Ciencia de Datos en Spark con *sparklyr* : : **GUÍA RÁPIDA**

Intro

sparklyr es una interfaz de R para **Apache Spark™. *sparklyr*** nos permite escribir todo el código de nuestro análisis en R, pero con el verdadero procesamiento real sucediendo dentro de los clusters de Spark. Manipula y modela a gran escala fácilmente usando R y Spark mediante *sparklyr.*

Importar
- Desde R (**copy_to()**)
- Leer un archivo (**spark_read_**)
- Leer una tabla Hive (**tbl()**)

Domar Datos
- Verbo **dplyr**.
- Transformador de características(**ft_**)
- Acceso Spark SQL (**DBI**)

Visualizar
- Colectar resultados, graficar en R
- Usar **dbplot**

Modelo
- Spark MLlib (**ml_**)
- Extensión H2O

Comunicar
Recolecta los resultados en **R share** usando **rmarkdown**

R for Data Science, Grolemund & Wickham

Importar

Importar datos en Spark, no R

LEE UN ARCHIVO EN SPARK

Argumentos que aplican a todas las funciones:
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()**
TEXTO	**spark_read_text()**
TABLA HIVE	**spark_read_table()**
ORC	**spark_read_orc()**
LIBSVM	**spark_read_libsvm()**
JDBC	**spark_read_jdbc()**
DELTA	**spark_read_delta()**

COPIAR UN DATA FRAME DE R A SPARK

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

DE UNA TABLA EN HIVE

dplyr::**tbl(**scr, …**)** Crea una referencia a la tabla sin cargarla a la memoria

Domar Datos

VERBOS DPLYR

Traduce a sentencias Spark SQL

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

TRANSFORMADORES DE CARACTERÍSTICAS

ft_binarizer() - Valores asignados basados en el límite

ft_bucketizer() - Columna numérica a columna discretizada

ft_count_vectorizer() - Extrae vocabulario del documento

ft_discrete_cosine_transform() - Coseno discreta a vector real

ft_elementwise_product() - Producto basado en elementos entre 2 columnas

ft_hashing_tf() - Asigna una secuencia de términos a sus frecuencias de términos utilizando el truco de *hashing*

ft_idf() - Calcular la frecuencia de documentos inversa (IDF) dada una colección de documentos

ft_imputer() - Estimador de imputación para completar valores perdidos, utiliza la media o la mediana de las columnas

ft_index_to_string() - Index labels back to label as strings

ft_interaction() - Toma columnas del tipo Doble y Vector y devuelve un vector achatado de sus interacciones características

ft_max_abs_scaler() - Reescalar cada función individualmente al rango [-1, 1]

ft_min_max_scaler() - Reescalar la escala de cada función individualmente a un rango común [min, max] linealmente

ft_ngram() - Convierte la matriz de entrada de *strings* en una matriz de n-gramas.

ft_bucketed_random_projection_lsh()
ft_minhash_lsh() - Funciones de Hashing sensibles a la localidad para distancia euclidiana y distancia Jaccard (MinHash)

ft_normalizer() - Normaliza un vector para que tenga norma de unidad usando la p-norm dada

ft_one_hot_encoder()- Vectores continuos a binarios

ft_pca() - Proyecta vectores a un espacio dimensional menor de los top k componentes principales

ft_quantile_discretizer() - Valores categóricos continuos a agrupados

ft_regex_tokenizer() - Extrae tokens usando el patrón regex provisto para dividir el texto

ft_standard_scaler() - Elimina la media y la escala a la varianza de la unidad utilizando estadísticas de resumen de columna

ft_stop_words_remover() - Filtra las palabras vacías del *input*

ft_string_indexer() - Columna de etiquetas en una columna de índices de etiquetas

ft_tokenizer() - Convierte a minúsculas y luego los separa por espacios en blanco

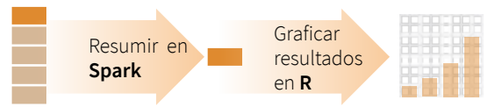
ft_vector_assembler() - Combina vectores en una sola fila-vector

ft_vector_indexer() - Indexa columnas de caract. categóricas en un dataset de Vector

ft_vector_slicer() -Toma un vector de características y genera uno nuevo con un subconjunto de las caract. originales

ft_word2vec() - *Word2Vec* transforma una palabra en código

Visualizar



DPLYR + GGPLOT2

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

Resumir en Spark

Recolectar resultados en R

Crear gráfico

DBPLOT

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

dbplot_histogram(data, x, bins = 30, binwidth = NULL**)** - Calcula los bins del histograma en Spark y grafica en ggplot2

dbplot_raster(data, x, y, fill = n(), resolution = 100, complete = FALSE**)** - Visualiza 2 variables continuas. Usar en lugar de geom_point ()

Ciencia de Datos en Spark con *sparklyr* : : **GUÍA RÁPIDA**

Modelado

REGRESIÓN

ml_linear_regression() - Regresión lineal

ml_aft_survival_regression() - Modelo paramétrico de regresión de supervivencia llamado AFT (por las siglas en inglés *Accelerated Failure Time*)

ml_generalized_linear_regression() - Modelo generalizado de regresión lineal

ml_isotonic_regression() - Actualmente implementados usando el algoritmo PAVA. Sólo soporta algoritmos univariados (única característica)

ml_random_forest_regressor() - Regresión usando *Random Forests* (bosques aleatorios).

CLASIFICACIÓN

ml_linear_svc() - Clasificación usando máquinas de vectores de soporte lineales

ml_logistic_regression() - regresión logística

ml_multilayer_perceptron_classifier() - Modelo de clasificación basado en el Perceptrón Multicapa

ml_naive_bayes() - Clasificadores Bayesianos Naive. Es compatible con Multinomial NB que puede manejar datos discretos con soporte finito

ml_one_vs_rest() - Reducción de clasificación multiclase a clasificación binaria, usando la estrategia “uno contra todos”.

ÁRBOLES

ml_decision_tree_classifier() | ml_decision_tree() | ml_decision_tree_regressor() - clasificación y regresión utilizando árboles de decisión

ml_gbt_classifier() | ml_gradient_boosted_trees() | ml_gbt_regressor() - Clasificación binaria y regresión usando *Gradient Boosted Trees (*árboles de potenciación del gradiente)

ml_random_forest_classifier() - Clasificación y regresión usando *Random Forests*.

ml_feature_importances(model,...)ml_tree_feature_importance(model) - Importancia de las características para los modelos de árboles

AGRUPAMIENTO (*Clustering*)

ml_bisecting_kmeans() - Algoritmo *bisecting K-means* (bisección de k-medias) basado en la publicación.

ml_lda() | ml_describe_topics() | ml_log_likelihood() | ml_log_perplexity() | ml_topics_matrix() - LDA topic model diseñada para documentos de texto

ml_gaussian_mixture() - Maximización de expectativas para los Modelos Gaussianos Multivariados (GMM)

ml_kmeans() | ml_compute_cost() - Agrupamiento de k-medias con soporte para k-medias

CRECIMIENTO FP

ml_fpgrowth() | ml_association_rules() | ml_freq_itemsets() - Algoritmo paralelo de crecimiento FP para extraer conjuntos de elementos frecuentes.

CARACTERÍSTICAS (*Features*)

ml_chisquare_test(x,features,label) - Prueba de independencia de Pearson para cada característica contra la etiqueta

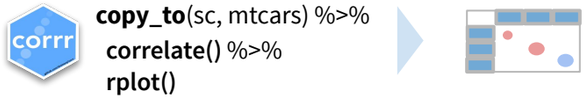
ml_default_stop_words() - Carga las palabras vacías predeterminadas para el idioma dado

STATS

ml_summary() - Extrae una métrica desde el objeto resumen de un modelo Spark ML

ml_corr() - Calcula matriz de correlación

El paquete **correlate** se integra con sparklyr



RECOMENDACIÓN

ml_als() | ml_recommend() - Recomendación usando la factorización de la matriz de Cuadrados Mínimos Alternos (*ALS- Alternating Least Squares*)

EVALUACIÓN

ml_clustering_evaluator() - Evaluador para *clustering*

ml_evaluate() - Calcular métricas de rendimiento

ml_binary_classification_evaluator() | ml_binary_classification_eval() | ml_classification_eval() - Un conjunto de funciones para calcular métricas de rendimiento para modelos de predicción.

UTILIDADES

ml_call_constructor() - Identifica el constructor ML de sparklyr asociado para la JVM (Máquina Virtual Java)

ml_model_data() - Extrae datos asociados a un modelo Spark ML

ml_standardize_formula() - Genera un string de fórmula usando *inputs* de usuarios, para ser usado en el constructor “ ml_model ”

ml_uid() - Extrae el UID de un objeto ML

Comenzar una sesión de Spark

YARN CLIENT

1. Instale RStudio Server en uno de los nodos existentes, preferiblemente un nodo límite.
2. Localice la ruta al directorio principal de Spark del cluster, normalmente es "/ usr / lib / spark"
3. Ejemplo de configuración básica

```
conf <- spark_config()
conf$spark.executor.memory <- "300M"
conf$spark.executor.cores <- 2
conf$spark.executor.instances <- 3
conf$spark.dynamicAllocation.enabled<-"false"
```

4. Abra una conexión (algunas configuraciones básicas incluidas en el ejemplo)

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

YARN CLUSTER

1. Asegúrese de tener copias de los archivos yarn-site.xml y hive-site.xml en el servidor RStudio
2. Señale variables de entorno a las rutas correctas

```
Sys.setenv(JAVA_HOME="[Path]")
Sys.setenv(SPARK_HOME ="[Path]")
Sys.setenv(YARN_CONF_DIR ="[Path]")
```

3. Abra una conexión

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

CLUSTER INDEPENDIENTE

1. Instale RStudio Server en uno de los nodos existentes o en un servidor en la misma LAN
2. Instale una versión local de Spark:

```
spark_install (version = "2.0.1")
```

3. Abra una conexión:

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

MODO LOCAL

No se requiere cluster. Usar sólo con fines de aprendizaje

1. Instale una versión local de Spark

```
spark_install("2.3")
```

2. Abra una conexión

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

KUBERNETES

1. Use lo siguiente para obtener el host y el puerto

```
system2("kubectl", "cluster-info")
```

2. Abra una conexión

```
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. Instale Rstudio Server en uno de los nodos
2. Abra una conexión

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

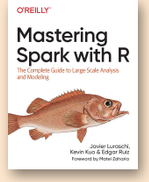
CLOUD

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

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

Más información

spark.rstudio.com



therinspark.com