"h2o"

April 28, 2015

R topics documented:

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:	0.2.2.15
Branch:	rel-severi
Date:	Tue Apr 28 16:46:37 PDT 2015
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 at localhost: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: Tom Kraljevic <tomk@0xdata.com>

References

- 0xdata Homepage
- H2O Documentation
- H2O on Github

aaa *Starting H2O For examples*

Description

Starting H2O For examples

Examples

```
h2o.init()
```

apply,H2OFrame-method *Apply on H2O Datasets*

Description

Method for apply on H2OFrame objects.

Usage

```
## S4 method for signature H2OFrame
apply(X, MARGIN, FUN, ...)
```

Arguments

X	an H2OFrame object on which apply will operate.
MARGIN	the vector on which the function will be applied over, either 1 for rows or 2 for columns.
FUN	the function to be applied.
...	optional arguments to FUN.

Value

Produces a new H2OFrame of the output of the applied function. The output is stored in H2O so that it can be used in subsequent H2O processes.

See Also

apply for the base generic

Examples

```
localH2O = h2o.init()
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(apply(iris.hex, 1, sum))
```

`as.character,H2OFrame-method`

Convert H2O Data to Characters

Description

Converts an H2O column into character columns.

Usage

```
## S4 method for signature H2OFrame
as.character(x)
```

Arguments

x	a column from an H2OFrame data set. localH2O <- h2o.init() iris.hex <- as.h2o(iris) iris.hex[,5] <- as.character(iris.hex[,5])

`as.data.frame.H2OFrame`

Converts a Parsed H2O data into a Data Frame

Description

Downloads the H2O data and then scans it in to an R data frame.

Usage

```
## S3 method for class H2OFrame
as.data.frame(x, ...)
```

Arguments

x	An H2OFrame object.
...	Further arguments to be passed down from other methods.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
as.data.frame(prostate.hex)
```

`as.environment,H2OFrame-method`

Convert H2O Data to an R Enviornment

Description

Converts an H2OFrame to an environment.

Usage

```
## S4 method for signature H2OFrame
as.environment(x)
```

Arguments

x	an H2OFrame class object.

Value

Returns an R environment object based on the H2OFrame. localH2O <- h2o.init() prosPath <- system.file("extdata", "prostate.csv", package="h2o") prostate.hex <- h2o.uploadFile(localH2O, path = prosPath) names(as.environment) aa <- as.environment(prostate.hex) ls(aa)

`as.factor,H2OFrame-method`

Convert H2O Data to Factors

Description

Convert a column into a factor column.

Usage

```
## S4 method for signature H2OFrame
as.factor(x)
```

Arguments

x	a column from an H2OFrame data set.

See Also

`is.factor`.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.hex[,2] <- as.factor(prostate.hex[,2])
summary(prostate.hex)
```

---

as.h2o *R data.frame -> H2OFrame*

Description

Import a local R data frame to the H2O cloud.

Usage

```
as.h2o(object, conn = h2o.getConnection(), destination_frame = "")
```

Arguments

object	An R data frame.
conn	An H2OConnection object containing the IP address and port number of the H2O server.
destination_frame	A string with the desired name for the H2O Frame.

---

as.matrix.h2o *Converts H2O Data to an R Matrix*

Description

Convert an H2OFrame object to a matrix, which allows subsequent data frame operations within the R environment.

Usage

```
## S3 method for class H2OFrame
as.matrix(x, ...)
```

Arguments

x	An H2OFrame object
...	Additional arguments to be passed to or from

Value

Returns a matrix in the R enviornment.

Note

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.

See Also

`as.matrix` for the base `R` implementation.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.matrix <- as.matrix(prostate.hex)
summary(prostate.matrix)
head(prostate.matrix)
```

---

as.numeric,H2OFrame-method

Convert H2O Data to Numeric

Description

Converts an H2O column into a numeric value column.

Usage

```
## S4 method for signature H2OFrame
as.numeric(x)
```

Arguments

x	a column from an H2OFrame data set. localH2O <- h2o.init() prosPath <- system.file("extdata", "prostate.csv", package="h2o") prostate.hex <- h2o.uploadFile(localH2O, path = prosPath) prostate.hex[,2] <- as.factor(prostate.hex[,2]) prostate.hex[,2] <- as.numeric(prostat.hex[,2])

ASTNode-class *The ASTNode class.*

Description

This class represents a node in the abstract syntax tree. An ASTNode has a root. The root has children that either point to another ASTNode, or to a leaf node, which may be of type ASTNumeric or ASTFrame.

Usage

```
## S4 method for signature ASTNode
show(object)
```

Arguments

object	An ASTNode class object.

Slots

`root` Object of type `Node`

`children` Object of type `list`

colnames<-,H2OFrame,H2OFrame-method *Returns Column Names for a Parsed H2O Data Object.*

Description

Returns column names for an H2OFrame object.

Usage

```
## S4 replacement method for signature H2OFrame,H2OFrame
colnames(x) <- value

## S4 replacement method for signature H2OFrame,character
colnames(x) <- value

## S4 method for signature H2OFrame
names(x)

## S4 replacement method for signature H2OFrame
names(x) <- value
```

```
## S4 method for signature H2OFrame
colnames(x)

## S4 method for signature H2OFrame
names(x)
```

Arguments

x	An H2OFrame object.
value	a character string to rename columns.

See Also

colnames for the base R method.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
summary(iris.hex)
colnames(iris.hex)
```

---

cut.H2OFrame *Cut 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 the level one, the next is level two, etc.

Usage

```
## S3 method for class H2OFrame
cut(x, breaks, labels = NULL, include.lowest = FALSE,
  right = TRUE, dig.lab = 3, ...)
```

Arguments

x	An H2OFrame object with numeric columns.
breaks	A numeric vector of two or more unique cut points.
labels	Labels for the levels of the resulting category. By default, labels are constructed sing "(a,b]" interval notation.
include.lowest	Logical, indicationg if an 'x[i]' equal to the lowest (or highest, for right = FALSE 'breaks' value should be included

right	/codeLogical, indicating if the intervals should be closed on the right (opened on the left) or vice versa.
dig.lab	Integer which is used when labels are not given, determines the number of digits used in formatting the beak numbers.
...	Further arguments passed to or from other methods.

Value

Returns an H2OFrame object containing the factored data with intervals as levels.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

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

---

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(object, data)
```

Arguments

object	An H2OAutoEncoderModel object that represents the model to be used for anomaly detection.
data	An H2OFrame object.

Value

Returns an H2OFrame object containing the reconstruction MSE.

See Also

h2o.deeplearning for making an H2OAutoEncoderModel.

Examples

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

---

h2o.anyFactor *Check H2OFrame columns for factors*

Description

Determines if any column of an H2OFrame object contains categorical data.

Usage

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

Arguments

x	An H2OFrame object.

Value

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

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.importFile(localH2O, path = irisPath)
h2o.anyFactor(iris.hex)
```

h2o.assign *Rename an H2O object.*

Description

Makes a copy of the data frame and gives it the desired the key.

Usage

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

Arguments

data	An H2OFrame object
key	The hex key to be associated with the H2O parsed data object

h2o.auc *Retrieve an H2O AUC metric*

Description

Retrieves the AUC value from an H2OBinomialMetrics.

Usage

```
h2o.auc(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	An H2OBinomialMetrics object.
train	Retrieve the training AUC
valid	Retrieve the validation AUC
...	extra arguments to be passed if 'object' is of type H2OModel (e.g. train=TRUE)

See Also

h2o.giniCoef for the Gini coefficient, h2o.mse for MSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.auc(perf)
```

`h2o.avg_between_ss` *Get the average between sum of squares.*

Description

Get the average between sum of squares.

Usage

```
h2o.avg_between_ss(object, train = FALSE, valid = FALSE, ...)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training metric.
`valid`	Retreive the validation metric.
`...`	further arguments to be passed on (currently unimplemented)

`h2o.avg_ss` *Get the average sum of squares.*

Description

Get the average sum of squares.

Usage

```
h2o.avg_ss(object, train = FALSE, valid = FALSE, ...)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training metric.
`valid`	Retreive the validation metric.
`...`	further arguments to be passed on (currently unimplemented)

h2o.avg_within_ss *Get the average wtihin sum of squares.*

Description

Get the average wtihin sum of squares.

Usage

```
h2o.avg_within_ss(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	An H2OClusteringModel object.
train	Retrieve the training metric.
valid	Retreive the validation metric.
...	further arguments to be passed on (currently unimplemented)

h2o.cbind *Combine H2O Datasets by Columns*

Description

Takes a sequence of H2O data sets and combines them by column

Usage

```
h2o.cbind(...)
```

Arguments

...	A sequence of H2OFrame arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.

Value

An H2OFrame object containing the combined ... arguments column-wise.

See Also

cbind for the base R method.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.cbind <- h2o.cbind(prostate.hex, prostate.hex)
head(prostate.cbind)
```

---

h2o.centers *Retrieve the Model Centers*

Description

Retrieve the Model Centers

Usage

```
h2o.centers(object, ...)
```

Arguments

object	An H2OClusteringModel object.
...	further arguments to be passed on (currently unimplemented)

---

h2o.centersSTD *Retrieve the Model Centers STD*

Description

Retrieve the Model Centers STD

Usage

```
h2o.centersSTD(object, ...)
```

Arguments

object	An H2OClusteringModel object.
...	further arguments to be passed on (currently unimplemented)

h2o.clearLog *Delete All H2O R Logs*

Description

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

Usage

```
h2o.clearLog()
```

See Also

h2o.startLogging, h2o.stopLogging, h2o.openLog

Examples

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
h2o.clearLog()
```

h2o.clusterInfo *Print H2O cluster info*

Description

Print H2O cluster info

Usage

```
h2o.clusterInfo(conn = h2o.getConnection())
```

Arguments

conn	H2O connection object

h2o.clusterIsUp *Determine if an H2O cluster is up or not*

Description

Determine if an H2O cluster is up or not

Usage

```
h2o.clusterIsUp(conn = h2o.getConnection())
```

Arguments

conn	H2O connection object

Value

TRUE if the cluster is up; FALSE otherwise

h2o.clusterStatus *Return the status of the cluster*

Description

Retrieve information on the status of the cluster running H2O.

Usage

```
h2o.clusterStatus(conn = h2o.getConnection())
```

Arguments

conn	the H2OConnection object containing the IP address and port of the server running H2O.

See Also

H2OConnection, h2o.init

Examples

```
localH2O <- h2o.init()
h2o.clusterStatus(localH2O)
```

h2o.cluster_sizes *Retrieve the cluster sizes*

Description

Retrieve the cluster sizes

Usage

```
h2o.cluster_sizes(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	An H2OClusteringModel object.
train	Retrieve the training metric.
valid	Retrieve the validation metric.
...	further arguments to be passed on (currently unimplemented)

h2o.confusionMatrix *Access H2O Confusion Matrices*

Description

Retrieve either a single or many confusion matrices from H2O objects.

Usage

```
h2o.confusionMatrix(object, ...)

## S4 method for signature H2OModel
h2o.confusionMatrix(object, newdata, train = FALSE,
  valid = FALSE, ...)

## S4 method for signature H2OModelMetrics
h2o.confusionMatrix(object, thresholds)
```

Arguments

object	Either an H2OModel object or an H2OModelMetrics object.
...	Extra arguments for extracting train or valid confusion matrices.
newdata	An H2OFrame object that can be scored on. Requires a valid response column.
train	Retrieve the training metric.
valid	Retreive the validation metric.
thresholds	(Optional) A value or a list of values between 0.0 and 1.0. This value is only used in the case of H2OBinomialMetrics objects.

Details

The H2OModelMetrics version of this function will only take H2OBinomialMetrics or H2OMultinomialMetrics objects. If no threshold is specified, all possible thresholds are selected.

Value

Calling this function on H2OModel objects returns a confusion matrix corresponding to the `predict` function. If used on an H2OBinomialMetrics object, returns a list of matrices corresponding to the number of thresholds specified.

See Also

`predict` for generating prediction frames, `h2o.performance` for creating H2OModelMetrics.

Examples

```
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)
hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
h2o.confusionMatrix(model, hex)
# Generating a ModelMetrics object
perf <- h2o.performance(model, hex)
h2o.confusionMatrix(perf)
```

---

h2o.createFrame *Data Frame Creation in H2O*

Description

Creates a data frame in H2O with real-valued, categorical, integer, and binary columns specified by the user.

Usage

```
h2o.createFrame(conn = h2o.getConnection(), key = "", rows = 10000,
  cols = 10, randomize = TRUE, value = 0, real_range = 100,
  categorical_fraction = 0.2, factors = 100, integer_fraction = 0.2,
  integer_range = 100, binary_fraction = 0.1, binary_ones_fraction = 0.02,
  missing_fraction = 0.01, response_factors = 2, has_response = FALSE,
  seed)
```

Arguments

conn	A H2OConnection object.
key	A string indicating the destination key. If empty, this will be auto-generated by H2O.
rows	The number of rows of data to generate.
cols	The number of columns of data to generate. Excludes the response column if has_response = TRUE.
randomize	A logical value indicating whether data values should be randomly generated. This must be TRUE if either categorical_fraction or integer_fraction is non-zero.
value	If randomize = FALSE, then all real-valued entries will be set to this value.
real_range	The range of randomly generated real values.
categorical_fraction	The fraction of total columns that are categorical.
factors	The number of (unique) factor levels in each categorical column.
integer_fraction	The fraction of total columns that are integer-valued.
integer_range	The range of randomly generated integer values.
binary_fraction	The fraction of total columns that are binary-valued.
binary_ones_fraction	The fraction of values in a binary column that are set to 1.
missing_fraction	The fraction of total entries in the data frame that are set to NA.
response_factors	If has_response = TRUE, then this is the number of factor levels in the response column.
has_response	A logical value indicating whether an additional response column should be prepended to the final H2O data frame. If set to TRUE, the total number of columns will be cols+1.
seed	A seed used to generate random values when randomize = TRUE.

Value

Returns a H2OFrame object.

Examples

```
library(h2o)
localH2O <- h2o.init()
hex <- h2o.createFrame(localH2O, rows = 1000, cols = 100, categorical_fraction = 0.1,
                       factors = 5, integer_fraction = 0.5, integer_range = 1,
                       has_response = TRUE)
head(hex)
summary(hex)
```

```
hex2 <- h2o.createFrame(localH2O, rows = 100, cols = 10, randomize = FALSE, value = 5,
                        categorical_fraction = 0, integer_fraction = 0)
summary(hex2)
```

`h2o.ddply` *# children <- list(c(paste0('',nAggs), unlist(lapply(aggs, function(l) .args.to.ast(.args=l))))) Split H2O Dataset, Apply Function, and Return Results*

Description

For each subset of an H2O data set, apply a user-specified function, then comine the results.

Usage

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

Arguments

`.data`	An H2OFrame 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.
`.progress`	Name of the progress bar to use. #TODO: (Currently unimplemented)
`...`	Additional arguments passed on to `.fun`. #TODO: (Currently unimplemented)

Value

Returns a H2OFrame object containing the results from the split/apply operation, arranged

See Also

`ddply` for the plyr library implementation.

Examples

```
library(h2o)
localH2O <- h2o.init()

# Import iris dataset to H2O
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
# Add function taking mean of sepal_len column
fun = function(df) { sum(df[,1], na.rm = T)/nrow(df) }
# Apply function to groups by class of flower
# uses h2os ddply, since iris.hex is an H2OFrame object
res = h2o.ddply(iris.hex, "class", fun)
head(res)
```

h2o.deepfeatures *Feature Generation via H2O Deep Learning Model*

Description

Extract the non-linea feature from an H2O data set using an H2O deep learning model.

Usage

```
h2o.deepfeatures(object, data, layer = 1)
```

Arguments

object	An H2OModel object that represents the deep learning model to be used for feature extraction.
data	An H2OFrame object.
layer	Index of the hidden layer to extract.

Value

Returns an H2OFrame object with as many features as the number of units in the hidden layer of the specified index.

See Also

link{h2o.deeplearning} for making deep learning models.

Examples

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
prostate.dl = h2o.deeplearning(x = 3:9, y = 2, training_frame = prostate.hex,
                               hidden = c(100, 200), epochs = 5)
prostate.deepfeatures_layer1 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 1)
prostate.deepfeatures_layer2 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 2)
head(prostate.deepfeatures_layer1)
head(prostate.deepfeatures_layer2)
```

h2o.deeplearning *Build a Deep Learning Neural Network*

Description

Performs Deep Learning neural networks on an H2OFrame

Usage

```
h2o.deeplearning(x, y, training_frame, model_id = "",
  override_with_best_model, n_folds = 0, validation_frame, checkpoint,
  autoencoder = FALSE, use_all_factor_levels = TRUE,
  activation = c("Rectifier", "Tanh", "TanhWithDropout",
  "RectifierWithDropout", "Maxout", "MaxoutWithDropout"), hidden = c(200,
  200), epochs = 10, train_samples_per_iteration = -2, seed,
  adaptive_rate = TRUE, rho = 0.99, epsilon = 1e-08, rate = 0.005,
  rate_annealing = 1e-06, rate_decay = 1, momentum_start = 0,
  momentum_ramp = 1e+06, momentum_stable = 0,
  nesterov_accelerated_gradient = TRUE, input_dropout_ratio = 0,
  hidden_dropout_ratios, l1 = 0, l2 = 0, max_w2 = Inf,
  initial_weight_distribution = c("UniformAdaptive", "Uniform", "Normal"),
  initial_weight_scale = 1, loss = c("Automatic", "CrossEntropy",
  "MeanSquare", "Absolute", "Huber"), score_interval = 5,
  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 = FALSE, class_sampling_factors,
  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, average_activation, sparsity_beta, max_categorical_features,
  reproducible = FALSE, export_weights_and_biases = FALSE, ...)
```

Arguments

x	A vector containing the character names of the predictors in the model.
y	The name of the response variable in the model.
training_frame	An H2OFrame object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
override_with_best_model	Logcial. If TRUE, override the final model with the best model found during traning. Defaults to TRUE.
n_folds	(Optional) Number of folds for cross-validation. If nfolds >= 2, then validation must remain empty.

validation_frame
: (Optional) An H2OFrame object indicating the validation dataset used to contruct the 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."

autoencoder
: Enable auto-encoder for model building.

use_all_factor_levels
: Logical. Use all factor levels of categorical variance. Otherwise the first factor level is omittted (without loss of accuracy). Useful for variable imporotances and auto-enabled for autoencoder.

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))

epochs
: How many times the dataset shoud 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); or **-2** autotuning (default)

seed
: Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded

adaptive_rate
: Logical. Adaptive learning rate (ADAELTA)

rho
: Adaptive learning rate time decay factor (similarity to prior updates)

epsilon
: Adaptive learning rate parameter, similar to learn rate annealing during initial training phase. Typical values are between 1.0e-10 and 1.0e-4

rate
: Learning rate (higher => less stable, lower => slower convergence)

rate_annealing
: Learning rate annealing: $(rate)/(1 + rate_annealing * samples)$

rate_decay
: Learning rate decay factor between layers (N-th layer: $rate * \alpha^(N - 1)$)

momentum_start
: Initial momentum at the beginning of traning (try 0.5)

momentum_ramp
: Number of training samples for which momentum increases

momentum_stable
: Final momentum after ther amp is over (try 0.99)

nesterov_accelerated_gradient
: Logical. Use Nesterov accelerated gradient (recommended)

input_dropout_ratio
: A fraction of the features for each training row to be omitted from training in order to improve generalization (dimension sampling).

hidden_dropout_ratios
: Input layer dropout ration (can improve generalization) specify one value per hidden layer, defaults to 0.5

l1
: L1 regularization (can add stability and improve generalization, cause 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. Rectifier)

`initial_weight_distribution`
Can be "Uniform", "UniformAdaptive", or "Normal"

`initial_weight_scale`
Unifrom: -value ... value, Normal: stddev

`loss` Loss function: Automatic, CrossEntropy (for classification only), MeanSquare, Absolute (experimental) or Huber (experimental)

`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 ration computation(for multi-class only, 0 to disable)

`balance_classes`
Balance training data class counts via over/under-sampling (for imbalanced data)

`class_sampling_factors`
Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

`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 approximations in back-propagation)

`ignore_const_cols`
: Igrnore constant training columns (no information can be gained anwyay)

`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

`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 $numRows * numNodes$

`sparse`
: Sparse data handling (Experimental)

`col_major`
: Use a column major weight matrix for input layer. Can speed up forward proagation, but might slow down backpropagation (Experimental)

`average_activation`
: Average activation for sparse auto-encoder (Experimental)

`sparsity_beta`
: Sparsity regularization (Experimental)

`max_categorical_features`
: Max. number of categorical features, enforced via hashing Experimental)

`reproducible`
: Force reproducibility on small data (will be slow - only uses 1 thread)

`export_weights_and_biases`
: Whether to export Neural Network weights and biases to H2O Frames"

`...`
: extra parameters to pass onto functions (not implemented)

See Also

`predict.H2OModel` for prediction.

Examples

```
library(h2o)
localH2O <- h2o.init()

irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
iris.dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex)
```

---

h2o.dim *Returns the Dimensions of a Parsed H2O Data Object.*

Description

Returns the number of rows and columns for an H2OFrame object.

Usage

```
## S4 method for signature H2OFrame
dim(x)
```

Arguments

x	An H2OFrame object.

See Also

dim for the base R method.

Examples

```
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
dim(iris.hex)
```

---

h2o.downloadAllLogs *Download H2O Log Files to Disk*

Description

h2o.downloadAllLogs downloads all H2O log files to local disk. Generally used for debugging purposes.

Usage

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

Arguments

conn	An H2OConnection object pointing to a running H2O cluster.
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

H2OConnection

h2o.downloadCSV *Download H2O Data to Disk*

Description

Download an H2O data set to a CSV file on the local disk

Usage

```
h2o.downloadCSV(data, filename)
```

Arguments

data	an H2OFrame object to be downloaded.
filename	A string indicating the name that the CSV file should be should be saved to.

Warning

Files located on the H2O server may be very large! Make sure you have enough hard drive psace to accomoadet the entire file.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(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.exportFile *Export an H2O Data Frame to a File*

Description

Exports an H2OFrame (which can be either VA or FV) to a file. This file may be on the H2O instace's local filesystem, or to HDFS (preface the path with hdfs://) or to S3N (preface the path with s3n://).

Usage

```
h2o.exportFile(data, path, force = FALSE)
```

Arguments

data	An H2OFrame 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 line of the file.
force	logical, indicates how to deal with files that already exist.

Details

In the case of existing files `forse = TRUE` will overwrite the file. Otherwise, the operation will fail.

Examples

```
## Not run:
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(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.exportHDFS *Export a Model to HDFS*

Description

Exports an H2OModel to HDFS.

Usage

```
h2o.exportHDFS(object, path, force = FALSE)
```

Arguments

object	an H2OModel class object.
path	The path to write the model to. Must include the driectory and filename.
force	logical, indicates how to deal with files that already exist.

h2o.gbm *Gradient Boosted Machines*

Description

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

Usage

```
h2o.gbm(x, y, training_frame, model_id, distribution = c("AUTO", "gaussian",
  "bernoulli", "multinomial"), ntrees = 50, max_depth = 5, min_rows = 10,
  learn_rate = 0.1, nbins = 20, validation_frame = NULL,
  balance_classes = FALSE, max_after_balance_size = 1, seed, nfolds,
  score_each_iteration, ...)
```

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).
training_frame	An H2OFrame object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
distribution	A character string. The loss function to be implemented. Must be "AUTO", "bernoulli", "multinomial", or "gaussian"
ntrees	A nonnegative integer that determines the number of trees to grow.
max_depth	Maximum depth to grow the tree.
min_rows	Minimum number of rows to assign to teminal nodes.
learn_rate	An interger from 0.0 to 1.0
nbins	Number of bins to use in building histogram.
validation_frame	An H2OFrame object indicating the validation dataset used to contruct the confusion matrix. If left blank, this defaults to the training data when nfolds = 0
balance_classes	logical, indicates whether or not to 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)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded

`nfolds` (Optional) Number of folds for cross-validation. If `nfolds >= 2`, then `validation` must remain empty. **Currently not supported**

`score_each_iteration` Attempts to score each tree.

`...` extra arguments to pass on (currently no implemented)

Details

The default distribution function will guess the model type based on the response column typerun properly the response column must be an numeric for "gaussian" or an enum for "bernoulli" or "multinomial".

See Also

`predict.H2OModel` for prediction.

Examples

```
library(h2o)
localH2O = h2o.init()

# Run regression GBM on australia.hex data
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
independent <- c("premax", "salmax","minairtemp", "maxairtemp", "maxsst",
                 "maxsoilmoist", "Max_czcs")
dependent <- "runoffnew"
h2o.gbm(y = dependent, x = independent, training_frame = australia.hex,
        ntrees = 3, max_depth = 3, min_rows = 2)
```

---

h2o.getConnection *Retrieve an H2O Connection*

Description

Attempt to recover an h2o connection.

Usage

```
h2o.getConnection()
```

Value

Returns an H2OConnection object.

h2o.getFrame *Get an R Reference to an H2O Dataset*

Description

Get the reference to a frame with the given frame_id in the H2O instance.

Usage

```
h2o.getFrame(frame_id, conn = h2o.getConnection(), linkToGC = FALSE)
```

Arguments

frame_id	A string indicating the unique frame of the dataset to retrieve.
conn	H2OConnection object containing the IP address and port of the server running H2O.
linkToGC	a logical value indicating whether to remove the underlying frame from the H2O cluster when the R proxy object is garbage collected.

h2o.getGLMModel *Resolve a GLM H2O Futures Model*

Description

Turns an H2OModelFuture into a model of the correct type.

Usage

```
h2o.getGLMModel(keys, conn)
```

Arguments

keys	an H2OModelFuture or correct job key.
conn	a corresponding H2OConnection class object.

Value

Returns the correct H2OModel for the created model.

h2o.getModel *Get an R reference to an H2O model*

Description

Returns a reference to an existing model in the H2O instance.

Usage

```
h2o.getModel(model_id, conn = h2o.getConnection(), linkToGC = FALSE)
```

Arguments

model_id	A string indicating the unique model_id of the model to retrieve.
conn	H2OConnection object containing the IP address and port of the server running H2O.
linkToGC	a logical value indicating whether to remove the underlying model from the H2O cluster when the R proxy object is garbage collected.

Value

Returns an object that is a subclass of H2OModel.

Examples

```
library(h2o)
localH2O <- h2o.init()

iris.hex <- as.h2o(iris, localH2O, "iris.hex")
model_id <- h2o.gbm(x = 1:4, y = 5, training_frame = iris.hex)@model_id
model.retrieved <- h2o.getModel(model_id, localH2O)
```

h2o.getTimezone *Get the Time Zone on the H2O Cloud*

Description

Get the Time Zone on the H2O Cloud

Usage

```
h2o.getTimezone(conn = h2o.getConnection())
```

Arguments

conn	An H2OConnection object.

h2o.giniCoef *Retrieve the GINI Coefficcient*

Description

Retrieves the GINI coefficient from an H2OBinomialMetrics.

Usage

```
h2o.giniCoef(object, ...)
```

Arguments

object	an H2OBinomialMetrics object.
...	extra arguments to be passed if 'object' is of type H2OModel (e.g. train=TRUE)

See Also

h2o.auc for AUC, h2o.giniCoef for the GINI coefficient, and h2o.metric for the various. See h2o.performance for creating H2OModelMetrics objects. threshold metrics.

Examples

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.giniCoef(perf)
```

h2o.glm *H2O Generalized Linear Models*

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, training_frame, model_id, validation_frame, max_iterations = 50,
  beta_epsilon = 0, solver = c("IRLSM", "L_BFGS"), standardize = TRUE,
  family = c("gaussian", "binomial", "poisson", "gamma", "tweedie"),
  link = c("family_default", "identity", "logit", "log", "inverse",
  "tweedie"), tweedie_variance_power = NaN, tweedie_link_power = NaN,
  alpha = 0.5, prior = 0, lambda = 1e-05, lambda_search = FALSE,
  nlambdas = -1, lambda_min_ratio = -1, use_all_factor_levels = FALSE,
  nfolds, beta_constraints = NULL, ...)
```

Arguments

x	A vector containing the names or indices of the predictor variables to use in building the GLM model.
y	A character string or index that represent the response variable in the model.
training_frame	An H2OFrame object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
validation_frame	An H2OFrame object containing the variables in the model.
max_iterations	A non-negative integer specifying the maximum number of iterations.
beta_epsilon	A non-negative number specifying the magnitude of the maximum difference between the coefficient estimates from successive iterations. Defines the convergence criterion for h2o.glm.
solver	A character string specifying the solver used: IRLSM (supports more features), L_BFGS (scales better for datasets with many columns)
standardize	A logical value indicating whether the numeric predictors should be standardized to have a mean of 0 and a variance of 1 prior to training the models.
family	A character string specifying the distribution of the model: gaussian, binomial, poisson, gamma, tweedie.
link	A character string specifying the link function. The default is the canonical link for the family. The supported links for each of the family specifications are: "gaussian": "identity", "log", "inverse" "binomial": "logit", "log" "poisson": "log", "identity" "gamma": "inverse", "log", "identity" "tweedie": "tweedie"
tweedie_variance_power	A numeric specifying the power for the variance function when family = "tweedie".
tweedie_link_power	A numeric specifying the power for the link function when family = "tweedie".
alpha	A numeric in [0, 1] specifying the elastic-net mixing parameter. The elastic-net penalty is defined to be: $$P(\alpha,\beta) = (1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1 = \sum_j [(1-\alpha)/2\beta_j^2 + \alpha\|\beta_j\|]$$

, making `alpha = 1` the lasso penalty and `alpha = 0` the ridge penalty.

`prior` (Optional) A numeric specifying the prior probability of class 1 in the response when `family = "binomial"`. The default prior is the observational frequency of class 1.

`lambda` A non-negative shrinkage parameter for the elastic-net, which multiplies $P(\alpha, \beta)$ in the objective function. When `lambda = 0`, no elastic-net penalty is applied and ordinary generalized linear models are fit.

`lambda_search` A logical value indicating whether to conduct a search over the space of lambda values starting from the lambda max, given `lambda` is interpreted as lambda min.

`nlambdas` The number of lambda values to use when `lambda_search = TRUE`.

`lambda_min_ratio`
Smallest value for lambda as a fraction of lambda.max. By default if the number of observations is greater than the the number of variables then `lambda_min_ratio` = 0.0001; if the number of observations is less than the number of variables then `lambda_min_ratio` = 0.01.

`use_all_factor_levels`
A logical value indicating whether dummy variables should be used for all factor levels of the categorical predictors. When `TRUE`, results in an over parameterized models.

`nfolds` (Currently Unimplemented)

`beta_constraints`
A data.frame or H2OParsedData object with the columns ["names", "lower_bounds", "upper_bounds", "beta_given"], where each row corresponds to a predictor in the GLM. "names" contains the predictor names, "lower"/"upper_bounds", are the lower and upper bounds of beta, and "beta_given" is some supplied starting values for the coefficients.

`...` (Currently Unimplemented)

See Also

`predict.H2OModel` for prediction.

`h2o.group_by` *Group and Apply by Column*

Description

Performs a group by and apply similar to ddply.

Usage

```
h2o.group_by(data, by, ..., gb.control = list(na.methods = NULL, col.names =
  NULL))
```

Arguments

data	an H2OFrame object.
by	a list of column names
gb.control	a list of how to handle NA values in the dataset as well as how to name output columns. See Details: for more help.
...	any supported aggregate function.

Details

In the case of na.methods within gb.control, there are three possible settings. "all" will include NAs in computation of functions. "rm" will completely remove all NA fields. "ignore" will remove NAs from the numerator but keep the rows for computational purposes. If a list smaller than the number of columns groups is supplied, the list will be padded by "ignore".

Similar to na.methods, col.names will pad the list with the default column names if the length is less than the number of colums groups supplied.

Value

Returns a new H2OFrame object with columns equivalent to the number of groups created

---

h2o.head *Return the Head or Tail of an H2O Dataset.*

Description

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

Usage

```
## S4 method for signature H2OFrame
head(x, n = 6L, ...)

## S4 method for signature H2OFrame
tail(x, n = 6L, ...)
```

Arguments

x	An H2OFrame 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.
...	Further arguments passed to or from other methods.

Value

A data frame containing the first or last n rows of an H2OFrame object.

Examples

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

---

h2o.hit_ratio_table *Retrieve the Hit Ratios*

Description

Retrieve the Hit Ratios

Usage

```
h2o.hit_ratio_table(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	An H2OModel object.
train	Retrieve the training metric.
valid	Retreive the validation metric.
...	further arguments to be passed on (currently unimplemented)

---

h2o.ifelse *H2O Apply Conditional Statement*

Description

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

Usage

```
## S4 method for signature H2OFrame,ANY,ANY
ifelse(test, yes, no)

## S4 method for signature ANY,H2OFrame,H2OFrame
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 funciton and returned values cannot be categorical in nature.

Value

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

Examples

```
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)
```

---

h2o.importFile *Import Files into H2O*

Description

Imports files into an H2O cloud. The default behavior is to pass-through to the parse phase automatically.

Usage

```
h2o.importFolder(path, conn = h2o.getConnection(), pattern = "",
  destination_frame = "", parse = TRUE, header = NA, sep = "",
  col.names = NULL)

h2o.importURL(path, conn = h2o.getConnection(), destination_frame = "",
  parse = TRUE, header = NA, sep = "", col.names = NULL)

h2o.importHDFS(path, conn = h2o.getConnection(), pattern = "",
  destination_frame = "", parse = TRUE, header = NA, sep = "",
  col.names = NULL)

h2o.uploadFile(path, conn = h2o.getConnection(), destination_frame = "",
  parse = TRUE, header = NA, sep = "", col.names = NULL,
  col.types = NULL)
```

Arguments

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.
conn	an H2OConnection class object.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
destination_frame	(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 `sep = ""`, the parser will automatically detect the separator.
col.names	(Optional) A H2ORawData or H2OFrame (`version = 2`) object containing a single delimited line with the column names for the file.
col.types	(Optional) A vector to specify whether columns should be forced to a certain type upon import parsing.

Details

Other than `h2o.uploadFile`, if the given path is relative, then it will be relative to the start location of the H2O instance. Addtionally, the file must be on the same machine as the H2O cloud. In the case of `h2o.uploadFile`, a relative path will resolve relative to the working directory of the current R session.

Import an entire directory of files. If the given path is relative, then it will be relative to the start location of the H2O instance. The default behavior is to pass-through to the parse phase automatically.

`h2o.importURL` and `h2o.importHDFS` are both deprecated functions. Instead, use `h2o.importFile`

Examples

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

h2o.init *Initialize and Connect to H2O*

Description

Attempts to start and/or connect to and H2O instance.

Usage

```
h2o.init(ip = "127.0.0.1", port = 54321, startH2O = TRUE,
  forceDL = FALSE, Xmx, beta = FALSE, assertion = TRUE, license = NULL,
  nthreads = -2, max_mem_size = NULL, min_mem_size = NULL,
  ice_root = tempdir(), strict_version_check = FALSE)
```

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 try to start H2O from R if no connection with H2O is detected. This is only possible if ip = "localhost" or ip = "127.0.0.1". If an existing connection is detected, R does not start H2O.
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. This value is only used when R starts H2O.
Xmx	(Optional) (DEPRECATED) A character 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. This value is only used when R starts H2O.
beta	(Optional) A logical value indicating whether H2O should launch in beta mode. This value is only used when R starts H2O.
assertion	(Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes. This value is only used when R starts H2O.
license	(Optional) A character string value specifying the full path of the license file. This value is only used when R starts H2O.
nthreads	(Optional) Number of threads in the thread pool. This relates very closely to the number of CPUs used. -2 means use the CRAN default of 2 CPUs. -1 means use all CPUs on the host. A positive integer specifies the number of CPUs directly. This value is only used when R starts H2O.
max_mem_size	(Optional) A character 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. This value is only used when R starts H2O.

`min_mem_size` (Optional) A `character` string specifying the minimum 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. This value is only used when R starts H2O.

`ice_root` (Optional) A directory to handle object spillage. The defaul varies by OS.

`strict_version_check` (Optional) Setting this to FALSE is unsupported and should only be done when advised by technical support.

Details

By defualt, this method first checks if an H2O instance is connectible. If it cannot connect and `start = TRUE` with `ip = "localhost"`, it will attempt to start and instance of H2O at localhost:54321. Otherwise it stops 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

this method will load it and return a `H2OConnection` object containing the IP address and port number of the H2O server.

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 R package documentation for more details. `h2o.shutdown` for shutting down from R.

Examples

```
## Not run:
# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R with the default settings.
localH2O = h2o.init()

# Try to connect to a local H2O instance.
# If not found, raise an error.
localH2O = h2o.init(startH2O = FALSE)

# Try to connect to a local H2O instance that is already running.
```

```
# If not found, start a local H2O instance from R with 5 gigabytes of memory.
localH2O = h2o.init(max_mem_size = "5g")

# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R that uses 5 gigabytes of memory.
localH2O = h2o.init(max_mem_size = "5g")

## End(Not run)
```

`h2o.insertMissingValues` *Inserting Missing Values to an H2O DataFrame*

Description

This is primarily used for testing. Randomly replaces a user-specified fraction of entries in a H2O dataset with missing values.

Usage

```
h2o.insertMissingValues(data, fraction = 0.1, seed = -1)
```

Arguments

`data`	An H2OFrame object representing the dataset.
`fraction`	A number between 0 and 1 indicating the fraction of entries to replace with missing.
`seed`	A random number used to select which entries to replace with missing values. Default of `seed = -1` will automatically generate a seed in H2O.

WARNING

This will modify the original dataset. Unless this is intended, this function should only be called on a subset of the original.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.importFile(localH2O, path = irisPath)
summary(iris.hex)
irismiss.hex <- h2o.insertMissingValues(iris.hex, fraction = 0.25)
head(irismiss.hex)
summary(irismiss.hex)
```

h2o.killMinus3 *Dump the stack into the JVM's stdout.*

Description

A poor man's profiler, but effective.

Usage

```
h2o.killMinus3(conn = h2o.getConnection())
```

Arguments

conn	an H2OConnection class object.

h2o.kmeans *KMeans Model in H2O*

Description

Performs k-means clustering on an H2O dataset.

Usage

```
h2o.kmeans(training_frame, x, k, model_id, max_iterations = 1000,
  standardize = TRUE, init = c("Furthest", "Random", "PlusPlus"), seed)
```

Arguments

training_frame	An H2OFrame object containing the variables in the model.
x	(Optional) A vector containing the data columns on which k-means operates.
k	The number of clusters. Must be between 1 and 1e7 inclusive. k may be omitted if the user specifies the initial centers in the init parameter. If k is not omitted, in this case, then it should be equal to the number of user-specified centers.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
max_iterations	The maximum number of iterations allowed. Must be between 0
standardize	Logical, indicates whether the data should be standardized before running k-means.
init	A character string that selects the initial set of k cluster centers. Possible values are "Random": for random initialization, "PlusPlus": for k-means plus initialization, or "Furthest": for initialization at the furthest point from each successive center. Additionally, the user may specify a the initial centers as a matrix, data.frame, H2OFrame, or list of vectors. For matrices, data.frames, and H2OFrames, each row of the respective structure is an initial center. For lists of vectors, each vector is an initial center.

seed (Optional) Random seed used to initialize the cluster centroids.

Value

Returns an object of class H2OClusteringModel.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.kmeans(training_frame = prostate.hex, k = 10, x = c("AGE", "RACE", "VOL", "GLEASON"))
```

---

h2o.length *Returns the Length of a Parsed H2O Data Object.*

Description

Returns the length of an H2OFrame

Usage

```
## S4 method for signature H2OFrame
length(x)
```

Arguments

x An H2OFrame object.

See Also

length for the base R method.

Examples

```
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
length(iris.hex)
```

h2o.levels *Return the levels from the column requested column.*

Description

Return the levels from the column requested column.

Arguments

x	An H2OFrame object.
i	The index of the column whose domain is to be returned.

See Also

levels for the base R method.

Examples

```
localH2O <- h2o.init()
iris.hex <- as.h2o(localH2O, iris)
h2o.levels(iris.hex, 5)  # returns "setosa"     "versicolor" "virginica"
```

h2o.listTimezones *List all of the Time Zones Acceptable by the H2O Cloud.*

Description

List all of the Time Zones Acceptable by the H2O Cloud.

Usage

```
h2o.listTimezones(conn = h2o.getConnection())
```

Arguments

conn	An H2OConnection object.

h2o.loadModel *Load H2O Model from HDFS or Local Disk*

Description

Load a saved H2O model from disk.

Usage

```
h2o.loadModel(path, conn = h2o.getConnection())
```

Arguments

path	The path of the H2O Model to be imported.
conn	an H2OConnection object contianing the IP address and port of the server running H2O.

Value

Returns a H2OModel object of the class corresponding to the type of model built.

See Also

h2o.saveModel, H2OModel

Examples

```
## Not run:
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, destination_frame = "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 *Log a message on the server-side logs*

Description

This is helpful when running several pieces of work one after the other on a single H2O cluster and you want to make a notation in the H2O server side log where one piece of work ends and the next piece of work begins.

Usage

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

Arguments

message	A character string with the message to write to the log.
conn	An H2OConnection object pointing to a running H2O cluster.

Details

h2o.logAndEcho sends a message to H2O for logging. Generally used for debugging purposes.

See Also

H2OConnection

h2o.logloss *Retrieve the Log Loss Value*

Description

Retrieves the log loss output for a H2OBinomialMetrics or H2OMultinomialMetrics object

Usage

```
h2o.logloss(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	a H2OModelMetrics object of the correct type.
train	Retrieve the training metric.
valid	Retreive the validation metric.
...	Extra arguments to be passed if 'object' is of type H2OModel (e.g. train=TRUE)

h2o.ls *List Keys on an H2O Cluster*

Description

Accesses a list of object keys in the running instance of H2O.

Usage

```
h2o.ls(conn = h2o.getConnection())
```

Arguments

conn	An H2OConnection object containing the IP address and port number of the H2O server.

Value

Returns a list of hex keys in the current H2O instance.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.ls(localH2O)
```

h2o.makeGLMModel *Remake an H2O GLM Model*

Description

This function allows the usage of new beta constraints to create an GLM model, from an existing model.

Usage

```
h2o.makeGLMModel(model, beta)
```

Arguments

model	an H2OModel corresponding from a h2o.glm call.
beta	a new set of beta_constraints

h2o.match *Value Matching in H2O*

Description

match and %in% return values similar to the base R generic functions.

Usage

```
## S4 method for signature H2OFrame
match(x, table, nomatch = 0, incomparables = NULL)

## S4 method for signature H2OFrame
x %in% table
```

Arguments

x	a categorical vector from an H2OFrame object with values to be matched.
table	an R object to match x against.
nomatch	the value to be returned in the case when no match is found.
incomparables	a vector of calues that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value.

See Also

match for base R implementation.

Examples

```
h2o.init()
hex <- as.h2o(iris)
match(hex[,5], c("setosa", "versicola"))
```

h2o.mean *Mean of a column*

Description

Obtain the mean of a column of a parsed H2O data object.

Usage

```
## S4 method for signature H2OFrame
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An H2OFrame object.
trim	The fraction (0 to 0.5) of observations to trim from each end of x before the mean is computed.
na.rm	A logical value indicating whether NA or missing values should be stripped before the computation.
...	Further arguments to be passed from or to other methods.

See Also

mean for the base R implementation.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
mean(prostate.hex$AGE)
```

---

h2o.merge *Merge Two H2O Data Frames*

Description

Merges two H2OFrame objects by shared column names. Unlike the base R implementation, h2o.merge only supports merging through shared column names.

Usage

```
h2o.merge(x, y, all.x = FALSE, all.y = FALSE)
```

Arguments

x,y	H2OFrame objects
all.x	a logical value indicating whether or not shared values are preserved or ignored in x.
all.y	a logical value indicating whether or not shared values are preserved or ignored in y.

Details

In order for h2o.merge to work in multinode clusters, one of the datasets must be small enough to exist in every node. Currently, this function only supports all.x = TRUE. All other permutations will fail.

Examples

```
h2o.init()
left <- data.frame(fruit = c(apple, orange, banana, lemon, strawberry, blueberry),
color = c(red, orange, yellow, yellow, red, blue))
right <- data.frame(fruit = c(apple, orange, banana, lemon, strawberry, watermelon),
citrus = c(FALSE, TRUE, FALSE, TRUE, FALSE, FALSE))
l.hex <- as.h2o(left)
r.hex <- as.h2o(right)
left.hex <- h2o.merge(l.hex, r.hex, all.x = TRUE)
```

---

h2o.metric *H2O Model Metric Accessor Functions*

Description

A series of functions that retrieve model metric details.

Usage

```
h2o.metric(object, thresholds, metric)

h2o.F0point5(object, thresholds)

h2o.F1(object, thresholds)

h2o.F2(object, thresholds)

h2o.accuracy(object, thresholds)

h2o.error(object, thresholds)

h2o.maxPerClassError(object, thresholds)

h2o.mcc(object, thresholds)

h2o.precision(object, thresholds)

h2o.tpr(object, thresholds)

h2o.fpr(object, thresholds)

h2o.fnr(object, thresholds)

h2o.tnr(object, thresholds)

h2o.recall(object, thresholds)
```

```
h2o.sensitivity(object, thresholds)

h2o.fallout(object, thresholds)

h2o.missrate(object, thresholds)

h2o.specificity(object, thresholds)
```

Arguments

object	An H2OModelMetrics object of the correct type.
thresholds	A value or a list of values between 0.0 and 1.0.
metric	A specified paramter to retrieve.

Details

Many of these functions have an optional thresholds parameter. Currently only increments of 0.1 are allowed. If not specified, the functions will return all possible values. Otherwise, the function will return the value for the indicated threshold.

Currently, the these functions are only supported by H2OBinomialMetrics objects.

Value

Returns either a single value, or a list of values.

See Also

h2o.auc for AUC, h2o.giniCoef for the GINI coefficient, and h2o.mse for MSE. See h2o.performance for creating H2OModelMetrics objects.

Examples

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.F1(perf)
```

h2o.month *Convert Milliseconds to Months in H2O Datasets*

Description

Converts the entries of a H2OFrame object from milliseconds to months (on a 0 to 11) scale.

Usage

```
h2o.month(x)

month(x)

## S3 method for class H2OFrame
month(x)
```

Arguments

x	An H2OFrame object.

Value

A H2OFrame object containing the entries of x converted to months of the year.

See Also

h2o.year

h2o.mse *Retrieves Mean Squared Error Value*

Description

Retrieves the mean squared error value from an H2OModelMetrics object.

Usage

```
h2o.mse(object, train = FALSE, valid = FALSE, ...)
```

Arguments

object	An H2OModelMetrics object of the correct type.
train	Retrieve the training metric.
valid	Retreive the validation metric.
...	Extra arguments to be passed if 'object' is of type H2OModel (e.g. train=TRUE)

Details

This function only supports H2OBinomialMetrics, H2OMultinomialMetrics, and H2ORegressionMetrics objects.

See Also

h2o.auc for AUC, h2o.mse for MSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples

```
library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.mse(perf)
```

---

h2o.naiveBayes *Naive Bayes Model in H2O*

Description

Compute naive Bayes probabilities on an H2O dataset.

Usage

```
h2o.naiveBayes(x, y, training_frame, model_id, laplace = 0,
  threshold = 0.001, eps = 0)
```

Arguments

x	A vector containing the names or indices of the predictor variables to use in building the 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.
training_frame	An H2OFrame object containing the variables in the model.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
laplace	A positive number controlling Laplace smoothing. The default zero disables smoothing.
threshold	The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.
eps	A threshold cutoff to deal with numeric instability, must be positive.

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

Returns an object of class H2OModel.

Examples

```
localH2O <- h2o.init()
votesPath <- system.file("extdata", "housevotes.csv", package="h2o")
votes.hex <- h2o.uploadFile(localH2O, path = votesPath, header = TRUE)
h2o.naiveBayes(x = 2:17, y = 1, training_frame = votes.hex, laplace = 3)
```

---

h2o.networkTest *View Network Traffic Speed*

Description

View speed with various file sizes.

Usage

```
h2o.networkTest(conn = h2o.getConnection())
```

Arguments

conn	an H2OConnection object.

Value

Returns a table listing the network speed for 1B, 10KB, and 10MB.

h2o.nrow *The Number of Rows/Columns of an H2O Dataset*

Description

Returns a count of the number of rows or columns in an H2OFrame object.

Usage

```
## S4 method for signature H2OFrame
nrow(x)

## S4 method for signature H2OFrame
ncol(x)
```

Arguments

x	An H2OFrame object.

See Also

dim for all the dimensions. nrow for the default R method.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
nrow(iris.hex)
ncol(iris.hex)
```

h2o.num_iterations *Retrieve the number of iterations.*

Description

Retrieve the number of iterations.

Usage

```
h2o.num_iterations(object)
```

Arguments

object	An H2OClusteringModel object.
...	further arguments to be passed on (currently unimplemented)

h2o.openLog *View H2O R Logs*

Description

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

Usage

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

Arguments

type	Currently unimplemented.

See Also

h2o.startLogging, h2o.stopLogging, h2o.clearLog

Examples

```
## Not run:
# Skip running this to avoid windows being opened during R CMD check
library(h2o)
localH2O = h2o.init()

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 *H2O Data Parsing*

Description

The second phase in the data ingestion step.

Usage

```
h2o.parseRaw(data, destination_frame = "", header = NA, sep = "",
  col.names = NULL, col.types = NULL)
```

Arguments

data	An H2ORawData object to be parsed.
destination_frame	(Optional) The hex key assigned to the parsed file.
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 H2OFrame object containing a single delimited line with the column names for the file.
col.types	(Optional) A vector specifying the types to attempt to force over columns.

Details

Parse the Raw Data produced by the import phase.

---

h2o.parseSetup *Get a parse setup back for the staged data.*

Description

Get a parse setup back for the staged data.

Usage

```
h2o.parseSetup(data, destination_frame = "", header = NA, sep = "",
  col.names = NULL, col.types = NULL)
```

Arguments

data	An H2ORawData object to be parsed.
destination_frame	(Optional) The hex key assigned to the parsed file.
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 H2OFrame object containing a single delimited line with the column names for the file.
col.types	(Optional) A vector specifying the types to attempt to force over columns.

h2o.performance *Model Performance Metrics in H2O*

Description

Given a trained h2o model, compute its performance on the given dataset

Usage

```
h2o.performance(model, data = NULL, train = FALSE, valid = FALSE, ...)
```

Arguments

model	An H2OModel object
data	An H2OFrame. The model will make predictions on this dataset, and subsequently score them. The dataset should match the dataset that was used to train the model, in terms of column names, types, and dimensions. If data is passed in, then train and valid are ignored.
train	A logical value indicating whether to return the training metrics (constructed during training).
valid	A logical value indicating whether to return the validation metrics (constructed during training).
...	Extra args passed in for use by other functions.

Value

Returns an object of the H2OModelMetrics subclass.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.hex$CAPSULE <- as.factor(prostate.hex$CAPSULE)
prostate.gbm <- h2o.gbm(3:9, "CAPSULE", prostate.hex)
h2o.performance(model = prostate.gbm, data=prostate.hex)
```

h2o.prcomp *Quadratically Regularized PCA Model in H2O*

Description

Principal components analysis with quadratic regularization of an H2O dataset.

Usage

```
h2o.prcomp(training_frame, x, k, center = TRUE, scale. = FALSE, model_id,
  gamma = 0, max_iterations = 1000, init = "PlusPlus", seed)
```

Arguments

training_frame	An H2OFrame object containing the variables in the model.
x	(Optional) A vector containing the data columns on which k-means operates.
k	The rank of the resulting decomposition. Must be between 1 and 1e7 inclusive. k may be omitted if the user specifies the initial centers in the init parameter. If k is not omitted, in this case, then it should be equal to the number of user-specified centers.
center	logical, indicating whether variables should be shifted or zero centered
scale.	logical, indicating whether or not to scale to unit variance
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
gamma	Quadratic regularization weight, should be greater than 0.
max_iterations	The maximum number of iterations to run alternating minimization. Must be between 0 and 1e6 inclusive.
init	A character string that selects the initial set of k cluster centers. Possible values are "PlusPlus": for k-means++ initialization, or a user-specified initial Y as a matrix, data.frame, H2OFrame, or list of vectors.
seed	(Optional) Random seed used to initialize the cluster centroids with k-means++ initialization.

Value

Returns an object of class H2ODimReductionModel.

Examples

```
library(h2o)
localH2O <- h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(localH2O, path = ausPath)
h2o.prcomp(training_frame = australia.hex, k = 8, center = TRUE, scale. = TRUE)
```

`h2o.randomForest` *Build a Big Data Random Forest Model*

Description

Builds a Random Forest Model on an H2OFrame

Usage

```
h2o.randomForest(x, y, training_frame, model_id, validation_frame,
  mtries = -1, sample_rate = 2/3, build_tree_one_node = FALSE,
  ntrees = 50, max_depth = 20, min_rows = 1, nbins = 20,
  balance_classes = FALSE, max_after_balance_size = 5, seed, ...)
```

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 1, and increasing from left to right. (The response must be either an integer or a categorical variable).
`training_frame`	An `H2OFrame` object containing the variables in the model.
`model_id`	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
`validation_frame`	An `H2OFrame` object containing the variables in the model.
`mtries`	Columns to randomly select at each level, or -1 for sqrt(#cols).
`sample_rate`	Sample rate, from 0. to 1.0.
`build_tree_one_node`	Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets.
`ntrees`	A nonnegative integer that determines the number of trees to grow.
`max_depth`	Maximum depth to grow the tree.
`min_rows`	Minimum number of rows to assign to teminal nodes.
`nbins`	Number of bins to use in building histogram.
`balance_classes`	logical, indicates whether or not to 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)
`seed`	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
`...`	(Currently Unimplemented)

Value

Creates a H2OModel object of the right type.

See Also

`predict.H2OModel` for prediction.

`h2o.rbind` *Combine H2O Datasets by Rows*

Description

Takes a sequence of H2O data sets and combines them by rows

Usage

```
h2o.rbind(...)
```

Arguments

`...`	A sequence of H2OFrame arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.

Value

An H2OFrame object containing the combined ... arguments column-wise.

See Also

`rbind` for the base R method.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
prostate.cbind <- h2o.rbind(prostate.hex, prostate.hex)
head(prostate.cbind)
```

h2o.removeAll *Remove All Objects on the H2O Cluster*

Description

Removes the data from the h2o cluster, but does not remove the local references.

Usage

```
h2o.removeAll(conn = h2o.getConnection(), timeout_secs = 0)
```

Arguments

conn	An H2OConnection object containing the IP address and port number of the H2O server.
timeout_secs	Timeout in seconds. Default is no timeout.

See Also

h2o.rm

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
h2o.ls(localH2O)
h2o.removeAll(localH2O)
h2o.ls(localH2O)
```

h2o.rep_len *Replicate Elements of Vectors or Lists into H2O*

Description

h2o.rep performs just as rep does. It replicates the values in x in the H2O backend.

Usage

```
h2o.rep_len(x, length.out)
```

Arguments

x	a vector (of any mode including a list) or a factor
length.out	non negative integer. The desired length of the output vector.

Value

Creates a H2OFrame vector of the same type as x

`h2o.rm` *Delete Objects In H2O*

Description

Remove the h2o Big Data object(s) having the key name(s) from ids.

Usage

```
h2o.rm(ids, conn = h2o.getConnection())
```

Arguments

`ids`	The hex key associated with the object to be removed.
`conn`	An H2OConnection object containing the IP address and port number of the H2O server.

See Also

`h2o.assign`, `h2o.ls`

`h2o.runif` *Produe a Vector of Random Uniform Numbers*

Description

Creates a vector of random uniform numbers equal in length to the length of the specified H2O dataset.

Usage

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

Arguments

`x`	An H2OFrame object.
`seed`	A random seed used to generate draws from the uniform distribution.

Value

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

Examples

```
library(h2o)
localH2O = h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(localH2O, path = prosPath, destination_frame = "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 *Save an H2O Model Object to Disk*

Description

Save an H2OModel to disk.

Usage

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

Arguments

object	an H2OModel object.
dir	string indicating the directory the model will be written to.
name	string name of the file.
filename	the full path to the file.
force	logical, indicates how to deal with files that already exist.

Details

In the case of existing files `forse = TRUE` will overwrite the file. Otherwise, the operation will fail.

See Also

`h2o.loadModel` for loading a model to H2O from disk

Examples

```
## Not run:
library(h2o)
localH2O <- h2o.init()
prostate.hex <- h2o.uploadFile(localH2O, path = paste("https://raw.github.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.scale *Scaling and Centering of an H2O Frame*

Description

Centers and/or scales the columns of an H2O dataset.

Usage

```
## S3 method for class H2OFrame
scale(x, center = TRUE, scale = TRUE)
```

Arguments

x	An H2OFrame object.
center	either a logical value or numeric vector of length equal to the number of columns of x.
scale	either a logical value or numeric vector of length equal to the number of columns of x.

Examples

```
library(h2o)
localH2O <- h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

# Scale and center all the numeric columns in iris data set
scale(iris.hex[, 1:4])
```

h2o.scoreHistory *Retrieve Model Score History*

Description

Retrieve Model Score History

Usage

```
h2o.scoreHistory(object, ...)
```

Arguments

object	An H2OModel object.
...	further arguments to be passed on (currently unimplemented)

h2o.sd *Standard Deviation of a column of data.*

Description

Obtain the standard deviation of a column of data.

Usage

```
## S4 method for signature H2OFrame
sd(x, na.rm = FALSE)
```

Arguments

x	An H2OFrame object.
na.rm	logical. Should missing values be removed?

See Also

h2o.var for variance, and sd for the base R implementation.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
sd(prostate.hex$AGE)
```

h2o.setLevel *Set a Factor Column to a Level*

Description

A method to set a factor column to one of the levels.

Usage

```
h2o.setLevel(x, level)
```

Arguments

x	a column from an H2OFrame object.
level	The level at which the column will be set.

Details

Replace all other occurences with 'level' in a factor column.

Value

An object of class H2OFrame.

Examples

```
localH2O <- h2o.init()
hex <- as.h2o(localH2O  , iris)
hex$Species <- h2o.setLevel(hex$Species, "versicolor")
```

h2o.setTimezone *Set the Time Zone on the H2O Cloud*

Description

Set the Time Zone on the H2O Cloud

Usage

```
h2o.setTimezone(tz, conn = h2o.getConnection())
```

Arguments

tz	The desired timezone.
conn	An H2OConnection object.

h2o.shim *Deprecated Script Shim*

Description

Due to the many improvements implemented in H2O-Dev and the differences in architecture between H2O and H2O-Dev, some parameters, options, and objects are no longer supported. To assist our legacy H2O users in upgrading their workflows for compatibility with H2O-Dev, we have developed the "Deprecated Script Shim" tool to detect deprecated parameters, options, and objects in H2O scripts being imported into H2O-Dev and suggest updated alternatives.

Usage

```
h2o.shim(enable = TRUE)
```

Arguments

enable	a logical value indicating whether the shim should be enabled or disabled.

See Also

https://github.com/h2oai/h2o-dev/blob/master/h2o-docs/src/product/upgrade/H2ODevPortingRScripts.md, For more information on converting legacy H2O scripts so that they will run in H2O-Dev

h2o.shutdown *Shut Down H2O Instance*

Description

Shut down the specified instance. All data will be lost.

Usage

```
h2o.shutdown(conn = h2o.getConnection(), prompt = TRUE)
```

Arguments

conn	An H2OConnection object containing the IP address and port of the server running H2O.
prompt	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

```
# Dont run automatically to prevent accidentally shutting down a cloud
## Not run:
library(h2o)
localH2O = h2o.init()
h2o.shutdown(localH2O)

## End(Not run)
```

---

h2o.splitFrame *Split an H2O Data Set*

Description

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

Usage

```
h2o.splitFrame(data, ratios = 0.75, destination_frames)
```

Arguments

data	An H2OFrame object representing the dataste to split.
ratios	A numeric value or array indicating the ratio of total rows contained in each split. Must total up to less than 1.
destination_frames	An array of frame IDs equal to the number of ratios specified plus one.

Examples

```
library(h2o)
localH2O = h2o.init()
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.startGLMJob` *Start an H2O Generalized Linear Model Job*

Description

Creates a background H2O GLM job.

Usage

```
h2o.startGLMJob(x, y, training_frame, model_id, validation_frame,
  max_iterations = 50, beta_epsilon = 0, solver = c("IRLSM", "L_BFGS"),
  standardize = TRUE, family = c("gaussian", "binomial", "poisson", "gamma",
  "tweedie"), link = c("family_default", "identity", "logit", "log",
  "inverse", "tweedie"), tweedie_variance_power = NaN,
  tweedie_link_power = NaN, alpha = 0.5, prior = 0, lambda = 1e-05,
  lambda_search = FALSE, nlambdas = -1, lambda_min_ratio = 1,
  use_all_factor_levels = FALSE, nfolds = 0, beta_constraints = NULL, ...)
```

Arguments

`x`	A vector containing the names or indices of the predictor variables to use in building the GLM model.
`y`	A character string or index that represent the response variable in the model.
`training_frame`	An `H2OFrame` object containing the variables in the model.
`model_id`	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
`validation_frame`	An `H2OFrame` object containing the variables in the model.
`max_iterations`	A non-negative integer specifying the maximum number of iterations.
`beta_epsilon`	A non-negative number specifying the magnitude of the maximum difference between the coefficient estimates from successive iterations. Defines the convergence criterion for `h2o.glm`.
`solver`	A character string specifying the solver used: IRLSM (supports more features), L_BFGS (scales better for datasets with many columns)
`standardize`	A logical value indicating whether the numeric predictors should be standardized to have a mean of 0 and a variance of 1 prior to training the models.

`family` A character string specifying the distribution of the model: gaussian, binomial, poisson, gamma, tweedie.

`link` A character string specifying the link function. The default is the canonical link for the `family`. The supported links for each of the `family` specifications are:
`"gaussian"`: `"identity"`, `"log"`, `"inverse"`
`"binomial"`: `"logit"`, `"log"`
`"poisson"`: `"log"`, `"identity"`
`"gamma"`: `"inverse"`, `"log"`, `"identity"`
`"tweedie"`: `"tweedie"`

`tweedie_variance_power`
A numeric specifying the power for the variance function when `family = "tweedie"`.

`tweedie_link_power`
A numeric specifying the power for the link function when `family = "tweedie"`.

`alpha` A numeric in [0, 1] specifying the elastic-net mixing parameter. The elastic-net penalty is defined to be:

$$P(\alpha, \beta) = (1 - \alpha)/2||\beta||_2^2 + \alpha||\beta||_1 = \sum_j [(1 - \alpha)/2\beta_j^2 + \alpha|\beta_j|]$$

, making `alpha = 1` the lasso penalty and `alpha = 0` the ridge penalty.

`prior` (Optional) A numeric specifying the prior probability of class 1 in the response when `family = "binomial"`. The default prior is the observational frequency of class 1.

`lambda` A non-negative shrinkage parameter for the elastic-net, which multiplies $P(\alpha, \beta)$ in the objective function. When `lambda = 0`, no elastic-net penalty is applied and ordinary generalized linear models are fit.

`lambda_search` A logical value indicating whether to conduct a search over the space of lambda values starting from the lambda max, given `lambda` is interpreted as lambda min.

`nlambdas` The number of lambda values to use when `lambda_search = TRUE`.

`lambda_min_ratio`
Smallest value for lambda as a fraction of lambda.max. By default if the number of observations is greater than the the number of variables then `lambda_min_ratio` = 0.0001; if the number of observations is less than the number of variables then `lambda_min_ratio` = 0.01.

`use_all_factor_levels`
A logical value indicating whether dummy variables should be used for all factor levels of the categorical predictors. When `TRUE`, results in an over parameterized models.

`nfolds` (Currently Unimplemented)

`beta_constraints`
A data.frame or H2OParsedData object with the columns ["names", "lower_bounds", "upper_bounds", "beta_given"], where each row corresponds to a predictor in the GLM. "names" contains the predictor names, "lower"/"upper_bounds", are the lower and upper bounds of beta, and "beta_given" is some supplied starting values for the coefficients.

`...` (Currently Unimplemented)

Value

Returns a H2OModelFuture class object.

See Also

h2o.getGLMModel for resolving the H2OModelFuture object.

---

h2o.startLogging *Start Writing H2O R Logs*

Description

Begin logging H2o R POST commands and error responses to local disk. Used primarily for debuggin purposes.

Usage

```
h2o.startLogging(file)
```

Arguments

file	a character string name for the file, automatically generated

See Also

h2o.stopLogging, h2o.clearLog, h2o.openLog

Examples

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

h2o.stopLogging *Stop Writing H2O R Logs*

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.clearLog, h2o.openLog

Examples

```
library(h2o)
localH2O = h2o.init()
h2o.startLogging()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
h2o.stopLogging()
```

h2o.summary *Summarizes the columns of a H2O data frame.*

Description

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

Usage

```
## S4 method for signature H2OFrame
summary(object, ...)
```

Arguments

object	An H2OFrame object.
...	Further arguments passed to or from other methods.

Value

A table displaying the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column, and the levels and category counts of the levels in each categorical column.

Examples

```
library(h2o)
localH2O = h2o.init()
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])
```

`h2o.table` *Cross Tabulation and Table Creation in H2O*

Description

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

Usage

```
h2o.table(x, y = NULL)
```

Arguments

x	An H2OFrame object with at most two columns.
y	An H2OFrame similar to x, or NULL.

Value

Returns a tabulated H2OFrame object.

Examples

```
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
summary(prostate.hex)

# Counts of the ages of all patients
head(h2o.table(prostate.hex[,3]))
h2o.table(prostate.hex[,3])

# 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)])
```

h2o.var — *Variance of a column.*

Description

Obtain the variance of a column of a parsed H2O data object.

Usage

```
## S4 method for signature H2OFrame
var(x, y = NULL, na.rm = FALSE, use)
```

Arguments

x	An H2OFrame object.
y	NULL (default) or a column of an H2OFrame object. The default is equivalent to y = x (but more efficient).
na.rm	logical. Should missing values be removed?
use	An optional character string to be used in the presence of missing values. This must be one of the following strings. "everything", "all.obs", or "complete.obs".

See Also

var for the base R implementation. h2o.sd for standard deviation.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath)
var(prostate.hex$AGE)
```

h2o.varimp — *Retrieve the variable importance.*

Description

Retrieve the variable importance.

Usage

```
h2o.varimp(object, ...)
```

Arguments

object	An H2OModel object.
...	further arguments to be passed on (currently unimplemented)

h2o.within_mse *Get the Within MSE*

Description

Get the Within MSE

Usage

```
h2o.within_mse(object, ...)
```

Arguments

object	An H2OClusteringModel object.
...	further arguments to be passed on (currently unimplemented)

h2o.year *Convert Milliseconds to Years in H2O Datasets*

Description

Conver the entries of a H2OFrame object from milliseconds to years, indexed starting from 1900.

Usage

```
h2o.year(x)

year(x)

## S3 method for class H2OFrame
year(x)
```

Arguments

x	An H2OFrame object.

Details

This method calls the function of the MutableDateTime class in Java.

Value

A H2OFrame object containig the entries of x converted to years starting from 1900, e.g. 69 corresponds to the year 1969.

See Also

h2o.month

H2OConnection-class *The H2OConnection class.*

Description

This class represents a connection to an H2O cloud.

Usage

```
## S4 method for signature H2OConnection
show(object)
```

Arguments

object	an H2OConnection object.

Details

Because H2O is not a master-slave architecture, there is no restriction on which H2O node is used to establish the connection between R (the client) and H2O (the server).

A new H2O connection is established via the h2o.init() function, which takes as parameters the 'ip' and 'port' of the machine running an instance to connect with. The default behavior is to connect with a local instance of H2O at port 54321, or to boot a new local instance if one is not found at port 54321.

Slots

`ip` A `character` string specifying the IP address of the H2O cloud.

`port` A `numeric` value specifying the port number of the H2O cloud.

`mutable` An `H2OConnectionMutableState` object to hold the mutable state for the H2O connection.

H2OFrame-class *The H2OFrame class*

Description

The H2OFrame class

Usage

```
## S4 method for signature H2OFrame
show(object)
```

Arguments

object	An H2OConnection object.

Slots

conn An H2OConnection object specifying the connection to an H2O cloud.

frame_id A character string specifying the identifier for the frame in the H2O cloud.

finalizers A list object containing environments with finalizers that remove objects from the H2O cloud.

mutable An H2OFrameMutableState object to hold the mutable state for the H2O frame.

H2OFrame-Extract *Extract or Replace Parts of an H2OFrame Object*

Description

Operators to extract or replace parts of H2OFrame objects.

Usage

```
## S4 method for signature H2OFrame
x[i, j, ..., drop = TRUE]

## S4 method for signature H2OFrame
x$name

## S4 method for signature H2OFrame
x[[i, exact = TRUE]]

## S4 replacement method for signature H2OFrame
x[i, j, ...] <- value

## S4 replacement method for signature H2OFrame
x$name <- value

## S4 replacement method for signature H2OFrame
x[[i]] <- value
```

Arguments

x	object from which to extract element(s) or in which to replace element(s).
i,j,...	indices specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or will be matched to the names.
drop	a logical, whether or not to attempt to reduce dimensions to the lowest possible.
name	a literal character string or a name (possibly backtick quoted).

exact	controls possible partial matching of [[when extracting a character
value	an array-like H2O object similar to x.

H2OModel-class *The H2OModel object.*

Description

This virtual class represents a model built by H2O.

Usage

```
## S4 method for signature H2OModel
show(object)
```

Arguments

object	an H2OModel object.

Details

This object has slots for the key, which is a character string that points to the model key existing in the H2O cloud, the data used to build the model (an object of class H2OFrame).

Slots

conn Object of class H2OConnection, which is the client object that was passed into the function call.

key A character string specifying the key for the model fit in the H2O cloud's key-value store.

finalizers A list object containing environments with finalizers that remove keys from the H2O key-value store.

algorithm A character string specifying the algorithm that were used to fit the model.

parameters A list containing the parameter settings that were used to fit the model that differ from the defaults.

allparameters A list containg all parameters used to fit the model.

model A list containing the characteristics of the model returned by the algorithm.

H2OModelFuture-class *H2O Future Model*

Description

A class to contain the information for background model jobs.

Slots

`conn` an H2OConnection

`job_key` a character key representing the identification of the job process.

`model_id` the final identifier for the model

See Also

H2OModel for the final model types.

H2OModelMetrics-class *The H2OModelMetrics Object.*

Description

A class for constructing performance measures of H2O models.

Usage

```
## S4 method for signature H2OModelMetrics
show(object)

## S4 method for signature H2OBinomialMetrics
show(object)

## S4 method for signature H2OMultinomialMetrics
show(object)

## S4 method for signature H2ORegressionMetrics
show(object)

## S4 method for signature H2OClusteringMetrics
show(object)

## S4 method for signature H2OAutoEncoderMetrics
show(object)
```

Arguments

`object` An `H2OModelMetrics` object

H2OObject-class *The H2OObject class*

Description

The H2OObject class

Usage

```
## S4 method for signature H2OObject
initialize(.Object, ...)
```

Arguments

.Object	an H2OObject
...	additional parameters to pass on to functions

Slots

conn An H2OConnection object specifying the connection to an H2O cloud.

id A character string specifying the key in the H2O cloud's key-value store.

finalizers A list object containing environments with finalizers that remove keys from the H2O key-value store.

H2ORawData-class *The H2ORawData class.*

Description

This class represents data in a post-import format.

Usage

```
## S4 method for signature H2ORawData
show(object)
```

Arguments

object	a H2ORawData object.

Details

Data ingestion is a two-step process in H2O. First, a given path to a data source is _imported_ for validation by the user. The user may continue onto _parsing_ all of the data into memory, or the user may choose to back out and make corrections. Imported data is in a staging area such that H2O is aware of the data, but the data is not yet in memory.

The H2ORawData is a representation of the imported, not yet parsed, data.

Slots

`conn` An `H2OConnection` object containing the IP address and port number of the H2O server.

`frame_id` An object of class `"character"`, which is the name of the key assigned to the imported data.

`finalizers` A `list` object containing environments with finalizers that remove objects from the H2O cloud.

`H2OS4groupGeneric` *S4 Group Generic Functions for H2O*

Description

Methods for group generic functions and H2O objects.

Usage

```
## S4 method for signature missing,H2OFrame
Ops(e1, e2)

## S4 method for signature H2OFrame,missing
Ops(e1, e2)

## S4 method for signature H2OFrame,H2OFrame
Ops(e1, e2)

## S4 method for signature numeric,H2OFrame
Ops(e1, e2)

## S4 method for signature H2OFrame,numeric
Ops(e1, e2)

## S4 method for signature H2OFrame,character
Ops(e1, e2)

## S4 method for signature character,H2OFrame
Ops(e1, e2)

## S4 method for signature H2OFrame
Math(x)

## S4 method for signature H2OFrame
Math2(x, digits)

## S4 method for signature H2OFrame
Summary(x, ..., na.rm = FALSE)
```

```
## S4 method for signature H2OFrame
!x

## S4 method for signature H2OFrame
is.na(x)

## S4 method for signature H2OFrame
t(x)

## S4 method for signature H2OFrame
log(x, ...)

## S4 method for signature H2OFrame
trunc(x, ...)

## S4 method for signature H2OFrame,H2OFrame
x %*% y
```

Arguments

x,y,e1,e2	objects.
digits	number of digits to be used in round or signif
na.rm	logical: should missing values be removed?
...	further arguments passed to or from methods

H2OW2V-class *The H2OW2V object.*

Description

This class represents a h2o-word2vec object.

---

is.factor,H2OFrame-method *Is H2O Data Frame column a enum*

Description

Is H2O Data Frame column a enum

Usage

```
## S4 method for signature H2OFrame
is.factor(x)
```

Arguments

x	an H2OFrame object column.

Value

Returns logical value.

`median,H2OFrame-method` *H2O Median*

Description

Compute the airthmetic mean of a H2OFrame.

Usage

```
## S4 method for signature H2OFrame
median(x, na.rm = TRUE)
```

Arguments

x	An H2OFrame object.
na.rm	a logical, indicating whether na's are omitted.

Examples

```
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(localH2O, path = prosPath, destination_frame = "prostate.hex")
```

`ModelAccessors` *Accessor Methods for H2OModel Object*

Description

Function accessor methods for various H2O output fields.

Usage

```
getParms(object)

## S4 method for signature H2OModel
getParms(object)

getCenters(object)

getCentersStd(object)

getWithinMSE(object)

getAvgWithinSS(object)

getAvgBetweenSS(object)

getAvgSS(object)

getIterations(object)

getClusterSizes(object)

## S4 method for signature H2OClusteringModel
getCenters(object)

## S4 method for signature H2OClusteringModel
getCentersStd(object)

## S4 method for signature H2OClusteringModel
getWithinMSE(object)

## S4 method for signature H2OClusteringModel
getAvgWithinSS(object)

## S4 method for signature H2OClusteringModel
getAvgBetweenSS(object)

## S4 method for signature H2OClusteringModel
getAvgSS(object)

## S4 method for signature H2OClusteringModel
getIterations(object)

## S4 method for signature H2OClusteringModel
getClusterSizes(object)
```

Arguments

object	an H2OModel class object.

Node-class *The Node class.*

Description

An object of type Node inherits from an H2OFrame, but holds no H2O-aware data. Every node in the abstract syntax tree has as its ancestor this class.

This class represents an operator between one or more H2O objects. ASTApply nodes are always root nodes in a tree and are never leaf nodes. Operators are discussed more in depth in ops.R.

Details

Every node in the abstract syntax tree will have a symbol table, which is a dictionary of types and names for all the relevant variables and functions defined in the current scope. A missing symbol is therefore discovered by looking up the tree to the nearest symbol table defining that symbol.

---

predict.H2OModel *Predict on an H2O Model*

Description

Obtains predictions from various fitted H2O model objects.

Usage

```
## S3 method for class H2OModel
predict(object, newdata, ...)

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

Arguments

object	a fitted H2OModel object for which prediction is desired
newdata	A H2OFrame object in which to look for variables with which to predict.
...	additional arguments to pass on.

Details

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

Value

Returns an H2OFrame object with probabilites and default predictions.

See Also

`link{h2o.deeplearning}`, `link{h2o.gbm}`, `link{h2o.glm}`, `link{h2o.randomForest}` for model generation in h2o.

`print.H2OTable` *Print method for H2OTable objects*

Description

Print method for H2OTable objects

Usage

```
## S3 method for class H2OTable
print(x, ...)
```

Arguments

`x`	An H2OTable object
`...`	Further arguments passed to or from other methods.

Value

The original x object

`quantile` *Quantiles of H2O Data Frame.*

Description

Obtain and display quantiles for H2O parsed data.

Usage

```
## S3 method for class H2OFrame
quantile(x, probs = c(0.001, 0.01, 0.1, 0.25, 0.333, 0.5,
  0.667, 0.75, 0.9, 0.99, 0.999), combine_method = c("interpolate", "average",
  "avg", "low", "high"), ...)
```

Arguments

x	An H2OFrame object with a single numeric column.
probs	Numeric vector of probabilities with values in [0,1].
combine_method	How to combine quantiles for even sample sizes. Default is to do linear interpolation. E.g., If method is "lo", then it will take the lo value of the quantile. Abbreviations for average, low, and high are acceptable (avg, lo, hi).
...	Further arguments passed to or from other methods.

Details

quantile.H2OFrame, a method for the quantile generic. Obtain and return quantiles for an H2OFrame object.

Value

A vector describing the percentiles at the given cutoffs for the H2OFrame object.

Examples

```
# Request quantiles for an H2O parsed data set:
library(h2o)
localH2O <- h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(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])
```

---

sapply,H2OFrame-method

Apply Over a List in H2O

Description

Functions equivalent to the default R sapply

Usage

```
## S4 method for signature H2OFrame
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Arguments

X	an H2OFrame object on which apply will operate.
FUN	the function to be applied.
simplify,USE.NAMES	ignored parameters from base funciton
...	optional arguments to FUN.

See Also

link[base]{sapply} for the base implementation. export

---

summary,H2OModel-method *Print the Model Summary*

Description

Print the Model Summary

Usage

```
## S4 method for signature H2OModel
summary(object, ...)
```

Arguments

object	An H2OModel object.
...	further arguments to be passed on (currently unimplemented)

---

transform.H2OFrame *Transform Columns in an H2OFrame Object.*

Description

Functions that facilitate column transformations of an H2OFrame object.

Usage

```
## S3 method for class H2OFrame
transform(_data, ...)

## S3 method for class H2OFrame
within(data, expr, ...)
```

Arguments

_data,data	An H2OFrame object.
...	For transform method, column transformations in the form tag=value.
expr	For within method, column transformations specified as an expression.

See Also

transform, within for the base R methods.

Examples

```
library(h2o)
localH2O <- h2o.init()
iris.hex <- as.h2o(iris, localH2O)
transformed1 <- transform(iris.hex,
                          Sepal.Ratio = Sepal.Length / Sepal.Width,
                          Petal.Ratio = Petal.Length / Petal.Width )
transformed1
transformed2 <- within(iris.hex,
                       {Sepal.Product <- Sepal.Length * Sepal.Width
                        Petal.Product <- Petal.Length * Petal.Width
                        Sepal.Petal.Ratio <- Sepal.Product / Petal.Product
                        Sepal.Length <- Sepal.Width <- NULL
                        Petal.Length <- Petal.Width <- NULL
                        })
transformed2
```

Index