Fast Scalable R with H2O

Patrick Aboyoun   Spencer Aiello   Eric Eckstrand
Anqi Fu   Mark Landry   Jessica Lanford

http://h2o.ai/resources/

August 2015: Third Edition

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

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

August 2015: Third Edition

Contents

1 Introduction

This documentation describes how to use H2O in the R environment. More information on H2O's system and algorithms (as well as R user documentation) is available at the H2O website at `http://docs.h2o.ai`.

R uses a REST API to connect to H2O, so you can specify the IP address and port number of the H2O instance in the R environment to use H2O in R or launch H2O from R. Data sets are not directly transmitted through the REST API. Instead, sending a command (for example, an HDFS path to import a data set) either through the browser or the REST API performs the specified task.

The data set is then assigned an identifier, which uses the .hex file type in H2O, that is used as a reference in commands to the web server. After preparing the dataset for modeling by defining significant data and removing insignificant data, H2O creates a model representing the results of the data analysis. These models are assigned IDs that are used as references in commands. 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 binomial, gamma, Poisson, and Tweedie distributions. Each serves a different purpose, and depending on distribution and link function, can be used for prediction or classification.

This booklet demonstrates the use of H2O's implementation of GLM in an R environment. For more information on GLM, there is an additional booklet (similar to this one) available at `http://h2o.ai/resources/`.

H2O supports Spark, YARN, and all versions of Hadoop. Hadoop is a scalable open-source file system that uses clusters for distributed storage and dataset processing. Depending on the size of your data, H2O can run on your desktop or scale using multiple nodes with Hadoop, an EC2 cluster, or S3 storage.

H2O nodes run as JVM invocations on Hadoop nodes. For performance reasons, we recommend that you do not run an H2O node on the same hardware as the Hadoop NameNode.

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

H2O helps R users make the leap from laptop-based processing to large-scale environments. Hadoop lets 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.

This document describes the four steps of data analysis with H2O:

1. installing H2O
2. preparing your data for modeling (data munging)
3. creating a model using simple but powerful machine learning algorithms
4. scoring your models

2 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 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 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`.

3 Installation

To use H2O with R, start H2O outside of R and connect to it, or launch H2O from R. However, if you launch H2O from R and close the R session, the H2O session closes as well. The H2O session directs R to the datasets and models located in H2O.

This following sections describe:

- installing R
- installing H2O from R
- making a build from source code

3.1 Installing R

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.

3.2 Installing H2O from R

To load a recent H2O package from CRAN, run:

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

Note: The version of H2O in CRAN is often one release behind the current version.

For the latest recommended version, download the latest stable H2O-3 build from the H2O download page:

1. Go to `http://h2o.ai/download`.
2. Choose the latest stable H2O-3 build.
3. Click the "Install in R" tab.
4. Copy and paste the commands into your R session.

After H2O is installed on your system, verify the installation completed successfully by initializing H2O:

```
library(h2o)

#Start H2O on your local machine using all available cores
#(By default, CRAN policies limit use to only 2 cores)
h2o.init(nthreads = -1)
```

```

#Get help
?h2o.glm
?h2o.gbm

#Show a demo
demo(h2o.glm)
demo(h2o.gbm)
```

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
$ R CMD INSTALL h2o-r/R/src/contrib/h2o_****.tar.gz
* installing to library    /Users/H2OUser/.Rlibrary
* installing *source* package    h 2 o    ...
** 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

To specify the number of CPUs for the H2O session, use the `nthreads =` parameter in the `h2o.init` command. `-2` uses the CRAN default of 2 CPUs. `-1` uses all CPUs on the host, which is strongly recommended. To use a specific number of CPUs, enter a positive integer.

To specify the maximum amount of memory for the H2O session, use the `max_mem_size` parameter in the `h2o.init` command. The value must a multiple of 1024 greater than 2MB. Append the letter `m` or `M` to indicate megabytes, or `g` or `G` to indicate gigabytes.

If you do not specify a value for `max_mem_size` when you run `h2o.init`, the default heap size of the H2O instance running on 32-bit Java is 1g.

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.

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. -

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

```
library(h2o)
# Starts H2O using localhost IP, port 54321, all CPUs, and
    4g of memory
h2o.init(ip = 'localhost', port = 54321, nthreads= -1, max
   _mem_size = '4g')
```

After successfully launching, R displays output similar to the following example:

```
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
```

To launch H2O locally with default initialization arguments, use the following:

```
h2o.init()
```

To connect to an established H2O cluster (in a multi-node Hadoop environment, for example) specify the IP address and port number for the established cluster using the `ip` and `port` parameters in the `h2o.init()` command.

```
h2o.init(ip = "123.45.67.89", port = 54321)
```

4.2 Launching from the Command Line

A simple way to launch H2O from the command line is to download the H2O zip file from the H2O download page. Unzip and launch H2O with the following:

```
unzip h2o-3.2.0.1-*.zip
cd h2o-3.2.0.1-*
java -jar h2o.jar
```

See the H2O Documentation for additional JVM and H2O command line options. After launching the H2O instance, connect to it from R with `h2o.init()` as described above.

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
```

- You must launch the Hadoop-specific H2O driver jar (`h2odriver.jar`) for your Hadoop distribution. Specific driver jar files are available for the following Hadoop versions:

  - `cdh5.2`
  - `cdh5.3`
  - `cdh5.4.2`
  - `hdp2.1`
  - `hdp2.2`
  - `mapr3.1.1`
  - `mapr4.0.1`
  - `mapr5.0`

- 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.

4.5 Checking Cluster Status

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

```
> library(h2o)
> h2o.init()
> h2o.clusterInfo()
```

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 contains information 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, 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 limited by the total amount of memory allocated to the H2O instance, not by R's ability to handle data. 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. If your request exceeds the amount of data supported by R, the R session

will crash. If possible, we recommend only taking subsets of the entire data set (the necessary data columns or rows) instead of the whole data set.

- `as.h2o()` transfers data from R to the H2O instance. For successful data transfer, we recommend confirming enough memory is allocated to the H2O instance.
- `str.H2OFrame()` 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 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.

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

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

# Apply function to groups by class of flower
# uses h2o's ddply, since iris.hex is an H2OFrame object
> res = h2o.ddply(iris.hex, "class", function(df) { sum(df
    [,1], na.rm = T)/nrow(df) })
> 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
- Naïve Bayes
- Principal Components Analysis (PCA)
- K-means
- Generalized Linear Models (GLM)
- Gradient Boosted Regression (GBM)
- Distributed Random Forest (DRF)

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.

6.1 Supervised Learning

Generalized Linear Models (GLM): Provides flexible generalization of ordinary linear regression for response variables with error distribution models other than a Gaussian (normal) distribution. GLM unifies various other statistical models, including Poisson, linear, logistic, and others when using ℓ_1 and ℓ_2 regularization.

Distributed Random Forest: Averages multiple decision trees, each created on different random samples of rows and columns. It is easy to use, non-linear, and provides feedback on the importance of each predictor in the model, making it one of the most robust algorithms for noisy data.

Gradient Boosting (GBM): Produces 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.

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

Naïve Bayes: Generates 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.

6.2 Unsupervised Learning

K-Means: Reveals groups or clusters of data points for segmentation. It clusters observations into k-number of points with the nearest mean.

Anomaly Detection: Identifies the outliers in your data by invoking the deep learning autoencoder, a powerful pattern recognition model.

6.3 Modeling Constructs

Grid Search: Performs standard hyper-parameter optimization to simplify model configuration.

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

7 Demo: GLM

The following demo demonstrates how to:

1. import a file
2. define significant data
3. view data
4. create testing and training sets using sampling
5. define the model
6. display the results

```
# Import dataset and display summary
> library(h2o)
> h2o.init()
> airlinesURL = "https://s3.amazonaws.com/h2o-airlines-
    unpacked/allyears2k.csv"
> airlines.hex = h2o.importFile(path = airlinesURL,
    destination_frame = "airlines.hex")
> summary(airlines.hex)

# View quantiles and histograms
#high_na_columns = h2o.ignoreColumns(data = airlines.hex)
> quantile(x = airlines.hex$ArrDelay, na.rm = TRUE)
> h2o.hist(airlines.hex$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[,which(colnames(airlines.hex)
    =="Cancelled")])}
> 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(training_frame=airlines.train, x
    =X, y=Y, family = "binomial", alpha = 0.5)
# View model information: training statistics, performance
    , important variables
> summary(airlines.glm)

# Predict using GLM model
> pred = h2o.predict(object = airlines.glm, newdata =
    airlines.test)
# Look at summary of predictions: probability of TRUE
    class (p1)
> summary(pred$p1)
```

8 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`).

8.1 Importing Files

The H2O package consolidates all of the various supported import functions using `h2o.importFile()`. There are a few ways to import files shown in the following examples:

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

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

#To import from HDFS and connect to H2O in R using the IP
   and port of an H2O instance running on your  Hadoop
   cluster:
h2o.init(ip= <IPAddress>, port =54321, nthreads = -1)
pathToData = "hdfs://mr-0xd6.h2oai.loc/datasets/airlines_
   all.csv"
airlines.hex = h2o.importFile(path = pathToData,
   destination_frame = "airlines.hex")
 |================================================| 100%
```

8.2 Uploading Files

To upload a file in a directory local to your H2O instance, use `h2o.importFile()`. `h2o.uploadFile()` uploads data local to your H2O instance as well as 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(path = irisPath, destination_
    frame = "iris.hex")
|====================================================|
    100%
```

8.3 Finding Factors

To determine if any column in a data set contains categorical data (also known as a factor), 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(path = irisPath)
|====================================================| 100%
> h2o.anyFactor(iris.hex)
[1] TRUE
```

8.4 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(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
```

8.5 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 when using this command to convert 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(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.R)
       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
       ....
```

8.6 Transferring Data Frames

To transfer a data frame from the R environment to the H2O instance, use `as.h2o()`. In the parentheses, specify the object in the R environment to convert to an H2O object. Optionally, include the name of the destination frame in H2O. Precede the destination frame name with `destination_frame =` and enclose the name 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(iris, destination_frame= "iris.hex")
|=====================================================================|
   100%
> head(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
```

8.7 Renaming Data Frames

To rename a dataframe on the server running H2O for a data set manipulated in R, use `h2o.assign()`. 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 it can be analyzed using H2O without overwriting the original data set.

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

8.8 Viewing Column Names

To view 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"
```

8.9 Getting Minimum and Maximum Values

To view 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
```

8.10 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.

```
> prosPath <- system.file("extdata", "prostate.csv", package="h2o")
> prostate.hex <- h2o.importFile(path = prosPath)
# Returns the percentiles at 0, 10, 20, ..., 100%
> prostate.qs <- quantile(prostate.hex$PSA, probs = (1:10)/10)
> prostate.qs
 10%     20%     30%     40%     50%     60%     70%     80%     90%    100%
 2.60    4.48    5.77    7.40    8.75   11.00  13.70   20.16   33.21  139.70
# Take the outliers or the bottom and top 10% of data
> PSA.outliers <- prostate.hex[prostate.hex$PSA <= prostate.qs["10%"] | prostate.
    hex$PSA >=   prostate.qs["90%"],]
# Check that the number of rows return is about 20% of the original data
> nrow(prostate.hex)
[1] 380
> nrow(PSA.outliers)
[1] 78
> nrow(PSA.outliers)/nrow(prostate.hex)
[1] 0.2052632
```

8.11 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(prostate.hex)
 ID                CAPSULE           AGE             RACE            DPROS
 Min.   :  1.00   Min.   :0.0000   Min.   :43.00   Min.   :0.000   Min.   :1.000
 1st Qu.: 95.75   1st Qu.:0.0000   1st Qu.:62.00   1st Qu.:1.000   1st Qu.:1.000
 Median :190.50   Median :0.0000   Median :67.00   Median :1.000   Median :2.000
 Mean   :190.50   Mean   :0.4026   Mean   :66.04   Mean   :1.087   Mean   :2.271
 3rd Qu.:285.25   3rd Qu.:1.0000   3rd Qu.:71.00   3rd Qu.:1.000   3rd Qu.:3.000
 Max.   :380.00   Max.   :1.0000   Max.   :79.00   Max.   :2.000   Max.   :4.000
 DCAPS           PSA               VOL             GLEASON
 Min.   :1.000   Min.   :  0.300   Min.   : 0.00   Min.   :0.000
 1st Qu.:1.000   1st Qu.:  4.900   1st Qu.: 0.00   1st Qu.:6.000
 Median :1.000   Median :  8.664   Median :14.20   Median :6.000
 Mean   :1.108   Mean   : 15.409   Mean   :15.81   Mean   :6.384
 3rd Qu.:1.000   3rd Qu.: 17.063   3rd Qu.:26.40   3rd Qu.:7.000
 Max.   :2.000   Max.   :139.700   Max.   :97.60   Max.   :9.000
```

8.12 Summarizing Data in a Table

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

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

```
# Counts of the ages of all patients
> head(as.data.frame(h2o.table(prostate.hex[,"AGE"])))
   AGE Count
1   43     1
2   47     1
3   50     2
4   51     3
5   52     2
6   53     4

# Two-way table of ages (rows) and race (cols) of all
   patients
# Example: For the first row there is one count of a 43
   year old that's labeled as RACE = 0
> h2o.table(prostate.hex[,c("AGE","RACE")])
H2OFrame with 53 rows and 3 columns
```

```

First 10 rows:
   AGE RACE count
1   53    1     3
2   61    1    12
3   70    0     1
4   75    1    11
5   74    1    13
6   76    2     1
7   53    2     1
8   52    1     2
9   61    2     1
10  60    1     9
```

8.13 Generating Random Numbers

To append a column of random numbers to an H2O data frame for testing/training data splits that are used 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()`.

```
> prosPath <- system.file("extdata", "prostate.csv",
   package="h2o")
> prostate.hex <- h2o.importFile(path = prosPath)

## Creates object for uniform distribution on prostate
   data set
> s <- h2o.runif(prostate.hex)
> summary (s)  ## Summarize the results of h2o.runif
 rnd
 Min.   :0.000863
 1st Qu.:0.239763
 Median :0.507936
 Mean   :0.506718
 3rd Qu.:0.765194
 Max.   :0.993178
```

```
## Create training set with threshold of 0.8
> prostate.train <- prostate.hex[s <= 0.8,]
##Assign name to training set
> prostate.train <- h2o.assign(prostate.train, "prostate.
   train")
## Create test set with threshold to filter values greater
    than 0.8
> prostate.test <- prostate.hex[s > 0.8,]
## Assign name to test set
> prostate.test <- h2o.assign(prostate.test, "prostate.
   test")
## Combine results of test & training sets, then display
   result
> nrow(prostate.train) + nrow(prostate.test)
[1] 380
> nrow(prostate.hex) ## Matches the full set
[1] 380
```

8.14 Splitting Frames

To generate two subsets (according to specified ratios) from an existing H2O data set for testing/training, use `h2o.splitFrame()`. h2o.splitFrame() returns contiguous sections of the data without random sampling.

```
# 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]]
```

8.15 Getting Frames

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

```
> prostate.hex <- h2o.getFrame(frame_id = "prostate.hex")
```

8.16 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 a GBM with the ID `GBM_8e4591a9b413407b983d73fbd9eb44cf` is in the key-value (KV) store.

```
> gbm.model <- h2o.getModel(model_id = "GBM_8
   e4591a9b413407b983d73fbd9eb44cf")
```

8.17 Listing H2O Objects

To generate a list of all H2O objects generated during a session and each objects size in bytes, use `h2o.ls()`.

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

8.18 Removing H2O Objects

To remove an H2O object on the server associated with the 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(ids = c("prostate.train","prostate.test"))
> h2o.ls()
```

8.19 Adding Functions

User-defined functions no longer need to be added explicitly to the H2O instance. An R function can be defined and executed against an H2OFrame.

```
# Create an R functional expression
> simpleFun <- function(x) { 2*x + 5 }
# Evaluate the expression across prostate's AGE column
> calculated <- simpleFun(prostate.hex[,"AGE"])
> h2o.cbind(prostate.hex[,"AGE"], calculated)

H2OFrame with 380 rows and 2 columns

First 10 rows:
   AGE AGE0
1    65  135
2    72  149
3    70  145
4    76  157
5    69  143
6    71  147
7    68  141
8    61  127
9    69  143
10   68  141
```

9 Running Models

This section describes how to run the following model types:

- Gradient Boosted Models (GBM)
- Generalized Linear Models (GLM)
- K-Means
- Principal Components Analysis (PCA)

as well as how to generate predictions.

9.1 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)
> h2o.init(nthreads = -1)
> data(iris)
> iris.hex <- as.h2o(iris,destination_frame = "iris.hex")
> iris.gbm <- h2o.gbm(y = 1, x = 2:5, training_frame =
   iris.hex, ntrees = 10,
    max_depth = 3,min_rows = 2, learn_rate = 0.2,
        distribution= "gaussian")

# To obtain the Mean-squared Error by tree from the model
   object:
> iris.gbm@model$scoring_history

Scoring History:
             timestamp   duration number_of_trees training
                 _MSE training_deviance
```

```
1  2015-09-11 09:50:16  0.005 sec                 1
    0.47256            0.47256
2  2015-09-11 09:50:16  0.008 sec                 2
    0.33494            0.33494
3  2015-09-11 09:50:16  0.011 sec                 3
    0.24291            0.24291
4  2015-09-11 09:50:16  0.014 sec                 4
    0.18414            0.18414
5  2015-09-11 09:50:16  0.017 sec                 5
    0.14363            0.14363
6  2015-09-11 09:50:16  0.020 sec                 6
    0.11677            0.11677
7  2015-09-11 09:50:16  0.023 sec                 7
    0.09916            0.09916
8  2015-09-11 09:50:16  0.026 sec                 8
    0.08649            0.08649
9  2015-09-11 09:50:16  0.029 sec                 9
    0.07761            0.07761
10 2015-09-11 09:50:16  0.032 sec                10
    0.07071            0.07071
```

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

```
> iris.gbm2 <- h2o.gbm(y = 5, x = 1:4, training_frame =
    iris.hex, ntrees = 15
                      , max_depth = 5, min_rows = 2, learn_
                        rate = 0.01, distribution= "
                        multinomial")

> iris.gbm2@model$training_metrics

H2OMultinomialMetrics: gbm
** Reported on training data. **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("iris.hex")`
MSE: (Extract with `h2o.mse`) 0.3293958
R^2: (Extract with `h2o.r2`) 0.5059063
Logloss: (Extract with `h2o.logloss`) 0.8533637
Confusion Matrix: Extract with `h2o.confusionMatrix(<model
    >,train=TRUE)`)
=================================================
           setosa versicolor virginica      Error    Rate
setosa         50          0         0 0.00000000  0 / 50
versicolor      0         49         1 0.02000000  1 / 50
virginica       0          1        49 0.02000000  1 / 50
Totals         50         50        50 0.01333333 2 / 150

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model
    >,train=TRUE)`
==================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.986667
2 2  1.000000
3 3  1.000000
```

9.2 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)`.

```
> prostate.hex <- h2o.importFile(path = "https://raw.
    github.com/h2oai/h2o/master/smalldata/logreg/prostate.
    csv"
                               , destination_frame = "
                                 prostate.hex")

> prostate.glm<-h2o.glm(y = "CAPSULE", x = c("AGE","RACE",
    "PSA","DCAPS"), training_frame = prostate.hex,
              family = "binomial", nfolds = 10, alpha =
                 0.5)
> prostate.glm@model$cross_validation_metrics

H2OBinomialMetrics: glm
** Reported on cross-validation data. **
Description: 10-fold cross-validation on training data

MSE:  0.2093902
R^2:  0.1294247
LogLoss:  0.6095525
AUC:  0.6909965
Gini:  0.381993
Null Deviance:  513.8229
Residual Deviance:  463.2599
AIC:  473.2599

Confusion Matrix for F1-optimal threshold:
         0   1    Error      Rate
```

```
0       122 105 0.462555  =105/227
1        41 112 0.267974   =41/153
Totals 163 217 0.384211  =146/380

Maximum Metrics:
                      metric threshold    value idx
1                     max f1  0.312978 0.605405 216
2                     max f2  0.138305 0.772727 377
3               max f0point5  0.400689 0.628141 110
4               max accuracy  0.400689 0.700000 110
5              max precision  0.998848 1.000000   0
6           max absolute_MCC  0.400689 0.357638 110
7 max min_per_class_accuracy  0.330976 0.621145 181
```

9.3 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).

```
> h2o.kmeans(training_frame = iris.hex, k = 3, x = 1:4)

Model Details:
==============

H2OClusteringModel: kmeans
Model ID:  K-means_model_R_1441989204383_30
Model Summary:
  number_of_rows number_of_clusters number_of_categorical_
     columns number_of_iterations within_cluster_sum_of_
     squares
1            150                  3
                                0                    8
                        139.09920
  total_sum_of_squares between_cluster_sum_of_squares
1            596.00000                      456.90080

```

```

H2OClusteringMetrics: kmeans
** Reported on training data. **


Total Within SS:  139.0992
Between SS:  456.9008
Total SS:  596
Centroid Statistics:
  centroid     size within_cluster_sum_of_squares
1        1 44.00000                      43.34674
2        2 50.00000                      47.35062
3        3 56.00000                      48.40184
```

9.4 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(path = ausPath)

> australia.pca <- h2o.prcomp(training_frame = australia.
    hex, transform = "STANDARDIZE",k = 3)
> australia.pca
Model Details:
==============

H2ODimReductionModel: pca
Model Key:  PCA_model_R_1441989204383_36
Importance of components:
                            pc1      pc2      pc3
Standard deviation     1.750703 1.512142 1.031181
Proportion of Variance 0.383120 0.285822 0.132917
Cumulative Proportion  0.383120 0.668942 0.801859
```

9.5 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. For classification, the predict column is the model's discrete prediction, based on maximum F1 by default; the individual class probabilities are the remaining columns in the data frame. It is common to utilize the p1 column for binary classification, if a raw probability is desired.

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

H2OFrame with 380 rows and 3 columns

First 10 rows:
   predict          p0        p1
1        0 0.74476265 0.2552373
```

```
2          1 0.39763451 0.6023655
3          1 0.41268532 0.5873147
4          1 0.37270563 0.6272944
5          1 0.64649990 0.3535001
6          1 0.43367145 0.5663285
7          1 0.26542251 0.7345775
8          1 0.06143281 0.9385672
9          0 0.73057373 0.2694263
10         1 0.46709293 0.5329071
```

10 H2O Community Resources

For more information about H2O, visit the following open-source sites:

H2O Full Documentation:	`http://docs.h2o.ai`
H2O-3 Github Repository:	`https://github.com/h2oai/h2o-3`
Open-source discussion forum:	`h2ostream@googlegroups.com` `groups.google.com/d/forum/h2ostream`
Issue tracking:	`http://jira.h2o.ai`

11 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`

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

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

12 Appendix: Commands

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

12.1 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.

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.splitFrame`: Split an existing H2O data set according to user-specified ratios.

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.

12.2 General Data Operations

Subscripting example to pull pieces from data object.

```
x[j]  ## note: chooses column J, not row J
x[i, j]
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.

12.3 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.

12.4 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.group_by`: Apply an aggregate function to each group of an H2O dataset.

Tabulation

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

12.5 Data Munging

General Column Manipulations

`is.na`: Display missing elements.

Element Index Selection

`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.

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.

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.

12.6 Data Modeling

Model Training: Supervised Learning

`h2o.deeplearning`: Perform Deep Learning neural networks on an H2OParsedData object.

`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.naiveBayes`: Build gradient boosted classification trees and gradient boosted regression trees on a parsed data set.

`h2o.prcomp`: Perform principal components analysis on the given data set.

`h2o.randomForest`: Perform random forest classification on a data set.

Model Training: Unsupervised Learning

`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.

`h2o.kmeans`: Perform k-means clustering on a data set.

Grid Search

`h2o.grid`: Efficient method to build multiple models with different hyperparameters.

Model Scoring

`h2o.predict`: Obtain predictions from various fitted H2O model objects.

Classification Model Helpers

`h2o.accuracy`: Get the between cluster sum of squares.

`h2o.auc`: Retrieve the AUC (area under ROC curve).

`h2o.confusionMatrix`: Display prediction errors for classification data from a column of predicted responses and a column of actual (reference) responses in H2O.

`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.

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.

Clustering Model Helper

`h2o.betweenss`: Get the between cluster sum of squares.

`h2o.centers`: Retrieve the Model Centers.

12.7 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.loadModel`: Load an H2OModel object from disk.

`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.startLogging`: Begin logging H2O R POST commands and error responses.

`h2o.stopLogging`: Stop logging H2O R POST commands and error responses.

H2O String Manipulation

`h2o.gsub`: String global substitution (all occurrences).

`h2o.strsplit`: String Split.

`h2o.sub`: String substitution (first occurrence).

`h2o.tolower`: Convert characters to lower case.

`h2o.toupper`: Convert characters to upper case.

`h2o.trim`: Trim spaces.