Machine Learning with Sparkling Water: H2O + Spark

Michal Malohlava Alex Tellez Jessica Lanford

---

http://h2o.ai/resources

November 2015: First Edition

Machine Learning with Sparkling Water: H2O + Spark
by Michal Malohlava, Alex Tellez & Jessica Lanford

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

November 2015: First Edition

Photos by ©H2O.ai, Inc.

Printed in the United States of America.

Contents

1 What is H2O?

H2O is fast, scalable, open-source machine learning and deep learning for smarter applications. With H2O, enterprises like PayPal, Nielsen Catalina, Cisco, and others can use all their data without sampling to get accurate predictions faster. Advanced algorithms such as deep learning, boosting, and bagging ensembles are built-in to help application designers create smarter applications through elegant APIs. Some of our initial customers have built powerful domain-specific predictive engines for recommendations, customer churn, propensity to buy, dynamic pricing, and fraud detection for the insurance, healthcare, telecommunications, ad tech, retail, and payment systems industries.

Using in-memory compression, H2O handles billions of data rows in-memory, even with a small cluster. To make it easier for non-engineers to create complete analytic workflows, H2O's platform includes interfaces for R, Python, Scala, Java, JSON, and CoffeeScript/JavaScript, as well as a built-in web interface, Flow. H2O is designed to run in standalone mode, on Hadoop, or within a Spark Cluster, and typically deploys within minutes.

H2O includes many common machine learning algorithms, such as generalized linear modeling (linear regression, logistic regression, etc.), Naïve Bayes, principal components analysis, k-means clustering, and others. H2O also implements best-in-class algorithms at scale, such as distributed random forest, gradient boosting, and deep learning. Customers can build thousands of models and compare the results to get the best predictions.

H2O is nurturing a grassroots movement of physicists, mathematicians, and computer scientists to herald the new wave of discovery with data science by collaborating closely with academic researchers and industrial data scientists. Stanford university giants Stephen Boyd, Trevor Hastie, Rob Tibshirani advise the H2O team on building scalable machine learning algorithms. With hundreds of meetups over the past three years, H2O has become a word-of-mouth phenomenon, growing amongst the data community by a hundred-fold, and is now used by 30,000+ users and is deployed using R, Python, Hadoop, and Spark in 2000+ corporations.

Try it out

- Download H2O directly at http://h2o.ai/download.
- Install H2O's R package from CRAN at https://cran.r-project.org/web/packages/h2o/.
- Install the Python package from PyPI at https://pypi.python.org/pypi/h2o/.

Join the community

- To learn about our meetups, training sessions, hackathons, and product updates, visit http://h2o.ai.
- Visit the open source community forum at https://groups.google.com/d/forum/h2ostream.
- Join the chat at https://gitter.im/h2oai/h2o-3.

2 Sparkling Water Introduction

Sparkling Water allows users to combine the fast, scalable machine learning algorithms of H2O with the capabilities of Spark. With Sparkling Water, users can drive computation from Scala, R, or Python and use the H2O Flow UI, providing an ideal machine learning platform for application developers.

Spark is an elegant and powerful general-purpose, open-source, in-memory platform with tremendous momentum. H2O is an in-memory application for machine learning that is reshaping how people apply math and predictive analytics to their business problems.

Integrating these two open-source environments provides a seamless experience for users who want to make a query using Spark SQL, feed the results into H2O Deep Learning to build a model, make predictions, and then use the results again in Spark. For any given problem, better interoperability between tools provides a better experience.

For additional examples, please visit the Sparkling Water GitHub repository at https://github.com/h2oai/sparkling-water/tree/master/examples.

2.1 Typical Use Case

Sparkling Water excels in leveraging existing Spark-based workflows needed to call advanced machine learning algorithms. A typical example involves data munging with help of Spark API, where a prepared table is passed to an H2O algorithm. The constructed model estimates different metrics based on the testing data or gives a prediction that can then be used in the rest of the Spark workflow.

2.2 Features

Sparkling Water provides transparent integration for the H2O engine and its machine learning algorithms into the Spark platform, enabling:

- Use of H2O algorithms in Spark workflow
- Transformation between H2O and Spark data structures
- Use of Spark RDDs and DataFrames as input for H2O algorithms
- Use of H2O Frames as input for MLlib algorithms
- Transparent execution of Sparkling Water applications on top of Spark

2.3 Supported Data Sources

Currently, Sparkling Water can use the following data source types:

- Standard Resilient Distributed Dataset (RDD) API for loading data and transforming it into H2OFrames
- H2O API for loading data directly into H2OFrame from file(s) stored on:
  - local filesystems
  - HDFS
  - S3
  - HTTP/HTTPS

For more details, please refer to the H2O documentation at http://docs.h2o.ai.

2.4 Supported Data Formats

Sparkling Water can read data stored in the following formats:

- CSV
- SVMLight
- ARFF

For more details, please refer to the H2O documentation at http://docs.h2o.ai.

2.5 Supported Spark Execution Environments

Sparkling Water can run on top of Spark in the following ways:

- as a local cluster (where the master node is `local`, `local[*]`, or `local-cluster[...]`)
- as a standalone cluster[1]
- in a YARN environment[2]

3 Design

Sparkling Water is designed to be executed as a regular Spark application. It provides a way to initialize H2O services on each node in the Spark cluster and access data stored in data structures of Spark and H2O.

Since Sparkling Water is primarily designed as Spark application, it is launched inside a Spark executor created after submitting the application. At this point, H2O starts services, including distributed key-value (K/V) store and memory manager, and orchestrates them into a cloud. The topology of the created cloud replicates the topology of the underlying Spark cluster.

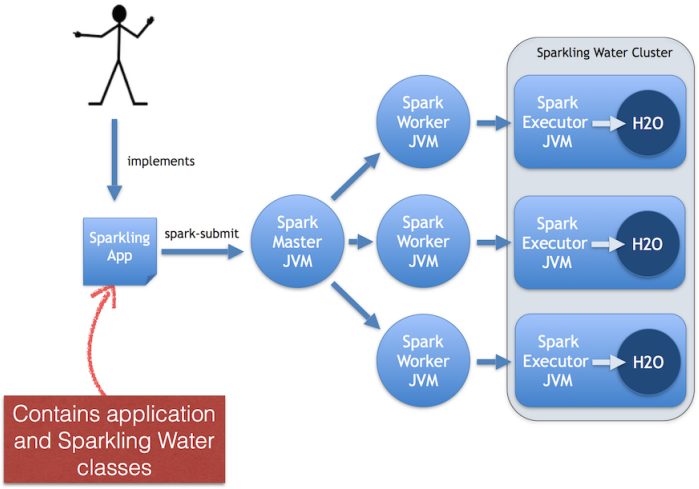

Figure 1: Sparkling Water design depicting deployment of the Sparkling Water application to the standalone Spark cluster.

[1]Refer to the Spark documentation Spark Standalone Model
[2]Refer to the Spark documentation Running Spark on YARN

3.1 Data Sharing between Spark and H2O

Sparkling Water enables transformation between different types of RDDs and H2O's H2OFrame, and vice versa.

When converting from an H2OFrame to an RDD, a wrapper is created around the H2OFrame to provide an RDD-like API. In this case, data is not duplicated but served directly from the underlying H2OFrame.

Converting from an RDD/DataFrame to an H2OFrame requires data duplication because it transfers data from the RDD storage into H2OFrame. However, data stored in an H2OFrame is heavily compressed and does not need to be preserved in RDD.

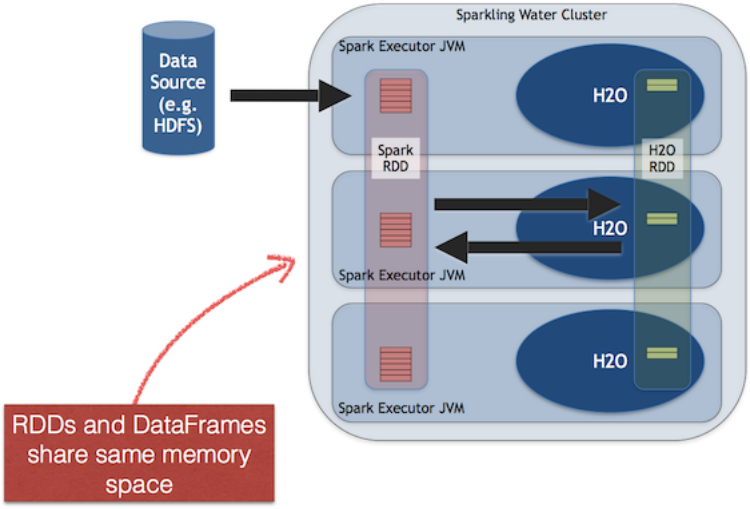

Figure 2: Sharing between Spark and H2O inside an executor JVM.

3.2 Provided Primitives

Sparkling Water provides several primitives (for more information, refer to Table 1). Before using H2O algorithms and data structures, the first step is to create and start the `H2OContext` instance using the `val hc = new H2OContext(sc).start()` call.

The `H2OContext` contains the necessary information for running H2O services and exposes methods for data transformation between the Spark RDD or DataFrame and the H2O Frame. Starting `H2OContext` involves a distributed

operation that contacts all accessible Spark executor nodes and initializes H2O services (such as the key-value store and RPC) inside the executors' JVMs.

When `H2OContext` is running, H2O data structures and algorithms can be manipulated. The key data structure is `H2OFrame`, which represents a distributed table composed of vectors. A new H2O frame can be created using one of the following methods:

- loading a cluster local file (a file located on each node of the cluster):

```
1 val h2oFrame = new H2OFrame(new File("/data/iris.
    csv"))
```

- loading a file from HDFS/S3/S3N/S3A:

```
1 val h2oFrame = new H2OFrame(URI.create("hdfs://
    data/iris.csv"))
```

- transforming Spark RDD or DataFrame:

```
1 val h2oFrame = h2oContext.asH2OFrame(rdd)
```

- referencing existing H2O frame by its key

```
1 val h2oFrame = new H2OFrame("iris.hex")
```

When the `H2OContext` is running, any H2O algorithm can be called. Most of provided algorithms are located in the `hex` package. Calling an algorithm is composed of two steps:

- Specifying parameters:

```
1 val train: H2OFrame = new H2OFrame(new File("
    prostate.csv"))
2 val gbmParams = new GBMParameters()
3 gbmParams._train = train
4 gbmParams._response_column = 'CAPSULE
5 gbmParams._ntrees = 10
```

- Creating the model builder and launching computations. The `trainModel` method is non-blocking and returns a job representing the computation.

```
1 val gbmModel = new GBM(gbmParams).trainModel.get
```

Concept	API Representation	Description
H2O context	`H2OContext`[3]	Contains the H2O state and provides primitives to publish `RDD` as `H2OFrame` and vice versa. Follows design principles of Spark primitives such as `SparkContext` or `SQLContext`
H2O entry point	`H2O`[4]	Represents the entry point for accessing H2O services. Contains information about running H2O services, including a list of nodes and the status of the distributed K/V datastore.
H2O Frame	`H2OFrame`[5]	A data structure representing a table of values. The table is column-based and provides column and row accessors.
H2O Algorithm	package `hex`	Represents the H2O machine learning algorithms library, including DeepLearning, GBM, GLM, DRF, and other algorithms.

Table 1: Sparkling Water primitives

4 Programming API

4.1 Scala

= main methods of H2OContext

- creating algo parameters, calling parameters

5 Deployment

Since Sparkling Water is designed as a regular Spark application, its deployment cycle is strictly driven by Spark deployment strategies (refer to Spark documen-

tation[6]). Spark applications are deployed by the `spark-submit` [7] script that handles all deployment scenarios:

```
./bin/spark-submit \
--class <main-class>
--master <master-url> \
--conf <key>=<value> \
... # other options
<application-jar> \
[application-arguments]
```

- `--class`: Name of main class with `main` method to be executed. For example, the `water.SparklingWaterDriver` application launches H2O services.
- `--master`: Location of Spark cluster
- `--conf`: Specifies any configuration property using the format `key=value`
- `application-jar`: Jar file with all classes and dependencies required for application execution
- `application-arguments`: Arguments passed to the main method of the class via the `--class` option

Sparkling Water supports deployments to the following Spark cluster types:

- Local cluster
- Standalone cluster
- YARN cluster

5.1 Local cluster

The local cluster is identified by the following master URLs - `local`, `local[K]`, or `local[*]`. In this case, the cluster is composed of a single JVM and is created during application submission.

For example, the following command will run the ChicagoCrimeApp application inside a single JVM with a heap size of 5g:

```
$SPARK_HOME/bin/spark-submit \
  --conf spark.executor.memory=5g \
```

[6] Spark deployment guide http://spark.apache.org/docs/latest/cluster-overview.html

[7] Submitting Spark applications http://spark.apache.org/docs/latest/submitting-applications.html

```
3   --conf spark.driver.memory=5g \
4   --master local[*] \
5   --class org.apache.spark.examples.h2o.
        ChicagoCrimeApp \
6   sparkling-water-assembly-1.5.1-all.jar
```

5.2 On Standalone Cluster

For AWS deployments or local private clusters, the standalone cluster deployment[8] is typical. Additionally, a Spark standalone cluster is also provided by Hadoop distributions like CDH or HDP. The cluster is identified by the URL `spark://IP:PORT`.

The following command deploys the ChicagoCrimeApp on a standalone cluster where the master node is exposed on IP mr-0xd10-precise1.0xdata.loc and port 7077:

```
1 $SPARK_HOME/bin/spark-submit \
2   --conf spark.executor.memory=5g \
3   --conf spark.driver.memory=5g \
4   --master spark://mr-0xd10-precise1.0xdata.loc:7077 \
5   --class org.apache.spark.examples.h2o.
        ChicagoCrimeApp \
6   sparkling-water-assembly-1.5.1-all.jar
```

In this case, the standalone Spark cluster must be configured to provide the requested 5g of memory per executor node.

5.3 On YARN Cluster

Because it provides effective resource management and control, most production environments use YARN for cluster deployment.[9] In this case, the environment must contain the shell variable `HADOOP_CONF_DIR` or `YARN_CONF_DIR`.

```
1 $SPARK_HOME/bin/spark-submit \
2   --conf spark.executor.memory=5g \
3   --conf spark.driver.memory=5g \
4   --num-executors 5 \
5   --master yarn-client \
```

[8] Refer to Spark documentation http://spark.apache.org/docs/latest/spark-standalone.html

[9] See Spark documentation http://spark.apache.org/docs/latest/running-on-yarn.html

```
6   --class org.apache.spark.examples.h2o.
        ChicagoCrimeApp \
7 sparkling-water-assembly-1.5.1-all.jar
```

The command in the example above creates a YARN job and requests 5 nodes, each with 5G of memory. The `yarn-client` option forces driver to run in the client process.

6 Building a Standalone Application

7 References

H2O.ai Team. **H2O website**, 2015. URL http://h2o.ai

H2O.ai Team. **H2O documentation**, 2015. URL http://docs.h2o.ai

H2O.ai Team. **H2O GitHub Repository**, 2015. URL https://github.com/h2oai

H2O.ai Team. **H2O Datasets**, 2015. URL http://data.h2o.ai

H2O.ai Team. **H2O JIRA**, 2015. URL https://jira.h2o.ai

H2O.ai Team. **H2Ostream**, 2015. URL https://groups.google.com/d/forum/h2ostream

H2O.ai Team. **H2O R Package Documentation**, 2015. URL http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Rdoc.html