

Implements the transforms required for fitting a dataset against an R model formula. Currently we support a limited subset of the R operators, including ~, ., :, +, and -. Also see the R formula docs here: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/formula

Usage

ft_r_formula(
  x,
  formula = NULL,
  features_col = "features",
  label_col = "label",
  force_index_label = FALSE,
  uid = random_string("r_formula_"),
  ...
)

Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.
formula R formula as a character string or a formula. Formula objects are converted to character strings directly and the environment is not captured.
features_col Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.
label_col Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.
force_index_label (Spark 2.1.0+) Force to index label whether it is numeric or string type. Usually we index label only when it is string type. If the formula was used by classification algorithms, we can force to index label even it is numeric type by setting this param with true. Default: FALSE.
uid A character string used to uniquely identify the feature transformer.
... Optional arguments; currently unused.
Details

The basic operators in the formula are:

- ~ separate target and terms
- + concat terms, "+ 0" means removing intercept
- - remove a term, "- 1" means removing intercept
- : interaction (multiplication for numeric values, or binarized categorical values)
- . all columns except target

Suppose a and b are double columns, we use the following simple examples to illustrate the effect of RFormula:

- \( y \sim a + b \) means model \( y \sim w_0 + w_1 \times a + w_2 \times b \) where \( w_0 \) is the intercept and \( w_1, w_2 \) are coefficients.
- \( y \sim a + b + a : b - 1 \) means model \( y \sim w_1 \times a + w_2 \times b + w_3 \times a \times b \) where \( w_1, w_2, w_3 \) are coefficients.

RFormula produces a vector column of features and a double or string column of label. Like when formulas are used in R for linear regression, string input columns will be one-hot encoded, and numeric columns will be cast to doubles. If the label column is of type string, it will be first transformed to double with StringIndexer. If the label column does not exist in the DataFrame, the output label column will be created from the specified response variable in the formula.

In the case where \( x \) is a tbl_spark, the estimator fits against \( x \) to obtain a transformer, which is then immediately used to transform \( x \), returning a tbl_spark.

Value

The object returned depends on the class of \( x \).

- spark_connection: When \( x \) is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When \( x \) is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When \( x \) is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
**ft_sql_transformer**  
*Feature Transformation – SQLTransformer*

**Description**

Implements the transformations which are defined by SQL statement. Currently we only support SQL syntax like ‘SELECT ... FROM __THIS__ ...’ where ‘__THIS__’ represents the underlying table of the input dataset. The select clause specifies the fields, constants, and expressions to display in the output, it can be any select clause that Spark SQL supports. Users can also use Spark SQL built-in function and UDFs to operate on these selected columns.

**Usage**

```r
ft_sql_transformer(
  x,
  statement = NULL,
  uid = random_string("sql_transformer_"),
  ...
)
```

```r
ft_dplyr_transformer(x, tbl, uid = random_string("dplyr_transformer_")), ...)
```

**Arguments**

- **x**
  A spark_connection, ml_pipeline, or a tbl_spark.
- **statement**
  A SQL statement.
- **uid**
  A character string used to uniquely identify the feature transformer.
- **...**
  Optional arguments; currently unused.
- **tbl**
  A tbl_spark generated using dplyr transformations.

**Details**

`ft_dplyr_transformer()` is a wrapper around `ft_sql_transformer()` that takes a tbl_spark instead of a SQL statement. Internally, the `ft_dplyr_transformer()` extracts the dplyr transformations used to generate tbl as a SQL statement then passes it on to `ft_sql_transformer()`. Note that only single-table dplyr verbs are supported and that the sdf_ family of functions are not.

**Value**

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark.
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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()

---

**ft_standard_scaler**  
*Feature Transformation – StandardScaler (Estimator)*

**Description**

Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set. The "unit std" is computed using the corrected sample standard deviation, which is computed as the square root of the unbiased sample variance.

**Usage**

```r
ft_standard_scaler(  
x,  
input_col = NULL,  
output_col = NULL,  
with_mean = FALSE,  
with_std = TRUE,  
uid = random_string("standard_scaler_"),  
...  
)
```

**Arguments**

- `x`  
  A spark_connection, ml_pipeline, or a tbl_spark.

- `input_col`  
  The name of the input column.

- `output_col`  
  The name of the output column.

- `with_mean`  
  Whether to center the data with mean before scaling. It will build a dense output, so take care when applying to sparse input. Default: FALSE

- `with_std`  
  Whether to scale the data to unit standard deviation. Default: TRUE

- `uid`  
  A character string used to uniquely identify the feature transformer.

- `...`  
  Optional arguments; currently unused.
Details

In the case where \( x \) is a tbl_spark, the estimator fits against \( x \) to obtain a transformer, which is then immediately used to transform \( x \), returning a tbl_spark.

Value

The object returned depends on the class of \( x \).

- **spark_connection**: When \( x \) is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When \( x \) is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When \( x \) is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width")

iris_tbl %>%
  ft_vector_assembler(input_col = features, output_col = "features_temp") %>%
  ft_standard_scaler(input_col = "features_temp", output_col = "features", with_mean = TRUE)

## End(Not run)
```
ft_stop_words_remover  Feature Transformation – StopWordsRemover (Transformer)

Description

A feature transformer that filters out stop words from input.

Usage

```r
ft_stop_words_remover(
  x,
  input_col = NULL,
  output_col = NULL,
  case_sensitive = FALSE,
  stop_words = ml_default_stop_words(spark_connection(x), "english"),
  uid = random_string("stop_words_remover "),
  ...
)
```

Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **case_sensitive**: Whether to do a case sensitive comparison over the stop words.
- **stop_words**: The words to be filtered out.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark.
ft_string_indexer

Feature Transformation – StringIndexer (Estimator)

Description

A label indexer that maps a string column of labels to an ML column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in \([0, \text{numLabels})\], ordered by label frequencies. So the most frequent label gets index 0. This function is the inverse of ft_index_to_string.

Usage

```scala
ft_string_indexer(
    x,
    input_col = NULL,
    output_col = NULL,
    handle_invalid = "error",
    string_order_type = "frequencyDesc",
    uid = random_string("string_indexer_"),
    ...
)
ml_labels(model)
```

```scala
ft_string_indexer_model(
    x,
    input_col = NULL,
    output_col = NULL,
    labels,
    handle_invalid = "error",
    uid = random_string("string_indexer_model_"),
    ...
)
```
ft_string_indexer

Arguments

x  A spark_connection, ml_pipeline, or a tbl_spark.
input_col  The name of the input column.
output_col  The name of the output column.
handle_invalid  (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"
string_order_type  (Spark 2.3+) How to order labels of string column. The first label after ordering is assigned an index of 0. Options are "frequencyDesc", "frequencyAsc", "alphabetDesc", and "alphabetAsc". Defaults to "frequencyDesc".
uid  A character string used to uniquely identify the feature transformer.
...  Optional arguments; currently unused.
model  A fitted StringIndexer model returned by ft_string_indexer()
labels  Vector of labels, corresponding to indices to be assigned.

Details

In the case where x is a tbl_spark, the estimator fits against x to obtain a transformer, which is then immediately used to transform x, returning a tbl_spark.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark

ml_labels() returns a vector of labels, corresponding to indices to be assigned.

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.

ft_index_to_string

Other feature transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estim ator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
ft_tokenizer  Feature Transformation – Tokenizer (Transformer)

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

Usage
```r
ft_tokenizer(
  x,
  input_col = NULL,
  output_col = NULL,
  uid = random_string("tokenizer_"),
  ...  
)
```

Arguments
- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.

Value
The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl_spark**: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

See Also
See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, etc.
ft_vector_assembler

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_cols = NULL,
  output_col = NULL,
  uid = random_string("vector_assembler_"),
  ...
)

Arguments

x: A spark_connection, ml_pipeline, or a tbl_spark.
input_cols: The names of the input columns
output_col: The name of the output column.
uid: A character string used to uniquely identify the feature transformer.
...: Optional arguments; currently unused.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns a ml_transformer, a ml_estimator, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the transformer or estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a transformer is constructed then immediately applied to the input tbl_spark, returning a tbl_spark
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 transformers: ft_binarizer(), ft_bucketizer(), ft_chisq_selector(), ft_count_vectorizer(), ft_dct(), ft_elementwise_product(), ft_feature_hasher(), ft_hashing_tf(), ft_idf(), ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(), ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(), ft_pca(), ft_polynomial_expansion(), ft_quantile_discretizer(), ft_r_formula(), ft_regex_tokenizer(), ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(), ft_tokenizer(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- `ml_pipeline`: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.

- `tbl_spark`: When `x` is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomialExpansion()`, `ft_quantileDiscretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_slicer()`, `ft_word2vec()`
Arguments

\[
x, \text{input\_col}, \text{output\_col}, \text{indices}, \text{uid}, \ldots
\]

- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **input\_col**: The name of the input column.
- **output\_col**: The name of the output column.
- **indices**: An vector of indices to select features from a vector column. Note that the indices are 0-based.
- **uid**: A character string used to uniquely identify the feature transformer.
- **\ldots**: Optional arguments; currently unused.

Value

The object returned depends on the class of \(x\).

- **spark\_connection**: When \(x\) is a `spark\_connection`, the function returns a `ml\_transformer`, `ml\_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.
- **ml\_pipeline**: When \(x\) is a `ml\_pipeline`, the function returns a `ml\_pipeline` with the transformer or estimator appended to the pipeline.
- **tbl\_spark**: When \(x\) is a `tbl\_spark`, a transformer is constructed then immediately applied to the input `tbl\_spark`, returning a `tbl\_spark`.

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft\_binarizer()`, `ft\_bucketizer()`, `ft\_chisq\_selector()`, `ft\_count\_vectorizer()`, `ft\_dct()`, `ft\_elementwise\_product()`, `ft\_feature\_hasher()`, `ft\_hashing\_tf()`, `ft\_idf()`, `ft\_imputer()`, `ft\_index\_to\_string()`, `ft\_interaction()`, `ft\_lsh`, `ft\_max\_abs\_scaler()`, `ft\_min\_max\_scaler()`, `ft\_ngram()`, `ft\_normalizer()`, `ft\_one\_hot\_encoder\_estimator()`, `ft\_one\_hot\_encoder()`, `ft\_pca()`, `ft\_polynomial\_expansion()`, `ft\_quantile\_discretizer()`, `ft\_r\_formula()`, `ft\_regex\_tokenizer()`, `ft\_sql\_transformer()`, `ft\_standard\_scaler()`, `ft\_stop\_words\_remover()`, `ft\_string\_indexer()`, `ft\_tokenizer()`, `ft\_vector\_assembler()`, `ft\_vector\_indexer()`, `ft\_word2vec()`

---

**ft\_word2vec**

*Feature Transformation – Word2Vec (Estimator)*

**Description**

Word2Vec transforms a word into a code for further natural language processing or machine learning process.
Usage

```
ft_word2vec(
  x,
  input_col = NULL,
  output_col = NULL,
  vector_size = 100,
  min_count = 5,
  max_sentence_length = 1000,
  num_partitions = 1,
  step_size = 0.025,
  max_iter = 1,
  seed = NULL,
  uid = random_string("word2vec_"),
  ...
)
```

```
ml_find_synonyms(model, word, num)
```

Arguments

- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **input_col**: The name of the input column.
- **output_col**: The name of the output column.
- **vector_size**: The dimension of the code that you want to transform from words. Default: 100
- **min_count**: The minimum number of times a token must appear to be included in the word2vec model’s vocabulary. Default: 5
- **max_sentence_length**: (Spark 2.0.0+) Sets the maximum length (in words) of each sentence in the input data. Any sentence longer than this threshold will be divided into chunks of up to max_sentence_length size. Default: 1000
- **num_partitions**: Number of partitions for sentences of words. Default: 1
- **step_size**: Param for Step size to be used for each iteration of optimization (> 0).
- **max_iter**: 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.
- **uid**: A character string used to uniquely identify the feature transformer.
- **...**: Optional arguments; currently unused.
- **model**: A fitted Word2Vec model, returned by `ft_word2vec()`.
- **word**: A word, as a length-one character vector.
- **num**: Number of words closest in similarity to the given word to find.

Details

In the case where `x` is a tbl_spark, the estimator fits against `x` to obtain a transformer, which is then immediately used to transform `x`, returning a tbl_spark.
hive_context_config

Value

The object returned depends on the class of \( x \).

- **spark_connection**: When \( x \) is a `spark_connection`, the function returns a `ml_transformer`, a `ml_estimator`, or one of their subclasses. The object contains a pointer to a Spark Transformer or Estimator object and can be used to compose Pipeline objects.

- **ml_pipeline**: When \( x \) is a `ml_pipeline`, the function returns a `ml_pipeline` with the transformer or estimator appended to the pipeline.

- **tbl_spark**: When \( x \) is a `tbl_spark`, a transformer is constructed then immediately applied to the input `tbl_spark`, returning a `tbl_spark`.

\[ \text{ml_find_synonyms()} \] returns a DataFrame of synonyms and cosine similarities.

See Also

See [http://spark.apache.org/docs/latest/ml-features.html](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 transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`

---

**hive_context_config**  
*Runtime configuration interface for Hive*

**Description**

Retrieves the runtime configuration interface for Hive.

**Usage**

`hive_context_config(sc)`

**Arguments**

- **sc** A `spark_connection`. 
invoke \hspace{1em} \textit{Invoke a Method on a JVM Object}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
invoke(jobj, method, ...)
invoke_static(sc, class, method, ...)
invoke_new(sc, class, ...)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{jobj} \hspace{1em} An R object acting as a Java object reference (typically, a spark\_jobj).
  \item \texttt{method} \hspace{1em} The name of the method to be invoked.
  \item \texttt{...} \hspace{1em} Optional arguments, currently unused.
  \item \texttt{sc} \hspace{1em} A spark\_connection.
  \item \texttt{class} \hspace{1em} The name of the Java class whose methods should be invoked.
\end{itemize}

\textbf{Details}

Use each of these functions in the following scenarios:

\begin{verbatim}
invoke \hspace{1em} Execute a method on a Java object reference (typically, a spark\_jobj).
invoke\_static \hspace{1em} Execute a static method associated with a Java class.
invoke\_new \hspace{1em} Invoke a constructor associated with a Java class.
\end{verbatim}

\textbf{Examples}

\begin{verbatim}
sc <- spark_connect(master = "spark://HOST:PORT")
spark_context(sc) %>%
  invoke("textFile", "file.csv", 1L) %>%
  invoke("count")
\end{verbatim}

\textit{livy\_config} \hspace{1em} \textit{Create a Spark Configuration for Livy}
livy_config

Description
Create a Spark Configuration for Livy

Usage
livy_config(
  config = spark_config(),
  username = NULL,
  password = NULL,
  negotiate = FALSE,
  custom_headers = list("X-Requested-By" = "sparklyr"),
  ...
)

Arguments
  config       Optional base configuration
  username     The username to use in the Authorization header
  password     The password to use in the Authorization header
  negotiate    Whether to use gssnegotiate method or not
  custom_headers List of custom headers to append to http requests. Defaults to list("X-Requested-By" = "sparklyr").
  ...

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.

The default value of "custom_headers" is set to list("X-Requested-By" = "sparklyr") in order to facilitate connection to Livy servers with CSRF protection enabled.

Additional parameters for Livy sessions are:
  proxy_user User to impersonate when starting the session
  jars jars to be used in this session
  py_files Python files to be used in this session
  files files to be used in this session
  driver_memory Amount of memory to use for the driver process
  driver_cores Number of cores to use for the driver process
  executor_memory Amount of memory to use per executor process
  executor_cores Number of cores to use for each executor
  num_executors Number of executors to launch for this session
  archives Archives to be used in this session
  queue The name of the YARN queue to which submitted
name  The name of this session
heartbeat_timeout  Timeout in seconds to which session be orphaned

Note that queue is supported only by version 0.4.0 of Livy or newer. If you are using the older one, specify queue via `config` (e.g. `config = spark_config(spark.yarn.queue = "my_queue")`).

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,
  stdout = "",
  stderr = "",
  ...
)
```

```
livy_service_stop()
```

Arguments

```
version            The version of ‘livy’ to use.
spark_version      The version of ‘spark’ to connect to.
stdout, stderr     where output to ‘stdout’ or ‘stderr’ should be sent. Same options as `system2`.
...                 Optional arguments; currently unused.
```
ml-params

Spark ML – ML Params

Description
Helper methods for working with parameters for ML objects.

Usage
ml_is_set(x, param, ...)
ml_param_map(x, ...)
ml_param(x, param, allow_null = FALSE, ...)
ml_params(x, params = NULL, allow_null = FALSE, ...)

Arguments
x A Spark ML object, either a pipeline stage or an evaluator.
param The parameter to extract or set.
... Optional arguments; currently unused.
allow_null Whether to allow NULL results when extracting parameters. If FALSE, an error will be thrown if the specified parameter is not found. Defaults to FALSE.
params A vector of parameters to extract.

ml-persistence

Spark ML – Model Persistence

Description
Save/load Spark ML objects

Usage
ml_save(x, path, overwrite = FALSE, ...)

## S3 method for class 'ml_model'
ml_save(
  x,
  path,
  overwrite = FALSE,
  type = c("pipeline_model", "pipeline"),
  ...
)

ml_load(sc, path)
Arguments

- **x**: A ML object, which could be a `ml_pipeline_stage` or a `ml_model`.
- **path**: The path where the object is to be serialized/deserialized.
- **overwrite**: Whether to overwrite the existing path, defaults to `FALSE`.
- **...**: Optional arguments; currently unused.
- **type**: Whether to save the pipeline model or the pipeline.
- **sc**: A Spark connection.

Value

- `ml_save()` serializes a Spark object into a format that can be read back into `sparklyr` or by the Scala or PySpark APIs. When called on `ml_model` objects, i.e. those that were created via the `tbl_spark-formula` signature, the associated pipeline model is serialized. In other words, the saved model contains both the data processing (`RFormulaModel`) stage and the machine learning stage.

- `ml_load()` reads a saved Spark object into `sparklyr`. It calls the correct Scala load method based on parsing the saved metadata. Note that a `PipelineModel` object saved from a `sparklyr ml_model` via `ml_save()` will be read back in as an `ml_pipeline_model`, rather than the `ml_model` object.

---

### Spark ML – Transform, fit, and predict methods (ml_ interface)

#### Description

Methods for transformation, fit, and prediction. These are mirrors of the corresponding `sdf-transform-methods`.

#### Usage

```r
is_ml_transformer(x)

is_ml_estimator(x)

ml_fit(x, dataset, ...)

ml_transform(x, dataset, ...)

ml_fit_and_transform(x, dataset, ...)

ml_predict(x, dataset, ...)

## S3 method for class 'ml_model_classification'
ml_predict(x, dataset, probability_prefix = "probability_", ...)
```
Arguments

- `x`: A `ml_estimator`, `ml_transformer` (or a list thereof), or `ml_model` object.
- `dataset`: A `tbl_spark`.
- `...`: Optional arguments; currently unused.
- `probability_prefix`: String used to prepend the class probability output columns.

Details

These methods are

Value

When `x` is an estimator, `ml_fit()` returns a transformer whereas `ml_fit_and_transform()` returns a transformed dataset. When `x` is a transformer, `ml_transform()` and `ml_predict()` return a transformed dataset. When `ml_predict()` is called on a `ml_model` object, additional columns (e.g. probabilities in case of classification models) are appended to the transformed output for the user's convenience.

ml-tuning Spark ML – Tuning

Description

Perform hyper-parameter tuning using either K-fold cross validation or train-validation split.

Usage

```
ml_sub_models(model)
ml_validation_metrics(model)
ml_cross_validator(
    x, 
    estimator = NULL, 
    estimator_param_maps = NULL, 
    evaluator = NULL, 
    num_folds = 3, 
    collect_sub_models = FALSE, 
    parallelism = 1, 
    seed = NULL, 
    uid = random_string("cross_validator_"), 
    ...
)
ml_train_validation_split(
```
x,
estimator = NULL,
estimator_param_maps = NULL,
evaluator = NULL,
train_ratio = 0.75,
collect_sub_models = FALSE,
parallelism = 1,
seed = NULL,
uid = random_string("train_validation_split_"),
...
)

**Arguments**

- **model**
  A cross validation or train-validation-split model.

- **x**
  A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.

- **estimator**
  A `ml_estimator` object.

- **estimator_param_maps**
  A named list of stages and hyper-parameter sets to tune. See details.

- **evaluator**
  A `ml_evaluator` object, see `ml_evaluator`.

- **num_folds**
  Number of folds for cross validation. Must be >= 2. Default: 3

- **collect_sub_models**
  Whether to collect a list of sub-models trained during tuning. If set to `FALSE`, then only the single best sub-model will be available after fitting. If set to `TRUE`, then all sub-models will be available. Warning: For large models, collecting all sub-models can cause OOMs on the Spark driver.

- **parallelism**
  The number of threads to use when running parallel algorithms. Default is 1 for serial execution.

- **seed**
  A random seed. Set this value if you need your results to be reproducible across repeated calls.

- **uid**
  A character string used to uniquely identify the ML estimator.

- **...**
  Optional arguments; currently unused.

- **train_ratio**
  Ratio between train and validation data. Must be between 0 and 1. Default: 0.75

**Details**

`ml_cross_validator()` performs k-fold cross validation while `ml_train_validation_split()` performs tuning on one pair of train and validation datasets.

**Value**

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns an instance of a `ml_cross_validator` or `ml_train_validation_split` object.
• `ml_pipeline`: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the tuning estimator appended to the pipeline.

• `tbl_spark`: When `x` is a `tbl_spark`, a tuning estimator is constructed then immediately fit with the input `tbl_spark`, returning a `ml_cross_validation_model` or a `ml_train_validation_split_model` object.

For cross validation, `ml_sub_models()` returns a nested list of models, where the first layer represents fold indices and the second layer represents param maps. For train-validation split, `ml_sub_models()` returns a list of models, corresponding to the order of the estimator param maps.

`ml_validation_metrics()` returns a data frame of performance metrics and hyperparameter combinations.

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

# Create a pipeline
pipeline <- ml_pipeline(sc) %>%
  ft_r_formula(Species ~ . ) %>%
  ml_random_forest_classifier()

# Specify hyperparameter grid
grid <- list(
  random_forest = list(
    num_trees = c(5, 10),
    max_depth = c(5, 10),
    impurity = c("entropy", "gini")
  )
)

# Create the cross validator object
cv <- ml_cross_validator(
  sc, estimator = pipeline, estimator_param_maps = grid,
  evaluator = ml_multiclass_classification_evaluator(sc),
  num_folds = 3,
  parallelism = 4
)

# Train the models
cv_model <- ml_fit(cv, iris_tbl)

# Print the metrics
ml_validation_metrics(cv_model)

## End(Not run)
```
ml_aft_survival_regression

Spark ML – Survival Regression

Description

Fit a parametric survival regression model named accelerated failure time (AFT) model (see Accelerated failure time model (Wikipedia)) based on the Weibull distribution of the survival time.

Usage

```r
ml_aft_survival_regression(
  x, 
  formula = NULL, 
  censor_col = "censor", 
  quantile_probabilities = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99), 
  fit_intercept = TRUE, 
  max_iter = 100L, 
  tol = 1e-06, 
  aggregation_depth = 2, 
  quantiles_col = NULL, 
  features_col = "features", 
  label_col = "label", 
  prediction_col = "prediction", 
  uid = random_string("aft_survival_regression_"), 
  ... 
)
```

```r
ml_survival_regression(
  x, 
  formula = NULL, 
  censor_col = "censor", 
  quantile_probabilities = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99), 
  fit_intercept = TRUE, 
  max_iter = 100L, 
  tol = 1e-06, 
  aggregation_depth = 2, 
  quantiles_col = NULL, 
  features_col = "features", 
  label_col = "label", 
  prediction_col = "prediction", 
  uid = random_string("aft_survival_regression_"), 
  response = NULL, 
  features = NULL, 
  ... 
)
```
Arguments

- **x**
  A spark_connection, ml_pipeline, or a tbl_spark.

- **formula**
  Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.

- **censor_col**
  Censor column name. The value of this column could be 0 or 1. If the value is 1, it means the event has occurred i.e. uncensored; otherwise censored.

- **quantile_probabilities**
  Quantile probabilities array. Values of the quantile probabilities array should be in the range (0, 1) and the array should be non-empty.

- **fit_intercept**
  Boolean; should the model be fit with an intercept term?

- **max_iter**
  The maximum number of iterations to use.

- **tol**
  Param for the convergence tolerance for iterative algorithms.

- **aggregation_depth**
  (Spark 2.1.0+) Suggested depth for treeAggregate (>= 2).

- **quantiles_col**
  Quantiles column name. This column will output quantiles of corresponding quantileProbabilities if it is set.

- **features_col**
  Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.

- **label_col**
  Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`.

- **prediction_col**
  Prediction column name.

- **uid**
  A character string used to uniquely identify the ML estimator.

- **...**
  Optional arguments; see Details.

- **response**
  (Deprecated) The name of the response column (as a length-one character vector.)

- **features**
  (Deprecated) The name of features (terms) to use for the model fit.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

`ml_survival_regression()` is an alias for `ml_aft_survival_regression()` for backwards compatibility.
Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- tbl_spark, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_decision_tree_classifier(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()

Examples

## Not run:
library(survival)
library(sparklyr)

sc <- spark_connect(master = "local")
ovarian_tbl <- sdf_copy_to(sc, ovarian, name = "ovarian_tbl", overwrite = TRUE)

partitions <- ovarian_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

ovarian_training <- partitions$training
ovarian_test <- partitions$test

sur_reg <- ovarian_training %>%
  ml_aft_survival_regression(futime ~ ecog_ps + rx + age + resid_ds, censor_col = "fustat")

pred <- ml_predict(sur_reg, ovarian_test)
pred

## End(Not run)
ml_als

Spark ML – ALS

Description

Perform recommendation using Alternating Least Squares (ALS) matrix factorization.

Usage

ml_als(
  x,
  formula = NULL,
  rating_col = "rating",
  user_col = "user",
  item_col = "item",
  rank = 10,
  reg_param = 0.1,
  implicit_prefs = FALSE,
  alpha = 1,
  nonnegative = FALSE,
  max_iter = 10,
  num_user_blocks = 10,
  num_item_blocks = 10,
  checkpoint_interval = 10,
  cold_start_strategy = "nan",
  intermediate_storage_level = "MEMORY_AND_DISK",
  final_storage_level = "MEMORY_AND_DISK",
  uid = random_string("als_"),
  ...
)

ml_recommend(model, type = c("items", "users"), n = 1)

Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.

formula Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details. The ALS model requires a specific formula format, please use rating_col ~ user_col + item_col.

rating_col Column name for ratings. Default: "rating"

user_col Column name for user ids. Ids must be integers. Other numeric types are supported for this column, but will be cast to integers as long as they fall within the integer value range. Default: "user"
item_col  Column name for item ids. Ids must be integers. Other numeric types are supported for this column, but will be cast to integers as long as they fall within the integer value range. Default: "item"

rank  Rank of the matrix factorization (positive). Default: 10

reg_param  Regularization parameter.

implicit_prefs  Whether to use implicit preference. Default: FALSE.

alpha  Alpha parameter in the implicit preference formulation (nonnegative).

nonnegative  Whether to apply nonnegativity constraints. Default: FALSE.

max_iter  Maximum number of iterations.

num_user_blocks  Number of user blocks (positive). Default: 10

num_item_blocks  Number of item blocks (positive). Default: 10

checkpoint_interval  Set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.

cold_start_strategy  (Spark 2.2.0+) Strategy for dealing with unknown or new users/items at prediction time. This may be useful in cross-validation or production scenarios, for handling user/item ids the model has not seen in the training data. Supported values: - "nan": predicted value for unknown ids will be NaN. - "drop": rows in the input DataFrame containing unknown ids will be dropped from the output DataFrame containing predictions. Default: "nan".

intermediate_storage_level  (Spark 2.0.0+) StorageLevel for intermediate datasets. Pass in a string representation of StorageLevel. Cannot be "NONE". Default: "MEMORY_AND_DISK".

final_storage_level  (Spark 2.0.0+) StorageLevel for ALS model factors. Pass in a string representation of StorageLevel. Default: "MEMORY_AND_DISK".

uid  A character string used to uniquely identify the ML estimator.

...  Optional arguments; currently unused.

model  An ALS model object

type  What to recommend, one of items or users

n  Maximum number of recommendations to return

Details

ml_recommend() returns the top n users/items recommended for each item/user, for all items/users. The output has been transformed (exploded and separated) from the default Spark outputs to be more user friendly.
**Value**

ALS attempts to estimate the ratings matrix \( R \) as the product of two lower-rank matrices, \( X \) and \( Y \), i.e. \( X \cdot Y^T = R \). Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.

This is a blocked implementation of the ALS factorization algorithm that groups the two sets of factors (referred to as "users" and "products") into blocks and reduces communication by only sending one copy of each user vector to each product block on each iteration, and only for the product blocks that need that user’s feature vector. This is achieved by pre-computing some information about the ratings matrix to determine the "out-links" of each user (which blocks of products it will contribute to) and "in-link" information for each product (which of the feature vectors it receives from each user block it will depend on). This allows us to send only an array of feature vectors between each user block and product block, and have the product block find the users’ ratings and update the products based on these messages.

For implicit preference data, the algorithm used is based on "Collaborative Filtering for Implicit Feedback Datasets", available at [https://doi.org/10.1109/ICDM.2008.22](https://doi.org/10.1109/ICDM.2008.22), adapted for the blocked approach used here.

Essentially instead of finding the low-rank approximations to the rating matrix \( R \), this finds the approximations for a preference matrix \( P \) where the elements of \( P \) are 1 if \( r \) is greater than 0 and 0 if \( r \) is less than or equal to 0. The ratings then act as 'confidence' values related to strength of indicated user preferences rather than explicit ratings given to items.

The object returned depends on the class of \( x \).

- **spark_connection**: When \( x \) is a spark_connection, the function returns an instance of a ml_als recommender object, which is an Estimator.
- **ml_pipeline**: When \( x \) is a ml_pipeline, the function returns a ml_pipeline with the recommender appended to the pipeline.
- **tbl_spark**: When \( x \) is a tbl_spark, a recommender estimator is constructed then immediately fit with the input tbl_spark, returning a recommendation model, i.e. ml_als_model.

**Examples**

```r
# Not run:

library(sparklyr)
sc <- spark_connect(master = "local")
movies <- data.frame(
  user = c(1, 2, 0, 1, 2, 0),
  item = c(1, 1, 1, 2, 2, 0),
  rating = c(3, 1, 2, 4, 5, 4)
)
movies_tbl <- sdf_copy_to(sc, movies)
model <- ml_als(movies_tbl, rating ~ user + item)
ml_predict(model, movies_tbl)
```
ml_recommend(model, type = "item", 1)
## End(Not run)

---

**ml_als_tidiers**

**Tidying methods for Spark ML ALS**

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```r
## S3 method for class 'ml_model_als'
tidy(x, ...)

## S3 method for class 'ml_model_als'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_als'
glance(x, ...)
```

**Arguments**

- `x` a Spark ML model.
- `...` extra arguments (not used.)
- `newdata` a tbl_spark of new data to use for prediction.

---

**ml_bisecting_kmeans**

**Spark ML – Bisecting K-Means Clustering**

**Description**

A bisecting k-means algorithm based on the paper "A comparison of document clustering techniques" by Steinbach, Karypis, and Kumar, with modification to fit Spark. The algorithm starts from a single cluster that contains all points. Iteratively it finds divisible clusters on the bottom level and bisects each of them using k-means, until there are k leaf clusters in total or no leaf clusters are divisible. The bisecting steps of clusters on the same level are grouped together to increase parallelism. If bisecting all divisible clusters on the bottom level would result more than k leaf clusters, larger clusters get higher priority.
ml_bisecting_kmeans

Usage

ml_bisecting_kmeans(
  x,
  formula = NULL,
  k = 4,
  max_iter = 20,
  seed = NULL,
  min_divisible_cluster_size = 1,
  features_col = "features",
  prediction_col = "prediction",
  uid = random_string("bisecting_bisecting_kmeans_"),
...
)

Arguments

x
  A spark_connection, ml_pipeline, or a tbl_spark.

formula
  Used when x is a tbl_spark. R formula as a character string or a formula.
  This is used to transform the input dataframe before fitting, see ft_r_formula for
details.

k
  The number of clusters to create

max_iter
  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.

min_divisible_cluster_size
  The minimum number of points (if greater than or equal to 1.0) or the minimum
  proportion of points (if less than 1.0) of a divisible cluster (default: 1.0).

features_col
  Features column name, as a length-one character vector. The column should
  be single vector column of numeric values. Usually this column is output by
  ft_r_formula.

prediction_col
  Prediction column name.

uid
  A character string used to uniquely identify the ML estimator.

...
  Optional arguments, see Details.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a
  ml_estimator object. The object contains a pointer to a Spark Estimator object and can be
  used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the clus-
tering estimator appended to the pipeline.
- tbl_spark: When x is a tbl_spark, an estimator is constructed then immediately fit with the
  input tbl_spark, returning a clustering model.
ml_chisquare_test

- tbl_spark, with formula or features specified: When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the estimator. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model. This signature does not apply to ml_lda().

See Also

See http://spark.apache.org/docs/latest/ml-clustering.html for more information on the set of clustering algorithms.

Other ml clustering algorithms: ml_gaussian_mixture(), ml_kmeans(), ml_lda()

Examples

```r
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  select(-Species) %>%
  ml_bisecting_kmeans(k = 4, Species ~ .)
## End(Not run)
```

---

**ml_chisquare_test**  
*Chi-square hypothesis testing for categorical data.*

**Description**

Conduct Pearson’s independence test for every feature against the label. For each feature, the (feature, label) pairs are converted into a contingency matrix for which the Chi-squared statistic is computed. All label and feature values must be categorical.

**Usage**

`ml_chisquare_test(x, features, label)`

**Arguments**

- `x` A tbl_spark.
- `features` The name(s) of the feature columns. This can also be the name of a single vector column created using `ft_vector_assembler()`.
- `label` The name of the label column.
Value

A data frame with one row for each (feature, label) pair with p-values, degrees of freedom, and test statistics.

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
features <- c("Petal_Width", "Petal_Length", "Sepal_Length", "Sepal_Width")
ml_chisquare_test(iris_tbl, features = features, label = "Species")
## End(Not run)
```

---

**ml_clustering_evaluator**

*Spark ML - Clustering Evaluator*

**Description**

Evaluator for clustering results. The metric computes the Silhouette measure using the squared Euclidean distance. The Silhouette is a measure for the validation of the consistency within clusters. It ranges between 1 and -1, where a value close to 1 means that the points in a cluster are close to the other points in the same cluster and far from the points of the other clusters.

**Usage**

```r
ml_clustering_evaluator(
  x, 
  features_col = "features", 
  prediction_col = "prediction", 
  metric_name = "silhouette", 
  uid = random_string("clustering_evaluator_"), 
  ... 
)
```

**Arguments**

- `x` A `spark_connection` object or a `tbl_spark` containing label and prediction columns. The latter should be the output of `sdf_predict`.
- `features_col` Name of features column.
- `prediction_col` Name of the prediction column.
- `metric_name` The performance metric. Currently supports "silhouette".
- `uid` A character string used to uniquely identify the ML estimator.
- `...` Optional arguments; currently unused.
Value

The calculated performance metric

Examples

## Not run:
```
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

formula <- Species ~ .

# Train the models
kmeans_model <- ml_kmeans(iris_training, formula = formula)
b_kmeans_model <- ml_bisecting_kmeans(iris_training, formula = formula)
gmm_model <- ml_gaussian_mixture(iris_training, formula = formula)

# Predict
pred_kmeans <- ml_predict(kmeans_model, iris_test)
pred_b_kmeans <- ml_predict(b_kmeans_model, iris_test)
pred_gmm <- ml_predict(gmm_model, iris_test)

# Evaluate
ml_clustering_evaluator(pred_kmeans)
ml_clustering_evaluator(pred_b_kmeans)
ml_clustering_evaluator(pred_gmm)
```

## End(Not run)

---

**ml_corr**

Compute correlation matrix

Description

Compute correlation matrix

Usage

```
ml_corr(x, columns = NULL, method = c("pearson", "spearman"))
```

Arguments

- **x**: A tbl_spark.
columns       The names of the columns to calculate correlations of. If only one column is specified, it must be a vector column (for example, assembled using `ft_vectorAssembler()`).
method        The method to use, either "pearson" or "spearman".

Value
A correlation matrix organized as a data frame.

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
features <- c("Petal_Width", "Petal_Length", "Sepal_Length", "Sepal_Width")
ml_corr(iris_tbl, columns = features, method = "pearson")
## End(Not run)
```

ml_decision_tree_classifier

Spark ML – Decision Trees

Description
Perform classification and regression using decision trees.

Usage

```r
ml_decision_tree_classifier(
  x,
  formula = NULL,
  max_depth = 5,
  max_bins = 32,
  min_instances_per_node = 1,
  min_info_gain = 0,
  impurity = "gini",
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  checkpoint_interval = 10,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
)```
uid = random_string("decision_tree_classifier_"),
...

ml_decision_tree(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  variance_col = NULL,
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  checkpoint_interval = 10L,
  impurity = "auto",
  max_bins = 32L,
  max_depth = 5L,
  min_info_gain = 0,
  min_instances_per_node = 1L,
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256L,
  uid = random_string("decision_tree_"),
  response = NULL,
  features = NULL,
  ...
)

ml_decision_tree_regressor(
  x,
  formula = NULL,
  max_depth = 5,
  max_bins = 32,
  min_instances_per_node = 1,
  min_info_gain = 0,
  impurity = "variance",
  seed = NULL,
  cache_node_ids = FALSE,
  checkpoint_interval = 10,
  max_memory_in_mb = 256,
  variance_col = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("decision_tree_regressor_"),
  ...
)
Arguments

x  A spark_connection, ml_pipeline, or a tbl_spark.

formula  Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.

max_depth  Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.

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.

min_instances_per_node  Minimum number of instances each child must have after split.

min_info_gain  Minimum information gain for a split to be considered at a tree node. Should be >= 0, defaults to 0.

impurity  Criterion used for information gain calculation. Supported: "entropy" and "gini" (default) for classification and "variance" (default) for regression. For ml_decision_tree, setting "auto" will default to the appropriate criterion based on model type.

seed  Seed for random numbers.

thresholds  Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value \( p/t \) is predicted, where \( p \) is the original probability of that class and \( t \) is the class’s threshold.

cache_node_ids  If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.

checkpoint_interval  Set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.

max_memory_in_mb  Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.

features_col  Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.

label_col  Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.

prediction_col  Prediction column name.

probability_col  Column name for predicted class conditional probabilities.
raw_prediction_col

Raw prediction (a.k.a. confidence) column name.

uid

A character string used to uniquely identify the ML estimator.

Optional arguments; see Details.

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.

variance_col

(Optional) Column name for the biased sample variance of prediction.

response

(Deprecated) The name of the response column (as a length-one character vector.)

features

(Deprecated) The name of features (terms) to use for the model fit.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

ml_decision_tree is a wrapper around ml_decision_tree_regressor.tbl_spark and ml_decision_tree_classifier.tbl_spark and calls the appropriate method based on model type.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- tbl_spark, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()
Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)
iris_training <- partitions$training
iris_test <- partitions$test

dt_model <- iris_training %>%
  ml_decision_tree(Species ~ .)
pred <- ml_predict(dt_model, iris_test)
ml_multiclass_classification_evaluator(pred)
## End(Not run)
```

---

### `ml_default_stop_words`

**Description**

Loads the default stop words for the given language.

**Usage**

```r
ml_default_stop_words(
  sc,
  language = c("english", "danish", "dutch", "finnish", "french", "german",
               "hungarian", "italian", "norwegian", "portuguese", "russian", "spanish", "swedish",
               "turkish"),
  ...
)
```

**Arguments**

- `sc` A `spark_connection`
- `language` A character string.
- `...` Optional arguments; currently unused.

**Details**

Value

A list of stop words.

See Also

`ft_stop_words_remover`

---

### `ml_evaluate`

*Evaluate the Model on a Validation Set*

**Description**

Compute performance metrics.

**Usage**

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_model_logistic_regression`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_logistic_regression_model`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_model_linear_regression`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_linear_regression_model`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_model_generalized_linear_regression`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_generalized_linear_regression_model`

```r
ml_evaluate(x, dataset)
```

#### S3 method for class `ml_evaluator`

```r
ml_evaluate(x, dataset)
```

**Arguments**

- `x` An ML model object or an evaluator object.
- `dataset` The dataset to be validate the model on.
Description

A set of functions to calculate performance metrics for prediction models. Also see the Spark ML Documentation https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.ml.evaluation.package

Usage

```scala
ml_binary_classification_evaluator(
  x,
  label_col = "label",
  raw_prediction_col = "rawPrediction",
  metric_name = "areaUnderROC",
  uid = random_string("binary_classification_evaluator_"),
  ...
)

ml_binary_classification_eval(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "areaUnderROC"
)

ml_multiclass_classification_evaluator(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "f1",
  uid = random_string("multiclass_classification_evaluator_"),
  ...
)

ml_classification_eval(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "f1"
)

ml_regression_evaluator(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "rmse",
)```
uid = random_string("regression_evaluator_"),
...
)

Arguments

x A spark_connection object or a tbl_spark containing label and prediction columns. The latter should be the output of sdf_predict.

label_col Name of column string specifying which column contains the true labels or values.

raw_prediction_col Raw prediction (a.k.a. confidence) column name.

metric_name The performance metric. See details.

uid A character string used to uniquely identify the ML estimator.

Optional arguments; currently unused.

prediction_col Name of the column that contains the predicted label or value NOT the scored probability. Column should be of type Double.

Details

The following metrics are supported

- Binary Classification: areaUnderROC (default) or areaUnderPR (not available in Spark 2.X.)
- Multiclass Classification: f1 (default), precision, recall, weightedPrecision, weightedRecall or accuracy; for Spark 2.X: f1 (default), weightedPrecision, weightedRecall or accuracy.
- Regression: rmse (root mean squared error, default), mse (mean squared error), r2, or mae (mean absolute error.)

ml_binary_classification_eval() is an alias for ml_binary_classification_evaluator() for backwards compatibility.

ml_classification_eval() is an alias for ml_multiclass_classification_evaluator() for backwards compatibility.

Value

The calculated performance metric

Examples

```r
# Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test
```
# for multiclass classification
rf_model <- mtcars_training %>%
  ml_random_forest(cyl ~ ., type = "classification")

pred <- ml_predict(rf_model, mtcars_test)
ml_multiclass_classification_evaluator(pred)

# for regression
rf_model <- mtcars_training %>%
  ml_random_forest(cyl ~ ., type = "regression")

pred <- ml_predict(rf_model, mtcars_test)
ml_regression_evaluator(pred, label_col = "cyl")

# for binary classification
rf_model <- mtcars_training %>%
  ml_random_forest(am ~ gear + carb, type = "classification")

pred <- ml_predict(rf_model, mtcars_test)
ml_binary_classification_evaluator(pred)

## End(Not run)

---

**ml_feature_importances**

*Spark ML - Feature Importance for Tree Models*

**Description**

Spark ML - Feature Importance for Tree Models

**Usage**

```r
ml_feature_importances(model, ...)
ml_tree_feature_importance(model, ...)
```

**Arguments**

- `model`: A decision tree-based model.
- `...`: Optional arguments; currently unused.
ml_fpgrowth

**Value**

For `ml_model`, a sorted data frame with feature labels and their relative importance. For `ml_prediction_model`, a vector of relative importances.

**Description**

A parallel FP-growth algorithm to mine frequent itemsets.

**Usage**

```r
ml_fpgrowth(
  x,
  items_col = "items",
  min_confidence = 0.8,
  min_support = 0.3,
  prediction_col = "prediction",
  uid = random_string("fpgrowth_"),
  ...
)
```

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `items_col`: Items column name. Default: "items"
- `min_confidence`: Minimal confidence for generating Association Rule. `min_confidence` will not affect the mining for frequent itemsets, but will affect the association rules generation. Default: 0.8
- `min_support`: Minimal support level of the frequent pattern. [0.0, 1.0]. Any pattern that appears more than (min_support * size-of-the-dataset) times will be output in the frequent itemsets. Default: 0.3
- `prediction_col`: Prediction column name.
- `uid`: A character string used to uniquely identify the ML estimator.
- `...`: Optional arguments; currently unused.
- `model`: A fitted FP-Growth model returned by `ml_fpgrowth()`
ml_gaussian_mixture

Spark ML – Gaussian Mixture clustering.

Description

This class performs expectation maximization for multivariate Gaussian Mixture Models (GMMs). A GMM represents a composite distribution of independent Gaussian distributions with associated "mixing" weights specifying each’s contribution to the composite. Given a set of sample points, this class will maximize the log-likelihood for a mixture of k Gaussians, iterating until the log-likelihood changes by less than `tol`, or until it has reached the max number of iterations. While this process is generally guaranteed to converge, it is not guaranteed to find a global optimum.

Usage

```r
ml_gaussian_mixture(
  x,
  formula = NULL,
  k = 2,
  max_iter = 100,
  tol = 0.01,
  seed = NULL,
  features_col = "features",
  prediction_col = "prediction",
  probability_col = "probability",
  uid = random_string("gaussian_mixture_"),
  ...)
```

Arguments

- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **formula**: Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- **k**: The number of clusters to create.
- **max_iter**: The maximum number of iterations to use.
- **tol**: Param for the convergence tolerance for iterative algorithms.
- **seed**: A random seed. Set this value if you need your results to be reproducible across repeated calls.
- **features_col**: Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- **prediction_col**: Prediction column name.
**probability_col**  
Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.

**uid**  
A character string used to uniquely identify the ML estimator.

...  
Optional arguments, see Details.

### Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns an instance of a `ml_estimator` object. The object contains a pointer to a Spark Estimator object and can be used to compose Pipeline objects.

- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the clustering estimator appended to the pipeline.

- **tbl_spark**: When `x` is a `tbl_spark`, an estimator is constructed then immediately fit with the input `tbl_spark`, returning a clustering model.

- **tbl_spark**, with formula or features specified: When `formula` is specified, the input `tbl_spark` is first transformed using an `RFormula` transformer before being fit by the estimator. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`. This signature does not apply to `ml_lda()`.

### See Also

See [http://spark.apache.org/docs/latest/ml-clustering.html](http://spark.apache.org/docs/latest/ml-clustering.html) for more information on the set of clustering algorithms.

Other ml clustering algorithms: `ml_bisecting_kmeans()`, `ml_kmeans()`, `ml_lda()`

### Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

gmm_model <- ml_gaussian_mixture(iris_tbl, Species ~ .)
pred <- sdf_predict(iris_tbl, gmm_model)
ml_clustering_evaluator(pred)

## End(Not run)
```
ml_gbt_classifier

Spark ML – Gradient Boosted Trees

Description

Perform binary classification and regression using gradient boosted trees. Multiclass classification
is not supported yet.

Usage

ml_gbt_classifier(
  x,
  formula = NULL,
  max_iter = 20,
  max_depth = 5,
  step_size = 0.1,
  subsampling_rate = 1,
  feature_subset_strategy = "auto",
  min_instances_per_node = 1L,
  max_bins = 32,
  min_info_gain = 0,
  loss_type = "logistic",
  seed = NULL,
  thresholds = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  uid = random_string("gbt_classifier_"),
  ...
)

ml_gradient_boosted_trees(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  checkpoint_interval = 10,
  loss_type = c("auto", "logistic", "squared", "absolute"),
  ...
ml_gbt_classifier

```r
max_bins = 32,
max_depth = 5,
max_iter = 20L,
min_info_gain = 0,
min_instances_per_node = 1,
step_size = 0.1,
subsampling_rate = 1,
feature_subset_strategy = "auto",
seed = NULL,
thresholds = NULL,
cache_node_ids = FALSE,
max_memory_in_mb = 256,
uid = random_string("gradient_boosted_trees_"),
response = NULL,
features = NULL,
...
)
ml_gbt_regressor(
  x,
  formula = NULL,
  max_iter = 20,
  max_depth = 5,
  step_size = 0.1,
  subsampling_rate = 1,
  feature_subset_strategy = "auto",
  min_instances_per_node = 1,
  max_bins = 32,
  min_info_gain = 0,
  loss_type = "squared",
  seed = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("gbt_regressor_"),
  ...
)
```

**Arguments**

- **x**
  - A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.

- **formula**
  - Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.

- **max_iter**
  - Maximum number of iterations.
max_depth  Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.

step_size  Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator. (default = 0.1)

subsampling_rate  Fraction of the training data used for learning each decision tree, in range (0, 1]. (default = 1.0)

feature_subset_strategy  The number of features to consider for splits at each tree node. See details for options.

min_instances_per_node  Minimum number of instances each child must have after split.

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.

min_info_gain  Minimum information gain for a split to be considered at a tree node. Should be >= 0, defaults to 0.

loss_type  Loss function which GBT tries to minimize. Supported: "squared" (L2) and "absolute" (L1) (default = squared) for regression and "logistic" (default) for classification. For ml_gradient_boosted_trees, setting "auto" will default to the appropriate loss type based on model type.

seed  Seed for random numbers.

thresholds  Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value \(p/t\) is predicted, where \(p\) is the original probability of that class and \(t\) is the class’s threshold.

checkpoint_interval  Set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.

cache_node_ids  If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.

max_memory_in_mb  Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.

features_col  Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.

label_col  Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.

prediction_col  Prediction column name.

probability_col  Column name for predicted class conditional probabilities.
ml_gbt_classifier

raw_prediction_col
   Raw prediction (a.k.a. confidence) column name.

uid
   A character string used to uniquely identify the ML estimator.

...  
   Optional arguments; see Details.

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.

response
   (Deprecated) The name of the response column (as a length-one character vector).

features
   (Deprecated) The name of features (terms) to use for the model fit.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

The supported options for feature_subset_strategy are

- "auto": Choose automatically for task: If num_trees == 1, set to "all". If num_trees > 1 (forest), set to "sqrt" for classification and to "onethird" for regression.
- "all": use all features
- "onethird": use 1/3 of the features
- "sqrt": use use sqrt(number of features)
- "log2": use log2(number of features)
- "n": when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features. (default = "auto")

ml_gradient_boosted_trees is a wrapper around ml_gbt_regressor.tbl_spark and ml_gbt_classifier.tbl_spark and calls the appropriate method based on model type.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
• tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.

• tbl_spark, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()

Examples

## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)
iris_training <- partitions$training
iris_test <- partitions$test
gbt_model <- iris_training %>%
  ml_gradient_boosted_trees(Sepal_Length ~ Petal_Length + Petal_Width)
pred <- ml_predict(gbt_model, iris_test)
ml_regression_evaluator(pred, label_col = "Sepal_Length")
## End(Not run)

ml_generalized_linear_regression

Spark ML – Generalized Linear Regression

Description

Perform regression using Generalized Linear Model (GLM).
ml_generalized_linear_regression(  
  x, 
  formula = NULL, 
  family = "gaussian", 
  link = NULL, 
  fit_intercept = TRUE, 
  offset_col = NULL, 
  link_power = NULL, 
  link_prediction_col = NULL, 
  reg_param = 0, 
  max_iter = 25, 
  weight_col = NULL, 
  solver = "irls", 
  tol = 1e-06, 
  variance_power = 0, 
  features_col = "features", 
  label_col = "label", 
  prediction_col = "prediction", 
  uid = random_string("generalized_linear_regression_"), 
  ... 
)

Arguments

x  A spark_connection, ml_pipeline, or a tbl_spark.
formula  Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.
family  Name of family which is a description of the error distribution to be used in the model. Supported options: "gaussian", "binomial", "poisson", "gamma" and "tweedie". Default is "gaussian".
link  Name of link function which provides the relationship between the linear predictor and the mean of the distribution function. See for supported link functions.
fit_intercept  Boolean; should the model be fit with an intercept term?
offset_col  Offset column name. If this is not set, we treat all instance offsets as 0.0. The feature specified as offset has a constant coefficient of 1.0.
link_power  Index in the power link function. Only applicable to the Tweedie family. Note that link power 0, 1, -1 or 0.5 corresponds to the Log, Identity, Inverse or Sqrt link, respectively. When not set, this value defaults to 1 - variancePower, which matches the R "statmod" package.
link_prediction_col  Link prediction (linear predictor) column name. Default is not set, which means we do not output link prediction.
reg_param  Regularization parameter (aka lambda)
max_iter  The maximum number of iterations to use.
weight_col: The name of the column to use as weights for the model fit.
solver: Solver algorithm for optimization.
tol: Param for the convergence tolerance for iterative algorithms.
variance_power: Power in the variance function of the Tweedie distribution which provides the relationship between the variance and mean of the distribution. Only applicable to the Tweedie family. (see Tweedie Distribution (Wikipedia)) Supported values: 0 and [1, Inf). Note that variance power 0, 1, or 2 corresponds to the Gaussian, Poisson or Gamma family, respectively.
features_col: Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by \texttt{ft_r_formula}.
label_col: Label column name. The column should be a numeric column. Usually this column is output by \texttt{ft_r_formula}.
prediction_col: Prediction column name.
uid: A character string used to uniquely identify the ML estimator.
...
Optional arguments; see Details.

Details

When \texttt{x} is a \texttt{tbl_spark} and \texttt{formula} (alternatively, response and features) is specified, the function returns a \texttt{ml_model} object wrapping a \texttt{ml_pipeline_model} which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument \texttt{predicted_label_col} (defaults to \texttt{"predicted_label"}) can be used to specify the name of the predicted label column. In addition to the fitted \texttt{ml_pipeline_model}, \texttt{ml_model} objects also contain a \texttt{ml_pipeline} object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by \texttt{ml_save} with \texttt{type = "pipeline"} to facilitate model refresh workflows.

Valid link functions for each family is listed below. The first link function of each family is the default one.

- \texttt{gaussian}: "identity", "log", "inverse"
- \texttt{binomial}: "logit", "probit", "loglog"
- \texttt{poisson}: "log", "identity", "sqrt"
- \texttt{gamma}: "inverse", "identity", "log"
- \texttt{tweedie}: power link function specified through \texttt{link_power}. The default link power in the tweedie family is \(1 - \text{variance\_power}\).

Value

The object returned depends on the class of \texttt{x}.

- \texttt{spark_connection}: When \texttt{x} is a \texttt{spark_connection}, the function returns an instance of a \texttt{ml_estimator} object. The object contains a pointer to a Spark \texttt{Predictor} object and can be used to compose \texttt{Pipeline} objects.
- \texttt{ml_pipeline}: When \texttt{x} is a \texttt{ml_pipeline}, the function returns a \texttt{ml_pipeline} with the predictor appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- tbl_spark, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_gbt_classifier(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()

Examples

```r
## Not run:
library(sparklyr)

sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
tcars_test <- partitions$test

# Specify the grid
family <- c("gaussian", "gamma", "poisson")
link <- c("identity", "log")
family_link <- expand.grid(family = family, link = link, stringsAsFactors = FALSE)
family_link <- data.frame(family_link, rmse = 0)

# Train the models
for (i in 1:nrow(family_link)) {
  glm_model <- mtcars_training %>%
    ml_generalized_linear_regression(mpg ~ .,
      family = family_link[i, 1],
      link = family_link[i, 2]
    )

  pred <- ml_predict(glm_model, mtcars_test)
  family_link[i, 3] <- ml_regression_evaluator(pred, label_col = "mpg")
}

family_link
```

## End(Not run)
Description

These methods summarize the results of Spark ML models into tidy forms.

Usage

```r
## S3 method for class 'ml_model_generalized_linear_regression'
tidy(x, exponentiate = FALSE, ...)

## S3 method for class 'ml_model_linear_regression'
tidy(x, ...)

## S3 method for class 'ml_model_generalized_linear_regression'
augment(
  x,
  newdata = NULL,
  type.residuals = c("working", "deviance", "pearson", "response"),
  ...
)

## S3 method for class 'ml_model_linear_regression'
augment(
  x,
  newdata = NULL,
  type.residuals = c("working", "deviance", "pearson", "response"),
  ...
)

## S3 method for class 'ml_model_generalized_linear_regression'
glance(x, ...)

## S3 method for class 'ml_model_linear_regression'
glance(x, ...)```

Arguments

- `x`: a Spark ML model.
- `exponentiate`: For GLM, whether to exponentiate the coefficient estimates (typical for logistic regression.)
- `...`: extra arguments (not used.)
- `newdata`: a tbl_spark of new data to use for prediction.
- `type.residuals`: type of residuals, defaults to "working". Must be set to "working" when newdata is supplied.
Details
The residuals attached by `augment` are of type "working" by default, which is different from the default of "deviance" for `residuals()` or `sdf_residuals()`.

ml_isotonic_regression

Spark ML – Isotonic Regression

Description
Currently implemented using parallelized pool adjacent violators algorithm. Only univariate (single feature) algorithm supported.

Usage

```r
ml_isotonic_regression(
  x, 
  formula = NULL, 
  feature_index = 0, 
  isotonic = TRUE, 
  weight_col = NULL, 
  features_col = "features", 
  label_col = "label", 
  prediction_col = "prediction", 
  uid = random_string("isotonic_regression_"), 
  ... 
)
```

Arguments

- **x** A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **formula** Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- **feature_index** Index of the feature if `features_col` is a vector column (default: 0), no effect otherwise.
- **isotonic** Whether the output sequence should be isotonic/increasing (true) or antitonic/decreasing (false). Default: true
- **weight_col** The name of the column to use as weights for the model fit.
- **features_col** Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- **label_col** Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`. 
prediction_col  Prediction column name.
uid                A character string used to uniquely identify the ML estimator.
...               Optional arguments; see Details.

Details

When \( x \) is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

Value

The object returned depends on the class of \( x \).

- **spark_connection**: When \( x \) is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- **ml_pipeline**: When \( x \) is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
- **tbl_spark**: When \( x \) is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- **tbl_spark**, with formula specified: When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()  

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
```
iris_test <- partitions$test

iso_res <- iris_tbl %>%
  ml_isotonic_regression(Petal_Length ~ Petal_Width)

pred <- ml_predict(iso_res, iris_test)
pred

## End(Not run)

---

**ml_isotonic_regression_tidiers**

*Tidying methods for Spark ML Isotonic Regression*

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```r
## S3 method for class 'ml_model_isotonic_regression'
tidy(x, ...)

## S3 method for class 'ml_model_isotonic_regression'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_isotonic_regression'
glance(x, ...)
```

**Arguments**

- `x` a Spark ML model.
- `...` extra arguments (not used.)
- `newdata` a tbl_spark of new data to use for prediction.

---

**ml_kmeans**

*Spark ML – K-Means Clustering*

**Description**

K-means clustering with support for k-means|| initialization proposed by Bahmani et al. Using `ml_kmeans()` with the formula interface requires Spark 2.0+. 
### Usage

```r
ml_kmeans(
  x,
  formula = NULL,
  k = 2,
  max_iter = 20,
  tol = 1e-04,
  init_steps = 2,
  init_mode = "k-means||",
  seed = NULL,
  features_col = "features",
  prediction_col = "prediction",
  uid = random_string("kmeans_"),
  ...
)

ml_compute_cost(model, dataset)
```

### Arguments

- **x**
  - A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **formula**
  - Used when `x` is a `tbl_spark`. A character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- **k**
  - The number of clusters to create.
- **max_iter**
  - The maximum number of iterations to use.
- **tol**
  - Param for the convergence tolerance for iterative algorithms.
- **init_steps**
  - Number of steps for the k-means|| initialization mode. This is an advanced setting – the default of 2 is almost always enough. Must be > 0. Default: 2.
- **init_mode**
  - Initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means++ (Bahmani et al., Scalable K-Means++, VLDB 2012). Default: k-means||.
- **seed**
  - A random seed. Set this value if you need your results to be reproducible across repeated calls.
- **features_col**
  - Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- **prediction_col**
  - Prediction column name.
- **uid**
  - A character string used to uniquely identify the ML estimator.
- **...**
  - Optional arguments, see Details.
- **model**
  - A fitted K-means model returned by `ml_kmeans()`.
- **dataset**
  - Dataset on which to calculate K-means cost.
Value

The object returned depends on the class of \( x \).

- `spark_connection`: When \( x \) is a `spark_connection`, the function returns an instance of a `ml_estimator` object. The object contains a pointer to a Spark Estimator object and can be used to compose Pipeline objects.
- `ml_pipeline`: When \( x \) is a `ml_pipeline`, the function returns a `ml_pipeline` with the clustering estimator appended to the pipeline.
- `tbl_spark`: When \( x \) is a `tbl_spark`, an estimator is constructed then immediately fit with the input `tbl_spark`, returning a clustering model.
- `tbl_spark`, with `formula` or `features` specified: When `formula` is specified, the input `tbl_spark` is first transformed using a `RFormula` transformer before being fit by the estimator. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`. This signature does not apply to `ml_lda()`.

`ml_compute_cost()` returns the K-means cost (sum of squared distances of points to their nearest center) for the model on the given data.

See Also

See [http://spark.apache.org/docs/latest/ml-clustering.html](http://spark.apache.org/docs/latest/ml-clustering.html) for more information on the set of clustering algorithms.

Other `ml` clustering algorithms: `ml_bisecting_kmeans()`, `ml_gaussian_mixture()`, `ml_lda()`

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
ml_kmeans(iris_tbl, Species ~ .)
## End(Not run)
```
Usage

```r
ml lda(
  x,
  formula = NULL,
  k = 10,
  max_iter = 20,
  doc_concentration = NULL,
  topic_concentration = NULL,
  subsampling_rate = 0.05,
  optimizer = "online",
  checkpoint_interval = 10,
  keep_last_checkpoint = TRUE,
  learning_decay = 0.51,
  learning_offset = 1024,
  optimize_doc_concentration = TRUE,
  seed = NULL,
  features_col = "features",
  topic_distribution_col = "topicDistribution",
  uid = random_string("lda_"),
  ...
)
```

```r
ml_describe_topics(model, max_terms_per_topic = 10)
ml_log_likelihood(model, dataset)
ml_log_perplexity(model, dataset)
ml_topics_matrix(model)
```

Arguments

- `x`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `formula`: Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- `k`: The number of clusters to create.
- `max_iter`: The maximum number of iterations to use.
- `doc_concentration`: Concentration parameter (commonly named "alpha") for the prior placed on documents' distributions over topics ("theta"). See details.
- `topic_concentration`: Concentration parameter (commonly named "beta" or "eta") for the prior placed on topics' distributions over terms.
- `subsampling_rate`: (For Online optimizer only) Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1]. Note that this
should be adjusted in sync with max_iter so the entire corpus is used. Specifically, set both so that maxIterations * miniBatchFraction greater than or equal to 1.

**optimizer**

Optimizer or inference algorithm used to estimate the LDA model. Supported: "online" for Online Variational Bayes (default) and "em" for Expectation-Maximization.

**checkpoint_interval**

Set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.

**keep_last_checkpoint**

(Spark 2.0.0+) (For EM optimizer only) If using checkpointing, this indicates whether to keep the last checkpoint. If FALSE, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care. Note that checkpoints will be cleaned up via reference counting, regardless.

**learning_decay**

(For Online optimizer only) Learning rate, set as an exponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence. This is called "kappa" in the Online LDA paper (Hoffman et al., 2010). Default: 0.51, based on Hoffman et al.

**learning_offset**

(For Online optimizer only) A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less. This is called "tau0" in the Online LDA paper (Hoffman et al., 2010) Default: 1024, following Hoffman et al.

**optimize_doc_concentration**

(For Online optimizer only) Indicates whether the doc_concentration (Dirichlet parameter for document-topic distribution) will be optimized during training. Setting this to true will make the model more expressive and fit the training data better. Default: FALSE

**seed**

A random seed. Set this value if you need your results to be reproducible across repeated calls.

**features_col**

Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.

**topic_distribution_col**

Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.

**uid**

A character string used to uniquely identify the ML estimator.

**...**

Optional arguments, see Details.

**model**

A fitted LDA model returned by `ml_lda()`.

**max_terms_per_topic**

Maximum number of terms to collect for each topic. Default value of 10.

**dataset**

test corpus to use for calculating log likelihood or log perplexity
ml_lda

Details

For `ml_lda.tbl_spark` with the formula interface, you can specify named arguments in `~` that will be passed `ft_regex_tokenizer()`, `ft_stop_words_remover()`, and `ft_count_vectorizer()`. For example, to increase the default `min_token_length`, you can use `ml_lda(dataset, ~ text, min_token_length = 4)`.

Terminology for LDA:
- "term" = "word": an element of the vocabulary
- "token": instance of a term appearing in a document
- "topic": multinomial distribution over terms representing some concept
- "document": one piece of text, corresponding to one row in the input data


Input data (`features_col`): LDA is given a collection of documents as input data, via the `features_col` parameter. Each document is specified as a Vector of length `vocab_size`, where each entry is the count for the corresponding term (word) in the document. Feature transformers such as `ft_tokenizer` and `ft_count_vectorizer` can be useful for converting text to word count vectors

Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a `spark_connection`, the function returns an instance of a `ml_estimator` object. The object contains a pointer to a Spark Estimator object and can be used to compose Pipeline objects.
- `ml_pipeline`: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the clustering estimator appended to the pipeline.
- `tbl_spark`: When `x` is a `tbl_spark`, an estimator is constructed then immediately fit with the input `tbl_spark`, returning a clustering model.
- `tbl_spark`, with `formula` or `features` specified: When `formula` is specified, the input `tbl_spark` is first transformed using a `RFormula` transformer before being fit by the estimator. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`. This signature does not apply to `ml_lda()`.

`ml_describe_topics` returns a DataFrame with topics and their top-weighted terms.

`ml_log_likelihood` calculates a lower bound on the log likelihood of the entire corpus

Parameter details

doc_concentration: This is the parameter to a Dirichlet distribution, where larger values mean more smoothing (more regularization). If not set by the user, then `doc_concentration` is set automatically. If set to singleton vector `[alpha]`, then alpha is replicated to a vector of length `k` in fitting. Otherwise, the `doc_concentration` vector must be length `k`. (default = automatic)

Optimizer-specific parameter settings:
- EM
• Currently only supports symmetric distributions, so all values in the vector should be the same.
• Values should be greater than 1.0
• default = uniformly \((50 / k) + 1\), where \(50/k\) is common in LDA libraries and +1 follows from Asuncion et al. (2009), who recommend a +1 adjustment for EM.

Online
• Values should be greater than or equal to 0
• default = uniformly \((1.0 / k)\), following the implementation from here

topic_concentration:
This is the parameter to a symmetric Dirichlet distribution.
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.
If not set by the user, then topic_concentration is set automatically. (default = automatic)

Optimizer-specific parameter settings:
EM
• Value should be greater than 1.0
• default = 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.

Online
• Value should be greater than or equal to 0
• default = \((1.0 / k)\), following the implementation from here.

topic_distribution_col:  This uses a variational approximation following Hoffman et al. (2010), where the approximate distribution is called "gamma." Technically, this method returns this approximation "gamma" for each document.

See Also
See [http://spark.apache.org/docs/latest/ml-clustering.html](http://spark.apache.org/docs/latest/ml-clustering.html) for more information on the set of clustering algorithms.

Other ml clustering algorithms: \texttt{ml_bisecting_kmeans()}, \texttt{ml_gaussian_mixture()}, \texttt{ml_kmeans()}

Examples
```r
## Not run:
library(janeaustenr)
library(dplyr)
sc <- spark_connect(master = "local")

lines_tbl <- sdf_copy_to(sc,
austen_books()[c(1:30), ],
name = "lines_tbl",
overwrite = TRUE
)

# transform the data in a tidy form
```
ml_lda_tidiers

Tidying methods for Spark ML LDA models

Description

These methods summarize the results of Spark ML models into tidy forms.

Usage

## S3 method for class 'ml_model_lda'
tidy(x, ...)

## S3 method for class 'ml_model_lda'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_lda'
glance(x, ...)

Arguments

x  a Spark ML model.

...  extra arguments (not used.)

newdata a tbl_spark of new data to use for prediction.
ml_linear_regression  

Spark ML – Linear Regression

Description
Perform regression using linear regression.

Usage

```r
ml_linear_regression(
  x,
  formula = NULL,
  fit_intercept = TRUE,
  elastic_net_param = 0,
  reg_param = 0,
  max_iter = 100,
  weight_col = NULL,
  loss = "squaredError",
  solver = "auto",
  standardization = TRUE,
  tol = 1e-06,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("linear_regression_"),
  ...)
```

Arguments

- **x** A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **formula** Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- **fit_intercept** Boolean; should the model be fit with an intercept term?
- **elastic_net_param** ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.
- **reg_param** Regularization parameter (aka lambda)
- **max_iter** The maximum number of iterations to use.
- **weight_col** The name of the column to use as weights for the model fit.
- **loss** The loss function to be optimized. Supported options: "squaredError" and "huber". Default: "squaredError"
- **solver** Solver algorithm for optimization.
ml_linear_regression

standardization

Whether to standardize the training features before fitting the model.

tol

Param for the convergence tolerance for iterative algorithms.

features_col

Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.

label_col

Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.

prediction_col

Prediction column name.

uid

A character string used to uniquely identify the ML estimator.

... Optional arguments; see Details.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
- tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- tbl_spark, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()
Examples

```r
## Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test

lm_model <- mtcars_training %>%
  ml_linear_regression(mpg ~ .)

pred <- ml_predict(lm_model, mtcars_test)
ml_regression_evaluator(pred, label_col = "mpg")

## End(Not run)
```

ml_linear_svc
Spark ML – LinearSVC

Description

Perform classification using linear support vector machines (SVM). This binary classifier optimizes the Hinge Loss using the OWLQN optimizer. Only supports L2 regularization currently.

Usage

```r
ml_linear_svc(
x,  
formula = NULL,
fit_intercept = TRUE,
reg_param = 0,
max_iter = 100,
standardization = TRUE,
weight_col = NULL,
tol = 1e-06,
threshold = 0,
aggregation_depth = 2,
features_col = "features",
label_col = "label",
prediction_col = "prediction",
raw_prediction_col = "rawPrediction",
uid = random_string("linear_svc_"),
...  
)
```
ml_linear_svc

Arguments

x          A spark_connection, ml_pipeline, or a tbl_spark.
formula    Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.
fit_intercept  Boolean; should the model be fit with an intercept term?
reg_param    Regularization parameter (aka lambda)
max_iter    The maximum number of iterations to use.
standardization  Whether to standardize the training features before fitting the model.
weight_col    The name of the column to use as weights for the model fit.
tol         Param for the convergence tolerance for iterative algorithms.
threshold    in binary classification prediction, in range [0, 1].
aggregation_depth (Spark 2.1.0+) Suggested depth for treeAggregate (>= 2).
features_col Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.
label_col    Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.
prediction_col Prediction column name.
raw_prediction_col Raw prediction (a.k.a. confidence) column name.
uid          A character string used to uniquely identify the ML estimator.
...          Optional arguments; see Details.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

Value

The object returned depends on the class of x.

- spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimater object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
• **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the predictor appended to the pipeline.

• **tbl_spark**: When `x` is a `tbl_spark`, a predictor is constructed then immediately fit with the input `tbl_spark`, returning a prediction model.

• **tbl_spark, with `formula` specified**: When `formula` is specified, the input `tbl_spark` is first transformed using a `RFormula` transformer before being fit by the predictor. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`.

**See Also**

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

**Examples**

```r
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  filter(Species != "setosa") %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

tab <- iris_training %>%
  ml_linear_svc(Species ~ .)
pred <- ml_predict(tab, iris_test)
ml_binary_classification_evaluator(pred)

## End(Not run)
```

---

**Description**

These methods summarize the results of Spark ML models into tidy forms.
Usage

```r
## S3 method for class 'ml_model_linear_svc'
tidy(x, ...)

## S3 method for class 'ml_model_linear_svc'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_linear_svc'
glance(x, ...)
```

Arguments

- `x` a Spark ML model.
- `...` extra arguments (not used.)
- `newdata` a tbl_spark of new data to use for prediction.

Description

Perform classification using logistic regression.

Usage

```r
ml_logistic_regression(x,
  formula = NULL,
  fit_intercept = TRUE,
  elastic_net_param = 0,
  reg_param = 0,
  max_iter = 100,
  threshold = 0.5,
  thresholds = NULL,
  tol = 1e-06,
  weight_col = NULL,
  aggregation_depth = 2,
  lower_bounds_on_coefficients = NULL,
  lower_bounds_on_intercepts = NULL,
  upper_bounds_on_coefficients = NULL,
  upper_bounds_on_intercepts = NULL,
  features_col = "features",
  label_col = "label",
  family = "auto",
  prediction_col = "prediction",
)```
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("logistic_regression_"),
...
)

Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.
formula Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.
fit_intercept Boolean; should the model be fit with an intercept term?
elastic_net_param ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.
reg_param Regularization parameter (aka lambda)
max_iter The maximum number of iterations to use.
threshold in binary classification prediction, in range [0, 1].
thresholds Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value \( p/t \) is predicted, where \( p \) is the original probability of that class and \( t \) is the class’s threshold.
tol Param for the convergence tolerance for iterative algorithms.
weight_col The name of the column to use as weights for the model fit.
aggregation_depth (Spark 2.1.0+) Suggested depth for treeAggregate (>= 2).
lower_bounds_on_coefficients (Spark 2.2.0+) Lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.
lower_bounds_on_intercepts (Spark 2.2.0+) Lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.
upper_bounds_on_coefficients (Spark 2.2.0+) Upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.
upper_bounds_on_intercepts (Spark 2.2.0+) Upper bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.
features_col  Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.

label_col  Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.

family  (Spark 2.1.0+) Param for the name of family which is a description of the label distribution to be used in the model. Supported options: "auto", "binomial", and "multinomial."

prediction_col  Prediction column name.

probability_col  Column name for predicted class conditional probabilities.

raw_prediction_col  Raw prediction (a.k.a. confidence) column name.

uid  A character string used to uniquely identify the ML estimator.

...  Optional arguments; see Details.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

Value

The object returned depends on the class of x.

• spark_connection: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.

• ml_pipeline: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.

• tbl_spark: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.

• tbl_spark, with formula specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.
## ml_logistic_regression_tidiers

Tidying methods for Spark ML Logistic Regression

### Description

These methods summarize the results of Spark ML models into tidy forms.

### Usage

**## S3 method for class 'ml_model_logistic_regression'**

tidy(x, ...)

**## S3 method for class 'ml_model_logistic_regression'**
augment(x, newdata = NULL, ...)

**## S3 method for class 'ml_model_logistic_regression'**
glance(x, ...)

---

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test

lr_model <- mtcars_training %>%
  ml_logistic_regression(am ~ gear + carb)

pred <- ml_predict(lr_model, mtcars_test)
ml_binary_classification_evaluator(pred)

## End(Not run)
```
**ml_model_data**

Extracts data associated with a Spark ML model.

**Arguments**

- `x`: a Spark ML model.
- `...`: extra arguments (not used.)
- `newdata`: a tbl_spark of new data to use for prediction.

**Value**

A tbl_spark

---

**ml_multilayer_perceptron_classifier**

*Spark ML – Multilayer Perceptron*

Classification model based on the Multilayer Perceptron. Each layer has sigmoid activation function, output layer has softmax.

**Usage**

```r
ml_multilayer_perceptron_classifier(
  x,
  formula = NULL,
  layers = NULL,
  max_iter = 100,
  step_size = 0.03,
  tol = 1e-06,
  block_size = 128,
  solver = "l-bfgs",
  seed = NULL,
)```

---
initial_weights = NULL,
thresholds = NULL,
features_col = "features",
label_col = "label",
prediction_col = "prediction",
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("multilayer_perceptron_classifier_"),
...}

ml_multilayer_perceptron(
x,  
formula = NULL,
layers,  
max_iter = 100,  
step_size = 0.03,  
tol = 1e-06,  
block_size = 128,  
solver = "l-bfgs",  
seed = NULL,  
initial_weights = NULL,  
features_col = "features",  
label_col = "label",  
thresholds = NULL,  
prediction_col = "prediction",  
probability_col = "probability",  
raw_prediction_col = "rawPrediction",  
uid = random_string("multilayer_perceptron_classifier_"),  
response = NULL,  
features = NULL,  
...}

Arguments

x A spark_connection, ml_pipeline, or a tbl_spark.
formula Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.
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.
max_iter The maximum number of iterations to use.
step_size Step size to be used for each iteration of optimization (> 0).
tol Param for the convergence tolerance for iterative algorithms.
block_size Block size for stacking input data in matrices to speed up the computation. Data is stacked within partitions. If block size is more than remaining data in a partition then it is adjusted to the size of this data. Recommended size is between 10 and 1000. Default: 128

solver The solver algorithm for optimization. Supported options: "gd" (minibatch gradient descent) or "l-bfgs". Default: "l-bfgs"

seed A random seed. Set this value if you need your results to be reproducible across repeated calls.

initial_weights The initial weights of the model.

thresholds Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class’s threshold.

features_col Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by ft_r_formula.

label_col Label column name. The column should be a numeric column. Usually this column is output by ft_r_formula.

prediction_col Prediction column name.

probability_col Column name for predicted class conditional probabilities.

raw_prediction_col Raw prediction (a.k.a. confidence) column name.

uid A character string used to uniquely identify the ML estimator.

... Optional arguments; see Details.

response (Deprecated) The name of the response column (as a length-one character vector.)

features (Deprecated) The name of features (terms) to use for the model fit.

Details

When x is a tbl_spark and formula (alternatively, response and features) is specified, the function returns a ml_model object wrapping a ml_pipeline_model which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument predicted_label_col (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted ml_pipeline_model, ml_model objects also contain a ml_pipeline object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by ml_save with type = "pipeline" to facilitate model refresh workflows.

ml_multilayer_perceptron() is an alias for ml_multilayer_perceptron_classifier() for backwards compatibility.
Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.

- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.

- **tbl_spark**: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.

- **tbl_spark**, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_naive_bayes(), ml_one_vs_rest(), ml_random_forest_classifier()

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)
iris_training <- partitions$training
iris_test <- partitions$test

mlp_model <- iris_training %>%
  ml_multilayer_perceptron_classifier(Species ~ ., layers = c(4,3,3))
pred <- ml_predict(mlp_model, iris_test)
ml_multiclass_classification_evaluator(pred)
## End(Not run)
```
ml_multilayer_perceptron_tidiers

Tidying methods for Spark ML MLP

Description

These methods summarize the results of Spark ML models into tidy forms.

Usage

## S3 method for class `ml_model_multilayer_perceptron_classification`
tidy(x, ...)

## S3 method for class `ml_model_multilayer_perceptron_classification`
augment(x, newdata = NULL, ...)

## S3 method for class `ml_model_multilayer_perceptron_classification`
glance(x, ...)

Arguments

x a Spark ML model.

... extra arguments (not used.)

newdata a tbl_spark of new data to use for prediction.

ml_naive_bayes Spark ML – Naive-Bayes

Description

Naive Bayes Classifiers. It supports Multinomial NB (see here) which can handle finitely supported discrete data. For example, by converting documents into TF-IDF vectors, it can be used for document classification. By making every vector a binary (0/1) data, it can also be used as Bernoulli NB (see here). The input feature values must be nonnegative.

Usage

ml_naive_bayes(
x,
  formula = NULL,
  model_type = "multinomial",
  smoothing = 1,
  thresholds = NULL,
  weight_col = NULL,
  features_col = "features",
)
```r
label_col = "label",
prediction_col = "prediction",
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("naive_bayes_"),
...
)
```

**Arguments**

- `x` A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `formula` Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- `model_type` The model type. Supported options: "multinomial" and "bernoulli". (default = multinomial)
- `smoothing` The (Laplace) smoothing parameter. Defaults to 1.
- `thresholds` Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value \( p/t \) is predicted, where \( p \) is the original probability of that class and \( t \) is the class’s threshold.
- `weight_col` (Spark 2.1.0+) Weight column name. If this is not set or empty, we treat all instance weights as 1.0.
- `features_col` Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- `label_col` Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`.
- `prediction_col` Prediction column name.
- `probability_col` Column name for predicted class conditional probabilities.
- `raw_prediction_col` Raw prediction (a.k.a. confidence) column name.
- `uid` A character string used to uniquely identify the ML estimator.
- `...` Optional arguments; see Details.

**Details**

When `x` is a `tbl_spark` and `formula` (alternatively, `response` and `features`) is specified, the function returns a `ml_model` object wrapping a `ml_pipeline_model` which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument `predicted_label_col` (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted `ml_pipeline_model`, `ml_model` objects also contain a `ml_pipeline` object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by `ml_save` with type = "pipeline" to facilitate model refresh workflows.
Value

The object returned depends on the class of x.

- **spark_connection**: When x is a spark_connection, the function returns an instance of a ml_estimator object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- **ml_pipeline**: When x is a ml_pipeline, the function returns a ml_pipeline with the predictor appended to the pipeline.
- **tbl_spark**: When x is a tbl_spark, a predictor is constructed then immediately fit with the input tbl_spark, returning a prediction model.
- **tbl_spark**, with formula: specified When formula is specified, the input tbl_spark is first transformed using a RFormula transformer before being fit by the predictor. The object returned in this case is a ml_model which is a wrapper of a ml_pipeline_model.

See Also

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

Examples

```r
## Not run:
sd <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sdf, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

nb_model <- iris_training %>%
  ml_naive_bayes(Species ~ .)

pred <- ml_predict(nb_model, iris_test)

ml_multiclass_classification_evaluator(pred)
## End(Not run)
```
**ml_naive_bayes_tidiers**

*Tidying methods for Spark ML Naive Bayes*

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```r
## S3 method for class 'ml_model_naive_bayes'
tidy(x, ...)

## S3 method for class 'ml_model_naive_bayes'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_naive_bayes'
glance(x, ...)
```

**Arguments**

- **x**: a Spark ML model.
- **...**: extra arguments (not used.)
- **newdata**: a tbl_spark of new data to use for prediction.

---

**ml_one_vs_rest**

*Spark ML – OneVsRest*

**Description**

Reduction of Multiclass Classification to Binary Classification. Performs reduction using one against all strategy. For a multiclass classification with k classes, train k models (one per class). Each example is scored against all k models and the model with highest score is picked to label the example.

**Usage**

```r
ml_one_vs_rest(
  x,
  formula = NULL,
  classifier = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("one_vs_rest_"),
  ...
)
```
Arguments

- **x**: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- **formula**: Used when `x` is a `tbl_spark`. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see `ft_r_formula` for details.
- **classifier**: Object of class `ml_estimator`. Base binary classifier that we reduce multiclass classification into.
- **features_col**: Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.
- **label_col**: Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`.
- **prediction_col**: Prediction column name.
- **uid**: A character string used to uniquely identify the ML estimator.
- **...**: Optional arguments; see Details.

Details

When `x` is a `tbl_spark` and `formula` (alternatively, `response` and `features`) is specified, the function returns a `ml_model` object wrapping a `ml_pipeline_model` which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument `predicted_label_col` (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted `ml_pipeline_model`, `ml_model` objects also contain a `ml_pipeline` object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by `ml_save` with type = "pipeline" to facilitate model refresh workflows.

Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns an instance of a `ml_estimator` object. The object contains a pointer to a Spark Predictor object and can be used to compose Pipeline objects.
- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the predictor appended to the pipeline.
- **tbl_spark**: When `x` is a `tbl_spark`, a predictor is constructed then immediately fit with the input `tbl_spark`, returning a prediction model.
- **tbl_spark`, with `formula` specified**: When `formula` is specified, the input `tbl_spark` is first transformed using a `RFormula` transformer before being fit by the predictor. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`.
ml_pipeline

See Also

See http://spark.apache.org/docs/latest/ml-classification-regression.html for more information on the set of supervised learning algorithms.

Other ml algorithms: ml_aft_survival_regression(), ml_decision_tree_classifier(), ml_gbt_classifier(), ml_generalized_linear_regression(), ml_isotonic_regression(), ml_linear_regression(), ml_linear_svc(), ml_logistic_regression(), ml_multilayer_perceptron_classifier(), ml_naive_bayes(), ml_random_forest_classifier()

---

ml_pca_tidiers  

Tidying methods for Spark ML Principal Component Analysis

Description

These methods summarize the results of Spark ML models into tidy forms.

Usage

## S3 method for class 'ml_model_pca'
tidy(x, ...)

## S3 method for class 'ml_model_pca'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_pca'
glance(x, ...)

Arguments

x  
a Spark ML model.
...
  extra arguments (not used.)
newdata  
a tbl_spark of new data to use for prediction.

---

ml_pipeline  

Spark ML – Pipelines

Description

Create Spark ML Pipelines

Usage

ml_pipeline(x, ..., uid = random_string("pipeline_"))
Arguments

- **x**: Either a spark_connection or ml_pipeline_stage objects.
- **...**: ml_pipeline_stage objects.
- **uid**: A character string used to uniquely identify the ML estimator.

Value

When `x` is a spark_connection, `ml_pipeline()` returns an empty pipeline object. When `x` is a ml_pipeline_stage, `ml_pipeline()` returns an ml_pipeline with the stages set to `x` and any transformers or estimators given in ....

### Description

Perform classification and regression using random forests.

### Usage

```r
ml_random_forest_classifier(
  x,
  formula = NULL,
  num_trees = 20,
  subsampling_rate = 1,
  max_depth = 5,
  min_instances_per_node = 1,
  feature_subset_strategy = "auto",
  impurity = "gini",
  min_info_gain = 0,
  max_bins = 32,
  seed = NULL,
  thresholds = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  uid = random_string("random_forest_classifier_"),
  ...
)
```
ml_random_forest_classifier

ml_random_forest(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  feature_subset_strategy = "auto",
  impurity = "auto",
  checkpoint_interval = 10,
  max_bins = 32,
  max_depth = 5,
  num_trees = 20,
  min_info_gain = 0,
  min_instances_per_node = 1,
  subsampling_rate = 1,
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  uid = random_string("random_forest_"),
  response = NULL,
  features = NULL,
  ...
)

ml_random_forest_regressor(
  x,
  formula = NULL,
  num_trees = 20,
  subsampling_rate = 1,
  max_depth = 5,
  min_instances_per_node = 1,
  feature_subset_strategy = "auto",
  impurity = "variance",
  min_info_gain = 0,
  max_bins = 32,
  seed = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  uid = random_string("random_forest_regressor_"),
  ...
)
Arguments

\( x \)  A spark_connection, ml_pipeline, or a tbl_spark.

\( \text{formula} \)  Used when \( x \) is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see ft_r_formula for details.

\( \text{num_trees} \)  Number of trees to train (>= 1). If 1, then no bootstrapping is used. If > 1, then bootstrapping is done.

\( \text{subsampling_rate} \)  Fraction of the training data used for learning each decision tree, in range (0, 1]. (default = 1.0)

\( \text{max_depth} \)  Maximum depth of the tree (>= 0); that is, the maximum number of nodes separating any leaves from the root of the tree.

\( \text{min_instances_per_node} \)  Minimum number of instances each child must have after split.

\( \text{feature_subset_strategy} \)  The number of features to consider for splits at each tree node. See details for options.

\( \text{impurity} \)  Criterion used for information gain calculation. Supported: "entropy" and "gini" (default) for classification and "variance" (default) for regression. For ml_decision_tree, setting "auto" will default to the appropriate criterion based on model type.

\( \text{min_info_gain} \)  Minimum information gain for a split to be considered at a tree node. Should be >= 0, defaults to 0.

\( \text{max_bins} \)  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.

\( \text{seed} \)  Seed for random numbers.

\( \text{thresholds} \)  Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value \( p/t \) is predicted, where \( p \) is the original probability of that class and \( t \) is the class’s threshold.

\( \text{checkpoint_interval} \)  Set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.

\( \text{cache_node_ids} \)  If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.

\( \text{max_memory_in_mb} \)  Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.
features_col  Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by `ft_r_formula`.

label_col  Label column name. The column should be a numeric column. Usually this column is output by `ft_r_formula`.

prediction_col  Prediction column name.

probability_col  Column name for predicted class conditional probabilities.

raw_prediction_col  Raw prediction (a.k.a. confidence) column name.

uid  A character string used to uniquely identify the ML estimator.

...  Optional arguments; see Details.

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.

response  (Deprecated) The name of the response column (as a length-one character vector.)

features  (Deprecated) The name of features (terms) to use for the model fit.

Details

When `x` is a `tbl_spark` and `formula` (alternatively, `response` and `features`) is specified, the function returns a `ml_model` object wrapping a `ml_pipeline_model` which contains data pre-processing transformers, the ML predictor, and, for classification models, a post-processing transformer that converts predictions into class labels. For classification, an optional argument `predicted_label_col` (defaults to "predicted_label") can be used to specify the name of the predicted label column. In addition to the fitted `ml_pipeline_model`, `ml_model` objects also contain a `ml_pipeline` object where the ML predictor stage is an estimator ready to be fit against data. This is utilized by `ml_save` with `type = "pipeline"` to facilitate model refresh workflows.

The supported options for `feature_subset_strategy` are

- "auto": Choose automatically for task: If `num_trees == 1`, set to "all". If `num_trees > 1` (forest), set to "sqrt" for classification and to "onethird" for regression.
- "all": use all features
- "onethird": use 1/3 of the features
- "sqrt": use use sqrt(number of features)
- "log2": use log2(number of features)
- "n": when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features. (default = "auto")

`ml_random_forest` is a wrapper around `ml_random_forest_regressor.tbl_spark` and `ml_random_forest_classifier.tbl_spark` and calls the appropriate method based on model type.
### Value

The object returned depends on the class of `x`.

- **spark_connection**: When `x` is a `spark_connection`, the function returns an instance of a `ml_estimator` object. The object contains a pointer to a Spark Predictor object and can be used to compose `Pipeline` objects.

- **ml_pipeline**: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the predictor appended to the pipeline.

- **tbl_spark**: When `x` is a `tbl_spark`, a predictor is constructed then immediately fit with the input `tbl_spark`, returning a prediction model.

- **tbl_spark**, with `formula` specified: When `formula` is specified, the input `tbl_spark` is first transformed using a `RFormula` transformer before being fit by the predictor. The object returned in this case is a `ml_model` which is a wrapper of a `ml_pipeline_model`.

### See Also

See [http://spark.apache.org/docs/latest/ml-classification-regression.html](http://spark.apache.org/docs/latest/ml-classification-regression.html) for more information on the set of supervised learning algorithms.

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`

### Examples

```r
## Not run:
sd <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

rf_model <- iris_training %>%
  ml_random_forest(Species ~ ., type = "classification")

pred <- ml_predict(rf_model, iris_test)
ml_multiclass_classification_evaluator(pred)
## End(Not run)
```
**ml_stage**

*Spark ML – Pipeline stage extraction*

**Description**

Extraction of stages from a Pipeline or PipelineModel object.

**Usage**

```r
ml_stage(x, stage)
ml_stages(x, stages = NULL)
```

**Arguments**

- `x`: A `ml_pipeline` or a `ml_pipeline_model` object.
- `stage`: The UID of a stage in the pipeline.
- `stages`: The UIDs of stages in the pipeline as a character vector.

**Value**

- For `ml_stage()`: The stage specified.
- For `ml_stages()`: A list of stages. If `stages` is not set, the function returns all stages of the pipeline in a list.

---

**ml_summary**

*Spark ML – Extraction of summary metrics*

**Description**

Extracts a metric from the summary object of a Spark ML model.

**Usage**

```r
ml_summary(x, metric = NULL, allow_null = FALSE)
```

**Arguments**

- `x`: A Spark ML model that has a summary.
- `metric`: The name of the metric to extract. If not set, returns the summary object.
- `allow_null`: Whether null results are allowed when the metric is not found in the summary.
**Description**
These methods summarize the results of Spark ML models into tidy forms.

**Usage**
```r
## S3 method for class 'ml_model_aft_survival_regression'
tidy(x, ...)
```
```r
## S3 method for class 'ml_model_aft_survival_regression'
augment(x, newdata = NULL, ...)
```
```r
## S3 method for class 'ml_model_aft_survival_regression'
glance(x, ...)
```

**Arguments**
- `x` a Spark ML model.
- `...` extra arguments (not used.)
- `newdata` a tbl_spark of new data to use for prediction.

**Description**
These methods summarize the results of Spark ML models into tidy forms.

**Usage**
```r
## S3 method for class 'ml_model_decision_tree_classification'
tidy(x, ...)
```
```r
## S3 method for class 'ml_model_decision_tree_regression'
tidy(x, ...)
```
```r
## S3 method for class 'ml_model_decision_tree_classification'
augment(x, newdata = NULL, ...)
```
```r
## S3 method for class 'ml_model_decision_tree_regression'
augment(x, newdata = NULL, ...)
```
Arguments

- x: a Spark ML model.
- ...: extra arguments (not used.)
- newdata: a tbl_spark of new data to use for prediction.
ml_uid

Spark ML – UID

Description

Extracts the UID of an ML object.

Usage

ml_uid(x)

Arguments

x	A Spark ML object

ml_unsupervised_tidiers

Tidying methods for Spark ML unsupervised models

Description

These methods summarize the results of Spark ML models into tidy forms.

Usage

## S3 method for class 'ml_model_kmeans'
tidy(x, ...)

## S3 method for class 'ml_model_kmeans'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_kmeans'
glance(x, ...)

## S3 method for class 'ml_model_bisecting_kmeans'
tidy(x, ...)

## S3 method for class 'ml_model_bisecting_kmeans'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_bisecting_kmeans'
glance(x, ...)

## S3 method for class 'ml_model_gaussian_mixture'
tidy(x, ...)

## S3 method for class 'ml_model_gaussian_mixture'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_gaussian_mixture'
glance(x, ...)

### Arguments

- **x**: a Spark ML model.
- **...**: extra arguments (not used.)
- **newdata**: a tbl_spark of new data to use for prediction.

---

### na.replace

**Replace Missing Values in Objects**

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

**Usage**

```r
ga.replace(object, ...)
```

**Arguments**

- **object**: An **R** object.
- **...**: Arguments passed along to implementing methods.

---

### random_string

**Random string generation**

Generate a random string with a given prefix.

**Usage**

```r
random_string(prefix = "table")
```

**Arguments**

- **prefix**: A length-one character vector.
**reactiveSpark**  
*Reactive spark reader*

**Description**

Given a spark object, returns a reactive data source for the contents of the spark object. This function is most useful to read Spark streams.

**Usage**

`reactiveSpark(x, intervalMillis = 1000, session = NULL)`

**Arguments**

- `x`: An object coercable to a Spark DataFrame.
- `intervalMillis`: Approximate number of milliseconds to wait to retrieve updated data frame. This can be a numeric value, or a function that returns a numeric value.
- `session`: The user session to associate this file reader with, or NULL if none. If non-null, the reader will automatically stop when the session ends.

**registerDoSpark**  
*Register a Parallel Backend*

**Description**

Registers a parallel backend using the foreach package.

**Usage**

`registerDoSpark(spark_conn, ...)`

**Arguments**

- `spark_conn`: spark connection to use
- `...`: additional options for sparklyr parallel backend (currently only the only valid option is nocompile = T, F)

**Value**

None
Examples

```r
## Not run:

sc <- spark_connect(master = "local")
registerDoSpark(sc, nocompile = FALSE)

## End(Not run)
```

---

### 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**

```r
register_extension(package)
```

```r
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**

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

```r
sdf_load_table(sc, name)
```

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

```r
sdf_load_parquet(sc, path)
```
Arguments

x A spark_connection, ml_pipeline, or 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.

Description

Deprecated methods for transformation, fit, and prediction. These are mirrors of the corresponding ml-transform-methods.

Usage

sdf_predict(x, model, ...)
sdf_transform(x, transformer, ...)
sdf_fit(x, estimator, ...)
sdf_fit_and_transform(x, estimator, ...)

Arguments

x A tbl_spark.
model A ml_transformer or a ml_model object.
... Optional arguments passed to the corresponding ml_ methods.
transformer A ml_transformer object.
estimator A ml_estimator object.

Value

sdf_predict(), sdf_transform(), and sdf_fit_and_transform() return a transformed dataframe whereas sdf_fit() returns a ml_transformer.
sdf_along

Create DataFrame for along Object

Description

Creates a DataFrame along the given object.

Usage

sdf_along(sc, along, repartition = NULL, type = c("integer", "integer64"))

Arguments

sc
The associated Spark connection.

along
Takes the length from the length of this argument.

repartition
The number of partitions to use when distributing the data across the Spark cluster.

type
The data type to use for the index, either "integer" or "integer64".

sdf_bind

Bind multiple Spark DataFrames by row and column

Description

sdf_bind_rows() and sdf_bind_cols() are implementation of the common pattern of do.call(rbind,sdfs) or do.call(cbind,sdfs) for binding many Spark DataFrames into one.

Usage

sdf_bind_rows(..., id = NULL)

sdf_bind_cols(...)

Arguments

...
Spark tbls to combine.
Each argument can either be a Spark DataFrame or a list of Spark DataFrames
When row-binding, columns are matched by name, and any missing columns
will be filled with NA.
When column-binding, rows are matched by position, so all data frames must
have the same number of rows.

id
Data frame identifier.
When id is supplied, a new column of identifiers is created to link each row to
its original Spark DataFrame. The labels are taken from the named arguments
to sdf_bind_rows(). When a list of Spark DataFrames is supplied, the labels
are taken from the names of the list. If no names are found a numeric sequence
is used instead.
**Details**

The output of `sdf_bind_rows()` will contain a column if that column appears in any of the inputs.

**Value**

`sdf_bind_rows()` and `sdf_bind_cols()` return `tbl_spark`

---

<table>
<thead>
<tr>
<th>sdf_broadcast</th>
<th>Broadcast hint</th>
</tr>
</thead>
</table>

**Description**

Used to force broadcast hash joins.

**Usage**

`sdf_broadcast(x)`

**Arguments**

- `x` A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.

---

<table>
<thead>
<tr>
<th>sdf_checkpoint</th>
<th>Checkpoint a Spark DataFrame</th>
</tr>
</thead>
</table>

**Description**

Checkpoint a Spark DataFrame

**Usage**

`sdf_checkpoint(x, eager = TRUE)`

**Arguments**

- `x` an object coercible to a Spark DataFrame
- `eager` whether to truncate the lineage of the DataFrame
**sdf_coalesce**  
*Coalesces a Spark DataFrame*

**Description**
Coalesces a Spark DataFrame

**Usage**
```
sdf_coalesce(x, partitions)
```

**Arguments**
- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **partitions**: number of partitions

---

**sdf_collect**  
*Collect a Spark DataFrame into R.*

**Description**
Collects a Spark dataframe into R.

**Usage**
```
sdf_collect(object, ...)
```

**Arguments**
- **object**: Spark dataframe to collect
- **...**: Additional options.
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, struct_columns, ...)

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

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?

struct_columns
(only supported with Spark 2.4.0 or higher) A list of columns from the source data frame that should be converted to Spark SQL StructType columns. The source columns can contain either json strings or nested lists. All rows within each source column should have identical schemas (because otherwise the conversion result will contain unexpected null values or missing values as Spark currently does not support schema discovery on individual rows within a struct column).

...
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_random_split(), sdf_register(), sdf_sample(), sdf_sort()
Examples

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

---

**sdf_crosstab**  
*Cross Tabulation*

**Description**

Builds a contingency table at each combination of factor levels.

**Usage**

```r
sdf_crosstab(x, col1, col2)
```

**Arguments**

- `x`: A Spark DataFrame
- `col1`: The name of the first column. Distinct items will make the first item of each row.
- `col2`: The name of the second column. Distinct items will make the column names of the DataFrame.

**Value**

A DataFrame containing the contingency table.

---

**sdf_debug_string**  
*Debug Info for Spark DataFrame*

**Description**

Prints plan of execution to generate `x`. This plan will, among other things, show the number of partitions in parenthesis at the far left and indicate stages using indentation.

**Usage**

```r
sdf_debug_string(x, print = TRUE)
```

**Arguments**

- `x`: An R object wrapping, or containing, a Spark DataFrame.
- `print`: Print debug information?
**sdf_describe**  
*Compute summary statistics for columns of a data frame*

**Description**

Compute summary statistics for columns of a data frame

**Usage**

```r
sdf_describe(x, cols = colnames(x))
```

**Arguments**

- `x`: An object coercible to a Spark DataFrame
- `cols`: Columns to compute statistics for, given as a character vector

---

**sdf_dim**  
*Support for Dimension Operations*

**Description**

`sdf_dim()`, `sdf_nrow()` and `sdf_ncol()` provide similar functionality to `dim()`, `nrow()` and `ncol()`.

**Usage**

```r
sdf_dim(x)
sdf_nrow(x)
sdf_ncol(x)
```

**Arguments**

- `x`: An object (usually a `spark_tbl`).
sdf_drop_duplicates  \textit{Remove duplicates from a Spark DataFrame}

\textbf{Description}
Remove duplicates from a Spark DataFrame

\textbf{Usage}
\begin{verbatim}
sdf_drop_duplicates(x, cols = NULL)
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \textbf{x}  An object coercible to a Spark DataFrame
\item \textbf{cols}  Subset of Columns to consider, given as a character vector
\end{itemize}

sdf_is_streaming  \textit{Spark DataFrame is Streaming}

\textbf{Description}
Is the given Spark DataFrame a streaming data?

\textbf{Usage}
\begin{verbatim}
sdf_is_streaming(x)
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \textbf{x}  A \texttt{spark_connection}, \texttt{ml_pipeline}, or a \texttt{tbl_spark}.
\end{itemize}

sdf_last_index  \textit{Returns the last index of a Spark DataFrame}

\textbf{Description}
Returns the last index of a Spark DataFrame. The Spark mapPartitionsWithIndex function is used to iterate through the last nonempty partition of the RDD to find the last record.

\textbf{Usage}
\begin{verbatim}
sdf_last_index(x, id = "id")
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \textbf{x}  A \texttt{spark_connection}, \texttt{ml_pipeline}, or a \texttt{tbl_spark}.
\item \textbf{id}  The name of the index column.
\end{itemize}
**sdf_len**

Create DataFrame for Length

**Description**

Creates a DataFrame for the given length.

**Usage**

```r
sdf_len(sc, length, repartition = NULL, type = c("integer", "integer64"))
```

**Arguments**

- `sc`: The associated Spark connection.
- `length`: The desired length of the sequence.
- `repartition`: The number of partitions to use when distributing the data across the Spark cluster.
- `type`: The data type to use for the index, either "integer" or "integer64".

---

**sdf_num_partitions**

Gets number of partitions of a Spark DataFrame

**Description**

Gets number of partitions of a Spark DataFrame

**Usage**

```r
sdf_num_partitions(x)
```

**Arguments**

- `x`: A spark_connection, ml_pipeline, or a tbl_spark.
### sdf_persist

**Persist a Spark DataFrame**

**Description**

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

**Usage**

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

**Arguments**

- `x`: A `spark_connection`, `ml_pipeline`, or 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_pivot

**Pivot a Spark DataFrame**

**Description**

Construct a pivot table over a Spark Dataframe, using a syntax similar to that from `reshape2::dcast`.

**Usage**

```r
sdf_pivot(x, formula, fun.aggregate = "count")
```
Arguments

x  A spark_connection, ml_pipeline, or a tbl_spark.
formula A two-sided R formula of the form x_1 + x_2 + ... ~ y_1. The left-hand side of the formula indicates which variables are used for grouping, and the right-hand side indicates which variable is used for pivoting. Currently, only a single pivot column is supported.
fun.aggregate How should the grouped dataset be aggregated? Can be a length-one character vector, giving the name of a Spark aggregation function to be called; a named R list mapping column names to an aggregation method, or an R function that is invoked on the grouped dataset.

Examples

## Not run:
library(sparklyr)
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

# aggregating by mean
iris_tbl %>%
  mutate(Petal_Width = ifelse(Petal_Width > 1.5, "High", "Low" )) %>%
  sdf_pivot(Petal_Width ~ Species,
            fun.aggregate = list(Petal_Length = "mean"))

# aggregating all observations in a list
iris_tbl %>%
  mutate(Petal_Width = ifelse(Petal_Width > 1.5, "High", "Low" )) %>%
  sdf_pivot(Petal_Width ~ Species,
            fun.aggregate = list(Petal_Length = "collect_list"))

## End(Not run)
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.

`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

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

Arguments

- **x**: A `spark_connection`, `ml_pipeline`, or 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_random_split**  
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**

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

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,

```r
sdf_random_split(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_register()`, `sdf_sample()`, `sdf_sort()`

Examples

```r
## 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_random_split(training = 0.6, test = 0.4)
print(partitions)

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

## End(Not run)
```

---

`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`: A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `column`: The name of a column within `x`.

Details

It is expected for this operation to preserve row order.
### 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**

```r
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_random_split()`, `sdf_sample()`, `sdf_sort()`

### sdf_repartition

**Repartition a Spark DataFrame**

**Description**

Repartition a Spark DataFrame

**Usage**

```r
sdf_repartition(x, partitions = NULL, partition_by = NULL)
```

**Arguments**

- `x` A `spark_connection`, `ml_pipeline`, or a `tbl_spark`.
- `partitions` number of partitions
- `partition_by` vector of column names used for partitioning, only supported for Spark 2.0+
sdf_residuals.ml_model_generalized_linear_regression

**Model Residuals**

**Description**

This generic method returns a Spark DataFrame with model residuals added as a column to the model training data.

**Usage**

```r
## S3 method for class 'ml_model_generalized_linear_regression'
sdf_residuals(
  object,
  type = c("deviance", "pearson", "working", "response"),
  ...
)
```

**Arguments**

- `object` Spark ML model object.
- `type` type of residuals which should be returned.
- `...` additional arguments

**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**

```r
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_random_split()`, `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` A `spark_connection`, `ml_pipeline`, or 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_separate_column

**Separate a Vector Column into Scalar Columns**

**Description**
Given a vector column in a Spark DataFrame, split that into \( n \) separate columns, each column made up of the different elements in the column \( column \).

**Usage**
```
sdf_separate_column(x, column, into = NULL)
```

**Arguments**
- **x**: A spark_connection, ml_pipeline, or a tbl_spark.
- **column**: The name of a (vector-typed) column.
- **into**: A specification of the columns that should be generated from \( column \). This can either be a vector of column names, or an R list mapping column names to the (1-based) index at which a particular vector element should be extracted.

### sdf_seq

**Create DataFrame for Range**

**Description**
Creates a DataFrame for the given range

**Usage**
```
sdf_seq(
sc,
from = 1L,
to = 1L,
by = 1L,
repartition = type,
type = c("integer", "integer64")
)
```

**Arguments**
- **sc**: The associated Spark connection.
- **from, to**: The start and end to use as a range
- **by**: The increment of the sequence.
- **repartition**: The number of partitions to use when distributing the data across the Spark cluster.
- **type**: The data type to use for the index, either "integer" or "integer64".
### 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_random_split()`, `sdf_register()`, `sdf_sample()`

---

### sdf_sql

**Spark DataFrame from SQL**

**Description**

Defines a Spark DataFrame from a SQL query, useful to create Spark DataFrames without collecting the results immediately.

**Usage**

`sdf_sql(sc, sql)`

**Arguments**

- **sc**: A `spark_connection`.
- **sql**: a 'SQL' query used to generate a Spark DataFrame.
sdf_with_sequential_id

Add a Sequential ID Column to a Spark DataFrame

Description
Add a sequential ID column to a Spark DataFrame. The Spark zipWithIndex function is used to produce these. This differs from sdf_with_unique_id in that the IDs generated are independent of partitioning.

Usage
sdf_with_sequential_id(x, id = "id", from = 1L)

Arguments
- x: A spark_connection, ml_pipeline, or a tbl_spark.
- id: The name of the column to host the generated IDs.
- from: The starting value of the id column

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: A spark_connection, ml_pipeline, or a tbl_spark.
- id: The name of the column to host the generated IDs.
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.

Description

These routines allow you to manage your connections to Spark.

Usage

```r
spark_connect(
  master,
  spark_home = Sys.getenv("SPARK HOME"),
  method = c("shell", "livy", "databricks", "test", "qubole"),
  app_name = "sparklyr",
  version = NULL,
  config = spark_config(),
  extensions = sparklyr::registered_extensions(),
  packages = NULL,
  ...
)

spark_connection_is_open(sc)

spark_disconnect(sc, ...)

spark_disconnect_all()

spark_submit(
  master,
  file,
  spark_home = Sys.getenv("SPARK HOME"),
  app_name = "sparklyr",
  version = NULL,
  config = spark_config(),
  extensions = sparklyr::registered_extensions(),
  ...
)
```

Arguments

- `master` Spark cluster url to connect to. Use "local" to connect to a local instance of Spark installed via `spark_install`.
spark_apply

The path to a Spark installation. Defaults to the path provided by the SPARK_HOME environment variable. If SPARK_HOME is defined, it will 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. Default connection method is "shell" to connect using spark-submit, use "livy" to perform remote connections using HTTP, or "databricks" when using a Databricks clusters.

app_name

The application name to be used while running in the Spark cluster.

version

The version of Spark to use. Required for "local" Spark connections, optional otherwise.

config

Custom configuration for the generated Spark connection. See spark_config for details.

extensions

Extension R packages to enable for this connection. By default, all packages enabled through the use of sparklyr::register_extension will be passed here.

packages

A list of Spark packages to load. For example, "delta" or "kafka" to enable Delta Lake or Kafka. Also supports full versions like "io.delta:delta-core_2.11:0.4.0". This is similar to adding packages into the sparklyr.shell.packages configuration option. Notice that the version parameter is used to choose the correct package, otherwise assumes the latest version is being used.

... Optional arguments; currently unused.

sc A spark_connection.

file Path to R source file to submit for batch execution.

Details

When using method = "livy", jars are downloaded from GitHub but the path to a local sparklyr JAR can also be specified through the livy.jars setting.

Examples

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

spark_disconnect(sc)
```

---

**spark_apply** | Apply an R Function in Spark

**Description**

Applies an R function to a Spark object (typically, a Spark DataFrame).
Usage

spark_apply(
    x,  
    f,  
    columns = NULL,  
    memory = !is.null(name),  
    group_by = NULL,  
    packages = NULL,  
    context = NULL,  
    name = NULL,  
    barrier = NULL,  
    ...  
)

Arguments

x
An object (usually a spark_tbl) coercable to a Spark DataFrame.

f
A function that transforms a data frame partition into a data frame. The function f has signature f(df, context, group1, group2, ...) where df is a data frame with the data to be processed, context is an optional object passed as the context parameter and group1 to groupN contain the values of the group_by values. When group_by is not specified, f takes only one argument. Can also be an rlang anonymous function. For example, as ~ .x + 1 to define an expression that adds one to the given .x data frame.

columns
A vector of column names or a named vector of column types for the transformed object. When not specified, a sample of 10 rows is taken to infer the output columns automatically, to avoid this performance penalty, specify the column types. The sample size is configurable using the sparklyr.apply.schema.infer configuration option.

memory
Boolean; should the table be cached into memory?

group_by
Column name used to group by data frame partitions.

packages
Boolean to distribute .libPaths() packages to each node, a list of packages to distribute, or a package bundle created with spark_apply_bundle(). Defaults to TRUE or the sparklyr.apply.packages value set in spark_config(). For clusters using Yarn cluster mode, packages can point to a package bundle created using spark_apply_bundle() and made available as a Spark file using config$sparklyr.shell.files. For clusters using Livy, packages can be manually installed on the driver node.

For offline clusters where available.packages() is not available, manually download the packages database from https://cran.r-project.org/web/packages/packages.rds and set Sys.setenv(sparklyr.apply.packagesdb = "<path1-to-rds>"). Otherwise, all packages will be used by default.

For clusters where R packages already installed in every worker node, the spark.r.libpaths config entry can be set in spark_config() to the local packages library. To specify multiple paths collapse them (without spaces) with a comma delimiter (e.g., "/lib/path/one,/lib/path/two").
context Optional object to be serialized and passed back to f().
name Optional table name while registering the resulting data frame.
barrier Optional to support Barrier Execution Mode in the scheduler.
... Optional arguments; currently unused.

Configuration

spark_config() settings can be specified to change the workers environment.
For instance, to set additional environment variables to each worker node use the sparklyr.apply.env.*
config, to launch workers without --vanilla use sparklyr.apply.options.vanilla set to FALSE,
to run a custom script before launching Rscript use sparklyr.apply.options.rscript.before.

Examples

## Not run:
library(sparklyr)
sc <- spark_connect(master = "local[3]"

# creates an Spark data frame with 10 elements then multiply times 10 in R
sdf_len(sc, 10) %>% spark_apply(function(df) df * 10)

# using barrier mode
sdf_len(sc, 3, repartition = 3) %>%
  spark_apply(nrow, barrier = TRUE, columns = c(id = "integer")) %>%
  collect()

## End(Not run)

---

spark_apply_bundle Create Bundle for Spark Apply

Description

Creates a bundle of packages for spark_apply().

Usage

spark_apply_bundle(packages = TRUE, base_path = getwd(), session_id = NULL)

Arguments

packages List of packages to pack or TRUE to pack all.
base_path Base path used to store the resulting bundle.
session_id An optional ID string to include in the bundle file name to allow the bundle to be session-specific
spark_apply_log    Log Writer for Spark Apply

Description

    Writes data to log under spark_apply().

Usage

    spark_apply_log(..., level = "INFO")

Arguments

    ...  Arguments to write to log.
    level  Severity level for this entry; recommended values: INFO, ERROR or WARN.

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,
        jar_path = NULL,
        jar_dep = 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.
spark_config

Options

- **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.

- **jar_path**: The path to the jar tool to be used during compilation of your Spark extension.

- **jar_dep**: An optional list of additional jar dependencies.

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

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

### Arguments

- **file**: Name of the configuration file
- **use_default**: TRUE to use the built-in defaults provided in this package

### Details

Read Spark configuration using the `config` package.

### Value

Named list with configuration data
**Description**

Convenience function to initialize a Kubernetes configuration instead of `spark_config()`, exposes common properties to set in Kubernetes clusters.

**Usage**

```r
spark_config_kubernetes(
  master,
  version = "2.3.2",
  image = "spark:sparklyr",
  driver = random_string("sparklyr-"),
  account = "spark",
  jars = "local:///opt/sparklyr",
  forward = TRUE,
  executors = NULL,
  conf = NULL,
  timeout = 120,
  ports = c(8880, 8881, 4040),
  fix_config = identical(.Platform$OS.type, "windows"),
  ...
)
```

**Arguments**

- `master` Kubernetes url to connect to, found by running `kubectl cluster-info`.
- `version` The version of Spark being used.
- `image` Container image to use to launch Spark and sparklyr. Also known as `spark.kubernetes.container.image`.
- `driver` Name of the driver pod. If not set, the driver pod name is set to "sparklyr" suffixed by id to avoid name conflicts. Also known as `spark.kubernetes.driver.pod.name`.
- `account` Service account that is used when running the driver pod. The driver pod uses this service account when requesting executor pods from the API server. Also known as `spark.kubernetes.authenticate.driver.serviceAccountName`.
- `jars` Path to the sparklyr jars: either, a local path inside the container image with the sparklyr jars copied when the image was created or, a path accessible by the container where the sparklyr jars were copied. You can find a path to the sparklyr jars by running `system.file("java/",package = "sparklyr")`.
- `forward` Should ports used in sparklyr be forwarded automatically through Kubernetes? Default to `TRUE` which runs `kubectl port-forward` and `pkill kubectl` on disconnection.
- `executors` Number of executors to request while connecting.
spark_config_packages

Description

Creates Spark Configuration

Usage

```r
spark_config_packages(config, packages, version)
```

Arguments

- **config**: The Spark configuration object.
- **packages**: A list of named packages or versioned packages to add.
- **version**: The version of Spark being used.

spark_config_settings

Description

Retrieves available sparklyr settings that can be used in configuration files or `spark_config()`.

Usage

```r
spark_config_settings()
```
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_connection-class
spark_connection class

Description
spark_connection class

spark_connection_find  
Find Spark Connection

Description
Finds an active spark connection in the environment given the connection parameters.

Usage
spark_connection_find(master = NULL, app_name = NULL, method = NULL)

Arguments
master  
The Spark master parameter.

app_name  
The Spark application name.

method  
The method used to connect to Spark.
spark_context_config

Runtime configuration interface for the Spark Context.

Description
Retrieves the runtime configuration interface for the Spark Context.

Usage
spark_context_config(sc)

Arguments
sc A spark_connection.

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**

```r
code
spark_default_compilation_spec(
  pkg = infer_active_package_name(),
  locations = NULL
)
```

**Arguments**

- `pkg` The package containing Spark extensions to be compiled.
- `locations` Additional locations to scan. By default, the directories `/opt/scala` and `/usr/local/scala` will be scanned.

---

spark_dependency

*Define a Spark dependency*

**Description**

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

**Usage**

```r
code
spark_dependency(
  jars = NULL,
  packages = NULL,
  initializer = NULL,
  catalog = NULL,
  repositories = NULL,
  ...
)
```

**Arguments**

- `jars` Character vector of full paths to JAR files.
- `packages` Character vector of Spark packages names.
- `initializer` Optional callback function called when initializing a connection.
- `catalog` Optional location where extension JAR files can be downloaded for Livy.
- `repositories` Character vector of Spark package repositories.
- `...` Additional optional arguments.
**spark_dependency_fallback**

*Fallback to Spark Dependency*

**Value**

An object of type `spark_dependency`.

**spark_dependency_fallback**

<table>
<thead>
<tr>
<th>spark_version</th>
<th>The Spark version being requested in <code>spark_dependencies</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>supported_versions</td>
<td>The Spark versions that are supported by this extension.</td>
</tr>
</tbody>
</table>

**Description**

Helper function to assist falling back to previous Spark versions.

**Usage**

```r
spark_dependency_fallback(spark_version, supported_versions)
```

**Arguments**

- `spark_version`: The Spark version being requested in `spark_dependencies`.
- `supported_versions`: The Spark versions that are supported by this extension.

**Value**

A Spark version to use.

---

**spark_extension**

*Create Spark Extension*

**Description**

Creates an R package ready to be used as an Spark extension.

**Usage**

```r
spark_extension(path)
```

**Arguments**

- `path`: Location where the extension will be created.
spark_home_set

Set the SPARK_HOME environment variable

Description

Set the SPARK_HOME environment variable. This slightly speeds up some operations, including the connection time.

Usage

```r
spark_home_set(path = NULL, ...)
```

Arguments

- `path` A string containing the path to the installation location of Spark. If NULL, the path to the most latest Spark/Hadoop versions is used.
- `...` Additional parameters not currently used.

Value

The function is mostly invoked for the side-effect of setting the SPARK_HOME environment variable. It also returns TRUE if the environment was successfully set, and FALSE otherwise.

Examples

```r
## Not run:
# Not run due to side-effects
spark_home_set()

## End(Not run)
```

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

```r
spark_jobj(x, ...)
```
spark_jobj-class

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_jobj-class  spark_jobj class

spark_load_table  Reads from a Spark Table into a Spark DataFrame.

Description

spark_jobj class

spark_load_table  Reads from a Spark Table into a Spark DataFrame.

Usage

spark_load_table(
  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://", "s3a://", 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?
spark_read_csv

See Also

Other Spark serialization routines: spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()

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 = NULL,  
    path = name,  
    header = TRUE,  
    columns = NULL,  
    infer_schema = is.null(columns),  
    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://", "s3a://" and "file://" protocols.
hdr Boolean; should the first row of data be used as a header? Defaults to TRUE.
columns A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.
infer_schema Boolean; should column types be automatically inferred? Requires one extra pass over the data. Defaults to is.null(columns).
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?
... Optional arguments; currently unused.

Details

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://). If you are reading from a secure S3 bucket be sure to set the following in your spark-defaults.conf spark.hadoop.fs.s3a.access.key, spark.hadoop.fs.s3a.secret.key or any of the methods outlined in the aws-sdk documentation Working with AWS credentials In order to work with the...
newer s3a:// protocol also set the values for spark.hadoop.fs.s3a.impl and spark.hadoop.fs.s3a.endpoint. In addition, to support v4 of the S3 api be sure to pass the -Dcom.amazonaws.services.s3.enableV4 driver options for the config key spark.driver.extraJavaOptions For instructions on how to configure s3n:// check the hadoop documentation: 

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_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()

---

**spark_read_delta**

Read from Delta Lake into a Spark DataFrame.

**Description**

Read from Delta Lake into a Spark DataFrame.

**Usage**

```r
spark_read_delta(
  sc,
  path,
  name = NULL,
  version = NULL,
  timestamp = NULL,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  ...
)
```

**Arguments**

- `sc` A spark_connection.
- `path` The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- `name` The name to assign to the newly generated table.
- `version` The version of the delta table to read.
- `timestamp` The timestamp of the delta table to read. For example, "2019-01-01" or "2019-01-01'"T"00:00:00.000Z".
- `options` A list of strings with additional options.
spark_read_jdbc

Read from JDBC connection into a Spark DataFrame.

Description

Read from JDBC connection into a Spark DataFrame.

Usage

spark_read_jdbc(
  sc,
  name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  ...
)

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?)

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_jdbc(),
spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(),
spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(),
spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(),
spark_write_table(), spark_write_text()
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 = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  ...
)

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://", "s3a://" 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.

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()
spark_read_libsvm

Read libsvm file into a Spark DataFrame.

Description

Read libsvm file into a Spark DataFrame.

Usage

```r
spark_read_libsvm(
  sc,
  name = NULL,
  path = name,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  ...
)
```

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?

columns  A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.

...  Optional arguments; currently unused.

Details

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure to set the following in your spark-defaults.conf
spark.hadoop.fs.s3a.access.key, spark.hadoop.fs.s3a.secret.key or any of the methods outlined in the aws-sdk documentation Working with AWS credentials In order to work with the newer s3a:// protocol also set the values for spark.hadoop.fs.s3a.impl and spark.hadoop.fs.s3a.endpoint. In addition, to support v4 of the S3 api be sure to pass the -Dcom.amazonaws.services.s3.enableV4 driver options for the config key spark.driver.extraJavaOptions For instructions on how to configure s3n:// check the hadoop documentation: s3n authentication properties

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()
spark_read_orc

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://", "s3a://" and "file://" protocols.
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?
...  Optional arguments; currently unused.

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()

---

spark_read_orc  Read a ORC file into a Spark DataFrame

Description

Read a ORC file into a Spark DataFrame.

Usage

spark_read_orc(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  schema = NULL,
  ...
)
spark_read_parquet

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://", "s3a://" and "file://" protocols.

- **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?

- **columns**
  A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.

- **schema**
  A (java) read schema. Useful for optimizing read operation on nested data.

- ... Optional arguments; currently unused.

Details

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()
Usage

```r
spark_read_parquet(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  schema = NULL,
  ...
)
```

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://", "s3a://" and "file://" protocols.
- **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?
- **columns**: A vector of column names or a named vector of column types. If specified, the elements can be "binary" for `BinaryType`, "boolean" for `BooleanType`, "byte" for `ByteType`, "integer" for `IntegerType`, "integer64" for `LongType`, "double" for `DoubleType`, "character" for `StringType`, "timestamp" for `TimestampType` and "date" for `DateType`.
- **schema**: A (java) read schema. Useful for optimizing read operation on nested data.
- **...**: Optional arguments; currently unused.

Details

You can read data from HDFS (`hdfs://`), S3 (`s3a://`), as well as the local file system (`file://`). If you are reading from a secure S3 bucket be sure to set the following in your `spark-defaults.conf`

```text
spark.hadoop.fs.s3a.access.key, spark.hadoop.fs.s3a.secret.key
```

or any of the methods outlined in the [aws-sdk documentation Working with AWS credentials](https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/s3-get-started.html). In order to work with the newer `s3a://` protocol also set the values for `spark.hadoop.fs.s3a.impl` and `spark.hadoop.fs.s3a.endpoint`. In addition, to support v4 of the S3 api be sure to pass the `-Dcom.amazonaws.services.s3.enableV4` driver options for the config key `spark.driver.extraJavaOptions`. For instructions on how to configure `s3n://` check the hadoop documentation: [s3n authentication properties](https://docs.aws.amazon.com/hadoop-guide/latest/ug/s3authentication.html).
spark_read_source

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(),
spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_source(),
spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(),
spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(),
spark_write_table(), spark_write_text()

spark_read_source Read from a generic source into a Spark DataFrame.

Description

Read from a generic source into a Spark DataFrame.

Usage

spark_read_source(
  sc,
  name = NULL,
  path = name,
  source,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  ...
)

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://”, “s3a://” and “file://” protocols.
source A data source capable of reading data.
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?)
owrite Boolean; overwrite the table with the given name if it already exists?
spark_read_table

Reads from a Spark Table into a Spark DataFrame.

Description

Reads from a Spark Table into a Spark DataFrame.

Usage

spark_read_table(
  sc,
  name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  columns = NULL,
  ...
)

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?)
spark_read_text

Read a Text file into a Spark DataFrame

Description

Read a text file into a Spark DataFrame.

Usage

spark_read_text(
  sc,
  name = NULL,
  path = name,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  options = list(),
  whole = FALSE,
  ...
)

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://”, “s3a://” and “file://” protocols.
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?)
### spark_save_table

Saves a Spark DataFrame as a Spark table

**Description**

Saves a Spark DataFrame and as a Spark table.

**Usage**

```r
spark_save_table(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://", "s3a://" and "file://" protocols.
- **mode**
  - A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also [http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes](http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes) for your version of Spark.
- **options**
  - A list of strings with additional options.

**Details**

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure to set the following in your spark-defaults.conf:

- `spark.hadoop.fs.s3a.access.key`
- `spark.hadoop.fs.s3a.secret.key`

or any of the methods outlined in the aws-sdk documentation Working with AWS credentials. In order to work with the newer s3a:// protocol also set the values for spark.hadoop.fs.s3a.impl and spark.hadoop.fs.s3a.endpoint.

In addition, to support v4 of the S3 api be sure to pass the `-Dcom.amazonaws.services.s3.enableV4` driver options for the config key `spark.driver.extraJavaOptions` For instructions on how to configure s3n:// check the hadoop documentation: [s3n authentication properties](http://hadoop.apache.org/docs/r150500/api/org/apache/hadoop/conf/Configuration.html#setAuthenticationProvider).

**See Also**

- Other Spark serialization routines: `spark_load_table()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_save_table()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`
**spark_session_config**

**Description**

Retrieves or sets runtime configuration entries for the Spark Session

**Usage**

```r
spark_session_config(sc, config = TRUE, value = NULL)
```

**Arguments**

- `sc`: A `spark_connection`
- `config`: The configuration entry name(s) (e.g., "spark.sql.shuffle.partitions"). Defaults to NULL to retrieve all configuration entries.
- `value`: The configuration value to be set. Defaults to NULL to retrieve configuration entries.

---

**spark_table_name**

**Generate a Table Name from Expression**

**Description**

Attempts to generate a table name from an expression; otherwise, assigns an auto-generated generic name with "sparklyr_" prefix.

**Usage**

```r
spark_table_name(expr)
```

**Arguments**

- `expr`: The expression to attempt to use as name
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`.

---

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(),
  mode = NULL,
  partition_by = NULL,
  ...
)
spark_write_delta

Arguments

- **x**: A Spark DataFrame or dplyr operation
- **path**: The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" 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 ".
- **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 default values, defaults to NULL.
- **options**: A list of strings with additional options.
- **mode**: A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also [http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes](http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes) for your version of Spark.
- **partition_by**: A character vector. Partitions the output by the given columns on the file system.
- **...**: Optional arguments; currently unused.

See Also

Other Spark serialization routines: `spark_load_table()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_save_table()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

`spark_write_delta`  
Writes a Spark DataFrame into Delta Lake

Description

Writes a Spark DataFrame into Delta Lake.

Usage

```r
spark_write_delta(
  x,  
  path,  
  mode = NULL,  
  options = list(),  
  partition_by = NULL,  
  ...
)
```
spark_write_jdbc

 Arguments

 x A Spark DataFrame or dplyr operation

 path The path to the file. Needs to be accessible from the cluster. Supports the “hdfs://”, “s3a://” and “file://” protocols.

 mode A character element. Specifies the behavior when data or table already exists. Supported values include: ‘error’, ‘append’, ‘overwrite’ and ignore. Notice that ‘overwrite’ will also change the column structure.

 For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.

 options A list of strings with additional options.

 partition_by A character vector. Partitions the output by the given columns on the file system.

 ... Optional arguments; currently unused.

 See Also

 Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(), spark_write_source(), spark_write_table(), spark_write_text()

 spark_write_jdbc  Writes a Spark DataFrame into a JDBC table

 Description

 Writes a Spark DataFrame into a JDBC table.

 Usage

 spark_write_jdbc(
   x,
   name,
   mode = NULL,
   options = list(),
   partition_by = NULL,
   ...
 )
**spark_write_json**

Write a Spark DataFrame to a JSON file

**Description**

Serialize a Spark DataFrame to the JavaScript Object Notation format.

**Usage**

```r
spark_write_json(
  x, path, 
  mode = NULL, 
  options = list(), 
  partition_by = 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://", "s3a://" and "file://" protocols.
- `mode`: A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also [http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes](http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes) for your version of Spark.
- `options`: A list of strings with additional options.
- `partition_by`: A character vector. Partitions the output by the given columns on the file system.
- `...`: Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: `spark_load_table()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_save_table()`, `spark_write_csv()`, `spark_write_delata()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`
mode: A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.

options: A list of strings with additional options.

partition_by: A character vector. Partitions the output by the given columns on the file system.

...: Optional arguments; currently unused.

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read.jdbc(), spark_read.json(), spark_read.libsvm(), spark_read.orc(), spark_read.parquet(), spark_read.source(), spark_read_table(), spark_read.text(), spark_save_table(), spark_write_csv(), spark_write.delta(), spark_write.jdbc(), spark_write.orc(), spark_write.parquet(), spark_write.source(), spark_write_table(), spark_write.text()

spark_write_orc: Write a Spark DataFrame to a ORC file

Description

Serialize a Spark DataFrame to the ORC format.

Usage

spark_write_orc(
  x, 
  path, 
  mode = NULL, 
  options = list(), 
  partition_by = 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://", "s3a://" and "file://" protocols.

mode: A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.
spark_write_parquet

Write a Spark DataFrame to a Parquet file

Description

Serialize a Spark DataFrame to the Parquet format.

Usage

```r
spark_write_parquet(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = 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://", "s3a://" and "file://" protocols.
- `mode` A character element. Specifies the behavior when data or table already exists. Supported values include: "error", "append", "overwrite" and "ignore". Notice that "overwrite" will also change the column structure. For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.
- `options` A list of strings with additional options. See http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration.
- `partition_by` A character vector. Partitions the output by the given columns on the file system.
- `...` Optional arguments; currently unused.
spark_write_source

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_source(), spark_write_table(), spark_write_text()

spark_write_source  Writes a Spark DataFrame into a generic source

Description

Writes a Spark DataFrame into a generic source.

Usage

spark_write_source(
  x,
  source,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)

Arguments

x  A Spark DataFrame or dplyr operation
source  A data source capable of reading data.
mode  A character element. Specifies the behavior when data or table already exists. Supported values include: ‘error’, ‘append’, ‘overwrite’ and ignore. Notice that ‘overwrite’ will also change the column structure. For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.
options  A list of strings with additional options.
partition_by  A character vector. Partitions the output by the given columns on the file system.
...  Optional arguments; currently unused.

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(), spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(), spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(), spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_source(), spark_write_table(), spark_write_text()
spark_write_table  Writes a Spark DataFrame into a Spark table

Description

 Writes a Spark DataFrame into a Spark table.

Usage

spark_write_table(
  x,
  name,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)

Arguments

 x  A Spark DataFrame or dplyr operation
 name  The name to assign to the newly generated table.
 mode  A character element. Specifies the behavior when data or table already exists.
       Supported values include: ‘error’, ‘append’, ‘overwrite’ and ignore. Notice that
       ‘overwrite’ will also change the column structure.
               For more details see also http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes for your version of Spark.
 options  A list of strings with additional options.
 partition_by  A character vector. Partitions the output by the given columns on the file
               system.
 ...  Optional arguments; currently unused.

See Also

Other Spark serialization routines: spark_load_table(), spark_read_csv(), spark_read_delta(),
spark_read_jdbc(), spark_read_json(), spark_read_libsvm(), spark_read_orc(), spark_read_parquet(),
spark_read_source(), spark_read_table(), spark_read_text(), spark_save_table(), spark_write_csv(),
spark_write_delta(), spark_write_jdbc(), spark_write_json(), spark_write_orc(), spark_write_parquet(),
spark_write_source(), spark_write_text()
spark_write_text

Write a Spark DataFrame to a Text file

Description

Serialize a Spark DataFrame to the plain text format.

Usage

```r
spark_write_text(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = 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://", "s3a://" and "file://" protocols.
- **mode**: A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also [http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes](http://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes) for your version of Spark.
- **options**: A list of strings with additional options.
- **partition_by**: A character vector. Partitions the output by the given columns on the file system.
- **...**: Optional arguments; currently unused.

See Also

Other Spark serialization routines: `spark_load_table()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_save_table()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`
src_databases

*Show database list*

**Description**
Show database list

**Usage**
```
src_databases(sc, ...)
```

**Arguments**
- `sc` A `spark_connection`.
- `...` Optional arguments; currently unused.

---

stream_find

*Find Stream*

**Description**
Finds and returns a stream based on the stream’s identifier.

**Usage**
```
stream_find(sc, id)
```

**Arguments**
- `sc` The associated Spark connection.
- `id` The stream identifier to find.

**Examples**
```
## Not run:
split <- spark_connect(master = "local")
sdf_len(split, 10) %>%
  spark_write_parquet(path = "parquet-in")

stream <- stream_read_parquet(split, "parquet-in") %>%
  stream_write_parquet("parquet-out")

stream_id <- stream_id(stream)
stream_find(split, stream_id)

## End(Not run)
```
**stream_generate_test**  
*Generate Test Stream*

**Description**
Generates a local test stream, useful when testing streams locally.

**Usage**
```r
stream_generate_test(
    df = rep(1:1000),
    path = "source",
    distribution = floor(10 + 1e+05 * stats::dbinom(1:20, 20, 0.5)),
    iterations = 50,
    interval = 1
)
```

**Arguments**
- `df` The data frame used as a source of rows to the stream, will be cast to data frame if needed. Defaults to a sequence of one thousand entries.
- `path` Path to save stream of files to, defaults to "source".
- `distribution` The distribution of rows to use over each iteration, defaults to a binomial distribution. The stream will cycle through the distribution if needed.
- `iterations` Number of iterations to execute before stopping, defaults to fifty.
- `interval` The interval in seconds use to write the stream, defaults to one second.

**Details**
This function requires the `callr` package to be installed.

---

**stream_id**  
*Spark Stream's Identifier*

**Description**
Retrieves the identifier of the Spark stream.

**Usage**
```r
stream_id(stream)
```

**Arguments**
- `stream` The spark stream object.
stream_read_csv

---

**stream_name**  
*Spark Stream’s Name*

---

**Description**
Retrieves the name of the Spark stream if available.

**Usage**

```
stream_name(stream)
```

**Arguments**

- `stream`  
The spark stream object.

---

**stream_read_csv**  
*Read CSV Stream*

---

**Description**
Reads a CSV stream as a Spark dataframe stream.

**Usage**

```
stream_read_csv(
  sc,
  path,
  name = NULL,
  header = TRUE,
  columns = NULL,
  delimiter = ",",
  quote = "\",
  escape = "\\",
  charset = "UTF-8",
  null_value = NULL,
  options = list(),
  ...
)
```

**Arguments**

- `sc`  
A spark_connection.

- `path`  
The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.

- `name`  
The name to assign to the newly generated stream.
stream_read_csv

header  Boolean; should the first row of data be used as a header? Defaults to TRUE.
columns  A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType,"integer" for IntegerType,"integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.
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.
...  Optional arguments; currently unused.

See Also

Other Spark stream serialization: stream_read_delta(), stream_read_json(), stream_read_kafka(),
stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(),
stream_write_console(), stream_write_csv(), stream_write_delta(), stream_write_json(),
stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_parquet(),
stream_write_text()

Examples

## Not run:

sc <- spark_connect(master = "local")
dir.create("csv-in")
write.csv(iris, "csv-in/data.csv", row.names = FALSE)
csv_path <- file.path("file://", getwd(), "csv-in")
stream <- stream_read_csv(sc, csv_path) %>% stream_write_csv("csv-out")
stream_stop(stream)

## End(Not run)
stream_read_delta  
Read Delta Stream

Description
Reads a Delta Lake table as a Spark dataframe stream.

Usage
stream_read_delta(sc, path, name = NULL, options = list(), ...)

Arguments
- sc: A spark_connection.
- path: The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- name: The name to assign to the newly generated stream.
- options: A list of strings with additional options.
- ...: Optional arguments; currently unused.

Details
Please note that Delta Lake requires installing the appropriate package by setting the packages parameter to "delta" in spark_connect().

See Also
Other Spark stream serialization: stream_read_csv(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_console(), stream_write_csv(), stream_write_delta(), stream_write_json(), stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_parquet(), stream_write_text()

Examples
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4", packages = "delta")
sdf.len(sc, 5) %>% spark_write_delta(path = "delta-test")

stream <- stream_read_delta(sc, "delta-test") %>%
  stream_write_json("json-out")
stream_stop(stream)
**stream_read_json**

## Description

Reads a JSON stream as a Spark dataframe stream.

## Usage

```r
stream_read_json(sc, path, name = NULL, columns = NULL, options = list(), ...)
```

## Arguments

- `sc` A `spark_connection`
- `path` The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- `name` The name to assign to the newly generated stream.
- `columns` A vector of column names or a named vector of column types. If specified, the elements can be "binary" for `BinaryType`, "boolean" for `BooleanType`, "byte" for `ByteType`, "integer" for `IntegerType`, "integer64" for `LongType`, "double" for `DoubleType`, "character" for `StringType`, "timestamp" for `TimestampType` and "date" for `DateType`.
- `options` A list of strings with additional options.
- `...` Optional arguments; currently unused.

## See Also

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

## Examples

```r
## Not run:

sc <- spark_connect(master = "local")
dir.create("json-in")
jsonlite::write_json(list(a = c(1,2), b = c(10,20)), "json-in/data.json")

json_path <- file.path("file://", getwd(), "json-in")
```
stream <- stream_read_json(sc, json_path) %>% stream_write_json("json-out")

stream_stop(stream)

## End(Not run)

---

**stream_read_kafka**  
*Read Kafka Stream*

**Description**
Reads a Kafka stream as a Spark dataframe stream.

**Usage**

```r
stream_read_kafka(sc, name = NULL, options = list(), ...)
```

**Arguments**

- `sc`  
  A `spark_connection`.
- `name`  
  The name to assign to the newly generated stream.
- `options`  
  A list of strings with additional options.
- `...`  
  Optional arguments; currently unused.

**Details**
Please note that Kafka requires installing the appropriate package by setting the `packages` parameter to "kafka" in `spark_connect()`

**See Also**
Other Spark stream serialization:  
- `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

**Examples**

```r
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.3", packages = "kafka")
read_options <- list(kafka.bootstrap.servers = "localhost:9092", subscribe = "topic!")
```
write_options <- list(kafka.bootstrap.servers = "localhost:9092", topic = "topic2")

stream <- stream_read_kafka(sc, options = read_options) %>%
  stream_write_kafka(options = write_options)

stream_stop(stream)

## End(Not run)

---

**stream_read_orc**  

**Read ORC Stream**

**Description**

Reads an ORC stream as a Spark dataframe stream.

**Usage**

```r
stream_read_orc(sc, path, name = NULL, columns = NULL, options = list(), ...)
```

**Arguments**

- `sc`  
  A spark_connection.

- `path`  
  The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.

- `name`  
  The name to assign to the newly generated stream.

- `columns`  
  A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.

- `options`  
  A list of strings with additional options.

- `...`  
  Optional arguments; currently unused.

**See Also**

Other Spark stream serialization:  

- `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`
Examples

```r
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>% spark_write_orc("orc-in")
stream <- stream_read_orc(sc, "orc-in") %>% stream_write_orc("orc-out")
stream_stop(stream)

## End(Not run)
```

---

**stream_read_parquet**  
Read Parquet Stream

**Description**

Reads a parquet stream as a Spark dataframe stream.

**Usage**

```r
stream_read_parquet(
  sc, 
  path, 
  name = NULL, 
  columns = NULL, 
  options = list(), 
  ...
)
```

**Arguments**

- `sc`  
  A spark_connection.
- `path`  
  The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- `name`  
  The name to assign to the newly generated stream.
- `columns`  
  A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType.
- `options`  
  A list of strings with additional options.
- `...`  
  Optional arguments; currently unused.
**stream_read_socket**

**See Also**

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

**Examples**

```r
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>% spark_write_parquet("parquet-in")
stream <- stream_read_parquet(sc, "parquet-in") %>% stream_write_parquet("parquet-out")
stream_stop(stream)
## End(Not run)
```

---

**stream_read_socket**  
Read Socket Stream

**Description**

Reads a Socket stream as a Spark dataframe stream.

**Usage**

`stream_read_socket(sc, name = NULL, columns = NULL, options = list(), ...)`

**Arguments**

- `sc`: A `spark_connection`.
- `name`: The name to assign to the newly generated stream.
- `columns`: A vector of column names or a named vector of column types. If specified, the elements can be "binary" for `BinaryType`, "boolean" for `BooleanType`, "byte" for `ByteType", "integer" for `IntegerType", "integer64" for `LongType", "double" for `DoubleType", "character" for `StringType", "timestamp" for `TimestampType" and "date" for `DateType".
- `options`: A list of strings with additional options.
- `...`: Optional arguments; currently unused.
See Also

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

Examples

```r
## Not run:

sc <- spark_connect(master = "local")
# Start socket server from terminal, example: nc -lk 9999
stream <- stream_read_socket(sc, options = list(host = "localhost", port = 9999))
stream

## End(Not run)
```

### stream_read_text

**Read Text Stream**

**Description**

Reads a text stream as a Spark dataframe stream.

**Usage**

```r
stream_read_text(sc, path, name = NULL, options = list(), ...)
```

**Arguments**

- `sc` A spark_connection.
- `path` The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- `name` The name to assign to the newly generated stream.
- `options` A list of strings with additional options.
- `...` Optional arguments; currently unused.

**See Also**

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`
stream_render

Examples

```r
## Not run:

sc <- spark_connect(master = "local")
dir.create("text-in")
writeLines("A text entry", "text-in/text.txt")
text_path <- file.path("file://", getwd(), "text-in")

stream <- stream_read_text(sc, text_path) %>% stream_write_text("text-out")
stream_stop(stream)

## End(Not run)
```

---

stream_render

Render Stream

Description

Collects streaming statistics to render the stream as an `htmlwidget`.

Usage

```r
stream_render(stream = NULL, collect = 10, stats = NULL, ...)
```

Arguments

- `stream`:
  - The stream to render.
- `collect`:
  - The interval in seconds to collect data before rendering the `htmlwidget`.
- `stats`:
  - Optional stream statistics collected using `stream_stats()`, when specified, `stream` should be omitted.
- `...`:
  - Additional optional arguments.

Examples

```r
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local")
dir.create("iris-in")
write.csv(iris, "iris-in/iris.csv", row.names = FALSE)

stream <- stream_read_csv(sc, "iris-in") %>%
  stream_write_csv("iris-out")
```
stream_stats

Description

Collects streaming statistics, usually, to be used with `stream_render()` to render streaming statistics.

Usage

`stream_stats(stream, stats = list())`

Arguments

- `stream` - The stream to collect statistics from.
- `stats` - An optional stats object generated using `stream_stats()`.

Value

A stats object containing streaming statistics that can be passed back to the `stats` parameter to continue aggregating streaming stats.

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>%
  spark_write_parquet(path = "parquet-in")

stream <- stream_read_parquet(sc, "parquet-in") %>%
  stream_write_parquet("parquet-out")

stream_stats(stream)
## End(Not run)
```
**stream_stop**

*Stops a Spark Stream*

**Description**

Stops processing data from a Spark stream.

**Usage**

```
stream_stop(stream)
```

**Arguments**

- `stream` The spark stream object to be stopped.

---

**stream_trigger_continuous**

*Spark Stream Continuous Trigger*

**Description**

Creates a Spark structured streaming trigger to execute continuously. This mode is the most performant but not all operations are supported.

**Usage**

```
stream_trigger_continuous(checkpoint = 5000)
```

**Arguments**

- `checkpoint` The checkpoint interval specified in milliseconds.

**See Also**

```
stream_trigger_interval
```
stream_trigger_interval

*Spark Stream Interval Trigger*

**Description**

Creates a Spark structured streaming trigger to execute over the specified interval.

**Usage**

```
stream_trigger_interval(interval = 1000)
```

**Arguments**

- `interval` The execution interval specified in milliseconds.

**See Also**

- `stream_trigger_continuous`

---

stream_view

*View Stream*

**Description**

Opens a Shiny gadget to visualize the given stream.

**Usage**

```
stream_view(stream, ...)
```

**Arguments**

- `stream` The stream to visualize.
- `...` Additional optional arguments.

**Examples**

```r
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local")
dir.create("iris-in")
write.csv(iris, "iris-in/iris.csv", row.names = FALSE)

stream_read_csv(sc, "iris-in/") %>%
```

---
stream_watermark

```r
stream_watermark("iris-out/") %>%
  stream_view() %>%
  stream_stop()

## End(Not run)
```

---

### stream_watermark

**Watermark Stream**

**Description**

Ensures a stream has a watermark defined, which is required for some operations over streams.

**Usage**

```r
stream_watermark(x, column = "timestamp", threshold = "10 minutes")
```

**Arguments**

- **x**: An object coercable to a Spark Streaming DataFrame.
- **column**: The name of the column that contains the event time of the row, if the column is missing, a column with the current time will be added.
- **threshold**: The minimum delay to wait to data to arrive late, defaults to ten minutes.

---

### stream_write_console

**Write Console Stream**

**Description**

Writes a Spark dataframe stream into console logs.

**Usage**

```r
stream_write_console(
  x,
  mode = c("append", "complete", "update"),
  options = list(),
  trigger = stream_trigger_interval(),
  ...
)
```
stream_write_csv

Arguments

x A Spark DataFrame or dplyr operation
mode Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
options A list of strings with additional options.
trigger The trigger for the stream query, defaults to micro-batches running every 5 seconds. See stream_trigger_interval and stream_trigger_continuous.
... Optional arguments; currently unused.

See Also

Other Spark stream serialization: stream_read_csv(), stream_read_delta(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_csv(), stream_write_delta(), stream_write_json(), stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_parquet(), stream_write_text()

Examples

## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>% dplyr::transmute(text = as.character(id)) %>% spark_write_text("text-in")
stream <- stream_read_text(sc, "text-in") %>% stream_write_console()
stream_stop(stream)

## End(Not run)

stream_write_csv
Write CSV Stream

Description

Writes a Spark dataframe stream into a tabular (typically, comma-separated) stream.

Usage

stream_write_csv(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
)
stream_write_csv

```r
checkpoint = file.path(path, "checkpoint"),
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://", "s3a://" and "file://" protocols.
- **mode**: Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
- **trigger**: The trigger for the stream query, defaults to micro-batches running every 5 seconds. See `stream_trigger_interval` and `stream_trigger_continuous`.
- **checkpoint**: The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
- **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 "\"".
- **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 default values, defaults to NULL.
- **options**: A list of strings with additional options.
- **...**: Optional arguments; currently unused.

See Also

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

Examples

```r
## Not run:
sc <- spark_connect(master = "local")

dir.create("csv-in")
```
write.csv(iris, "csv-in/data.csv", row.names = FALSE)

csv_path <- file.path("file://", getwd(), "csv-in")

stream <- stream_read_csv(sc, csv_path) %>% stream_write_csv("csv-out")

stream_stop(stream)

### End(Not run)

---

**stream_write_delta**  
*Write Delta Stream*

---

**Description**

Writes a Spark dataframe stream into a Delta Lake table.

**Usage**

```r
stream_write_delta(
  x,
  path,
  mode = c("append", "complete", "update"),
  checkpoint = file.path("checkpoints", random_string("")),
  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://", "s3a://" and "file://" protocols.
- **mode**: Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
- **checkpoint**: The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
- **options**: A list of strings with additional options.
- **...**: Optional arguments; currently unused.

**Details**

Please note that Delta Lake requires installing the appropriate package by setting the `packages` parameter to "delta" in `spark_connect()`.
stream_write_json

See Also

Other Spark stream serialization: stream_read_csv(), stream_read_delta(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_console(), stream_write_csv(), stream_write_json(), stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_parquet(), stream_write_text()

Examples

```r
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4", packages = "delta")

dir.create("text-in")
writeLines("A text entry", "text-in/text.txt")

text_path <- file.path("file://", getwd(), "text-in")

stream <- stream_read_text(sc, text_path) %>% stream_write_delta(path = "delta-test")
stream_stop(stream)

## End(Not run)
```

---

stream_write_json Write JSON Stream

Description

 Writes a Spark dataframe stream into a JSON stream.

Usage

```r
stream_write_json(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  ...
)
```
stream_write_kafka

Write Kafka Stream

Description

Writes a Spark dataframe stream into a Kafka stream.

Arguments

- **x**: A Spark DataFrame or dplyr operation
- **path**: The destination path. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- **mode**: Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
- **trigger**: The trigger for the stream query, defaults to micro-batches running every 5 seconds. See `stream_trigger_interval` and `stream_trigger_continuous`.
- **checkpoint**: The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
- **options**: A list of strings with additional options.
- **...**: Optional arguments; currently unused.

See Also

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_kafka()`, `stream_write_memory()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
dir.create("json-in")
jsonlite::write_json(list(a = c(1,2), b = c(10,20)), "json-in/data.json")
json_path <- file.path("file://", getwd(), "json-in")
stream <- stream_read_json(sc, json_path) %>% stream_write_json("json-out")
stream_stop(stream)

## End(Not run)
```
Usage

stream_write_kafka(
  x,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path("checkpoints", random_string("")),
  options = list(),
  ...
)

Arguments

x A Spark DataFrame or dplyr operation
mode Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
trigger The trigger for the stream query, defaults to micro-batches running every 5 seconds. See stream_trigger_interval and stream_trigger_continuous.
checkpoint The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
options A list of strings with additional options.
... Optional arguments; currently unused.

Details

Please note that Kafka requires installing the appropriate package by setting the packages parameter to "kafka" in spark_connect()

See Also

Other Spark stream serialization: stream_read_csv(), stream_read_delta(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_console(), stream_write_csv(), stream_write_delta(), stream_write_json(), stream_write_memory(), stream_write_orc(), stream_write_parquet(), stream_write_text()

Examples

## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "2.3", packages = "kafka")
read_options <- list(kafka.bootstrap.servers = "localhost:9092", subscribe = "topic1")
write_options <- list(kafka.bootstrap.servers = "localhost:9092", topic = "topic2")

stream <- stream_read_kafka(sc, options = read_options) %>%
  stream_write_kafka(options = write_options)
stream_stop(stream)

## End(Not run)

---

**stream_write_memory**  
Write Memory Stream

---

**Description**

Writes a Spark dataframe stream into a memory stream.

**Usage**

```r
stream_write_memory(
  x,
  name = random_string("sparklyr_tmp_"),
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path("checkpoints", name, random_string("")),
  options = list(),
  ...
)
```

**Arguments**

- **x**: A Spark DataFrame or dplyr operation
- **name**: The name to assign to the newly generated stream.
- **mode**: Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
- **trigger**: The trigger for the stream query, defaults to micro-batches running every 5 seconds. See `stream_trigger_interval` and `stream_trigger_continuous`.
- **checkpoint**: The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
- **options**: A list of strings with additional options.
- **...**: Optional arguments; currently unused.

**See Also**

Other Spark stream serialization: `stream_read_csv()`, `stream_read_delta()`, `stream_read_json()`, `stream_read_kafka()`, `stream_read_orc()`, `stream_read_parquet()`, `stream_read_socket()`, `stream_read_text()`, `stream_write_console()`, `stream_write_csv()`, `stream_write_delta()`, `stream_write_json()`, `stream_write_kafka()`, `stream_write_orc()`, `stream_write_parquet()`, `stream_write_text()`
**stream_write_orc**

## Examples

```r
## Not run:
sc <- spark_connect(master = "local")
dir.create("csv-in")
write.csv(iris, "csv-in/data.csv", row.names = FALSE)
csv_path <- file.path("file://", getwd(), "csv-in")
stream <- stream_read_csv(sc, csv_path) %>% stream_write_memory("csv-out")
stream_stop(stream)

## End(Not run)
```

---

**stream_write_orc**  
**Write a ORC Stream**

### Description

Writes a Spark dataframe stream into an ORC stream.

### Usage

```r
stream_write_orc(
  x,  
  path,  
  mode = c("append", "complete", "update"),  
  trigger = stream_trigger_interval(),  
  checkpoint = file.path(path, "checkpoints", random_string("")),  
  options = list(),  
  ...
)
```

### Arguments

- **x** A Spark DataFrame or dplyr operation
- **path** The destination path. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
- **mode** Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
- **trigger** The trigger for the stream query, defaults to micro-batches running every 5 seconds. See `stream_trigger_interval` and `stream_trigger_continuous`.  

**stream_write_parquet**

checkpoint The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.

options A list of strings with additional options.

... Optional arguments; currently unused.

See Also

Other Spark stream serialization: *stream_read_csv*, *stream_read_delta*, *stream_read_json*, *stream_read_kafka*, *stream_read_orc*, *stream_read_parquet*, *stream_read_socket*, *stream_read_text*, *stream_write_console*, *stream_write_csv*, *stream_write_delta*, *stream_write_json*, *stream_write_kafka*, *stream_write_memory*, *stream_write_parquet*, *stream_write_text*

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>% spark_write_orc("orc-in")

stream <- stream_read_orc(sc, "orc-in") %>% stream_write_orc("orc-out")

stream_stop(stream)

## End(Not run)
```

---

**stream_write_parquet**  
Write Parquet Stream

Description

Writes a Spark dataframe stream into a parquet stream.

Usage

```r
stream_write_parquet(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  ...
)
```
stream_write_text

Arguments

x A Spark DataFrame or dplyr operation
path The destination path. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.
mode Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".
trigger The trigger for the stream query, defaults to micro-batches running every 5 seconds. See stream_trigger_interval and stream_trigger_continuous.
checkpoint The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.
options A list of strings with additional options.
... Optional arguments; currently unused.

See Also

Other Spark stream serialization: stream_read_csv(), stream_read_delta(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_console(), stream_write_csv(), stream_write_delta(), stream_write_json(), stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_text()

Examples

```r
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>% spark_write_parquet("parquet-in")
stream <- stream_read_parquet(sc, "parquet-in") %>% stream_write_parquet("parquet-out")
stream_stop(stream)

## End(Not run)
```

Description

Writes a Spark dataframe stream into a text stream.
stream_write_text

Usage

stream_write_text(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  ...
)

Arguments

x A Spark DataFrame or dplyr operation

path The destination path. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.

mode Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".

trigger The trigger for the stream query, defaults to micro-batches running every 5 seconds. See stream_trigger_interval and stream_trigger_continuous.

checkpoint The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.

options A list of strings with additional options.

See Also

Other Spark stream serialization: stream_read_csv(), stream_read_delta(), stream_read_json(), stream_read_kafka(), stream_read_orc(), stream_read_parquet(), stream_read_socket(), stream_read_text(), stream_write_console(), stream_write_csv(), stream_write_delta(), stream_write_json(), stream_write_kafka(), stream_write_memory(), stream_write_orc(), stream_write_parquet()

Examples

## Not run:

going {  
  sc <- spark_connect(master = "local")  
  dir.create("text-in")  
  writeLines("A text entry", "text-in/text.txt")  
  text_path <- file.path("file://", getwd(), "text-in")  
  stream <- stream_read_text(sc, text_path) %>% stream_write_text("text-out")  
  stream_stop(stream)  
}
## 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

```r
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_change_db

**Use specific database**

### Description

Use specific database

### Usage

```r
tbl_change_db(sc, name)
```

### Arguments

- `sc`: A `spark_connection`.
- `name`: The database name.
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>A spark_connection.</td>
</tr>
<tr>
<td>name</td>
<td>The table name.</td>
</tr>
</tbody>
</table>
Index

Topic internal
  spark_config_packages, 171

augment.ml_model_aft_survival_regression
  (ml_survival_regression_tiders), 137
augment.ml_model_als (ml_als_tiders), 76
augment.ml_model_bisecting_kmeans
  (ml_unsupervised_tiders), 139
augment.ml_model_decision_tree_classification
  (ml_tree_tiders), 137
augment.ml_model_decision_tree_regression
  (ml_tree_tiders), 137
augment.ml_model_gaussian_mixture
  (ml_unsupervised_tiders), 139
augment.ml_model_gbt_classification
  (ml_tree_tiders), 137
augment.ml_model_gbt_regression
  (ml_tree_tiders), 137
augment.ml_model_generalized_linear_regression
  (ml_glm_tiders), 101
augment.ml_model_isotonic_regression
  (ml_isotonic_regression_tiders), 104
augment.ml_model_kmeans
  (ml_unsupervised_tiders), 139
augment.ml_model_lda (ml_lda_tiders), 111
augment.ml_model_linear_regression
  (ml_glm_tiders), 101
augment.ml_model_linear_svc
  (ml_linear_svc_tiders), 116
augment.ml_model_logistic_regression
  (ml_logistic_regression_tiders), 120
augment.ml_model_multilayer_perceptron_classification
  (ml_multilayer_perceptron_tiders), 125
augment.ml_model_naive_bayes
  (ml_naive_bayes_tiders), 128
augment.ml_model_pca (ml_pca_tiders),
  130
augment.ml_model_random_forest_classification
  (ml_tree_tiders), 137
augment.ml_model_random_forest_regression
  (ml_tree_tiders), 137

checkpoint_directory, 8
compile_package_jars, 9, 168, 174
config, 169
connection_config, 9
copy_to.spark_connection, 10
cut, 14
download_scalac, 11
ensure, 11
find_scalac, 12
ft_binarizer, 12, 15, 17, 19–21, 23, 24,
  26–29, 31, 33, 35–37, 39, 40, 42, 43,
  45, 46, 48, 50, 51, 53–55, 57–59, 61
ft_bucketed_random_projection_lsh
  (ft_lsh), 30
ft_bucketizer, 13, 14, 17, 19–21, 23, 24,
  26–29, 31, 33, 35–37, 39, 40, 42, 43,
  45, 46, 48, 50, 51, 53–55, 57–59, 61
ft_chisq_selector, 13, 15, 19–21, 23,
  24, 26–29, 31, 33, 35–37, 39, 40, 42,
  43, 45, 46, 48, 50, 51, 53–55, 57–59,
  61
ft_count_vectorizer, 13, 15, 17, 20, 21,
  23, 24, 26–29, 31, 33, 35–37, 39, 40,
  42, 43, 45, 46, 48, 50, 51, 53–55,
  57–59, 61, 109

231
ft_discrete_cosine_transform (ft_dct), 19
ft_dplyr_transrformer (ft_sql_transformer), 49
ft_elementwise_product, 13, 15, 17, 19, 20,
20, 23, 24, 26–29, 31, 33, 35–37, 39,
40, 42, 43, 45, 46, 48, 50, 51, 53–55,
57–59, 61
ft_feature hasher, 13, 15, 17, 19–21, 22,
24, 26–29, 31, 33, 35–37, 39, 40, 42,
43, 45, 46, 48, 50, 51, 53–55, 57–59,
61
ft_hashing_tf, 13, 15, 17, 19–21, 22, 23, 24,
26–29, 31, 33, 35–37, 39, 40, 42, 43,
45, 46, 48, 50, 51, 53–55, 57–59, 61
ft_idf, 13, 15, 17, 19–21, 22, 23, 24, 25, 27–29,
31, 33, 35–37, 39, 40, 42, 43, 45, 46,
ft_imputer, 13, 15, 17, 19–21, 23, 24, 26, 26,
28, 29, 31, 33, 35, 38–40, 42, 43,
45, 46, 48, 50, 51, 53–55, 57–59, 61
ft_index_to_string, 13, 15, 17, 19–21, 23,
24, 26, 27, 29, 31, 33, 35, 36,
38–40, 42, 43, 45, 46, 48, 50, 51,
53–55, 57–59, 61
ft_interaction, 13, 15, 17, 19–21, 23, 24,
26–28, 28, 31, 33, 35, 36, 38–40, 42,
43, 45, 46, 48, 50, 51, 53–55, 57–59,
61
ft_lsh, 13, 15, 17, 19–21, 23, 24,
26–29, 31, 33, 35, 36, 38–40, 42,
43, 45, 46, 48, 50, 51, 53–55, 57–59,
61
ft_lsh_utils, 31
ft_max abs scaler, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 32, 35, 36, 38–40, 42,
43, 45, 46, 48, 50, 51, 53–55, 57–59,
61
ft_min max scaler, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 33, 34, 36, 38–40, 42,
43, 45, 47, 48, 50, 51, 53–55, 57–59,
61
ft_minhash_lsh (ft_lsh), 30
ft_ngram, 13, 15, 17, 19–21, 23, 24, 26–29,
31, 33, 35, 38–40, 42, 43, 45, 47,
ft_normalizer, 13, 15, 17, 19–21, 23, 24,
26–29, 31, 33, 35, 36, 37, 39, 40, 42,
43, 45, 47, 48, 50, 51, 53–55, 57–59,
61
ft_one_hot_encoder, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 33, 35, 36, 38, 39, 40,
42, 43, 45, 47, 48, 50, 51, 53–55,
57–59, 61
ft_one_hot_encoder_estimator, 13, 15, 17,
19–21, 23, 24, 26–29, 31, 33, 35, 36,
38, 39, 39, 42, 43, 45, 47, 48, 50, 51,
53–55, 57–59, 61
ft_pca, 13, 15, 17, 19–21, 23, 24, 26–29, 31,
33, 35, 36, 38–40, 40, 43, 45, 47, 48,
50, 51, 53, 54, 56–59, 61
ft_polynomial_expansion, 13, 15, 17,
19–21, 23, 24, 26–29, 31, 33, 35, 36,
38–40, 42, 43, 43, 47, 48, 50, 51, 53,
54, 56–59, 61
ft_quantile_discretizer, 13, 15, 17,
19–21, 23, 24, 26–29, 31, 33, 35, 36,
38–40, 42, 43, 43, 47, 48, 50, 51, 53,
54, 56–59, 61
ft_r_formula, 13, 15–17, 19–21, 23, 24,
26–29, 31, 33, 35, 36, 38–40, 42, 43,
45, 47, 47, 50, 51, 53, 54, 56–59, 61,
71, 73, 77, 83, 91, 94, 95, 98, 99,
102, 105, 107, 108, 112, 113, 115,
118, 119, 122, 123, 126, 129, 133,
134
ft_regex_tokenizer, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 33, 35, 36, 38–40, 42,
43, 45, 45, 48, 50, 51, 53, 54, 56–59,
61
ft_sql_transformer, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 33, 35, 36, 38–40, 42,
43, 45, 47, 48, 49, 51, 53, 54, 56–59,
61
ft_standard scaler, 13, 15, 17, 19–21, 23,
24, 26–29, 31, 33, 35, 36, 38–40, 42,
43, 45, 47, 48, 50, 50, 53, 54, 56–59,
61
ft_stop_words_remover, 13, 15, 17, 19–21,
23, 24, 26–29, 31, 33, 35, 36, 38–40, 42,
43, 45, 47, 48, 50, 51, 53, 54, 56–59,
61
ft_string_indexer_model
INDEX
87
ml_save, 71, 84, 96, 99, 103, 113, 115, 119, 123, 126, 129, 134
ml_save (ml-persistence), 65
ml_stage, 136
ml_stages (ml_stage), 136
ml_sub_models (ml-tuning), 67
ml_summary, 136
ml_survival_regression
  (ml_aft_survival_regression), 70
ml_survival_regression_tidiers, 137
ml_topics_matrix (ml_lda), 106
ml_train_validation_split (ml-tuning), 67
ml_transform (ml-transform-methods), 66
ml_tree_feature_importance
  (ml_feature_importances), 89
ml_tree_tidiers, 137
ml_uid, 139
ml_unsupervised_tidiers, 139
ml_validation_metrics (ml-tuning), 67
ml_vocabulary (ft_count_vectorizer), 17

NA, 140
na.replace, 140
numeric_version, 192

random_string, 140
reactiveSpark, 141
register_extension, 142
registerDoSpark, 141
registered_extensions
  (register_extension), 142

sdf-saveload, 142
sdf-transform-methods, 66, 143
sdf_along, 144
sdf_bind, 144
sdf_bind_cols (sdf_bind), 144
sdf_bind_rows (sdf_bind), 144
sdf_broadcast, 145
sdf_checkpoint, 145
sdf_coalesce, 146
sdf_collect, 146
sdf_copy_to, 147, 156, 157, 159, 161
sdf_crosstab, 148
sdf_debug_string, 148
sdf_describe, 149
sdf_dim, 149
sdf_drop_duplicates, 150
sdf_fit (sdf-transform-methods), 143
sdf_fit_and_transform
  (sdf-transform-methods), 143
sdf_import (sdf_copy_to), 147
sdf_is_streaming, 150
sdf_last_index, 150
sdf_len, 151
sdf_load_parquet (sdf-saveload), 142
sdf_load_table (sdf-saveload), 142
sdf_ncol (sdf_dim), 149
sdf_nrow (sdf_dim), 149
sdf_num_partitions, 151
sdf_partition (sdf_random_split), 155
sdfPersist, 152
sdf_pivot, 152
sdf_predict, 79, 88
sdf_predict (sdf-transform-methods), 143
sdf_project, 153
sdf_quantile, 154
sdf_random_split, 147, 155, 157, 159, 161
sdf_read_column, 156
sdf_register, 147, 156, 157, 159, 161
sdf_repartition, 157
sdf_residuals
  (sdf_residuals.ml_model_generalized_linear_regression), 158
sdf_residuals.ml_model_generalized_linear_regression, 158
sdf_sample, 147, 156, 157, 158, 161
sdf_save_parquet (sdf-saveload), 142
sdf_save_table (sdf-saveload), 142
sdf_schema, 159
sdf_separate_column, 160
sdf_seq, 160
sdf_sort, 147, 156, 157, 159, 161
sdf_sql, 161
sdf_transform (sdf-transform-methods), 143
sdf_with_sequential_id, 162
sdf_with_unique_id, 162
spark-api, 163
spark-connections, 164
spark_apply, 165
spark_apply_bundle, 167
spark_apply_log, 168
spark_compilation_spec, 168
218, 219, 221–224, 226–228
stream_read_parquet, 205–209, 210, 212, 218, 219, 221–224, 226–228
stream_read_socket, 205–209, 211, 212, 218, 219, 221–224, 226–228
stream_read_text, 205–209, 211, 212, 218, 219, 221–224, 226–228
stream_render, 213
stream_stats, 214
stream_stop, 215
stream_trigger_continuous, 215, 216, 218, 219, 222–225, 227, 228
stream_trigger_interval, 215, 216, 218, 219, 222–225, 227, 228
stream_view, 216
stream_watermark, 217
stream_write_console, 205–209, 211, 212, 217, 219, 221–224, 226–228
stream_write_csv, 205–209, 211, 212, 218, 219, 221–224, 226–228
stream_write_delta, 205–209, 211, 212, 218, 219, 220, 222–224, 226–228
stream_write_json, 205–209, 211, 212, 218, 219, 221, 223, 224, 226–228
stream_write_kafka, 205–209, 211, 212, 218, 219, 221, 222, 222, 224, 226–228
stream_write_memory, 205–209, 211, 212, 218, 219, 221–223, 224, 226–228
stream_write_orc, 205–209, 211, 212, 218, 219, 221–224, 225, 227, 228
stream_write_parquet, 205–209, 211, 212, 218, 219, 221–224, 226, 226, 228
stream_write_text, 205–209, 211, 212, 218, 219, 221–224, 226, 227, 227

tbl_cache, 229
tbl_change_db, 229
tbl_uncache, 230
tidy.ml_model_aft_survival_regression
  (ml_survival_regression tidiers), 137
tidy.ml_model_gaussian_mixture
  (ml_unsupervised tidiers), 139
tidy.ml_model_gbt_classification
  (ml_tree tidiers), 137
tidy.ml_model_gbt_regression
  (ml_tree tidiers), 137
tidy.ml_model_generalized_linear_regression
  (ml_glm tidiers), 101
tidy.ml_model_isotonic_regression
  (ml_isotonic_regression tidiers), 104
tidy.ml_model_kmeans
  (ml_unsupervised tidiers), 139
tidy.ml_model_lda (ml_nda tidiers), 111
tidy.ml_model_linear_regression
  (ml_glm tidiers), 101
tidy.ml_model_linear_svc
  (ml_linear_svc tidiers), 116
tidy.ml_model_logistic_regression
  (ml_logistic_regression tidiers), 120
tidy.ml_model_multilayer_perceptron_classification
  (ml_multilayer_perceptron tidiers), 125
tidy.ml_model_naive_bayes
  (ml_naive_bayes tidiers), 128
tidy.ml_model_pca (ml_pca tidiers), 130
tidy.ml_model_random_forest_classification
  (ml_tree tidiers), 137
tidy.ml_model_random_forest_regression
  (ml_tree tidiers), 137