NATIONAL TECHNOLOGICAL INSTITUTE OF MEXICO
TECHNOLOGICAL INSTITUTE OF TIJUANA

ACADEMIC SUBDIRECTION
Systems and Computing Department

SEMESTER
August - December 2021

ACADEMIC CAREER
Eng. Information and Communication Technologies

SUBJECT AND KEY:
Big Data BDD-1704TI9A

STUDENT’S NAME AND REGISTRATION:
Castillo Ramirez Guadalupe 17213043
Velázquez Farrera César Alejandro 17212937

NAME OF THE JOB:
Practice #1

UNIT:
Unit III

NAME OF THE TEACHER:
M.C. José Christian Romero Hernández

Practice Instructions:

Analyze and document in its corresponding branch of your repository, the regression code Logistics in Spark, this code can be found in the link I provide.

In this practice, we will work with a set of false advertising data, this practice seeks to indicate whether a particular user clicked on an ad. An attempt will be made to create a supervised machine learning model using Logistic Regression, where an attempt is made to predict whether a user clicked on an ad, tracking user characteristics.

‘Daily Time Spent on Site’: Time spent on the site per user.
‘Age’: Age of the client.
‘Area Income’: Average earnings of the geographic area.
‘Daily Internet Usage’: Average time spent on the internet by the user
‘Ad Topic Line’: Ad title
‘City’: City of residence
‘Male’: Indicates if the consumer is male
‘Country’: Country of residence of the user
‘Timestamp’: Time at which the consumer clicked on the ad or on the window is closed.
'Clicked on Ad': 0 or 1, indicating if you clicked on the ad.

To initialize this practice, it is necessary to load the data into memory. For this, it’s necessary to open a Spark session in a terminal. In this case, the Spark-Shell 2.4.7 environment is used on the Pop OS! operating system, the same as Ubuntu 21.04. To initialize a session on Linux the command is as follows:

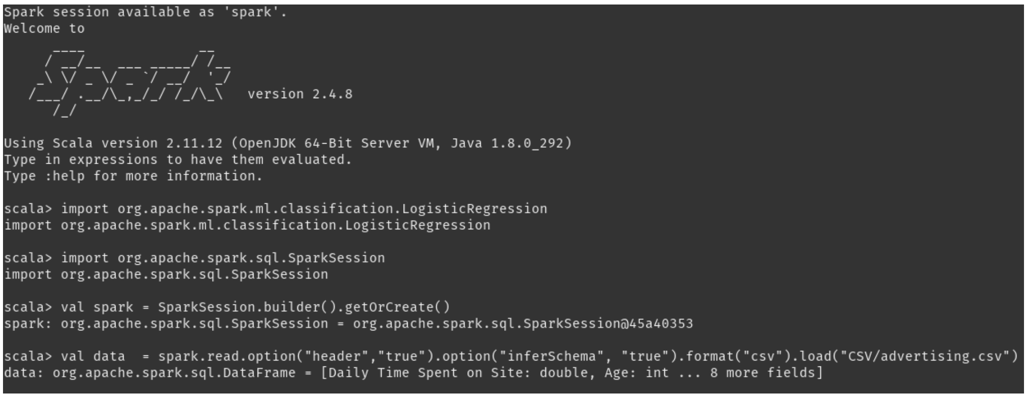
```
spark-shell
```

In the context of supervised machine learning, logistic regression is used to predict the categories of a data set. Classification consists of visualizing the data and assigning the classes (or labels) to assign them. To load the data into memory and the logistic regression model, it is necessary to invoke the following libraries:

```
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.sql.SparkSession
```

A Spark session is a unified entry point to the Spark application, it provides a way to interact with more functionality of the same using a smaller amount of buildings. Once the session is initialized in Spark, the data is loaded into the session created in the first line of code.

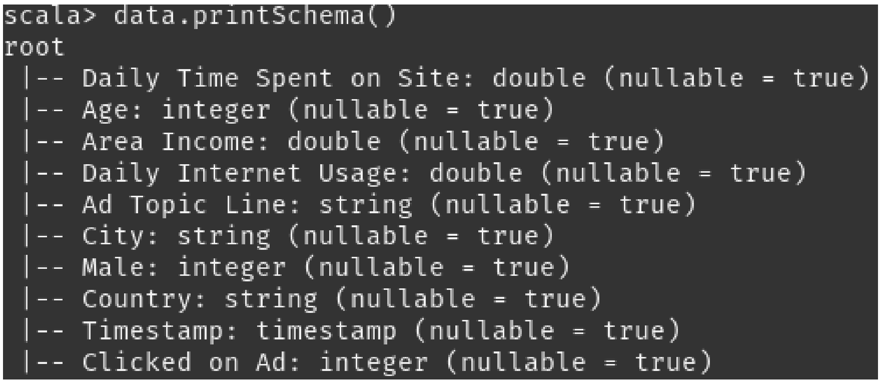
```
val spark = SparkSession.builder().getOrCreate()
val data = spark.read.option("header","true").option("inferSchema",
"true").format("csv").load("CSV/advertising.csv")
```



```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.8
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_292)
Type in expressions to have them evaluated.
Type :help for more information.

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

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

scala> val spark = SparkSession.builder().getOrCreate()
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@45a40353

scala> val data  = spark.read.option("header","true").option("inferSchema", "true").format("csv").load("CSV/advertising.csv")
data: org.apache.spark.sql.DataFrame = [Daily Time Spent on Site: double, Age: int ... 8 more fields]
```

To learn more about the data set, you can show a schema of it performing the following instruction:

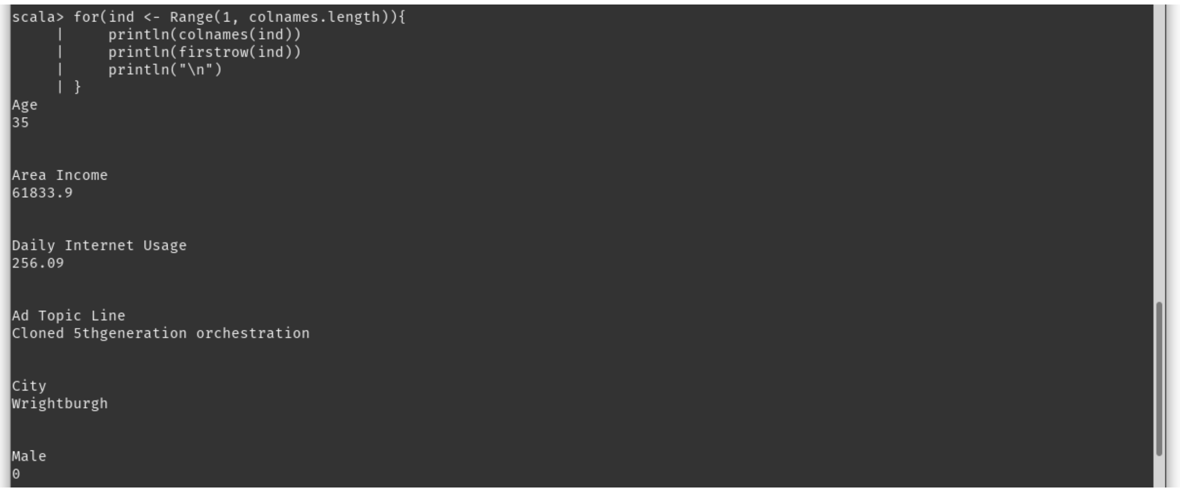
```
data.printSchema()
```



```
scala> data.printSchema()
root
 |-- Daily Time Spent on Site: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Area Income: double (nullable = true)
 |-- Daily Internet Usage: double (nullable = true)
 |-- Ad Topic Line: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Male: integer (nullable = true)
 |-- Country: string (nullable = true)
 |-- Timestamp: timestamp (nullable = true)
 |-- Clicked on Ad: integer (nullable = true)
```

The following method is used to display a record of the dataset with its respective columns:

```
val colnames = data.columns
val firstrow = data.head(1)(0)
println("\n")
println("Example data row")
for(ind <- Range(1, colnames.length)){
    println(colnames(ind))
    println(firstrow(ind))
    println("\n")
}
```



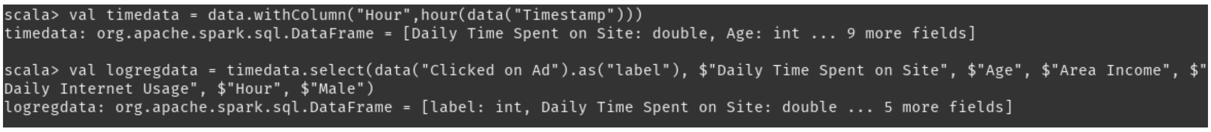
```
scala> for(ind <- Range(1, colnames.length)){
     |     println(colnames(ind))
     |     println(firstrow(ind))
     |     println("\n")
     | }
Age
35


Area Income
61833.9


Daily Internet Usage
256.09


Ad Topic Line
Cloned 5thgeneration orchestration


City
Wrightburgh


Male
0
```

Data preparation

At the beginning of this document, the purpose of this practice was highlighted, to create a machine learning model using the data set from the jcromendez repository. The model must predict whether the user clicked on the displayed ad based on its characteristics.

Therefore, we must carry out the following instructions. First the column "Clicked on Ad" will be renamed as the label to be predicted, then, register the other columns and finally, create a column that indicates what time of day the click was made.

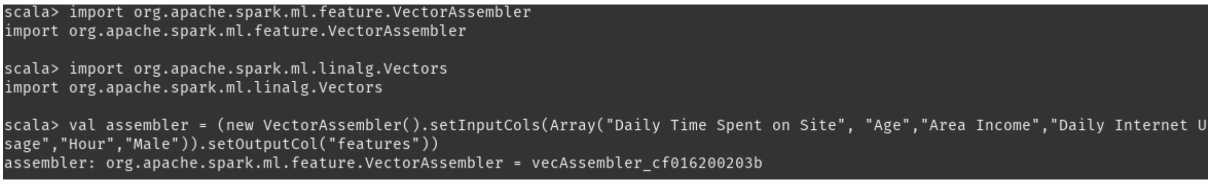
```
val logregdata = timedata.select(data("Clicked on Ad").as("label"), $"Daily Time
Spent on Site", $"Age", $"Area Income", $"Daily Internet Usage", $"Hour", $"Male")

val timedata = data.withColumn("Hour",hour(data("Timestamp")))
```



```
scala> val timedata = data.withColumn("Hour",hour(data("Timestamp")))
timedata: org.apache.spark.sql.DataFrame = [Daily Time Spent on Site: double, Age: int ... 9 more fields]

scala> val logregdata = timedata.select(data("Clicked on Ad").as("label"), $"Daily Time Spent on Site", $"Age", $"Area Income", $"
Daily Internet Usage", $"Hour", $"Male")
logregdata: org.apache.spark.sql.DataFrame = [label: int, Daily Time Spent on Site: double ... 5 more fields]
```

To register the user characteristics to the linear regression model, it is necessary to import the vector and vector assembler libraries. In the following instruction a new column is created with the name of characteristics with all those values and attributes of the user.

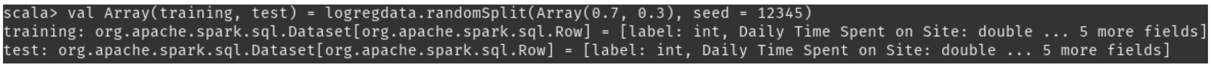
```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors

val assembler = (new VectorAssembler().setInputCols(Array("Daily Time Spent on
Site", "Age","Area Income","Daily Internet
Usage","Hour","Male")).setOutputCol("features"))
```



```
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> val assembler = (new VectorAssembler().setInputCols(Array("Daily Time Spent on Site", "Age","Area Income","Daily Internet U
sage","Hour","Male")).setOutputCol("features"))
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_cf016200203b
```

As a good practice when using machine learning, 70% of the data will be assigned to training the model and the remaining 30% will be used to test the accuracy of the model.

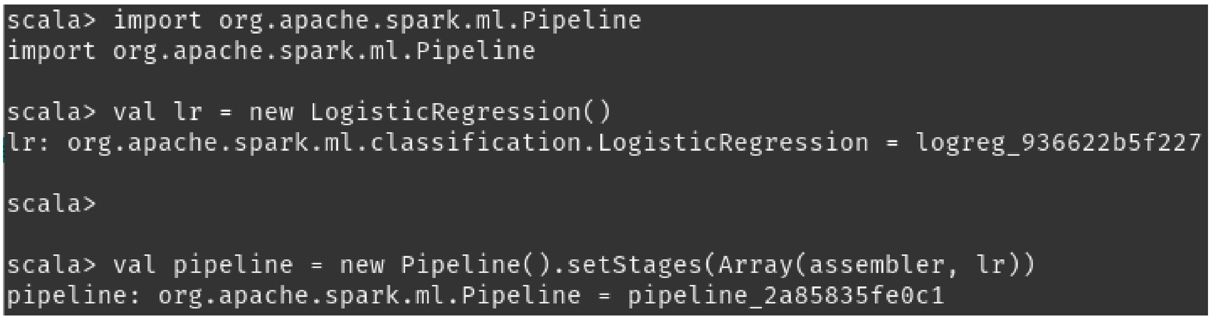
```
val Array(training, test) = logregdata.randomSplit(Array(0.7, 0.3), seed = 12345)
```



```
scala> val Array(training, test) = logregdata.randomSplit(Array(0.7, 0.3), seed = 12345)
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, Daily Time Spent on Site: double ... 5 more fields]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, Daily Time Spent on Site: double ... 5 more fields]
```

To enter the data into the model, it is necessary to use a Pipeline, its function is to automate the data entry that the model will occupy in its creation. To store the model it is necessary to create an empty value for it.

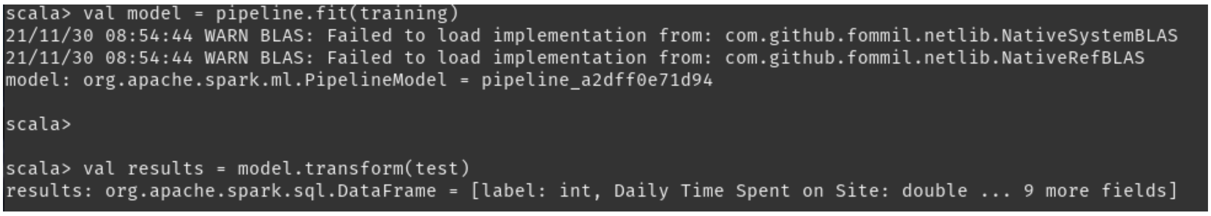
```
import org.apache.spark.ml.Pipeline
val lr = new LogisticRegression()

val pipeline = new Pipeline().setStages(Array(assembler, lr))
```



```
scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> val lr = new LogisticRegression()
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_936622b5f227

scala>

scala> val pipeline = new Pipeline().setStages(Array(assembler, lr))
pipeline: org.apache.spark.ml.Pipeline = pipeline_2a85835fe0c1
```

To train the model, the following instruction is used, training and storing the machine learning model in the value "model"; finally, the results are stored in the “results” value for later precision tests.

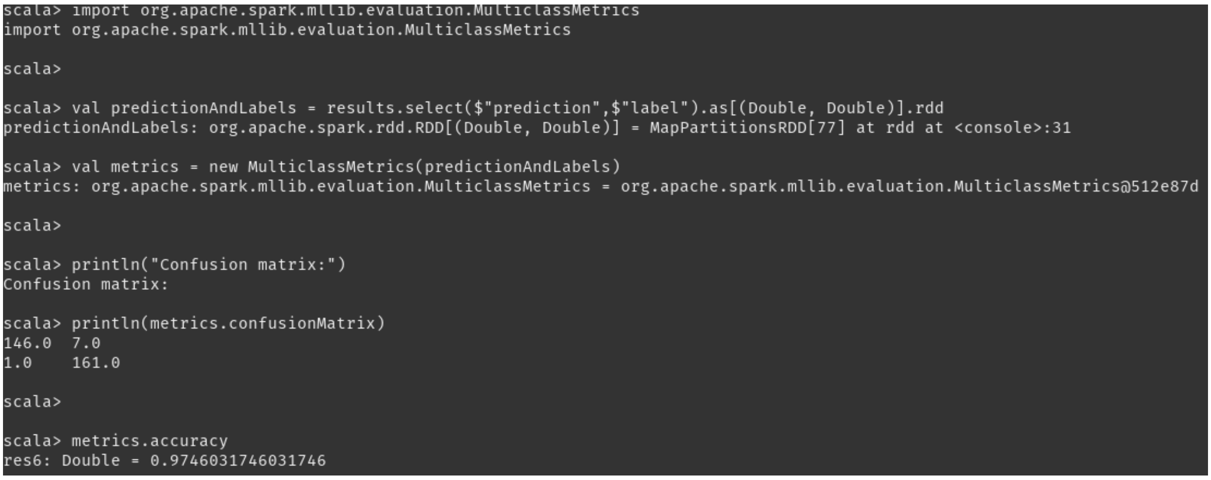
```
val model = pipeline.fit(training)

val results = model.transform(test)
```



```
scala> val model = pipeline.fit(training)
21/11/30 08:54:44 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
21/11/30 08:54:44 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
model: org.apache.spark.ml.PipelineModel = pipeline_a2dff0e71d94

scala>

scala> val results = model.transform(test)
results: org.apache.spark.sql.DataFrame = [label: int, Daily Time Spent on Site: double ... 9 more fields]
```

If you want to know the performance of the model against the test data, it is necessary to import a metric evaluation library. Then the results obtained are converted to RDD type to display the confusion matrix.

```
import org.apache.spark.mllib.evaluation.MulticlassMetrics

val predictionAndLabels = results.select($"prediction",$"label").as[(Double,
Double)].rdd
val metrics = new MulticlassMetrics(predictionAndLabels)

println("Confusion matrix:")
println(metrics.confusionMatrix)

metrics.accuracy
```



```
scala> import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.evaluation.MulticlassMetrics

scala>

scala> val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd
predictionAndLabels: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[77] at rdd at <console>:31

scala> val metrics = new MulticlassMetrics(predictionAndLabels)
metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.apache.spark.mllib.evaluation.MulticlassMetrics@512e87d

scala>

scala> println("Confusion matrix:")
Confusion matrix:

scala> println(metrics.confusionMatrix)
146.0  7.0
1.0    161.0

scala>

scala> metrics.accuracy
res6: Double = 0.9746031746031746
```

We can see in the confusion matrix that the predictions of customers who clicked on the ad is 146 predicting correctly and one incorrectly; of the users who did not click are 161 people with 7 erroneous registrations. We can see that the precision of this model by injecting the test data is 97.46%. Which is very satisfactory for a logistic regression model.