NATIONAL TECHNOLOGICAL INSTITUTE OF MEXICO
TECHNOLOGICAL INSTITUTE OF TIJUANA

ACADEMIC SUBDIRECTION
Systems and Computing Department

SEMESTER
August – December 2021

ACADEMIC CAREER
Eng. Information and Comunications Technologies

SUBJECT AND KEY:
Big Data BDD – 1704TI9A

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

NAME OF WORK:
Practice #3 – Random Forest Classifier

UNIT:
Unit II

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

Tijuana Baja California **Due Date:** November 5th, 2021

Introduction

In this present document, the topic of Random Forest Classifier, how it works and examples will be explained. The example in this document is done with the documentation provided by Apache Spark in the following URL: [https://spark.apache.org/docs/2.4.7/ml-classification-regression.html#random-forest-classifier]

A random forest classifier is an estimator that seeks to fit a defined number of classifier trees into various subsample of the data and uses arithmetic means to improve the precision of the model and control over-fitting. A random forest generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.
Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. The forest it builds, is an ensemble of decision trees, trained with the "bagging method". The bagging method is a combination of learning models that increases the overall result.

Random forest are frequently used as a "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

Development

The example uses a Spark-Shell enviroment, powered by Java Runtime and as a imput language Scala is used. Spark is available in MacOS, Windows and Linux Distros.

The example below demostrates how to load a LibSVM file, parse it as an RDD of the column labeled pojnt and then perform classifications using decision tree with GilinImpurity as a measure, and lastly add a maximum tree depth of 5.

Step 1: Load the following libraries that will be used throughout the example. The libraries are:

- ml.Pipeline
- classification.RandomForestClassificationModel
- classification.RandomForestClassifier
- MulticlassClassificationEvaluator
- feature.IndexToString
- feature.StringIndexer
- feature.VectorIndexer

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

import org.apache.spark.ml.classification.{RandomForestClassificationModel,
RandomForestClassifier}

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}
```

Step 2: Load and parse the data file into a data frame.

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

Step 3: Index the labels (to predict), adding metadata to the label column and fit the whole dataset to include all labels in index.

```
val labelIndexer = new StringIndexer()

        .setInputCol("label")

        .setOutputCol("indexedLabel")

        .fit(data)
```

Step 4: Automatically identify all categorical features and index them. Set the maxCategories so features with a > 4 distinct values are treated as continuous.

```
val featureIndexer = new VectorIndexer()

    .setInputCol("features")

    .setOutputCol("indexedFeatures")

    .setMaxCategories(4)

    .fit(data)
```

Step 5: Split the data frame into two parts (training_set, test_set) in 70% and 30% of the data respectively.

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

Step 6: Train the RandomForest model, setting the maximum number of trees allowed to ten.

```
val randomforest = new RandomForestClassifier()
    .setLabelCol("indexedLabel")
    .setFeaturesCol("indexedFeatures")
    .setNumTrees(10)
```

Step 7: Return the indexed labels back to their original state.

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

Step 8: Train model. This also runs the indexers.

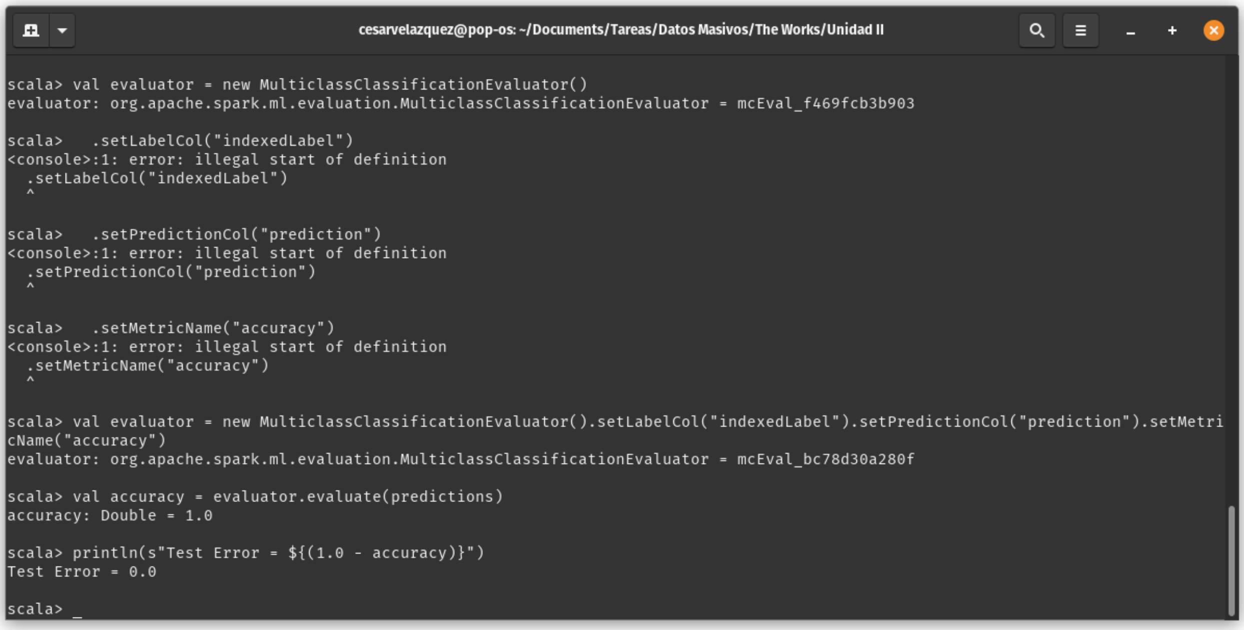
```
val pipeline = new Pipeline()
    .setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))
val model = pipeline.fit(trainingData)
```

Step 9: Make the predictions and compute the test error

```
val predictions = model.transform(testData)
val accuracy = evaluator.evaluate(predictions)
println(s"Test Error = ${(1.0 - accuracy)}")
```

Conclusions

As stated before, the Random Forest classifiers predicts the values of the labels more accurately than standard decisions trees.


cesarvelazquez@pop-os: ~/Documents/Tareas/Datos Masivos/The Works/Unidad II

```
scala> val evaluator = new MulticlassClassificationEvaluator()
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_f469fcb3b903

scala>   .setLabelCol("indexedLabel")
<console>:1: error: illegal start of definition
  .setLabelCol("indexedLabel")
  ^

scala>   .setPredictionCol("prediction")
<console>:1: error: illegal start of definition
  .setPredictionCol("prediction")
  ^

scala>   .setMetricName("accuracy")
<console>:1: error: illegal start of definition
  .setMetricName("accuracy")
  ^

scala> val evaluator = new MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredictionCol("prediction").setMetri
cName("accuracy")
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_bc78d30a280f

scala> val accuracy = evaluator.evaluate(predictions)
accuracy: Double = 1.0

scala> println(s"Test Error = ${(1.0 - accuracy)}")
Test Error = 0.0

scala> _
```

As we can see in the image above, we can see that the testing error is at 0%. This means that the model predicts the data 100% correctly with this particular dataset. Not all predictions are 100% correct, it depends highly in the dataset that is tested upon, byt, generally it's more accurate than decision trees.