TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

EXAMEN

Carrera: Ingeniería En Sistemas Computacionales Período: Febrero-junio 2021

Materia: Datos Masivos Grupo: **6:00 pm** Salón: **6:00 pm**

Unidad (es) a evaluar: Unidad 2 Tipo de examen: Práctico Fecha:**23/05/22**

Catedrático: José Christian Romero Hernández Firma del maestro: Calificación:

Alumnos:

Lopez Higuera Saul Alfredo

Ramos Rivera Manuel Isaí

No. Control:

#18210493

#17212931

Instrucciones

Desarrollar las siguientes instrucciones en Spark con el leguaje de programación Scala, utilizando solo la documentacion de la librería de Machine Learning Mllib de Spark y Google.

1. Cargar en un dataframe Iris.csv que se encuentra en https://github.com/jcromerohdz/iris, elaborar la liempieza de datos necesaria para ser procesado por el siguiente algoritmo (Importante, esta limpieza debe ser por medio de un script de Scala en Spark) .

a. Utilice la librería Mllib de Spark el algoritmo de Machine Learning multilayer perceptron

```
///Utilice la libreria Mllib de Spark el algoritmo de Machine Learning multilayer perceptron

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.IntegerType
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors
///Cargar sesion spark
var spark = SparkSession.builder().getOrCreate()
//Cargar Iris.csv
val df = spark.read.format("csv").option("inferSchema","true").option("header","true").csv("iris.csv")
```

```
scala> import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

scala> import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.sql.types.IntegerType
import org.apache.spark.sql.types.IntegerType

scala> import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.StringIndexer

scala> import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorAssembler

scala> import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.linalg.Vectors

scala> var spark = SparkSession.builder().getOrCreate()
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@58c93be3

scala> val df = spark.read.format("csv").option("inferSchema","true").option("header","true").csv("iris.csv")
df: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 3 more fields]
```

2. ¿Cuáles son los nombres de las columnas?

```
//Cuales son los nombres de las columnas?
df.columns
```

```
scala> df.columns
res0: Array[String] = Array(sepal_length, sepal_width, petal_length, petal_width, species)
```

3. ¿Cómo es el esquema?

```
//Como es el esquema?
df.printSchema()
```

```
scala> df.printSchema()
root
 |-- sepal_length: double (nullable = true)
 |-- sepal_width: double (nullable = true)
 |-- petal_length: double (nullable = true)
 |-- petal_width: double (nullable = true)
 |-- species: string (nullable = true)
```

4. Imprime las primeras 5 columnas.

```
//Imprime las primeras 5 columnas.
df.select($"sepal_length",$"sepal_width",$"petal_length",$"petal_width",$"species").show()
```

```
scala> df.select($"sepal_length",$"sepal_width",$"petal_length",$"petal_width",$"species").show()
+------------+-----------+------------+-----------+-------+
|sepal_length|sepal_width|petal_length|petal_width|species|
+------------+-----------+------------+-----------+-------+
|         5.1|        3.5|         1.4|        0.2| setosa|
|         4.9|        3.0|         1.4|        0.2| setosa|
|         4.7|        3.2|         1.3|        0.2| setosa|
|         4.6|        3.1|         1.5|        0.2| setosa|
|         5.0|        3.6|         1.4|        0.2| setosa|
|         5.4|        3.9|         1.7|        0.4| setosa|
|         4.6|        3.4|         1.4|        0.3| setosa|
|         5.0|        3.4|         1.5|        0.2| setosa|
|         4.4|        2.9|         1.4|        0.2| setosa|
|         4.9|        3.1|         1.5|        0.1| setosa|
|         5.4|        3.7|         1.5|        0.2| setosa|
|         4.8|        3.4|         1.6|        0.2| setosa|
|         4.8|        3.0|         1.4|        0.1| setosa|
|         4.3|        3.0|         1.1|        0.1| setosa|
|         5.8|        4.0|         1.2|        0.2| setosa|
|         5.7|        4.4|         1.5|        0.4| setosa|
|         5.4|        3.9|         1.3|        0.4| setosa|
|         5.1|        3.5|         1.4|        0.3| setosa|
|         5.7|        3.8|         1.7|        0.3| setosa|
|         5.1|        3.8|         1.5|        0.3| setosa|
+------------+-----------+------------+-----------+-------+
only showing top 20 rows
```

w.facebook.com/Coordinacioninfotic

5. Usa el método describe () para aprender más sobre los datos del DataFrame.

```
//Usa el metodo describe () para aprender mas sobre los datos del DataFrame.
df.describe()
```

```
scala> df.describe()
res3: org.apache.spark.sql.DataFrame = [summary: string, sepal_length: string ... 4 more fields]
```

6. Haga la transformación pertinente para los datos categóricos los cuales serán nuestras etiquetas a clasificar.

```
//Haga la transformacion pertinente para los datos categoricos los cuales seran nuestras etiquetas a clasificar.

//Creamos un vector assambler y combinamos todas estas columnas

val Vassembler = new VectorAssembler().setInputCols(Array("sepal_length","sepal_width","petal_length","petal_width")).setOutputCol("features")
val output = Vassembler.transform(df)
output.show()

//Transformamos la columna especial en numerica y nombrandola como label
val labelIndexer = new StringIndexer().setInputCol("species").setOutputCol("label").fit(df) //special note: col must always be called label to be automatic
val indexed = labelIndexer.transform(output)
indexed.show()
```

```
scala> val Vassembler = new VectorAssembler().setInputCols(Array("sepal_length","sepal_width","petal_length","petal_width")).setOutputCol("features")
Vassembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_9dd5ddacc1b6

scala> val output = Vassembler.transform(df)
output: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 4 more fields]

scala> output.show()
+------------+-----------+------------+-----------+-------+-----------------+
|sepal_length|sepal_width|petal_length|petal_width|species|         features|
+------------+-----------+------------+-----------+-------+-----------------+
|         5.1|        3.5|         1.4|        0.2| setosa|[5.1,3.5,1.4,0.2]|
|         4.9|        3.0|         1.4|        0.2| setosa|[4.9,3.0,1.4,0.2]|
|         4.7|        3.2|         1.3|        0.2| setosa|[4.7,3.2,1.3,0.2]|
|         4.6|        3.1|         1.5|        0.2| setosa|[4.6,3.1,1.5,0.2]|
|         5.0|        3.6|         1.4|        0.2| setosa|[5.0,3.6,1.4,0.2]|
|         5.4|        3.9|         1.7|        0.4| setosa|[5.4,3.9,1.7,0.4]|
|         4.6|        3.4|         1.4|        0.3| setosa|[4.6,3.4,1.4,0.3]|
|         5.0|        3.4|         1.5|        0.2| setosa|[5.0,3.4,1.5,0.2]|
|         4.4|        2.9|         1.4|        0.2| setosa|[4.4,2.9,1.4,0.2]|
|         4.9|        3.1|         1.5|        0.1| setosa|[4.9,3.1,1.5,0.1]|
|         5.4|        3.7|         1.5|        0.2| setosa|[5.4,3.7,1.5,0.2]|
|         4.8|        3.4|         1.6|        0.2| setosa|[4.8,3.4,1.6,0.2]|
|         4.8|        3.0|         1.4|        0.1| setosa|[4.8,3.0,1.4,0.1]|
|         4.3|        3.0|         1.1|        0.1| setosa|[4.3,3.0,1.1,0.1]|
|         5.8|        4.0|         1.2|        0.2| setosa|[5.8,4.0,1.2,0.2]|
|         5.7|        4.4|         1.5|        0.4| setosa|[5.7,4.4,1.5,0.4]|
|         5.4|        3.9|         1.3|        0.4| setosa|[5.4,3.9,1.3,0.4]|
|         5.1|        3.5|         1.4|        0.3| setosa|[5.1,3.5,1.4,0.3]|
|         5.7|        3.8|         1.7|        0.3| setosa|[5.7,3.8,1.7,0.3]|
|         5.1|        3.8|         1.5|        0.3| setosa|[5.1,3.8,1.5,0.3]|
+------------+-----------+------------+-----------+-------+-----------------+
only showing top 20 rows
```

```
scala> val labelIndexer = new StringIndexer().setInputCol("species").setOutputCol("label").fit(df) //special note: col must always be called label to be automatic
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx_8b77eb1c2ebb

scala> val indexed = labelIndexer.transform(output)
indexed: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 5 more fields]

scala> indexed.show()
+------------+-----------+------------+-----------+-------+-----------------+-----+
|sepal_length|sepal_width|petal_length|petal_width|species|         features|label|
+------------+-----------+------------+-----------+-------+-----------------+-----+
|         5.1|        3.5|         1.4|        0.2| setosa|[5.1,3.5,1.4,0.2]|  2.0|
|         4.9|        3.0|         1.4|        0.2| setosa|[4.9,3.0,1.4,0.2]|  2.0|
|         4.7|        3.2|         1.3|        0.2| setosa|[4.7,3.2,1.3,0.2]|  2.0|
|         4.6|        3.1|         1.5|        0.2| setosa|[4.6,3.1,1.5,0.2]|  2.0|
|         5.0|        3.6|         1.4|        0.2| setosa|[5.0,3.6,1.4,0.2]|  2.0|
|         5.4|        3.9|         1.7|        0.4| setosa|[5.4,3.9,1.7,0.4]|  2.0|
|         4.6|        3.4|         1.4|        0.3| setosa|[4.6,3.4,1.4,0.3]|  2.0|
|         5.0|        3.4|         1.5|        0.2| setosa|[5.0,3.4,1.5,0.2]|  2.0|
|         4.4|        2.9|         1.4|        0.2| setosa|[4.4,2.9,1.4,0.2]|  2.0|
|         4.9|        3.1|         1.5|        0.1| setosa|[4.9,3.1,1.5,0.1]|  2.0|
|         5.4|        3.7|         1.5|        0.2| setosa|[5.4,3.7,1.5,0.2]|  2.0|
|         4.8|        3.4|         1.6|        0.2| setosa|[4.8,3.4,1.6,0.2]|  2.0|
|         4.8|        3.0|         1.4|        0.1| setosa|[4.8,3.0,1.4,0.1]|  2.0|
|         4.3|        3.0|         1.1|        0.1| setosa|[4.3,3.0,1.1,0.1]|  2.0|
|         5.8|        4.0|         1.2|        0.2| setosa|[5.8,4.0,1.2,0.2]|  2.0|
|         5.7|        4.4|         1.5|        0.4| setosa|[5.7,4.4,1.5,0.4]|  2.0|
|         5.4|        3.9|         1.3|        0.4| setosa|[5.4,3.9,1.3,0.4]|  2.0|
|         5.1|        3.5|         1.4|        0.3| setosa|[5.1,3.5,1.4,0.3]|  2.0|
|         5.7|        3.8|         1.7|        0.3| setosa|[5.7,3.8,1.7,0.3]|  2.0|
|         5.1|        3.8|         1.5|        0.3| setosa|[5.1,3.8,1.5,0.3]|  2.0|
+------------+-----------+------------+-----------+-------+-----------------+-----+
only showing top 20 rows
```

7. Construya el modelo de clasificación y explique su arquitectura.

```scala
//Construya el modelo de clasificacion y explique su arquitectura.

//dividira en un 70% de entrenamiento y un 30% de prueba
val splits = indexed.randomSplit(Array(0.7, 0.3), seed = 1234L)
val train = splits(0)
val test = splits(1)

//Arreglo de las capas de la red neuronal,en este caso escogemos ciertos valores del mismo arreglo de la capa ya mencionada
val layers = Array[Int](4, 4, 4, 3)

// Creacion del modelo de entrenamiento
val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L).setMaxIter(100)

// Variable para el modelo de entrenamiento
val model = trainer.fit(train)

// Valores de la precision
val result = model.transform(test)
val predictionAndLabels = result.select("prediction", "label")
val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
```

```
scala> val splits = indexed.randomSplit(Array(0.7, 0.3), seed = 1234L)
splits: Array[org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]] = Array([sepal_length: double, sepal_width: double ... 5 more fields], [sepal_length: double, sepal_width: double ... 5 more fields])

scala> val train = splits(0)
train: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [sepal_length: double, sepal_width: double ... 5 more fields]

scala> val test = splits(1)
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [sepal_length: double, sepal_width: double ... 5 more fields]

scala> val layers = Array[Int](4, 4, 4, 3)
layers: Array[Int] = Array(4, 4, 4, 3)

scala> val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L).setMaxIter(100)
trainer: org.apache.spark.ml.classification.MultilayerPerceptronClassifier = mlpc_902199773270

scala> val model = trainer.fit(train)
22/05/23 19:29:52 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/05/23 19:29:52 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
22/05/23 19:29:54 ERROR StrongWolfeLineSearch: Encountered bad values in function evaluation. Decreasing step size to 0.5
22/05/23 19:29:54 ERROR StrongWolfeLineSearch: Encountered bad values in function evaluation. Decreasing step size to 0.25
22/05/23 19:29:54 ERROR StrongWolfeLineSearch: Encountered bad values in function evaluation. Decreasing step size to 0.125
model: org.apache.spark.ml.classification.MultilayerPerceptronClassificationModel = mlpc_902199773270

scala> val result = model.transform(test)
result: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 8 more fields]

scala> val predictionAndLabels = result.select("prediction", "label")
predictionAndLabels: org.apache.spark.sql.DataFrame = [prediction: double, label: double]

scala> val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_f9c0634197f1
```

8. Imprima los resultados del modelo

```scala
// Imprime los valores de precision
println(s"Test set accuracy = ${evaluator.evaluate(predictionAndLabels)}")

//Muestra la distribucion de los datos
println(s"train: ${train.count}, test: ${test.count()}")
//Mostrar el valor real de la tabla frente a la prediccion
result.select("features", "label", "prediction").show(test.count().asInstanceOf[Int])
```

```
scala> println(s"Test set accuracy = ${evaluator.evaluate(predictionAndLabels)}")
Test set accuracy = 1.0

scala> println(s"train: ${train.count}, test: ${test.count()}")
train: 110, test: 40

scala> result.select("features", "label", "prediction").show(test.count().asInstanceOf[Int])
+-----------------+-----+----------+
|         features|label|prediction|
+-----------------+-----+----------+
|[4.3,3.0,1.1,0.1]|  2.0|       2.0|
|[4.4,2.9,1.4,0.2]|  2.0|       2.0|
|[4.4,3.0,1.3,0.2]|  2.0|       2.0|
|[4.8,3.1,1.6,0.2]|  2.0|       2.0|
|[5.0,3.3,1.4,0.2]|  2.0|       2.0|
|[5.0,3.4,1.5,0.2]|  2.0|       2.0|
|[5.0,3.6,1.4,0.2]|  2.0|       2.0|
|[5.1,3.4,1.5,0.2]|  2.0|       2.0|
|[5.1,3.8,1.5,0.3]|  2.0|       2.0|
|[5.2,2.7,3.9,1.4]|  0.0|       0.0|
|[5.2,4.1,1.5,0.1]|  2.0|       2.0|
|[5.3,3.7,1.5,0.2]|  2.0|       2.0|
|[5.4,3.4,1.5,0.4]|  2.0|       2.0|
|[5.5,2.3,4.0,1.3]|  0.0|       0.0|
|[5.6,2.9,3.6,1.3]|  0.0|       0.0|
|[5.7,2.5,5.0,2.0]|  1.0|       1.0|
|[5.8,2.7,3.9,1.2]|  0.0|       0.0|
|[5.8,2.8,5.1,2.4]|  1.0|       1.0|
|[5.8,4.0,1.2,0.2]|  2.0|       2.0|
|[5.9,3.0,5.1,1.8]|  1.0|       1.0|
|[6.0,2.2,4.0,1.0]|  0.0|       0.0|
|[6.0,2.9,4.5,1.5]|  0.0|       0.0|
|[6.0,3.4,4.5,1.6]|  0.0|       0.0|
|[6.1,2.6,5.6,1.4]|  1.0|       1.0|
|[6.1,2.8,4.0,1.3]|  0.0|       0.0|
|[6.1,3.0,4.9,1.8]|  1.0|       1.0|
|[6.2,2.8,4.8,1.8]|  1.0|       1.0|
|[6.2,2.9,4.3,1.3]|  0.0|       0.0|
|[6.3,3.3,4.7,1.6]|  0.0|       0.0|
|[6.3,3.4,5.6,2.4]|  1.0|       1.0|
|[6.5,3.0,5.2,2.0]|  1.0|       1.0|
|[6.5,3.0,5.5,1.8]|  1.0|       1.0|
|[6.7,3.0,5.2,2.3]|  1.0|       1.0|
|[6.7,3.1,5.6,2.4]|  1.0|       1.0|
|[6.8,3.0,5.5,2.1]|  1.0|       1.0|
|[6.9,3.1,4.9,1.5]|  0.0|       0.0|
|[6.9,3.1,5.1,2.3]|  1.0|       1.0|
|[7.0,3.2,4.7,1.4]|  0.0|       0.0|
|[7.3,2.9,6.3,1.8]|  1.0|       1.0|
+-----------------+-----+----------+
```

Instrucciones de evaluación

- Tiempo de entrega 4 dias
- Al terminar poner el código y la explicación en la rama (branch) correspondiete de su github asi mismo realizar su explicación de la solución en su google drive.
- Finalmente defender su desarrollo en un video de 6-8 min el cual servira para dar su calificación, este video debe subirse a youtube para ser compartido por un link público.