Fast Scalable R with H2O

Patrick Aboyoun Spencer Aiello Anqi Fu Jessica Lanford

http://h2o.gitbooks.io/r-with-h2o/

August 2015: Third Edition

Fast Scalable R with H2O
by Patrick Aboyoun, Spencer Aiello, Anqi Fu & Jessica Lanford

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

August 2015: Third Edition

Photos by ©H2O.ai, Inc.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

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, Cisco, and others can use all their data without sampling to get accurate predictions faster. Advanced algorithms, like 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, AdTech, 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 was built alongside (and on top of) Hadoop and Spark Clusters 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, time series, k-means clustering, and others. H2O also implements best-in-class algorithms at scale, such as 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

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

Join the community

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

2 Introduction

This documentation describes the functionality of R in H2O. Further information on H2O's system and algorithms (as well as R user documentation) can be found at the H2O website at `http://docs.h2o.ai`.

This introductory section describes how H2O works with R, followed by a brief overview of generalized linear models (GLM).

R requires a reference object to the H2O instance because it uses a REST API to send functions to H2O. Data sets are not transmitted directly through the REST API. Instead, the user sends a command (for example, an HDFS path to the data set) either through the browser or via the REST API to ingest data from disk.

The data set is then assigned a Key in H2O that you can use as a reference in future commands to the web server. After preparing your dataset for modeling by defining the significant data and removing the insignificant data, you can create models to represent the results of the data analysis. One of the most popular models for data analysis is GLM.

GLM estimates regression models for outcomes following exponential distributions in general. In addition to the Gaussian (i.e. normal) distribution, these include Poisson, binomial, gamma and Tweedie distributions. Each serves a different purpose, and depending on distribution and link function choice, it can be used either for prediction or classification.

H2O supports Spark, YARN, and all versions of Hadoop. Hadoop is a scalable open-source file system that uses clusters to enable distributed storage and processing of datasets. Depending on your data size, you can get started on your desktop or scale using multiple nodes with Hadoop.

H2O nodes run as JVM invocations on Hadoop nodes. For performance reasons, we recommend you avoid running an H2O node on the same hardware as the Hadoop NameNode if possible.

Since H2O nodes run as mapper tasks in Hadoop, administrators can see them in the normal JobTracker and TaskTracker frameworks. This provides process-level (i.e. JVM instance-level) visibility.

H2O helps R users make the leap from laptop-based processing to large-scale environments. Hadoop helps H2O users scale their data processing capabilities based on their current needs. Using H2O, R, and Hadoop, you can create a complete end-to-end data analysis solution. For more information about H2O on Hadoop, refer to `http://docs.h2o.ai`.

This document will walk you through the four steps of data analysis with H2O: installing H2O, preparing your data for modeling (data munging), creating a model using state-of-the-art machine learning algorithms, and scoring your models.

3 Installation

To use H2O with R, you can start H2O outside of R and connect to it, or you can launch H2O from R. However, if you launch H2O from R and close the R session, the H2O instance is closed as well. The client object is used to direct R to datasets and models located in H2O.

3.1 Installing R or R Studio

To download R:

1. Go to `http://cran.r-project.org/mirrors.html`.
2. Select your closest local mirror.
3. Select your operating system (Linux, OS X, or Windows).
4. Depending on your OS, download the appropriate file, along with any required packages.
5. When the download is complete, unzip the file and install.

To download R Studio:

1. Go to `http://www.rstudio.com/products/rstudio/`.
2. Select your deployment type (desktop or server).
3. Download the file.
4. When the download is complete, unzip the file and install.

3.2 Installing H2O in R

Load the latest CRAN H2O package by running

```
install.packages("h2o")
```

Note: Our push to CRAN will be behind the bleeding edge version and due to resource constraints, may be behind the published version. However, there is a best-effort to keep the versions the same.

```
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
if (! ("methods" %in% rownames(installed.packages()))) { install.packages("methods") }
if (! ("statmod" %in% rownames(installed.packages()))) { install.packages("statmod") }
if (! ("stats" %in% rownames(installed.packages()))) { install.packages("stats") }
if (! ("graphics" %in% rownames(installed.packages()))) { install.packages("graphics") }
if (! ("RCurl" %in% rownames(installed.packages()))) { install.packages("RCurl") }
if (! ("jsonlite" %in% rownames(installed.packages()))) { install.packages("jsonlite") }
if (! ("tools" %in% rownames(installed.packages()))) { install.packages("tools") }
if (! ("utils" %in% rownames(installed.packages()))) { install.packages("utils") }
```

```
# Now we download, install and initialize the H2O package for R (replacing
#the * with the latest version number obtained from the H2O download page)
install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws.com/h2o/master/*/R")))
library(h2o)
localH2O = h2o.init(nthreads = -1)
```

3.3 Making a build from the Source Code

If you are a developer who wants to make changes to the R package before building and installing it, pull the source code from Git (`https://github.com/h2oai/h2o-3`) and follow the instructions in at `https://github.com/h2oai/h2o-3/blob/master/README.md`.

After making the build, navigate to the top-level h2o-3 directory using cd ∼/h2o-3, then run the following (replacing the asterisks [*] with the version number) and install.

```
./gradlew clean
./gradlew build
$ cd target/Rcran
$ R CMD INSTALL h2o-r/R/src/contrib/h2o_****.tar.gz
* installing to library    /Users/H2OUser/.Rlibrary
* installing *source* package    h2o    ...
** R
** demo
** inst
** preparing package for lazy loading
Creating a generic function for ...[output truncated]
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (h2o)
```

4 H2O Initialization

This section describes how to launch H2O:

- from R
- from the command line
- on Hadoop
- on an EC2 cluster

4.1 Launching from R

If you do not specify the argument max_mem_size when you run h2o.init(nthreads = -1), the default heap size of the H2O instance running on 32-bit Java is 1g. H2O checks the Java version and suggests an upgrade if you are running 32-bit Java. On 64-bit Java, the heap size is 1/4 of the total memory available on the machine. The nthreads = -1 parameter allows H2O to use all CPUs on the host, which is recommended.

For best performance, the allocated memory should be 4x the size of your data, but never more than the total amount of memory on your computer. For larger data sets, we recommend running on a server or service with more memory available for computing.

To launch H2O from R, run the following in R:

```
library(h2o)
# Start H2O on localhost, port 54321, with 4g of memory using all CPUs
localH2O <- h2o.init(ip = 'localhost', port = 54321, nthreads= -1, max_
   mem_size = '4g')
```

R displays the following output:

```
Successfully connected to http://localhost:54321
R is connected to H2O cluster:
   H2O cluster uptime:         11 minutes 35 seconds
   H2O cluster version:        2.7.0.1497
   H2O cluster name:           H2O_started_from_R
   H2O cluster total nodes:    1
   H2O cluster total memory:   3.56 GB
   H2O cluster total cores:    8
   H2O cluster allowed cores:  8
   H2O cluster healthy:        TRUE
```

If you are operating on a single node, initialize H2O using all CPUs with:

```
h2o_server = h2o.init(nthreads = -1)
```

To connect with an existing H2O cluster node other than the default localhost:54321, specify the IP address and port number in the parentheses. For example:

```
h2o_cluster = h2o.init(ip = "192.555.1.123", port = 12345, nthreads = -1)
```

4.2 Launching from the Command Line

After launching the H2O instance, initialize the connection by running `h2o.init(nthreads = -1)` with the IP address and port number of a node in the cluster. In the following example, change 192.168.1.161 to your machine's local host.

```
library(h2o)
localH2O <- h2o.init(ip = ’192.168.1.161’, port = 54321, nthreads = -1)
```

4.3 Launching on Hadoop

To launch H2O nodes and form a cluster on the Hadoop cluster, run:

```
hadoop jar h2odriver.jar -nodes 1 -mapperXmx 6g -output hdfsOutputDirName
```

- For each major release of each distribution of Hadoop, there is a driver jar file that the user will need to launch H2O with. Currently available driver jar files in each build of H2O include `h2odriver_cdh5.3.jar`, `h2odriver_hdp2.1.jar`, and `mapr3.1.1.jar`.
- The above command launches exactly one 6g node of H2O; however, we recommend launching the cluster with 4 times the memory of your data file.
- `mapperXmx` is the mapper size or the amount of memory allocated to each node.
- `nodes` is the number of nodes requested to form the cluster.
- `output` is the name of the directory created each time a H2O cloud is created so it is necessary for the name to be unique each time it is launched.

4.4 Launching on an EC2

Note: If you would like to try out H2O on an EC2 cluster, `http://play.h2o.ai` is the easiest way to get started. H2O Play provides access to a temporary cluster managed by H2O.

If you would still like to set up your own EC2 cluster, follow the instructions below to build a cluster of EC2 instances by running the following commands on the host that can access the nodes using a public DNS name.

1. Edit h2o-cluster-launch-instances.pyto include your SSH key name and security group name, as well as any other environment-specific variables:

   ```
   ./h2o-cluster-launch-instances.py
   ./h2o-cluster-distribute-h2o.sh
   ```

   —OR—

   ```
   ./h2o-cluster-launch-instances.py
   ./h2o-cluster-download-h2o.sh
   ```

   Note: The second method may be faster than the first, since download pulls from S3.

2. Distribute the credentials using ./h2o-cluster-distribute-aws-credentials.sh.

   Note: If you are running H2O using an IAM role, it is not necessary to distribute the AWS credentials to all the nodes in the cluster. The latest version of H2O can access the temporary access key.

   Caution: Distributing the AWS credentials copies the Amazon AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY to the instances to enable S3 and S3N access. Use caution when adding your security keys to the cloud.

3. Start H2O by launching one H2O node per EC2 instance: ./h2o-cluster-start-h2o.sh. Wait 60 seconds after entering the command before entering it on the next node.

4. In your internet browser, substitute any of the public DNS node addresses for IP_ADDRESS in the following example: http://IP_ADDRESS:54321

5. To start H2O: ./h2o-cluster-start-h2o.sh

6. To stop H2O: ./h2o-cluster-stop-h2o.sh

7. To shut down the cluster, use your Amazon AWS console to shut down the cluster manually. For more information, refer to the Amazon AWS documentation at http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/UsingEMR_TerminateJobFlow.html.

4.5 Checking Cluster Status

To check the status and health of the H2O cluster, use h2o.clusterInfo().

```
> library(h2o)
> localH2O = h2o.init(ip = 'localhost', port = 54321, nthreads = -1)
> h2o.clusterInfo(localH2O)
```

An easy-to-read summary of information about the cluster displays.

```
R is connected to H2O cluster:
  H2O cluster uptime:         43 minutes 43 seconds
  H2O cluster version:        2.7.0.1497
  H2O cluster name:           H2O_started_from_R
  H2O cluster total nodes:    1
  H2O cluster total memory:   3.56 GB
  H2O cluster total cores:    8
  H2O cluster allowed cores:  8
  H2O cluster healthy:        TRUE
```

5 Data Preparation in R

The following section describes some important points to remember about data preparation (also known as data munging) and some of the tools and methods available in H2O, as well as a data training example.

5.1 Notes

- Although it may seem like you are manipulating the data in R due to the look and feel, once the data has been passed to H2O, all data munging occurs in the H2O instance. The information is passed to R through JSON APIs, so some functions may not have another method.
- You are not limited by R's ability to handle data, but by the total amount of memory allocated to the H2O instance. To process large data sets, make sure to allocate enough memory. For more information, refer to Launching from R.
- You can manipulate datasets with thousands of factor levels using H2O in R, so if you ask H2O to display a table in R with information from high cardinality factors, the results may overwhelm R's capacity.
- To manipulate data in R and not in H2O, use `as.data.frame()`, `as.h2o()`, and `str()`.
  - `as.data.frame()` converts an H2O data frame into an R data frame. Be aware that if your request exceeds R′s capabilities due to the amount of data, the R session will crash. If possible, we recommend only taking subsets of the entire data set (the necessary data columns or rows), and not the whole data set.
  - `as.h2o()` transfers data from R to the H2O instance. We recommend ensuring that you allocate enough memory to the H2O instance for successful data transfer.
  - `str()` returns the elements of the new object to confirm that the data transferred correctly. It′s a good way to verify there were no data loss or conversion issues.

5.2 Tools and Methods

The following section describes some of the tools and methods available in H2O for data preparation.

- **Data Profiling**: Quickly summarize the shape of your dataset to avoid bias or missing information before you start building your model. Missing data, zero values, text, and a visual distribution of the data are visualized automatically upon data ingestion.
- **Summary Statistics**: Visualize your data with summary statistics to get the mean, standard deviation, min, max, or quantile (for numeric columns) or cardinality and counts (for enum columns), and a preview of the data set.
- **Aggregate, Filter, Bin, and Derive Columns**: Build unique views with Group functions, Filtering, Binning, and Derived Columns.
- **Slice, Log Transform, and Anonymize**: Normalize and partition to get your data into the right shape for modeling, and anonymize to remove confidential information.
- **Variable Creation**: Highly customizable variable value creation to hone in on the key data characteristics to model.
- **PCA**: Principal Component Analysis makes feature selection easy with a simple interface and standard input values to reduce the many dimensions in your dataset into key components.
- **Training and Validation Sampling Plan**: Design a random or stratified sampling plan to generate data sets for model training and scoring.

5.3 Demo: Creating Aggregates from Split Data

The following section depicts an example of creating aggregates for data training using ddply(). Using this method, you can split your dataset and apply a function to the subsets.

To apply a user-specified function to each subset of an H2O dataset and combine the results, use ddply(), with the name of the H2O object, the variable name, and the function in the parentheses. For more information about functions, refer to h2o.addFunction in the Appendix.

```
library(h2o)
localH2O = h2o.init(nthreads = -1)

# Import iris dataset to H2O
> irisPath = system.file("extdata", "iris_wheader.csv", package = "h2o")
> iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")

# Add function taking mean of sepal_len column
> fun = function(df) { sum(df[,1], na.rm = T)/nrow(df) }
> h2o.addFunction(localH2O, fun)

# Apply function to groups by class of flower
# uses h2o's ddply, since iris.hex is an H2OParsedData object
> res = ddply(iris.hex, "class", fun)
> head(res)
```

6 Models

The following section describes the features and functions of some common models available in H2O. For more information about running these models in R using H2O, refer to “Running Models.”

H2O supports the following models: Deep Learning, Generalized Linear Models (GLM), Gradient Boosted Regression (GBM), K-Means, Naïve Bayes, Principal Components Analysis (PCA), Principal Components Regression (PCR), Random Forest (RF), and Cox Proportional Hazards (PH).

The list is growing quickly, so check back often at www.h2o.ai to see the latest additions. The following list describes some common model types and features.

Generalized Linear Models (GLM): A flexible generalization of ordinary linear regression for response variables that have error distribution models other than a normal distribution. GLM unifies various other statistical models, including linear, logistic, Poisson, and more.

Decision trees: Used in RF; a decision support tool that uses a tree-like graph or model of decisions and their possible consequences.

Gradient Boosting (GBM): A method to produce a prediction model in the form of an ensemble of weak prediction models. It builds the model in a stage-wise fashion and is generalized by allowing an arbitrary differentiable loss function. It is one of the most powerful methods available today.

K-Means: A method to uncover groups or clusters of data points often used for segmentation. It clusters observations into k certain points with the nearest mean.

Anomaly Detection: Identify the outliers in your data by invoking a powerful pattern recognition model, the Deep Learning Auto-Encoder.

Deep Learning: Model high-level abstractions in data by using non-linear transformations in a layer-by-layer method. Deep learning is an example of supervised learning and can make use of unlabeled data that other algorithms cannot.

Naïve Bayes: A probabilistic classifier that assumes the value of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. It is often used in text categorization.

Grid Search: The standard way of performing hyper-parameter optimization to make model configuration easier. It is measured by cross-validation of an independent data set.

After creating a model, use it to make predictions. For more information about predictions, refer to "Predictions."

6.1 Demo: GLM

The following demo demonstrates how to import a file, define significant data, view data, create testing and training sets using sampling, define the model, and display the results.

```
# Import dataset and display summary
airlinesURL = "https://s3.amazonaws.com/h2o-airlines-unpacked/allyears2k.csv"
airlines.hex = h2o.importFile(localH2O, path = airlinesURL, key = "airlines.hex")%%please update - fails here
summary(airlines.hex)

# Define columns to ignore, quantiles and histograms
high_na_columns = h2o.ignoreColumns(data = airlines.hex)
delay_quantiles = quantile(x = airlines.hex$ArrDelay, na.rm = TRUE)
hist(airlines.hex$ArrDelay)

# Find number of flights by airport
originFlights = h2o.ddply(airlines.hex, 'Origin', nrow)
originFlights.R = as.data.frame(originFlights)

# Find number of cancellations per month
flightsByMonth = h2o.ddply(airlines.hex,"Month", nrow)
flightsByMonth.R = as.data.frame(originFlights)

# Find months with the highest cancellation ratio
fun = function(df) {sum(df$Cancelled)}
h2o.addFunction(h, fun)
cancellationsByMonth = h2o.ddply(airlines.hex,"Month", fun)
cancellation_rate = cancellationsByMonth$C1/flightsByMonth$C1)
rates_table = cbind(flightsByMonth$Month, cancellation_rate)
rates_table.R = as.data.frame(rates.table)

# Construct test and train sets using sampling
airlines.split = h2o.splitFrame(data = airlines.hex,ratios = 0.85)
airlines.train = airlines.split[[1]]
airlines.test = airlines.split[[2]]

# Display a summary using table-like functions
h2o.table(airlines.train$Cancelled)
h2o.table(airlines.test$Cancelled)

# Set predictor and response variables
Y = "IsDepDelayed"
X = c("Origin", "Dest", "DayofMonth", "Year", "UniqueCarrier", "DayOfWeek", "Month", "DepTime", "ArrTime", "Distance")
# Define the data for the model and display the results
```

```
airlines.glm <- h2o.glm(data=airlines.train, x=X, y=Y, family = "binomial", alpha = 0.5)

# Predict using GLM model
pred = h2o.predict(object = airlines.glm, newdata = airlines.test)
```

7 Data Manipulation in R

The following section describes some common R commands. For a complete command list, including parameters, refer to `http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Rdoc.html`. For additional help within R's Help tab, precede the command with a question mark (for example, `?h2o`) for suggested commands containing the search terms. For more information on a command, precede the command with two question marks (`??h2o`).

7.1 Importing Files

The H2O package consolidates all of the various supported import functions using `h2o.importFile()`. Although `h2o.importFolder` and `h2o.importHDFS` will still work, these functions are deprecated and should be updated to `h2o.importFile()`. There are a few ways to import files:

```

# To import small iris data file from H2O's package:
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
|================================================| 100%

# To import an entire folder of files as one data object:
pathToFolder = "/Users/Amy/data/airlines/"
airlines.hex = h2o.importFile(localH2O, path = pathToFolder, key = "airlines.hex")
|================================================| 100%

# To import from HDFS, connect to your Hadoop cluster and start an H2O instance in R using the IP that was specified by Hadoop:
remoteH2O = h2o.init(ip= <IPAddress>, port =54321, nthreads = -1)
pathToData = "hdfs://mr-0xd6.h2oai.loc/datasets/airlines_all.csv"
airlines.hex = h2o.importFile(remoteH2O, path = pathToData, key = "airlines.hex")
|================================================| 100%
```

7.2 Uploading Files

To upload a file in a directory local to your H2O instance, we recommend `h2o.importFile()`. The alternative is to use `h2o.uploadFile()` which can also upload data local to your H2O instance in addition to uploading data local to your R session. In the parentheses, specify the H2O reference object in R and the complete URL or normalized file path for the file.

```
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.uploadFile(localH2O, path = irisPath, key = "iris.hex")
|===================================================| 100%
```

7.3 Finding Factors

To determine if any column in a data set is a factor (contains categorical data), use `h2o.anyFactor()` with the name of the R reference object in the parentheses.

```
irisPath = system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
|===================================================| 100%
h2o.anyFactor(iris.hex)
[1] TRUE
```

7.4 Converting to Factors

To convert an integer into a non-ordered factor (also called an enum or categorical), use `as.factor()` with the name of the R reference object in parentheses followed by the number of the column to convert in brackets.

```
# Import prostate data
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.importFile(localH2O, path = prosPath)
|===================================================| 100%
# Converts column 4 (RACE) to an enum
is.factor(prostate.hex[,4])
[1] FALSE
prostate.hex[,4]<-as.factor(prostate.hex[,4])
is.factor(prostate.hex[,4])
[1] TRUE
# Summary will return a count of the factors
summary(prostate.hex[,4])
 RACE
 1 :341
 2 : 36
 0 :  3
```

7.5 Converting Data Frames

To convert an H2O parsed data object into an R data frame that can be manipulated using R commands, use `as.data.frame()` with the name of the R reference object in the parentheses.

Caution: While this can be very useful, be careful using this command when converting H2O parsed data objects. H2O can easily handle data sets that are often too large to be handled equivalently well in R.

```
# Creates object that defines path
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
# Imports data set
prostate.hex = h2o.importFile(localH2O, path = prosPath)
|===================================================| 100%
# Converts current data frame (prostate data set) to an R data frame
prostate.R <- as.data.frame(prostate.hex)
# Displays a summary of data frame where the summary was executed in R
summary(prostate.data.frame)
       ID             CAPSULE            AGE             RACE
Min.   :  1.00   Min.   :0.0000   Min.   :43.00   Min.   :0.000
```

```
1st Qu.: 95.75   1st Qu.:0.0000   1st Qu.:62.00   1st Qu.:1.000
       ....
```

7.6 Transferring Data Frames

To transfer a data frame from the R environment to the H2O instance, use `as.h2o()`. In the parentheses, specify the name of the h2o.init object that communicates with R and H2O and the object in the R environment to be converted to an H2O object. Optionally, you can include the reference to the H2O instance (the key). Precede the key with `key=` and enclose the key in quotes as in the following example.

```
# Import the iris data into H2O
> data(iris)
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
6            5.4         3.9          1.7         0.4     setosa

# Converts R object "iris" into H2O object "iris.hex"
iris.hex = as.h2o(localH2O, iris, key= "iris.hex")
|==============================================================| 100%
IP Address: localhost
Port      : 54321
Parsed Data Key: iris.hex

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

7.7 Renaming Data Frames

To rename a dataframe on the server running H2O for a data set manipulated in R, use `h2o.assign()`. For instance, in the following example, the prostate data set was uploaded to the H2O instance and the data was manipulated to remove outliers. `h2o.assign()` saves the new data set on the H2O server so that it can be analyzed using H2O without overwriting the original data set.

```
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex<-h2o.importFile(localH2O, path = prosPath)
|====================================================| 100%
## Assign a new name to prostate dataset in the KV store
prostate.hex@key
[1] "prostate.hex"
prostate.hex <- h2o.assign(data = prostate.hex, key = "newName.hex")
prostate.hex@key
[1] "newName.hex"
```

7.8 Getting Column Names

To obtain a list of the column names in the data set, use colnames() or names() with the name of the R reference object in the parentheses.

```
##Displays the titles of the columns
colnames(iris.hex)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
names(iris.hex)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

7.9 Getting Minimum and Maximum Values

To obtain the maximum values for the real-valued columns in a data set, use max() with the name of the R reference object in the parentheses.

To obtain the minimum values for the real-valued columns in a data set, use min() with the name of the R reference object in the parentheses.

```
min(prostate.hex$AGE)
[1] 43
max(prostate.hex$AGE)
[1] 79
```

7.10 Getting Quantiles

To request quantiles for an H2O parsed data set, use quantile() with the name of the R reference object in the parentheses. To request a quantile for a single numerical column, use quantile(ReferenceObject$ColumnName), where ReferenceObject represents the R reference object name and ColumnName represents the name of the specified column. When you request for a full parsed data set consisting of a single column, quantile() displays a matrix with quantile information for the data set.

[redacted]

7.11 Summarizing Data

To generate a summary (similar to the one in R) for each of the columns in the data set, use `summary()` with the name of the R reference object in the parentheses. For continuous real functions, this produces a summary that includes information on quartiles, min, max, and mean. For factors, this produces information about counts of elements within each factor level.

```
summary(australia.hex)
 premax          salmax          minairtemp      maxairtemp      maxsst
 Min.   : 18.0   Min.   :3441   Min.   :272.6   Min.   :285.0   Min.   :285697
 1st Qu.: 75.0   1st Qu.:3490   1st Qu.:277.0   1st Qu.:292.0   1st Qu.:290491
 Median :150.0   Median :3533   Median :278.8   Median :299.9   Median :293643
 Mean   :161.5   Mean   :3529   Mean   :279.9   Mean   :297.5   Mean   :295676
 3rd Qu.:250.0   3rd Qu.:3558   3rd Qu.:282.0   3rd Qu.:302.4   3rd Qu.:301942
 Max.   :450.0   Max.   :3650   Max.   :290.0   Max.   :310.0   Max.   :303697
 maxsoilmoist     Max_czcs         runoffnew
 Min.   : 0.000   Min.   : 0.160   Min.   :   0.0
 1st Qu.: 0.000   1st Qu.: 0.629   1st Qu.:   0.0
 Median : 4.000   Median : 1.020   Median :  19.0
 Mean   : 5.117   Mean   : 1.369   Mean   : 232.2
 3rd Qu.: 9.000   3rd Qu.: 1.705   3rd Qu.: 300.0
 Max.   :16.000   Max.   :11.370   Max.   :2400.0
```

7.12 Summarizing Data in a Table

To summarize the data, use `h2o.table()`. Because H2O can handle larger data sets, it is possible to generate tables that are larger than R's capacity.

To summarize multiple columns, use `head(h2o.table (ObjectName[, c(ColumnNumber,ColumnNumber)]))` where `ObjectName` is the name of the object in R and `ColumnNumber` is the number of the column.

[redacted]

7.13 Generating Random Uniformly Distributed Numbers

To append a column of random numbers to an H2O data frame for facilitating creation of testing/training data splits for analysis and validation in H2O, use `h2o.runif()` with the name of the R reference object in the parentheses. This method is best for customized frame splitting; otherwise, use `h2o.splitFrame()`. However, `h2o.runif()` is not as fast or stable as `h2o.splitFrame()`.

[redacted]

7.14 Splitting Frames

To generate two subsets (according to specified ratios) from an existing H2O data set for testing/training, use `h2o.splitFrame()`. This method is preferred over `h2o.runif` because it is faster and more stable.

```
# Splits data in prostate data frame with a ratio of 0.75
prostate.split <- h2o.splitFrame(data = prostate.hex , ratios = 0.75)
# Creates training set from 1st data set in split
prostate.train <- prostate.split[[1]]
# Creates testing set from 2st data set in split
prostate.test <- prostate.split[[2]]
```

7.15 Getting Frames

To create a reference object to the data frame in H2O, use `h2o.getFrame()`. This is helpful for users that alternate between the web UI and the R API or multiple users accessing the same H2O instance. The following example assumes prostate.hex is in the key-value (KV) store.

```
prostate.hex <- h2o.getFrame(h2o = localH2O, key = "prostate.hex")
```

7.16 Getting Models

To create a reference object for the model in H2O, use `h2o.getModel()`. This is helpful for users that alternate between the web UI and the R API or multiple users accessing the same H2O instance. The following example assumes gbm.model is in the key-value (KV) store.

```
gbm.model <- h2o.getModel(h2o = localH2O, key = "GBM_8e4591a9b413407b983d73fbd9eb44cf"
```

7.17 Listing H2O Objects

To generate a list of all H2O objects generated during a session, along with each objects size in bytes, use `h2o.ls()` with the address of the instance in the parentheses. If the instance is local, use localH2O.

```
h2o.ls(localH2O)
                                                 Key       Bytesize
      1               GBM_8e4591a9b413407b983d73fbd9eb44cf    40617
      2               GBM_a3ae2edf5dfadbd9ba5dc2e9560c405d     1516
```

7.18 Removing H2O Objects

To remove an H2O object on the server associated with an object in the R environment, use `h2o.rm()`. For optimal performance, we recommend removing the object from the R environment as well using `remove()`, with the name of the object in the parentheses. If you do not specify an R environment, then the current environment is used.

```
h2o.rm(object = localH2O, keys = "prostate.train")
```

7.19 Adding Functions

To add a user-defined function in R to the H2O instance, use `h2o.addFunction()`, with the IP address of the H2O instance and the function in the parentheses.

```
# Send the functional expression to H2O
simpleFun <- h2o.addFunction(localH2O, function(x) { 2*x + 5 }, "simpleFun")
# Evaluate the expression across prostate’s AGE column
calculated = h2o.exec(expr_to_execute = simpleFun(prostate.hex[,"AGE"]), h2o
= localH2O)
cbind(prostate.hex[,"AGE"], calculated)
IP Address: localhost
Port      : 54321
Parsed Data Key: Last.value.152

  AGE fun
1  65 135
2  72 149
```

```
3  70 145
4  76 157
5  69 143
6  71 147
```

8 Running Models

To run the models, use the following commands.

8.1 Gradient Boosted Models (GBM)

To generate gradient boosted models for developing forward-learning ensembles, use `h2o.gbm()`. In the parentheses, define x (the predictor variable vector), y (the integer or categorical response variable), the distribution type (multinomial is the default, gaussian is used for regression), and the name of the H2OParsedData object.

For more information, use `help(h2o.gbm)`.

```
library(h2o)
localH2O <- h2o.init(nthreads = -1)
iris.hex <- as.h2o(localH2O, object = iris, headers = T, key = "iris.hex")
iris.gbm <- h2o.gbm(y = 1, x = 2:5, data = iris.hex, n.trees = 10,
interaction.depth = 3, n.minobsinnode = 2, shrinkage = 0.2, distribution= "gaussian")
|=================================================| 100%
# To obtain the Mean-squared Error by tree from the model object:
iris.gbm@model[,"err"]
 [1] 0.68112220 0.47215388 0.33393673 0.24465574 0.18596269 0.14500129
 [7] 0.11792983 0.10003321 0.08793070 0.07862922 0.07232574
```

To generate a classification model that uses labels, use `distribution= "multinomial"`:

[redacted]

8.2 Generalized Linear Models (GLM)

Generalized linear models (GLM) are some of the most commonly-used models for many types of data analysis use cases. While some data analysis can be done using general linear models, if the variables are more complex, general linear models may not be as accurate. For example, if the dependent variable has a non-continuous distribution or if the effect of the predictors is not linear, generalized linear models will produce more accurate results than general linear models.

Generalized Linear Models (GLM) estimate regression models for outcomes following exponential distributions in general. In addition to the Gaussian (i.e. normal) distribution, these include Poisson, binomial, gamma and Tweedie distributions. Each serves a different purpose, and depending on distribution and link function choice, it can be used either for prediction or classification.

H2O's GLM algorithm fits the generalized linear model with elastic net penalties. The model fitting computation is distributed, extremely fast,and scales extremely well for models with a limited number (˜ low thousands) of predictors with non-zero coefficients. The algorithm can compute models for a single value of a penalty argument or the full regularization path, similar to glmnet. It can compute gaussian (linear), logistic, poisson, and gamma regression models.

To generate a generalized linear model for developing linear models for exponential distributions, use `h2o.glm()`. You can apply regularization to the model by adjusting the lambda and alpha parameters. For more information, use `help(h2o.glm)`.

[redacted]

[redacted]

8.3 K-Means

To generate a K-Means model for data characterization, use `h2o.kmeans()`. This algorithm does not rely on a dependent variable. For more information, use `help(h2o.kmeans)`.

```
iris.km = h2o.kmeans(data = iris.hex, centers = 3, cols = 1:4)
|====================================================| 100%
print(iris.km)
IP Address: localhost
Port      : 54321
Parsed Data Key: iris.hex
K-Means Model Key: KMeans2_937845cadf924db4612c3a7d8f9744a0
K-means clustering with 3 clusters of sizes 50, 38, 62
Cluster means:
        C1       C2       C3       C4
1 5.006000 3.418000 1.464000 0.244000
2 6.850000 3.073684 5.742105 2.071053
3 5.901613 2.748387 4.393548 1.433871
  ....
```

8.4 Principal Components Analysis (PCA)

To map a set of variables onto a subspace using linear transformations, use `h2o.prcomp()`. This is the first step in Principal Components Regression. For more information, use `help(h2o.prcomp)`.

```
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
|====================================================| 100%
australia.pca <- h2o.prcomp(data = australia.hex, standardize = TRUE)
|====================================================| 100%
      ....
PCA Model Key: PCA_8fbc38e360de5b3c1ae6b7cc754b499c
Standard deviations:
 1.750703 1.512142 1.031181 0.8283127 0.6083786 0.5481364 0.4181621 0.2314953
      ....
summary(australia.pca)
Importance of components: PC1       PC2       PC3        PC4        PC5
    PC6
Standard deviation     1.7507032 1.5121421 1.0311814 0.82831266 0.60837860
0.54813639
Proportion of Variance 0.3831202 0.2858217 0.1329169 0.08576273 0.04626556
0.03755669
Cumulative Proportion  0.3831202 0.6689419 0.8018588 0.88762155 0.93388711
0.97144380
```

8.5 Principal Components Regression (PCR)

To map a set of variables to a set of linearly independent variables, use `h2o.pcr()`. The variables in the new set are linearly independent linear combinations of the original variables and exist in a lower-dimension subspace. This transformation is prepended to the regression model to improve results. For more information, use `help(h2o.pcr)`.

[redacted]

8.6 Predictions

The following section describes some of the prediction methods available in H2O.

Predict: Generate outcomes of a data set with any model. Predict with GLM, GBM, Decision Trees or Deep Learning models.

Confusion Matrix: Visualize the performance of an algorithm in a table to understand how a model performs.

Area Under Curve (AUC): A graphical plot to visualize the performance of a model by its sensitivity, true positives and false positives to select the best model.

Hit Ratio: A classification matrix to visualize the ratio of the number of correctly classified and incorrectly classified cases.

PCA Score: Determine how well your feature selection fits a particular model.

Multi-Model Scoring: Compare and contrast multiple models on a data set to find the best performer to deploy into production.

To apply an H2O model to a holdout set for predictions based on model results, use `h2o.predict()`. In the following example, H2O generates a model and then displays the predictions for that model.

```
prostate.fit = h2o.predict(object = prostate.glm, newdata = prostate.hex)
(prostate.fit)
```

```
  predict        X0        X1
1       0 0.7452267 0.2547732
2       1 0.3969807 0.6030193
3       1 0.4120950 0.5879050
4       1 0.3726134 0.6273866
5       1 0.6465137 0.3534863
6       1 0.4331880 0.5668120
```

9 Support

There are multiple ways to request support for H2O:

Email: h2ostream@googlegroups.com

H2OStream on Google Groups: https://groups.google.com/d/forum/h2ostream

JIRA: http://jira.0xdata.com/

Meetup information: http://h2o.ai/events

10 References

R Package: http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Rdoc.html

R Ensemble documentation: http://www.stat.berkeley.edu/~ledell/R/h2oEnsemble.pdf

Slide deck: http://h2o.ai/blog/2013/08/big-data-science-in-h2o-with-r/

R project website: http://www.r-project.org

11 Appendix: Commands

The following section lists some common commands by function that are available in R and a brief description of each command.

11.1 Data Set Operations

Data Import/Export

`h2o.downloadCSV`: Download a H2O dataset to a CSV file on local disk.
`h2o.exportFile`: Export H2O Data Frame to a File.
`h2o.importFile`: Import a file from the local path and parse it.
`h2o.parseRaw`: Parse a raw data file.
`h2o.uploadFile`: Upload a file from the local drive and parse it.

Native R to H2O Coercion

`as.h2o`: Convert an R object to an H2O object.

H2O to Native R Coercion

`as.data.frame`: Check if an object is a data frame, or coerce it if possible.
`as.matrix`: Convert the specified argument to a matrix.
`as.table`: Build a contingency table of the counts at each combination of factor levels.

Data Generation

`h2o.createFrame`: Create an H2O data frame, with optional randomization.
`h2o.runif`: Produce a vector of random uniform numbers.
`h2o.interaction`: Create interaction terms between categorical features of an H2O Frame.

Data Sampling / Splitting

h2o.sample: Sample an existing H2O Frame by number of observations.
h2o.splitFrame: Split an existing H2O data set according to user-specified ratios.
h2o.nFoldExtractor: Split an existing H2O data set into N folds and return a specified holdout split, and the rest.

Missing Data Handling

h2o.impute: Impute a column of data using the mean, median, or mode.
h2o.insertMissingValues: Replaces a user-specified fraction of entries in a H2O dataset with missing values.
h2o.ignoreColumns: Returns columns' names of a parsed H2O data object that are recommended to be ignored in an analysis per the specified ratio in max_na.

11.2 General Data Operations

Subscripting example to pull pieces from data object.

```
x[i]
x[i, j, ... , drop = TRUE]
x[[i]]
x$name

x[i] <- value
x[i, j, ...] <- value
x[[i]] <- value
x$i <- value
```

Subsetting
head, tail: Return the First or Last Part of an Object

Concatenation

c: Combine Values into a Vector or List
h2o.cbind: Take a sequence of H2O datasets and combine them by column.

Data Attributes

colnames: Return column names for a parsed H2O data object.
colnames<-: Retrieve or set the row or column names of a matrix-like object.
names: Get the name of an object.
names<-: Set the name of an object.
dim: Retrieve the dimension of an object.
length: Get the length of vectors (including lists) and factors.
nrow: Return a count of the number of rows in an H2OParsedData object.
ncol: Return a count of the number of columns in an H2OParsedData object.
h2o.anyFactor: Check if an H2O parsed data object has any categorical data columns.
is.factor: Check if a given column contains categorical data.

Data Type Coercion

as.factor: Convert a column from numeric to factor.
as.Date: Converts a column from factor to date.

11.3 Methods from Group Generics

Math (H2O)

`abs`: Compute the absolute value of x.
`sign`: Return a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).
`sqrt`: Computes the principal square root of x, $\sqrt{x}$.
`ceiling`: Take a single numeric argument x and return a numeric vector containing the smallest integers not less than the corresponding elements of x.
`floor`: Take a single numeric argument x and return a numeric vector containing the largest integers not greater than the corresponding elements of x.
`trunc`: Take a single numeric argument x and return a numeric vector containing the integers formed by truncating the values in x toward 0.
`log`: Compute logarithms (by default, natural logarithms).
`exp`: Compute the exponential function.

Math (generic)

`cummax`: Display a vector of the cumulative maxima of the elements of the argument.
`cummin`: Display a vector of the cumulative minima of the elements of the argument.
`cumprod`: Display a vector of the cumulative products of the elements of the argument.
`cumsum`: Display a vector of the cumulative sums of the elements of the argument.
`log10`: Compute common (i.e., base 10) logarithms
`log2`: Compute binary (i.e., base 2) logarithms.
`log1p`: Compute log(1+x) accurately also for $|x| << 1$.
`acos`: Compute the trigonometric arc-cosine.
`acosh`: Compute the hyperbolic arc-cosine.
`asin`: Compute the trigonometric arc-sine.
`asinh`: Compute the hyperbolic arc-sine.
`atan`: Compute the trigonometric arc-tangent.
`atanh`: Compute the hyperbolic arc-tangent.
`expm1`: Compute exp(x) - 1 accurately also for $|x| << 1$.
`cos`: Compute the trigonometric cosine.
`cosh`: Compute the hyperbolic cosine.
`cospi`: Compute the trigonometric two-argument arc-cosine.
`sin`: Compute the trigonometric sine.
`sinh`: Compute the hyperbolic sine.
`sinpi`: Compute the trigonometric two-argument arc-sine.
`tan`: Compute the trigonometric tangent.
`tanh`: Compute the hyperbolic tangent.
`tanpi`: Compute the trigonometric two-argument arc-tangent.
`gamma`: Display the gamma function γx
`lgamma`: Display the natural logarithm of the absolute value of the gamma function.
`digamma`: Display the first derivative of the logarithm of the gamma function.
`trigamma`: Display the second derivative of the logarithm of the gamma function.

Math2 (H2O)

`round`: Round the values to the specified number of decimal places (default 0).
`signif`: Round the values to the specified number of significant digits.

Summary (H2O)

`max`: Display the maximum of all the input arguments.
`min`: Display the minimum of all the input arguments.

`range`: Display a vector containing the minimum and maximum of all the given arguments.
`sum`: Calculate the sum of all the values present in its arguments.

Summary (generic)

`prod`: Display the product of all values present in its arguments.
`any`: Given a set of logical vectors, determine if at least one of the values is true.
`all`: Given a set of logical vectors, determine if all of the values are true.

11.4 Other Aggregations

Non-Group Generic Summaries

`mean`: Generic function for the (trimmed) arithmetic mean.
`sd`: Calculate the standard deviation of a column of continuous real valued data.
`var`: Compute the variance of x.
`summary`: Produce result summaries of the results of various model fitting functions.
`quantile`: Obtain and display quantiles for H2O parsed data.

Row / Column Aggregation

`apply`: Apply a function over an H2O parsed data object (an array).

Group By Aggregation

`h2o.ddply`: Split H2O dataset, apply a function, and display results.
`h2o.addFunction`: Add a function defined in R to the H2O server for future use.

Tabulation

`h2o.table`: Use the cross-classifying factors to build a table of counts at each combination of factor levels.

11.5 Data Munging

General Column Manipulations

`is.na`: Display missing elements.
`unique`: Display a vector, data frame, or array with duplicate elements/rows removed.

Element Index Selection

`findInterval`: Find Interval Numbers or Indices.
`h2o.which`: Display the row numbers for which the condition is true.

Conditional Element Value Selection

`h2o.ifelse`: Apply conditional statements to numeric vectors in H2O parsed data objects.

Numeric Column Manipulations

`h2o.cut`: Convert H2O Numeric Data to Factor.
`diff`: Display suitably lagged and iterated differences.

Character Column Manipulations

`h2o.strsplit`: Splits the given factor column on the input split.
`h2o.tolower`: Change the elements of a character vector to lower case.
`h2o.toupper`: Change the elements of a character vector to lower case.
`h2o.trim`: Remove leading and trailing white space.
`h2o.gsub`: Match a pattern & replace all instances of the matched pattern with the replacement string globally.
`h2o.sub`: Match a pattern & replace the first instance of the matched pattern with the replacement string.

Factor Level Manipulations

`h2o.levels`: Display a list of the unique values found in a column of categorical data.
`revalue`: Replace specified values with new values in a factor or character vector.

Date Manipulations

`h2o.month`: Convert the entries of a H2OParsedData object from milliseconds to months (on a 0 to 11 scale).
`h2o.year`: Convert the entries of a H2OParsedData object from milliseconds to years, indexed starting from 1900.

Matrix Operations

`%*%`: Multiply two matrices, if they are conformable.
`t`: Given a matrix or data.frame x, t returns the transpose of x.

11.6 Data Modeling

Model Training

`h2o.coxph`: Fit a Cox Proportional Hazards Model.
`h2o.gbm`: Build gradient boosted classification trees and gradient boosted regression trees on a parsed data set.
`h2o.glm`: Fit a generalized linear model, specified by a response variable, a set of predictors, and a description of the error distribution.
`h2o.kmeans`: Perform k-means clustering on a data set.
`h2o.naiveBayes`: Build gradient boosted classification trees and gradient boosted regression trees on a parsed data set.
`h2o.pcr`: Run GLM regression on PCA results, and allow for transformation of test data to match PCA transformations of training data.
`h2o.prcomp`: Perform principal components analysis on the given data set.
`h2o.randomForest`: Perform random forest classification on a data set.

Deep Learning

`h2o.deeplearning`: Perform Deep Learning neural networks on an H2OParsedData object.
`h2o.anomaly`: Detect anomalies in a H2O dataset using a H2O deep learning model with auto-encoding.
`h2o.deepfeatures`: Extract the non-linear features from a H2O dataset using a H2O deep learning model.

Model Scoring

`predict.H2OModel`: Obtain predictions from various fitted H2O model objects.

Classification Model Helpers

`h2o.confusionMatrix`: Display prediction errors for classification data from a column of predicted responses and a column of actual (reference) responses in H2O.
`h2o.gains`: Construct the gains table and lift charts for binary outcome algorithms.
`h2o.hit_ratio_table`: Retrieve the Hit Ratios. If `train`, `valid`, and `xval` parameters are FALSE (default), then the training Hit Ratios value is returned. If more than one parameter is set to TRUE, then a named list of Hit Ratio tables are returned, where the names are `train`, `valid`, or `xval`.
`h2o.performance`: Evaluate the predictive performance of a model via various measures.

Clustering Helper

`h2o.gapStatistic`: Measure the suitability of the fit of a clustering algorithm.

Regression Model Helper

`h2o.mse`: Display the mean squared error calculated from a column of predicted responses and a column of actual (reference) responses in H2O.

GLM Helper

`h2o.getGLMLambdaModel`: Retrieve the H2O GLM model built using a specific value of lambda from a lambda search.

11.7 H2O Cluster Operations

H2O Key Value Store Access

`h2o.assign`: Assign H2O hex.keys to objects in their R environment.
`h2o.getFrame`: Get a reference to an existing H2O data set.
`h2o.getModel`: Get a reference to an existing H2O model.
`h2o.ls:` Display a list of object keys in the running instance of H2O.
`h2o.rm`: Remove H2O objects from the server where the instance of H2O is running, but does not remove it from the R environment.

H2O Object Serialization

`h2o.loadAll`: Load all H2OModel object in a directory from disk that was saved using `h2o.saveModel` or `h2o.saveAll`.
`h2o.loadModel`: Load an H2OModel object from disk.
`h2o.saveAll`: Save all H2OModel objects to disk to be loaded back into H2O using `h2o.loadModel` or `h2o.loadAll`.
`h2o.saveModel`: Save an H2OModel object to disk to be loaded back into H2O using `h2o.loadModel`.

H2O Cluster Connection

`h2o.init (nthreads = -1)`: Connect to a running H2O instance using all CPUs on the host and check the local H2O R package is the correct version.
`h2o.shutdown`: Shut down the specified H2O instance. All data on the server will be lost!

H2O Load Balancing

`h2o.rebalance`: Rebalance (repartition) an existing H2O data set into given number of chunks (per Vec), for load-balancing across multiple threads or nodes.

H2O Cluster Information

`h2o.clusterInfo`: Display the name, version, uptime, total nodes, total memory, total cores and health of a cluster running H2O.
`h2o.clusterStatus`: Retrieve information on the status of the cluster running H2O.

H2O Logging

`h2o.clearLog`: Clear all H2O R command and error response logs from the local disk.
`h2o.downloadAllLogs`: Download all H2O log files to the local disk.
`h2o.logAndEcho`: Write a message to the H2O Java log file and echo it back.
`h2o.openLog`: Open existing logs of H2O R POST commands and error responses on the local disk.
`h2o.getLogPath`: Get the file path for the H2O R command and error response logs.
`h2o.setLogPath`: Set the file path for the H2O R command and error response logs.
`h2o.startLogging`: Begin logging H2O R POST commands and error responses.
`h2o.stopLogging`: Stop logging H2O R POST commands and error responses.