TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO

TECNOLÓGICO DE TIJUANA
SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE FEBRERO-JUNIO 2022

CARRERA

Ingeniería en informática e Ingeniería en Sistemas

Computacionales

MATERIA
Datos masivos
TÍTULO
Práctica evaluatoria, unidad #2
Integrantes:

Munguía Silva Edgar Geovanny #17212344
Perez López Alicia Guadalupe ##18210514

NOMBRE DEL MAESTRO
Jose Christian Romero Hernadez

Tijuana Baja California 23 de mayo del 2022

Introducción.
En esta práctica, usaremos un algoritmo de aprendizaje automático llamado perceptrón multicapa(multilayer perceptron), usaremos sus bibliotecas para realizar con éxito la práctica de evaluación de la unidad 2 del curso de big data(datos masivos).

Código.

Importando las librerías necesarias.

```
##Import necessary libraries

import
org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.{VectorAssembler,
StringIndexer}
import org.apache.spark.ml.linalg.Vectors
```

```
scala> import org.apache.spark.ml.classification.MultilayerPerceptro
import org.apache.spark.ml.classification.MultilayerPerceptronClassi

scala> import org.apache.spark.ml.evaluation.MulticlassClassificatio
import org.apache.spark.ml.evaluation.MulticlassClassificationEvalua

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

scala> import org.apache.spark.ml.feature.{VectorAssembler, StringIn
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}

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

Cargar el data frame (proporcionado por el maestro).

```
## Load dataframe (provided by the teacher)

val
```

```
csvfile=spark.read.format("csv").option("header","true").option("inferSchema", "true").load("iris.csv")
```

Limpiar los datos.

```
## Clean data
//Nombrar nuestro dataframe
val Clean =csvfile.na.drop()
```

Mostrar los nombres de las columnas.

```
##Show columns name

Clean.columns
```

Mostrar el esquema.

```
## show schema
Clean.printSchema
```

Imprimir las primeras 5 columnas.

```
##print the first 5 columns

Clean.show(5)
```

Usar el método describe() para aprender más acerca de los datos en el dataframe.

```
##Use the describe() method to learn more about the data in
the DataFrame.

Clean.describe().show
```

Hacer las transformaciones pertinentes para los datos categóricos, los cuales serán nuestras etiquetas a clasificar.

```
##Make the pertinent transformation for the categorical data
which will be
our labels to classify.
```

```
val labelIndexer = new StringIndexer().setInputCol("species").setOutputCol("indexedLabel").fit(Clean)

//sustituir la columna "species" con nuestra columna "indexedLabel" y la vamos a mostrar con el nombre de "label"

val indexed = labelIndexer.transform(Clean).drop("species").withColumnRenamed("indexedLabel", "label")
indexed.describe().show()
```

```
scala> val indexed = labelIndexer.transform(Clean).drop("species").withColumnRenamed("indexedLabel", "label")
indexed: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 3 more fields]

scala> indexed.describe().show()
+-------+------------------+-------------------+------------------+------------------+------------------+
|summary|      sepal_length|        sepal_width|      petal_length|       petal_width|             label|
+-------+------------------+-------------------+------------------+------------------+------------------+
|  count|               150|                150|               150|               150|               150|
|   mean| 5.843333333333335| 3.0540000000000007|3.7586666666666693|1.1986666666666672|               1.0|
| stddev|0.8280661279778637|0.43359431136217375| 1.764420419952262|0.7631607417008414|0.8192319205190403|
|    min|               4.3|                2.0|               1.0|               0.1|               0.0|
|    max|               7.9|                4.4|               6.9|               2.5|               2.0|
+-------+------------------+-------------------+------------------+------------------+------------------+
```

Se unen las columnas en una sola, la cual se llamará “features”.

```
// Uniremos las columnas en una sola, la cual llamaremos features

val assembler = new VectorAssembler().setInputCols(Array("sepal_length","sepal_width","petal_length","petal_width")).setOutputCol("features")
val  features = assembler.transform(indexed)
val labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(indexed)
println(s"Found labels: ${labelIndexer.labels.mkString("[", ", ", "]")}")

features.show
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx_80c3cb92333e

scala> println(s"Found labels: ${labelIndexer.labels.mkString("[", ", ", "]")}")
Found labels: [1.0, 0.0, 2.0]

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

Se construye el modelo de clasificación y se explica su arquitectura.

```
##Build the classification model and explain its architecture

vamos a separar el dataset en 30% en datos de prueba y un 70% en datos de entrenamiento establecemos la semilla de aleatoriedad
val splits = features.randomSplit(Array(0.7, 0.3), seed = 1234L)
val train = splits(0)
val test = splits(1)
val layers = Array[Int](4, 5, 4, 3)
val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L).setMaxIter(100)
val model = trainer.fit(train)
val result = model.transform(test)
```

Se imprimen los resultados del modelo.

```
##Print the model results
val predictionAndLabels = result.select("prediction",
"label")
predictionAndLabels.show
```

```
scala> predictionAndLabels.show
+----------+-----+
|prediction|label|
+----------+-----+
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       0.0|  0.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       0.0|  0.0|
|       0.0|  0.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       1.0|  1.0|
|       2.0|  2.0|
|       1.0|  1.0|
+----------+-----+
only showing top 20 rows
```

```
val evaluator = new
MulticlassClassificationEvaluator().setMetricName("accuracy")
println(s"Test set accuracy =
${evaluator.evaluate(predictionAndLabels)}")
```

Y finalmente, obtenemos un test de precisión de **0.95**.

```
scala> println(s"Test set accuracy = ${evaluator.evaluate(predictionAndLabels)}")
Test set accuracy = 0.95
```

Conclusions.

Edgar Munguia: In this unit, we learned more about machine learning and some of the algorithms using this technology. Machine learning is the main topic in technology in the present years because it represents the future of technology and data. We, as future data scientists, we will use this kind of algorithms to make our live easier when working with this kind of data, so, in conclusion, we can say, that machine learning its a powerful tool for the treatment of data, and of course, it will bring a lot of benefits for us, as data scientists.

Alicia Perez: This evaluative practice, in my opinion, was a little more complex since it contained the topics that were presented in class, and the practices based on them, if it was a small challenge and the truth is that it is a bit difficult for me to transmit what I recently learned, but achieved.

Link video (Youtube): *https://www.youtube.com/watch?v=esK8jToP6E0*
GitHub: *https://github.com/Aliciap26/DATOS-MASIVOS/tree/unit_2*