|  |
| --- |
|  |
|  |  |
|  |  |
|  | package cat.mhyark.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.DecisionTreeClassifier; |
|  | import org.apache.spark.ml.classification.DecisionTreeClassificationModel; |
|  | import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator; |
|  | import org.apache.spark.ml.feature.\*; |
|  | import org.apache.spark.sql.Dataset; |
|  | import org.apache.spark.sql.Row; |
|  | import org.apache.spark.sql.SQLContext; |
|  | import org.apache.spark.sql.SparkSession; |
|  | import org.apache.spark.sql.types.DataTypes; |
|  | import org.apache.spark.sql.types.StructField; |
|  | import org.apache.spark.sql.types.StructType; |
|  |  |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  | // $example off$ |
|  |  |
|  |  |
|  | import org.apache.spark.mllib.evaluation.MulticlassMetrics; |
|  |  |
|  | public class TreeClassifier { |
|  |  |
|  | private SQLContext ss; |
|  | private StringIndexer indexer\_training; |
|  | private StringIndexer indexer\_realtime; |
|  | private Pipeline pipeline; |
|  | private VectorAssembler assembler; |
|  | private PipelineModel model; |
|  | private DecisionTreeClassifier dt; |
|  | private DurationTransformer myTransformer; |
|  |  |
|  | Dataset<Row> trainingData; |
|  | Dataset<Row> testData; |
|  |  |
|  | public TreeClassifier(SQLContext ss) { |
|  |  |
|  | this.ss = ss; |
|  |  |
|  |
|  |
|  |
|  |
|  | indexer\_training = new StringIndexer() |
|  | .setInputCol("applications") |
|  | .setOutputCol("label"); |
|  |  |
|  | String[] inputCols = {"source", "destination", "protocol", "info", "bytes}; |
|  | assembler = new VectorAssembler(); |
|  | assembler.setInputCols(inputCols); |
|  | assembler.setOutputCol("features"); |
|  |  |
|  |  |
|  | // Automatically identify categorical features, and index them. |
|  | //VectorIndexerModel featureIndexer = new VectorIndexer() |
|  | // .setInputCol("features") |
|  | // .setOutputCol("indexedFeatures") |
|  | // .setMaxCategories(4) // features with > 4 distinct values are treated as continuous. |
|  | // .fit(data); |
|  |  |
|  |  |
|  | // Train a DecisionTree model. |
|  | dt = new DecisionTreeClassifier() |
|  | .setLabelCol("label") |
|  | .setFeaturesCol("features"); |
|  |  |
|  | } |
|  |  |
|  | public void train(String datasetPath) { |
|  |  |
|  | List<StructField> fields = Arrays.asList( |
|  | DataTypes.createStructField("protocol", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("src\_port", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("dest\_port", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("packets", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("bytes", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("start\_time", DataTypes.DoubleType, true), |
|  | DataTypes.createStructField("end\_time", DataTypes.DoubleType, true), |
|  | DataTypes.createStructField("duration", DataTypes.DoubleType, true), |
|  | DataTypes.createStructField("bytesXpakts", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("toS", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("urg", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("ack", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("psh", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("rst", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("syn", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("fin", DataTypes.IntegerType, 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); |
|  |  |
|  | //data.show(5); |
|  |  |
|  | StringIndexerModel indexerModel = indexer\_training.fit(data); |
|  |  |
|  |  |
|  | // Split the data into training and test sets (30% held out for testing). |
|  | Dataset<Row>[] splits = data.randomSplit(new double[]{0.7, 0.3}); |
|  | trainingData = splits[0]; |
|  | testData = splits[1]; |
|  |  |
|  | // Convert indexed labels back to original labels.indexedLabel |
|  | IndexToString labelConverter = new IndexToString() |
|  | .setInputCol("prediction") |
|  | .setOutputCol("predictedLabel") |
|  | .setLabels(indexerModel.labels()); |
|  |  |
|  | // Chain indexers and tree in a Pipeline. |
|  | pipeline = new Pipeline().setStages(new PipelineStage[]{indexer\_training, assembler, dt, labelConverter}); |
|  |  |
|  | model = pipeline.fit(data); |
|  | //DecisionTreeClassificationModel treeModel = (DecisionTreeClassificationModel) (model.stages()[2]); |
|  | //System.out.println("Learned classification tree model:\n" + treeModel.toDebugString()); |
|  | } |
|  |  |
|  | public Dataset<Row> predict(String datasetPath) { |
|  |  |
|  | List<StructField> fields = Arrays.asList( |
|  | DataTypes.createStructField("no", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("time", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("source", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("destination", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("protocol", DataTypes.IntegerType, true), |
|  | DataTypes.createStructField("lenght", DataTypes.DoubleType, true), |
|  | DataTypes.createStructField("info", DataTypes.DoubleType, 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); |
|  |  |
|  | \*/ |
|  |  |
|  |  |
|  | //StringIndexerModel indexerModel = indexer\_training.fit(data); |
|  |  |
|  | // Make predictions. |
|  | Dataset<Row> predictions = model.transform(testData); |
|  |  |
|  | // Select example rows to display. |
|  | predictions.select("prediction", "features").show(5); |
|  |  |
|  | // Select (prediction, true label) and compute test error. |
|  | MulticlassClassificationEvaluator evaluator = new MulticlassClassificationEvaluator() |
|  | .setLabelCol("label") |
|  | .setPredictionCol("prediction") |
|  | .setMetricName("accuracy"); |
|  | double accuracy = evaluator.evaluate(predictions); |
|  | System.out.println("Test Error = " + (1.0 - accuracy)); |
|  |  |
|  | System.out.println("\*\*\*EVALUATION METRICS\*\*\*"); |
|  | System.out.println("ACCURACY: " + accuracy); |
|  | evaluator.setMetricName("weightedPrecision"); |
|  | double weightedPrecision = evaluator.evaluate(predictions); |
|  | System.out.println("WEIGHTED PRECISION: " + weightedPrecision); |
|  | evaluator.setMetricName("weightedRecall"); |
|  | double weightedRecall = evaluator.evaluate(predictions); |
|  | System.out.println("WEIGHTED RECALL: " + weightedRecall); |
|  | evaluator.setMetricName("f1"); |
|  | double f1 = evaluator.evaluate(predictions); |
|  | System.out.println("F1: " + f1); |
|  |  |
|  |  |
|  | DecisionTreeClassificationModel treeModel = (DecisionTreeClassificationModel) (model.stages()[2]); |
|  | System.out.println("Learned classification tree model:\n" + treeModel.toDebugString()); |
|  |  |
|  | return predictions; |
|  | } |
|  |  |
|  | public Dataset<Row> predict\_realtime(Dataset data) { |
|  | //StringIndexerModel indexerModel = indexer\_training.fit(data); |
|  |  |
|  | Dataset<Row> predictions = model.transform(data); |
|  |  |
|  | // Select example rows to display. |
|  | predictions.select("prediction", "features").show(5); |
|  | //predictions.printSchema(); |
|  | //predictions.show(5); |
|  |  |
|  | return predictions; |
|  | } |
|  |  |
|  | public void save(String path) throws Exception { |
|  | //model.write().overwrite().save(path); |
|  | model.save(path); |
|  | } |
|  |  |
|  | public void load(String path) throws Exception { |
|  | model = PipelineModel.load(path); |
|  | } |
|  |  |
|  | private Dataset preprocess(Dataset data) { |
|  | return data; |
|  | } |
|  | } |