ORAAH 2.7.0 Change List Summary

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

ORAAH 2.7.0 release marks another major step to bring Spark based high performance analytics to the Oracle R Advanced Analytics platform. ORAAH continues to support legacy Hadoop mapReduce based analytics as well in the 2.x product releases, but we encourage our customers to move their code base to use the Spark versions (when available) of same functionality.

ORAAH 2.7.0 introduces a new neural network family orch.neural2. With orch.neural2, the model building and scoring are fully Apache Spark-enabled. Since the codebase involves R, C and Java, we redesigned the code to connect R and Java using the rJava package, and connect Java with C using JNI. The core calculation remains in C while the main iteration loop stays in Java. Like orch.lm2 and orch.glm2, the new function takes advantage of the RDD-based Oracle distributed model matrix and Oracle R formula distributed parser. We also provide full support for a summary method to associate the weights with the network nodes.

ORAAH 2.7.0 also introduces a new high performance linear regression, based on parallel distributed normal equations and Cholesky factorization orch.lm2. Being fully Apache Spark-enabled, ORAAH LM can efficiently handle both numeric and high cardinality factor variables. ORAAH LM automatically switches to out-of-core mode if the input data does not fit into distributed memory.

ORAAH 2.7.0 also extends support for Apache Spark MLlib functions, Oracle Formula support, and Distributed Model Matrix data structure. Support for summary on models built with ORAAH analytics provides near identical information that users may expect from an R model summary. Oracle formula support is greatly improved, including more syntax expressions.

1 ORAAH 2.7.0 Changes

1.1 ORCHcore

ORCH core includes a number of bug fixes and code improvement as lists below.

1.1.1 Non backward-compatible API changes:

- None.

1.1.2 Backward-compatible API changes:

- hdfs.push()

Fixed error reporting in `hdfs.push` if user tries to push a Hive table to HDFS. Also fixed error reporting if `hdfs.toHive` is used for data without metadata. `na.strings` metadata was wrongly set for a table having null values pushed into HDFS using `hdfs.push()` This caused faulty interpretation of data and failure of analytics. This issue was resolved in this release.

1.1.3 New API functions:

- None.

1.1.4 Other changes:

- Spark logging.

Now Spark console logging is turned off by default but you can choose to turn it on by specifying spark.ui showConsoleProgress = "true" within spark.connect() additional parameters.

- Support for Cloudera parcels.

Auto detection of Hadoop environment has been improved for Cloudera parcels and RPM-based installation of Cloudera Distribution for Hadoop. Setting of `HADOOP_HOME` / `ORCH_HADOOP_HOME` and `ORCH_STREAMING_LIB` will not be necessary for standard RPM and CDH parcel installs.

- New packages included in "supporting" archive.

New Hive transparency layer uses JDBC/Thrift connection to work with Apache Hive instead of Hive CLI. So now ORAAH has new supporting packages (RJDBC and DBI) in supporting packages zip. The ORAAH client installer installs these packages for the user as a part of supporting packages install.

- Antlr library is now a part of ORAAH.

New Oracle Formula and Distributed Model Matrix use Antlr runtime. Also it is needed by the Spark application for ORAAH MLlib analytics. Antlr runtime is a part of ORCHcore package and is added to the application when Spark context is created.

- Kryo serializer.

ORAAH will not automatically configure Spark session to use Kryo Serializer by default to further improve our analytics performance. While the performance boost varies and is not guaranteed in all execution situations, we have seen improvement up to 15% in overall analytics performance.

- More control over JVM used within ORAAH.

New environment variables can be used to control the rJava JVM memory parameters *-Xmx* and *-XX:MaxPermSize* using ORCH_JAVA_XMX (Default 1G, if not specified) and ORCH_JAVA_MAX_PERM (defaults to 256M, if not specified), respectively.

- L-GPL version of rJava.

L-GPL version of rJava. ORAAH ships with the L-GPL version of the rJava package.

1.2 ORCHstats

With the ORAAH 2.6.0 release we introduced support for selected algorithms from Apache MLlib with support from our proprietary optimized data transformation and data storage algorithms. This interface is built on top of Apache Spark and designed to improve performance and interoperability of MLlib and ORAAH machine learning functionality with R. Building on the initial success of the Apache MLlib integration, ORAAH 2.6.0 expands coverage of exported MLlib algorithms and greatly improves support for the R formula engine. As of ORAAH 2.7.0 you can also use MLlib Gaussian Mixture Model (GMM) to build models and perform predictions using GMM models.

Decision Tree and Random Forest now automatically detect impurity and number of classes. The overloaded summary function can be used on MLlib models as well (though support for summary is very limited due to limited support of summary in MLlib 1.6.x).

Machine learning and statistical algorithms require a Distributed Model Matrix (DMM) for their training phase. For supervised learning algorithms, DMM captures a target variable and explanatory terms; for unsupervised learning DMM captures explanatory terms only. Distributed Model Matrices have their own implementation of the R formula, which closely follows the R formula syntax, but is much more efficient from the performance perspective. See the reference manual for detailed information about features supported in the Oracle formula.

Another major change in ORAAH 2.7.0 is a new neural network family orch.neural2. With orch.neural2, the model building and scoring are completely implemented with Spark. It no longer relies on any Hadoop MapReduce implementation. Since the codebase involves R, C and Java, we redesigned the code to connect R and Java using the rJava package, and connect Java with C using JNI. The core calculation remains in C while the main iteration loop stays in Java. Like orch.lm2 and orch.glm2, the new function uses the RDD-based distributed model matrix and the Oracle R formula parser. It shares the same LBFGS solver with the upcoming neural network function in Oracle Advanced Analytics. We also provide a summary method to associate the weights with the network nodes.

Table 1: API difference

||**Neural 1**|**Neural 2**	
|formula|Y|Y	
|dfs.dat /data|Y (“dfs.dat”)|Y (arg name change to “data”	consistent with lm2	**glm2)**
|weight|Y|Y	
|xlev|Y|N	
|hiddenSizes|Y|Y	
|activations|Y|Y	
|gradTolerance|Y (default: 0.1	details as below)|Y (default: 1E-9	**details as below)**
|maxIterations|Y|Y	
|objMinProgress|Y (default: 1E-6	details below)|Y (default: 1E-6	**details below)**
|lowerBound|Y|Y	
|upperBound|Y|Y	
|nUpdates|Y|Y	
|scaleHessian|Y|Y	
|Trace / verbose|Y (“trace”)|Y (arg name change to “verbose”	consistent with lm2	**glm2)**
|nMappers|Y|N	
|nReducers|Y|N	
|mapSplit|Y|N	
|seed|N|Y	
|maxBlockRows|N|Y	

Defaults are same if not noted

STOPPING CRITERIA

1. gradTolerance: In nerual2, the gradient is normalized by multiplying 1/N (the number of rows). So the default tolerance is changed from 0.1 to 1E-9.
2. objMinProgress In neural1, the objMinProgress is calculated as a relative error (normalized by the value of objective function) if the value of objective function is greater than 1, and it is calculated as an absolute error if the value of the objective function is less than 1.

In neural2, the cutoff of relative error and absolute error is changed from 1 to 1E-12. And the objective function is normalized by multiplying 1/N (the number of rows). The default value is unchanged as 1E-6.

Prepare function In orch.neural, in order to use the spark implementation, users needed to invoke orch.prepare.model.matrix() to generate a model matrix in the RDD structure, then feed the resulting model matrix in “dfs.data” to invoke orch.neural. In orch.neural2, the user can directly invoke orch.neural2() with a dfs.id data object. There is another option that allows a user to invoke a new prepare function orch.prepare.model.matrix2 to generate a new distributed model matrix, then feed into orch.neural2(). The benefit of the latter is that the user needs only to generate a persisted model matrix once, and can invoke orch.neural2() multiple times with different network configurations.

Note
Nerual2 no longer needs orch.getXlevels().

1.2.1 Non backward-compatible API changes:

- orch.neural2.

The 'orch.neural2' function solves multilayer feed-forward neural network models. It supports an arbitrary number of hidden layers and an arbitrary number of neurons per layer. Each layer could be assigned a different activation function. The L-BFGS algorithm is used to solve the underlying unconstrained nonlinear optimization problem.

1.2.2 Backward-compatible API changes:

- hdfs.write()

hdfs.write() now writes out the metadata alongside the data files at the location specified by *outPath* parameter. This function now also returns an ORCH hdfs.id object after the data is successful written into hdfs. This object can be used to directly perform hdfs operations (hdfs.get, hdfs.head, hdfs.describe, hdfs.meta, etc...) and an additional step of hdfs.attach() or hdfs.id() is avoided (even the sampling caused by hdfs.attach). This makes using the predicted data easier and significantly faster.

Example:

```
# regression with iris dataset
IRIS <- hdfs.put(iris)
mod<- orch.neural2(formula        = Sepal.Length ~.,
                   data           = IRIS,
                   hiddenSizes    = c(10, 10),
                   activations    = c("sigmoid", "tanh", "linear"),
                   seed           = 0,
                   objMinProgress = 1e-5,
                   maxIterations  = 400,
                   verbose        = TRUE)
summary(mod)
p <- predict(mod, IRIS, supplemental=c("Species", "Sepal.Length"))
IrPred.dfs <- hdfs.write(p, "IrPred", overwrite=TRUE)
IrPred <- hdfs.get(IrPred.dfs)

# binary classification with kyphosis dataset
library(rpart)
KYPHOSIS <- hdfs.put(kyphosis)
mod<- orch.neural2(formula        = Kyphosis ~.,
                   data           = KYPHOSIS,
                   hiddenSizes    = c(20, 20),
                   activations    = c("sigmoid", "sigmoid", "entropy"),
                   seed           = 0,
                   verbose        = TRUE)
p <- predict(mod, KYPHOSIS, supplemental=c("Age", "Kyphosis"))
KyPred.dfs <- hdfs.write(p, "KyPred", overwrite=TRUE)
KyPred <- hdfs.get(KyPred.dfs)
```

1.2.3 New API functions:

- orch.lm2()

High performance linear regression, based on parallel distributed normal equations and Cholesky factorization. orch.lm2 is used to carry out linear regression 'y = X * beta + e', where 'y' is the response, 'X' is the design matrix, 'beta' is a vector of regression coefficients, and 'e' is the error. The implementation is based on parallel distributed normal equations '$X^{T * X * beta = X}T$ y', and parallel supernodal Cholesky factorization. orch.lm2 can efficiently handle both numeric and high cardinality factor variables. orch.lm2 automatically switches to the out-of-core mode, if the input data does not fit into the distributed memory.

Example:

```
library(rpart)
data <- hdfs.put(kyphosis)
model <- orch.lm2(formula = Age ~ log(Number) + Kyphosis, data = data)
pred <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

- orch.ml.gmm()

MLlib Multivariate GMM.MLlib Gaussian Mixture Model. Gaussian mixture models (GMM) are often used for data clustering. Usually, fitted GMMs cluster by assigning query data points to the multivariate normal components that maximize the component posterior probability given the data. Moreover, GMM clustering can accommodate clusters that have different sizes and correlation structures within them. Like most clustering methods, you must specify the number of desired clusters before fitting the model. The number of clusters specifies the number of components in the GMM.

Example:

```
library(rpart)
data <- hdfs.put(kyphosis)
model <- orch.ml.gmm(formula = ~ Number + Age, data = data)
pred <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

- demo(orch_mllib)

Demo "orch_mllib" has been added to demonstrate the usage of different MLlib analytics exposed through ORAAH. Various kinds of models are created and used for prediction. This demo also demonstrates how to persist a model for scoring later in a new session.

- summary(<MLlib model>)

The summary function for MLlib models created using ORAAH prints the results as exposed by the MLlib API. Summary information provided by MLlib is rather limited compared to summary information available for ORAAH models built with orch.lm2, orch.glm2, and orch.neural2.

Example:

```
library(rpart)
data <- hdfs.put(kyphosis)
model <- orch.ml.svm(formula = Kyphosis ~ Number + Age, data = data)
summary(model)
```

- summary(<ORAAH model>)

The summary function for native ORAAH models prints full summary results, very closely matching R models summary statistics. Summary is currently supported for orch.glm2, orch.lm2, and orch.neural2.

Example:

```
library(rpart)
data <- hdfs.put(kyphosis)
model <- orch.glm2(formula = Kyphosis ~ Number + Age, data = data)
summary(model)
```

1.2.4 Other changes:

- Intel MKL updated.

Intel MKL was upgraded and new library files added to the installer package: Intel® Math Kernel Library Version 2017.0.0 Build 20160801 for Intel® 64 architecture applications.

1.3 OREbase

OREbase is now includes better support for Hive, especially on BDA secure cluster with enabled SSL connection and Keberos authentication. OREbase also includes a number of bugfixes and improvements across the codebase. See below for more details.

1.3.1 Non backward-compatible API changes:

- None.

1.3.2 Backward-compatible API changes:

- ore.is.connected()

Users can check if the ORE transparency layer is connected to Hive or to Oracle RDBMS by specifying "type" argument in ore.is.connected function. ore.is.connected(type="HIVE") returns TRUE only if the ORE transparency layer is currently connected to Hive and FALSE otherwise. Not specifying the argument results in TRUE if the ORE transparency layer is connected to any of the supported backend databases.

Example:

```
ore.is.connected() # FALSE
ore.connect(
  user = "demo",
  password = "demo",
  host = "demo.us.oracle.com",
  port = 10000,
  schema = "default",
  type = "HIVE")
ore.is.connected(type="HIVE")    # TRUE
ore.is.connected(type="ORACLE") # FALSE
ore.is.connected()               # TRUE
```

- ore.connect()

ore.connect() for type="HIVE" now supports additional parameters for users to connect to HiveServer2 running in various modes such as: Kerberos authentication, SSL, or any other Hiveserver2 mode. Users can specify parameters like "principal" to specify the Kerberos server principal for the host where HiveServer2 is running, "sslTrustStore" to specify the path to the client's truststore file and trustStorePassword to specify the password for the truststore. Check the Hive Documentation for the various modes and the parameters needed to connect in those modes.

Example:

```
ore.connect(
  user = "demo",
  password = "demo",
  host = "demo.us.oracle.com",
  port = 10000,
  schema = "default",
  type = "HIVE",
  principal = "hive/demo.us.oracle.com@DEV.ORACLE.COM",
  ssl = "true",
  sslTrustStore = "/opt/cloudera/security/jks/sharsecur.truststore",
  trustStorePassword="PASSWORD")
```

1.3.3 Other changes:

- Automated Hive JDBC driver lookup.

ORAAH now checks for the Hive and Hadoop libraries at certain known locations for different type of Hive installations like RPM, or Cloudera Parcel. If you have Hive running from a custom install location you can specify environment variables ORCH_HADOOP_HOME (or default HADOOP_HOME) and ORCH_HIVE_HOME (or default HIVE_HOME) and the required libraries will be picked from there.

2 Copyright Notice