|  |  |
| --- | --- |
|  | package cat.marc.uni.tfg; |
|  |  |
|  | import org.apache.spark.ml.Pipeline; |
|  | import org.apache.spark.ml.PipelineModel; |
|  | import org.apache.spark.ml.PipelineStage; |
|  | import org.apache.spark.ml.classification.\*; |
|  | import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator; |
|  | import org.apache.spark.ml.feature.\*; |
|  | import org.apache.spark.ml.util.MetadataUtils; |
|  | import org.apache.spark.rdd.RDD; |
|  | import org.apache.spark.sql.Dataset; |
|  | import org.apache.spark.sql.Row; |
|  | import org.apache.spark.sql.SQLContext; |
|  | import org.apache.spark.sql.types.DataTypes; |
|  | import org.apache.spark.sql.types.StructField; |
|  | import org.apache.spark.sql.types.StructType; |
|  | import scala.Tuple2; |
|  | import scala.tools.nsc.backend.icode.Primitives; |
|  |  |
|  | import org.apache.spark.mllib.evaluation.MulticlassMetrics; |
|  |  |
|  | import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  | import java.util.Map; |
|  |  |
|  | /\*\* |
|  | \* Created by mhyark on 01/01/21. |
|  | \*/ |
|  | public class TestBank { |
|  |  |
|  | private SQLContext ss; |
|  |  |
|  | // Tests and Scores |
|  | private ArrayList<String> tests; |
|  | private ArrayList<TestScore> scores; |
|  |  |
|  | // Pipelines |
|  | private StringIndexerModel indexer\_training\_model; |
|  | private StringIndexer indexer\_training; |
|  | private Pipeline pipeline; |
|  | private VectorAssembler assembler; |
|  | private VectorIndexer vector\_indexer; |
|  | private VectorIndexerModel vector\_indexer\_model; |
|  |  |
|  | // Datasets |
|  | private String datasetPath; |
|  | private Dataset training\_dataset; |
|  | private Dataset testing\_dataset; |
|  |  |
|  | public TestBank(SQLContext ss, ArrayList<String> test\_names, String datasetPath) { |
|  | this.ss = ss; |
|  | this.tests = test\_names; |
|  | this.datasetPath = datasetPath; |
|  | this.scores = new ArrayList<TestScore>(); |
|  |  |
|  | // DATASETS |
|  | List<StructField> fields = Arrays.asList( |
|  | DataTypes.createStructField("No", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("Time", DataTypes.DoubleType, true), |
|  | DataTypes.createStructField("Source", DataTypes.StringType, true), |
|  | DataTypes.createStructField("Destination", DataTypes.StringType, true), |
|  | DataTypes.createStructField("Protocol", DataTypes.StringType, true), |
|  | DataTypes.createStructField("Lenght", DataTypes.IntType, true), |
|  | DataTypes.createStructField("Info", DataTypes.StringType, true), |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | DataTypes.createStructField("applications", DataTypes.StringType, true)); |
|  |  |
|  | StructType schema = DataTypes.createStructType(fields); |
|  |  |
|  | Dataset<Row> data = ss |
|  | .read() |
|  | .format("com.databricks.spark.csv") |
|  | .schema(schema) |
|  | .option("header", "false") |
|  | .load(datasetPath); |
|  |  |
|  | Dataset<Row>[] splits = data.randomSplit(new double[]{0.7, 0.3}); |
|  | training\_dataset = splits[0]; |
|  | testing\_dataset = splits[1]; |
|  |  |
|  | // PIPELINE PREPARATION |
|  | //indexer\_training\_model = new StringIndexer() |
|  | indexer\_training = new StringIndexer() |
|  | .setInputCol("applications") |
|  | .setOutputCol("label"); |
|  | //.fit(training\_dataset); |
|  |  |
|  | //vector\_indexer\_model = new VectorIndexer() |
|  | vector\_indexer = new VectorIndexer() |
|  | .setInputCol("features\_prev") |
|  | .setOutputCol("features") |
|  | .setMaxCategories(2); |
|  |  |
|  |  |
|  | // PREPROCESSING |
|  | //DurationTransformer myTransformer = new DurationTransformer("time"); |
|  |  |
|  | String[] inputCols = {"source", "destination", "protocol", "info"}; |
|  | assembler = new VectorAssembler(); |
|  | assembler.setInputCols(inputCols); |
|  | assembler.setOutputCol("features\_prev"); |
|  | } |
|  |  |
|  | private PipelineModel train(Pipeline pipeline) { |
|  | return pipeline.fit(training\_dataset); |
|  | } |
|  |  |
|  | private Dataset<Row> test(PipelineModel model) { |
|  | return model.transform(testing\_dataset); |
|  | } |
|  |  |
|  | private void setScores(Dataset<Row> predictions, String name) { |
|  | TestScore ts = new TestScore(); |
|  | ts.test\_name = name; |
|  |  |
|  | MulticlassMetrics metrics = new MulticlassMetrics(predictions.select("prediction", "label")); |
|  |  |
|  | MulticlassClassificationEvaluator evaluator = new MulticlassClassificationEvaluator() |
|  | .setLabelCol("label") |
|  | .setPredictionCol("prediction"); |
|  |  |
|  | evaluator.setMetricName("accuracy"); |
|  | double accuracy = evaluator.evaluate(predictions); |
|  | ts.accuracy = accuracy; |
|  |  |
|  | evaluator.setMetricName("weightedPrecision"); |
|  | ts.weighted\_precision = evaluator.evaluate(predictions); |
|  | evaluator.setMetricName("weightedRecall"); |
|  | ts.weighted\_recall = evaluator.evaluate(predictions); |
|  | evaluator.setMetricName("f1"); |
|  | ts.f1 = evaluator.evaluate(predictions); |
|  |  |
|  | ts.confusion\_matrix = metrics.confusionMatrix(); |
|  |  |
|  | scores.add(ts); |
|  | } |
|  |  |
|  | private void getScores() { |
|  | System.out.println("\n\n\*\*\*\*\*\*\*\*\*\*\* TEST RESULTS \*\*\*\*\*\*\*\*\*\*\*\n"); |
|  | int size = tests.size(); |
|  | for (int i = 0; i < size; i++) { |
|  | TestScore ts = scores.get(i); |
|  | System.out.println(ts.test\_name); |
|  | System.out.println("\tAccuracy: " + ts.accuracy); |
|  | System.out.println("\tF1 : " + ts.f1); |
|  | System.out.println("\tWeighted Precission: " + ts.weighted\_precision); |
|  | System.out.println("\tWeighted Recall: " + ts.weighted\_recall); |
|  | System.out.println("\n"); |
|  | System.out.println("Confusion Matrix:"); |
|  | System.out.println(ts.confusion\_matrix.toString()); |
|  | System.out.println("\n"); |
|  | } |
|  | } |
|  |  |
|  | public void run() { |
|  | int size = tests.size(); |
|  | for (int i = 0; i < size; ++i) { |
|  | PipelineModel model; |
|  | Dataset<Row> predictions; |
|  |  |
|  | switch (tests.get(i)) { |
|  |  |
|  | case "MultinomialLogisticRegression": |
|  |  |
|  | LogisticRegression lr = new LogisticRegression() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features") |
|  | .setMaxIter(10) |
|  | .setRegParam(0.3) |
|  | .setElasticNetParam(0.8); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, lr}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Multinomial Logistic Regression"); |
|  |  |
|  | break; |
|  |  |
|  | case "DecisionTree": |
|  |  |
|  | DecisionTreeClassifier dt = new DecisionTreeClassifier() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features"); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, vector\_indexer, dt}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Decisions Tree"); |
|  |  |
|  |  |
|  | //vector\_indexer\_model = (VectorIndexerModel) (model.stages()[2]); |
|  | //Map<Integer, Map<Double, Integer>> categoryMaps = vector\_indexer\_model.javaCategoryMaps(); |
|  | //MetadataUtils.getCategoricalFeatures(training\_dataset.schema()("features")); |
|  | // |
|  | //Map<Object, Object> categoryMaps = MetadataUtils.getCategoricalFeatures(training\_dataset.select("features").schema()); |
|  |  |
|  | /\* |
|  | System.out.println("\n\n\*\*\*\*\*\*\*\*\*\* ALERT \*\*\*\*\*\*\*\*\*"); |
|  | System.out.println("Chose " + categoryMaps.size() + " categorical features:"); |
|  | for (Object feature : categoryMaps.keySet()) { |
|  | System.out.print(" " + feature); |
|  | } |
|  | System.out.println("\n\n"); |
|  | training\_dataset.printSchema(); |
|  | \*/ |
|  |  |
|  |  |
|  | System.out.println("\n"); |
|  | //StringIndexerModel stringIndexerModelAux = (StringIndexerModel) (model.stages()[0]); |
|  | //System.out.println("\n\n" + "\*\*\*\*\*\*\* AASDASDASDASDASDASDASDADAS \*\*\*\*\*\*\*\*\*"); |
|  | //System.out.println("NUMERO DE CLASSES: " + stringIndexerModelAux.labels().length + "\n\n"); |
|  | break; |
|  |  |
|  | case "RandomForest": |
|  |  |
|  | RandomForestClassifier rf = new RandomForestClassifier() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features"); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, rf}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Random Forest"); |
|  |  |
|  | break; |
|  |  |
|  |  |
|  | // ACTUALMENT NOMÉS SUPORTA CLASSIFICACIÓ BINARIA |
|  | /\* |
|  | case "Gradient-boostedTree": |
|  |  |
|  | GBTClassifier gbt = new GBTClassifier() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features") |
|  | .setMaxIter(10); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, gbt}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Gradient-boosted Tree"); |
|  |  |
|  | break; |
|  | \*/ |
|  |  |
|  | case "MultilayerPerceptron": |
|  |  |
|  |  |
|  | // CALDRÀ COMPROVAR EL NUMERO FINAL DE FEATURES I DE CLASSES!!! |
|  |  |
|  | // specify layers for the neural network: |
|  | // input layer of size 4 (features), two intermediate of size 5 and 4 |
|  | // and output of size 48 (classes) |
|  | int[] layers = new int[] {14, 5, 4, 48}; |
|  |  |
|  | // create the trainer and set its parameters |
|  | MultilayerPerceptronClassifier trainer = new MultilayerPerceptronClassifier() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features") |
|  | .setLayers(layers) |
|  | .setBlockSize(128) |
|  | .setSeed(1234L) |
|  | .setMaxIter(100); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, trainer}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Multilayer Perceptron"); |
|  |  |
|  | break; |
|  |  |
|  | // NO DISPONIBLE A LA VERSIO 2.1.1 |
|  | /\* |
|  | case "LinearSuportVectorMachine": |
|  |  |
|  | LinearSVC lsvc = new LinearSVC() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features") |
|  | .setMaxIter(10) |
|  | .setRegParam(0.1); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, lsvc}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Linear Suport Vector Machine"); |
|  |  |
|  | break; |
|  | \*/ |
|  |  |
|  | case "One-vs-Rest": |
|  |  |
|  | // configure the base classifier. |
|  | LogisticRegression classifier = new LogisticRegression() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features") |
|  | .setMaxIter(10) |
|  | .setTol(1E-6) |
|  | .setFitIntercept(true); |
|  |  |
|  | // instantiate the One Vs Rest Classifier. |
|  | OneVsRest ovr = new OneVsRest().setClassifier(classifier); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, ovr}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "One-vs-Rest"); |
|  |  |
|  | break; |
|  |  |
|  | case "NaiveBayes": |
|  |  |
|  | NaiveBayes nb = new NaiveBayes() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features"); |
|  |  |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, nb}); |
|  | model = train(pipeline); |
|  | predictions = test(model); |
|  | setScores(predictions, "Naive Bayes"); |
|  | break; |
|  |  |
|  | } |
|  | } |
|  |  |
|  | getScores(); |
|  | } |
|  | } |