Tecnológico Nacional de México
Instituto Tecnológico de Tijuana

Subdirección Académica
Departamento de Sistemas y Computación

Semestre:
Agosto – Diciembre 2021

Carrera:
Ingeniería en Tecnologías de la Información y Comunicaciones
Ingeniería en Sistemas Computacionales

Materia y serie:
Datos Masivos BDD-1704TI9A

Unidad a evaluar: Unidad III

Nombre de la Tarea:
Práctica 1

Nombre del Alumno:
Sifuentes Martinez Manuel Javier 17212934
Flores Gonzalez Luis Diego C16211486

Nombre del docente:
José Christian Romero Hernández

Practice 1: Logistic regression project

Import a SparkSession with the Logistic Regression library

These are the libraries to be able to carry out subsequent actions of the practice, LogisticRegression to be able to create an object of this that will be used in the Pipeline, and SparkSession, which is what allows us to create a session in the current practice.

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

Use the Error report code

To control errors in the code run, the log4j library is imported.

```
import org.apache.log4j._
Logger.getLogger ("org") .setLevel (Level.ERROR)
```

Create a Spark session

Here the session is declared, having imported the respective library, it is saved in the spark variable.

```
val spark = SparkSession.builder (). getOrCreate ()
```

Use Spark to read the Advertising csv file.

The dataset to be used has the name of advertising, in order to load it into memory the load method is used, which specifies the path where it is located. There are more parameters, such as "format", for the type of file format and "option", so that the first line of the data set is taken as the headers of these.

```
val data = spark.read.option ("header","true") .option
("inferSchema", "true") .format ("csv") .load ("advertising.csv")
```

Print the Schema of the DataFrame

The schema shows the types of data in which the columns of the dataset come, this is important to know whether or not to perform a data cleanup.

```
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)
```

Print an example line

There are two ways to print the first line of the dataset, the easiest is to use the head method, this returns the line, but without the columns to which each one belongs, to be able to visualize these columns the data set must be traversed with the for loop.

```
data.head (1)

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> data.head (1)
res6: Array[org.apache.spark.sql.Row] =
Array([68.95,35,61833.9,256.09,Cloned 5thgeneration
orchestration,Wrightburgh,0,Tunisia,2016-03-27 00:53:11.0,0])
```

Showing 5 example:

```
Example data row

Age
35
Area Income
61833.9
Daily Internet Usage
256.09
Ad Topic Line
Cloned 5thgeneration orchestration
City
Wrightburgh
Male
0
Country
```

```
Tunisia
Timestamp
2016-03-27 00:53:11.0
Clicked on Ad
0
```

Do the following:

- Rename the column "Clicked on Ad" to "label"
- Take the following columns as features "Daily Time Spent on Site", "Age", "Area Income", "Daily Internet Usage", "Timestamp" , "Male"
- Create a new column called "Hour" of the Timestamp containing the "Hour of the click"

The method "withColumn" is used to transform data types, so specifying the column changes to the required data type , and renaming a column is done with the "as" method.

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

Import VectorAssembler and Vectors

These libraries are the ones that help to make the transformation of data to the characteristics that Logistic Regression will work with.

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

Create a new VectorAssembler object called assembler for the features.

In this step, all the columns are grouped into the same one, which bears the name of features, this result is stored in the “assembler” variable.

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

Use “ randomSplit ”to create train and test data divided into 70/30

Two different vectors must be created, one for the training data, which is the one with which the regression algorithm will be adjusted, and the other which is the test algorithm, which It is with which the success effectiveness achieved in the linear regression will be shown, all this maintaining a degree of randomness with the seed.

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

Pipeline Import The Pipeline
library must be imported, which is the one that will help to gather in a single variable, the steps through which they must to pass the data.

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

Create a new LogisticRegression object called "lr"
The LogisticRegression object is created, and declaring a new Pipeline object, the "assembler" and "lr" are passed as parameters, which will be the steps through which the data will pass for perform linear regression. At the end the training data is passed to fit the linear regression to the data.

```
val lr = new LogisticRegression()
val pipeline = new Pipeline() .setStages (Array(assembler, lr))
val model = pipeline.fit (training)
```

Fit (fit) the pipeline for the training set
Once the model has been trained, the test data is passed, so that the results with linear regression. Finally the result is taken in the test set with "transform".

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

For Metrics and Evaluation amount MulticlassMetrics
Once the results have been obtained, they must be evaluated to verify the accuracy of the prediction, and for that MulticlassMetrics is imported.

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

Convert the test results to RDD using .as and .rdd
The results of the predictioncolumns are converted and "label"as type double, and stored in the "predictionAndLabels" variable.

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

Initialize a MulticlassMetrics object
Once transformed, the data is evaluated with the help of the MulticlassMetrics object and stored in the metrics variable .

```
val metrics = new MulticlassMetrics(predictionAndLabels)
```

Print the Confusion matrix
The confusion matrix is used to evaluate false positives and negatives, with true positives and negatives, and these results are displayed, as well as the precision obtained from the linear regression using the accuracy method.

```
println ("Confusion matrix:")
scala> println (metrics.confusionMatrix)
146.0  7.0
1.0    161.0


scala> metrics.accuracy
res13: Double = 0.9746031746031746
```