EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

National Technological Institute of Mexico
Technological Institute of Tijuana

ACADEMIC SUBDIRECTION
Systems and Computing Department

SEMESTER
August - December 2021

CARRER
Information and Communication Technologies Engineer

SUBJECT AND KEY:
Big Data BDD-1704TI9A

STUDENT'S NAME AND REGISTRATION:
Velázquez Farrera César Alejandro 17212937

NAME OF THE JOB:
Evaluative Practice - Unit 2

UNIT TO BE EVALUATED
Unit II

TEACHER'S NAME:
M.C. José Christian Romero Hernández

Tijuana, Baja California **Due Date:** Noviembre 19, 2021

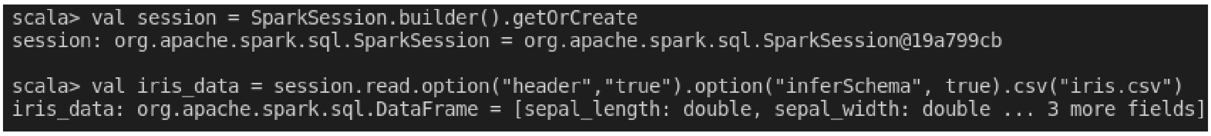
Introduction

In this evaluative practice, we will try to create a machine learning model that can predict the species type of a flower based on its characteristics.

1. Load into a dataframe Iris.csv found in https://github.com/jcromerohdz/iris, ework on cleaning the data necessary to be processed by the following algorithm (**Important, this cleaning must be done by middle of a Scala script in Spark**)

    a. Use Spark Mllib library Machine Learning algorithm **multilayer perceptron**

```
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.IndexToString
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.Pipeline

val session = SparkSession.builder().getOrCreate

val iris_data = session.read.option("header","true").option("inferSchema",
true).csv("iris.csv")
```

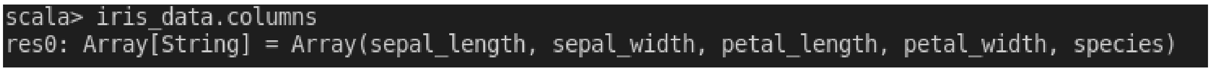


```
scala> val session = SparkSession.builder().getOrCreate
session: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@19a799cb

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

2. What are the names of the columns?

```
iris_data.columns
```

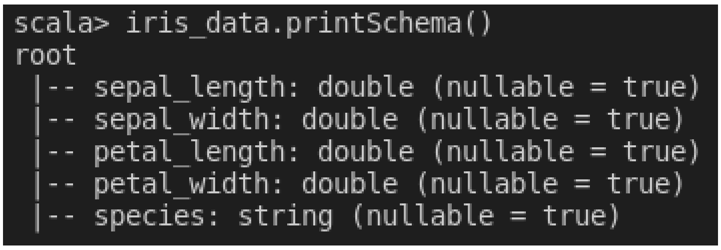
sepal_length, sepal_width, petal_length, petal_width, species



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

3. How is the scheme?

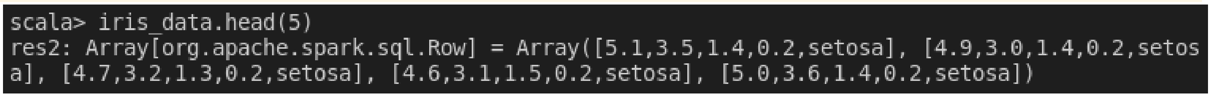
```
//getting to know the dataset
iris_data.printSchema()
```



```
scala> iris_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)
```

4. Print the first 5 columns.

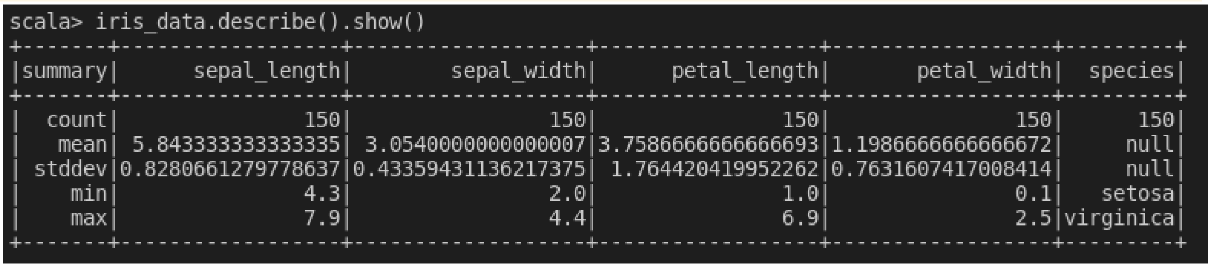
```
//Printing the first five rows of data
iris_data.head(5)
```



```
scala> iris_data.head(5)
res2: Array[org.apache.spark.sql.Row] = Array([5.1,3.5,1.4,0.2,setosa], [4.9,3.0,1.4,0.2,setos
a], [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. Use the describe () method to learn more about the data in the DataFrame.

```
//getting to know the dataset
iris_data.printSchema()
```



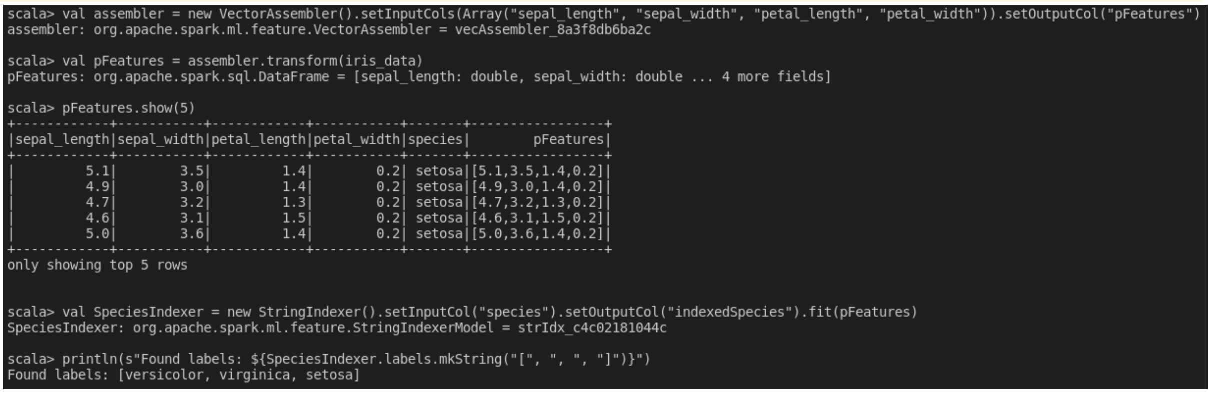
```
scala> iris_data.describe().show()
+-------+------------------+-------------------+------------------+------------------+---------+
|summary|      sepal_length|        sepal_width|      petal_length|       petal_width|  species|
+-------+------------------+-------------------+------------------+------------------+---------+
|  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|
+-------+------------------+-------------------+------------------+------------------+---------+
```

6. Make the pertinent transformation for the categorical data which will be our labels to be classified.

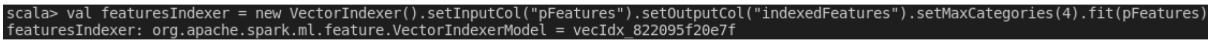
```
//Setting the input columns to a single one as pFeatures
val assembler = new VectorAssembler().setInputCols(Array("sepal_length",
"sepal_width", "petal_length", "petal_width")).setOutputCol("pFeatures")
val pFeatures = assembler.transform(iris_data)
pFeatures.show(5)

//Indexing the labels (species)
val SpeciesIndexer = new
StringIndexer().setInputCol("species").setOutputCol("indexedSpecies").fit(pF
eatures)
println(s"Found labels: ${SpeciesIndexer.labels.mkString("[", ", ", "]")}")

//Indexing the features
```

```scala
val featuresIndexer = new VectorIndexer().setInputCol("pFeatures").setOutputCol("indexedFeatures").setMaxCategories(4).fit(pFeatures)

//Split the dataset into two parts, one for training set and another for testing.
val splits = pFeatures.randomSplit(Array(0.7, 0.3))
val train = splits(0)
val test = splits(1)
```

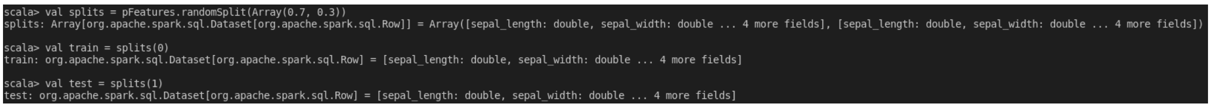
```scala
val layers = Array[Int](4,5,4,3)
```



```
scala> val assembler = new VectorAssembler().setInputCols(Array("sepal_length", "sepal_width", "petal_length", "petal_width")).setOutputCol("pFeatures")
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_8a3f8db6ba2c

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

scala> pFeatures.show(5)
+------------+-----------+------------+-----------+-------+-----------------+
|sepal_length|sepal_width|petal_length|petal_width|species|        pFeatures|
+------------+-----------+------------+-----------+-------+-----------------+
|         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]|
+------------+-----------+------------+-----------+-------+-----------------+
only showing top 5 rows


scala> val SpeciesIndexer = new StringIndexer().setInputCol("species").setOutputCol("indexedSpecies").fit(pFeatures)
SpeciesIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx_c4c02181044c

scala> println(s"Found labels: ${SpeciesIndexer.labels.mkString("[", ", ", "]")}")
Found labels: [versicolor, virginica, setosa]
```



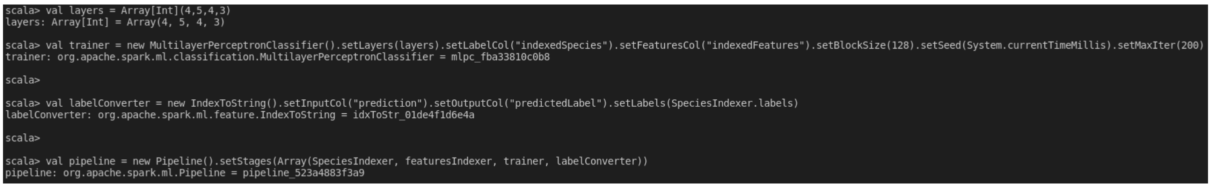
```
scala> val featuresIndexer = new VectorIndexer().setInputCol("pFeatures").setOutputCol("indexedFeatures").setMaxCategories(4).fit(pFeatures)
featuresIndexer: org.apache.spark.ml.feature.VectorIndexerModel = vecIdx_822095f20e7f
```



```
scala> val splits = pFeatures.randomSplit(Array(0.7, 0.3))
splits: Array[org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]] = Array([sepal_length: double, sepal_width: double ... 4 more fields], [sepal_length: double, sepal_width: double ... 4 more fields])

scala> val train = splits(0)
train: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [sepal_length: double, sepal_width: double ... 4 more fields]

scala> val test = splits(1)
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [sepal_length: double, sepal_width: double ... 4 more fields]
```

7. Build the classification model and explain its architecture.

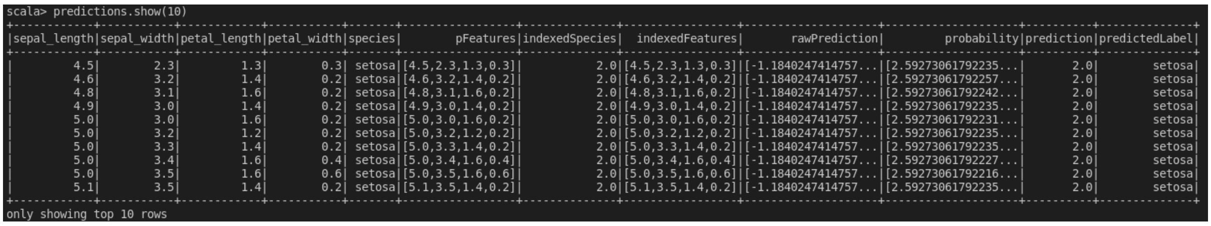
```scala
//Training the trainer
val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("indexedSpecies").setFeaturesCol("indexedFeatures").setBlockSize(128).setSeed(System.currentTimeMillis).setMaxIter(200)

val labelConverter = new IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(SpeciesIndexer.labels)

val pipeline = new Pipeline().setStages(Array(SpeciesIndexer, featuresIndexer, trainer, labelConverter))
val model = pipeline.fit(train)
```



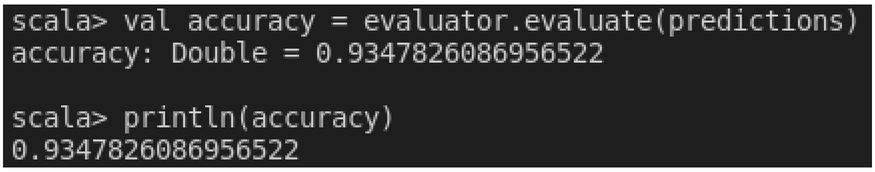
```
scala> val layers = Array[Int](4,5,4,3)
layers: Array[Int] = Array(4, 5, 4, 3)

scala> val trainer = new MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("indexedSpecies").setFeaturesCol("indexedFeatures").setBlockSize(128).setSeed(System.currentTimeMillis).setMaxIter(200)
trainer: org.apache.spark.ml.classification.MultilayerPerceptronClassifier = mlpc_fba33810c0b8

scala>

scala> val labelConverter = new IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(SpeciesIndexer.labels)
labelConverter: org.apache.spark.ml.feature.IndexToString = idxToStr_01de4f1d6e4a

scala>

scala> val pipeline = new Pipeline().setStages(Array(SpeciesIndexer, featuresIndexer, trainer, labelConverter))
pipeline: org.apache.spark.ml.Pipeline = pipeline_523a4883f3a9
```

8. Print the results of the model.

```
val predictions = model.transform(test)
predictions.show(10)

val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedSpecies").setPredict
ionCol("prediction").setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)
println(accuracy)
```



```
scala> predictions.show(10)
+------------+-----------+------------+-----------+-------+-----------------+--------------+-----------------+--------------------+--------------------+----------+--------------+
|sepal_length|sepal_width|petal_length|petal_width|species|        pFeatures|indexedSpecies|  indexedFeatures|       rawPrediction|         probability|prediction|predictedLabel|
+------------+-----------+------------+-----------+-------+-----------------+--------------+-----------------+--------------------+--------------------+----------+--------------+
|         4.5|        2.3|         1.3|        0.3| setosa|[4.5,2.3,1.3,0.3]|           2.0|[4.5,2.3,1.3,0.3]|[-1.1840247414757...|[2.59273061792235...|       2.0|        setosa|
|         4.6|        3.2|         1.4|        0.2| setosa|[4.6,3.2,1.4,0.2]|           2.0|[4.6,3.2,1.4,0.2]|[-1.1840247414757...|[2.59273061792257...|       2.0|        setosa|
|         4.8|        3.1|         1.6|        0.2| setosa|[4.8,3.1,1.6,0.2]|           2.0|[4.8,3.1,1.6,0.2]|[-1.1840247414757...|[2.59273061792242...|       2.0|        setosa|
|         4.9|        3.0|         1.4|        0.2| setosa|[4.9,3.0,1.4,0.2]|           2.0|[4.9,3.0,1.4,0.2]|[-1.1840247414757...|[2.59273061792235...|       2.0|        setosa|
|         5.0|        3.0|         1.6|        0.2| setosa|[5.0,3.0,1.6,0.2]|           2.0|[5.0,3.0,1.6,0.2]|[-1.1840247414757...|[2.59273061792231...|       2.0|        setosa|
|         5.0|        3.2|         1.2|        0.2| setosa|[5.0,3.2,1.2,0.2]|           2.0|[5.0,3.2,1.2,0.2]|[-1.1840247414757...|[2.59273061792235...|       2.0|        setosa|
|         5.0|        3.3|         1.4|        0.2| setosa|[5.0,3.3,1.4,0.2]|           2.0|[5.0,3.3,1.4,0.2]|[-1.1840247414757...|[2.59273061792235...|       2.0|        setosa|
|         5.0|        3.4|         1.6|        0.4| setosa|[5.0,3.4,1.6,0.4]|           2.0|[5.0,3.4,1.6,0.4]|[-1.1840247414757...|[2.59273061792227...|       2.0|        setosa|
|         5.0|        3.5|         1.6|        0.6| setosa|[5.0,3.5,1.6,0.6]|           2.0|[5.0,3.5,1.6,0.6]|[-1.1840247414757...|[2.59273061792216...|       2.0|        setosa|
|         5.1|        3.5|         1.4|        0.2| setosa|[5.1,3.5,1.4,0.2]|           2.0|[5.1,3.5,1.4,0.2]|[-1.1840247414757...|[2.59273061792235...|       2.0|        setosa|
+------------+-----------+------------+-----------+-------+-----------------+--------------+-----------------+--------------------+--------------------+----------+--------------+
only showing top 10 rows
```



```
scala> val accuracy = evaluator.evaluate(predictions)
accuracy: Double = 0.9347826086956522

scala> println(accuracy)
0.9347826086956522
```

Conclusión

The Perceptron Multilayer classifier is an easy model to use, as you only need to specify the object's classes (features) and species, split the data, and train a model. Based on the model, we can see that it predicts 93.47% of the data correctly. What makes this model reliable when you want to make other predictions.