"Package 'h2o'"

March 8, 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:	0.2.0.1
Branch:	rel-selberg
Date:	Sun Mar 08 20:28:41 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 on `http://127.0.0.1:54321`, where the IP is "127.0.0.1" and the port is 54321. If H2O is running on a cluster, you must provide the IP and port of the remote machine as arguments to the h2o.init() call.

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

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

Author(s)

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

Maintainer: Anqi Fu <anqi@0xdata.com>

References

- 0xdata Homepage
- H2O Documentation
- H2O on Github

Examples

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

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

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

# Shutdown local H2O instance when finished
h2o.shutdown(localH2O)
```

---

apply,H2OFrame-method *Overloaded 'apply' method from base::*

Description

'apply' operates on H2OFrames (ASTs or H2OFrame objects) and returns an object of type H2OFrame.

Usage

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

Details

Overall Plan:

passes an AST of the format

(apply $X #MARGIN $FUN a1 a2 ...)

ASTApply will parse additional arguments to an AST[] _args. This array must be 1 less the number of args passed to FUN. Otherwise, throw an exception.

Pass the additional by calling _fun.exec(env, _args)

as.data.frame.H2OFrame

Converts a Parsed H2O data into a Data Frame

Description

Downloads the H2O data and then scan 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.H2OFrame(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(), key = "")
```

Arguments

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

`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)
```

`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)
```

Slots

`root` Object of type `Node`

`children` Object of type `list`

`ClassesIntro` *Class definitions and their 'show' & 'summary' methods.*

Description

To conveniently and safely pass messages between R and H2O, this package relies on S4 objects to capture and pass state. This R file contains all of the h2o package's classes as well as their complementary 'show' methods. The end user will typically never have to reason with these objects directly, as there are S3 accessor methods provided for creating new objects.

`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.

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)
```

Export intro *Data Export*

Description

Export data to local disk or HDFS. Save models to local disk or HDFS.

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.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.
deparse.level	Integer controlling the construction of column names. ##Currently unimplemented.##

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.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.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
head(hex)
summary(hex)

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

h2o.crossValidate *Cross Validate an H2O Model*

Description

Cross Validate an H2O Model

Usage

```
h2o.crossValidate(model, nfolds, model.type = c("gbm", "glm", "deeplearning"),
  params, strategy = c("mod1", "random"), ...)
```

h2o.cut *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, key = "iris.hex")
summary(iris.hex)

# Cut sepal length column into intervals determined by min/max/quantiles
sepal_len.cut = cut.H2OFrame(iris.hex$sepal_len, c(4.2, 4.8, 5.8, 6, 8))
head(sepal_len.cut)
summary(sepal_len.cut)
```

---

h2o.ddply *Split H2O Dataset, Apply Function, and Return Results*

Description

For each subset of 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, key = "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.deeplearning` *Build a Deep Learning Neural Network*

Description

Performs Deep Learning neural networks on an H2OFrame

Usage

```
h2o.deeplearning(x, y, training_frame, destination_key = "",
  override_with_best_model, do_classification = TRUE, 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, 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, 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)
```

Arguments

`x`	A vector containing the `character` names of the predictors in the model.
`y`	The name of the response variable in the model.
`override_with_best_model`	Logcial. If `TRUE`, override the final model with the best model found during traning. Defaults to `TRUE`.
`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** auto-tuning (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)

`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)

`l1`
: L1 regularization (can add stability and imporve 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. Can be "Automatic", "MeanSquare", or "CrossEntropy"

`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)

`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)

data	An H2OFrame object containing the variables in the model.
key	(Optional) The unique character hex key assigned to the resulting model. If none is given, a key will automatically be generated.
classification	Logical. Indicates whether the algorithm should conduct classification.
nfolds	(Optional) Number of folds for cross-validation. If nfolds >= 2, then validation must remain empty.
validation	(Optional) An H2OFrame object indicating the validation dataset used to contruct the confusion matrix. If left blank, this defaults to the training data when nfolds = 0
nesterov_accelarated_gradient	Logical. Use Nesterov accelerated gradient (reccomended)
input_dropout_ratios	Input layer dropout ration (can improve generalization) specify one value per hidden layer, defaults to 0.5

See Also

predict.H2ODeepLearningModel for prediction.

Examples

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

irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(localH2O, path = irisPath)
indep <- names(iris.hex)[1:4]
dep <- names(iris.hex)[5]
iris.dl <- h2o.deeplearning(x = indep, y = dep, data = iris.hex, activation = "Tanh", epochs = 5)
```

---

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

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

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

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.
An	H2OFrame data frame.

Details

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

Examples

```
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")
```

---

h2o.exportHDFS *Export a Model to HDFS*

Description

Exports an H2OModel to HDFS.

Usage

```
h2o.exportHDFS(object, path)
```

Arguments

object	an H2OModel class object.
path	The path to write the model to. Must include the driectory and filename.

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, do_classification, ..., destination_key,
  loss = c("AUTO", "bernoulli", "multinomial", "gaussian"), ntrees = 50,
  max_depth = 5, min_rows = 10, learn_rate = 0.1, nbins = 20,
  variable_importance = FALSE, validation_frame = FALSE,
  balance_classes = FALSE, max_after_balance_size = 1, seed,
  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.
loss	Defaults to "AUTO" A character string. The loss function to be implemented. Must be "AUTO" or "Bernoulli"
ntrees	Defaults to 50 A nonnegative integer that determines the number of trees to grow.
max_depth	Defaults to 5 Maximum depth to grow the tree.
min_rows	Defaults to 10 Minimum number of rows to assign to teminal nodes.
learn_rate	Defaults to 0.1 An interger from 0.0 to 1.0
nbins	Defaults to 20 Number of bins to use in building histogram.
variable_importance	#TODO: NEED TO FINISH
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	Defaults to FALSE logical, indicates whether or not to balance training data class counts via over/under-sampling (for imbalanced data)
max_after_balance_size	Defaults to 1 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
key	(Optional) The unique hex key assigned to the resulting model. If none is given, a key will automatically be generated.
nfolds	(Optional) Number of folds for cross-validation. If `nfolds >= 2`, then `validation` must remain empty.

See Also

`predict.H2OGBMModel` for prediction.

Examples

```
#TODO GBM wasnt working example needs to be redone, maybe
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, data = australia.hex, ntrees = 3,
        max_depth = 3, min_rows = 2)
```

---

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

Description

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

Usage

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

Arguments

key	A string indicating the unique hex key 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 key from the H2O cluster when the R proxy object is garbage collected.

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(key, conn = h2o.getConnection(), linkToGC = FALSE)
```

Arguments

key	A string indicating the unique hex key 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 key 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")
key <- h2o.gbm(x = 1:4, y = 5, training_frame = iris.hex)@key
model.retrieved <- h2o.getModel(key, localH2O)
```

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, destination_key, validation_frame, ...,
  max_iter = 50, beta_eps = 0, do_classification = FALSE,
  balance_classes = FALSE, class_sampling_factors,
  max_after_balance_size = 5, solver = c("ADMM", "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, prior1 = 0, lambda = 1e-05,
  lambda_search = FALSE, nlambdas = -1, lambda_min_ratio = 1,
  use_all_factor_levels = FALSE, n_folds = 0)
```

Arguments

x

y

training_frame

destination_key

validation_frame

...

max_iter

beta_eps

do_classification

balance_classes

class_sampling_factors

max_after_balance_size

solver

standardize

family

link

tweedie_variance_power

tweedie_link_power

alpha

prior1

lambda

lambda_search

```
nlambdas
lambda_min_ratio

use_all_factor_levels


n_folds
score_each_iteration

higher_accuracy
```

---

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.importFile *Import A File*

Description

Import a single file. 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.

Usage

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

h2o.importFolder *Data Import*

Description

Importing data is a _lazy_ parse of the data. It adds an extra step so that a user may specify a variety of options including a header file, separator type, and in the future column type. Additionally, the import phase provides feedback on whether or not a folder or group of files may be imported together.

Usage

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

Details

Import a Folder of Files

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.importHDFS *Import HDFS*

Description

Import from an HDFS location.

Usage

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

h2o.importURL *Import A URL*

Description

Import a data source from a URL.

Usage

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

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.

`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, or type h2o in the R console. h2o.shutdown for shutting down from R.

Examples

```
# 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")
```

---

h2o.insertMissingValues

Inserting Missing Values to an H2O DataFrame

Description

Inserting Missing Values to an H2O DataFrame

Usage

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

WARNING

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

h2o.kmeans *KMeans Model in H2O*

Description

Performs k-means clustering on an H2O dataset.

Usage

```
h2o.kmeans(training_frame, x, k, destination_key, 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.
destination_key	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
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 H2OKMeansModel.

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

Description

Load a saved H2O model from disk.

Usage

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

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.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.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.networkTest *View Network Traffic Speed*

Description

View Network Traffic Speed

Usage

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

---

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.parseRaw` *H2O Data Parsing*

Description

The second phase in the data ingestion step.

Usage

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

Details

Parse the Raw Data produced by the import phase.

`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)
```

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.

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,
  destination_key, 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.
`destination_key`	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
`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.
`transform`	A character string that indicates how the training data should be transformed before running PCA. Possible values are "NONE": for no transformation, "DEMEAN": for subtracting the mean of each column, "DESCALE": for dividing by the standard deviation of each column, "STANDARDIZE": for demeaning and descaling, and "NORMALIZE": for demeaning and dividing each column by its range (max - min).

Value

Returns an object of class H2ODimReductionModel.

Examples

```
library(h2o)
localH2O <- h2o.init()
arrestsPath <- system.file("extdata", "australia.csv", package="h2o")
arrests.hex <- h2o.uploadFile(localH2O, path = arrestsPath)
h2o.prcomp(training_frame = arrests.hex, k = 4, max_iterations = 100)
```

---

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.
deparse.level	Integer controlling the construction of column names. ##Currently unimplemented.##

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 Keys 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())
```

Arguments

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

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.rm *Delete Objects In H2O*

Description

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

Usage

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

Arguments

keys	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.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.
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 Key*

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, key = "iris.hex")
summary(iris.hex)

# Scale and center all the numeric columns in iris data set
h2o.scale(iris.hex[, 1:4])
```

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.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.synonym *Find Synonyms Using an H2OW2V object*

Description

Find Synonyms Using an H2OW2V object

Usage

```
h2o.synonym(word2vec, target, count)
```

Arguments

word2vec:	An H2OW2V model.
target:	A single word, or a vector of words.
count:	The top ‘count‘ synonyms will be returned.

---

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 integer or factor 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, key = "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.uploadFile *Upload Data*

Description

Upload local files to the H2O instance.

Usage

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

---

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.word2vec *Word2Vec*

Description

Create a word2vec object.

Usage

```
h2o.word2vec(trainingFrame, minWordFreq, wordModel, normModel,
  negExCnt = NULL, vecSize, windowSize, sentSampleRate, initLearningRate,
  epochs)
```

Arguments

wordModel	- SkipGram or CBOW
normModel	- Hierarchical softmax or Negative sampling
vecSize	- Size of word vectors
sentSampleRate	- Sampling rate in sentences to generate new n-grams
initLearningRate	- Starting alpha value. This tempers the effect of progressive information as learning progresses.
epochs	- Number of iterations data is run through.
	* Constructor used for hierarchical softmax cases.
numNegEx	- Number of negative samples used per word
vocabKey	- Key pointing to frame of [Word, Cnt] vectors
winSize	- Size of word window
wordModel	- SkipGram or CBOW
vocabKey	- Key pointing to frame of [Word, Cnt] vectors
vecSize	- Size of word vectors
winSize	- Size of word window
sentSampleRate	- Sampling rate in sentences to generate new n-grams
initLearningRate	- Starting alpha value. This tempers the effect of progressive information as learning progresses.
epochs	- Number of iterations data is run through.

Details

Two cases below: 1. Negative Sampling; 2. Hierarchical Softmax

* Constructor used for specifying the number of negative sampling cases.

H2OConnection-class *The H2OConnection class.*

Description

This class represents a connection to an H2O cloud.

Usage

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

## 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, ...)
```

Methods (by generic)

- Ops: For missing,H2OFrame
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For H2OFrame,missing
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For H2OFrame,H2OFrame
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For numeric,H2OFrame
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For H2OFrame,numeric
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For H2OFrame,character
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Ops: For character,H2OFrame
  "+", "-", "*", "^", "%%", "%/%", "/" "==", ">", "<", "!=", "<=", ">=", "&", "|", "**"
- Math: Generics
  "abs", "sign", "sqrt", "ceiling", "floor", "trunc", "cummax", "cummin", "cumprod", "cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh", "atan", "atanh", "exp", "expm1", "cos", "cosh", "cospi", "sin", "sinh", "sinpi", "tan", "tanh", "tanpi", "gamma", "lgamma", "digamma", "trigamma"
- Math2: Generics
  "round", "signif"
- Summary: Generics
  "max", "min", "range", "prod", "sum", "any", "all"
- !: Generic "!"
- is.na: Generic "is.na"
- t: Generic "t"
- log: Generic "log"
- trunc: Generic "trunc"

Slots

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

key A character string specifying the key for the frame 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.

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	
name	

H2OModel-class *The H2OModel object.*

Description

This virtual class represents a model built by H2O.

Usage

```
## S4 method for signature H2OModel
show(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.

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

H2OModelMetrics-class *The H2OModelMetrics Object.*

Description

A class for constructing performance measures of H2O models.

Usage

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

H2OObject-class *The H2OObject class*

Description

The H2OObject class

Usage

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

Slots

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

`key` 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)
```

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.

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

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

Returns Boolean.

Usage

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

`LazyEval` *The H2OFrame "lazy" evaluators: Evaulate an AST.*

Description

The pattern below is necessary in order to swap out S4 objects *in the calling frame*, and the code re-use is necessary in order to safely assign back to the correct environment (i.e. back to the correct calling scope).

`MethodsIntro` *A Mix of H2O-specific and Overloaded R methods.*

Description

Below we have a mix of h2o and overloaded R methods according to the following ToC:

Details

H2O Methods: ————

h2o.ls, h2o.rm, h2o.assign, h2o.createFrame, h2o.splitFrame, h2o.ignoreColumns, h2o.insertMissingValues, h2o.cut, h2o.table

Time & Date: '*' matches "Frame" and "ParsedData" –> indicates method dispatch via UseMethod ————

year.H2O*, month.H2O*, diff.H2O*

Methods are grouped according to the data types upon which they operate. There is a grouping of H2O specifc methods and methods that are overloaded from the R language (e.g. summary, head, tail, dim, nrow).

Important Developer Notes on the Lazy Evaluators: ————————————————————-

The H2OFrame "lazy" evaluators: Evaulate an AST.

The pattern below is necessary in order to swap out S4 objects *in the calling frame*, and the code re-use is necessary in order to safely assign back to the correct environment (i.e. back to the correct calling scope). If you *absolutely* need to nest calls like this, you _MUST_ correctly track the names all the way down, and then all the way back up the scopes. Here's the example pattern: Number of columns

Num Columns of an AST.

Evaluate the AST and produce the ncol of the eval'ed AST.

ncol.H2OFrame <- function(x) ID <- as.list(match.call())$x # try to get the ID from the call if(length(as.list(substitute(x))) > 1) ID <- "Last.value" # get an appropriate ID .force.eval(h2o.getConnection(), x, ID = ID, rID = 'x') # call the force eval ID <- ifelse(ID == "Last.value", ID, x@key) # bridge the IDs between the force.eval and the parent frame assign(ID, x, parent.frame()) # assign the eval'd frame into the parent env ncol(get(ID, parent.frame())) # get the object back from the parent and perform the op

Take this line-by-line: Line 1: grab the ID from the arg list, this ID is what we want the key to be in H2O Line 2: if there is no suitable ID (i.e. we have some object, not a named thing), assign to Last.value Line 3: 1. Get a handle to h2o (h2o.getConnection()) 2. x is the ast we want to eval 3. ID is the identifier we want the eventual object to have at the end of the day 4. rID is used in .force.eval to assign back into *this* scope (i.e. child scope -> parent scope) Line 4: The identifier in the parent scope will either be Last.value, or the key of the H2OFrame *NB: x is _guaranteed_ to be an H2OFrame object at this point (this is post .force.eval) Line 5: assign from *this* scope, into the parent scope Line 6: Do

---

`MethodsMisc-descrip` *Methods that don't fit into the S4 group generics:*

Description

This also handles the cases where the Math ops have multiple args (e.g. <e2><80><99>log<e2><80><99> and <e2><80><98>trunc<e2><80><99>)

Details

<e2><80><98>"!"<e2><80><99>, <e2><80><98>"is.na"<e2><80><99>, <e2><80><98>"t"<e2><80><99>, <e2><80><98>"trunc"<e2><80><99>

`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 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.

`OpsIntro-descrip` *Overview:* ———

Description

R operators mixed with H2OFrame objects.

Details

Operating on an object of type H2OFrame triggers the rollup of the expression _to be executed_ : the expression itself is not evaluated. Instead, an AST is built up from the R expression using R's built-in parser (which handles operator precedence), and, in the case of assignment, is stashed into the variable in the assignment.

The AST is bound to an R variable as a promise to evaluate the expression on demand. When evaluation is forced, the AST is walked, converted to JSON, and shipped over to H2O. The result returned by H2O is a key pointing to the newly created frame.

Methods may have a non-H2OFrame return type. Any extra preprocessing of data returned by H2O is discussed in each instance, as it varies from method to method.

What's implemented? ————————

Many of R's generic S3 methods may be mixed with H2OFrame objects wherein the result is coerced to the appropraitely typed object (typically an H2OFrame object).

A list of R's generic methods may be found by calling 'getGenerics()'. Likewise, a call to 'h2o.getGenerics()' will list the operations that are permissible with H2OFrame objects.

S3 methods are divided into four groups: Math, Ops, Complex, and Summary. H2OFrame methods follow these divisions as well, with the exception of Complex, which are unimplemented.

More precicely, the group divisions follow the S4 divisions: Ops, Math, Math2, Summary.

See also groupGeneric.

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(0.01, 0.05, 0.1, 0.25, 0.333, 0.5,
  0.667, 0.75, 0.9, 0.95, 0.99), ...)
```

Arguments

x	An H2OFrame object with a single numeric column.
probs	Numeric vector of probabilities with values in [0,1].
...	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])
```

---

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])
```

`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
```