Introduction to PySpark

Para crear una conexion → Creamos una instancia de la clase SparkContext, requiere unos argumentos que se pueden meter en un objeto SparkConf() ← Conexion al cluster

SparkSession → Interfaz con la conexión.

Imaginemos que el contexto se llama sc

Para ver la version que esta corriendo en el cluster → sc.version

La estructura básica de Spark es RDD (Resilient Distributed Dataset), usamos Spark DataFrame → Optimiza directamente las operaciones.

Para crear una sparkSession →
```
from pyspark.sql import SparkSession
session = SparkSession.builder.getOrCreate()
```

Para ver las tablas creadas en el cluster → spark.catalog.listTables()

Se puede realizar consultas SQL a esas tablas de la siguiente forma →
```
query = "SELECT ... "
resultado = session.sql(query)
resultado.show()
```
← Nombre de la SparkSession Creada

Ese resultado se puede convertir a un dataFrame de pandas → df = resultado.toPandas()

spark_df.show() ← para ver los datos

Para crear un Spark DataFrame a partir de un pd.DataFrame → spark_df = session.createDataFrame(pd_df)

Para crear una tabla temporal en Spark a partir del dataFrame → spark_df.createOrReplaceTempView(name='...', ...)

Se pueden leer directamente archivos a SparkDataFrame → spark_df = session.read.csv(ruta, header=True) ← coge la primera fila como nombre de columnas

Para crear una nueva columna → spark_df = spark_df.withColumn('nueva columna', spark_df.vieja_col * 2)

El método filter funciona igual que un WHERE de SQL → spark_df.filter(condicion)

" " select " " " " SELECT " " → spark_df.select('col_1', 'col_2', ...)

Para hacer operaciones sobre columnas en SELECT y guardalos como alias → spark_df.select(spark_df.airtime/60).alias('duracion')

spark_df.selectExp('air_time/60 as duracion')

Para aplicar funciones de agregación como min o max, usamos el groupBy → spark_df.groupBy().min(column)

Dentro del groupBy podemos meter columnas

groupBy('clase').min(column)

Para unir dataFrames usamos el método .join() → spark_df = spark_df_1.join(spark_df_2, on='key_column', how='left/right')

Introduction to PySpark

El decimal se lleva double

Para modificar el tipo de dato de una columna → `spark_df = spark_df.withColumn("nombre", spark_df.col.cast("integer"))`

Para convertir una columna categórica en un OneHotEncode →
```
PASO 1 → indexer = StringIndexer(inputCol='nombre_col', outputCol='nombre_col')
PASO 2 → encoder = OneHotEncoder(  "  "  ,  "  )
```

Para combinar varias columnas en 1 → `assembler = VectorAssembler(inputCols=['col.1', ... ], outputCol='nombre_col')`

Una vez tenemos los indexer, encode y assemble los metemos en 1 Pipeline →
```
from pyspark.ml import Pipeline
pipe = Pipeline(stages=[indexer, encoder, assembler])
```

Hay que realizar estas transformaciones antes de hacer el split train-test

Para aplicar estas transformaciones → `data_transf = pipe.fit(spark_df).transform(spark_df)`

Para realizar el split sobre los datos transformados → `train, test = data_transf.randomSplit([0.6, 0.4])` (60% train no % test)

Para crear una regresión logística →
```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression()
```

Para evaluar modelos, cuenta con un evaluador →
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator()
evaluator = Binary... (metricName = 'areaUnderROC')
```

Para buscar los parámetros optimos, primero hay que crear un grid →
```
import pyspark.ml.tuning as tune
grid = tune.ParamGridBuilder()
grid = grid.addGrid(Parametro, valores)
grid = grid.build()
```

Para entrenar con validación cruzada →
```
cv = tune.CrossValidator(estimator = lr,
                         estimatorParamMaps = grid,
                         evaluator = evaluator)
```
PARA ENTRENAR
```
model = cv.fit(data_train)
Mejor = model.bestModel
```

Para realizar predicciones con el modelo → `resultados = modelo.transform(data_test)`

" aplicar el evaluador sobre los resultados → `evaluator.evaluate(resultados)`