

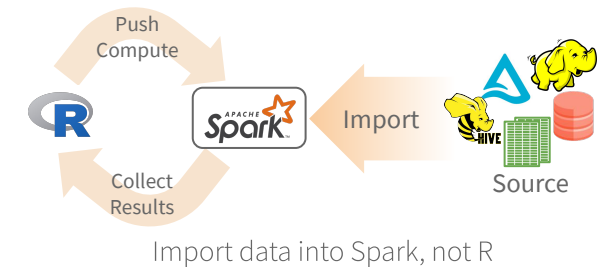
Data Science in Spark with *sparklyr* : : CHEAT SHEET



Intro

sparklyr is an R interface for Apache Spark™. *sparklyr* enables us to write all of our analysis code in R, but have the actual processing happen inside Spark clusters. Easily manipulate and model large-scale using R and Spark via *sparklyr*.

Import



READ A FILE INTO SPARK

Arguments that apply to all functions:
sc, name, path, options=list(), repartition=0,
memory=TRUE, overwrite=TRUE

CSV

`spark_read_csv(header = TRUE,
columns=NULL,
infer_schema=TRUE, delimiter = ";",
quote = "\"", escape = "\\\"", charset =
"UTF-8", null_value = NULL)`

JSON

`spark_read_json()`

PARQUET

`spark_read_parquet()`

TEXT

`spark_read_text()`

HIVE TABLE

`spark_read_table()`

ORC

`spark_read_orc()`

LIBSVM

`spark_read_libsvm()`

JDBC

`spark_read_jdbc()`

DELTA

`spark_read_delta()`

R DATA FRAME INTO SPARK

`dplyr::copy_to(dest, df, name)`

FROM A TABLE IN HIVE

`dplyr::tbl(scr, ...)` Creates a reference to the table without loading it into memory



Import

- From R (`copy_to()`)
- Read a file (`spark_read_*`)
- Read Hive table (`tbl()`)



Wrangle

- `dplyr` verb
- Feature transformer (`ft_*`)
- Direct Spark SQL (`DBI`)

Visualize

- Collect result, plot in R
- Use `dbplot`

Model

- Spark MLlib (`m1_*`)
- H2O Extension

Communicate
Collect results into R
share using
`rmarkdown`

[R for Data Science](#)
[Grolemund & Wickham](#)

Wrangle

DPLYR VERBS

Translates into Spark SQL statements

`copy_to(sc, mtcars) %>%
mutate(trm = ifelse(am == 0, "auto", "man")) %>%
group_by(trm) %>%
summarise_all(mean)`

FEATURE TRANSFORMERS

- `ft_binarizer()` - Assigned values based on threshold
- `ft_bucketizer()` - Numeric column to discretized column
- `ft_count_vectorizer()` - Extracts a vocabulary from document
- `ft_discrete_cosine_transform()` - 1D discrete cosine transform of a real vector
- `ft_elementwise_product()` - Element-wise product between 2 cols
- `ft_hashing_tf()` - Maps a sequence of terms to their term frequencies using the hashing trick.
- `ft_idf()` - Compute the Inverse Document Frequency (IDF) given a collection of documents
- `ft_imputer()` - Imputation estimator for completing missing values, uses the mean or the median of the columns
- `ft_index_to_string()` - Index labels back to label as strings
- `ft_interaction()` - Takes in Double and Vector type columns and outputs a flattened vector of their feature interactions



`ft_max_abs_scaler()` - Rescale each feature individually to range [-1, 1]



`ft_min_max_scaler()` - Rescale each feature individually to a common range [min, max] linearly



`ft_ngram()` - Converts the input array of strings into an array of n-grams



`ft_bucketed_random_projection_lsh()`
`ft_minhash_lsh()` - Locality Sensitive Hashing functions for Euclidean distance and Jaccard distance (MinHash)



`ft_normalizer()` - Normalize a vector to have unit norm using the given p-norm



`ft_one_hot_encoder()` - Continuous to binary vectors



`ft_pca()` - Project vectors to a lower dimensional space of top k principal components



`ft_quantile_discretizer()` - Continuous to binned categorical values



`ft_regex_tokenizer()` - Extracts tokens either by using the provided regex pattern to split the text



`ft_standard_scaler()` - Removes the mean and scaling to unit variance using column summary statistics



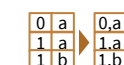
`ft_stop_words_remover()` - Filters out stop words from input



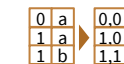
`ft_string_indexer()` - Column of labels into a column of label indices.



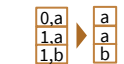
`ft_tokenizer()` - Converts to lowercase and then splits it by white spaces



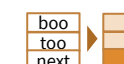
`ft_vector_assembler()` - Combine vectors into single row-vector



`ft_vector_indexer()` - Indexing categorical feature columns in a dataset of Vector

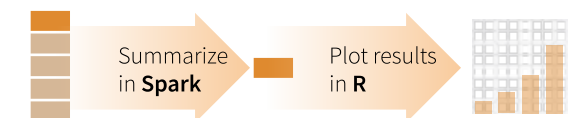


`ft_vector_slicer()` - Takes a feature vector and outputs a new feature vector with a subarray of the original features



`ft_word2vec()` - Word2Vec transforms a word into a code

Visualize



DPLYR + GGLOT2

`copy_to(sc, mtcars) %>%
group_by(cyl) %>%
summarise(mpg_m = mean(mpg)) %>%
collect() %>%
ggplot() +
geom_col(aes(cyl, mpg_m))`

Summarize in Spark
Collect results in R
Create plot

DBPLOT

`copy_to(sc, mtcars) %>%
dbplot_histogram(mpg) +
labs(title = "Histogram of MPG")`

`dbplot_histogram(data, x, bins = 30, binwidth = NULL)` - Calculates the histogram bins in Spark and plots in ggplot2
`dbplot_raster(data, x, y, fill = n(), resolution = 100, complete = FALSE)` - Visualize 2 continuous variables. Use instead of `geom_point()`

Data Science in Spark with *sparklyr* : : CHEAT SHEET



Modeling

REGRESSION

ml_linear_regression() - Regression using linear regression.

ml_aft_survival_regression() - Parametric survival regression model named accelerated failure time (AFT) model

ml_generalized_linear_regression() - Generalized linear regression model

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

ml_random_forest_regressor() - Regression using random forests.

CLASSIFICATION

ml_linear_svc() - Classification using linear support vector machines

ml_logistic_regression() - Logistic regression

ml_multilayer_perceptron_classifier() - Classification model based on the Multilayer Perceptron.

ml_naive_bayes() - Naive Bayes Classifiers. It supports Multinomial NB which can handle finitely supported discrete data

ml_one_vs_rest() - Reduction of Multiclass Classification to Binary Classification. Performs reduction using one against all strategy.

TREE

ml_decision_tree_classifier() | **ml_decision_tree()** | **ml_decision_tree_regressor()** - Classification and regression using decision trees

ml_gbt_classifier() | **ml_gradient_boosted_trees()** | **ml_gbt_regressor()** - Binary classification and regression using gradient boosted trees

ml_random_forest_classifier() - Classification and regression using random forests.

ml_feature_importances(model,...) | **ml_tree_feature_importance(model)** - Feature Importance for Tree Models

CLUSTERING

ml_bisecting_kmeans() - A bisecting k-means algorithm based on the paper

ml_lda() | **ml_describe_topics()** | **ml_log_likelihood()** | **ml_log_perplexity()** | **ml_topics_matrix()** - LDA topic model designed for text documents.

ml_gaussian_mixture() - Expectation maximization for multivariate Gaussian Mixture Models (GMMs)

ml_kmeans() | **ml_compute_cost()** - K-means clustering with support for k-means

FP GROWTH

ml_fpgrowth() | **ml_association_rules()** | **ml_freq_itemsets()** - A parallel FP-growth algorithm to mine frequent itemsets.

FEATURE

ml_chisquare_test(x,features,label) - Pearson's independence test for every feature against the label

ml_default_stop_words() - Loads the default stop words for the given language

STATS

ml_summary() - Extracts a metric from the summary object of a Spark ML model

ml_corr() - Compute correlation matrix

correlate package integrates with sparklyr



```
copy_to(sc, mtcars) %>%  
correlate() %>%  
rplot()
```



RECOMMENDATION

ml_als() | **ml_recommend()** - Recommendation using Alternating Least Squares matrix factorization

EVALUATION

ml_clustering_evaluator() - Evaluator for clustering

ml_evaluate() - Compute performance metrics

ml_binary_classification_evaluator() | **ml_binary_classification_eval()** | **ml_classification_eval()** - A set of functions to calculate performance metrics for prediction models.

UTILITIES

ml_call_constructor() - Identifies the associated sparklyr ML constructor for the JVM

ml_model_data() - Extracts data associated with a Spark ML model

ml_standardize_formula() - Generates a formula string from user inputs, to be used in `ml_model`` constructor

ml_uid() - Extracts the UID of an ML object.

Start a Spark session

YARN CLIENT

1. Install RStudio Server on one of the existing nodes, preferably an edge node
2. Locate path to the cluster's Spark Home Directory, it normally is `/usr/lib/spark`
3. Basic configuration example

```
conf <- spark_config()  
conf$spark.executor.memory <- "300M"  
conf$spark.executor.cores <- 2  
conf$spark.executor.instances <- 3  
conf$spark.dynamicAllocation.enabled <- "false"
```
4. Open a connection (some base configurations included in the example)

```
sc <- spark_connect(master = "yarn",  
  spark_home = "/usr/lib/spark/",  
  version = "2.1.0", config = conf)
```

YARN CLUSTER

1. Make sure to have copies of the yarn-site.xml and hive-site.xml files in the RStudio Server
2. Point environment variables to the correct paths

```
Sys.setenv(JAVA_HOME="[Path]")  
Sys.setenv(SPARK_HOME="[Path]")  
Sys.setenv(YARN_CONF_DIR="[Path]")
```
3. Open a connection

```
sc <- spark_connect(master = "yarn-cluster")
```

STANDALONE CLUSTER

1. Install RStudio Server on one of the existing nodes or a server in the same LAN
2. Install a local version of Spark:

```
spark_install(version = "2.0.1")
```
3. Open a connection

```
spark_connect(master="spark://host:port",  
  version = "2.0.1",  
  spark_home = spark_home_dir())
```

LOCAL MODE

No cluster required. Use for learning purposes only

1. Install a local version of Spark:

```
spark_install("2.3")
```
2. Open a connection

```
sc <- spark_connect(master="local")
```

KUBERNETES

1. Use the following to obtain the Host and Port

```
system2("kubectl", "cluster-info")
```
2. Open a connection

```
sc <- spark_connect(config =  
  spark_config_kubernetes(  
    "k8s://https://[HOST]:[PORT]",  
    account = "default",  
    image = "docker.io/owner/repo:version",  
    version = "2.3.1"))
```

MESOS

1. Install RStudio Server on one of the nodes
2. Open a connection

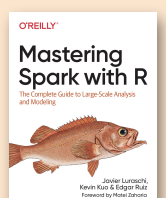
```
sc <- spark_connect(master="[Mesos URL]")
```

CLOUD

Databricks - `spark_connect(method = "databricks")`

Qubole - `spark_connect(method = "qubole")`

More Information



spark.rstudio.com

therinspark.com

