


National Technological Institute of Mexico
Technological Institute of Tijuana

Academic Subdirectorate

Systems and computing department

February-June 2021

Big Data

Unit II
Evaluative practice

Salazar Ibarra Jesus Rodrigo16212542

Juan Daniel Camacho Manabe17210534

Teacher
JOSE CHRISTIAN ROMERO HERNANDEZ

Pevaluative practice

Pull request: https://github.com/ElsellamaJesus/BigData/pull/3

Video: https://youtu.be/aQ-J9HckjKo

Instructions

- 1. Load into an Iris.csv dataframe found in https://github.com/jcromerohdz/iris, elaborate the necessary data to be processed by the following algorithm (Important, this cleaning must be through a Scala script in Spark).
- to. Use the Spark Mllib library the Machine Learning algorithm corresponding to multilayer perceptron
- 2. What are the names of the columns?
- 3. What is the scheme like?
- 4. Print the first 5 columns.
- 5. Use the describe () method to learn more about the data in the DataFrame.
- 6. Make the transformationorn pertinent for categorical data which will be our labels to be classified.
- 7. Build the classification modelorny explain its architecture.
- 8. Print the model results

Evaluation instructions

- Delivery time 4 days

- When finished, put the code and the explanation in the corresponding branch of your github, and also make your explanation of the solution in your google drive. - Finally defend its development in a video of 8-10 min which will serve to give your rating, this video must be uploaded to YouTube to be shared by a public link.

	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

Fig 1. Dataframe Iris

Code

Answer what is requested in the instructions with Spark-DataFrames using the file "Iris.csv".

1. We import the necessary libraries for the cleaning, analysis and interpretation of the data.

```
// Import of 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
import org.apache.spark.ml.Pipeline
import org.apache.spark.sql.SparkSession
```

2. Start a simple Spark session.

```
val spark = SparkSession.builder.appName
("MultilayerPerceptronClassifierExample") .getOrCreate ()
```

Explanation: At point 2, you have to start a session in Spark, for which we already import the sql library to start the session, finally we assign the created session to a variable.

```
scala> val spark = SparkSession.builder.appName("MultilayerPerceptronClassifierExample").getOrCreate()
21/06/01 03:26:38 WARN SparkSession$Builder: Using an existing SparkSession; some spark core configurations may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@499f9003
```

3. Load the file iris.csv Load into a dataframe Iris.csv, (https://github.com/jcromerohdz/BigData/blob/master/Spark_DataFrame/ContainsNull.scala)

```
val df = spark.read.option ("header","true") .option ("inferSchema",
"true") .format ("csv") .load ("/home/js/Desk/examen/iris.csv")
```

Explanation: In point 3, we append the dataframe to use, assign it to a variable that we call "df (dataframe)" and infer the data. It should be noted that the csv file was not in the same folder as the scala file, so the path was specified.

```
scala> val df = spark.read.option("header","true").option("inferSchema", "true").format("csv").load("/home/daniel-camacho/Escritorio/Iris.csv"
)
df: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 3 more fields]
```

4. We clean the data

```
val data = df.na.drop ()
```

Explanation: In point 4, we use the "na.drop" function to eliminate the rows that contain null values, since if they have no value they will only cause the use of more resources and errors throughout the analysis.

```
scala> val data = df.na.drop()
data: org.apache.spark.sql.DataFrame = [sepal_length: double, sepal_width: double ... 3 more fields]
```

5. What are the names of the columns?

```
data.columns
```

Explanation: At point 5, with a property called ".columns", we only called the names of the columns of our dataframe.

```
res0: Array[String] = Array(sepal_length, sepal_width, petal_length, petal_width, species)
```

6. What is the scheme like?

```
data.printSchema ()
```

Explanation: In point 6, with the function ".printSchema ()" we show a table with the complete panorama of the dataframe.

```
scala> data.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)
```

7. Print the first 5 columns.

```
data.show (5)
```

Explanation: In point 7, with the function ".show ()", we show the first values of the dataframe, which in this one are the first 5.

```
scala> data.show(5)
+------------+-----------+------------+-----------+-------+
|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|
+------------+-----------+------------+-----------+-------+
only showing top 5 rows
```

8. Describe the nature of the data

```
data.describe.show ()
```

Explanation: In point 8, with the function ".describe ()", we show the characteristics of the data.

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

9. We make the pertinent transformation for the categorical data which will be our labels to be classified.

```
 val assembler = new VectorAssembler() .setInputCols
(Array("sepal_length","sepal_width","petal_length","petal_width")).
setOutputCol ("features")
```

Explanation: At point 9, we have to convert the selected columns into a vector, which will be our characteristics.

```
scala> val assembler = new VectorAssembler().setInputCols(Array("sepal_length","sepal_width","petal_length","petal_width")).setOutputCol("Feat
ures")
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_cae526d333ba
```

10. Define output data

```
 val output = assembler.transform (data)
```

Explanation: At point 10, we have to transform the features using the dataframe.

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

11. Index data

```
 val indexer = new StringIndexer() .setInputCol ("species")
.setOutputCol ("label")
 val indexed = indexer.fit (output) .transform (output)
```

Explanation: At point 11, we create indexes of the species column that is transformed into numerical data and we fit with the output vector “output”.

```
scala>  val indexer = new StringIndexer().setInputCol("species").setOutputCol("label")
indexer: org.apache.spark.ml.feature.StringIndexer = strIdx_d99239ac24f1

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

12. Build the classification model and explain the architecture.

```
val splits = indexed.randomSplit (Array(0.6, 0.4), seed = 1234L)
val train = splits (0)
val test = splits (1)
```

Explanation: At point 12, we have to randomly divide the data into training "train 60% (0.6)" and test "test 40% (0.4)", then we assign variables to the selected data.

```
scala> val splits = indexed.randomSplit(Array(0.6, 0.4), 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]
```

13. We create the neural network

```
val layers = Array[Int] (4, 5, 4, 3)


val trainer = new MultilayerPerceptronClassifier() .setLayers (layers)
.setBlockSize (128) .setSeed (1234L) .setMaxIter (100)
```

Explanation: In point 13, we define the stages of the neural network, four input layer

elements, two cultured layers, one of five and one of four elements respectively, and three output layer elements. Next we assign the training data, the model with which we will work in this case, we were asked to use “Multilayer Perceptron Classifier”, the neural network where we will train, the size of the data blocks, a randomness seed and the maximum number of iterations in the neural network.

```
scala>  val layers = Array[Int](4, 5, 4, 3)
layers: Array[Int] = Array(4, 5, 4, 3)

scala>  val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L).setMaxIter(100)
trainer: org.apache.spark.ml.classification.MultilayerPerceptronClassifier = mlpc_a0b6e702dbfa
```

14. Train the model

```
val model = trainer.fit (train)
val result = model.transform (test)
val predictionAndLabels = result.select ("prediction", "label")
```

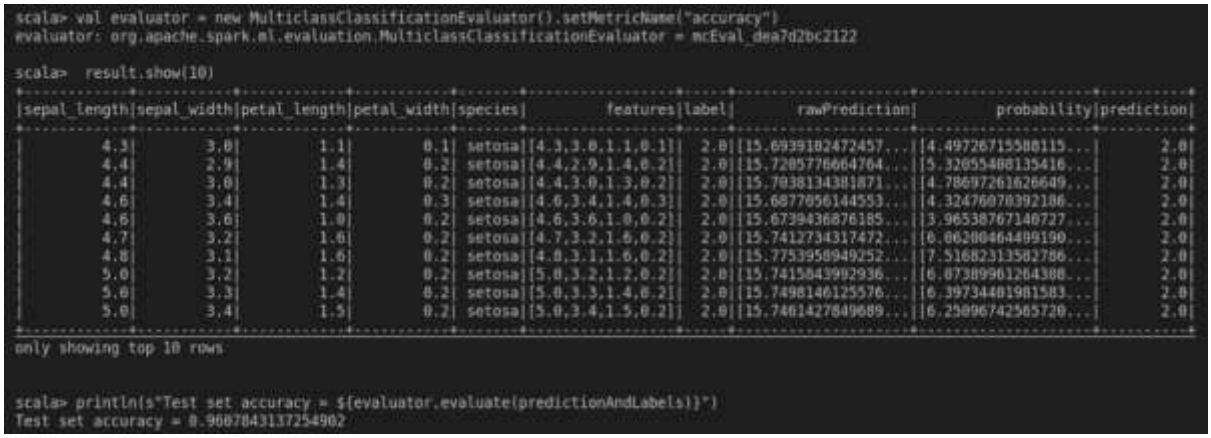
Explanation: In point 14, Train the model with the data selected for training. Evaluate the training result using the same model but with the test data, and print the prediction results vs. those we already had in the dataframe.

```
scala> val model = trainer.fit(train)
21/06/01 03:38:46 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
21/06/01 03:38:46 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
model: org.apache.spark.ml.classification.MultilayerPerceptronClassificationModel = mlpc_a0b6e702dbfa

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]
```

15. Evaluate the model

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

Explanation: In point 15, Determine a metric to evaluate the model, that is, the precision. Print the first 50 rows of the resulting data frame. And finally, print the data classification precision percentage.



```
scala> val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_dea7d2bc2122

scala>  result.show(10)
+------------+-----------+------------+-----------+-------+-----------------+-----+--------------------+--------------------+----------+
|sepal_length|sepal_width|petal_length|petal_width|species|         features|label|       rawPrediction|         probability|prediction|
+------------+-----------+------------+-----------+-------+-----------------+-----+--------------------+--------------------+----------+
|         4.3|        3.0|         1.1|        0.1| setosa|[4.3,3.0,1.1,0.1]|  2.0|[15.6939182472457...|[4.49726715588115...|       2.0|
|         4.4|        2.9|         1.4|        0.2| setosa|[4.4,2.9,1.4,0.2]|  2.0|[15.7285770664764...|[5.32055408135416...|       2.0|
|         4.4|        3.0|         1.3|        0.2| setosa|[4.4,3.0,1.3,0.2]|  2.0|[15.7038134381871...|[4.78697261626649...|       2.0|
|         4.6|        3.4|         1.4|        0.3| setosa|[4.6,3.4,1.4,0.3]|  2.0|[15.6877056144553...|[4.32476078392186...|       2.0|
|         4.6|        3.6|         1.0|        0.2| setosa|[4.6,3.6,1.0,0.2]|  2.0|[15.6739436876185...|[3.96538767140727...|       2.0|
|         4.7|        3.2|         1.6|        0.2| setosa|[4.7,3.2,1.6,0.2]|  2.0|[15.7412734317472...|[6.06200464499190...|       2.0|
|         4.8|        3.1|         1.6|        0.2| setosa|[4.8,3.1,1.6,0.2]|  2.0|[15.7753958949252...|[7.51682313582786...|       2.0|
|         5.0|        3.2|         1.2|        0.2| setosa|[5.0,3.2,1.2,0.2]|  2.0|[15.7415843992936...|[6.87389961264308...|       2.0|
|         5.0|        3.3|         1.4|        0.2| setosa|[5.0,3.3,1.4,0.2]|  2.0|[15.7498146125576...|[6.39734401981583...|       2.0|
|         5.0|        3.4|         1.5|        0.2| setosa|[5.0,3.4,1.5,0.2]|  2.0|[15.7461427849689...|[6.25096742585720...|       2.0|
+------------+-----------+------------+-----------+-------+-----------------+-----+--------------------+--------------------+----------+
only showing top 10 rows


scala> println(s"Test set accuracy = ${evaluator.evaluate(predictionAndLabels)}")
Test set accuracy = 0.9607843137254902
```

Conclution

We have learned the basics of classification, the main algorithms and how to adjust them according to the problem we face. There is always the risk of overtraining or biases where the data has a large difference between one and the other.