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 II

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

First you must import the libraries to use, they contain the Methods that will later allow to perform all the procedures and actions to be able to perform the classification by the Decision tree method.

```
import org.apache.spark.ml.Pipeline
import
org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.classification.DecisionTreeClassifier
import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature. {IndexToString, StringIndexer,
VectorIndexer}
```

Afterwards, the data set to be analyzed must be loaded, so the format is specified, in this case "libsvm" and in the method load specifies the path where the file is located, being in the same directory, only the name sample_libsvm_data.txt is specified.

```
val data = spark.read.format ("libsvm") .load
("sample_libsvm_data.txt")
```

Once the data is loaded, the next step is to declare the following indexing vectors, which do not modify the current data, they are only indexed, this allows to optimize the procedures to be carried out.

In the setInputCol method the name of the data column is specified, and in setOutputCol how the indexed column will be temporarily called.

```
val labelIndexer = new StringIndexer() .setInputCol ("label")
.setOutputCol ("indexedLabel") .fit (data)
val featureIndexer = new VectorIndexer() .setInputCol ("features")
.setOutputCol ("indexedFeatures") .setMaxCategories (4)
```

Already having the two vectors, now the data must be separated between the test and training data (test and training). declare another array and with the help of the randomSplit method, the data is separated, specifying that 70% of the data will be used for training, and the remainder for testing.

```
val Array(trainingData, testData) = data.randomSplit (Array(0.7,
0.3))
```

In this step the model object is generated, specifying how the column that will carry the label that will be used for the predictions will be called, and then the characteristics to be used to predict

```
val dt = new DecisionTreeClassifier() .setLabelCol
("indexedLabel") .setFeaturesCol ("indexedFeatures")
```

The new IndexToString object will take care of converting existing columns to indices, all storing in the labelConverter variable.

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

Inside the Pipeline is where everything created so far is related, it is specified that the label indexer (labelIndexer) will be used, characteristics (featureIndexer), the model (dt) and finally the converter (labelConverter), which is the removes the indexes that had been created previously.

```
val pipeline = new Pipeline() .setStages (Array(labelIndexer,
featureIndexer, dt, labelConverter))
```

Follow the training stage, here the created pipeline is used, and with the help of the fit method, the data source to be used is specified. This data will go through the pipeline stages, which were declared in the previous step.

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

Once the training has been carried out, we already have data to be able to make predictions, so the declared variable to store the training data is specified, and with the transform method, the separate data is passed as a parameter above to make the predictions. These will go through the same steps as the previous ones and in the end the predictions will return.

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

Here the predictions data is selected, and the parameters are the name of the columns contained in the predictions variable, which are the label that was predicted, the real label and the characteristics and at the end , with the show method it is specified that only the first 5 rows are taken.

```
predictions.select ("predictedLabel", "label", "features") .show
(5)
```

Result:

```
predictions: org.apache.spark.sql.DataFrame = [label: double, features: vector
... 6 more fields]
+ -------------- + ----- + -------------------- +
|predictedLabel|label|            features|
+ -------------- + ----- + -------------------- +
|           0.0|  0.0| (692, [98,99,100,1...|
|           0.0|  0.0| (692, [122,123,124...|
|           0.0|  0.0| (692, [123,124,125. ..|
|           0.0|  0.0| (692, [123,124,125...|
|           0.0|  0.0| (692, [124,125,126...|
+ ---------- ---- + ----- + -------------------- +
only showing top 5 rows
```

The evaluator, which is what is declared in the following code, provides the declared metrics, which is through the setMetricName method, specifying that the accuracy is wanted, then the error obtained by the model is printed, completely subtracting (1), the percentage of accuracy.

```
val evaluator = new MulticlassClassificationEvaluator()
.setLabelCol ("indexedLabel") .setPredictionCol ("prediction")
.setMetricName ("accuracy")
val accuracy = evaluator.evaluate (predictions)
println (s "Test Error = $ {(1.0 - accuracy)}")
```

Result:

```
accuracy: Double = 0.9722222222222222
Test Error = 0.02777777777777779
```

Finally, the visualization of the decision tree model is explicitly shown, specifying that only 2 stages be carried out and also with the help of the asInstanceOf method, passing as a parameter the type of model to be carried out, in this case DecisionTreeClassificationModel.

```
val treeModel = model.stages (2) .asInstanceOf
[DecisionTreeClassificationModel]
println (s "Learned classification tree model: \ n $
{treeModel.toDebugString}")
```

Result:

```
Learned classification tree model:
 DecisionTreeClassificationModel (uid=dtc_a688c0785eea) of depth 2 with 5 do
not give
  If (feature 406 <= 22.0)
   If (feature 99 in {2.0})
    Predict: 0.0
   Else (feature 99 not in {2.0})
    Predict: 1.0
  Else (feature 406 > 22.0)
   Predict: 0.0
```