IBM Big SQL

Use Case. Outdoor Equipement

> IBM Big SQL 5.0.1
>
> Spark 2
>
> IBM Data Server Manager 2.1.3

The Outdoor Equipment analytics team wants to incentivize sales by sending targeted coupons. That is, they want to somehow predict what coupon a customer is more likely to use, send such coupon to the customer, and hopefully lead to a sale.

The team has been collecting demographic data of their customers as well as the product categories they tend to buy.

SQL queries

1. Features:

```
DROP TABLE features;
CREATE EXTERNAL HADOOP TABLE features
    (ID VARCHAR(32),
    GENDER VARCHAR(1),
    AGE INT,
    MARITAL_STATUS VARCHAR(32),
    PROFESSION VARCHAR(32))
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    LOCATION '/user/bigsql/couponing/features';
SELECT * FROM features;
```

2. Query MLModel (Spark2):

```
SELECT model.*
 FROM TABLE(SYSHADOOP.EXECSPARK(
                                class => 'QueryMLModel',
                                model_uri => CAST('/models/CrossValidatedDecis
ionTree.sparkmodel' AS SYSHADOOP.URI),
                                feature_names => 'id,gender,age,marital_status
,profession',
                                feature_values => TABLE(SELECT * FROM bigsql.f
eatures) AS in,
                                label_name => 'predicted_label',
                                label_type => 'string'
                               )
           ) AS model;
```

3. Customers:

```
DROP TABLE customers;
CREATE EXTERNAL HADOOP TABLE customers
    (ID VARCHAR(32),
    FIRST_NAME VARCHAR(32),
    LAST_NAME VARCHAR(32),
    EMAIL VARCHAR(32))
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    LOCATION '/user/bigsql/couponing/customers';
SELECT * FROM CUSTOMERS;
```

4. Customers MLModel:

```
SELECT customers.*,
       model.predicted_label
FROM TABLE(SYSHADOOP.EXECSPARK(
                                class => 'QueryMLModel',
                                model_uri => CAST('/models/CrossValidatedDecis
ionTree.sparkmodel' AS SYSHADOOP.URI),
                                feature_names => 'id,gender,age,marital_status
,profession',
                                feature_values => TABLE(SELECT * FROM features
) as in,
                                label_name => 'predicted_label',
                                label_type => 'string'
                              )
          ) AS model
INNER JOIN customers ON model.id = customers.id;
```

5. Coupons (MongoDB):

```
CREATE SERVER "MONGODB_SERVER_COUPONING" TYPE MONGODB WRAPPER "ODBC" OPTIONS(DBNAME 'couponing', HOST 'hdp01.ibm.es', PORT '27017');
CREATE NICKNAME couponing_coupons FOR mongodb_server_couponing.couponing.coupons;
-- SET TYPES
ALTER NICKNAME couponing_coupons ALTER COLUMN "_ID" LOCAL TYPE INTEGER;
ALTER NICKNAME couponing_coupons ALTER COLUMN "CATEGORY" LOCAL TYPE VARCHAR(32);
ALTER NICKNAME couponing_coupons ALTER COLUMN "PRODUCT" LOCAL TYPE VARCHAR(32);
-- CHECK
SELECT * FROM couponing_coupons;
```

6. Customers Predictions:

```
DROP TABLE predicted_coupon_matches;
CREATE HADOOP TABLE predicted_coupon_matches
    (ID VARCHAR(32),
    FIRST_NAME VARCHAR(32),
    LAST_NAME VARCHAR(32),
    EMAIL VARCHAR(32),
    PRODUCT VARCHAR(32),
    DISCOUNT INT,
    PREDICTED_LABEL VARCHAR(32));
INSERT INTO predicted_coupon_matches (
    SELECT customers.*,
            filtered_mongo_coupons.product, filtered_mongo_coupons.discount,
            model.predicted_label
    FROM TABLE(SYSHADOOP.EXECSPARK(
                                    class => 'QueryMLModel',
                                    model_uri => CAST('/models/CrossValidatedD
ecisionTree.sparkmodel' AS SYSHADOOP.URI),
                                    feature_names => 'id,gender,age,marital_st
atus,profession',
                                    feature_values => TABLE(SELECT * FROM feat
ures) as in,
                                    label_name => 'predicted_label',
                                    label_type => 'string'
                                  )
              ) AS model

    INNER JOIN customers ON model.id = customers.id
    RIGHT JOIN (    SELECT all_coupons.*
                    FROM couponing_coupons AS all_coupons
                    INNER JOIN
                      ( SELECT category, min("_ID") as coupon_id
                        FROM couponing_coupons
                        WHERE expired != 1
                        GROUP BY category
                      ) AS unique_coupon_per_category
                    ON all_coupons."_ID" = CAST(unique_coupon_per_category.cou
pon_id AS VARCHAR(32))
                ) as filtered_mongo_coupons
    ON filtered_mongo_coupons.category = model.predicted_label
);
SELECT * FROM predicted_coupon_matches;
```

7. Customers Notification:

```
DROP TABLE predicted_coupon_matches_dummy;
CREATE HADOOP TABLE predicted_coupon_matches_dummy
    (ID VARCHAR(32),
    FIRST_NAME VARCHAR(32),
    LAST_NAME VARCHAR(32),
    EMAIL VARCHAR(32),
    PRODUCT VARCHAR(32),
    DISCOUNT INT,
    PREDICTED_LABEL VARCHAR(32));
INSERT INTO predicted_coupon_matches_dummy VALUES ('XXXYYY', 'Francisco', 'Can
o', 'francisco.cano@ibm.com', 'Kelty TN2 Tent', 20, 'Camping Equipment');
CALL PROC16('bigsql','predicted_coupon_matches_dummy');
```

Code

1. Code for the `Spark QueryMLModel` scala class

```
package com.ibm.biginsights.bigsql.examples

import com.ibm.biginsights.bigsql.spark.SparkPtf
import org.apache.log4j.Logger
import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.types.StructField
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.types.DataTypes
import org.apache.spark.sql.SQLContext
import scala.collection.JavaConversions._
import org.apache.spark.ml.linalg.DenseVector
import org.apache.spark.sql.functions._

import scala.collection.mutable.ArrayBuffer

import org.apache.spark.ml.tuning.CrossValidatorModel

/**
 * A PTF that queries a Spark MLlib model for predictions based on the incomin
g features.
 *
 * How to use:
 *
 * Make sure that you have persisted a {@link org.apache.spark.ml.tuning.Cross
ValidatorModel} model from Spark.
 * Currently this PTF cannot return complex types.
 * Specify the feature values by sending a TABLE.
 * Also specify the name of the model's label as well as its type.
 *
 * Invocation example:
 *
 select model.predicted_label, model.probability
 from table(SYSHADOOP.EXECSPARK(
                                class => 'com.ibm.biginsights.bigsql.examples.
QueryMLModelScala',
                                model_uri => CAST('/models/CrossValidatedDecis
ionTree.sparkmodel' AS SYSHADOOP.URI),
                                feature_names => 'gender,age,marital_status,pr
ofession',
                                feature_values => TABLE(SELECT * FROM features
) as in,
                                label_name => 'predicted_label',
                                label_type => 'string'
                               )) as model
 *
 */
class QueryMLModel extends SparkPtf {
  val LOG = Logger.getLogger(classOf[QueryMLModel])

  override def describe(ctx: SQLContext, arguments: java.util.Map[String, Obje
ct]): StructType = {
    LOG.debug(s"TableInput::describe. Scalar arguments: $arguments")

    // make sure all arguments came in
    if (!arguments.containsKey("LABEL_NAME")
      || !arguments.containsKey("LABEL_TYPE")
      || !arguments.containsKey("FEATURE_VALUES"))
    {
      throw new RuntimeException("Describing requires parameters: LABEL_NAME,
LABEL_TYPE, FEATURE_VALUES");
    }

    // Return back the input features as well.
    val incomingSchema : StructType = arguments.get("FEATURE_VALUES").asInstan
ceOf[StructType]
    var outgoingSchema : StructType = new StructType
    incomingSchema.foreach { t => outgoingSchema = outgoingSchema.add(t) }

    // create fields for the label and label probability
    val labelField: StructField = createStructField(arguments.get("LABEL_NAME"
).asInstanceOf[String], arguments.get("LABEL_TYPE").asInstanceOf[String])
    val probabilityField: StructField = createStructField("PROBABILITY", "STRI
NG")
    outgoingSchema = outgoingSchema.add(labelField)
    outgoingSchema = outgoingSchema.add(probabilityField)

    LOG.debug("TableInput::describe. Scalar arguments = " + outgoingSchema)
    outgoingSchema
  }

  override def execute(ctx: SQLContext, arguments: java.util.Map[String, Objec
t]): DataFrame = {
    LOG.debug("TableInput::execute")

    // make sure all arguments came in
    if (!arguments.containsKey("FEATURE_NAMES")
      || !arguments.containsKey("FEATURE_VALUES")
      || !arguments.containsKey("MODEL_URI")) {
      throw new RuntimeException("Executing requires parameters: FEATURE_NAMES
, FEATURE_VALUES, MODEL_URI");
    }

    val featureNamesString: String = arguments.get("FEATURE_NAMES").asInstance
Of[String]
    val featureNames: Array[String] = featureNamesString.split(",")

    // Dataset
    var df = arguments.get("FEATURE_VALUES").asInstanceOf[DataFrame]
    LOG.debug("FEATURE_VALUES schema= " + df.schema)

    // hack hack hack
    // change column naming from the incoming "_0", "_1", "_2", etc. pattern i
nto the actual names
    if (df.columns(0).equals("_0")) {
      for (i <- 0 to (df.columns.length - 1))
      {
        df = df.withColumnRenamed("_" + i, featureNames(i))
      }
    }

    // load specified model
    val model: CrossValidatorModel = CrossValidatorModel.load(arguments.get("M
ODEL_URI").asInstanceOf[java.net.URI].toString)

    // predict
    var result: DataFrame = model.transform(df)
    LOG.debug("Result schema=" + result.schema)

    // convert the probability DenseVector into a consumable String
    val vectorToString = udf{ (v:DenseVector) => v.toString }
    result = result.withColumn("probability", vectorToString(col("probability"
)))

    // do selection of only the interesting fields
    var columnsToReturn: ArrayBuffer[String] = ArrayBuffer[String]()
    featureNames.foreach { f => columnsToReturn.add(f) }
    val labelName: String = arguments.get("LABEL_NAME").asInstanceOf[String]
    columnsToReturn.add(labelName)
    columnsToReturn.add("probability")
    val columnsToReturnAsSequence = columnsToReturn.toSeq
    result = result.select(columnsToReturnAsSequence.head, columnsToReturnAsSe
quence.tail: _*)

    result
  }

  override def destroy(ctx: SQLContext, arguments: java.util.Map[String, Objec
t]): Unit = {
    // NO-OP
  }

  override def cardinality(ctx: SQLContext, arguments: java.util.Map[String, O
bject]): Long = {
    val cardArg = arguments.get("CARD")
    if (cardArg != null) {
      cardArg.asInstanceOf[Int]
    } else {
      100
    }
  }

  def createStructField(name: String, fieldType: String): StructField =
    {
      if (fieldType == null || fieldType.length() == 0)
      {
        return DataTypes.createStructField(name, DataTypes.NullType, true)
      }

      val fieldTypeUpper = fieldType.toUpperCase();
      val structField : StructField = fieldTypeUpper match {
        case "STRING" => DataTypes.createStructField(name, DataTypes.StringTyp
e, true)
        case "VARCHAR" => DataTypes.createStructField(name, DataTypes.StringTy
pe, true)
        case "BINARY" => DataTypes.createStructField(name, DataTypes.BinaryTyp
e, true)
        case "BOOLEAN" => DataTypes.createStructField(name, DataTypes.BooleanT
ype, true)
        case "DATE" => DataTypes.createStructField(name, DataTypes.DateType, t
rue)
        case "TIMESTAMP" => DataTypes.createStructField(name, DataTypes.Timest
ampType, true)
        case "DOUBLE" => DataTypes.createStructField(name, DataTypes.DoubleTyp
e, true)
        case "FLOAT" => DataTypes.createStructField(name, DataTypes.FloatType,
 true)
        case "BYTE" => DataTypes.createStructField(name, DataTypes.ByteType, t
rue)
        case "INTEGER" => DataTypes.createStructField(name, DataTypes.IntegerT
ype, true)
        case "LONG" => DataTypes.createStructField(name, DataTypes.LongType, t
rue)
        case "SHORT" => DataTypes.createStructField(name, DataTypes.ShortType,
 true)
        case _ => DataTypes.createStructField(name, DataTypes.NullType, true);
      }

      structField
    }
}
```

2. `Telegram` java class procedure

```
package com.summit.send;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import COM.ibm.db2.app.UDF;

public class Notification extends UDF {

    static class NotificationMessageResult
    {
        public NotificationMessageResult(int incoming_id, int response_code)
        {
            this.incoming_id = incoming_id;
            this.response_code = response_code;
        }

        int incoming_id;
        int response_code;
    }

    public static void exec(String schemaName, String tableName, ResultSet[] o
utResults) throws IOException {
        ResultSet rs = null;
        try
        {
            Connection con = DriverManager.getConnection("jdbc:default:connect
ion");

            StringBuilder query = new StringBuilder();
            query.append("SELECT * from ").append(schemaName).append(".").appe
nd(tableName);

            PreparedStatement statement = con.prepareStatement(query.toString(
));
            rs = statement.executeQuery();

            List<NotificationMessageResult> results = new ArrayList<Notificati
onMessageResult>();
            while (rs.next())
            {
                int incomingId = rs.getInt("id");
                String email = rs.getString("email");
                String firstName = rs.getString("first_name");
                String lastName = rs.getString("last_name");
                String productName = rs.getString("product");
                int discount = rs.getInt("discount");
                int responseCode = sendMessage(incomingId, firstName, lastName
, email, productName, discount);
                NotificationMessageResult result = new NotificationMessageResu
lt(incomingId, responseCode);
                results.add(result);
            }
            outResults[0] = toResultSet(con, results);
        }
        catch (SQLException e)
        {
            // catch
            outResults[0] = null;
        }
    }

    public static final String BASEURL = "https://api.telegram.org/bot";
    public static final String TOKEN = "XXXYYYZZZ";
    public static final String PATH = "sendmessage";
    public static final String CHATID_FIELD = "chat_id";
    public static final String TEXT_FIELD = "text";

    public static int sendMessage(int id, String firstName, String lastName, S
tring email, String productName,
                                       int discount) throws IOException
    {
        String url = BASEURL + TOKEN + "/" + PATH;
        HttpPost httppost = new HttpPost(url);
        httppost.addHeader("Content-type", "application/x-www-form-urlencoded"
);
        httppost.addHeader("charset", "UTF-8");
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
        nameValuePairs.add(new BasicNameValuePair(CHATID_FIELD, id + ""));
        nameValuePairs.add(new BasicNameValuePair(TEXT_FIELD, String.format("H
ello %s, you have a %d discount on %s ",
                firstName, discount, productName)));
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs, "UTF-8"));
        HttpClient client = new DefaultHttpClient();
        HttpResponse response = client.execute(httppost);

        return response.getStatusLine().getStatusCode();
    }

    protected static ResultSet toResultSet(Connection con, List<NotificationMe
ssageResult> results) throws SQLException
    {
        if (results.size() == 0)
        {
            return null;
        }

        StringBuilder sb = new StringBuilder();

        sb.append("SELECT * FROM TABLE(VALUES");
        int count = 0;
        for (NotificationMessageResult r : results)
        {
            if (count > 0) sb.append(',');
            ++count;
            sb.append("\n  (");
            sb.append(r.incoming_id).append(", ");
            sb.append("'").append(r.response_code).append("'");
            sb.append(')');
        }
        sb.append("\n) AS T1 (INCOMING_ID, REST_CALL_RESPONSE_CODE)");

        String query = sb.toString();
        sb = null;

        Statement stmt = con.createStatement();
        try
        {
            return stmt.executeQuery(query);
        }
        catch (SQLException e)
        {
            return null;
        }
    }
}
```

Try by yourself! Try with the SandBox!

Meta

Francisco Cano – @fjcano – francisco.cano@ibm.com

Arancha Ocaña – @arancha_ocana – arancha_ocana@es.ibm.com