TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE FEBRERO-JUNIO 2022

CARRERA
Ingeniería en informática

MATERIA
Datos masivos

TÍTULO
Práctica # 3

Integrantes:
Munguía Silva Edgar Geovanny #17212344
Perez López Alicia Guadalupe ##18210514

NOMBRE DEL MAESTRO
Jose Christian Romero Hernadez

Tijuana Baja California 06 de mayo del 2022

Concepto de Random Forest.

Es un popular algoritmo de aprendizaje automático que pertenece a la técnica de machine learning. Se puede utilizar tanto para problemas de clasificación como de regresión en ML. Se basa en el concepto de aprendizaje conjunto, que es un proceso de combinación de múltiples clasificadores para resolver un problema complejo y mejorar el rendimiento del modelo. Como sugiere el nombre, "Random Forest es un clasificador que contiene una serie de árboles de decisión en varios subconjuntos del conjunto de datos dado y toma el promedio para mejorar la precisión predictiva de ese conjunto de datos". En lugar de depender de un árbol de decisiones, el bosque aleatorio toma la predicción de cada árbol y, en función de los votos mayoritarios de las predicciones, predice el resultado final.

Código de entrada de clasificación.

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.mllib.util.MLUtils

// Load and parse the data file.
val data = MLUtils.loadLibSVMFile(sc, "sample_libsvm_data.txt")
// Split the data into training and test sets (30% held out for
testing)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Train a RandomForest model.
// Empty categoricalFeaturesInfo indicates all features are
continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3 // Use more in practice.
val featureSubsetStrategy = "auto" // Let the algorithm choose.
val impurity = "gini"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainClassifier(trainingData, numClasses,
categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

// Evaluate model on test instances and compute test error
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
```

```
  (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 !=
r._2).count.toDouble / testData.count()
println(s"Test Error = $testErr")
println(s"Learned classification forest model:\n
${model.toDebugString}")

// Save and load model
model.save(sc, "target/tmp/myRandomForestClassificationModel")
val sameModel = RandomForestModel.load(sc,
"target/tmp/myRandomForestClassificationModel")
```

Salida.

```
Tree 0:

If (feature 245 <= 16.0)

If (feature 315 <= 64.0)

If (feature 467 <= 70.0)

Predict: 1.0

Else (feature 467 > 70.0)

Predict: 0.0

Else (feature 315 > 64.0)

Predict: 0.0

Else (feature 245 > 16.0)

Predict: 0.0

Tree 1:

If (feature 512 <= 1.5)

If (feature 688 <= 14.0)
```

```
Predict: 1.0

Else (feature 688 > 14.0)

Predict: 0.0

Else (feature 512 > 1.5)

Predict: 0.0

Tree 2:

If (feature 356 <= 19.0)If (feature 301 <= 27.0)

If (feature 351 <= 2.0)

Predict: 0.0

Else (feature 351 > 2.0)

Predict: 1.0

Else (feature 301 > 27.0)

Predict: 0.0

Else (feature 356 > 19.0)

Predict: 0.0

scala>
```

Código de entrada de regresión.

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.mllib.util.MLUtils

// Load and parse the data file.
val data = MLUtils.loadLibSVMFile(sc, "sample_libsvm_data.txt")
// Split the data into training and test sets (30% held out for
testing)
```

```
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Train a RandomForest model.
// Empty categoricalFeaturesInfo indicates all features are
continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3 // Use more in practice.
val featureSubsetStrategy = "auto" // Let the algorithm choose.
val impurity = "variance"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainRegressor(trainingData,
categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

// Evaluate model on test instances and compute test error
val labelsAndPredictions = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val testMSE = labelsAndPredictions.map{ case(v, p) => math.pow((v
- p), 2)}.mean()
println(s"Test Mean Squared Error = $testMSE")
println(s"Learned regression forest model:\n
${model.toDebugString}")

// Save and load model
model.save(sc, "target/tmp/myRandomForestRegressionModel")
val sameModel = RandomForestModel.load(sc,
"target/tmp/myRandomForestRegressionModel")
```

Salida.

```
Tree 0:

If (feature 406 <= 126.5)

Predict: 0.0
```

```
Else (feature 406 > 126.5) Predict: 1.0

Tree 1:

If (feature 406 <= 9.5)

Predict: 0.0

Else (feature 406 > 9.5) Predict: 1.0

Tree 2:

If (feature 406 <= 126.5)

Predict: 0.0
Else (feature 406 > 126.5) Predict: 1.0

scala>
```