Java Machine Learning Connector (JMLC)

Overview

The `Java Machine Learning Connector (JMLC)` API is a programmatic interface for interacting with SystemML in an embedded fashion. To use JMLC, the small footprint “in-memory” SystemML jar file needs to be included on the classpath of the Java application, since JMLC invokes SystemML in an existing Java Virtual Machine. Because of this, JMLC allows access to SystemML’s optimizations and fast linear algebra, but the bulk performance gain from running SystemML on a large Spark or Hadoop cluster is not available. However, this embeddable nature allows SystemML to be part of a production pipeline for tasks such as scoring.

The primary purpose of JMLC is as a scoring API, where your scoring function is expressed using SystemML’s DML (Declarative Machine Learning) language. Scoring occurs on a single machine in a single JVM on a relatively small amount of input data which produces a relatively small amount of output data. For consistency, it is important to be able to express a scoring function in the same DML language used for training a model, since different implementations of linear algebra (for instance MATLAB and R) can deliver slightly different results.

In addition to scoring, embedded SystemML can be used for tasks such as unsupervised learning (for example, clustering) in the context of a larger application running on a single machine.

Performance penalties include startup costs, so JMLC has facilities to perform some startup tasks once, such as script precompilation. Due to startup costs, it tends to be best practice to do batch scoring, such as scoring 1000 records at a time. For large amounts of data, it is recommended to run DML in one of SystemML’s distributed modes, such as Spark batch mode or Hadoop batch mode, to take advantage of SystemML’s distributed computing capabilities. JMLC offers embeddability at the cost of performance, so its use is dependent on the nature of the business use case being addressed.

Examples

JMLC is patterned loosely after JDBC. To interact with SystemML via JMLC, we can begin by creating a `Connection` object. We can then prepare (precompile) a DML script by calling the `Connection`'s `prepareScript` method, which returns a `PreparedScript` object. We can then call the `executeScript` method on the `PreparedScript` object to invoke this script.

Here, we see a "hello world" example, which invokes SystemML via JMLC and prints "hello world" to the console.

```
Connection conn = new Connection();
String dml = "print('hello world');";
PreparedScript script = conn.prepareScript(dml, new String[0], new String[0], false);
script.executeScript();
```

Next, let's consider a more practical example. Consider the following DML script. It takes input data matrix `X` and input model matrix `W`. Scores are computed, and it returns a (n x 1) matrix `predicted_y` consisting of the column indexes of the maximum values of each row in the `scores` matrix. Note that since values are being read in and out programmatically, we can ignore the parameters in the `read` and `write` statements.

DML

```
X = read("./tmp/X", rows=-1, cols=-1);
W = read("./tmp/W", rows=-1, cols=-1);

numRows = nrow(X);
numCols = ncol(X);
b = W[numCols+1,]
scores = X %*% W[1:numCols,] + b;
predicted_y = rowIndexMax(scores);

write(predicted_y, "./tmp", format="text");
```

In the Java below, we initialize SystemML by obtaining a `Connection` object. Next, we read in the above DML script (`"scoring-example.dml"`) as a `String`. We precompile this script by calling the `prepareScript` method on the `Connection` object with the names of the inputs (`"W"` and `"X"`) and outputs (`"predicted_y"`) to register.

Following this, we set matrix `"W"` and we set a matrix of input data `"X"`. We execute the script and read the resulting `"predicted_y"` matrix. We repeat this process. When done, we close the SystemML `Connection`.

Java

```
 package org.apache.sysml.example;

 import java.util.Random;

 import org.apache.sysml.api.jmlc.Connection;
 import org.apache.sysml.api.jmlc.PreparedScript;

 public class JMLCExample {

    public static void main(String[] args) throws Exception {

        // obtain connection to SystemML
        Connection conn = new Connection();

        // read in and precompile DML script, registering inputs and outputs
        String dml = conn.readScript("scoring-example.dml");
        PreparedScript script = conn.prepareScript(dml, new String[] { "W", "X" }, new S
tring[] { "predicted_y" }, false);

        double[][] mtx = matrix(4, 3, new double[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 });
        double[][] result = null;

        // set inputs, execute script, and obtain output
        script.setMatrix("W", mtx);
        script.setMatrix("X", randomMatrix(3, 3, -1, 1, 0.7));
        result = script.executeScript().getMatrix("predicted_y");
        displayMatrix(result);

        script.setMatrix("W", mtx);
        script.setMatrix("X", randomMatrix(3, 3, -1, 1, 0.7));
        result = script.executeScript().getMatrix("predicted_y");
        displayMatrix(result);

        script.setMatrix("W", mtx);
        script.setMatrix("X", randomMatrix(3, 3, -1, 1, 0.7));
        result = script.executeScript().getMatrix("predicted_y");
        displayMatrix(result);

        // close connection
        conn.close();
    }

     public static double[][] matrix(int rows, int cols, double[] vals) {
        double[][] matrix = new double[rows][cols];
        if ((vals == null) || (vals.length == 0)) {
            return matrix;
        }
        for (int i = 0; i < vals.length; i++) {
            matrix[i / cols][i % cols] = vals[i];
        }
        return matrix;
    }
```

```
    public static double[][] randomMatrix(int rows, int cols, double min, double max, do
uble sparsity) {
        double[][] matrix = new double[rows][cols];
        Random random = new Random(System.currentTimeMillis());
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (random.nextDouble() > sparsity) {
                    continue;
                }
                matrix[i][j] = (random.nextDouble() * (max - min) + min);
            }
        }
        return matrix;
    }

    public static void displayMatrix(double[][] matrix) {
        System.out.println("Matrix size:" + matrix.length + "x" + matrix[0].length);
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (j > 0) {
                    System.out.print(", ");
                }
                System.out.print("[" + i + "," + j + "]:" + matrix[i][j]);
            }
            System.out.println();
        }
    }

 }
```

For additional information regarding programmatic access to SystemML, please see the Spark MLContext Programming Guide (spark-mlcontext-programming-guide.html).