Utilización de PySpark - Uso, procesamiento y visualización de grandes volúmenes de datos.

Mario Alberto Castañeda Martínez - A01640152

Preparación del ambiente de trabajo para Big Data:

```
from google.colab import drive
drive.mount('/content/drive')
%cd "/content/drive/MyDrive/CONCENTRACION AI/data"
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/CONCENTRACION AI/data
```

```
!pip install pyspark py4j
```

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 4.2 MB/s eta
0:00:00
etadata (setup.py) ... ent already satisfied: py4j in
/usr/local/lib/python3.10/dist-packages (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... e=pyspark-3.5.0-py2.py3-
none-any.whl size=317425344
sha256=578f680675a06618ad375f56f6b95d8f83e37b390bc924bd21748396e20551d
e
  Stored in directory:
/root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01
940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

Después de haber preparado el ambiente con la importanción de Pyspark, ahora se crea una sesión de PySpark para empezar a importar datasets y generar modelos de regresión, clasificación y de agrupamiento:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("tec").getOrCreate()

# Se importan otras librerías necesarias
import pandas as pd
import numpy as np
```

```
 # Cargar dataset para el modelo de regresión:
training =
spark.read.format("libsvm").load("/content/drive/MyDrive/CONCENTRACION
AI/data/sample_linear_regression_data.txt")

training.show(5)
```

```
+-------------------+--------------------+
|              label|            features|
+-------------------+--------------------+
| -9.490009878824548|(10,[0,1,2,3,4,5,...|
| 0.2577820163584905|(10,[0,1,2,3,4,5,...|
| -4.438869807456516|(10,[0,1,2,3,4,5,...|
|-19.782762789614537|(10,[0,1,2,3,4,5,...|
| -7.966593841555266|(10,[0,1,2,3,4,5,...|
+-------------------+--------------------+
only showing top 5 rows
```

Modelo de Regresión Lineal obtenido en Pyspark:

```
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))

# Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
Coefficients: [0.0,0.3229251667740594,-
0.3438548034562219,1.915601702345841,0.05288058680386255,0.76596272045
9771,0.0,-0.15105392669186676,-0.21587930360904645,0.2202536918881343]
Intercept: 0.15989368442397356
numIterations: 6
objectiveHistory: [0.49999999999999994, 0.4967620357443381,
0.49363616643404634, 0.4936351537897608, 0.4936351214177871,
0.49363512062528014, 0.4936351206216114]
+-------------------+
|          residuals|
```

```
+-------------------+
|  -9.889232683103197|
|  0.5533794340053553|
|  -5.204019455758822|
| -20.566686715507508|
|    -9.4497405180564|
|  -6.909112502719487|
|  -10.00431602969873|
|  2.0623978070504845|
|  3.1117508432954772|
|  -15.89360822941938|
|  -5.036284254673026|
|  6.4832158769943335|
|  12.429497299109002|
|  -20.32003219007654|
|    -2.0049838218725|
| -17.867901734183793|
|   7.646455887420495|
| -2.2653482182417406|
|-0.10308920436195645|
|  -1.380034070385301|
+-------------------+
only showing top 20 rows

RMSE: 10.189077
r2: 0.022861
```

El modelo de regresión generado muestra un coeficiente de determinación muy bajo, de tal solo el 2%, lo que significa que el modelo generado no es bueno para pronosticar. Para mejorar el modelo, sería necesario aplicar distintos parámetros para ver si mejora el modelo. O incluso, se podría considerar una limpieza de base de datos para ver si funciona mejor el modelo.

En el modelo anterior se muestran también los residuos generados por el modelo de regresión, esto significa que el modelo además de arrojar un error promedio y coeficiente de determinación, también muestra los errores del modelo. Otra visualización importante que se observa en un principio, son los coeficientes del modelo y las iteraciones del modelo.

Modelo de Clasificación Random Forest:

```
 # Cargar dataset
training_2 =
spark.read.format("libsvm").load("/content/drive/MyDrive/CONCENTRACION
AI/data/sample_libsvm_data.txt")

training_2.show(5)

+-----+--------------------+
|label|            features|
+-----+--------------------+
```

```
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
+-----+--------------------+
only showing top 5 rows
```

Se aplica un modelo Random Forest, el cual es un modelo de clasificación que implementa múltiples árboles de decisión y que finalmente hace una predicción con el conjunto de datos de prueba. Se van a tomar 80% de los datos para prueba y un 20% para pronosticar o de prueba:

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer,
VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.
labelIndexer = StringIndexer(inputCol="label",
outputCol="indexedLabel").fit(training_2)

# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated
as continuous.
featureIndexer =\
    VectorIndexer(inputCol="features", outputCol="indexedFeatures",
maxCategories=4).fit(training_2)

# Split the data into training and test sets (20% held out for
testing)
(trainingData, testData) = training_2.randomSplit([0.8, 0.2])

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel",
featuresCol="indexedFeatures", numTrees=10)

# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction",
outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and forest in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf,
labelConverter])

# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
```

```
# Make predictions, with the test data
predictions = model.transform(testData)

# Select example rows to display the prediction and the original label
predictions.select("predictedLabel", "label", "features").show(5)

# Select (prediction, true label) and compute accuracy and test error:
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction",
metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print('Accuracy =', accuracy)
print("Test Error = %g" % (1.0 - accuracy))

rfModel = model.stages[2]
print(rfModel)  # summary only
```

```
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|           0.0|  0.0|(692,[100,101,102...|
|           0.0|  0.0|(692,[121,122,123...|
|           0.0|  0.0|(692,[123,124,125...|
|           0.0|  0.0|(692,[124,125,126...|
|           0.0|  0.0|(692,[125,126,127...|
+--------------+-----+--------------------+
only showing top 5 rows

Accuracy = 0.9545454545454546
Test Error = 0.0454545
RandomForestClassificationModel:
uid=RandomForestClassifier_8dce35cb2c21, numTrees=10, numClasses=2,
numFeatures=692
```

El modelo de clasificación generado muestra que el modelo está integrado por 10 árboles, con dos clases y con 692 características en el dataset.

Respecto a los resultados del modelo, se tiene una precisión o accuracy del 95% porciento, siendo un modelo muy apto para pronosticar nuevos datos.

Ahora se procede a visualizar una parte del modelo generado, en este caso, se mostrará el primer árbol del modelo:

```
# modelo RandomForestClassifier
rfModel = model.stages[2]

# Obtención de uno de los árboles del bosque
tree0 = rfModel.trees[0]
```

```
# Mostrar la estructura del árbol
print("Estructura del primer árbol del Random Forest:")
print(tree0.toDebugString)
```

```
Estructura del primer árbol del Random Forest:
DecisionTreeClassificationModel: uid=dtc_9e763e771931, depth=2,
numNodes=5, numClasses=2, numFeatures=692
  If (feature 484 <= 6.5)
   If (feature 331 <= 7.0)
    Predict: 0.0
   Else (feature 331 > 7.0)
    Predict: 1.0
  Else (feature 484 > 6.5)
   Predict: 1.0
```

Modelo de Agrupamiento KMeans:

```
 # Cargar dataset
training_3 =
spark.read.format("libsvm").load("/content/drive/MyDrive/CONCENTRACION
AI/data/sample_kmeans_data.txt")

training_3.show(5)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|           (3,[],[])|
|  1.0|(3,[0,1,2],[0.1,0...|
|  2.0|(3,[0,1,2],[0.2,0...|
|  3.0|(3,[0,1,2],[9.0,9...|
|  4.0|(3,[0,1,2],[9.1,9...|
+-----+--------------------+
only showing top 5 rows
```

Con el modelo de clustering, se utiliza KMeans para agrupar datos, finalmente, evaluando la calidad de los clusters, entre más cercano a 1 mayor es la calidad de los clusters generados por el modelo:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(training_3)

# Make predictions
predictions = model.transform(training_3)
```

```
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " +
str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.9997530305375207
Cluster Centers:
[9.1 9.1 9.1]
[0.1 0.1 0.1]
```

De acuerdo con el modelo generado, la calidad del modelo y los clusters es casi de un 100% siendo esto que el modelo es excelente para agrupar los datos del dataset. Se muestra en el resultado que existen dos clusters formados por el modelo.

Dentro de la actividad, se pudo implementar PySpark, así como su configuración para el espacio de trabajo. Seguido de esto se pudo realizar un seguimiento de la carga de los datasets y la aplicación de los modelos de regresión, clasificación y de agrupamiento. Al aplicar los modelos, se muestran los resultados y eficiencia de cada modelo en los conjuntos de prueba, junto con una visualización de modelo generado en PySpark.