
TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

NOMBRE DE LOS ALUMNOS:

GALAVIZ LONA OSCAR EDUARDO (N.CONTROL: 17212993)

MARQUEZ MILLAN SEASHELL VANESSA (N.CONTROL: )

Carrera: Ingeniería Informática

Semestre: 9no

MATERIA: Datos Masivos

PROFESOR: JOSE CHRISTIAN ROMERO HERNANDEZ

Examen U2

Unidad 2

## Introduccion

Para poder desarrollor esta practica primero debemos tener en cuenta ciertos puntos importantes en este caso se trata sobre las metodos de clasificacion, asi como tomar algunos puntos anteriores de lo ya antes visto, la clasificacion nos permite identificar o saber en el momento de la busqueda de datos especificos que tan probable es que ese dato sea encontrado o que salga, tambien es utilizado para identificar patrones que sean extraños o muy poco comunes dentro de archivo de datos. Estas clasificaciones ademas de permitinos identificar este tipo de patrones, tambien nos permite saber que exactitud tiene este y cuanta probabildad de margen de error tenga este, con el cual podemos identificar si se puede confiar al 100% del modelo o simplemente tomarlo como una opcion al momento del analisis de datos. Los diferentes modelos funcionan de maneras diferentes pero los datos pueden llegar a una concordancia, se puede decir que los modelos de clasificacion sirve para sacar estadisticas e identificar patrones dentro de las empresas que lo solicitan para poder tener un panorama de como se encuentra una empresa

## Development

**extra-.Import the library

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

### 1-.Load the data iris csv

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

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

### 2-.Names colums

```
//Clean the dataframe
val df = iris.withColumn("sepal_length",
$"sepal_length".cast(DoubleType)).withColumn("sepal_width",
$"sepal_width".cast(DoubleType)).withColumn("petal_length",
$"petal_length".cast(DoubleType)).withColumn("petal_width",
$"petal_width".cast(DoubleType))

//2-.Colum names
```

```
df.columns
```

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

### 3-.Schema

```
df.printSchema()
```

```
scala> df.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 first 5 columns

```
df.show(5)
```

```
scala> df.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
```

### 5-.use the method describe

```
df.describe().show()
```

```
scala> df.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-.Transformation and label we have to use

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

val indexerL = new
StringIndexer().setInputCol("species").setOutputCol("indexedLabel").fit(features)
val indexerF = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCate
gories(4).fit(features)

val splits = features.randomSplit(Array(0.6, 0.4))
val training = splits(0)
val test = splits(1)

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

**7-.Building model**

```
val trainer = new
MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("indexedLabel").set
FeaturesCol("indexedFeatures").setBlockSize(128).setSeed(System.currentTimeMillis)
.setMaxIter(200)
val converterL = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels
(indexerL.labels)
val pipeline = new Pipeline().setStages(Array(indexerL, indexerF, trainer,
converterL))

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

**8-. print the result of model**

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

val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredictionCol
("prediction").setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
println("Error = " + (1.0 - accuracy))
```

```
scala> println("Error = " + (1.0 - accuracy))
Error = 0.015873015873015928
```

## Conclusion

En el desarrollo de esta practica no fue complicado como tal pero si algo laboriosa porque al momento de querer buscar informacion nos daban resultados de ejemplos de spark pero en python y esto se tenia que

transformar a codigo scala pero eso no nos detuvo con la documentacion de spark y los ejemplos de algunos compañeros pudimos resolver todas las cuestiones que se tenian y evidentemente tambien tuvimos que entender y comprender el codigo y lo que estabamos haciendo aunque algunas partes del codigo eran partes que ya teniamos presentes por la unidad pasada en conclusion el poder utilizar estos metodos de clasificacion fue satisfactorio porque aprendimos a utilizar un modelo dentro de un archivo de datos que nos permitia tener un analisis estadistico de estos y para tener un interpretacion de los mismo asi como un panorama mas amplio del mismo identificando patrones o datos sobresalientes