"h2o"

May 24, 2016

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:	3.8.2.6
Branch:	rel-turchin
Date:	Tue May 24 10:57:55 PDT 2016
License:	Apache License (== 2.0)
Depends:	R (>= 2.13.0), RCurl, jsonlite, 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. 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. 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.

If you are using an older version of H2O, use the following porting guide to update your scripts: Porting Scripts

Author(s)

Anqi Fu, Tom Kraljevic and Petr Maj, with contributions from the H2O.ai team

Maintainer: Tom Kraljevic <tomk@0xdata.com>

References

- H2O.ai Homepage
- H2O Documentation
- H2O on GitHub

`aaa` *Starting H2O For examples*

Description

Starting H2O For examples

Examples

```
h2o.init()
```

apply *Apply on H2O Datasets*

Description

Method for apply on H2OFrame objects.

Usage

```
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

```
h2o.init()
irisPath = system.file("extdata", "iris.csv", package="h2o")
iris.hex = h2o.importFile(path = irisPath, destination_frame = "iris.hex")
summary(apply(iris.hex, 2, sum))
```

as.character.H2OFrame *Convert an H2OFrame to a String*

Description

Convert an H2OFrame to a String

Usage

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

Arguments

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

as.data.frame.H2OFrame *Converts parsed H2O data into an R 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

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

as.factor *Convert H2O Data to Factors*

Description

Convert a column into a factor column.

Usage

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

Arguments

x	a column from an H2OFrame data set.

See Also

is.factor.

Examples

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

Arguments

x	An R data frame.
destination_frame	A string with the desired name for the H2OFrame.

as.matrix.H2OFrame *Convert an H2OFrame to a matrix*

Description

Convert an H2OFrame to a matrix

Usage

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

Arguments

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

as.numeric *Convert H2O Data to Numeric*

Description

Converts an H2O column into a numeric value column.

Usage

```
as.numeric(x)
```

Arguments

x	a column from an H2OFrame data set.
...	Further arguments to be passed from or to other methods.

Examples

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

as.vector.H2OFrame *Convert an H2OFrame to a vector*

Description

Convert an H2OFrame to a vector

Usage

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

Arguments

x	An H2OFrame object
mode	Mode to coerce vector to

australia *Australia Coastal Data*

Description

Temperature, soil moisture, runoff, and other environmental measurements from the Australia coast. The data is available from http://cs.colby.edu/courses/S11/cs251/labs/lab07/AustraliaSubset.csv.

Format

A data frame with 251 rows and 8 columns

colnames *Returns the column names of an H2OFrame*

Description

Returns the column names of an H2OFrame

Usage

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

Arguments

x	An H2OFrame object.
do.NULL	logical. If FALSE and names are NULL, names are created.
prefix	for created names.

dim.H2OFrame *Returns the Dimensions of an H2OFrame*

Description

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

Usage

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

Arguments

x	An H2OFrame object.

See Also

dim for the base R method.

Examples

```
h2o.init()
iris.hex <- as.h2o(iris)
dim(iris.hex)
```

dimnames.H2OFrame *Column names of an H2OFrame*

Description

Column names of an H2OFrame

Usage

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

Arguments

x	An H2OFrame

`h2o.aic` *Retrieve the AIC. If "train", "valid", and "xval" parameters are FALSE (default), then the training AIC value is returned. If more than one parameter is set to TRUE, then a named vector of AICs are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the AIC. If "train", "valid", and "xval" parameters are FALSE (default), then the training AIC value is returned. If more than one parameter is set to TRUE, then a named vector of AICs are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.aic(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OModel or H2OModelMetrics.
`train`	Retrieve the training AIC
`valid`	Retrieve the validation AIC
`xval`	Retrieve the cross-validation AIC

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

Description

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

Usage

```
h2o.anomaly(object, data, per_feature = FALSE)
```

Arguments

`object`	An H2OAutoEncoderModel object that represents the model to be used for anomaly detection.
`data`	An H2OFrame object.
`per_feature`	Whether to return the per-feature squared reconstruction error

Value

Returns an H2OFrame object containing the reconstruction MSE or the per-feature squared error.

See Also

h2o.deeplearning for making an H2OAutoEncoderModel.

Examples

```
library(h2o)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(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)
prostate.anon.per.feature = h2o.anomaly(prostate.dl, prostate.hex, per_feature=TRUE)
head(prostate.anon.per.feature)
```

---

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)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.importFile(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 the AUC*

Description

Retrieves the AUC value from an H2OBinomialMetrics. If "train", "valid", and "xval" parameters are FALSE (default), then the training AUC value is returned. If more than one parameter is set to TRUE, then a named vector of AUCs are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

object	An H2OBinomialMetrics object.
train	Retrieve the training AUC
valid	Retrieve the validation AUC
xval	Retrieve the cross-validation AUC

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.betweenss` *Get the between cluster sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training betweenss value is returned. If more than one parameter is set to TRUE, then a named vector of betweenss' are returned, where the names are "train", "valid" or "xval".*

Description

Get the between cluster sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training betweenss value is returned. If more than one parameter is set to TRUE, then a named vector of betweenss' are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.betweenss(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training between cluster sum of squares
`valid`	Retrieve the validation between cluster sum of squares
`xval`	Retrieve the cross-validation between cluster sum of squares

h2o.biases — *Return the respective bias vector*

Description

Return the respective bias vector

Usage

```
h2o.biases(object, vector_id = 1)
```

Arguments

object	An H2OModel or H2OModelMetrics
vector_id	An integer, ranging from 1 to number of layers + 1, that specifies the bias vector to return.

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

---

h2o.centersSTD *Retrieve the Model Centers STD*

Description

Retrieve the Model Centers STD

Usage

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

Arguments

object	An H2OClusteringModel object.

`h2o.centroid_stats` *Retrieve the centroid statistics If "train", "valid", and "xval" parameters are FALSE (default), then the training centroid stats value is returned. If more than one parameter is set to TRUE, then a named list of centroid stats data frames are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the centroid statistics If "train", "valid", and "xval" parameters are FALSE (default), then the training centroid stats value is returned. If more than one parameter is set to TRUE, then a named list of centroid stats data frames are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.centroid_stats(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training centroid statistics
`valid`	Retrieve the validation centroid statistics
`xval`	Retrieve the cross-validation centroid statistics

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

---

h2o.clusterInfo *Print H2O cluster info*

Description

Print H2O cluster info

Usage

```
h2o.clusterInfo()
```

---

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	H2OConnection 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()
```

See Also

H2OConnection, h2o.init

Examples

```
h2o.init()
h2o.clusterStatus()
```

h2o.cluster_sizes *Retrieve the cluster sizes If "train", "valid", and "xval" parameters are FALSE (default), then the training cluster sizes value is returned. If more than one parameter is set to TRUE, then a named list of cluster size vectors are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the cluster sizes If "train", "valid", and "xval" parameters are FALSE (default), then the training cluster sizes value is returned. If more than one parameter is set to TRUE, then a named list of cluster size vectors are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

object	An H2OClusteringModel object.
train	Retrieve the training cluster sizes
valid	Retrieve the validation cluster sizes
xval	Retrieve the cross-validation cluster sizes

h2o.coef *Retrieve the model coefficeints*

Description

Retrieve the model coefficeints

Usage

```
h2o.coef(object)
```

Arguments

object	an H2OModel object.

h2o.coef_norm *Retrieve the normalized coefficients*

Description

Retrieve the normalized coefficients

Usage

```
h2o.coef_norm(object)
```

Arguments

object	an H2OModel object.

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, valid = FALSE, ...)

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

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.
valid	Retrieve the validation metric.
thresholds	(Optional) A value or a list of valid values between 0.0 and 1.0. This value is only used in the case of H2OBinomialMetrics objects.
metrics	(Optional) A metric or a list of valid metrics ("min_per_class_accuracy", "absolute_MCC", "tnr", "fnr", "fpr", "tpr", "precision", "accuracy", "f0point5", "f2", "f1"). 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 H2OFrame 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(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, time_fraction = 0, string_fraction = 0,
  missing_fraction = 0.01, response_factors = 2, has_response = FALSE,
  seed, seed_for_column_types)
```

Arguments

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.
time_fraction	The fraction of randomly created date/time columns.
string_fraction	The fraction of randomly created string columns.
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.

seed_for_column_types
: A seed used to generate random column types when randomize = TRUE.

Value

Returns an H2OFrame object.

Examples

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

---

h2o.cross_validation_fold_assignment

Retrieve the cross-validation fold assignment

Description

Retrieve the cross-validation fold assignment

Usage

```
h2o.cross_validation_fold_assignment(object)
```

Arguments

object
: An H2OModel object.

Value

Returns a H2OFrame

`h2o.cross_validation_holdout_predictions` *Retrieve the cross-validation holdout predictions*

Description

Retrieve the cross-validation holdout predictions

Usage

```
h2o.cross_validation_holdout_predictions(object)
```

Arguments

`object`	An H2OModel object.

Value

Returns a H2OFrame

`h2o.cross_validation_models` *Retrieve the cross-validation models*

Description

Retrieve the cross-validation models

Usage

```
h2o.cross_validation_models(object)
```

Arguments

`object`	An H2OModel object.

Value

Returns a list of H2OModel objects

h2o.cross_validation_predictions *Retrieve the cross-validation predictions*

Description

Retrieve the cross-validation predictions

Usage

```
h2o.cross_validation_predictions(object)
```

Arguments

object	An H2OModel object.

Value

Returns a list of H2OFrame objects

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

```
h2o.cut(x, breaks, labels = NULL, include.lowest = FALSE, right = TRUE,
  dig.lab = 3, ...)

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

Arguments

x	An H2OFrame object with a single numeric column.
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 break 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)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(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.day *Convert Milliseconds to Day of Month in H2O Datasets*

Description

Converts the entries of an H2OFrame object from milliseconds to days of the month (on a 1 to 31 scale).

Usage

```
h2o.day(x)

day(x)

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

Arguments

x	An H2OFrame object.

Value

An H2OFrame object containing the entries of x converted to days of the month.

See Also

h2o.month

---

h2o.dayOfWeek *Convert Milliseconds to Day of Week in H2O Datasets*

Description

Converts the entries of an H2OFrame object from milliseconds to days of the week (on a 0 to 6 scale).

Usage

```
h2o.dayOfWeek(x)

dayOfWeek(x)

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

Arguments

x	An H2OFrame object.

Value

An H2OFrame object containing the entries of x converted to days of the week.

See Also

h2o.day, h2o.month

---

h2o.dct *Compute DCT of an H2OFrame*

Description

Compute the Discrete Cosine Transform of every row in the H2OFrame

Usage

```
h2o.dct(data, destination_frame, dimensions, inverse = FALSE)
```

Arguments

data	An H2OFrame object representing the dataset to transform
destination_frame	A frame ID for the result
dimensions	An array containing the 3 integer values for height, width, depth of each sample. The product of HxWxD must total up to less than the number of columns. For 1D, use c(L,1,1), for 2D, use C(N,M,1).
inverse	Whether to perform the inverse transform

Examples

```
library(h2o)
h2o.init()
df <- h2o.createFrame(rows = 1000, cols = 8*16*24,
                      categorical_fraction = 0, integer_fraction = 0, missing_fraction = 0)
df1 <- h2o.dct(data=df, dimensions=c(8*16*24,1,1))
df2 <- h2o.dct(data=df1,dimensions=c(8*16*24,1,1),inverse=TRUE)
max(abs(df1-df2))

df1 <- h2o.dct(data=df, dimensions=c(8*16,24,1))
df2 <- h2o.dct(data=df1,dimensions=c(8*16,24,1),inverse=TRUE)
max(abs(df1-df2))

df1 <- h2o.dct(data=df, dimensions=c(8,16,24))
df2 <- h2o.dct(data=df1,dimensions=c(8,16,24),inverse=TRUE)
max(abs(df1-df2))
```

---

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 combine the results. This is an experimental feature.

Usage

```
h2o.ddply(X, .variables, FUN, ..., .progress = "none")
```

Arguments

X	An H2OFrame object to be processed.
.variables	Variables to split X by, either the indices or names of a set of columns.
FUN	Function to apply to each subset grouping.
...	Additional arguments passed on to FUN.
.progress	Name of the progress bar to use. #TODO: (Currently unimplemented)

Value

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

See Also

ddply for the plyr library implementation.

Examples

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

# Import iris dataset to H2O
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(path = irisPath, destination_frame = "iris.hex")
# Add function taking mean of sepal_len column
fun = function(df) { sum(df[,1], na.rm = TRUE)/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-linear 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)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(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 Neural Network*

Description

Builds a feed-forward multilayer artificial neural network on an H2OFrame

Usage

```
h2o.deeplearning(x, y, training_frame, model_id = "",
  overwrite_with_best_model, validation_frame = NULL, checkpoint = NULL,
  autoencoder = FALSE, pretrained_autoencoder = NULL,
  use_all_factor_levels = TRUE, standardize = TRUE,
  activation = c("Rectifier", "Tanh", "TanhWithDropout",
  "RectifierWithDropout", "Maxout", "MaxoutWithDropout"), hidden = c(200,
  200), epochs = 10, train_samples_per_iteration = -2,
  target_ratio_comm_to_comp = 0.05, 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, initial_weights = NULL,
  initial_biases = NULL, loss = c("Automatic", "CrossEntropy", "Quadratic",
  "Absolute", "Huber"), distribution = c("AUTO", "gaussian", "bernoulli",
  "multinomial", "poisson", "gamma", "tweedie", "laplace", "huber", "quantile"),
  quantile_alpha = 0.5, tweedie_power = 1.5, score_interval = 5,
  score_training_samples, score_validation_samples, score_duty_cycle,
  classification_stop, regression_stop, stopping_rounds = 5,
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "AUC", "r2",
```

```
"misclassification"), stopping_tolerance = 0, max_runtime_secs = 0,
quiet_mode, max_confusion_matrix_size, max_hit_ratio_k,
balance_classes = FALSE, class_sampling_factors, max_after_balance_size,
score_validation_sampling, missing_values_handling = c("MeanImputation",
"Skip"), 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, offset_column = NULL,
weights_column = NULL, nfolds = 0, fold_column = NULL,
fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
keep_cross_validation_predictions = FALSE,
keep_cross_validation_fold_assignment = 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.
overwrite_with_best_model	Logical. If TRUE, overwrite the final model with the best model found during training. Defaults to TRUE.
validation_frame	An H2OFrame object indicating the validation dataset used to construct the confusion matrix. Defaults to NULL. If left as NULL, 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.
pretrained_autoencoder	Pretrained autoencoder (either key or H2ODeepLearningModel) to initialize the model state of a supervised DL model with.
use_all_factor_levels	Logical. Use all factor levels of categorical variance. Otherwise the first factor level is omitted (without loss of accuracy). Useful for variable importances and auto-enabled for autoencoder.
standardize	Logical. If enabled, automatically standardize the data. If disabled, the user must provide properly scaled input data.
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 should be iterated (streamed), can be fractional.

`train_samples_per_iteration`
: Number of training samples (globally) per MapReduce iteration. Special values are: **0** one epoch; **-1** all available data (e.g., replicated training data); or **-2** auto-tuning (default)

`target_ratio_comm_to_comp`
: Target ratio of communication overhead to computation. Only for multi-node operation and train_samples_per_iteration=-2 (auto-tuning). Higher values can lead to faster convergence.

`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 training (try 0.5).

`momentum_ramp`
: Number of training samples for which momentum increases.

`momentum_stable`
: Final momentum after the 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`
: Hidden layer dropout ratio (can improve generalization) specify one value per hidden layer, defaults to 0.5.

`l1`
: L1 regularization (can add stability and improve generalization, causes many weights to become 0).

`l2`
: L2 regularization (can add stability and improve generalization, causes many weights to be small).

`max_w2`
: Constraint for squared sum of incoming weights per unit (e.g. Rectifier).

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

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

`initial_weights`
: Vector of frame ids for initial weight matrices

`initial_biases`
: Vector of frame ids for initial bias vectors

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

`distribution` A `character` string. The distribution function of the response. Must be "AUTO", "bernoulli", "multinomial", "poisson", "gamma", "tweedie", "laplace", "huber", "quantile" or "gaussian"

`quantile_alpha` Quantile (only for Quantile regression, must be between 0 and 1)

`tweedie_power` Tweedie power (only for Tweedie distribution, must be between 1 and 2).

`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).

`stopping_rounds`
Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve (by stopping_tolerance) for k=stopping_rounds scoring events. Can only trigger after at least 2k scoring events. Use 0 to disable.

`stopping_metric`
Metric to use for convergence checking, only for _stopping_rounds > 0 Can be one of "AUTO", "deviance", "logloss", "MSE", "AUC", "r2", "misclassification".

`stopping_tolerance`
Relative tolerance for metric-based stopping criterion (if relative improvement is not at least this much, stop).

`max_runtime_secs`
Maximum allowed runtime in seconds for model training. Use 0 to disable.

`quiet_mode` Enable quiet mode for less output to standard output.

`max_confusion_matrix_size`
Max. size (number of classes) for confusion matrices to be shown

`max_hit_ratio_k`
Max number (top K) of predictions to use for hit ratio computation (for multi-class only, 0 to disable).

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

`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.

`missing_values_handling`
: Handling of missing values. Either MeanImputation (default) or Skip.

`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`
: Ignore constant columns (no information can be gained anyway).

`force_load_balance`
: Force extra load balancing to increase training speed for small datasets (to keep all cores busy).

`replicate_training_data`
: Replicate the entire training dataset onto every node for faster training.

`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 (more efficient for data with lots of 0 values).

`col_major`
: Use a column major weight matrix for input layer. Can speed up forward propagation, 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 (requires setting the `seed` argument and this will be slow - only uses 1 thread).

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

`offset_column`
: Specify the offset column.

`weights_column`
: Specify the weights column.

`nfolds`
: (Optional) Number of folds for cross-validation.

`fold_column`
: (Optional) Column with cross-validation fold index assignment per observation.

`fold_assignment`
: Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.

`keep_cross_validation_predictions`
: Whether to keep the predictions of the cross-validation models.

`keep_cross_validation_fold_assignment`
: Whether to keep the cross-validation fold assignment.

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

See Also

predict.H2OModel for prediction.

Examples

```
library(h2o)
h2o.init()
iris.hex <- as.h2o(iris)
iris.dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex)

# now make a prediction
predictions <- h2o.predict(iris.dl, iris.hex)
```

---

h2o.describe *H2O Description of A Dataset*

Description

Reports the "Flow" style summary rollups on an instance of H2OFrame. Includes information about column types, mins/maxs/missing/zero counts/stds/number of levels

Usage

```
h2o.describe(frame)
```

Arguments

frame	An H2OFrame object.

Value

A table with the Frame stats.

Examples

```
library(h2o)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(path = prosPath)
h2o.describe(prostate.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(dirname = ".", filename = NULL)
```

Arguments

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.

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 space to accomodate the entire file.

Examples

```
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(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.download_pojo *Download the Scoring POJO (Plain Old Java Object) of an H2O Model*

Description

Download the Scoring POJO (Plain Old Java Object) of an H2O Model

Usage

```
h2o.download_pojo(model, path = "", getjar = TRUE)
```

Arguments

model	An H2OModel
path	The path to the directory to store the POJO (no trailing slash). If "", then print to to console. The file name will be a compilable java file name.
getjar	Whether to also download the h2o-genmodel.jar file needed to compile the POJO

Value

If path is "", then pretty print the POJO to the console. Otherwise save it to the specified directory.

Examples

```
library(h2o)
h <- h2o.init(nthreads=-1)
fr <- as.h2o(iris)
my_model <- h2o.gbm(x=1:4, y=5, training_frame=fr)

h2o.download_pojo(my_model)  # print the model to screen
# h2o.download_pojo(my_model, getwd())  # save the POJO and jar file to the current working
#                                         directory, NOT RUN
# h2o.download_pojo(my_model, getwd(), getjar = FALSE )  # save only the POJO to the current
```

```
#                                                          working directory, NOT RUN
h2o.download_pojo(my_model, getwd())  # save to the current working directory
```

`h2o.entropy` *Shannon entropy*

Description

Return the Shannon entropy of a string column. If the string is empty, the entropy is 0.

Usage

```
h2o.entropy(x)
```

Arguments

`x`	The column on which to calculate the entropy.

`h2o.exportFile` *Export an H2O Data Frame (H2OFrame) 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 object.
`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)
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(path = irisPath)

# These arent real paths
# 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.filterNACols *Filter NA Columns*

Description

Filter NA Columns

Usage

```
h2o.filterNACols(data, frac = 0.2)
```

Arguments

data	A dataset to filter on.
frac	The threshold of NAs to allow per column (columns >= this threshold are filtered)

`h2o.find_row_by_threshold` *Find the threshold, give the max metric. No duplicate thresholds allowed*

Description

Find the threshold, give the max metric. No duplicate thresholds allowed

Usage

```
h2o.find_row_by_threshold(object, threshold)
```

Arguments

`object`	H2OBinomialMetrics
`threshold`	number between 0 and 1

`h2o.find_threshold_by_max_metric` *Find the threshold, give the max metric*

Description

Find the threshold, give the max metric

Usage

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

Arguments

`object`	H2OBinomialMetrics
`metric`	"F1," for example

h2o.gainsLift *Access H2O Gains/Lift Tables*

Description

Retrieve either a single or many Gains/Lift tables from H2O objects.

Usage

```
h2o.gainsLift(object, ...)

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

## S4 method for signature H2OModelMetrics
h2o.gainsLift(object)
```

Arguments

object	Either an H2OModel object or an H2OModelMetrics object.
newdata	An H2OFrame object that can be scored on. Requires a valid response column.
valid	Retrieve the validation metric.
xval	Retrieve the cross-validation metric.
...	further arguments to be passed to/from this method.

Details

The H2OModelMetrics version of this function will only take H2OBinomialMetrics objects.

Value

Calling this function on H2OModel objects returns a Gains/Lift table corresponding to the `predict` function.

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, distribution = "bernoulli",
                 training_frame = hex, validation_frame = hex, nfolds=3)
h2o.gainsLift(model)              ## extract training metrics
h2o.gainsLift(model, valid=TRUE)  ## extract validation metrics (here: the same)
h2o.gainsLift(model, xval =TRUE)  ## extract cross-validation metrics
h2o.gainsLift(model, newdata=hex) ## score on new data (here: the same)
# Generating a ModelMetrics object
perf <- h2o.performance(model, hex)
h2o.gainsLift(perf)               ## extract from existing metrics object
```

---

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, checkpoint, ignore_const_cols = TRUE,
  distribution = c("AUTO", "gaussian", "bernoulli", "multinomial", "poisson",
  "gamma", "tweedie", "laplace", "quantile"), quantile_alpha = 0.5,
  tweedie_power = 1.5, ntrees = 50, max_depth = 5, min_rows = 10,
  learn_rate = 0.1, learn_rate_annealing = 1, sample_rate = 1,
  sample_rate_per_class, col_sample_rate = 1,
  col_sample_rate_change_per_level = 1, col_sample_rate_per_tree = 1,
  nbins = 20, nbins_top_level, nbins_cats = 1024, validation_frame = NULL,
  balance_classes = FALSE, class_sampling_factors,
  max_after_balance_size = 1, seed, build_tree_one_node = FALSE,
  nfolds = 0, fold_column = NULL, fold_assignment = c("AUTO", "Random",
  "Modulo", "Stratified"), keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  score_each_iteration = FALSE, score_tree_interval = 0,
  stopping_rounds = 0, stopping_metric = c("AUTO", "deviance", "logloss",
  "MSE", "AUC", "r2", "misclassification"), stopping_tolerance = 0.001,
  max_runtime_secs = 0, offset_column = NULL, weights_column = NULL,
  min_split_improvement, histogram_type = c("AUTO", "UniformAdaptive",
  "Random", "QuantilesGlobal", "RoundRobin"), max_abs_leafnode_pred)
```

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.

`checkpoint` "Model checkpoint (either key or H2ODeepLearningModel) to resume training with."

`ignore_const_cols`
A logical value indicating whether or not to ignore all the constant columns in the training frame.

`distribution` A `character` string. The distribution function of the response. Must be "AUTO", "bernoulli", "multinomial", "poisson", "gamma", "tweedie", "laplace", "quantile" or "gaussian"

`quantile_alpha` Quantile (only for Quantile regression, must be between 0 and 1)

`tweedie_power` Tweedie power (only for Tweedie distribution, must be between 1 and 2)

`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` Learning rate (from `0.0` to `1.0`)

`learn_rate_annealing`
Scale the learning rate by this factor after each tree (e.g., 0.99 or 0.999)

`sample_rate` Row sample rate per tree (from `0.0` to `1.0`)

`sample_rate_per_class`
Row sample rate per tree per class (one per class, from `0.0` to `1.0`)

`col_sample_rate`
Column sample rate per split (from `0.0` to `1.0`)

`col_sample_rate_change_per_level`
Relative change of the column sampling rate for every level (from 0.0 to 2.0)

`col_sample_rate_per_tree`
Column sample rate per tree (from `0.0` to `1.0`)

`nbins` For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point.

`nbins_top_level`
For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level.

`nbins_cats` For categorical columns (factors), build a histogram of this many bins, then split at the best point. Higher values can lead to more overfitting.

`validation_frame`
An H2OFrame object indicating the validation dataset used to contruct the confusion matrix. Defaults to NULL. If left as NULL, 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).

`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). Ignored if balance_classes is FALSE, which is the default behavior.

`seed`
: Seed for random numbers (affects sampling).

`build_tree_one_node`
: Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets.

`nfolds`
: (Optional) Number of folds for cross-validation.

`fold_column`
: (Optional) Column with cross-validation fold index assignment per observation

`fold_assignment`
: Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.

`keep_cross_validation_predictions`
: Whether to keep the predictions of the cross-validation models

`keep_cross_validation_fold_assignment`
: Whether to keep the cross-validation fold assignment.

`score_each_iteration`
: Attempts to score each tree.

`score_tree_interval`
: Score the model after every so many trees. Disabled if set to 0.

`stopping_rounds`
: Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve (by stopping_tolerance) for k=stopping_rounds scoring events. Can only trigger after at least 2k scoring events. Use 0 to disable.

`stopping_metric`
: Metric to use for convergence checking, only for _stopping_rounds > 0 Can be one of "AUTO", "deviance", "logloss", "MSE", "AUC", "r2", "misclassification".

`stopping_tolerance`
: Relative tolerance for metric-based stopping criterion (if relative improvement is not at least this much, stop)

`max_runtime_secs`
: Maximum allowed runtime in seconds for model training. Use 0 to disable.

`offset_column`
: Specify the offset column.

`weights_column`
: Specify the weights column.

`min_split_improvement`
: Minimum relative improvement in squared error reduction for a split to happen.

`histogram_type` What type of histogram to use for finding optimal split points Can be one of "AUTO", "UniformAdaptive", "Random", "QuantilesGlobal" or "RoundRobin".

`max_abs_leafnode_pred`
Maximum absolute value of a leaf node prediction.

Details

The default distribution function will guess the model type based on the response column type. In order to run 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)
h2o.init()

# Run regression GBM on australia.hex data
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(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, that will NOT be GC'd by default*

Description

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

Usage

```
h2o.getFrame(id)
```

Arguments

`id`	A string indicating the unique frame of the dataset to retrieve.

`h2o.getFutureModel` *Get future model*

Description

Get future model

Usage

```
h2o.getFutureModel(object)
```

Arguments

`object`	H2OModel

`h2o.getGLMFullRegularizationPath` *Extract full regularization path from glm model (assuming it was run with lambda search option)*

Description

Extract full regularization path from glm model (assuming it was run with lambda search option)

Usage

```
h2o.getGLMFullRegularizationPath(model)
```

Arguments

`model`	an H2OModel corresponding from a `h2o.glm` call.

h2o.getGrid *Get a grid object from H2O distributed K/V store.*

Description

Get a grid object from H2O distributed K/V store.

Usage

```
h2o.getGrid(grid_id, sort_by, decreasing)
```

Arguments

grid_id	ID of existing grid object to fetch
sort_by	Sort the models in the grid space by a metric. Choices are "logloss", "residual_deviance", "mse", "auc", "r2", "accuracy", "precision", "recall", "f1", etc.
decreasing	Specify whether sort order should be decreasing

Examples

```
library(h2o)
library(jsonlite)
h2o.init()
iris.hex <- as.h2o(iris)
h2o.grid("gbm", grid_id = "gbm_grid_id", x = c(1:4), y = 5,
         training_frame = iris.hex, hyper_params = list(ntrees = c(1,2,3)))
grid <- h2o.getGrid("gbm_grid_id")
# Get grid summary
summary(grid)
# Fetch grid models
model_ids <- grid@model_ids
models <- lapply(model_ids, function(id) { h2o.getModel(id)})
```

h2o.getId *Get back-end distributed key/value store id from an H2OFrame.*

Description

Get back-end distributed key/value store id from an H2OFrame.

Usage

```
h2o.getId(x)
```

Arguments

x	An H2OFrame

Value

The id

---

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)
```

Arguments

model_id	A string indicating the unique model_id of the model to retrieve.

Value

Returns an object that is a subclass of H2OModel.

Examples

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

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

---

h2o.getTimezone *Get the Time Zone on the H2O Cloud Returns a string*

Description

Get the Time Zone on the H2O Cloud Returns a string

Usage

```
h2o.getTimezone()
```

h2o.getTypes *Get the types-per-column*

Description

Get the types-per-column

Usage

```
h2o.getTypes(x)
```

Arguments

x	An H2OFrame

Value

A list of types

h2o.getVersion *Get h2o version*

Description

Get h2o version

Usage

```
h2o.getVersion()
```

h2o.giniCoef *Retrieve the GINI Coefficcient*

Description

Retrieves the GINI coefficient from an H2OBinomialMetrics. If "train", "valid", and "xval" parameters are FALSE (default), then the training GINIvalue is returned. If more than one parameter is set to TRUE, then a named vector of GINIs are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.giniCoef(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

object	an H2OBinomialMetrics object.
train	Retrieve the training GINI Coefficcient
valid	Retrieve the validation GINI Coefficcient
xval	Retrieve the cross-validation GINI Coefficcient

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 = NULL,
  ignore_const_cols = TRUE, max_iterations = 50, beta_epsilon = 0,
  solver = c("IRLSM", "L_BFGS"), standardize = TRUE,
  family = c("gaussian", "binomial", "poisson", "gamma", "tweedie",
  "multinomial"), link = c("family_default", "identity", "logit", "log",
  "inverse", "tweedie"), tweedie_variance_power = NaN,
  tweedie_link_power = NaN, alpha = 0.5, prior = NULL, lambda = 1e-05,
  lambda_search = FALSE, nlambdas = -1, lambda_min_ratio = -1,
  nfolds = 0, fold_column = NULL, fold_assignment = c("AUTO", "Random",
  "Modulo", "Stratified"), keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE, beta_constraints = NULL,
```

```
  offset_column = NULL, weights_column = NULL, intercept = TRUE,
  max_active_predictors = -1, interactions = NULL, objective_epsilon = -1,
  gradient_epsilon = -1, non_negative = FALSE, compute_p_values = FALSE,
  remove_collinear_columns = FALSE, max_runtime_secs = 0,
  missing_values_handling = c("MeanImputation", "Skip"))
```

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. Defaults to NULL.
ignore_const_cols	A logical value indicating whether or not to ignore all the constant columns in the training frame.
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. Must be from (0,1) exclusive range or NULL (no prior).

`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.

`nfolds` (Optional) Number of folds for cross-validation.

`fold_column` (Optional) Column with cross-validation fold index assignment per observation.

`fold_assignment`
Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.

`keep_cross_validation_predictions`
Whether to keep the predictions of the cross-validation models.

`keep_cross_validation_fold_assignment`
Whether to keep the cross-validation fold assignment.

`beta_constraints`
A data.frame or H2OParsedData object with the columns ["names", "lower_bounds", "upper_bounds", "beta_given", "rho"], where each row corresponds to a predictor in the GLM. "names" contains the predictor names, "lower_bounds" and "upper_bounds" are the lower and upper bounds of beta, "beta_given" is some supplied starting values for beta, and "rho" is the proximal penalty constant that is used with "beta_given". If "rho" is not specified when "beta_given" is then we will take the default rho value of zero.

`offset_column` Specify the offset column.

`weights_column` Specify the weights column.

`intercept` Logical, include constant term (intercept) in the model.

`max_active_predictors`
(Optional) Convergence criteria for number of predictors when using L1 penalty.

`interactions` A vector of column indices to interact pairwise. All combinations of two indices will be computed.

`objective_epsilon`
Convergence criteria. Converge if relative change in objective function is below this threshold.

`gradient_epsilon`
Convergence criteria. Converge if gradient l-infinity norm is below this threshold.

non_negative Logical, allow only positive coefficients.

compute_p_values
(Optional) Logical, compute p-values, only allowed with IRLSM solver and no regularization. May fail if there are collinear predictors.

remove_collinear_columns
(Optional) Logical, valid only with no regularization. If set, co-linear columns will be automatically ignored (coefficient will be 0).

max_runtime_secs
Maximum allowed runtime in seconds for model training. Use 0 to disable.

missing_values_handling
(Optional) Controls handling of missing values. Can be either "MeanImputation" or "Skip". MeanImputation replaces missing values with mean for numeric and most frequent level for categorical, Skip ignores observations with any missing value. Applied both during model training *AND* scoring.

... (Currently Unimplemented) coefficients.

Value

A subclass of H2OModel is returned. The specific subclass depends on the machine learning task at hand (if it's binomial classification, then an H2OBinomialModel is returned, if it's regression then a H2ORegressionModel is returned). The default print-out of the models is shown, but further GLM-specifc information can be queried out of the object. To access these various items, please refer to the seealso section below.

Upon completion of the GLM, the resulting object has coefficients, normalized coefficients, residual/null deviance, aic, and a host of model metrics including MSE, AUC (for logistic regression), degrees of freedom, and confusion matrices. Please refer to the more in-depth GLM documentation available here: http://h2o-release.s3.amazonaws.com/h2o-dev/rel-shannon/2/docs-website/h2o-docs/index.html#Data+Science+Algorithms-GLM,

See Also

predict.H2OModel for prediction, h2o.mse, h2o.auc, h2o.confusionMatrix, h2o.performance, h2o.giniCoef, h2o.logloss, h2o.varimp, h2o.scoreHistory

Examples

```
h2o.init()

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

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

```
# GLM variable importance
# Also see:
#   https://github.com/h2oai/h2o/blob/master/R/tests/testdir_demos/runit_demo_VI_all_algos.R
data.hex = h2o.importFile(
  path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv",
  destination_frame = "data.hex")
myX = 1:20
myY="y"
my.glm = h2o.glm(x=myX, y=myY, training_frame=data.hex, family="binomial", standardize=TRUE,
                 lambda_search=TRUE)
```

---

h2o.glrm *Generalized Low Rank Model*

Description

Generalized low rank decomposition of an H2O data frame.

Usage

```
h2o.glrm(training_frame, cols, k, model_id, validation_frame, loading_name,
  ignore_const_cols, transform = c("NONE", "DEMEAN", "DESCALE", "STANDARDIZE",
  "NORMALIZE"), loss = c("Quadratic", "L1", "Huber", "Poisson", "Hinge",
  "Logistic"), multi_loss = c("Categorical", "Ordinal"), loss_by_col = NULL,
  loss_by_col_idx = NULL, regularization_x = c("None", "Quadratic", "L2",
  "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex"),
  regularization_y = c("None", "Quadratic", "L2", "L1", "NonNegative",
  "OneSparse", "UnitOneSparse", "Simplex"), gamma_x = 0, gamma_y = 0,
  max_iterations = 1000, max_updates = 2 * max_iterations,
  init_step_size = 1, min_step_size = 0.001, init = c("Random",
  "PlusPlus", "SVD"), svd_method = c("GramSVD", "Power", "Randomized"),
  user_y = NULL, user_x = NULL, expand_user_y = TRUE,
  impute_original = FALSE, recover_svd = FALSE, seed,
  max_runtime_secs = 0)
```

Arguments

training_frame	An H2OFrame object containing the variables in the model.
cols	(Optional) A vector containing the data columns on which k-means operates.
k	The rank of the resulting decomposition. This must be between 1 and the number of columns in the training frame, inclusive.
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.

loading_name
: (Optional) The unique name assigned to the loading matrix X in the XY decomposition. Automatically generated if none is provided.

ignore_const_cols
: (Optional) A logical value indicating whether to ignore constant columns in the training frame. A column is constant if all of its non-missing values are the same value.

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).

loss
: A character string indicating the default loss function for numeric columns. Possible values are "Quadratic" (default), "L1", "Huber", "Poisson", "Hinge" and "Logistic".

multi_loss
: A character string indicating the default loss function for enum columns. Possible values are "Categorical" and "Ordinal".

loss_by_col
: A vector of strings indicating the loss function for specific columns by corresponding index in loss_by_col_idx. Will override loss for numeric columns and multi_loss for enum columns.

loss_by_col_idx
: A vector of column indices to which the corresponding loss functions in loss_by_col are assigned. Must be zero indexed.

regularization_x
: A character string indicating the regularization function for the X matrix. Possible values are "None" (default), "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", and "Simplex".

regularization_y
: A character string indicating the regularization function for the Y matrix. Possible values are "None" (default), "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", and "Simplex".

gamma_x
: The weight on the X matrix regularization term.

gamma_y
: The weight on the Y matrix regularization term.

max_iterations
: The maximum number of iterations to run the optimization loop. Each iteration consists of an update of the X matrix, followed by an update of the Y matrix.

max_updates
: The maximum number of updates of X or Y to run. Each update consists of an update of either the X matrix or the Y matrix. For example, if max_updates = 1 and max_iterations = 1, the algorithm will initialize X and Y, update X once, and terminate without updating Y.

init_step_size
: Initial step size. Divided by number of columns in the training frame when calculating the proximal gradient update. The algorithm begins at init_step_size and decreases the step size at each iteration until a termination condition is reached.

`min_step_size` Minimum step size upon which the algorithm is terminated.

`init` A character string indicating how to select the initial Y matrix. Possible values are "Random": for initialization to a random array from the standard normal distribution, "PlusPlus": for initialization using the clusters from k-means++ initialization, or "SVD": for initialization using the first k right singular vectors. Additionally, the user may specify the initial Y as a matrix, data.frame, H2OFrame, or list of vectors.

`svd_method` (Optional) A character string that indicates how SVD should be calculated during initialization. Possible values are "GramSVD": distributed computation of the Gram matrix followed by a local SVD using the JAMA package, "Power": computation of the SVD using the power iteration method, "Randomized": (default) approximate SVD by projecting onto a random subspace (see references).

`user_y` (Optional) A matrix, data.frame, H2OFrame, or list of vectors specifying the initial Y. Only used when init = "User". The number of rows must equal k.

`user_x` (Optional) A matrix, data.frame, H2OFrame, or list of vectors specifying the initial X. Only used when init = "User". The number of columns must equal k.

`expand_user_y` A logical value indicating whether the categorical columns of user_y should be one-hot expanded. Only used when init = "User" and user_y is specified.

`impute_original`
A logical value indicating whether to reconstruct the original training data by reversing the transformation during prediction. Model metrics are calculated with respect to the original data.

`recover_svd` A logical value indicating whether the singular values and eigenvectors should be recovered during post-processing of the generalized low rank decomposition.

`seed` (Optional) Random seed used to initialize the X and Y matrices.

`max_runtime_secs`
Maximum allowed runtime in seconds for model training. Use 0 to disable.

Value

Returns an object of class H2ODimReductionModel.

References

M. Udell, C. Horn, R. Zadeh, S. Boyd (2014). Generalized Low Rank Models[http://arxiv.org/abs/1410.0342]. Unpublished manuscript, Stanford Electrical Engineering Department. N. Halko, P.G. Martinsson, J.A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions[http://arxiv.org/abs/0909.4061]. SIAM Rev., Survey and Review section, Vol. 53, num. 2, pp. 217-288, June 2011.

See Also

`h2o.kmeans`, `h2o.svd`, `h2o.prcomp`

Examples

```
library(h2o)
h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(path = ausPath)
h2o.glrm(training_frame = australia.hex, k = 5, loss = "Quadratic", regularization_x = "L1",
         gamma_x = 0.5, gamma_y = 0, max_iterations = 1000)
```

---

h2o.grid *H2O Grid Support*

Description

Provides a set of functions to launch a grid search and get its results.

Usage

```
h2o.grid(algorithm, grid_id, ..., hyper_params = list(),
  is_supervised = NULL, do_hyper_params_check = FALSE,
  search_criteria = NULL)
```

Arguments

algorithm	Name of algorithm to use in grid search (gbm, randomForest, kmeans, glm, deeplearning, naivebayes, pca).
grid_id	(Optional) ID for resulting grid search. If it is not specified then it is autogenerated.
...	arguments describing parameters to use with algorithm (i.e., x, y, training_frame). Look at the specific algorithm - h2o.gbm, h2o.glm, h2o.kmeans, h2o.deepLearning - for available parameters.
hyper_params	List of lists of hyper parameters (i.e., `list(ntrees=c(1,2), max_depth=c(5,7))`).
is_supervised	(Optional) If specified then override the default heuristic which decides if the given algorithm name and parameters specify a supervised or unsupervised algorithm.
do_hyper_params_check	Perform client check for specified hyper parameters. It can be time expensive for large hyper space.
search_criteria	(Optional) List of control parameters for smarter hyperparameter search. The default strategy 'Cartesian' covers the entire space of hyperparameter combinations. Specify the 'RandomDiscrete' strategy to get random search of all the combinations of your hyperparameters. RandomDiscrete should be usually combined with at least one early stopping criterion, max_models and/or max_runtime_secs, e.g. `list(strategy = "RandomDiscrete", max_models = 42, max_runtime_s` or `list(strategy = "RandomDiscrete", stopping_metric = "AUTO", stopping_tolerance = 0` or `list(strategy = "RandomDiscrete", stopping_metric = "misclassification", stopping_`

Details

Launch grid search with given algorithm and parameters.

Examples

```
library(h2o)
library(jsonlite)
h2o.init()
iris.hex <- as.h2o(iris)
grid <- h2o.grid("gbm", x = c(1:4), y = 5, training_frame = iris.hex,
                 hyper_params = list(ntrees = c(1,2,3)))
# Get grid summary
summary(grid)
# Fetch grid models
model_ids <- grid@model_ids
models <- lapply(model_ids, function(id) { h2o.getModel(id)})
```

---

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.gsub` *String Global Substitute*

Description

Creates a copy of the target column in which each string has all occurence of the regex pattern replaced with the replacement substring.

Usage

```
h2o.gsub(pattern, replacement, x, ignore.case = FALSE)
```

Arguments

`pattern`	The pattern to replace.
`replacement`	The replacement pattern.
`x`	The column on which to operate.
`ignore.case`	Case sensitive or not

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

Description

Returns the first or last rows of an H2OFrame object.

Usage

```
h2o.head(x, ..., n = 6L)

## S3 method for class H2OFrame
head(x, ..., n = 6L)

h2o.tail(x, ..., n = 6L)

## S3 method for class H2OFrame
tail(x, ..., n = 6L)
```

Arguments

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

Value

An H2OFrame containing the first or last n rows of an H2OFrame object.

Examples

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

---

h2o.hist *Compute A Histogram*

Description

Compute a histogram over a numeric column. If breaks=="FD", the MAD is used over the IQR in computing bin width. Note that we do not beautify the breakpoints as R does.

Usage

```
h2o.hist(x, breaks = "Sturges", plot = TRUE)
```

Arguments

x	A single numeric column from an H2OFrame.
breaks	Can be one of the following: A string: "Sturges", "Rice", "sqrt", "Doane", "FD", "Scott" A single number for the number of breaks splitting the range of the vec into number of breaks bins of equal width A vector of numbers giving the split points, e.g., c(-50,213.2123,9324834)
plot	A logical value indicating whether or not a plot should be generated (default is TRUE).

`h2o.hit_ratio_table` *Retrieve the Hit Ratios If "train", "valid", and "xval" parameters are FALSE (default), then the training Hit Ratios value is returned. If more than one parameter is set to TRUE, then a named list of Hit Ratio tables are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the Hit Ratios If "train", "valid", and "xval" parameters are FALSE (default), then the training Hit Ratios value is returned. If more than one parameter is set to TRUE, then a named list of Hit Ratio tables are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

`object`	An H2OModel object.
`train`	Retrieve the training Hit Ratio
`valid`	Retrieve the validation Hit Ratio
`xval`	Retrieve the cross-validation Hit Ratio

`h2o.hour` *Convert Milliseconds to Hour of Day in H2O Datasets*

Description

Converts the entries of an H2OFrame object from milliseconds to hours of the day (on a 0 to 23 scale).

Usage

```
h2o.hour(x)

hour(x)

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

Arguments

`x`	An H2OFrame object.

Value

An H2OFrame object containing the entries of x converted to hours of the day.

See Also

h2o.day

---

h2o.ifelse *H2O Apply Conditional Statement*

Description

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

Usage

```
h2o.ifelse(test, yes, no)

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

Both numeric and categorical values can be tested. However when returning a yes and no condition both conditions must be either both categorical or numeric.

Value

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

Examples

```
h2o.init()
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(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, pattern = "", destination_frame = "", parse = TRUE,
  header = NA, sep = "", col.names = NULL, col.types = NULL,
  na.strings = NULL)

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

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

h2o.uploadFile(path, destination_frame = "", parse = TRUE, header = NA,
  sep = "", col.names = NULL, col.types = NULL, na.strings = NULL,
  progressBar = FALSE, parse_type = NULL)

h2o.read.svmlight(path, pattern = "", destination_frame = "")
```

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.
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) An H2OFrame 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.
na.strings	(Optional) H2O will interpret these strings as missing.
progressBar	(Optional) When FALSE, tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.
parse_type	(Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"

Details

Other than h2o.uploadFile, if the given path is relative, then it will be relative to the start location of the H2O instance. Additionally, 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

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

---

h2o.import_sql_select *Import SQL table that is result of SELECT SQL query into H2O*

Description

Creates a temporary SQL table from the specified sql_query. Runs multiple SELECT SQL queries on the temporary table concurrently for parallel ingestion, then drops the table. Be sure to start the h2o.jar in the terminal with your downloaded JDBC driver in the classpath: 'java -cp <path_to_h2o_jar>:<path_to_jdbc_drive water.H2OApp' Also see h2o.import_sql_table. Currently supported SQL databases are MySQL, PostgreSQL, and MariaDB. Support for Oracle 12g and Microsoft SQL Server

Usage

```
h2o.import_sql_select(connection_url, select_query, username, password,
  optimize = NULL)
```

Arguments

connection_url	URL of the SQL database connection as specified by the Java Database Connectivity (JDBC) Driver. For example, "jdbc:mysql://localhost:3306/menagerie?&useSSL=false"
select_query	SQL query starting with 'SELECT' that returns rows from one or more database tables.
username	Username for SQL server
password	Password for SQL server
optimize	(Optional) Optimize import of SQL table for faster imports. Experimental. Default is true.

Details

For example, my_sql_conn_url <- "jdbc:mysql://172.16.2.178:3306/ingestSQL?&useSSL=false" select_query <- "SELECT bikeid from citibike20k" username <- "root" password <- "abc123" my_citibike_data <- h2o.import_sql_select(my_sql_conn_url, select_query, username, password)

---

h2o.import_sql_table *Import SQL Table into H2O*

Description

Imports SQL table into an H2O cloud. Assumes that the SQL table is not being updated and is stable. Runs multiple SELECT SQL queries concurrently for parallel ingestion. Be sure to start the h2o.jar in the terminal with your downloaded JDBC driver in the classpath: 'java -cp <path_to_h2o_jar>:<path_to_jdbc_driver_jar> water.H2OApp' Also see h2o.import_sql_select. Currently supported SQL databases are MySQL, PostgreSQL, and MariaDB. Support for Oracle 12g and Microsoft SQL Server

Usage

```
h2o.import_sql_table(connection_url, table, username, password,
  columns = NULL, optimize = NULL)
```

Arguments

connection_url	URL of the SQL database connection as specified by the Java Database Connectivity (JDBC) Driver. For example, "jdbc:mysql://localhost:3306/menagerie?&useSSL=false"
table	Name of SQL table
username	Username for SQL server
password	Password for SQL server
columns	(Optional) Character vector of column names to import from SQL table. Default is to import all columns.
optimize	(Optional) Optimize import of SQL table for faster imports. Experimental. Default is true.

Details

For example, my_sql_conn_url <- "jdbc:mysql://172.16.2.178:3306/ingestSQL?&useSSL=false" table <- "citibike20k" username <- "root" password <- "abc123" my_citibike_data <- h2o.import_sql_table(my_sql_conn_url, table, username, password)

`h2o.impute` *Basic Imputation of H2O Vectors*

Description

Perform inplace imputation by filling missing values with aggregates computed on the "na.rm'd" vector. Additionally, it's possible to perform imputation based on groupings of columns from within data; these columns can be passed by index or name to the by parameter. If a factor column is supplied, then the method must be "mode".

Usage

```
h2o.impute(data, column = 0, method = c("mean", "median", "mode"),
  combine_method = c("interpolate", "average", "lo", "hi"), by = NULL,
  groupByFrame = NULL, values = NULL)
```

Arguments

`data`	The dataset containing the column to impute.
`column`	A specific column to impute, default of 0 means impute the whole frame.
`method`	"mean" replaces NAs with the column mean; "median" replaces NAs with the column median; "mode" replaces with the most common factor (for factor columns only);
`combine_method`	If method is "median", then choose how to combine quantiles on even sample sizes. This parameter is ignored in all other cases.
`by`	group by columns
`groupByFrame`	Impute the column col with this pre-computed grouped frame.
`values`	A vector of impute values (one per column). NaN indicates to skip the column

Details

The default method is selected based on the type of the column to impute. If the column is numeric then "mean" is selected; if it is categorical, then "mode" is selected. Other column types (e.g. String, Time, UUID) are not supported.

Value

an H2OFrame with imputed values

Examples

```
h2o.init()
fr <- as.h2o(iris, destination_frame="iris")
fr[sample(nrow(fr),40),5] <- NA  # randomly replace 50 values with NA
# impute with a group by
fr <- h2o.impute(fr, "Species", "mode", by=c("Sepal.Length", "Sepal.Width"))
```

---

h2o.init *Initialize and Connect to H2O*

Description

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

Usage

```
h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,
  forceDL = FALSE, enable_assertions = TRUE, license = NULL,
  nthreads = -2, max_mem_size = NULL, min_mem_size = NULL,
  ice_root = tempdir(), strict_version_check = TRUE,
  proxy = NA_character_, https = FALSE, insecure = FALSE,
  username = NA_character_, password = NA_character_)
```

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.
enable_assertions	(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.

`proxy` (Optional) A `character` string specifying the proxy path.

`https` (Optional) Set this to TRUE to use https instead of http.

`insecure` (Optional) Set this to TRUE to disable SSL certificate checking.

`username` (Optional) Username to login with.

`password` (Optional) Password to login with.

Details

By default, 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.
h2o.init()

# Try to connect to a local H2O instance.
# If not found, raise an error.
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.
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.
h2o.init(max_mem_size = "5g")

## End(Not run)
```

---

h2o.insertMissingValues

Inserting Missing Values to an H2O DataH2OFrame

Description

This is primarily used for testing. Randomly replaces a user-specified fraction of entries in an 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)
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.importFile(path = irisPath)
summary(iris.hex)
irismiss.hex <- h2o.insertMissingValues(iris.hex, fraction = 0.25)
head(irismiss.hex)
summary(irismiss.hex)
```

---

h2o.interaction *Categorical Interaction Feature Creation in H2O*

Description

Creates a data frame in H2O with n-th order interaction features between categorical columns, as specified by the user.

Usage

```
h2o.interaction(data, destination_frame, factors, pairwise, max_factors,
  min_occurrence)
```

Arguments

data	An H2OFrame object containing the categorical columns.
destination_frame	A string indicating the destination key. If empty, this will be auto-generated by H2O.
factors	Factor columns (either indices or column names).
pairwise	Whether to create pairwise interactions between factors (otherwise create one higher-order interaction). Only applicable if there are 3 or more factors.
max_factors	Max. number of factor levels in pair-wise interaction terms (if enforced, one extra catch-all factor will be made)
min_occurrence	Min. occurrence threshold for factor levels in pair-wise interaction terms

Value

Returns an H2OFrame object.

Examples

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

# Create some random data
myframe = h2o.createFrame(rows = 20, cols = 5,
                          seed = -12301283, randomize = TRUE, value = 0,
                          categorical_fraction = 0.8, factors = 10, real_range = 1,
                          integer_fraction = 0.2, integer_range = 10,
                          binary_fraction = 0, binary_ones_fraction = 0.5,
                          missing_fraction = 0.2,
                          response_factors = 1)
# Turn integer column into a categorical
myframe[,5] <- as.factor(myframe[,5])
head(myframe, 20)

# Create pairwise interactions
pairwise <- h2o.interaction(myframe, destination_frame = pairwise,
                            factors = list(c(1,2),c("C2","C3","C4")),
                            pairwise=TRUE, max_factors = 10, min_occurrence = 1)
head(pairwise, 20)
h2o.levels(pairwise,2)

# Create 5-th order interaction
higherorder <- h2o.interaction(myframe, destination_frame = higherorder, factors = c(1,2,3,4,5),
                               pairwise=FALSE, max_factors = 10000, min_occurrence = 1)
head(higherorder, 20)

# Limit the number of factors of the "categoricalized" integer column
# to at most 3 factors, and only if they occur at least twice
head(myframe[,5], 20)
trim_integer_levels <- h2o.interaction(myframe, destination_frame = trim_integers, factors = "C5",
                                       pairwise = FALSE, max_factors = 3, min_occurrence = 2)
head(trim_integer_levels, 20)

# Put all together
myframe <- h2o.cbind(myframe, pairwise, higherorder, trim_integer_levels)
myframe
head(myframe,20)
summary(myframe)
```

---

h2o.is_client *Check Client Mode Connection*

Description

Check Client Mode Connection

Usage

```
h2o.is_client()
```

`h2o.kfold_column` *Produce a k-fold column vector.*

Description

Create a k-fold vector useful for H2O algorithms that take a fold_assignments argument.

Usage

```
h2o.kfold_column(data, nfolds, seed = -1)
```

Arguments

`data`	A dataframe against which to create the fold column.
`nfolds`	The number of desired folds.
`seed`	A random seed, -1 indicates that H2O will choose one.

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

Description

A poor man's profiler, but effective.

Usage

```
h2o.killMinus3()
```

h2o.kmeans *KMeans Model in H2O*

Description

Performs k-means clustering on an H2O dataset.

Usage

```
h2o.kmeans(training_frame, x, k, model_id, ignore_const_cols = TRUE,
  max_iterations = 1000, standardize = TRUE, init = c("Furthest",
  "Random", "PlusPlus"), seed, nfolds = 0, fold_column = NULL,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE, max_runtime_secs = 0)
```

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.
ignore_const_cols	A logical value indicating whether or not to ignore all the constant columns in the training frame.
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 Frames, 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.
nfolds	(Optional) Number of folds for cross-validation.
fold_column	(Optional) Column with cross-validation fold index assignment per observation
fold_assignment	Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.

keep_cross_validation_predictions
: Whether to keep the predictions of the cross-validation models

keep_cross_validation_fold_assignment
: Whether to keep the cross-validation fold assignment.

max_runtime_secs
: Maximum allowed runtime in seconds for model training. Use 0 to disable.

Value

Returns an object of class H2OClusteringModel.

See Also

h2o.cluster_sizes, h2o.totss, h2o.num_iterations, h2o.betweenss, h2o.tot_withinss, h2o.withinss, h2o.centersSTD, h2o.centers

Examples

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

---

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

Description

Return the levels from the column requested column.

Usage

```
h2o.levels(x, i)
```

Arguments

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

See Also

levels for the base R method.

Examples

```
iris.hex <- as.h2o(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()
```

---

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

Description

Load a saved H2O model from disk.

Usage

```
h2o.loadModel(path)
```

Arguments

path	The path of the H2O Model to be imported. 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)
# h2o.init()
# prosPath = system.file("extdata", "prostate.csv", package = "h2o")
# prostate.hex = h2o.importFile(path = prosPath, destination_frame = "prostate.hex")
# prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#   training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# glmmodel.path = h2o.saveModel(prostate.glm, dir = "/Users/UserName/Desktop")
# glmmodel.load = h2o.loadModel(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)
```

Arguments

message	A character string with the message to write to the log.

Details

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

---

h2o.logloss *Retrieve the Log Loss Value*

Description

Retrieves the log loss output for a H2OBinomialMetrics or H2OMultinomialMetrics object If "train", "valid", and "xval" parameters are FALSE (default), then the training Log Loss value is returned. If more than one parameter is set to TRUE, then a named vector of Log Losses are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

object	a H2OModelMetrics object of the correct type.
train	Retrieve the training Log Loss
valid	Retrieve the validation Log Loss
xval	Retrieve the cross-validation Log Loss

---

h2o.ls *List Keys on an H2O Cluster*

Description

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

Usage

```
h2o.ls()
```

Value

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

Examples

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

---

h2o.lstrip *Strip set from left*

Description

Return a copy of the target column with leading characters removed. The set argument is a string specifying the set of characters to be removed. If omitted, the set argument defaults to removing whitespace.

Usage

```
h2o.lstrip(x, set = " ")
```

Arguments

x	The column whose strings should be lstrip-ed.
set	string of characters to be removed

h2o.makeGLMModel *Set betas of an existing H2O GLM Model*

Description

This function allows setting betas of an existing glm model.

Usage

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

Arguments

model	an H2OModel corresponding from a h2o.glm call.
beta	a new set of betas (a named vector)

h2o.match *Value Matching in H2O*

Description

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

Usage

```
h2o.match(x, table, nomatch = 0, incomparables = NULL)

match.H2OFrame(x, table, nomatch = 0, incomparables = NULL)

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)
h2o.match(hex[,5], c("setosa", "versicolor"))
```

---

h2o.mean *Mean of a column*

Description

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

Usage

```
h2o.mean(x, ..., na.rm = TRUE)

## S3 method for class H2OFrame
mean(x, ..., na.rm = TRUE)
```

Arguments

x	An H2OFrame object.
...	Further arguments to be passed from or to other methods.
na.rm	A logical value indicating whether NA or missing values should be stripped before the computation.

See Also

mean for the base R implementation.

Examples

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

h2o.mean_residual_deviance *Retrieve the Mean Residual Deviance value*

Description

Retrieves the Mean Residual Deviance value from an H2O model. If "train", "valid", and "xval" parameters are FALSE (default), then the training Mean Residual Deviance value is returned. If more than one parameter is set to TRUE, then a named vector of Mean Residual Deviances are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.mean_residual_deviance(object, train = FALSE, valid = FALSE,
  xval = FALSE)
```

Arguments

object	An H2OModel object.
train	Retrieve the training Mean Residual Deviance
valid	Retrieve the validation Mean Residual Deviance
xval	Retrieve the cross-validation Mean Residual Deviance

Examples

```
library(h2o)

h <- h2o.init()
fr <- as.h2o(iris)

m <- h2o.deeplearning(x=2:5,y=1,training_frame=fr)

h2o.mean_residual_deviance(m)
```

h2o.median *H2O Median*

Description

Compute the median of an H2OFrame.

Usage

```
h2o.median(x, na.rm = TRUE)

## S3 method for class H2OFrame
median(x, na.rm = TRUE)
```

Arguments

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

Examples

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

---

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, by.x = NULL, by.y = NULL,
  method = "hash")
```

Arguments

x,y	H2OFrame objects
all.x	If all.x is true, all rows in the x will be included, even if there is no matching row in y, and vice-versa for all.y.
all.y	see all.x
by.x	x columns used for merging.
by.y	y columns used for merging.
method	auto, radix, or hash (default)

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	(Optional) A value or a list of values between 0.0 and 1.0.
metric	(Optional) 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.mktime *Compute msec since the Unix Epoch*

Description

Compute msec since the Unix Epoch

Usage

```
h2o.mktime(year = 1970, month = 0, day = 0, hour = 0, minute = 0,
  second = 0, msec = 0)
```

Arguments

year	Defaults to 1970
month	zero based (months are 0 to 11)
day	zero based (days are 0 to 30)
hour	hour
minute	minute
second	second
msec	msec

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

Description

Converts the entries of an H2OFrame object from milliseconds to months (on a 1 to 12 scale).

Usage

```
h2o.month(x)

month(x)

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

Arguments

x	An H2OFrame object.

Value

An 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. If "train", "valid", and "xval" parameters are FALSE (default), then the training MSEvalue is returned. If more than one parameter is set to TRUE, then a named vector of MSEs are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

object	An H2OModelMetrics object of the correct type.
train	Retrieve the training MSE
valid	Retrieve the validation MSE
xval	Retrieve the cross-validation MSE

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.nacnt *Count of NAs per column*

Description

Gives the count of NAs per column.

Usage

```
h2o.nacnt(x)
```

Arguments

x	An H2OFrame object.

Examples

```
h2o.init()
iris.hex <- as.h2o(iris)
h2o.nacnt(iris.hex)  # should return all 0s
h2o.insertMissingValues(iris.hex)
h2o.nacnt(iris.hex)
```

h2o.naiveBayes *Naive Bayes Model in H2O*

Description

Compute naive Bayes probabilities on an H2O dataset.

Usage

```
h2o.naiveBayes(x, y, training_frame, validation_frame = NULL, model_id,
  ignore_const_cols = TRUE, laplace = 0, threshold = 0.001, eps = 0,
  nfolds = 0, fold_column = NULL, fold_assignment = c("AUTO", "Random",
  "Modulo", "Stratified"), seed, keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE, compute_metrics = TRUE,
  max_runtime_secs = 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. The response must be a categorical variable with at least two levels.
training_frame	An H2OFrame object containing the variables in the model.
validation_frame	An H2OFrame object containing the variables in the model. Defaults to NULL.
model_id	(Optional) The unique id assigned to the resulting model. If none is given, an id will automatically be generated.
ignore_const_cols	A logical value indicating whether or not to ignore all the constant columns in the training frame.
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.
nfolds	(Optional) Number of folds for cross-validation.
fold_column	(Optional) Column with cross-validation fold index assignment per observation
fold_assignment	Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.
seed	Seed for random numbers (affects sampling).
keep_cross_validation_predictions	Whether to keep the predictions of the cross-validation models
keep_cross_validation_fold_assignment	Whether to keep the cross-validation fold assignment.
compute_metrics	A logical value indicating whether model metrics should be computed. Set to FALSE to reduce the runtime of the algorithm.
max_runtime_secs	Maximum allowed runtime in seconds for model training. Use 0 to disable.

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.

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 H2OBinomialModel if the response has two categorical levels, and H2OMultinomialModel otherwise.

Examples

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

---

h2o.nchar *String length*

Description

String length

Usage

```
h2o.nchar(x)
```

Arguments

x	The column whose string lengths will be returned.

---

h2o.networkTest *View Network Traffic Speed*

Description

View speed with various file sizes.

Usage

```
h2o.networkTest()
```

Value

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

h2o.nlevels *Get the number of factor levels for this frame.*

Description

Get the number of factor levels for this frame.

Usage

```
h2o.nlevels(x)
```

Arguments

x	An H2OFrame object.

See Also

nlevels for the base R method.

h2o.no_progress *Disable Progress Bar*

Description

Disable Progress Bar

Usage

```
h2o.no_progress()
```

h2o.null_deviance *Retrieve the null deviance If "train", "valid", and "xval" parameters are FALSE (default), then the training null deviance value is returned. If more than one parameter is set to TRUE, then a named vector of null deviances are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the null deviance If "train", "valid", and "xval" parameters are FALSE (default), then the training null deviance value is returned. If more than one parameter is set to TRUE, then a named vector of null deviances are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.null_deviance(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

object	An H2OModel or H2OModelMetrics
train	Retrieve the training null deviance
valid	Retrieve the validation null deviance
xval	Retrieve the cross-validation null deviance

`h2o.null_dof` *Retrieve the null degrees of freedom If "train", "valid", and "xval" parameters are FALSE (default), then the training null degrees of freedom value is returned. If more than one parameter is set to TRUE, then a named vector of null degrees of freedom are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the null degrees of freedom If "train", "valid", and "xval" parameters are FALSE (default), then the training null degrees of freedom value is returned. If more than one parameter is set to TRUE, then a named vector of null degrees of freedom are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.null_dof(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

object	An H2OModel or H2OModelMetrics
train	Retrieve the training null degrees of freedom
valid	Retrieve the validation null degrees of freedom
xval	Retrieve the cross-validation null degrees of freedom

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.num_valid_substrings *Count of substrings >= 2 chars that are contained in file*

Description

Find the count of all possible substrings >= 2 chars that are contained in the specified line-separated text file.

Usage

```
h2o.num_valid_substrings(x, path)
```

Arguments

x	The column on which to calculate the number of valid substrings.
path	Path to text file containing line-separated strings to be referenced.

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:
h2o.init()

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

# Not run to avoid windows being opened during R CMD check
# 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, na.strings = NULL,
  blocking = FALSE, parse_type = NULL, chunk_size = NULL)
```

Arguments

data	An H2OFrame 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) An 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.
na.strings	(Optional) H2O will interpret these strings as missing.
blocking	(Optional) Tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.
parse_type	(Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"
chunk_size	size of chunk of (input) data in bytes

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, na.strings = NULL,
  parse_type = NULL)
```

Arguments

data	An H2OFrame 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) An 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.
na.strings	(Optional) H2O will interpret these strings as missing.
parse_type	(Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"

---

h2o.performance *Model Performance Metrics in H2O*

Description

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

Usage

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

Arguments

model	An H2OModel object
newdata	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 newdata is passed in, then train, valid, and xval 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).
xval	A logical value indicating whether to return the cross-validation metrics (constructed during training).
data	(DEPRECATED) An H2OFrame. This argument is now called 'newdata'.

Value

Returns an object of the H2OModelMetrics subclass.

Examples

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

---

h2o.prcomp *Principal Components Analysis*

Description

Principal components analysis of an H2O data frame using the power method to calculate the singular value decomposition of the Gram matrix.

Usage

```
h2o.prcomp(training_frame, x, k, model_id, ignore_const_cols = TRUE,
  max_iterations = 1000, transform = c("NONE", "DEMEAN", "DESCALE",
  "STANDARDIZE"), pca_method = c("GramSVD", "Power", "Randomized", "GLRM"),
  use_all_factor_levels = FALSE, compute_metrics = TRUE,
  impute_missing = FALSE, seed, max_runtime_secs = 0)
```

Arguments

training_frame	An H2OFrame object containing the variables in the model.
x	(Optional) A vector containing the data columns on which SVD operates.
k	The number of principal components to be computed. This must be between 1 and min(ncol(training_frame), nrow(training_frame)) inclusive.
model_id	(Optional) The unique hex key assigned to the resulting model. Automatically generated if none is provided.
ignore_const_cols	A logical value indicating whether or not to ignore all the constant columns in the training frame.
max_iterations	The maximum number of iterations to run each power iteration loop. Must be between 1 and 1e6 inclusive.
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).

pca_method
: A character string that indicates how PCA should be calculated. Possible values are "GramSVD": distributed computation of the Gram matrix followed by a local SVD using the JAMA package, "Power": computation of the SVD using the power iteration method, "Randomized": approximate SVD by projecting onto a random subspace (see references), "GLRM": fit a generalized low rank model with an l2 loss function (no regularization) and solve for the SVD using local matrix algebra.

use_all_factor_levels
: (Optional) A logical value indicating whether all factor levels should be included in each categorical column expansion. If FALSE, the indicator column corresponding to the first factor level of every categorical variable will be dropped. Defaults to FALSE.

compute_metrics
: (Optional) A logical value indicating whether to compute metrics on the training data, which requires additional calculation time. Only used if pca_method = "GLRM". Defaults to TRUE.

impute_missing
: (Optional) A logical value indicating whether missing values should be imputed with the mean of the corresponding column. This is necessary if too many entries are NA when using methods like GramSVD. Defaults to FALSE.

seed
: (Optional) Random seed used to initialize the right singular vectors at the beginning of each power method iteration.

max_runtime_secs
: Maximum allowed runtime in seconds for model training. Use 0 to disable.

Value

Returns an object of class H2ODimReductionModel.

References

N. Halko, P.G. Martinsson, J.A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions[http://arxiv.org/abs/0909.4061]. SIAM Rev., Survey and Review section, Vol. 53, num. 2, pp. 217-288, June 2011.

See Also

h2o.svd, h2o.glrm

Examples

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

h2o.proj_archetypes *Convert Archetypes to Features from H2O GLRM Model*

Description

Project each archetype in an H2O GLRM model into the corresponding feature space from the H2O training frame.

Usage

```
h2o.proj_archetypes(object, data, reverse_transform = FALSE)
```

Arguments

object	An H2ODimReductionModel object that represents the model containing archetypes to be projected.
data	An H2OFrame object representing the training data for the H2O GLRM model.
reverse_transform	(Optional) A logical value indicating whether to reverse the transformation from model-building by re-scaling columns and adding back the offset to each column of the projected archetypes.

Value

Returns an H2OFrame object containing the projection of the archetypes down into the original feature space, where each row is one archetype.

See Also

h2o.glrm for making an H2ODimReductionModel.

Examples

```
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(path = irisPath)
iris.glrm <- h2o.glrm(training_frame = iris.hex, k = 4, loss = "Quadratic",
                      multi_loss = "Categorical", max_iterations = 1000)
iris.parch <- h2o.proj_archetypes(iris.glrm, iris.hex)
head(iris.parch)
```

h2o.quantile *Quantiles of H2O Frames.*

Description

Obtain and display quantiles for H2O parsed data.

Usage

```
h2o.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"), weights_column = NULL, ...)

## 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"), weights_column = NULL, ...)
```

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).
weights_column	(Optional) String name of the observation weights column in x or an H2OFrame object with a single numeric column of observation weights.
...	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)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(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])
```

---

h2o.r2 *Retrieve the R2 value*

Description

Retrieves the R2 value from an H2O model. If "train", "valid", and "xval" parameters are FALSE (default), then the training R2 value is returned. If more than one parameter is set to TRUE, then a named vector of R2s are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.r2(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

object	An H2OModel object.
train	Retrieve the training R2
valid	Retrieve the validation set R2 if a validation set was passed in during model build time.
xval	Retrieve the cross-validation R2

Examples

```
library(h2o)

h <- h2o.init()
fr <- as.h2o(iris)

m <- h2o.deeplearning(x=2:5,y=1,training_frame=fr)

h2o.r2(m)
```

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 = NULL,
  ignore_const_cols = TRUE, checkpoint, mtries = -1,
  col_sample_rate_change_per_level = 1, sample_rate = 0.632,
  sample_rate_per_class, col_sample_rate_per_tree = 1,
  build_tree_one_node = FALSE, ntrees = 50, max_depth = 20,
  min_rows = 1, nbins = 20, nbins_top_level, nbins_cats = 1024,
  binomial_double_trees = FALSE, balance_classes = FALSE,
  class_sampling_factors, max_after_balance_size = 5, seed,
  offset_column = NULL, weights_column = NULL, nfolds = 0,
  fold_column = NULL, fold_assignment = c("AUTO", "Random", "Modulo",
  "Stratified"), keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  score_each_iteration = FALSE, score_tree_interval = 0,
  stopping_rounds = 0, stopping_metric = c("AUTO", "deviance", "logloss",
  "MSE", "AUC", "r2", "misclassification"), stopping_tolerance = 0.001,
  max_runtime_secs = 0, min_split_improvement, histogram_type = c("AUTO",
  "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin"))
```

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. Default is NULL.
ignore_const_cols	A logical value indicating whether or not to ignore all the constant columns in the training frame.
checkpoint	"Model checkpoint (either key or H2ODeepLearningModel) to resume training with."

`mtries` Number of variables randomly sampled as candidates at each split. If set to -1, defaults to sqrtp for classification, and p/3 for regression, where p is the number of predictors.

`col_sample_rate_change_per_level`
Relative change of the column sampling rate for every level (from 0.0 to 2.0)

`sample_rate` Row sample rate per tree (from `0.0` to `1.0`)

`sample_rate_per_class`
Row sample rate per tree per class (one per class, from `0.0` to `1.0`)

`col_sample_rate_per_tree`
Column sample rate per tree (from `0.0` 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` For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point.

`nbins_top_level`
For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level.

`nbins_cats` For categorical columns (factors), build a histogram of this many bins, then split at the best point. Higher values can lead to more overfitting.

`binomial_double_trees`
For binary classification: Build 2x as many trees (one per class) - can lead to higher accuracy.

`balance_classes`
logical, indicates whether or not to 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). Ignored if balance_classes is FALSE, which is the default behavior.

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

`offset_column` Specify the offset column.

`weights_column` Specify the weights column.

`nfolds` (Optional) Number of folds for cross-validation.

`fold_column` (Optional) Column with cross-validation fold index assignment per observation

`fold_assignment`
: Cross-validation fold assignment scheme, if fold_column is not specified, must be "AUTO", "Random", "Modulo", or "Stratified". The Stratified option will stratify the folds based on the response variable, for classification problems.

`keep_cross_validation_predictions`
: Whether to keep the predictions of the cross-validation models

`keep_cross_validation_fold_assignment`
: Whether to keep the cross-validation fold assignment.

`score_each_iteration`
: Attempts to score each tree.

`score_tree_interval`
: Score the model after every so many trees. Disabled if set to 0.

`stopping_rounds`
: Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve (by stopping_tolerance) for k=stopping_rounds scoring events. Can only trigger after at least 2k scoring events. Use 0 to disable.

`stopping_metric`
: Metric to use for convergence checking, only for _stopping_rounds > 0 Can be one of "AUTO", "deviance", "logloss", "MSE", "AUC", "r2", "misclassification".

`stopping_tolerance`
: Relative tolerance for metric-based stopping criterion (if relative improvement is not at least this much, stop)

`max_runtime_secs`
: Maximum allowed runtime in seconds for model training. Use 0 to disable.

`min_split_improvement`
: Minimum relative improvement in squared error reduction for a split to happen.

`histogram_type` What type of histogram to use for finding optimal split points Can be one of "AUTO", "UniformAdaptive", "Random", "QuantilesGlobal" or "RoundRobin".

`...` (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 and types of columns.

Value

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

See Also

rbind for the base R method.

Examples

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

h2o.reconstruct *Reconstruct Training Data via H2O GLRM Model*

Description

Reconstruct the training data and impute missing values from the H2O GLRM model by computing the matrix product of X and Y, and transforming back to the original feature space by minimizing each column's loss function.

Usage

```
h2o.reconstruct(object, data, reverse_transform = FALSE)
```

Arguments

object	An H2ODimReductionModel object that represents the model to be used for reconstruction.
data	An H2OFrame object representing the training data for the H2O GLRM model. Used to set the domain of each column in the reconstructed frame.
reverse_transform	(Optional) A logical value indicating whether to reverse the transformation from model-building by re-scaling columns and adding back the offset to each column of the reconstructed frame.

Value

Returns an H2OFrame object containing the approximate reconstruction of the training data;

See Also

h2o.glrm for making an H2ODimReductionModel.

Examples

```
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(path = irisPath)
iris.glrm <- h2o.glrm(training_frame = iris.hex, k = 4, transform = "STANDARDIZE",
                      loss = "Quadratic", multi_loss = "Categorical", max_iterations = 1000)
iris.rec <- h2o.reconstruct(iris.glrm, iris.hex, reverse_transform = TRUE)
head(iris.rec)
```

---

h2o.relevel *Reorders levels of an H2O factor, similarly to standard R's relevel.*

Description

The levels of a factor are reordered os that the reference level is at level 0, remaining levels are moved down as needed.

Usage

```
h2o.relevel(x, y)
```

Arguments

x	factor column in h2o frame
y	reference level (string)

Value

new reordered factor column

`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(timeout_secs = 0)
```

Arguments

`timeout_secs`	Timeout in seconds. Default is no timeout.

See Also

`h2o.rm`

Examples

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

`h2o.removeVecs` *Delete Columns from an H2OFrame*

Description

Delete the specified columns from the H2OFrame. Returns an H2OFrame without the specified columns.

Usage

```
h2o.removeVecs(data, cols)
```

Arguments

data	The H2OFrame.
cols	The columns to remove.

---

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 an H2OFrame vector of the same type as x

---

h2o.residual_deviance *Retrieve the residual deviance If "train", "valid", and "xval" parameters are FALSE (default), then the training residual deviance value is returned. If more than one parameter is set to TRUE, then a named vector of residual deviances are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the residual deviance If "train", "valid", and "xval" parameters are FALSE (default), then the training residual deviance value is returned. If more than one parameter is set to TRUE, then a named vector of residual deviances are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.residual_deviance(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

object	An H2OModel or H2OModelMetrics
train	Retrieve the training residual deviance
valid	Retrieve the validation residual deviance
xval	Retrieve the cross-validation residual deviance

`h2o.residual_dof` *Retrieve the residual degrees of freedom If "train", "valid", and "xval" parameters are FALSE (default), then the training residual degrees of freedom value is returned. If more than one parameter is set to TRUE, then a named vector of residual degrees of freedom are returned, where the names are "train", "valid" or "xval".*

Description

Retrieve the residual degrees of freedom If "train", "valid", and "xval" parameters are FALSE (default), then the training residual degrees of freedom value is returned. If more than one parameter is set to TRUE, then a named vector of residual degrees of freedom are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.residual_dof(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OModel or H2OModelMetrics
`train`	Retrieve the training residual degrees of freedom
`valid`	Retrieve the validation residual degrees of freedom
`xval`	Retrieve the cross-validation residual degrees of freedom

---

`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)
```

Arguments

`ids`	The object or hex key associated with the object to be removed or a vector/list of those things.

See Also

`h2o.assign`, `h2o.ls`

h2o.round *Round doubles/floats to the given number of decimal places.*

Description

Round doubles/floats to the given number of decimal places.

Usage

```
h2o.round(x, digits = 0)

round(x, digits = 0)
```

Arguments

x	An H2OFrame object.
digits	Number of decimal places to round doubles/floats. Rounding to a negative number of decimal places is

See Also

round for the base R implementation.

h2o.rstrip *Strip set from right*

Description

Return a copy of the target column with leading characters removed. The set argument is a string specifying the set of characters to be removed. If omitted, the set argument defaults to removing whitespace.

Usage

```
h2o.rstrip(x, set = " ")
```

Arguments

x	The column whose strings should be rstrip-ed.
set	string of characters to be removed

h2o.runif *Produce 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)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(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, path = "", force = FALSE)
```

Arguments

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

Details

In the case of existing files `force = 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)
# h2o.init()
# prostate.hex <- h2o.importFile(path = paste("https://raw.github.com",
#   "h2oai/h2o-2/master/smalldata/logreg/prostate.csv", sep = "/"),
#   destination_frame = "prostate.hex")
# prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#   training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# h2o.saveModel(object = prostate.glm, path = "/Users/UserName/Desktop", force=TRUE)

## End(Not run)
```

---

h2o.scale *Scaling and Centering of an H2OFrame*

Description

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

Usage

```
h2o.scale(x, center = TRUE, scale = TRUE)

## 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)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(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.

---

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

Description

Obtain the standard deviation of a column of data.

Usage

```
h2o.sd(x, na.rm = FALSE)

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

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

---

h2o.sdev *Retrieve the standard deviations of principal components*

Description

Retrieve the standard deviations of principal components

Usage

```
h2o.sdev(object)
```

Arguments

object	An H2ODimReductionModel object.

---

h2o.setLevels *Set Levels of H2O Factor Column*

Description

Works on a single categorical vector. New domains must be aligned with the old domains. This call has SIDE EFFECTS and mutates the column in place (does not make a copy).

Usage

```
h2o.setLevels(x, levels)
```

Arguments

x	A single categorical column.
levels	A character vector specifying the new levels. The number of new levels must match the number of old levels.

---

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

Description

Set the Time Zone on the H2O Cloud

Usage

```
h2o.setTimezone(tz)
```

Arguments

tz	The desired timezone.

---

h2o.show_progress *Enable Progress Bar*

Description

Enable Progress Bar

Usage

```
h2o.show_progress()
```

h2o.shutdown *Shut Down H2O Instance*

Description

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

Usage

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

Arguments

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)
h2o.init()
h2o.shutdown()

## End(Not run)
```

h2o.signif *Round doubles/floats to the given number of significant digits.*

Description

Round doubles/floats to the given number of significant digits.

Usage

```
h2o.signif(x, digits = 6)

signif(x, digits = 6)
```

Arguments

x	An H2OFrame object.
digits	Number of significant digits to round doubles/floats.

See Also

signif for the base R implementation.

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, seed = -1)
```

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.
seed	Random seed.

Examples

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

---

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

h2o.strsplit *String Split*

Description

String Split

Usage

```
h2o.strsplit(x, split)
```

Arguments

x	The column whose strings must be split.
split	The pattern to split on.

h2o.sub *String Substitute*

Description

Creates a copy of the target column in which each string has the first occurence of the regex pattern replaced with the replacement substring.

Usage

```
h2o.sub(pattern, replacement, x, ignore.case = FALSE)
```

Arguments

pattern	The pattern to replace.
replacement	The replacement pattern.
x	The column on which to operate.
ignore.case	Case sensitive or not

h2o.substring *Substring*

Description

Returns a copy of the target column that is a substring at the specified start and stop indices, inclusive. If the stop index is not specified, then the substring extends to the end of the original string. If start is longer than the number of characters in the original string, or is greater than stop, an empty string is returned. Negative start is coerced to 0.

Usage

```
h2o.substring(x, start, stop = "[]")

h2o.substr(x, start, stop = "[]")
```

Arguments

x	The column on which to operate.
start	The index of the first element to be included in the substring.
stop	Optional, The index of the last element to be included in the substring.

h2o.summary *Summarizes the columns of an H2OFrame.*

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

```
h2o.summary(object, factors = 6L, exact_quantiles = FALSE, ...)

## S3 method for class H2OFrame
summary(object, factors, exact_quantiles, ...)
```

Arguments

object	An H2OFrame object.
factors	The number of factors to return in the summary. Default is the top 6.
exact_quantiles	Compute exact quantiles or use approximation. Default is to use approximation.
...	Further arguments passed to or from other methods.

Details

By default it uses approximated version of quantiles computation, however, user can modify this behavior by setting up exact_quantiles argument to true.

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)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package="h2o")
prostate.hex = h2o.importFile(path = prosPath)
summary(prostate.hex)
summary(prostate.hex$GLEASON)
summary(prostate.hex[,4:6])
summary(prostate.hex, exact_quantiles=TRUE)
```

h2o.svd *Singular Value Decomposition*

Description

Singular value decomposition of an H2O data frame using the power method.

Usage

```
h2o.svd(training_frame, x, nv, destination_key, max_iterations = 1000,
  transform = "NONE", svd_method = c("GramSVD", "Power", "Randomized"),
  seed, use_all_factor_levels, max_runtime_secs = 0)
```

Arguments

training_frame	An H2OFrame object containing the variables in the model.
x	(Optional) A vector containing the data columns on which SVD operates.
nv	The number of right singular vectors to be computed. This must be between 1 and min(ncol(training_frame), nrow(training_frame)) inclusive.
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 each power iteration loop. Must be between 1 and 1e6 inclusive.
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).
svd_method	A character string that indicates how SVD should be calculated. Possible values are "GramSVD": distributed computation of the Gram matrix followed by a local SVD using the JAMA package, "Power": computation of the SVD using the power iteration method, "Randomized": approximate SVD by projecting onto a random subspace (see references).
seed	(Optional) Random seed used to initialize the right singular vectors at the beginning of each power method iteration.
use_all_factor_levels	(Optional) A logical value indicating whether all factor levels should be included in each categorical column expansion. If FALSE, the indicator column corresponding to the first factor level of every categorical variable will be dropped. Defaults to TRUE.
max_runtime_secs	Maximum allowed runtime in seconds for model training. Use 0 to disable.

Value

Returns an object of class H2ODimReductionModel.

References

N. Halko, P.G. Martinsson, J.A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions[http://arxiv.org/abs/0909.4061]. SIAM Rev., Survey and Review section, Vol. 53, num. 2, pp. 217-288, June 2011.

Examples

```
library(h2o)
h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.uploadFile(path = ausPath)
h2o.svd(training_frame = australia.hex, nv = 8)
```

---

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, dense = TRUE)

table.H2OFrame(x, y = NULL, dense = TRUE)
```

Arguments

x	An H2OFrame object with at most two columns.
y	An H2OFrame similar to x, or NULL.
dense	A logical for dense representation, which lists only non-zero counts, 1 combination per row. Set to FALSE to expand counts across all combinations.

Value

Returns a tabulated H2OFrame object.

Examples

```
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(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.tabulate *Tabulation between Two Columns of an H2OFrame*

Description

Simple Co-Occurrence based tabulation of X vs Y, where X and Y are two Vecs in a given dataset. Uses histogram of given resolution in X and Y. Handles numerical/categorical data and missing values. Supports observation weights.

Usage

```
h2o.tabulate(data, x, y, weights_column = NULL, nbins_x = 50,
  nbins_y = 50)
```

Arguments

data	An H2OFrame object.
x	predictor column
y	response column
weights_column	(optional) observation weights column
nbins_x	number of bins for predictor column
nbins_y	number of bins for response column

Value

Returns two TwoDimTables of 3 columns each count_table: X Y counts response_table: X meanY counts

Examples

```
library(h2o)
h2o.init()
df <- as.h2o(iris)
tab <- h2o.tabulate(data = df, x = "Sepal.Length", y = "Petal.Width",
             weights_column = NULL, nbins_x = 10, nbins_y = 10)
plot(tab)
```

`h2o.tolower` *To Lower*

Description

To Lower

Usage

```
h2o.tolower(x)
```

Arguments

`x`	An H2OFrame object whose strings should be lower'd

`h2o.totss` *Get the total sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training totss value is returned. If more than one parameter is set to TRUE, then a named vector of totss' are returned, where the names are "train", "valid" or "xval".*

Description

Get the total sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training totss value is returned. If more than one parameter is set to TRUE, then a named vector of totss' are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.totss(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training total sum of squares
`valid`	Retrieve the validation total sum of squares
`xval`	Retrieve the cross-validation total sum of squares

`h2o.tot_withinss` *Get the total within cluster sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training tot_withinss value is returned. If more than one parameter is set to TRUE, then a named vector of tot_withinss' are returned, where the names are "train", "valid" or "xval".*

Description

Get the total within cluster sum of squares. If "train", "valid", and "xval" parameters are FALSE (default), then the training tot_withinss value is returned. If more than one parameter is set to TRUE, then a named vector of tot_withinss' are returned, where the names are "train", "valid" or "xval".

Usage

```
h2o.tot_withinss(object, train = FALSE, valid = FALSE, xval = FALSE)
```

Arguments

`object`	An H2OClusteringModel object.
`train`	Retrieve the training total within cluster sum of squares
`valid`	Retrieve the validation total within cluster sum of squares
`xval`	Retrieve the cross-validation total within cluster sum of squares

---

`h2o.toupper` *To Upper*

Description

To Upper

Usage

```
h2o.toupper(x)
```

Arguments

`x`	An H2OFrame object whose strings should be upper'd

h2o.trim *Trim Space*

Description

Trim Space

Usage

```
h2o.trim(x)
```

Arguments

x	The column whose strings should be trimmed.

h2o.unique *H2O Unique*

Description

Extract unique values in the column.

Usage

```
h2o.unique(x)
```

Arguments

x	An H2OFrame object.

h2o.var *Variance of a column or covariance of columns.*

Description

Compute the variance or covariance matrix of one or two H2OFrames.

Usage

```
h2o.var(x, y = NULL, na.rm = FALSE, use)

var(x, y = NULL, na.rm = FALSE, use)
```

Arguments

x	An H2OFrame object.
y	NULL (default) or an H2OFrame. The default is equivalent to y = x.
na.rm	logical. Should missing values be removed?
use	An optional character string indicating how to handle missing values. This must be one of the following:

See Also

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

Examples

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

h2o.week — *Convert Milliseconds to Week of Week Year in H2O Datasets*

Description

Converts the entries of an H2OFrame object from milliseconds to weeks of the week year (starting from 1).

Usage

```
h2o.week(x)

week(x)

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

Arguments

x	An H2OFrame object.

Value

An H2OFrame object containing the entries of x converted to weeks of the week year.

See Also

h2o.month

h2o.weights — *Retrieve the respective weight matrix*

Description

Retrieve the respective weight matrix

Usage

```
h2o.weights(object, matrix_id = 1)
```

Arguments

object	An H2OModel or H2OModelMetrics
matrix_id	An integer, ranging from 1 to number of layers + 1, that specifies the weight matrix to return.

h2o.which *Which indices are TRUE?*

Description

Give the TRUE indices of a logical object, allowing for array indices.

Usage

```
h2o.which(x)
```

Arguments

x	An H2OFrame object.

See Also

which for the base R method.

Examples

```
h2o.init()
iris.hex <- as.h2o(iris)
h2o.which(iris.hex[,1]==4.4)
```

h2o.withinss *Get the Within SS*

Description

Get the Within SS

Usage

```
h2o.withinss(object)
```

Arguments

object	An H2OClusteringModel object.

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

Description

Convert the entries of an 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

An 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

H2OClusteringModel-class *The H2OClusteringModel object.*

Description

This virtual class represents a clustering model built by H2O.

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

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

`algorithm` A `character` string specifying the algorithm that was 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` containing all parameters used to fit the model.

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

size The number of points in each cluster.

totss Total sum of squared error to grand mean.

withinss A vector of within-cluster sum of squared error.

tot_withinss Total within-cluster sum of squared error.

betweenss Between-cluster sum of squared error.

`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.

`proxy` A `character` specifying the proxy path of the H2O cloud.

`https` Set this to TRUE to use https instead of http.

`insecure` Set this to TRUE to disable SSL certificate checking.

`username` Username to login with.

`password` Password to login with.

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

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

Description

Operators to extract or replace parts of H2OFrame objects.

Usage

```
## S3 method for class H2OFrame
data[row, col, drop = TRUE]

## S3 method for class H2OFrame
x$name

## S3 method for class H2OFrame
x[[i, exact = TRUE]]

## S3 method for class H2OFrame
x$name

## S3 method for class H2OFrame
x[[i, exact = TRUE]]

## S3 replacement method for class H2OFrame
data[row, col, ...] <- value

## S3 replacement method for class H2OFrame
data$name <- value

## S3 replacement method for class H2OFrame
data[[name]] <- value
```

Arguments

data	object from which to extract element(s) or in which to replace element(s).
row	index specifying row element(s) to extract or replace. Indices are numeric or character vectors or empty (missing) or will be matched to the names.
col	index specifying column element(s) to extract or replace.
drop	Unused
x	An H2OFrame
name	a literal character string or a name (possibly backtick quoted).
i	index
exact	controls possible partial matching of [[when extracting a character
...	Further arguments passed to or from other methods.
value	To be assigned

H2OGrid-class *H2O Grid*

Description

A class to contain the information about grid results

Format grid object in user-friendly way

Usage

```
## S4 method for signature H2OGrid
show(object)
```

Arguments

object	an H2OGrid object.

Slots

grid_id the final identifier of grid

model_ids list of model IDs which are included in the grid object

hyper_names list of parameter names used for grid search

failed_params list of model parameters which caused a failure during model building, it can contain a null value

failure_details list of detailed messages which correspond to failed parameters field

failure_stack_traces list of stack traces corresponding to model failures reported by failed_params and failure_details fields

failed_raw_params list of failed raw parameters

summary_table table of models built with parameters and metric information.

See Also

H2OModel for the final model types.

`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

`model_id` A `character` string specifying the key for the model fit in the H2O cloud's 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

`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)

## S4 method for signature H2ODimReductionMetrics
show(object)
```

Arguments

object	An H2OModelMetrics object

---

housevotes *United States Congressional Voting Records 1984*

Description

This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA lists nine different types of votes: voted for, paired for, and announced for (these three simplified to yea), voted against, paired against, and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three simplified to an unknown disposition).

Format

A data frame with 435 rows and 17 columns

Source

Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc., Washington, D.C., 1985

References

Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

---

iris *Edgar Anderson's Iris Data*

Description

Measurements in centimeters of the sepal length and width and petal length and width, respectively, for three species of iris flowers.

Format

A data frame with 150 rows and 5 columns

Source

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7, Part II, 179-188.

The data were collected by Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59, 2-5.

is.character *Check if character*

Description

Check if character

Usage

```
is.character(x)
```

Arguments

x	An H2OFrame object

is.factor *Check if factor*

Description

Check if factor

Usage

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

Arguments

x	An H2OFrame object

is.numeric *Check if numeric*

Description

Check if numeric

Usage

```
is.numeric(x)
```

Arguments

x	An H2OFrame object

Logical-or *Logical or for H2OFrames*

Description

Logical or for H2OFrames

Usage

```
"||"(x, y)
```

Arguments

x	An H2OFrame object
y	An H2OFrame object

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)

getWithinSS(object)

getTotWithinSS(object)

getBetweenSS(object)

getTotSS(object)

getIterations(object)

getClusterSizes(object)
```

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

## S4 method for signature H2OClusteringModel
getCentersStd(object)

## S4 method for signature H2OClusteringModel
getWithinSS(object)

## S4 method for signature H2OClusteringModel
getTotWithinSS(object)

## S4 method for signature H2OClusteringModel
getBetweenSS(object)

## S4 method for signature H2OClusteringModel
getTotSS(object)

## S4 method for signature H2OClusteringModel
getIterations(object)

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

Arguments

object	an H2OModel class object.

`na.omit.H2OFrame` *Remove Rows With NAs*

Description

Remove Rows With NAs

Usage

```
## S3 method for class H2OFrame
na.omit(object, ...)
```

Arguments

object	H2OFrame object
...	Ignored

names.H2OFrame *Column names of an H2OFrame*

Description

Column names of an H2OFrame

Usage

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

Arguments

x	An H2OFrame

Ops.H2OFrame *S3 Group Generic Functions for H2O*

Description

Methods for group generic functions and H2O objects.

Usage

```
## S3 method for class H2OFrame
Ops(e1, e2)

## S3 method for class H2OFrame
Math(x, ...)

## S3 method for class H2OFrame
Math(x, ...)

## S3 method for class H2OFrame
Math(x, ...)

## S3 method for class H2OFrame
Summary(x, ..., na.rm)

## S3 method for class H2OFrame
!x

## S3 method for class H2OFrame
is.na(x)
```

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

log(x, ...)

log10(x)

log2(x)

log1p(x)

trunc(x, ...)

x %*% y

nrow.H2OFrame(x)

ncol.H2OFrame(x)

## S3 method for class H2OFrame
length(x)

h2o.length(x)

## S3 replacement method for class H2OFrame
names(x) <- value

colnames(x) <- value
```

Arguments

e1	object
e2	object
x	object
...	Further arguments passed to or from other methods.
na.rm	logical. whether or not missing values should be removed
y	object
value	To be assigned

`plot.H2OModel` *Plot an H2O Model*

Description

Plots training set (and validation set if available) scoring history for an H2O Model

Usage

```
## S3 method for class H2OModel
plot(x, timestep = "AUTO", metric = "AUTO", ...)
```

Arguments

x	A fitted H2OModel object for which the scoring history plot is desired.
timestep	A unit of measurement for the x-axis.
metric	A unit of measurement for the y-axis.
...	additional arguments to pass on.

Details

This method dispatches on the type of H2O model to select the correct scoring history. The timestep and metric arguments are restricted to what is available in the scoring history for a particular type of model.

Value

Returns a scoring history plot.

See Also

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

Examples

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

df <- as.h2o(mlbench::mlbench.friedman1(10000,1))
rng <- h2o.runif(df, seed=1234)
train <- df[rng<0.8,]
valid <- df[rng>=0.8,]

gbm <- h2o.gbm(x = 1:10, y = "y", training_frame = train, validation_frame = valid,
  ntrees=500, learn_rate=0.01, score_each_iteration = TRUE)
plot(gbm)
plot(gbm, timestep = "duration", metric = "deviance")
plot(gbm, timestep = "number_of_trees", metric = "deviance")
plot(gbm, timestep = "number_of_trees", metric = "MSE")
```

`plot.H2OTabulate` *Plot an H2O Tabulate Heatmap*

Description

Plots the simple co-occurrence based tabulation of X vs Y as a heatmap, where X and Y are two Vecs in a given dataset.

Usage

```
## S3 method for class H2OTabulate
plot(x, xlab = x$cols[1], ylab = x$cols[2],
  base_size = 12, ...)
```

Arguments

`x`	An H2OTabulate object for which the heatmap plot is desired.
`xlab`	A title for the x-axis. Defaults to what is specified in the given H2OTabulate object.
`ylab`	A title for the y-axis. Defaults to what is specified in the given H2OTabulate object.
`base_size`	Base font size for plot.
`...`	additional arguments to pass on.

Value

Returns a ggplot2-based heatmap of co-occurance.

See Also

`link{h2o.tabulate}`

Examples

```
library(h2o)
h2o.init()
df <- as.h2o(iris)
tab <- h2o.tabulate(data = df, x = "Sepal.Length", y = "Petal.Width",
             weights_column = NULL, nbins_x = 10, nbins_y = 10)
plot(tab)
```

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	An 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. The order of the rows in the results is the same as the order in which the data was loaded, even if some rows fail (for example, due to missing values or unseen factor levels).

Value

Returns an H2OFrame object with probabilites and default predictions.

See Also

h2o.deeplearning, h2o.gbm, h2o.glm, h2o.randomForest for model generation in h2o.

predict_leaf_node_assignment.H2OModel *Predict the Leaf Node Assignment on an H2O Model*

Description

Obtains leaf node assignment from fitted H2O model objects.

Usage

```
predict_leaf_node_assignment.H2OModel(object, newdata, ...)

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

Arguments

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

Details

For every row in the test set, return a set of factors that identify the leaf placements of the row in all the trees in the model. The order of the rows in the results is the same as the order in which the data was loaded

Value

Returns an H2OFrame object with categorical leaf assignment identifiers for each tree in the model.

See Also

h2o.gbm and h2o.randomForest for model generation in h2o.

Examples

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

---

print.H2OFrame *Print An H2OFrame*

Description

Print An H2OFrame

Usage

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

Arguments

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

---

print.H2OTable *Print method for H2OTable objects*

Description

This will print a truncated view of the table if there are more than 20 rows.

Usage

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

Arguments

x	An H2OTable object
header	A logical value dictating whether or not the table name should be printed.
...	Further arguments passed to or from other methods.

Value

The original x object

---

prostate *Prostate Cancer Study*

Description

Baseline exam results on prostate cancer patients from Dr. Donn Young at The Ohio State University Comprehensive Cancer Center.

Format

A data frame with 380 rows and 9 columns

Source

Hosmer and Lemeshow (2000) Applied Logistic Regression: Second Edition.

range.H2OFrame *Range of an H2O Column*

Description

Range of an H2O Column

Usage

```
## S3 method for class H2OFrame
range(..., na.rm = TRUE)
```

Arguments

...	An H2OFrame object.
na.rm	ignore missing values

str.H2OFrame *Display the structure of an H2OFrame object*

Description

Display the structure of an H2OFrame object

Usage

```
## S3 method for class H2OFrame
str(object, ..., cols = FALSE)
```

Arguments

object	An H2OFrame.
...	Further arguments to be passed from or to other methods.
cols	Print the per-column str for the H2OFrame

summary,H2OGrid-method *Format grid object in user-friendly way*

Description

Format grid object in user-friendly way

Usage

```
## S4 method for signature H2OGrid
summary(object, show_stack_traces = FALSE)
```

Arguments

object	an H2OGrid object.
show_stack_traces	a flag to show stack traces for model failures

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)

walking *Muscular Actuations for Walking Subject*

Description

The musculoskeletal model, experimental data, settings files, and results for three-dimensional, muscle-actuated simulations at walking speed as described in Hamner and Delp (2013). Simulations were generated using OpenSim 2.4. The data is available from https://simtk.org/project/xml/downloads.xml?group_id=603.

Format

A data frame with 151 rows and 124 columns

References

Hamner, S.R., Delp, S.L. Muscle contributions to fore-aft and vertical body mass center accelerations over a range of running speeds. Journal of Biomechanics, vol 46, pp 780-787. (2013)

zzz *Shutdown H2O cloud after examples run*

Description

Shutdown H2O cloud after examples run

Examples

```
library(h2o)
h2o.init()
h2o.shutdown(prompt = FALSE)
Sys.sleep(3)
```

&& *Logical and for H2OFrames*

Description

Logical and for H2OFrames

Usage

```
"&&"(x, y)
```

Arguments

x	An H2OFrame object
y	An H2OFrame object

Index