

# Package 'h2o'

August 9, 2014

## R topics documented:

h2o-package . . . . .	1
AAA_DownloadAndStartBeforeExamples . . . . .	2
apply . . . . .	3
as.data.frame.H2OParsedData . . . . .	3
as.factor . . . . .	4
as.h2o . . . . .	5
as.matrix.H2OParsedData . . . . .	6
as.table.H2OParsedData . . . . .	6
cbind.H2OParsedData . . . . .	7
colnames . . . . .	8
diff.H2OParsedData . . . . .	9
doNotCallThisMethod...Unsupported . . . . .	9
Extremes . . . . .	10
h2o.addFunction . . . . .	11
h2o.anomaly . . . . .	12
h2o.anyFactor . . . . .	13
h2o.assign . . . . .	13
h2o.clearLogs . . . . .	14
h2o.clusterInfo . . . . .	15
h2o.clusterStatus . . . . .	15
h2o.confusionMatrix . . . . .	16
h2o.createFrame . . . . .	17
h2o.cut . . . . .	18
h2o.ddply . . . . .	19
h2o.deeplearning . . . . .	20
h2o.downloadAllLogs . . . . .	24
h2o.downloadCSV . . . . .	25
h2o.exec . . . . .	26
h2o.exportFile . . . . .	26
h2o.gapStatistic . . . . .	27
h2o.gbm . . . . .	29
h2o.getFrame . . . . .	31
h2o.getGLMLambdaModel . . . . .	31

h2o.getLogPath . . . . .	32
h2o.getModel . . . . .	33
h2o.glm . . . . .	33
h2o.hitRatio . . . . .	36
h2o.ignoreColumns . . . . .	37
h2o.importFile . . . . .	38
h2o.importFolder . . . . .	39
h2o.importHDFS . . . . .	40
h2o.importURL . . . . .	42
h2o.init . . . . .	43
h2o.kmeans . . . . .	45
h2o.loadModel . . . . .	46
h2o.logAndEcho . . . . .	47
h2o.ls . . . . .	47
h2o.month . . . . .	48
h2o.mse . . . . .	49
h2o.naiveBayes . . . . .	50
h2o.openLog . . . . .	51
h2o.parseRaw . . . . .	52
h2o.pcr . . . . .	53
h2o.performance . . . . .	55
h2o.prcomp . . . . .	56
h2o.predict . . . . .	57
h2o.randomForest . . . . .	58
h2o.rm . . . . .	60
h2o.runif . . . . .	61
h2o.saveModel . . . . .	62
h2o.setLogPath . . . . .	63
h2o.shutdown . . . . .	64
h2o.SpeeDRF . . . . .	65
h2o.splitFrame . . . . .	67
h2o.startLogging . . . . .	67
h2o.stopLogging . . . . .	68
h2o.table . . . . .	69
h2o.uploadFile . . . . .	69
h2o.year . . . . .	71
H2OClient-class . . . . .	71
H2ODeepLearningGrid-class . . . . .	72
H2ODeepLearningModel-class . . . . .	73
H2ODRFGrid-class . . . . .	74
H2ODRFModel-class . . . . .	75
H2OGBMGrid-class . . . . .	76
H2OGBMModel-class . . . . .	77
H2OGLMGrid-class . . . . .	78
H2OGLMModel-class . . . . .	79
H2OGLMModelList-class . . . . .	80
H2OGrid-class . . . . .	80
H2OKMeansGrid-class . . . . .	81

H2OKMeansModel-class . . . . .	82
H2OModel-class . . . . .	83
H2ONBModel-class . . . . .	83
H2OParsedData-class . . . . .	84
H2OPCAModel-class . . . . .	88
H2OPerfModel-class . . . . .	89
H2ORawData-class . . . . .	90
H2OSpeeDRFGrid-class . . . . .	90
H2OSpeeDRFModel-class . . . . .	91
head . . . . .	92
ifelse . . . . .	93
is.factor . . . . .	94
levels . . . . .	94
mean.H2OParsedData . . . . .	95
nrow . . . . .	96
plot.H2OPerfModel . . . . .	96
quantile.H2OParsedData . . . . .	97
Round . . . . .	98
screeplot.H2OPCAModel . . . . .	99
sd . . . . .	100
str . . . . .	101
sum . . . . .	101
summary . . . . .	102
summary.H2OPCAModel . . . . .	103
unique.H2OParsedData . . . . .	103
zzz_ShutdownAfterExamples . . . . .	104

---

h2o-package

*H2O R Interface*


---

## Description

This is a package for running H2O via its REST API from within R. To communicate with a H2O instance, the version of the R package must match the version of H2O. When connecting to a new H2O cluster, it is necessary to re-run the initializer.

## Details

```

Package: h2o
Type: Package
Version: 2.5.0.1461
Date: 2014-05-15
License: Apache License (== 2.0)
Depends: R (>= 2.13.0), RCurl, rjson, statmod, tools, methods, utils

```

This package allows the user to run basic H2O commands using R commands. In order to use it, you must first have H2O running (See [How to Start H2O](#)). To run H2O on your local machine, call `h2o.init` without any arguments, and H2O will be automatically launched on <http://127.0.0.1:54321>, where the IP is "127.0.0.1" and the port is 54321. If H2O is running on a cluster, you must provide the IP and port of the remote machine as arguments to the `h2o.init()` call.

H2O supports a number of standard statistical models, such as GLM, K-means, and Random Forest classification. For example, to run GLM, call `h2o.glm` with the H2O parsed data and parameters (response variable, error distribution, etc...) as arguments. (The operation will be done on the server associated with the data object where H2O is running, not within the R environment).

Note that no actual data is stored in the R workspace; and no actual work is carried out by R. R only saves the named objects, which uniquely identify the data set, model, etc on the server. When the user makes a request, R queries the server via the REST API, which returns a JSON file with the relevant information that R then displays in the console.

### Author(s)

Anqi Fu, Tom Kraljevic and Petr Maj, with contributions from the 0xdata team

Maintainer: Ariel Rao <ariel@0xdata.com>

### References

- [0xdata Homepage](#)
- [H2O Documentation](#)
- [H2O on Github](#)

### Examples

```
# Check connection with H2O and ensure local H2O R package matches server version.
# Optionally, ask for startH2O to start H2O if it's not already running.
# Note that for startH2O to work, the IP must be 127.0.0.1 or localhost with port 54321.
library(h2o)
localH2O = h2o.init(ip = "127.0.0.1", port = 54321, startH2O = TRUE)

# Import iris dataset into H2O and print summary
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)

# Attach H2O R package and run GLM demo
??h2o
demo(package = "h2o")
demo(h2o.prcomp)
```

---

 AAA\_DownloadAndStartBeforeExamples

*Download H2O jar file and Start H2O cloud before examples run (for H2O developers only)*

---

### Description

AAA\_DownloadAndStartBeforeExamples, download H2O jar file and start H2O cloud before examples run. This is only relevant for H2O developers during the building of the CRAN package.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
```

---

 apply

*Applies a function over an H2O parsed data object.*

---

### Description

Applies a function over an H2O parsed data object (an array).

### Usage

```
apply(X, MARGIN, FUN, ...)
```

### Arguments

X	An <a href="#">H2OParsedData</a> object.
MARGIN	The margin along which the function should be applied
FUN	The function to be applied by H2O.
...	Optional arguments to FUN. (Currently unimplemented).

### Value

Produces a new [H2OParsedData](#) of the output of the applied function. The output is stored in H2O so that it can be used in subsequent H2O processes.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(apply(iris.hex, 1, sum))
```

---

```
as.data.frame.H2OParsedData
```

*Converts a parsed H2O object to a data frame.*

---

### Description

Convert an [H2OParsedData](#) object to a data frame, which allows subsequent data frame operations within the R environment.

### Usage

```
## S3 method for class 'H2OParsedData'  
as.data.frame(x, ...)
```

### Arguments

x                    An [H2OParsedData](#) object.  
...                  Additional arguments to be passed to or from methods.

### Value

Returns a data frame in the R environment. Note that this call establishes the data set in the R environment, and subsequent operations on the data frame take place within R, not H2O. When data are large, users may experience

### Examples

```
library(h2o)  
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)  
prosPath = system.file("extdata", "prostate.csv", package="h2o")  
prostate.hex = h2o.importFile(localH2O, path = prosPath)  
prostate.data.frame <- as.data.frame(prostate.hex)  
summary(prostate.data.frame)  
head(prostate.data.frame)
```

---

```
as.factor
```

*Converts a column from numeric to factor*

---

### Description

Specify a column type to be factor (also called categorical or enumerative), rather than numeric.

### Usage

```
as.factor(x)
```

**Arguments**

x                    A column in an object of class `H2OParsedData`, or data frame.

**Value**

Returns the original object of class `H2OParsedData`, with the requested column specified as a factor, rather than numeric.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.hex[,4] = as.factor(prostate.hex[,4])
summary(prostate.hex)
```

---

as.h2o                    *Converts an R object to an H2O object*

---

**Description**

Convert an R object to an H2O object, copy contents of the object to the running instance of H2O

**Usage**

```
as.h2o(client, object, key = "", header, sep = "")
```

**Arguments**

client	The h2o.init object that facilitates communication between R and H2O.
object	The object in the R environment to be converted to an H2O object.
key	(Optional) A reference assigned to the object in the instance of H2O (the key part of the key-value store, where the value is the R object to be converted.)
header	(Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parse

**Details**

The R object to be converted to an H2O object should be named so that it can be used in subsequent analysis. Also note that the R object is converted to a parsed H2O data object, and will be treated as a data frame by H2O in subsequent analysis.

**Value**

Converts an R object to an H2O Parsed data object.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

data(iris)
summary(iris)
iris.r <- iris
iris.h2o <- as.h2o(localH2O, iris.r, key="iris.h2o")
class(iris.h2o)
```

---

as.matrix.H2OParsedData

*Converts a parsed H2O object to a matrix.*

---

**Description**

Convert an [H2OParsedData](#) object to a matrix, which allows subsequent data frame operations within the R environment.

**Usage**

```
## S3 method for class 'H2OParsedData'
as.matrix(x, ...)
```

**Arguments**

x                    An [H2OParsedData](#) object.  
...                   Additional arguments to be passed to or from methods.

**Value**

Returns a matrix in the R environment. Note that this call establishes the data set in the R environment, and subsequent operations on the matrix take place within R, not H2O. When data are large, users may experience significant slowdown.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.matrix <- as.matrix(prostate.hex)
summary(prostate.matrix)
head(prostate.matrix)
```

---

`as.table.H2OParsedData`*Converts a parsed H2O object to a table in R.*

---

**Description**

Convert an `H2OParsedData` object to a table, which allows subsequent data frame operations within the R environment.

**Usage**

```
## S3 method for class 'H2OParsedData'  
as.table(x, ...)
```

**Arguments**

`x` An `H2OParsedData` object.  
`...` Additional arguments to be passed to or from methods.

**Value**

Returns a table in the R environment. Note that this call establishes the data set in the R environment, and subsequent operations on the table take place within R, not H2O. When data are large, users may experience significant slowdown.

**Examples**

```
library(h2o)  
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)  
prosPath = system.file("extdata", "prostate.csv", package="h2o")  
prostate.hex = h2o.importFile(localH2O, path = prosPath)  
prostate.table <- as.table(prostate.hex)  
summary(prostate.table)  
head(prostate.table)
```

---

`cbind.H2OParsedData` *Combine H2O Datasets by Columns*

---

**Description**

`cbind.H2OParsedData`, a method for the `cbind` generic. Takes a sequence of H2O datasets and combines them by column.

**Usage**

```
## S3 method for class 'H2OParsedData'  
cbind(..., deparse.level = 1)
```

**Arguments**

- ... A sequence of [H2OParsedData](#) arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.
- deparse.level Integer controlling the construction of column names. Currently unimplemented.

**Value**

An [H2OParsedData](#) object containing the combined ... arguments column-wise.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.cbind = cbind(prostate.hex, prostate.hex)
head(prostate.cbind)
```

---

colnames

*Returns column names for a parsed H2O data object.*


---

**Description**

Returns column names for an [H2OParsedData](#) object.

**Usage**

```
colnames(x, do.NULL = TRUE, prefix = "col")
```

**Arguments**

- x An [H2OParsedData](#) object.
- do.NULL Logical value. If FALSE and names are NULL, names are created.
- prefix Character string denoting prefix for created column names.

**Value**

Returns a vector of column names.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)
colnames(iris.hex)
```

---

diff.H2OParsedData      *Lagged Differences of H2O Dataset*

---

### Description

diff.H2OParsedData, a method for the `diff` generic. Calculate the lagged and iterated differences of a single numeric column in a H2O dataset.

### Usage

```
## S3 method for class 'H2OParsedData'
diff(x, lag = 1, differences = 1, ...)
```

### Arguments

<code>x</code>	An <code>H2OParsedData</code> object.
<code>lag</code>	An integer indicating which lag to use. Must be greater than 0.
<code>differences</code>	An integer indicating the order of the differences. Must be greater than 0.
<code>...</code>	Potential further arguments. (Currently unimplemented).

### Value

An `H2OParsedData` object with a single numeric column containing the successive lagged and iterated differences. If `differences = 1`, this is equivalent to `x[(1+lag):n] - x[1:(n-lag)]`. For differences greater than 1, the algorithm is applied recursively to `x`.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
diff(prostate.hex$AGE)
```

---

doNotCallThisMethod...Unsupported

*Internal method used for testing at the API level.*

---

### Description

Fetches all of the model JSON for the model key passed in.

### Usage

```
doNotCallThisMethod...Unsupported(h2o, key)
```

**Arguments**

h2o	An h2o object returned from <code>h2o.init()</code> that represents the connection to the h2o cloud.
key	Any valid model key that exists in the h2o cluster.

**Value**

A blob of JSON.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
hex <- as.h2o(localH2O, iris)
m <- h2o.randomForest(x = 1:4, y = 5, data = hex)
doNotCallThisMethod...Unsupported(localH2O, m@key)
```

---

Extremes

*Maxima and Minima*

---

**Description**

Calculates the (parallel) minimum of the input values. This method extends the `min` generic to deal with `H2OParsedData` objects.

**Usage**

```
max(..., na.rm = FALSE)
min(..., na.rm = FALSE)
```

**Arguments**

...	Numeric, character or <code>H2OParsedData</code> arguments.
na.rm	Logical value where FALSE does not remove NA's in the calculation and TRUE removes NA's in the calculation.

**Value**

Returns the maximum or minimum over all the input arguments. For a `H2OParsedData` object, the function is calculated over all entries in the dataset. An error will occur if any of those entries is non-numeric.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package = "h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath, key = "australia.hex")
min(australia.hex)
```

---

h2o.addFunction	<i>Adds an R function to H2O</i>
-----------------	----------------------------------

---

### Description

Add a function defined in R to the H2O server, so it is recognized for future operations on H2O. This method is necessary because R functions are not automatically pushed across via the REST API to H2O.

### Usage

```
h2o.addFunction(object, fun, name)
```

### Arguments

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
fun	A function in R. Currently, only a subset of the R syntax is recognizable by H2O, and functions that fall outside this set will be rejected. Values referred to by fun must be defined within H2O, e.g. a H2O dataset must be referred to by its key name, not its H2OParsedData R variable name.
name	(Optional) A character string giving the name that the function should be saved under in H2O. If missing, defaults to the name that the function is saved under in R.

### Details

This method is intended to be used in conjunction with [h2o.ddply](#). The user must explicitly add the function he or she wishes to apply to H2O. Otherwise, the server will not recognize a function reference that only exists in R.

### See Also

[h2o.ddply](#)

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.addFunction(localH2O, function(x) { 2*x + 5 }, "simpleFun")
```

---

`h2o.anomaly`*Anomaly Detection via H2O Deep Learning Model*

---

**Description**

Detect anomalies in a H2O dataset using a H2O deep learning model with auto-encoding.

**Usage**

```
h2o.anomaly(data, model, key = "", threshold = -1.0)
```

**Arguments**

<code>data</code>	An <a href="#">H2OParsedData</a> object.
<code>model</code>	An <a href="#">H2ODeepLearningModel</a> object that represents the model to be used for anomaly detection. Must have been built with the argument <code>autoencoder = TRUE</code> in <a href="#">h2o.deeplearning</a> .
<code>key</code>	(Optional) The unique hex key assigned to the resulting dataset. If none is given, a key will automatically be generated.
<code>threshold</code>	(Optional) Threshold of reconstruction error for rows to be displayed in logs. If set to -1.0, this defaults to 10 times the MSE.

**Value**

Returns an [H2OParsedData](#) object with a single column containing the reconstruction MSE.

**See Also**

[H2OParsedData](#), [H2ODeepLearningModel](#), [h2o.deeplearning](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.dl = h2o.deeplearning(x = 3:9, y = 2, data = prostate.hex, autoencoder = TRUE,
                             hidden = c(10, 10), epochs = 5)
prostate.anon = h2o.anomaly(prostate.hex, prostate.dl)
head(prostate.anon)
```

---

h2o.anyFactor	<i>Determine if an H2O parsed data object contains categorical data.</i>
---------------	--

---

**Description**

Checks if an H2O parsed data object has any columns of categorical data.

**Usage**

```
h2o.anyFactor(x)
```

**Arguments**

x                    An [H2OParsedData](#) object.

**Value**

Returns a logical value indicating whether any of the columns in x are factors.

**See Also**

[H2OParsedData](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.anyFactor(iris.hex)
```

---

h2o.assign	<i>Assigns an H2O hex.key to an H2O object so that it can be utilized in subsequent calls</i>
------------	---

---

**Description**

Allows users to assign H2O hex.keys to objects in their R environment so that they can manipulate H2O data frames and parsed data objects.

**Usage**

```
h2o.assign(data, key)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object
key	The hex key to be associated with the H2O parsed data object

**Value**

The function returns an object of class [H2OParsedData](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
psa.qs = quantile(prostate.hex$PSA)
PSA.outliers = prostate.hex[prostate.hex$PSA <= psa.qs[2] | prostate.hex$PSA >= psa.qs[10],]
PSA.outliers = h2o.assign(PSA.outliers, "PSA.outliers")
summary(PSA.outliers)
head(prostate.hex)
head(PSA.outliers)
```

---

h2o.clearLogs

*Delete All H2O R Logs*

---

**Description**

Clear all H2O R command and error response logs from local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.clearLogs()
```

**See Also**

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.openLog](#), [h2o.getLogPath](#), [h2o.setLogPath](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
h2o.clearLogs()
```

---

h2o.clusterInfo      *Get Information on H2O Cluster*

---

### Description

Display the name, version, uptime, total nodes, total memory, total cores and health of a cluster running H2O.

### Usage

```
h2o.clusterInfo(client)
```

### Arguments

client      An [H2OClient](#) object containing the IP address and port of the server running H2O.

### See Also

[H2OClient](#)

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.clusterInfo(localH2O)
```

---

h2o.clusterStatus      *Retrieve Status of H2O Cluster*

---

### Description

Retrieve information on the status of the cluster running H2O.

### Usage

```
h2o.clusterStatus(client)
```

### Arguments

client      An [H2OClient](#) object containing the IP address and port of the server running H2O.

## Details

This method prints the status of the H2O cluster represented by `client`, consisting of the following information:

- `Version`: The version of H2O running on the cluster.
- `Cloud Name`: Name of the cluster.
- `Node Name`: Name of the node. (Defaults to the HTTP address).
- `Cloud Size`: Number of nodes in the cluster.

Furthermore, for each node, this function displays:

- `name`: Name of the node.
- `value_size_bytes`: Amount of data stored on the node.
- `free_mem_bytes`: Amount of free memory on the JVM.
- `max_mem_bytes`: Maximum amount of memory that the JVM will attempt to use.
- `free_disk_bytes`: Amount of free space on the disk that launched H2O.
- `max_disk_bytes`: Size of disk that launched H2O.
- `num_cpus`: Number of CPUs reported by JVM.
- `system_load`: Average system load.
- `rpcs`: Number of remote procedure calls.
- `last_contact`: Number of seconds since last heartbeat.

## See Also

[H2OClient](#), [h2o.init](#)

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.clusterStatus(localH2O)
```

---

`h2o.confusionMatrix`    *Build a Confusion Matrix from H2O Classification Predictions*

---

## Description

Constructs a confusion matrix from a column of predicted responses and a column of actual (reference) responses in H2O. Note that confusion matrices describe prediction errors for classification data only.

## Usage

```
h2o.confusionMatrix(data, reference)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object that represents the predicted response values. (Must be a single column).
reference	An <a href="#">H2OParsedData</a> object that represents the actual response values. Must have the same dimensions as data.

**Value**

Returns a confusion matrix with the actual value counts along the rows and the predicted value counts along the columns.

**See Also**

[H2OParsedData](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.gbm = h2o.gbm(x = 3:9, y = 2, data = prostate.hex)
prostate.pred = h2o.predict(prostate.gbm)
h2o.confusionMatrix(prostate.pred[,1], prostate.hex[,2])
```

---

h2o.createFrame	<i>Create an H2O Frame</i>
-----------------	----------------------------

---

**Description**

Create an H2O data frame from scratch, with optional randomization. Supports categoricals, integers, reals and missing values.

**Usage**

```
h2o.createFrame(object, key, rows, cols, seed, randomize, value, real_range,
  categorical_fraction, factors, integer_fraction, integer_range,
  missing_fraction, response_factors)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
key	The unique hex key assigned to the created frame.
rows	Number of rows
cols	Number of columns

seed	Random number seed
randomize	Whether frame should be randomized
value	Constant value (for randomize=false)
real_range	Range for real variables (-range ... range)
categorical_fraction	Fraction of categorical columns (for randomize=true)
factors	Factor levels for categorical variables
integer_fraction	Fraction of integer columns (for randomize=true)
integer_range	Range for integer variables (-range ... range)
missing_fraction	Fraction of missing values
response_factors	Number of factor levels of the first column (1=real, 2=binomial, N=multinomial)

**Value**

Returns an H2O data frame.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, beta = TRUE)
myframe = h2o.createFrame(localH2O, 'myframekey', rows = 1000, cols = 10,
  seed = -12301283, randomize = TRUE, value = 0, real_range = 2.0,
  categorical_fraction = 0.2, factors = 100,
  integer_fraction = 0.2, integer_range = 100, missing_fraction = 0.1,
  response_factors = 2)

head(myframe)
summary(myframe)
h2o.shutdown(localH2O)
```

---

h2o.cut

---

*Convert H2O Numeric Data to Factor*


---

**Description**

Divides the range of the H2O data into intervals and codes the values according to which interval they fall in. The leftmost interval corresponds to level one, the next is level two, etc.

**Usage**

```
h2o.cut(x, breaks)
```

**Arguments**

x	An <a href="#">H2OParsedData</a> object with numeric columns.
breaks	A numeric vector of two or more unique cut points.

**Value**

A [H2OParsedData](#) object containing the factored data with intervals as levels.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)

# Cut sepal length column into intervals determined by min/max/quantiles
sepal_len.cut = h2o.cut(iris.hex$sepal_len, c(4.2, 4.8, 5.8, 6, 8))
head(sepal_len.cut)
summary(sepal_len.cut)
```

---

h2o.ddply

*Split H2O dataset, apply function, and return results*


---

**Description**

For each subset of a H2O dataset, apply a user-specified function, then combine the results.

**Usage**

```
h2o.ddply(.data, .variables, .fun = NULL, ..., .progress = "none")
```

**Arguments**

.data	An <a href="#">H2OParsedData</a> object to be processed.
.variables	Variables to split .data by, either the indices or names of a set of columns.
.fun	Function to apply to each subset grouping. Must have been pushed to H2O using <a href="#">h2o.addFunction</a> .
...	Additional arguments passed on to .fun. (Currently unimplemented).
.progress	Name of the progress bar to use. (Currently unimplemented).

**Details**

This is an extension of the plyr library's ddply function to datasets loaded into H2O.

**Value**

An [H2OParsedData](#) object containing the results from the split/apply operation, arranged row-by-row.

**References**

Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. <http://www.jstatsoft.org/v40/i01/>.

**See Also**

[h2o.addFunction](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# 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
res = h2o.ddply(iris.hex, "class", fun)
head(res)
```

---

h2o.deeplearning

*H2O: Deep Learning Neural Networks*

---

**Description**

Performs Deep Learning neural networks on an [H2OParsedData](#) object.

**Usage**

```
h2o.deeplearning(x, y, data, key = "", override_with_best_model, classification = TRUE,
  nfold = 0, validation, checkpoint = "", autoencoder, use_all_factor_levels,
  activation, hidden, epochs, train_samples_per_iteration, seed, adaptive_rate,
  rho, epsilon, rate, rate_annealing, rate_decay, momentum_start,
  momentum_ramp, momentum_stable, nesterov_accelerated_gradient,
  input_dropout_ratio, hidden_dropout_ratios, l1, l2, max_w2,
  initial_weight_distribution, initial_weight_scale, loss,
  score_interval, score_training_samples, score_validation_samples,
  score_duty_cycle, classification_stop, regression_stop, quiet_mode,
```

max\_confusion\_matrix\_size, max\_hit\_ratio\_k, balance\_classes, max\_after\_balance\_size, score\_validation\_sampling, diagnostics, variable\_importances, fast\_mode, ignore\_const\_cols, force\_load\_balance, replicate\_training\_data, single\_node\_mode, shuffle\_training\_data, sparse, col\_major)

### Arguments

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
override_with_best_model	If enabled, override the final model with the best model found during training. Defaults to true.
classification	(Optional) A logical value indicating whether the algorithm should conduct classification.
nfolds	(Optional) Number of folds for cross-validation. If nfolds >= 2, then validation must remain empty.
validation	(Optional) An <a href="#">H2OParsedData</a> object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data when nfolds = 0.
checkpoint	"Model checkpoint (either key or H2ODeepLearningModel) to resume training with."
activation	A string indicating the activation function to use. Must be either "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout" or "MaxoutWithDropout".
hidden	Hidden layer sizes (e.g. c(100,100))
autoencoder	Enable auto-encoder for model building.
use_all_factor_levels	Use all factor levels of categorical variables. Otherwise, the first factor level is omitted (without loss of accuracy). Useful for variable importances and auto-enabled for autoencoder.
epochs	How many times the dataset should be iterated (streamed), can be fractional
train_samples_per_iteration	Number of training samples (globally) per MapReduce iteration. Special values are 0: one epoch, -1: all available data (e.g., replicated training data)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
adaptive_rate	Adaptive learning rate (ADADELTA)
rho	Adaptive learning rate time decay factor (similarity to prior updates)
epsilon	Adaptive learning rate smoothing factor (to avoid divisions by zero and allow progress)

rate Learning rate (higher => less stable, lower => slower convergence)  
 rate\_annealing Learning rate annealing:  $\text{rate} / (1 + \text{rate\_annealing} * \text{samples})$   
 rate\_decay Learning rate decay factor between layers (N-th layer:  $\text{rate} * \alpha^{(N-1)}$ )  
 momentum\_start Initial momentum at the beginning of training (try 0.5)  
 momentum\_ramp Number of training samples for which momentum increases  
 momentum\_stable Final momentum after the ramp is over (try 0.99)  
 nesterov\_accelerated\_gradient Use Nesterov accelerated gradient (recommended)  
 input\_dropout\_ratio Input layer dropout ratio (can improve generalization, try 0.1 or 0.2)  
 hidden\_dropout\_ratios Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5  
 l1 L1 regularization (can add stability and improve generalization, causes many weights to become 0)  
 l2 L2 regularization (can add stability and improve generalization, causes many weights to be small)  
 max\_w2 Constraint for squared sum of incoming weights per unit (e.g. for Rectifier)  
 initial\_weight\_distribution Initial Weight Distribution  
 initial\_weight\_scale Uniform: -value...value, Normal: stddev  
 loss Loss function  
 score\_interval Shortest time interval (in secs) between model scoring  
 score\_training\_samples Number of training set samples for scoring (0 for all)  
 score\_validation\_samples Number of validation set samples for scoring (0 for all)  
 score\_duty\_cycle Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring).  
 classification\_stop Stopping criterion for classification error fraction on training data (-1 to disable)  
 regression\_stop Stopping criterion for regression error (MSE) on training data (-1 to disable)  
 quiet\_mode Enable quiet mode for less output to standard output  
 max\_confusion\_matrix\_size Max. size (number of classes) for confusion matrices to be shown  
 max\_hit\_ratio\_k Max. number (top K) of predictions to use for hit ratio computation (for multi-class only, 0 to disable)

balance_classes	Balance training data class counts via over/under-sampling (for imbalanced data)
max_after_balance_size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
score_validation_sampling	Method used to sample validation dataset for scoring
diagnostics	Enable diagnostics for hidden layers
variable_importances	Compute variable importances for input features (Gedeon method) - can be slow for large networks
fast_mode	Enable fast mode (minor approximation in back-propagation)
ignore_const_cols	Ignore constant training columns (no information can be gained anyway)
force_load_balance	Force extra load balancing to increase training speed for small datasets (to keep all cores busy)
replicate_training_data	Replicate the entire training dataset onto every node for faster training on small datasets
single_node_mode	Run on a single node for fine-tuning of model parameters
shuffle_training_data	Enable shuffling of training data (recommended if training data is replicated and train_samples_per_iteration is close to #nodes x #rows)
sparse	Sparse data handling (Experimental).
col_major	Use a column major weight matrix for input layer. Can speed up forward propagation, but might slow down backpropagation (Experimental).

### Value

An object of class [H2ODeepLearningModel](#) with slots key, data, valid (the validation dataset) and model, where the last is a list of the following components:

confusion	The confusion matrix of the response, with actual observations as rows and predicted values as columns.
train_class_err	Classification error on the training dataset.
train_sqr_err	Mean-squared error on the training dataset.
valid_class_err	Classification error on the validation dataset.
valid_sqr_err	Mean-squared error on the validation dataset.

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.deeplearning(x = 1:4, y = 5, data = iris.hex, activation = "Tanh",
                 hidden = c(10, 10), epochs = 5)
```

---

h2o.downloadAllLogs     *Download H2O Log Files to Disk*

---

## Description

Download all H2O log files to local disk. Generally used for debugging purposes.

## Usage

```
h2o.downloadAllLogs(client, dirname = ".", filename = NULL)
```

## Arguments

client	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
dirname	(Optional) A character string indicating the directory that the log file should be saved in.
filename	(Optional) A character string indicating the name that the log file should be saved to.

## See Also

[H2OClient](#)

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.downloadAllLogs(localH2O, dirname = getwd(), filename = "h2o_logs.log")
file.info(paste(getwd(), "h2o_logs.log", sep = .Platform$file.sep))
file.remove(paste(getwd(), "h2o_logs.log", sep = .Platform$file.sep))
```

---

h2o.downloadCSV	<i>Download H2O Data to Disk</i>
-----------------	----------------------------------

---

## Description

Download a H2O dataset to a CSV file on local disk.

## Usage

```
h2o.downloadCSV(data, filename, quiet = FALSE)
```

## Arguments

data	An <a href="#">H2OParsedData</a> object to be downloaded.
filename	A character string indicating the name that the CSV file should be saved to.
quiet	(Optional) If TRUE, suppress status messages and progress bar.

## Details

WARNING: Files located on the H2O server may be very large! Make sure you have enough hard drive space to accommodate the entire file.

## See Also

[H2OParsedData](#)

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)

myFile = paste(getwd(), "my_iris_file.csv", sep = .Platform$file.sep)
h2o.downloadCSV(iris.hex, myFile)
file.info(myFile)
file.remove(myFile)
```

---

h2o.exec	<i>Execute a Command on H2O</i>
----------	---------------------------------

---

**Description**

Directly send over and execute an R expression in the H2O console.

**Usage**

```
h2o.exec(expr_to_execute, h2o, dest_key)
```

**Arguments**

expr_to_execute	An R expression that is supported by H2O. Currently, basic subsetting, arithmetic operations, logical filters, and simple expressions like <code>dim</code> are allowed.
h2o	(Optional)Point to an instance of H2O. If not given, <code>h2o.exec</code> will try to guess.
dest_key	(Optional)Give a destination key to the expression to be executed. If not given, <code>h2o.exec</code> will try to guess.

**Value**

A [H2OParsedData](#) object containing the result of the expression.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
hex <- as.h2o(localH2O, iris)
res1 = h2o.exec(hex[,1] + hex[,2])
head(res1)
res2 = h2o.exec(hex[,1] + hex[, 2] + hex[, 3] * hex[,4] / hex[,1])
head(res2)
res3<- h2o.exec(hex$nc<- ifelse(hex[,1]<5,log(hex[,3]+1),hex["Petal.Width"]/hex$Sepal.Width))
head(res3)
head(hex)
```

---

h2o.exportFile	<i>Export H2O Data Frame to a File.</i>
----------------	---

---

**Description**

Export an H2O Data Frame (which can be either VA or FV) to a file. This file may be on the H2O instance's local filesystem, or to HDFS (preface the path with `hdfs://`) or to S3N (preface the path with `s3n://`).

**Usage**

```
## Default method:
h2o.exportFile(data, path, force = FALSE)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> data frame.
path	The path to write the file to. Must include the directory and filename. May be prefaced with hdfs:// or s3n://. Each row of data appears as one line of the file.
force	(Optional) If force = TRUE any existing file will be overwritten. Otherwise if the file already exists the operation will fail.

**Value**

None. (The function will stop if it fails.)

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.exportFile(iris.hex, path = "/path/on/h2o/server/filesystem/iris.csv")
h2o.exportFile(iris.hex, path = "hdfs://path/in/hdfs/iris.csv")
h2o.exportFile(iris.hex, path = "s3n://path/in/s3/iris.csv")

## End(Not run)
```

---

h2o.gapStatistic

*Compute Gap Statistic from H2O Dataset*


---

**Description**

Compute the gap statistic of a H2O dataset. The gap statistic is a measure of the goodness of fit of a clustering algorithm. For each number of clusters  $k$ , it compares  $\log(W(k))$  with  $E^*[\log(W(k))]$  where the latter is defined via bootstrapping.

**Usage**

```
h2o.gapStatistic(data, cols = "", K.max = 10, B = 100, boot_frac = 0.33, seed = 0)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object.
cols	(Optional) A vector of column names or indices indicating the features to analyze. By default, all columns in the dataset are analyzed.
K.max	The maximum number of clusters to consider. Must be at least 2.
B	A positive integer indicating the number of Monte Carlo (bootstrap) samples for simulating the reference distribution.
boot_frac	Fraction of data size to replicate in each Monte Carlo simulation.
seed	(Optional) Random number seed for breaking ties between equal probabilities.

**Details**

IMPORTANT: Currently, you must initialize H2O with the flag `beta = TRUE` in `h2o.init` in order to use this method!

**Value**

A list containing the following components:

log_within_ss	Log of the pooled cluster within sum of squares per value of k.
boot_within_ss	Monte Carlo bootstrap replicate averages of log_within_ss per value of k.
se_boot_within_ss	Standard error from the Monte Carlo simulated data for each iteration.
gap_stats	Gap statistics per value of k.
k_opt	Optimal number of clusters.

**References**

Tibshirani, R., Walther, G. and Hastie, T. (2001). Estimating the number of data clusters via the Gap statistic. *Journal of the Royal Statistical Society B*, **63**, 411-423.

Tibshirani, R., Walther, G. and Hastie, T. (2000). Estimating the number of clusters in a dataset via the Gap statistic. Technical Report. Stanford.

**See Also**

[H2OParsedData](#), `h2o.kmeans`

**Examples**

```
# Currently still in beta, so don't automatically run example
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, beta = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.gapStatistic(iris.hex, K.max = 10, B = 100)

## End(Not run)
```

h2o.gbm

*H2O: Gradient Boosted Machines***Description**

Builds gradient boosted classification trees, and gradient boosed regression trees on a parsed data set.

**Usage**

```
h2o.gbm(x, y, distribution = "multinomial", data, key = "", n.trees = 10,
  interaction.depth = 5, n.minobsinnode = 10, shrinkage = 0.1, n.bins = 100,
  importance = FALSE, nfolds = 0, validation, balance.classes = FALSE,
  max.after.balance.size = 5)
```

**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the GBM model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index number starting at 0, and increasing from left to right. (The response must be either an integer or a categorical variable).
distribution	The type of GBM model to be produced: classification is "multinomial" (default), "gaussian" is used for regression.
data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
n.trees	(Optional) Number of trees to grow. Must be a nonnegative integer.
interaction.depth	(Optional) Maximum depth to grow the tree.
n.minobsinnode	(Optional) Minimum number of rows to assign to teminal nodes.
shrinkage	(Optional) A learning-rate parameter defining step size reduction.
n.bins	(Optional) Number of bins to use in building histogram.
importance	(Optional) A logical value indicating whether variable importance should be calculated. This will increase the amount of time for the algorithm to complete.
nfolds	(Optional) Number of folds for cross-validation. If nfolds >= 2, then validation must remain empty.
validation	(Optional) An <a href="#">H2OParsedData</a> object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data when nfolds = 0.
balance.classes	(Optional) Balance training data class counts via over/under-sampling (for imbalanced data)
max.after.balance.size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)

**Value**

An object of class `H2OGBMModel` with slots `key`, `data`, `valid` (the validation dataset) and `model`, where the last is a list of the following components:

<code>type</code>	The type of the tree.
<code>n.trees</code>	Number of trees grown.
<code>oob_err</code>	Out of bag error rate.
<code>forest</code>	A matrix giving the minimum, mean, and maximum of the tree depth and number of leaves.
<code>confusion</code>	Confusion matrix of the prediction when classification model is specified.

**References**

1. Elith, Jane, John R Leathwick, and Trevor Hastie. "A Working Guide to Boosted Regression Trees." *Journal of Animal Ecology* 77.4 (2008): 802-813
2. Friedman, Jerome, Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. "Discussion of Boosting Papers." *Ann. Statist* 32 (2004): 102-107
3. Hastie, Trevor, Robert Tibshirani, and J Jerome H Friedman. *The Elements of Statistical Learning*. Vol.1. N.p.: Springer New York, 2001. [http://www.stanford.edu/~hastie/local.ftp/Springer/OLD//ESLII\\_print4.pdf](http://www.stanford.edu/~hastie/local.ftp/Springer/OLD//ESLII_print4.pdf)

**See Also**

For more information see: <http://docs.0xdata.com>

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Run regression GBM on australia.hex data
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
independent <- c("premax", "salmax", "minairtemp", "maxairtemp", "maxsst",
  "maxsoilmoist", "Max_czcs")
dependent <- "runoffnew"
h2o.gbm(y = dependent, x = independent, data = australia.hex, n.trees = 3, interaction.depth = 3,
  n.minobsinnode = 2, shrinkage = 0.2, distribution= "gaussian")

## Not run:
# Run multinomial classification GBM on australia data
h2o.gbm(y = dependent, x = independent, data = australia.hex, n.trees = 3, interaction.depth = 3,
  n.minobsinnode = 2, shrinkage = 0.01, distribution= "multinomial")

## End(Not run)
```

---

h2o.getFrame	<i>Get Reference to H2O Data Set</i>
--------------	--------------------------------------

---

**Description**

Get a reference to an existing H2O data set.

**Usage**

```
h2o.getFrame(h2o, key)
```

**Arguments**

h2o	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
key	A string indicating the unique hex key of the data set to retrieve.

**Value**

Returns an object of class [H2OParsedData](#).

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
h2o.ls(localH2O)
iris.hex = h2o.getFrame(localH2O, "iris.hex")
h2o.shutdown(localH2O)
```

---

h2o.getGLMLambdaModel	<i>Get H2O GLM Model for Specific Lambda</i>
-----------------------	--

---

**Description**

Retrieve the H2O GLM model built using a specific value of lambda from a lambda search.

**Usage**

```
h2o.getGLMLambdaModel(model, lambda)
```

**Arguments**

model	An <a href="#">H2OGLMModel</a> object generated by <code>h2o.glm</code> with lambda search enabled.
lambda	The specific value of lambda for which model to retrieve. If that lambda was not include in or saved during the search, the method throws an error.

**Value**

Returns an object of class [H2OGLMModel](#).

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.srch = h2o.glm(x = 3:9, y = 2, data = prostate.hex, family = "binomial",
  nlambda = 3, lambda_search = TRUE, nfolds = 0)
random_lambda = sample(prostate.srch@model$params$lambda_all, 1)
random_model = h2o.getGLMLambdaModel(prostate.srch, random_lambda)
```

---

h2o.getLogPath	<i>Get Path Where H2O R Logs are Saved</i>
----------------	--

---

**Description**

Get the file path where H2O R command and error response logs are currently being saved.

**Usage**

```
h2o.getLogPath(type)
```

**Arguments**

type	Which log file's path to get. Either "Command" for POST commands sent between R and H2O, or "Error" for errors returned by H2O in the HTTP response.
------	--

**See Also**

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.clearLogs](#), [h2o.openLog](#), [h2o.setLogPath](#)

**Examples**

```
library(h2o)
h2o.getLogPath("Command")
h2o.getLogPath("Error")
```

---

h2o.getModel	<i>Get Reference to H2O Model</i>
--------------	-----------------------------------

---

**Description**

Get a reference to an existing H2O model.

**Usage**

```
h2o.getModel(h2o, key)
```

**Arguments**

h2o	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
key	A string indicating the unique hex key of the model to retrieve.

**Value**

Returns an object that is a subclass of [H2OModel](#).

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

iris.hex <- as.h2o(localH2O, iris, "iris.hex")
model <- h2o.randomForest(x = 1:4, y = 5, data = iris.hex)
model.retrieved <- h2o.getModel(localH2O, model@key)
h2o.shutdown(localH2O)
```

---

h2o.glm	<i>H2O: Generalized Linear Models</i>
---------	---------------------------------------

---

**Description**

Fit a generalized linear model, specified by a response variable, a set of predictors, and a description of the error distribution.

**Usage**

```
h2o.glm(x, y, data, key = "", family, link, nfolds = 0, alpha = 0.5, nlambda = -1,
  lambda.min.ratio = -1, lambda = 1e-5, epsilon = 1e-4, standardize = TRUE,
  prior, variable_importances = 1, use_all_factor_levels = 0, tweedie.p =
  ifelse(family == 'tweedie', 1.5, as.numeric(NA)), iter.max = 100,
  higher_accuracy = FALSE, lambda_search = FALSE, return_all_lambda = FALSE,
  max_predictors = -1)
```

**Arguments**

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
family	A description of the error distribution and corresponding link function to be used in the model. Currently, Gaussian, binomial, Poisson, gamma, and Tweedie are supported. When a model is specified as Tweedie, users must also specify the appropriate Tweedie power.
link	(Optional) The link function relates the linear predictor to the distribution function. Default is the canonical link for the specified family. The full list of supported links:  gaussian: identity, log, inverse binomial: logit, log poisson: log, identity gamma: inverse, log, identity tweedie: tweedie
nfolds	(Optional) Number of folds for cross-validation.
alpha	(Optional) The elastic-net mixing parameter, which must be in [0,1]. The penalty is defined to be  $P(\alpha, \beta) = (1 - \alpha)/2 \ \beta\ _2^2 + \alpha \ \beta\ _1 = \sum_j [(1 - \alpha)/2\beta_j^2 + \alpha \beta_j ]$ so alpha=1 is the lasso penalty, while alpha=0 is the ridge penalty.
nlambda	The number of lambda values when performing a search.
lambda.min.ratio	Smallest value for lambda as a fraction of lambda.max, the entry value, which is the smallest value for which all coefficients in the model are zero.
lambda	The shrinkage parameter, which multiplies $P(\alpha, \beta)$ in the objective. The larger lambda is, the more the coefficients are shrunk toward zero (and each other).
epsilon	(Optional) Number indicating the cutoff for determining if a coefficient is zero.
standardize	(Optional) Logical value indicating whether the data should be standardized (set to mean = 0, variance = 1) before running GLM.
prior	(Optional) Prior probability of class 1. Only used if family = "binomial". When omitted, prior will default to the frequency of class 1 in the response column.
variable_importances	(Optional) A numeric value either 0 or 1 to indicate whether the variable importances should be computed. Compute variable importances for input features. NOTE: If use_all_factor_levels is off the importance of the base level will NOT be shown.

<code>use_all_factor_levels</code>	(Optional) A numeric value either 0 or 1 to indicate whether all factor levels should be used. By default, first factor level is skipped from the possible set of predictors. Set this flag if you want use all of the levels. Needs sufficient regularization to solve!
<code>tweedie.p</code>	(Optional) The index of the power variance function for the tweedie distribution. Only used if <code>family = "tweedie"</code> .
<code>iter.max</code>	(Optional) Maximum number of iterations allowed.
<code>higher_accuracy</code>	(Optional) A logical value indicating whether to use line search. This will cause the algorithm to run slower, so generally, it should only be set to TRUE if GLM does not converge otherwise.
<code>lambda_search</code>	(Optional) A logical value indicating whether to conduct a search over the space of lambda values, starting from <code>lambda_max</code> . When this is set to TRUE, lambda will be interpreted as <code>lambda_min</code> .
<code>return_all_lambda</code>	(Optional) A logical value indicating whether to return every model built during the lambda search. Only used if <code>lambda_search = TRUE</code> . If <code>return_all_lambda = FALSE</code> , then only the model corresponding to the optimal lambda will be returned.
<code>max_predictors</code>	(Optional) When <code>lambda_search = TRUE</code> , the algorithm will stop training if the number of predictors exceeds this value. Ignored when <code>lambda_search = FALSE</code> or <code>max_predictors = -1</code> .

### Value

An object of class `H2OGLMModel` with slots `key`, `data`, `model` and `xval`. The slot `model` is a list of the following components:

<code>coefficients</code>	A named vector of the coefficients estimated in the model.
<code>rank</code>	The numeric rank of the fitted linear model.
<code>family</code>	The family of the error distribution.
<code>deviance</code>	The deviance of the fitted model.
<code>aic</code>	Akaike's Information Criterion for the final computed model.
<code>null.deviance</code>	The deviance for the null model.
<code>iter</code>	Number of algorithm iterations to compute the model.
<code>df.residual</code>	The residual degrees of freedom.
<code>df.null</code>	The residual degrees of freedom for the null model.
<code>y</code>	The response variable in the model.
<code>x</code>	A vector of the predictor variable(s) in the model.
<code>auc</code>	Area under the curve.
<code>training.err</code>	Average training error.
<code>threshold</code>	Best threshold.
<code>confusion</code>	Confusion matrix.

The slot `xval` is a list of `H2OGLMModel` objects representing the cross-validation models. (Each of these objects themselves has `xval` equal to an empty list).

**See Also**

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importHDFS](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostatePath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prostatePath, key = "prostate.hex")
h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"), data = prostate.hex,
        family = "binomial", nfolds = 0, alpha = 0.5, lambda_search = FALSE,
        use_all_factor_levels = 0, variable_importances = 0, higher_accuracy = FALSE)

# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
myX = setdiff(colnames(prostate.hex), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.glm(y = "VOL", x = myX, data = prostate.hex, family = "gaussian", nfolds = 0, alpha = 0.1,
        lambda_search = FALSE, use_all_factor_levels = 0, variable_importances = 0,
        higher_accuracy = FALSE)
```

---

h2o.hitRatio

---

*Compute Hit Ratio from H2O Classification Predictions*


---

**Description**

Compute the hit ratios from a prediction dataset and a column of actual (reference) responses in H2O. The hit ratio is the percentage of instances where the actual class of an observation is in the top k classes predicted by the model, where k is specified by the user. Note that the hit ratio can only be calculated for classification models.

**Usage**

```
h2o.hitRatio(prediction, reference, k = 10, seed = 0)
```

**Arguments**

prediction	An <a href="#">H2OParsedData</a> object that represents the predicted response values. Must have the same number of rows as reference.
reference	An <a href="#">H2OParsedData</a> object that represents the actual response values. (Must be a single column).
k	A positive integer indicating the maximum number of labels to use for hit ratio computation. Cannot be larger than the size of the response domain.
seed	(Optional) Random number seed for breaking ties between equal probabilities.

**Value**

Returns a numeric vector with the hit ratio for every level in the reference domain.

**See Also**[H2OParsedData](#)**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
iris.gbm = h2o.gbm(x = 1:4, y = 5, data = iris.hex)
iris.pred = h2o.predict(iris.gbm)
h2o.hitRatio(iris.pred, iris.hex[,5], k = 3)
```

---

h2o.ignoreColumns	<i>Returns columns' names of a parsed H2O data object that are recommended to be ignored in an analysis</i>
-------------------	---

---

**Description**

Returns columns' names of a parsed [H2OParsedData](#) object if the columns have high counts of NA entries, thresh

**Usage**

```
h2o.ignoreColumns(data, max_na = 0.2)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object.
max_na	A numeric between 0 and 1 representing the proportion of NAs in a column.

**Value**

Returns a vector of column names.

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
airlinesURL = "https://s3.amazonaws.com/h2o-airlines-unpacked/allyears2k.csv"
airlines.hex = h2o.importFile(localH2O, path = airlinesURL, key = "airlines.hex")
h2o.ignoreColumns(airlines.hex)

## End(Not run)
```

h2o.importFile

*Import Local Data File***Description**

Imports a file from the local path and parses it, returning an object containing the identifying hex key.

**Usage**

```
h2o.importFile(object, path, key = "", parse = TRUE, header, sep = "", col.names)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The path of the file to be imported. Each row of data appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> , the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.

**Details**

**WARNING:** In H2O, import is lazy! Do not modify the data on hard disk until after parsing is complete.

**Value**

If `parse = TRUE`, the function returns an object of class [H2OParsedData](#). Otherwise, when `parse = FALSE`, it returns an object of class [H2ORawData](#).

**See Also**

[h2o.importFolder](#), [h2o.importHDFS](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
class(iris.hex)
summary(iris.hex)
```

---

h2o.importFolder	<i>Import Local Directory of Data Files</i>
------------------	---

---

**Description**

Imports all the files in the local directory and parses them, concatenating the data into a single H2O data matrix and returning an object containing the identifying hex key.

**Usage**

```
h2o.importFolder(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The path of the folder directory to be imported. Each row of data appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) An <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.

**Details**

This method imports all the data files in a given folder and concatenates them together row-wise into a single matrix represented by a [H2OParsedData](#) object. The data files must all have the same number of columns, and the columns must be lined up in the same order, otherwise an error will be returned.

WARNING: In H2O, import is lazy! Do not modify the data files on hard disk until after parsing is complete.

**Value**

If `parse = TRUE`, the function returns an object of class [H2OParsedData](#). Otherwise, when `parse = FALSE`, it returns an object of class [H2ORawData](#).

**See Also**

[h2o.importFile](#), [h2o.importHDFS](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
myPath = system.file("extdata", "prostate_folder", package = "h2o")
prostate_all.hex = h2o.importFolder(localH2O, path = myPath)
class(prostate_all.hex)
summary(prostate_all.hex)

## End(Not run)
```

---

h2o.importHDFS

*Import from HDFS*

---

**Description**

Imports a HDFS file or set of files in a directory and parses them, returning a object containing the identifying hex key.

**Usage**

```
h2o.importHDFS(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The path of the file or folder directory to be imported. If it does not contain an absolute path, the file name is relative to the current working directory.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.

**Details**

When path is a directory, this method acts like [h2o.importFolder](#) and concatenates all data files in the folder into a single ValueArray object.

WARNING: In H2O, import is lazy! Do not modify the data files on hard disk until after parsing is complete.

**Value**

If parse = TRUE, the function returns an object of class [H2OParsedData](#). Otherwise, when parse = FALSE, it returns an object of class [H2ORawData](#).

**See Also**

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
## Not run:
# This is an example of how to import files from HDFS.
# The user must modify the path to his or her specific HDFS path for this example to run.
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
iris.hex = h2o.importHDFS(localH2O, path = paste("hdfs://192.168.1.161",
  "datasets/runit/iris_wheader.csv", sep = "/"), parse = TRUE)
class(iris.hex)
summary(iris.hex)
iris.fv = h2o.importHDFS(localH2O, path = paste("hdfs://192.168.1.161",
```

```

"datasets/runit/iris_wheader.csv", sep = "/"), parse = TRUE, version = 2)
class(iris.fv)

iris_folder.hex = h2o.importHDFS(localH2O, path = paste("hdfs://192.168.1.161",
"datasets/runit/iris_test_train", sep = "/"))
summary(iris_folder.hex)

## End(Not run)

```

---

h2o.importURL

*Import Data from URL*


---

### Description

Imports a file from the URL and parses it, returning an object containing the identifying hex key.

### Usage

```
h2o.importURL(object, path, key = "", parse = TRUE, header, sep = "", col.names)
```

### Arguments

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The complete URL of the file to be imported. Each row of data appears as one line of the file.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.

### Details

**WARNING:** In H2O, import is lazy! Do not modify the data on hard disk until after parsing is complete.

### Value

If parse = TRUE, the function returns an object of class [H2OParsedData](#). Otherwise, when parse = FALSE, it returns an object of class [H2ORawData](#).

**See Also**

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importHDFS](#), [h2o.uploadFile](#)

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.githubusercontent.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
class(prostate.hex)
summary(prostate.hex)

## End(Not run)
```

---

h2o.init

---

*Connect to H2O and Install R Package*


---

**Description**

Connects to a running H2O instance and checks the local H2O R package is the correct version (i.e. that the version of the R package and the version of H2O are the same).

**Usage**

```
h2o.init(ip = "127.0.0.1", port = 54321, startH2O = TRUE, forceDL = FALSE, Xmx,
  beta = FALSE, assertion = TRUE, license = NULL,
  max_mem_size = "1g", min_mem_size = "1g")
```

**Arguments**

ip	Object of class "character" representing the IP address of the server where H2O is running.
port	Object of class "numeric" representing the port number of the H2O server.
startH2O	(Optional) A logical value indicating whether to start the H2O launcher GUI if no connection with H2O is detected. This is only possible if ip = "localhost" or ip = "127.0.0.1".
forceDL	(Optional) A logical value indicating whether to force download of the H2O executable. Defaults to FALSE, so the executable will only be downloaded if it does not already exist in the h2o R library resources directory h2o/java/h2o.jar.
Xmx	DEPRECATED A string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This 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.
beta	(Optional) A logical value indicating whether H2O should be launch in beta mode.

assertion	(Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes.
license	(Optional) A string value specifying the full path of the license file.
max_mem_size	(Optional) A string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must be a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes.
min_mem_size	(Optional) A string specifying the minimum size, in bytes, of the memory allocation pool to H2O. This value must be a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes.

### Details

This method first checks if H2O is connectible. If it cannot connect and `startH2O = TRUE` with IP of localhost, it will attempt to start an instance of H2O with IP = localhost, port = 54321. Otherwise, it stops immediately with an error.

When initializing H2O locally, this method searches for `h2o.jar` in the R library resources (`system.file("java", "h2o.jar"` and if the file does not exist, it will automatically attempt to download the correct version from Amazon S3. The user must have Internet access for this process to be successful.

Once connected, the method checks to see if the local H2O R package version matches the version of H2O running on the server. If there is a mismatch and the user indicates she wishes to upgrade, it will remove the local H2O R package and download/install the H2O R package from the server.

### Value

Once the package is successfully installed, this method will load it and return a `H2OClient` object containing the IP address and port number of the H2O server. See the [H2O R package documentation](#) for more details, or type `??h2o` in the R console.

### Note

Users may wish to manually upgrade their package (rather than waiting until being prompted), which requires that they fully uninstall and reinstall the H2O package, and the H2O client package. You must unload packages running in the environment before upgrading. It's recommended that users restart R or R studio after upgrading.

### See Also

[h2o.shutdown](#)

### Examples

```
# Try to create a localhost connection to H2O.
localH2O = h2o.init()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = FALSE)
```

h2o.kmeans

*H2O: K-Means Clustering***Description**

Performs k-means clustering on a data set.

**Usage**

```
h2o.kmeans(data, centers, cols = "", key = "", iter.max = 10,
           normalize = FALSE, init = "none", seed = 0, dropNACols = FALSE)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
centers	The number of clusters k.
cols	(Optional) A vector containing the names of the data columns on which k-means runs. If blank, k-means clustering will be run on the entire data set.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
iter.max	(Optional) The maximum number of iterations allowed.
normalize	(Optional) A logical value indicating whether the data should be normalized before running k-means.
init	(Optional) Method by which to select the k initial cluster centroids. Possible values are "none" for random initialization, "plusplus" for k-means++ initialization, and "furthest" for initialization at the furthest point from each successive centroid. See the <a href="#">H2O K-means documentation</a> for more details.
seed	(Optional) Random seed used to initialize the cluster centroids.
dropNACols	(Optional) A logical value indicating whether to drop columns with more than 10% entries that are NA.

**Value**

An object of class [H2OKMeansModel](#) with slots key, data, and model, where the last is a list of the following components:

centers	A matrix of cluster centers.
cluster	A <a href="#">H2OParsedData</a> object containing the vector of integers (from 1 to k), which indicate the cluster to which each point is allocated.
size	The number of points in each cluster.
withiness	Vector of within-cluster sum of squares, with one component per cluster.
tot.withiness	Total within-cluster sum of squares, i.e., sum(withiness).

**See Also**

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importHDFS](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
h2o.kmeans(data = prostate.hex, centers = 10, cols = c("AGE", "RACE", "VOL", "GLEASON"))
```

---

h2o.loadModel

*Load a H2O Model.*

---

**Description**

Load a H2OModel object from disk that was saved using h2o.saveModel.

**Usage**

```
h2o.loadModel(object, path = "")
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The path of the H2O model file to be imported.

**Value**

Returns a [H2OModel](#) object of the class corresponding to the type of model built. Ex: A saved model built using GLM will return a [H2OGLMModel](#) object.

**See Also**

[h2o.saveModel](#), [H2OModel](#)

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
  data = prostate.hex, family = "binomial", nfolds = 10, alpha = 0.5)
glmmodel.path = h2o.saveModel(object = prostate.glm, dir = "/Users/UserName/Desktop")
glmmodel.load = h2o.loadModel(localH2O, glmmodel.path)

## End(Not run)
```

---

h2o.logAndEcho	<i>Write and Echo Message to H2O Log</i>
----------------	--

---

**Description**

Write a user-defined message to the H2O Java log file and echo it back to the user.

**Usage**

```
h2o.logAndEcho(conn, message)
```

**Arguments**

conn	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
message	A character string to write to the H2O Java log file.

**See Also**

[H2OClient](#), [h2o.downloadAllLogs](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.logAndEcho(localH2O, "Test log and echo method.")
```

---

h2o.ls	<i>Obtain a list of H2O keys from the running instance of H2O</i>
--------	---

---

**Description**

Allows users to access a list of object keys in the running instance of H2O

**Usage**

```
h2o.ls(object, pattern)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port number of the H2O server.
pattern	A string indicating the type of key to be returned. When pattern is left is unspecified all keys are returned.

**Value**

Returns a list of hex keys in the current instance of H2O, and their associated sizes in bytes.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
s = runif(nrow(prostate.hex))
prostate.train = prostate.hex[s <= 0.8,]
prostate.train = h2o.assign(prostate.train, "prostate.train")
h2o.ls(localH2O)
```

---

h2o.month

*Convert Milliseconds to Months in H2O Dataset*

---

**Description**

Converts the entries of a [H2OParsedData](#) object from milliseconds to months (on a 0 to 11 scale).

**Usage**

```
h2o.month(x)
```

```
## S3 method for class 'H2OParsedData'
month(x)
```

**Arguments**

x                    An [H2OParsedData](#) object.

**Details**

This method calls the functions of the `MutableDateTime` class in Java.

**Value**

A [H2OParsedData](#) object containing the entries of x converted to months of the year.

**See Also**

[h2o.year](#)

---

h2o.mse	<i>Returns the mean squared error from H2O Classification Predictions</i>
---------	---

---

## Description

Returns the mean squared error calculated from a column of predicted responses and a column of actual (reference) responses in H2O.

## Usage

```
h2o.mse(data, reference)
```

## Arguments

data	An <a href="#">H2OParsedData</a> object that represents the predicted response values. (Must be a single column).
reference	An <a href="#">H2OParsedData</a> object that represents the actual response values. Must have the same dimensions as data.

## Value

Returns the mean squared error as a continuous real numeric.

## See Also

[H2OParsedData](#)

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.glm = prostate.glm = h2o.glm(x = c("RACE", "PSA", "DCAPS"), y = "AGE",
  data = prostate.hex, family = "gaussian", nfolds = 10, alpha = 0.5)
prostate.pred = h2o.predict(prostate.glm)
h2o.mse(prostate.pred[,1], prostate.hex[,2])
```

h2o.naiveBayes

*H2O: Naive Bayes Classifier***Description**

Builds gradient boosted classification trees, and gradient boosted regression trees on a parsed data set.

**Usage**

```
h2o.naiveBayes(x, y, data, key = "", laplace = 0, dropNACols = FALSE)
```

**Arguments**

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An <a href="#">H2OParsedData</a> (version = 2) object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
laplace	(Optional) A positive number controlling Laplace smoothing. The default (0) disables Laplace smoothing.
dropNACols	(Optional) A logical value indicating whether to drop predictor columns with $\geq 20\%$ NAs.

**Details**

The naive Bayes classifier assumes independence between predictor variables conditional on the response, and a Gaussian distribution of numeric predictors with mean and standard deviation computed from the training dataset.

When building a naive Bayes classifier, every row in the training dataset that contains at least one NA will be skipped completely. If the test dataset has missing values, then those predictors are omitted in the probability calculation during prediction.

**Value**

An object of class [H2ONBModel](#) with slots key, data, and model, where the last is a list of the following components:

laplace	A positive number controlling Laplace smoothing. The default (0) disables Laplace smoothing.
levels	Categorical levels of the dependent variable.
apriori	Total occurrences of each level of the dependent variable.
apriori_prob	A-priori class distribution for the dependent variable.
tables	A list of tables, one for each predictor variable. For categorical predictors, the table displays, for each attribute level, the conditional probabilities given the target class. For numeric predictors, the table gives, for each target class, the mean and standard deviation of the variable.

**See Also**

For more information see: <http://docs.0xdata.com>

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Build naive Bayes classifier with categorical predictors
votesPath = system.file("extdata", "housevotes.csv", package="h2o")
votes.hex = h2o.importFile(localH2O, path = votesPath, header = TRUE)
summary(votes.hex)
h2o.naiveBayes(y = 1, x = 2:17, data = votes.hex, laplace = 3)

# Build naive Bayes classifier with numeric predictors
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.naiveBayes(y = 5, x = 1:4, data = iris.hex)
```

---

h2o.openLog

*View H2O R Logs*

---

**Description**

Open existing logs of H2O R POST commands and error responses on local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.openLog(type)
```

**Arguments**

type	Which log file to open. Either "Command" for POST commands sent between R and H2O, or "Error" for errors returned by H2O in the HTTP response.
------	--

**See Also**

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.clearLogs](#), [h2o.getLogPath](#), [h2o.setLogPath](#)

**Examples**

```
## Not run:
# Skip running this to avoid windows being opened during R CMD check
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
```

```
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()

h2o.openLog("Command")
h2o.openLog("Error")

## End(Not run)
```

---

`h2o.parseRaw`*Parse Raw Data File*

---

## Description

Parses a raw data file, returning an object containing the identifying hex key.

## Usage

```
h2o.parseRaw(data, key = "", header, sep = "", col.names)
```

## Arguments

<code>data</code>	An <a href="#">H2ORawData</a> object to be parsed.
<code>key</code>	(Optional) The hex key assigned to the parsed file.
<code>header</code>	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
<code>sep</code>	(Optional) The field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> , the parser will automatically detect the separator.
<code>col.names</code>	(Optional) A <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.

## Details

After the raw data file is parsed, it will be automatically deleted from the H2O server.

## Value

An object of class [H2OParsedData](#), representing the dataset that was parsed.

## See Also

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importHDFS](#), [h2o.importURL](#), [h2o.uploadFile](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.raw = h2o.importFile(localH2O, path = prosPath, parse = FALSE)
# Do not modify prostate.csv on disk at this point!
prostate.hex = h2o.parseRaw(data = prostate.raw, key = "prostate.hex")
# After parsing, it is okay to modify or delete prostate.csv
```

h2o.pcr

*H2O: Principal Components Regression***Description**

Runs GLM regression on PCA results, and allows for transformation of test data to match PCA transformations of training data.

**Usage**

```
h2o.pcr(x, y, data, key = "", ncomp, family, nfolds = 10, alpha = 0.5, lambda = 1e-05,
        epsilon = 1e-05, tweedie.p)
```

**Arguments**

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
ncomp	A number indicating the number of principal components to use in the regression model.
family	A description of the error distribution and corresponding link function to be used in the model. Currently, Gaussian, binomial, Poisson, gamma, and Tweedie are supported.
nfolds	(Optional) Number of folds for cross-validation. The default is 10.
alpha	(Optional) The elastic-net mixing parameter, which must be in [0,1]. The penalty is defined to be $P(\alpha, \beta) = (1 - \alpha)/2 \ \beta\ _2^2 + \alpha \ \beta\ _1 = \sum_j [(1 - \alpha)/2 \beta_j^2 + \alpha  \beta_j ]$ so alpha=1 is the lasso penalty, while alpha=0 is the ridge penalty.
lambda	(Optional) The shrinkage parameter, which multiplies $P(\alpha, \beta)$ in the objective. The larger lambda is, the more the coefficients are shrunk toward zero (and each other).
epsilon	(Optional) Number indicating the cutoff for determining if a coefficient is zero.
tweedie.p	The index of the power variance function for the tweedie distribution. Only used if family = "tweedie"

**Details**

This method standardizes the data, obtains the first `ncomp` principal components using PCA (in decreasing order of standard deviation), and then runs GLM with the components as the predictor variables.

**Value**

An object of class `H2OGLMModel` with slots `key`, `data`, `model` and `xval`. The slot `model` is a list of the following components:

<code>coefficients</code>	A named vector of the coefficients estimated in the model.
<code>rank</code>	The numeric rank of the fitted linear model.
<code>family</code>	The family of the error distribution.
<code>deviance</code>	The deviance of the fitted model.
<code>aic</code>	Akaike's Information Criterion for the final computed model.
<code>null.deviance</code>	The deviance for the null model.
<code>iter</code>	Number of algorithm iterations to compute the model.
<code>df.residual</code>	The residual degrees of freedom.
<code>df.null</code>	The residual degrees of freedom for the null model.
<code>y</code>	The response variable in the model.
<code>x</code>	A vector of the predictor variable(s) in the model.
<code>auc</code>	Area under the curve.
<code>training.err</code>	Average training error.
<code>threshold</code>	Best threshold.
<code>confusion</code>	Confusion matrix.

The slot `xval` is a list of `H2OGLMModel` objects representing the cross-validation models. (Each of these objects themselves has `xval` equal to an empty list).

**See Also**

[h2o.prcomp](#), [h2o.glm](#)

**Examples**

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Run PCR on Prostate Data
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.githubusercontent.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
h2o.pcr(x = c("AGE", "RACE", "PSA", "DCAPS"), y = "CAPSULE", data = prostate.hex, family = "binomial",
  nolds = 0, alpha = 0.5, ncomp = 2)

## End(Not run)
```

---

h2o.performance	<i>Performance Measures</i>
-----------------	-----------------------------

---

### Description

Evaluate the predictive performance of a model via various measures.

### Usage

```
h2o.performance(data, reference, measure = "accuracy", thresholds)
```

### Arguments

data	An <a href="#">H2OParsedData</a> object containing the predicted outcome scores. Must be a single column with the same number of rows as reference.
reference	An <a href="#">H2OParsedData</a> object containing the actual outcomes for comparison. Must be a single binary column with all entries in {0,1}.
measure	A character string indicating the performance measure to optimize. Must be one of the following: <ul style="list-style-type: none"> <li>• F1: F1 score, equal to <math>2 * (Precision * Recall) / (Precision + Recall)</math></li> <li>• accuracy: Accuracy of model, estimated as <math>(TP + TN) / (P + N)</math>.</li> <li>• precision: Precision of model, estimated as <math>TP / (TP + FP)</math>.</li> <li>• recall: Recall of model, i.e. the true positive rate <math>TP / P</math>.</li> <li>• specificity: Specificity of model, i.e. the true negative rate <math>TN / N</math>.</li> <li>• max_per_class_error: Maximum per class error in model.</li> </ul>
thresholds	(Optional) A numeric vector from 0 to 1 indicating the threshold values at which to compute the performance measure. If missing, the range will be automatically generated. TODO: Still not sure I understand what exactly these thresholds are, is it the FPR or something else?

### Value

An object of class [H2OPerfModel](#) with slots cutoffs, measure, perf (the performance measure selected), roc (data frame used to plot ROC) and model, where the last is a list of the following components:

auc	Area under the curve.
gini	Gini coefficient.
best_cutoff	Threshold value that optimizes the performance measure.
F1	F1 score at best cutoff.
accuracy	Accuracy value at best cutoff.
precision	Precision value at best cutoff.
recall	Recall value at best cutoff.

specificity      Specificity value at best cutoff.  
 max\_per\_class\_err      Maximum per class error at best cutoff.  
 confusion      Confusion matrix at best cutoff.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Run GBM classification on prostate.csv
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
prostate.gbm = h2o.gbm(y = 2, x = 3:9, data = prostate.hex)

# Calculate performance measures at threshold that maximizes precision
prostate.pred = h2o.predict(prostate.gbm)
head(prostate.pred)
h2o.performance(prostate.pred[,3], prostate.hex$CAPSULE, measure = "precision")
```

---

h2o.prcomp

*Principal Components Analysis*

---

### Description

Performs principal components analysis on the given data set.

### Usage

```
h2o.prcomp(data, tol = 0, cols = "", key = "", standardize = TRUE, retx = FALSE)
```

### Arguments

data	An <a href="#">H2OParsedData</a> object on which to run principal components analysis.
tol	(Optional) A value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to tol times the standard deviation of the first component.) With the default setting tol = 0, no components are omitted.
cols	(Optional) A vector of column names or indices indicating the features to perform PCA on. By default, all columns in the dataset are analyzed.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
standardize	(Optional) A logical value indicating whether the variables should be shifted to be zero centered and scaled to have unit variance before the analysis takes place.
retx	(Optional) A logical value indicating whether the rotated variables should be returned.

**Details**

The calculation is done by a singular value decomposition of the (possibly standardized) data set.

**Value**

An object of class [H2OPCAModel](#) with slots key, data, and model, where the last is a list of the following components:

standardized	A logical value indicating whether the data was centered and scaled.
sdev	The standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix).
rotation	The matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).

**Note**

The signs of the columns of the rotation matrix are arbitrary, and so may differ between different programs for PCA.

**See Also**

[h2o.pcr](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
australia.pca = h2o.prcomp(data = australia.hex, standardize = TRUE)
print(australia.pca)
```

---

h2o.predict

*H2O Model Predictions*

---

**Description**

Obtains predictions from various fitted H2O model objects.

**Usage**

```
h2o.predict(object, newdata)
```

**Arguments**

object	A fitted <a href="#">H2OModel</a> object for which prediction is desired.
newdata	(Optional) A <a href="#">H2OParsedData</a> object in which to look for variables with which to predict. If omitted, the data used to fit the model <code>object@data</code> are used.

**Details**

This method dispatches on the type of H2O model to select the correct prediction/scoring algorithm.

**Value**

A [H2OParsedData](#) object containing the predictions.

**See Also**

[h2o.glm](#), [h2o.kmeans](#), [h2o.randomForest](#), [h2o.prcomp](#), [h2o.gbm](#), [h2o.deeplearning](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostatePath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prostatePath, key = "prostate.hex")
prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"), data = prostate.hex,
  family = "binomial", nfolds = 0, alpha = 0.5)
# Get fitted values of prostate dataset
prostate.fit = h2o.predict(object = prostate.glm, newdata = prostate.hex)
summary(prostate.fit)
```

---

h2o.randomForest

*H2O: Random Forest*

---

**Description**

Performs random forest classification on a data set.

**Usage**

```
h2o.randomForest(x, y, data, key = "", classification = TRUE, ntree = 50,
  depth = 20, mtries = -1, sample.rate = 2/3, nbins = 100, seed = -1,
  importance = FALSE, nfolds = 0, validation, nodesize = 1,
  balance.classes = FALSE, max.after.balance.size = 5, doGrpSplit = TRUE,
  verbose = FALSE, oobee = TRUE, stat.type = "ENTROPY", type = "fast")
```

**Arguments**

- |      |   |
|------|---|
| x    | A vector containing the names or indices of the predictor variables to use in building the random forest model.   |
| y    | The name or index of the response variable. If the data does not contain a header, this is the column index, designated by increasing numbers from left to right. (The response must be either an integer or a categorical variable). |
| data | An <a href="#">H2OParsedData</a> object containing the variables in the model.  |

key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
classification	(Optional) A logical value indicating whether a classification model should be built (as opposed to regression).
ntree	(Optional) Number of trees to grow. (Must be a nonnegative integer).
depth	(Optional) Maximum depth to grow the tree.
mtries	(Optional) Number of variables randomly sampled as candidates at each split. If set to -1, defaults to sqrt(p) for classification, and p/3 for regression, where p is the number of predictors.
sample.rate	(Optional) Sampling rate for constructing data from which individual trees are grown.
nbins	(Optional) Build a histogram of this many bins, then split at best point.
seed	(Optional) Seed for building the random forest. If seed = -1, one will automatically be generated by H2O.
importance	(Optional) A logical value indicating whether to calculate variable importance. Set to FALSE to speed up computations.
nfolds	(Optional) Number of folds for cross-validation. If nfolds >= 2, then validation must remain empty.
validation	(Optional) An <a href="#">H2OParsedData</a> object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data when nfolds = 0.
nodesize	(Optional) Number of nodes to use for computation.
balance.classes	(Optional) Balance training data class counts via over/under-sampling (for imbalanced data)
max.after.balance.size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
doGrpSplit	Check non-contiguous group splits for categorical predictors
verbose	(Optional) A logical value indicating whether verbose results should be returned.
stat.type	(Optional) Type of statistic to use, equal to either "ENTROPY" or "GINI".
oobee	(Optional) A logical value indicating whether to calculate the out of bag error estimate.
type	(Optional) Default is "fast" mode, which builds trees in parallel and distributed, but requires all of the data to fit on a single node. Alternate mode is "BigData" mode, which builds a random forest layer-by-layer across your cluster and scales to any size data set.

### Value

An object of class [H2ODRFModel](#) with slots key, data, and model, where the last is a list of the following components:

ntree            Number of trees grown.

mse	Mean-squared error for each tree.
forest	A matrix giving the minimum, mean, and maximum of the tree depth and number of leaves.
confusion	Confusion matrix of the prediction.

### Examples

```
# Run an RF model on iris data
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
h2o.randomForest(y = 5, x = c(2,3,4), data = iris.hex, ntree = 50, depth = 100)
```

---

h2o.rm	<i>Removes H2O objects from the server where H2O is running.</i>
--------	--

---

### Description

Allows users to remove H2O objects from the server where the instance of H2O is running. This call acts on the H2O server through the R console, and does NOT remove the associated named object from the R environment.

### Usage

```
h2o.rm(object, keys)
```

### Arguments

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
keys	the hex key associated with the object to be removed.

### Note

Users may wish to remove an H2O object on the server that is associated with an object in the R environment. Recommended behavior is to also remove the object in the R environment. See the second example at the end of this section.

### See Also

[h2o.assign](#), [h2o.ls](#)

**Examples**

```
# Remove an H2O object from the server where H2O is running.
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")

# Remove an H2O object from the server and from the R environment
h2o.ls(localH2O)
h2o.rm(object = localH2O, keys = "prostate.hex")
remove(prostate.hex)
h2o.ls(localH2O)
```

---

h2o.runif	<i>Produces a vector of specified length contain random uniform numbers</i>
-----------	---

---

**Description**

Produces a vector of random uniform numbers.

**Usage**

```
h2o.runif(x, min = 0, max = 1, seed = -1)
```

**Arguments**

x	An <a href="#">H2OParsedData</a> object with number of rows equal to the number of elements the vector of random numbers should have.
min	An integer specifying the lower bound of the distribution.
max	An integer specifying the upper bound of the distribution.
seed	(Optional) Random seed used to generate draws from the uniform distribution. The default of -1 results in a seed equal to the current system time in milliseconds.

**Details**

x must be a [H2OParsedData](#) object so that H2O can generate random numbers aligned with the dataset for efficient large-scale sampling and filtering.

**Value**

A vector of random, uniformly distributed numbers. The elements are between 0 and 1 unless otherwise specified.

**Examples**

```

library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
s = h2o.runif(prostate.hex)
summary(s)

prostate.train = prostate.hex[s <= 0.8,]
prostate.train = h2o.assign(prostate.train, "prostate.train")
prostate.test = prostate.hex[s > 0.8,]
prostate.test = h2o.assign(prostate.test, "prostate.test")
nrow(prostate.train) + nrow(prostate.test)

```

---

h2o.saveModel	<i>Save a H2OModel object to disk.</i>
---------------	--

---

**Description**

Save a H2OModel object to a disk and can be loaded back into H2O using h2o.loadModel.

**Usage**

```
h2o.saveModel(object, dir="", name="", filename = "", save_cv = FALSE, force=FALSE)
```

**Arguments**

object	An <a href="#">H2OModel</a> object.
dir	Directory the model file will be written to.
name	Name of the file being saved.
filename	Full path of directory and name of file being saved. Will override dir and name parameters if also given.
save_cv	(Optional) If save_cv = TRUE all associated cross validation will be saved in the same base directory as the main model. If you don't save cross validation models, there will be warnings when loading the model.
force	(Optional) If force = TRUE any existing file will be overwritten. Otherwise if the file already exists the operation will fail.

**Value**

Returns path of model object saved.

**See Also**

[h2o.loadModel](#), [H2OModel](#)

## Examples

```
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prostate.hex = h2o.importFile(localH2O, path = paste("https://raw.githubusercontent.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
  data = prostate.hex, family = "binomial", nfolds = 10, alpha = 0.5)
h2o.saveModel(object = prostate.glm, dir = "/Users/UserName/Desktop", save_cv = TRUE, force = TRUE)

## End(Not run)
```

---

h2o.setLogPath

*Set Path Where H2O R Logs are Saved*

---

## Description

Set the file path where H2O R command and error response logs are currently being saved.

## Usage

```
h2o.setLogPath(path, type)
```

## Arguments

path	A character string indicating the new file path where logs should be saved.
type	Which log file's path to modify. Either "Command" for POST commands sent between R and H2O, or "Error" for errors returned by H2O in the HTTP response.

## See Also

[h2o.startLogging](#), [h2o.stopLogging](#), [h2o.clearLogs](#), [h2o.openLog](#), [h2o.getLogPath](#)

## Examples

```
library(h2o)
h2o.getLogPath("Command")
h2o.setLogPath(getwd(), "Command")
h2o.getLogPath("Command")
```

---

`h2o.shutdown`*Shutdown H2O server*

---

**Description**

Shuts down the specified H2O instance. All data on the server will be lost!

**Usage**

```
h2o.shutdown(client, prompt = TRUE)
```

**Arguments**

<code>client</code>	An <a href="#">H2OClient</a> client containing the IP address and port of the server running H2O.
<code>prompt</code>	(Optional) A logical value indicating whether to prompt the user before shutting down the H2O server.

**Details**

This method checks if H2O is running at the specified IP address and port, and if it is, shuts down that H2O instance. **WARNING:** All data, models, and other values stored on the server will be lost! Only call this function if you and all other clients connected to the H2O server are finished and have saved your work.

**Note**

Users must call `h2o.shutdown` explicitly in order to shut down the local H2O instance started by R. If R is closed before H2O, then an attempt will be made to automatically shut down H2O. This only applies to local instances started with `h2o.init`, not remote H2O servers.

**See Also**

[h2o.init](#)

**Examples**

```
# Don't run automatically to prevent accidentally shutting down a cloud
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.shutdown(localH2O)

## End(Not run)
```

h2o.SpeedRF

*H2O: Single-Node Random Forest***Description**

Performs single-node random forest classification on a data set.

**Usage**

```
h2o.SpeedRF(x, y, data, key = "", classification = TRUE, nfolds = 0, validation,
            mtries = -1, ntree = 50, depth = 20, sample.rate = 2/3, oobee = TRUE,
            importance = FALSE, nbins = 1024, seed = -1, stat.type = "ENTROPY",
            balance.classes = FALSE, verbose = FALSE)
```

**Arguments**

x	A vector containing the names or indices of the predictor variables to use in building the random forest model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index, designated by increasing numbers from left to right. (The response must be either an integer or a categorical variable).
data	An <a href="#">H2OParsedData</a> object containing the variables in the model.
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
classification	(Optional) A logical value indicating whether a classification model should be built (as opposed to regression).
nfolds	(Optional) Number of folds for cross-validation. If <code>nfolds &gt;= 2</code> , then validation must remain empty.
validation	(Optional) An <a href="#">H2OParsedData</a> object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data when <code>nfolds = 0</code> .
mtries	(Optional) Number of features to randomly select at each split in the tree. If set to the default of -1, this will be set to <code>sqrt(ncol(data))</code> , rounded down to the nearest integer.
ntree	(Optional) Number of trees to grow. (Must be a nonnegative integer).
depth	(Optional) Maximum depth to grow the tree.
sample.rate	(Optional) Sampling rate for constructing data from which individual trees are grown.
oobee	(Optional) A logical value indicating whether to calculate the out of bag error estimate.
importance	(Optional) A logical value indicating whether to compute variable importance measures. (If set to TRUE, the algorithm will take longer to finish.)
nbins	(Optional) Build a histogram of this many bins, then split at best point.

seed	(Optional) Seed for building the random forest. If seed = -1, one will automatically be generated by H2O.
stat.type	(Optional) Type of statistic to use, equal to either "ENTROPY" or "GINI".
balance.classes	(Optional) A logical value indicating whether classes should be rebalanced. Use for datasets where the levels of the response class are very unbalanced.
verbose	(Optional) A logical value indicating whether verbose results should be returned.

### Details

IMPORTANT: Currently, you must initialize H2O with the flag beta = TRUE in h2o.init in order to use this method!

This method runs random forest model building on a single node, as opposed to the multi-node implementation in [h2o.randomForest](#).

### Value

An object of class [H2OSpeedRFModel](#) with slots key, data, valid (the validation dataset), and model, where the last is a list of the following components:

params	Input parameters for building the model.
ntree	Number of trees grown.
depth	Depth of the trees grown.
nbins	Number of bins used in building the histogram.
classification	Logical value indicating if the model is classification.
mse	Mean-squared error for each tree.
confusion	Confusion matrix of the prediction.

### See Also

[H2OSpeedRFModel](#), [h2o.randomForest](#)

### Examples

```
# Currently still in beta, so don't automatically run example
## Not run:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, beta = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
h2o.SpeedRF(x = c(2,3,4), y = 5, data = iris.hex, ntree = 50, depth = 100)

## End(Not run)
```

---

h2o.splitFrame	<i>Split an H2O Data Set</i>
----------------	------------------------------

---

**Description**

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

**Usage**

```
h2o.splitFrame(data, ratios = 0.75, shuffle = FALSE)
```

**Arguments**

data	An <a href="#">H2OParsedData</a> object representing the dataset to split.
ratios	A numeric value or array indicating the ratio of total rows contained in each split.
shuffle	A logical value indicating whether to shuffle the rows before splitting.

**Value**

Returns a list of objects of class [H2OParsedData](#), each corresponding to one of the splits.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
iris.split = h2o.splitFrame(iris.hex, ratios = c(0.2, 0.5))
head(iris.split[[1]])
summary(iris.split[[1]])
```

---

h2o.startLogging	<i>Start Writing H2O R Logs</i>
------------------	---------------------------------

---

**Description**

Begin logging H2O R POST commands and error responses to local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.startLogging()
```

**See Also**

[h2o.stopLogging](#), [h2o.clearLogs](#), [h2o.openLog](#), [h2o.getLogPath](#), [h2o.setLogPath](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

---

h2o.stopLogging	<i>Stop Writing H2O R Logs</i>
-----------------	--------------------------------

---

**Description**

Halt logging of H2O R POST commands and error responses to local disk. Used primarily for debugging purposes.

**Usage**

```
h2o.stopLogging()
```

**See Also**

[h2o.startLogging](#), [h2o.clearLogs](#), [h2o.openLog](#), [h2o.getLogPath](#), [h2o.setLogPath](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

---

h2o.table	<i>Cross Tabulation of H2O Data</i>
-----------	-------------------------------------

---

**Description**

Uses the cross-classifying factors to build a table of counts at each combination of factor levels.

**Usage**

```
h2o.table(x, return.in.R = FALSE)
```

**Arguments**

`x` An [H2OParsedData](#) object with at most two integer or factor columns.  
`return.in.R` A logical value indicating whether the result should be converted into an R table.

**Value**

If `return.in.R = FALSE`, a [H2OParsedData](#) object containing the contingency table. This will just be the counts of each factor level when `x` has a single column. If `return.in.R = TRUE`, the H2O result will be pulled into R and converted into a table object.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
summary(prostate.hex)

# Counts of the ages of all patients
head(h2o.table(prostate.hex[,3]))
h2o.table(prostate.hex[,3], return.in.R = TRUE)

# Two-way table of ages (rows) and race (cols) of all patients
head(h2o.table(prostate.hex[,c(3,4)]))
h2o.table(prostate.hex[,c(3,4)], return.in.R = TRUE)
```

---

h2o.uploadFile	<i>Upload Local Data File</i>
----------------	-------------------------------

---

**Description**

Uploads a file from the local drive and parses it, returning an object containing the identifying hex key.

**Usage**

```
h2o.uploadFile(object, path, key = "", parse = TRUE, header,
  sep = "", col.names, silent = TRUE)
```

**Arguments**

object	An <a href="#">H2OClient</a> object containing the IP address and port of the server running H2O.
path	The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> , the parser will automatically detect the separator.
col.names	(Optional) A <a href="#">H2OParsedData</a> object containing a single delimited line with the column names for the file.
silent	(Optional) A logical value indicating whether or not to display an upload progress bar.

**Details**

WARNING: In H2O, import is lazy! Do not modify the data on hard disk until after parsing is complete.

**Value**

If `parse = TRUE`, the function returns an object of class [H2OParsedData](#). Otherwise, when `parse = FALSE`, it returns an object of class [H2ORawData](#).

**See Also**

[h2o.importFile](#), [h2o.importFolder](#), [h2o.importHDFS](#), [h2o.importURL](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.uploadFile(localH2O, path = prosPath, key = "prostate.hex")
class(prostate.hex)
summary(prostate.hex)
```

---

h2o.year	<i>Convert Milliseconds to Years in H2O Dataset</i>
----------	---

---

**Description**

Converts the entries of a [H2OParsedData](#) object from milliseconds to years, indexed starting from 1900.

**Usage**

```
h2o.year(x)
```

```
## S3 method for class 'H2OParsedData'
year(x)
```

**Arguments**

x                    An [H2OParsedData](#) object

**Details**

This method calls the functions of the `MutableDateTime` class in Java.

**Value**

A [H2OParsedData](#) object containing the entries of x converted to years starting from 1900, e.g. 69 corresponds to the year 1969.

**See Also**

[h2o.month](#)

---

H2OClient-class	<i>Class "H2OClient"</i>
-----------------	--------------------------

---

**Description**

An object representing the server/local machine on which H2O is running.

**Objects from the Class**

Objects can be created by calls of the form `new("H2OClient", ...)`

**Slots**

**ip:** Object of class "character" representing the IP address of the H2O server.

**port:** Object of class "numeric" representing the port number of the H2O server.

**Methods**

```

h2o.importFile signature(object = "H2OClient", path = "character", + key = "character", parse = "logical")
...
h2o.importFolder signature(object = "H2OClient", path = "character", + parse = "logical"):
...
h2o.importURL signature(object = "H2OClient", path = "character", + key = "character", parse = "logical")
...
show signature(object = "H2OClient"): ...

```

**Examples**

```
showClass("H2OClient")
```

---

```

H2ODeepLearningGrid-class
  Class "H2ODeepLearningGrid"

```

---

**Description**

Object representing the models built by a H2O Deep Learning neural networks grid search.

**Objects from the Class**

Objects can be created by calls of the form `new("H2ODeepLearningGrid", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.  
**data:** Object of class "H2OParsedData", which is the input data used to build the model.  
**model:** Object of class "list" containing "H2ODeepLearningModel" objects representing the models returned by the Deep Learning neural networks grid search.  
**sumtable:** Object of class "list" containing summary statistics of all the models returned by the Deep Learning neural networks grid search.

**Extends**

Class "[H2OGrid](#)", directly.

**Methods**

No methods defined with class "H2ODeepLearningGrid" in the signature.

**See Also**

[H2ODeepLearningModel](#), [h2o.deeplearning](#)

**Examples**

```
showClass("H2ODeepLearningGrid")
```

---

```
H2ODeepLearningModel-class  
  Class "H2ODeepLearningModel"
```

---

**Description**

A class for representing Deep Learning neural network models.

**Objects from the Class**

Objects can be created by calls of the form `new("H2ODeepLearningModel", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **confusion:** The confusion matrix of the response, with actual observations as rows and predicted values as columns.
- **train\_class\_err:** Classification error on the training dataset.
- **train\_sqr\_err:** Mean-squared error on the training dataset.
- **train\_cross\_entropy:** Cross-entropy on the training dataset.
- **valid\_class\_err:** Classification error on the validation dataset.
- **valid\_sqr\_err:** Mean-squared error on the validation dataset.
- **valid\_cross\_entropy:** Cross-entropy on the validation dataset.

**valid:** Object of class "H2OParsedData", representing the validation data set.

**xval:** List of objects of class "H2ODeepLearningModel", representing the n-fold cross-validation models.

**Extends**

Class "[H2OModel](#)", directly.

**Methods**

**show** signature(object = "H2ODeepLearningModel"): ...

**See Also**

[h2o.deeplearning](#)

**Examples**

```
showClass("H2ODeepLearningModel")
```

---

H2ODRFGrid-class	Class "H2ODRFGrid"
------------------	--------------------

---

### Description

Object representing the models built by a H2O distributed random forest grid search.

### Objects from the Class

Objects can be created by calls of the form `new("H2ODRFGrid", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2ODRFModel" objects representing the models returned by the distributed random forest grid search.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the distributed random forest grid search.

### Extends

Class "[H2OGrid](#)", directly.

### Methods

No methods defined with class "H2ODRFGrid" in the signature.

### See Also

[H2ODRFModel](#), [h2o.randomForest](#)

### Examples

```
showClass("H2ODRFGrid")
```

---

H2ODRFModel-class	Class "H2ODRFModel"
-------------------	---------------------

---

### Description

A class for representing random forest ensembles.

### Objects from the Class

Objects can be created by calls of the form `new("H2ODRFModel", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **type:** The type of the tree, which at this point must be classification.
- **ntree:** Number of trees grown.
- **oob\_err:** Out of bag error rate.
- **forest:** A matrix giving the minimum, mean, and maximum of the tree depth and number of leaves.
- **confusion:** Confusion matrix of the prediction.

**valid:** Object of class "H2OParsedData", which is the data used for validating the model.

**xval:** List of objects of class "H2ODRFModel", representing the n-fold cross-validation models.

### Extends

Class "[H2OModel](#)", directly.

### Methods

`show signature(object = "H2ODRFModel"): ...`

### See Also

[h2o.randomForest](#)

### Examples

```
showClass("H2ODRFModel")
```

---

H2OGBMGrid-class	Class "H2OGBMGrid"
------------------	--------------------

---

### Description

Object representing the models built by a H2O GBM grid search.

### Objects from the Class

Objects can be created by calls of the form `new("H2OGBMGrid", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2OGBMModel" objects representing the models returned by the GBM grid search.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the GBM grid search.

### Extends

Class "[H2OGrid](#)", directly.

### Methods

No methods defined with class "H2OGBMGrid" in the signature.

### See Also

[H2OGBMModel](#), [h2o.gbm](#)

### Examples

```
showClass("H2OGBMGrid")
```

---

H2OGBMModel-class	Class "H2OGBMModel"
-------------------	---------------------

---

### Description

A class for representing generalized boosted classification/regression models.

### Objects from the Class

Objects can be created by calls of the form `new("H2OGBMModel", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **err:** The mean-squared error in each tree.
- **cm:** (Only for classification). The confusion matrix of the response, with actual observations as rows and predicted values as columns.

**valid:** Object of class [H2OParsedData](#), which is the dataset used to validate the model.

**xval:** List of objects of class "H2OGBMModel", representing the n-fold cross-validation models.

### Extends

Class "[H2OModel](#)", directly.

### Methods

**show** signature(object = "H2OGBMModel"): ...

### See Also

[h2o.gbm](#)

### Examples

```
showClass("H2OGBMModel")
```

---

H2OGLMGrid-class	Class "H2OGLMGrid"
------------------	--------------------

---

### Description

Object representing the models built by a H2O GLM grid search.

### Objects from the Class

Objects can be created by calls of the form `new("H2OGLMGrid", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2OGLMModel" objects representing the models returned by the GLM grid search.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the GLM grid search.

### Extends

Class "[H2OGrid](#)", directly.

### Methods

No methods defined with class "H2OGLMGrid" in the signature.

### See Also

[H2OGLMModel](#), [h2o.glm](#)

### Examples

```
showClass("H2OGLMGrid")
```

---

H2OGLMMModel-class      *Class "H2OGLMMModel"*

---

### Description

A class for representing generalized linear models.

### Objects from the Class

Objects can be created by calls of the form `new("H2OGLMMModel", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **coefficients:** A named vector of the coefficients estimated in the model.
- **rank:** The numeric rank of the fitted linear model.
- **family:** The family of the error distribution.
- **deviance:** The deviance of the fitted model.
- **aic:** Akaike's Information Criterion for the final computed model.
- **null.deviance:** The deviance for the null model.
- **iter:** Number of algorithm iterations to compute the model.
- **df.residual:** The residual degrees of freedom.
- **df.null:** The residual degrees of freedom for the null model.
- **y:** The response variable in the model.
- **x:** A vector of the predictor variable(s) in the model.

**xval:** List of objects of class "H2OGLMMModel", representing the n-fold cross-validation models.

### Extends

Class "[H2OModel](#)", directly.

### Methods

`show` signature(object = "H2OGLMMModel"): ...

### See Also

[h2o.glm](#)

### Examples

```
showClass("H2OGLMMModel")
```

---

H2OGLMModelList-class *Class "H2OGLMModelList"*

---

### Description

Object representing the models built by a H2O GLM search over lambda values.

### Objects from the Class

Objects can be created by calls of the form `new("H2OGLMModelList", ...)`.

### Slots

`models`: Object of class "list" containing "H2OGLMModel" objects representing the models returned from the lambda search.

`best_model`: Object of class "numeric" indicating the index of the model with the optimal lambda value in the above list.

`lambdas`: Object of class "numeric" indicating the optimal lambda value from the lambda search.

### Methods

`show` signature(object = "H2OGLMModelList"): ...

`summary` signature(object = "H2OGLMModelList"): ...

### See Also

[H2OGLMModel](#), [h2o.glm](#)

### Examples

```
showClass("H2OGLMModelList")
```

---

H2OGrid-class *Class "H2OGrid"*

---

### Description

Object representing the models built by a H2O grid search algorithm.

### Objects from the Class

A virtual Class: No objects may be created from it.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2OModel" objects representing the models returned by the grid search algorithm.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the grid search algorithm.

**Methods**

**show** signature(object = "H2OGrid"): ...

**See Also**

[H2OGLMGrid](#), [H2OKMeansGrid](#), [H2ODRFGrid](#), [H2OGBMGrid](#), [H2ODeepLearningGrid](#)

**Examples**

```
showClass("H2OGrid")
```

---

H2OKMeansGrid-class    *Class "H2OKMeansGrid"*

---

**Description**

Object representing the models built by a H2O K-Means grid search.

**Objects from the Class**

Objects can be created by calls of the form `new("H2OKMeansGrid", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2OKMeansModel" objects representing the models returned by the K-Means grid search.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the K-Means grid search.

**Extends**

Class "[H2OGrid](#)", directly.

**Methods**

No methods defined with class "H2OKMeansGrid" in the signature.

**See Also**

[H2OKMeansModel](#), [h2o.kmeans](#)

**Examples**

```
showClass("H2OKMeansGrid")
```

---

H2OKMeansModel-class    *Class* "H2OKMeansModel"

---

**Description**

A class for representing k-means models.

**Objects from the Class**

Objects can be created by calls of the form `new("H2OKMeansModel", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **centers:** A matrix of cluster centers.
- **cluster:** A [H2OParsedData](#) object containing the vector of integers (from 1:k), which indicate the cluster to which each point is allocated.
- **size:** The number of points in each cluster.
- **withinss:** Vector of within-cluster sum of squares, with one component per cluster.
- **tot.withinss:** Total within-cluster sum of squares, i.e., `sum(withinss)`.

**Methods**

**show** signature(object = "H2OKMeansModel"): ...

**See Also**

[h2o.kmeans](#)

**Examples**

```
showClass("H2OKMeansModel")
```

---

H2OModel-class	Class "H2OModel"
----------------	------------------

---

**Description**

Object representing the model built by an H2O algorithm.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing the characteristics of the model returned by the algorithm.

**Methods**

No methods defined with class "H2OModel" in the signature.

**See Also**

[H2OGLMModel](#), [H2OKMeansModel](#), [H2ODRFModel](#), [H2OGBMModel](#), [H2OPCAModel](#), [H2ODeepLearningModel](#)

**Examples**

```
showClass("H2OModel")
```

---

H2ONBModel-class	Class "H2ONBModel"
------------------	--------------------

---

**Description**

A class for representing naive Bayes models.

**Objects from the Class**

Objects can be created by calls of the form `new("H2ONBModel", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **laplace:** A positive number controlling Laplace smoothing. The default (0) disables Laplace smoothing.
- **levels:** Categorical levels of the dependent variable.
- **apriori:** Total occurrences of each level of the dependent variable.
- **apriori\_prob:** A-priori class distribution for the dependent variable.
- **tables:** A list of tables, one for each predictor variable. For categorical predictors, the table displays, for each attribute level, the conditional probabilities given the target class. For numeric predictors, the table gives, for each target class, the mean and standard deviation of the variable.

**Extends**

Class "[H2OModel](#)", directly.

**Methods**

**show** signature(object = "H2ONBModel"): ...

**See Also**

[h2o.naiveBayes](#)

**Examples**

```
showClass("H2ONBModel")
```

---

H2OParsedData-class    *Class "H2OParsedData"*

---

**Description**

A class for representing imported data sets that have been parsed.

**Objects from the Class**

Objects can be created by calls of the form `new("H2OParsedData", ...)`.

**Slots**

h2o: Object of class "H2OClient", which is the client object that was passed into the function call.

key: Object of class "character", which is the hex key assigned to the imported data.

logic: Object of class "logical", indicating whether the "H2OParsedData" object represents logical data

any\_enum: Object of class "logical", indicating whether the frame has any factor columns.

ncols: Object of class "numeric", holds the number of columns of the "H2OParsedData" object.

nrows: Object of class "numeric", holds the number of rows of the "H2OParsedData" object.

col\_names: Object of class "vector", holds the column names of the "H2OParsedData" object.

**Methods**

- signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

- signature(e1 = "H2OParsedData", e2 = "numeric"): ...

- signature(e1 = "numeric", e2 = "H2OParsedData"): ...

! signature(x = "H2OParsedData"): ...

!= signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

!= signature(e1 = "H2OParsedData", e2 = "numeric"): ...

!= signature(e1 = "numeric", e2 = "H2OParsedData"): ...

[ signature(x = "H2OParsedData"): ...

[<- signature(x = "H2OParsedData"): ...

[[ signature(x = "H2OParsedData"): ...

[[<- signature(x = "H2OParsedData"): ...

\* signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

\* signature(e1 = "H2OParsedData", e2 = "numeric"): ...

\* signature(e1 = "numeric", e2 = "H2OParsedData"): ...

/ signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

/ signature(e1 = "H2OParsedData", e2 = "numeric"): ...

/ signature(e1 = "numeric", e2 = "H2OParsedData"): ...

& signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

& signature(e1 = "H2OParsedData", e2 = "logical"): ...

& signature(e1 = "H2OParsedData", e2 = "numeric"): ...

& signature(e1 = "logical", e2 = "H2OParsedData"): ...

& signature(e1 = "numeric", e2 = "H2OParsedData"): ...

%% signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

%% signature(e1 = "H2OParsedData", e2 = "numeric"): ...

%% signature(e1 = "numeric", e2 = "H2OParsedData"): ...

+ signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...

```

+ signature(e1 = "H2OParsedData", e2 = "numeric"): ...
+ signature(e1 = "numeric", e2 = "H2OParsedData"): ...
< signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
< signature(e1 = "H2OParsedData", e2 = "numeric"): ...
< signature(e1 = "numeric", e2 = "H2OParsedData"): ...
<= signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
<= signature(e1 = "H2OParsedData", e2 = "numeric"): ...
<= signature(e1 = "numeric", e2 = "H2OParsedData"): ...
== signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
== signature(e1 = "H2OParsedData", e2 = "numeric"): ...
== signature(e1 = "numeric", e2 = "H2OParsedData"): ...
> signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
> signature(e1 = "H2OParsedData", e2 = "numeric"): ...
> signature(e1 = "numeric", e2 = "H2OParsedData"): ...
>= signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
>= signature(e1 = "H2OParsedData", e2 = "numeric"): ...
>= signature(e1 = "numeric", e2 = "H2OParsedData"): ...
| signature(e1 = "H2OParsedData", e2 = "H2OParsedData"): ...
^ signature(e1 = "numeric", e2 = "H2OParsedData"): ...
^ signature(e1 = "H2OParsedData", e2 = "numeric"): ...
| signature(e1 = "H2OParsedData", e2 = "logical"): ...
| signature(e1 = "H2OParsedData", e2 = "numeric"): ...
| signature(e1 = "logical", e2 = "H2OParsedData"): ...
| signature(e1 = "numeric", e2 = "H2OParsedData"): ...
$ signature(x = "H2OParsedData"): ...
$<- signature(x = "H2OParsedData"): ...
abs signature(x = "H2OParsedData"): ...
apply signature(X = "H2OParsedData"): ...
as.data.frame signature(x = "H2OParsedData"): ...
as.factor signature(x = "H2OParsedData"): ...
ceiling signature(x = "H2OParsedData"): ...
colMeans signature(x = "H2OParsedData"): ...
colnames signature(x = "H2OParsedData"): ...
colnames<- signature(x = "H2OParsedData", value = "character"): ...
colnames<- signature(x = "H2OParsedData", value = "H2OParsedData"): ...
dim signature(x = "H2OParsedData"): ...
dim<- signature(x = "H2OParsedData"): ...

```

```
exp signature(x = "H2OParsedData"): ...  
findInterval signature(x = "H2OParsedData"): ...  
floor signature(x = "H2OParsedData"): ...  
h2o.cut signature(x = "H2OParsedData", breaks = "numeric"): ...  
h2o<- signature(x = "H2OParsedData", value = "H2OParsedData"): ...  
h2o<- signature(x = "H2OParsedData", value = "numeric"): ...  
head signature(x = "H2OParsedData"): ...  
histograms signature(object = "H2OParsedData"): ...  
ifelse signature(test = "H2OParsedData"): ...  
is.factor signature(x = "H2OParsedData"): ...  
is.na signature(x = "H2OParsedData"): ...  
length signature(x = "H2OParsedData"): ...  
levels signature(x = "H2OParsedData"): ...  
log signature(x = "H2OParsedData"): ...  
names signature(x = "H2OParsedData"): ...  
names<- signature(x = "H2OParsedData"): ...  
ncol signature(x = "H2OParsedData"): ...  
nrow signature(x = "H2OParsedData"): ...  
quantile signature(x = "H2OParsedData"): ...  
range signature(x = "H2OParsedData"): ...  
sd signature(x = "H2OParsedData"): ...  
show signature(object = "H2OParsedData"): ...  
sign signature(x = "H2OParsedData"): ...  
sqrt signature(x = "H2OParsedData"): ...  
summary signature(object = "H2OParsedData"): ...  
t signature(object = "H2OParsedData"): ...  
tail signature(x = "H2OParsedData"): ...  
trunc signature(x = "H2OParsedData"): ...  
var signature(x = "H2OParsedData"): ...
```

### See Also

[H2ORawData](#), [h2o.parseRaw](#)

### Examples

```
showClass("H2OParsedData")
```

---

H2OPCAModel-class      *Class "H2OPCAModel"*

---

### Description

A class for representing principal components analysis results.

### Objects from the Class

Objects can be created by calls of the form `new("H2OPCAModel", ...)`.

### Slots

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class [H2OParsedData](#), which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **standardized:** A logical value indicating whether the data was centered and scaled.
- **sdev:** The standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix).
- **rotation:** The matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).

### Extends

Class ["H2OModel"](#), directly.

### Methods

**show** signature(object = "H2OPCAModel"): ...

**plot** signature(x = "H2OPCAModel", y, ...): ...

**summary** signature(object = "H2OPCAModel"): ...

### See Also

[h2o.prcomp](#)

### Examples

```
showClass("H2OPCAModel")
```

---

H2OPerfModel-class      Class "H2OPerfModel"

---

### Description

A class for constructing performance measures of H2O models.

### Objects from the Class

Objects can be created by calls of the form `new("H2OPerfModel", ...)`.

### Slots

**cutoffs:** A numeric vector of threshold values.

**measure:** A numeric vector of performance values corresponding to the threshold values. The specific performance measure is given in `perf`.

**perf:** A character string indicating the performance measure used to evaluate the model. One of either "F1", "accuracy", "precision", "recall", "specificity", or "max\_per\_class\_error". See [h2o.performance](#) for a detailed description of each.

**model:** Object of class "list" containing the following elements:

- **auc:** Area under the curve.
- **gini:** Gini coefficient.
- **best\_cutoff:** Threshold value that optimizes the performance measure `perf`. If `perf` is "max\_per\_class\_error", it is minimized at this threshold, otherwise, it is maximized.
- **F1:** F1 score at best cutoff.
- **accuracy:** Accuracy value at best cutoff. Estimated as  $(TP + TN)/(P + N)$ .
- **precision:** Precision value at best cutoff. Estimated as  $TP/(TP + FP)$ .
- **recall:** Recall value at best cutoff, i.e. the true positive rate  $TP/P$ .
- **specificity:** Specificity value at best cutoff, i.e. the true negative rate  $TN/N$ .
- **max\_per\_class\_err:** Maximum per class error at best cutoff.
- **confusion:** Confusion matrix at best cutoff.

**roc:** A data frame with two columns: TPR = true positive rate and FPR = false positive rate, calculated at the listed cutoffs.

### Methods

**show** signature(object = "H2OPerfModel"): ...

**plot** signature(x = "H2OPerfModel", type, ...): ...

### See Also

[h2o.performance](#), [plot.H2OPerfModel](#)

### Examples

```
showClass("H2OPerfModel")
```

---

H2ORawData-class      *Class "H2ORawData"*

---

### Description

A class for representing imported data sets that have not been parsed.

### Objects from the Class

Objects can be created by calls of the form `new("H2ORawData", ...)`.

### Slots

**h2o:** Object of class "H2OClient", which is the client object that was passed into the function call.

**key:** Object of class "character", which is the hex key assigned to the imported data.

### Methods

**h2o.parseRaw** signature(`data = "H2OParsedData"`, `key = "character"`, `header = "logical"` `sep = "character"`, ...)

**show** signature(`object = "H2ORawData"`): ...

### See Also

[H2OParsedData](#)

### Examples

```
showClass("H2ORawData")
```

---

H2OSpeeDRFGrid-class      *Class "H2ODRFGrid"*

---

### Description

Object representing the models built by a H2O single-node random forest grid search.

### Objects from the Class

Objects can be created by calls of the form `new("H2OSpeeDRFGrid", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing "H2OSpeeDRFModel" objects representing the models returned by the distributed random forest grid search.

**sumtable:** Object of class "list" containing summary statistics of all the models returned by the distributed random forest grid search.

**Extends**

Class "H2OGrid", directly.

**Methods**

No methods defined with class "H2OSpeeDRFGrid" in the signature.

**See Also**

[H2OSpeeDRFModel](#), [h2o.SpeeDRF](#)

**Examples**

```
showClass("H2OSpeeDRFGrid")
```

---

H2OSpeeDRFModel-class *Class* "H2OSpeeDRFModel"

---

**Description**

A class for representing single-node random forest ensembles.

**Objects from the Class**

Objects can be created by calls of the form `new("H2OSpeeDRFModel", ...)`.

**Slots**

**key:** Object of class "character", representing the unique hex key that identifies the model.

**data:** Object of class "H2OParsedData", which is the input data used to build the model.

**model:** Object of class "list" containing the following elements:

- **ntree:** Number of trees grown.
- **mse:** Mean squared error for each tree.
- **confusion:** Confusion matrix of the prediction.

**valid:** Object of class "H2OParsedData", which is the data used for validating the model.

**xval:** List of objects of class "H2OSpeeDRFModel", representing the n-fold cross-validation models.

**Extends**

Class "[H2OModel](#)", directly.

**Methods**

`show signature(object = "H2OSpeeDRFModel"):` ...

**See Also**

[h2o.SpeeDRF](#)

**Examples**

```
showClass("H2OSpeeDRFModel")
```

---

head

*Return the First or Last Part of a H2O Dataset*

---

**Description**

Returns the first or last rows of an H2O parsed data object.

**Usage**

```
## S3 method for class 'H2OParsedData'
head(x, n = 6L, ...)
## S3 method for class 'H2OParsedData'
tail(x, n = 6L, ...)
```

**Arguments**

`x` An H2O parsed data object.

`n` (Optional) A single integer. If positive, number of rows in `x` to return. If negative, all but the `n` first/last number of rows in `x`.

`...` Arguments to be passed to or from other methods. (Currently unimplemented).

**Value**

A data frame containing the first or last `n` rows of an [H2OParsedData](#) object.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
head(australia.hex, 10)
tail(australia.hex, 10)
```

---

ifelse	<i>Applies conditional statements to an <a href="#">H2OParsedData</a> object.</i>
--------	---

---

### Description

Applies conditional statements to numeric vectors in H2O parsed data objects when the data are numeric.

### Usage

```
ifelse(test, yes, no)
```

### Arguments

test	A logical description of the condition to be met (>, <, =, etc...)
yes	The value to return if the condition is TRUE.
no	The value to return if the condition is FALSE.

### Details

Only numeric values can be tested, and only numeric results can be returned for either condition. Categorical data is not currently supported for this function and returned values cannot be categorical in nature.

### Value

Retruns a vector of new values matching the conditions stated in the ifelse call.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
australia.hex[,9] <- ifelse(australia.hex[,3] < 279.9, 1, 0)
summary(australia.hex)
```

---

is.factor	<i>Tells user if given column is categorical data or not.</i>
-----------	---

---

**Description**

Tells user if given column is categorical data or not.

**Usage**

```
is.factor(x)
```

**Arguments**

x Columns of an H2O parsed data object.

**Value**

A logical value TRUE if column contains categorical data, FALSE otherwise.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.hex[,4]=as.factor(prostate.hex[,4])
is.factor(prostate.hex[,4])
is.factor(prostate.hex[,3])
```

---

levels	<i>Levels of Categorical Data</i>
--------	-----------------------------------

---

**Description**

Returns a list of the unique values found in a column of categorical data.

**Usage**

```
levels(x)
```

**Arguments**

x Column of categorical data in an [H2OParsedData](#) object.

**Value**

Returns a list containing one entry for each unique value found in the column of categorical data.

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
levels(iris.hex[,5])
```

---

mean.H2OParsedData	<i>Arithmetic Mean of H2O Dataset</i>
--------------------	---------------------------------------

---

## Description

mean.H2OParsedData, a method for the [mean](#) generic. Calculate the mean of each numeric column in a H2O dataset.

## Usage

```
## S3 method for class 'H2OParsedData'
mean(x, trim = 0, na.rm = FALSE, ...)
```

## Arguments

x	An <a href="#">H2OParsedData</a> object.
trim	(The fraction (0 to 0.5) of observations to trim from each end of x before the mean is computed. (Currently unimplemented).
na.rm	Logical value indicating whether NA or missing values should be stripped before the computation.
...	Potential further arguments. (Currently unimplemented).

## Value

An [H2OParsedData](#) object of scalar numeric value representing the arithmetic mean of each numeric column of x. If x is not logical or numeric, then NA\_real\_ is returned, with a warning.

## Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
mean(prostate.hex$AGE)
```

---

nrow	<i>The Number of Rows/Columns of a H2O Dataset</i>
------	--

---

**Description**

Returns a count of the number of rows in an [H2OParsedData](#) object.

**Usage**

```
nrow(x)
ncol(x)
```

**Arguments**

x                    An [H2OParsedData](#) object.

**Value**

An integer of length 1 indicating the number of rows or columns in the dataset.

**See Also**

[dim](#) which returns all dimensions

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
nrow(iris.hex)
ncol(iris.hex)
```

---

plot.H2OPerfModel	<i>Scatterplot of H2O Performance Measures</i>
-------------------	--

---

**Description**

Draw scatter plot of a particular performance measure vs. thresholds for a H2O model, or the ROC curve.

**Usage**

```
## S3 method for class 'H2OPerfModel'
plot(x, type = "cutoffs", ...)
```

**Arguments**

x	An <a href="#">H2OPerfModel</a> object.
type	Either "cutoffs" to plot the performance measure <code>x@perf</code> versus thresholds <code>x@cutoffs</code> , or "roc" to plot the corresponding ROC curve (true positive rate vs. false positive rate).
...	Arguments to be passed to methods, such as graphical parameters (see <a href="#">par</a> for details).

**See Also**

[H2OPerfModel](#), [h2o.performance](#)

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)

# Run GBM classification on prostate.csv
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, key = "prostate.hex")
prostate.gbm = h2o.gbm(y = 2, x = 3:9, data = prostate.hex)

# Calculate performance measures at threshold that maximizes precision
prostate.pred = h2o.predict(prostate.gbm)
prostate.perf = h2o.performance(prostate.pred[,3], prostate.hex$CAPSULE, measure = "precision")

plot(prostate.perf, type = "cutoffs") # Plot precision vs. thresholds
plot(prostate.perf, type = "roc")    # Plot ROC curve
```

---

quantile.H2OParsedData

*Obtain and display quantiles for H2O parsed data.*

---

**Description**

quantile.H2OParsedData, a method for the [quantile](#) generic. Obtain and return quantiles for an [H2OParsedData](#) object.

**Usage**

```
## S3 method for class 'H2OParsedData'
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE, names = TRUE, type = 7, ...)
```

**Arguments**

x	An <a href="#">H2OParsedData</a> object with a single numeric column.
probs	numeric vector of probabilities with values in [0,1].
na.rm	logical; if true, any NA and NaN's are removed from x before the quantiles are computed.
names	logical; if true, the result has a names attribute.
type	integer selecting the quantile algorithm to use. Currently, only type 7 (linear interpolation) is supported.
...	further arguments passed to or from other methods.

**Details**

Note that H2O parsed data objects can be quite large, and are therefore often distributed across multiple nodes in an H2O cluster. As a result, percentiles at the 1st, 5th, 10th, 25th, 33, 50, 66, 75, 90, 95, 99th, and other values cannot be returned. This range includes the 1st quantile at the 25th percentile, median at the 50th percentile, and 3rd quantile at the 75th percentile.

**Value**

A vector describing the percentiles at the given cutoffs for the [H2OParsedData](#) object.

**Examples**

```
# Request quantiles for an H2O parsed data set:
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)

# Request quantiles for a subset of columns in an H2O parsed data set
quantile(prostate.hex[,3])
for(i in 1:ncol(prostate.hex))
  quantile(prostate.hex[,i])
```

---

Round

*Rounding of Numbers*


---

**Description**

round rounds the values in a [H2OParsedData](#) object to the specified number of decimal places.  
 signif rounds the values in a [H2OParsedData](#) object to the specified number of significant digits.

**Usage**

```
## S3 method for class 'H2OParsedData'
round(x, digits = 0)
## S3 method for class 'H2OParsedData'
signif(x, digits = 6)
```

**Arguments**

x	An <a href="#">H2OParsedData</a> object with numeric entries.
digits	Single number specifying decimal places (round) or significant digits (signif) to use. Negative values are interpreted as a power of ten, e.g. <code>round(x, digits = -2)</code> round to the nearest hundred.

**Details**

This method uses the IEC 60559 standard for rounding to the even digit, so 0.5 goes to 0 and -1.5 goes to -2. See the Java documentation of `RoundingMode.HALF_EVEN` for more details and examples.

**Value**

Returns a [H2OParsedData](#) object with each entry rounded as specified. An error will occur if any of these entries is non-numeric.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
iris.data = iris.hex[,1:3]

iris.rounded = round(iris.data)
head(iris.rounded)
iris.signif = signif(iris.data, 2)
head(iris.signif)
```

---

`screepLOT.H2OPCAModel` *Summarizes the columns of an H2O parsed FluidVecs data set.*

---

**Description**

`screepLOT.H2OPCAModel`, a method for the [screepLOT](#) generic. Plots the variances against the number of the principal component generated by [h2o.prcomp](#).

**Usage**

```
## S3 method for class 'H2OPCAModel'
screepLOT(x, npcs = min(10, length(x@model$sdev)), type = "barplot",
  main = paste("h2o.prcomp(", x@data@key, ")", sep=""), ...)
```

**Arguments**

x	An <a href="#">H2OPCAModel</a> object.
npcs	Number of components to be plotted.
type	Type of plot, must be either "barplot" or "lines".
main	Title of the plot.
...	Additional parameters to be passed to the plotting function.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package = "h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
australia.pca = h2o.prcomp(data = australia.hex, standardize = TRUE)
screepplot(australia.pca)
```

sd

*Standard Deviation of a Numeric Column of H2O Data***Description**

Calculates the standard deviation of a [H2OParsedData](#) column of continuous real valued data.

**Usage**

```
sd(x, na.rm = FALSE)
```

**Arguments**

x	An <a href="#">H2OParsedData</a> object containing numeric data.
na.rm	Logical value where FALSE does not remove NA's in the calculation and TRUE removes NA's in the calculation.

**Value**

Returns a vector of values of the standard deviations for the requested columns.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
sd(iris.hex[,4])
```

---

str *Display the Structure of a H2O Dataset*

---

### Description

A method for the [str](#) generic. Obtain information about H2O parsed data objects and their structure.

### Usage

```
## S3 method for class 'H2OParsedData'
str(object, ...)
```

### Arguments

object            An [H2OParsedData](#) object.  
 ...              Potential further arguments. (Currently unimplemented).

### Value

A table listing summary information including variable names, types (for example, enum or numeric), count of observations and columns.

### Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
str(prostate.hex)
```

---

sum *Sum of Numeric Values*

---

### Description

Calculates the sum of all the values present in its arguments. This method extends the [sum](#) generic to deal with [H2OParsedData](#) objects.

### Usage

```
sum(..., na.rm = FALSE)
```

### Arguments

...              Numeric, complex, logical or [H2OParsedData](#) arguments.  
 na.rm           Logical value where FALSE does not remove NA's in the calculation and TRUE removes NA's in the calculation.

**Value**

Returns the sum over all the input arguments. For a [H2OParsedData](#) object, the sum is taken over all entries in the dataset. An error will occur if any of those entries is non-numeric.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath, key = "australia.hex")
sum(australia.hex)
sum(c(400, 1234, -1250), TRUE, australia.hex[,1:4])
```

---

summary

*Summarizes the columns of a H2O Dataset*


---

**Description**

A method for the [summary](#) generic. Summarizes the columns of an H2O parsed object or subset of columns and rows using vector notation (e.g. dataset[row, col])

**Usage**

```
## S3 method for class 'H2OParsedData'
summary(object, ...)
```

**Arguments**

object	An <a href="#">H2OParsedData</a> object.
...	Additional arguments affecting the summary produced. (Currently unimplemented).

**Value**

A matrix displaying the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column included in the request call, a summary of the levels and member counts for each factor column. and a the levels and member counts of the elements in factor columns for all of the columns specified in the summary call.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
summary(prostate.hex)
summary(prostate.hex$GLEASON)
summary(prostate.hex[,4:6])
```

---

summary.H2OPCAModel    *Summarizes the H2O PCA Model*

---

### Description

summary.H2OPCAModel, a method for the [summary](#) generic. Summarizes the importance of each principal component returned by [h2o.prcomp](#).

### Usage

```
## S3 method for class 'H2OPCAModel'  
summary(object, ...)
```

### Arguments

object            An [H2OPCAModel](#) object.  
...                Additional arguments affecting the summary produced. (Currently unimplemented).

### Value

A matrix displaying the standard deviation, proportion of variance explained and cumulative proportion of variance explained by each principal component.

### Examples

```
library(h2o)  
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)  
ausPath = system.file("extdata", "australia.csv", package="h2o")  
australia.hex = h2o.importFile(localH2O, path = ausPath)  
australia.pca = h2o.prcomp(data = australia.hex, standardize = TRUE)  
summary(australia.pca)
```

---

unique.H2OParsedData    *Extract Unique Elements from H2O Dataset*

---

### Description

unique.H2OParsedData, a method for the [unique](#) generic. Returns a H2O dataset like x but with duplicate elements/rows removed.

### Usage

```
## S3 method for class 'H2OParsedData'  
unique(x, incomparables = FALSE, ...)  
  
h2o.unique(x, incomparables = FALSE, ...)
```

**Arguments**

x	An <code>H2OParsedData</code> object.
incomparables	A vector of values that cannot be compared, or FALSE which indicates all values can be compared. (Currently unimplemented).
...	Potential further arguments. (Currently only partially unimplemented).

**Details**

Only MARGIN = 2 is currently supported, that is, dropping duplicate rows in a H2O dataset. This method runs on top of ddply in H2O.

**Value**

An `H2OParsedData` with the same columns, but all duplicate rows removed.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
nrow(prostate.hex$AGE)
prosAge.uniq = unique(prostate.hex$AGE)
nrow(prosAge.uniq)
head(prosAge.uniq)
```

---

zzz\_ShutdownAfterExamples

*Shutdown H2O cloud after examples run (for H2O developers only)*

---

**Description**

zzz\_ShutdownAfterExamples, shutdown H2O cloud after examples run. This is only relevant for H2O developers during the building of the CRAN package.

**Examples**

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
h2o.shutdown(localH2O, prompt = FALSE)
Sys.sleep(2)
```