"h2o"

February 17, 2023

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:	3.41.0.6110
Branch:	master
Date:	Fri Feb 17 07:35:38 UTC 2023
License:	Apache License (== 2.0)
Depends:	R (>= 2.13.0), RCurl, jsonlite, statmod, tools, methods, utils

H2O is the scalable open source machine learning platform that offers parallelized implementations of many supervised and unsupervised machine learning algorithms such as Generalized Linear Models (GLM), Gradient Boosting Machines (including XGBoost), Random Forests, Deep Neural Networks (Deep Learning), Stacked Ensembles, Naive Bayes, Generalized Additive Models (GAM), ANOVA GLM, Maximum R GLM (maxrglm), Cox Proportional Hazards, K-Means, PCA, Word2Vec, as well as a fully automatic machine learning algorithm (H2O AutoML). As an example, to run GLM, call `h2o.glm` with the H2O parsed data and parameters (response variable, error distribution, etc.) as arguments.

This package enables the use of the H2O machine learning platform commands in R. To use H2O from R, you must start or connect to the "H2O cluster", the term we use to describe the backend H2O Java engine. 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 you have the H2O cluster running on a remote machine (e.g. AWS EC2), you must provide the IP and port of the remote machine as arguments to the `h2o.init` call.

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

Author(s)

Maintainer: Erin LeDell <erin@h2o.ai>

References

- H2O.ai Homepage
- H2O User Guide
- H2O on GitHub

`.addParm`	*TODO: No objects in this file are being used. Either remove file or use objects.*

Description

Append a <key,value> pair to a list.

Usage

```
.addParm(parms, k, v)
```

Arguments

`parms`	a list to add the <k,v> pair to
`k`	a key, typically the name of some algorithm parameter
`v`	a value, the value of the algorithm parameter

Details

Contained here are a set of helper methods that perform type checking on the value passed in.

`.check_for_ggplot2`	*Stop with a user friendly message if a user is missing the ggplot2 package or has an old version of it.*

Description

Stop with a user friendly message if a user is missing the ggplot2 package or has an old version of it.

Usage

```
.check_for_ggplot2(version = "3.0.0")
```

Arguments

`version`	minimal required ggplot2 version

`.collapse` *Helper Collapse Function*

Description

Collapse a character vector into a ','-sep array of the form: [thing1,thing2,...]

Usage

```
.collapse(v)
```

Arguments

`v`	Character vector.

`.consolidate_varimps` *Consolidate variable importances*

Description

Consolidation works in the following way: 1. if varimp variable is in x => add it to consolidated_varimps 2. for all remaining varimp variables: 1. find the longest prefix of varimp variable that is in x and add it to the consolidated varimp 2. if there was no match, throw an error 3. normalize the consolidated_varimps so they sum up to 1

Usage

```
.consolidate_varimps(model)
```

Arguments

`model`	H2OModel

Value

sorted named vector

`.create_leaderboard` *Create a leaderboard like data frame for* `models`

Description

Create a leaderboard like data frame for `models`

Usage

```
.create_leaderboard(models_info, leaderboard_frame, top_n = 20)
```

Arguments

`models_info`	H2OAutoML object or list of models
`leaderboard_frame`	when provided with list of models, use this frame to calculate metrics
`top_n`	create leaderboard with just top_n models

Value

a data.frame

`.customized_call` *A helper function that makes it easier to override/add params in a function call.*

Description

A helper function that makes it easier to override/add params in a function call.

Usage

```
.customized_call(fun, ..., overridable_defaults = NULL, overrides = NULL)
```

Arguments

`fun`	Function to be called
`...`	Parameters that can't be overridden
`overridable_defaults`	List of parameters and values that can be overridden
`overrides`	Parameters to add/override.

Value

result of `fun`

`.find_appropriate_column_name` *Tries to match a `fuzzy_col_name` with a column name that exists in `cols`.*

Description

Tries to match a `fuzzy_col_name` with a column name that exists in `cols`.

Usage

```
.find_appropriate_column_name(fuzzy_col_name, cols)
```

Arguments

`fuzzy_col_name`	a name to be decoded
`cols`	vector of columns that contain all possible column names, i.e., decode fuzzy_col_name must be in cols

Value

a correct column name

`.get_algorithm` *Get the algoritm used by the model_or_model_id*

Description

Get the algoritm used by the model_or_model_id

Usage

```
.get_algorithm(model_or_model_id, treat_xrt_as_algorithm = FALSE)
```

Arguments

`model_or_model_id`	Model object or a string containing model id
`treat_xrt_as_algorithm`	Try to find out if a model is XRT and if so report it as xrt

Value

algorithm name

`.get_domain_mapping` *Get a mapping between columns and their domains*

Description

Get a mapping between columns and their domains

Usage

```
.get_domain_mapping(model)
```

Arguments

`model`	an h2o model

Value

list containing a mapping from column to its domains (levels)

`.get_feature_count` *Get feature count sorted by the count descending.*

Description

Get feature count sorted by the count descending.

Usage

```
.get_feature_count(column)
```

Arguments

`column`	H2OFrame column

Value

named vector with feature counts

`.get_first_of_family` *Get first of family models*

Description

Get first of family models

Usage

```
.get_first_of_family(models, all_stackedensembles = FALSE)
```

Arguments

`models`	models or model ids
`all_stackedensembles`	if TRUE, select all stacked ensembles

`.h2o.doGET` *Just like doRawGET but fills in the default h2oRestApiVersion if none is provided*

Description

Just like doRawGET but fills in the default h2oRestApiVersion if none is provided

Usage

```
.h2o.doGET(h2oRestApiVersion, urlSuffix, parms, ...)
```

Arguments

`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, a default version is chosen for you.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`...`	(Optional) Additional parameters.

Value

A list object as described above

.h2o.doPOST — *Just like doRawPOST but fills in the default h2oRestApiVersion if none is provided*

Description

Just like doRawPOST but fills in the default h2oRestApiVersion if none is provided

Usage

```
.h2o.doPOST(h2oRestApiVersion, urlSuffix, parms, ...)
```

Arguments

`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, a default version is chosen for you.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`...`	(Optional) Additional parameters.

Value

A list object as described above

.h2o.doRawGET — *Perform a low-level HTTP GET operation on an H2O instance*

Description

Does not do any I/O level error checking. Caller must do its own validations. Does not modify the response payload in any way. Log the request and response if h2o.startLogging() has been called.

Usage

```
.h2o.doRawGET(
  conn = h2o.getConnection(),
  h2oRestApiVersion,
  urlSuffix,
  parms,
  ...
)
```

Arguments

`conn`	H2OConnection
`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, the version prefix is skipped.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`...`	(Optional) Additional parameters.

Details

The return value is a list as follows: $url – Final calculated URL. $postBody – The body of the POST request from client to server. $curlError – TRUE if a socket-level error occurred. FALSE otherwise. $curlErrorMessage – If curlError is TRUE a message about the error. $httpStatusCode – The HTTP status code. Usually 200 if the request succeeded. $httpStatusMessage – A string describing the httpStatusCode. $payload – The raw response payload as a character vector.

Value

A list object as described above

`.h2o.doRawPOST` *Perform a low-level HTTP POST operation on an H2O instance*

Description

Does not do any I/O level error checking. Caller must do its own validations. Does not modify the response payload in any way. Log the request and response if h2o.startLogging() has been called.

Usage

```
.h2o.doRawPOST(
  conn = h2o.getConnection(),
  h2oRestApiVersion,
  urlSuffix,
  parms,
  fileUploadInfo,
  ...
)
```

Arguments

`conn`	H2OConnection
`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, the version prefix is skipped.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`fileUploadInfo`	(Optional) Information to POST (NOTE: changes Content-type from XXX-www-url-encoded to multi-part). Use fileUpload(normalizePath("/path/to/file")).
`...`	(Optional) Additional parameters.

Details

The return value is a list as follows: $url – Final calculated URL. $postBody – The body of the POST request from client to server. $curlError – TRUE if a socket-level error occurred. FALSE otherwise. $curlErrorMessage – If curlError is TRUE a message about the error. $httpStatusCode – The HTTP status code. Usually 200 if the request succeeded. $httpStatusMessage – A string describing the httpStatusCode. $payload – The raw response payload as a character vector.

Value

A list object as described above

`.h2o.doSafeGET` *Perform a safe (i.e. error-checked) HTTP GET request to an H2O cluster.*

Description

This function validates that no CURL error occurred and that the HTTP response code is successful. If a failure occurred, then stop() is called with an error message. Since all necessary error checking is done inside this call, the valid payload is directly returned if the function successfully finishes without calling stop().

Usage

```
.h2o.doSafeGET(h2oRestApiVersion, urlSuffix, parms, ...)
```

Arguments

`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, a default version is chosen for you.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`...`	(Optional) Additional parameters.

Value

The raw response payload as a character vector

`.h2o.doSafePOST` *Perform a safe (i.e. error-checked) HTTP POST request to an H2O cluster.*

Description

This function validates that no CURL error occurred and that the HTTP response code is successful. If a failure occurred, then stop() is called with an error message. Since all necessary error checking is done inside this call, the valid payload is directly returned if the function successfully finishes without calling stop().

Usage

```
.h2o.doSafePOST(h2oRestApiVersion, urlSuffix, parms, fileUploadInfo, ...)
```

Arguments

`h2oRestApiVersion`	(Optional) A version number to prefix to the urlSuffix. If no version is provided, a default version is chosen for you.
`urlSuffix`	The partial URL suffix to add to the calculated base URL for the instance
`parms`	(Optional) Parameters to include in the request
`fileUploadInfo`	(Optional) Information to POST (NOTE: changes Content-type from XXX-www-url-encoded to multi-part). Use fileUpload(normalizePath("/path/to/file")).
`...`	(Optional) Additional parameters.

Value

The raw response payload as a character vector

`.h2o.is_progress` *Check if Progress Bar is Enabled*

Description

Check if Progress Bar is Enabled

Usage

```
.h2o.is_progress()
```

`.h2o.locate`	*Locate a file given the pattern <bucket>/<path/to/file> e.g. h2o:::.h2o.locate("smalldata/iris/iris22.csv") returns the absolute path to iris22.csv*

Description

Locate a file given the pattern <bucket>/<path/to/file> e.g. h2o:::.h2o.locate("smalldata/iris/iris22.csv") returns the absolute path to iris22.csv

Usage

```
.h2o.locate(pathStub, root.parent = NULL)
```

Arguments

`pathStub`	relative path
`root.parent`	search root directory

`.h2o.perfect_auc`	*Internal function that calculates a precise AUC from given probabilities and actual responses.*

Description

Note: The underlying implementation is not distributed and can only handle limited size of data. For internal use only.

Usage

```
.h2o.perfect_auc(probs, acts)
```

Arguments

`probs`	An H2OFrame holding vector of probabilities.
`acts`	An H2OFrame holding vector of actuals.

`.h2o.primitives`	*Map of operations known to H2O*

Description

Map of operations known to H2O

Usage

```
.h2o.primitives
```

Format

An object of class `character` of length 39.

`.h2o.__ALL_CAPABILITIES` *Capabilities endpoints*

Description

Capabilities endpoints

Usage

```
.h2o.__ALL_CAPABILITIES
```

Format

An object of class `character` of length 1.

`.h2o.__checkConnectionHealth` *Check H2O Server Health*

Description

Warn if there are sick nodes.

Usage

```
.h2o.__checkConnectionHealth()
```

`.h2o.__CREATE_FRAME` *H2OFrame Manipulation*

Description

H2OFrame Manipulation

Usage

```
.h2o.__CREATE_FRAME
```

Format

An object of class `character` of length 1.

`.h2o.__DECRYPTION_SETUP` *Decryption Endpoints*

Description

Decryption Endpoints

Usage

```
.h2o.__DECRYPTION_SETUP
```

Format

An object of class `character` of length 1.

`.h2o.__DKV` *Removal Endpoints*

Description

Removal Endpoints

Usage

```
.h2o.__DKV
```

Format

An object of class `character` of length 1.

`.h2o.__EXPORT_FILES` *Export Files Endpoint Generator*

Description

Export Files Endpoint Generator

Usage

```
.h2o.__EXPORT_FILES(frame)
```

Arguments

`frame`	H2OFrame

`.h2o.__FRAMES` *Inspect/Summary Endpoints*

Description

Inspect/Summary Endpoints

Usage

```
.h2o.__FRAMES
```

Format

An object of class `character` of length 1.

`.h2o.__IMPORT` *Import/Export Endpoints*

Description

Import/Export Endpoints

Usage

```
.h2o.__IMPORT
```

Format

An object of class `character` of length 1.

`.h2o.__JOBS` *Administrative Endpoints*

Description

Administrative Endpoints

Usage

```
.h2o.__JOBS
```

Format

An object of class `character` of length 1.

`.h2o.__LOGANDECHO` *Log and Echo Endpoint*

Description

Log and Echo Endpoint

Usage

```
.h2o.__LOGANDECHO
```

Format

An object of class `character` of length 1.

`.h2o.__MODELS` *Model Endpoint*

Description

Model Endpoint

Usage

```
.h2o.__MODELS
```

Format

An object of class `character` of length 1.

`.h2o.__MODEL_BUILDERS` *Model Builder Endpoint Generator*

Description

Model Builder Endpoint Generator

Usage

```
.h2o.__MODEL_BUILDERS(algo)
```

Arguments

`algo`	Cannonical identifier of H2O algorithm.

`.h2o.__MODEL_METRICS` *Model Metrics Endpoint*

Description

Model Metrics Endpoint

Usage

```
.h2o.__MODEL_METRICS(model, data)
```

Arguments

`model`	H2OModel.
`data`	H2OFrame.

`.h2o.__PARSE_SETUP` *Parse Endpoints*

Description

Parse Endpoints

Usage

```
.h2o.__PARSE_SETUP
```

Format

An object of class `character` of length 1.

`.h2o.__RAPIDS` *Rapids Endpoint*

Description

Rapids Endpoint

Usage

```
.h2o.__RAPIDS
```

Format

An object of class `character` of length 1.

`.h2o.__REST_API_VERSION` *H2O Package Constants*

Description

The API endpoints for interacting with H2O via REST are named here.

Usage

```
.h2o.__REST_API_VERSION
```

Format

An object of class `integer` of length 1.

Details

Additionally, environment variables for the H2O package are named here. Endpoint Version

`.h2o.__SEGMENT_MODELS_BUILDERS` *Segment Models Builder Endpoint Generator*

Description

Segment Models Builder Endpoint Generator

Usage

```
.h2o.__SEGMENT_MODELS_BUILDERS(algo)
```

Arguments

`algo`	Cannonical identifier of H2O algorithm.

`.h2o.__W2V_SYNONYMS` *Word2Vec Endpoints*

Description

Word2Vec Endpoints

Usage

```
.h2o.__W2V_SYNONYMS
```

Format

An object of class `character` of length 1.

`.has_model_coefficients` *Has the* `model` *coefficients?*

Description

Has the `model` coefficients?

Usage

```
.has_model_coefficients(model)
```

Arguments

`model`	Either a linear model with coefficients => TRUE, or something else => FALSE

Value

boolean

`.has_varimp` *Has the model variable importance?*

Description

Has the model variable importance?

Usage

```
.has_varimp(model)
```

Arguments

`model`	model or a string containing model id

Value

boolean

`.interpretable` *Is the model considered to be interpretable, i.e., simple enough.*

Description

Is the model considered to be interpretable, i.e., simple enough.

Usage

```
.interpretable(model)
```

Arguments

`model`	model or a string containing model id

Value

boolean

`.is_h2o_model` *Is the `model` an H2O model?*

Description

Is the `model` an H2O model?

Usage

```
.is_h2o_model(model)
```

Arguments

`model`	Either H2O model/model id => TRUE, or something else => FALSE

Value

boolean

.is_h2o_tree_model *Is the* `model` *a Tree-based H2O Model?*

Description

Is the `model` a Tree-based H2O Model?

Usage

```
.is_h2o_tree_model(model)
```

Arguments

`model`	Either tree-based H2O model/model id => TRUE, or something else => FALSE

Value

boolean

.is_plotting_to_rnotebook *Check if we are plotting in to r notebook.*

Description

Check if we are plotting in to r notebook.

Usage

```
.is_plotting_to_rnotebook()
```

Value

boolean

.leaderboard_for_row *Enhance leaderboard with per-model predictions.*

Description

Enhance leaderboard with per-model predictions.

Usage

```
.leaderboard_for_row(models_info, newdata, row_index, top_n = 20)
```

Arguments

`models_info`	models_info object
`newdata`	H2OFrame
`row_index`	index of the inspected row
`top_n`	leaderboard will contain top_n models

Value

H2OFrame

`.min_max` *Min-max normalization.*

Description

Min-max normalization.

Usage

```
.min_max(col)
```

Arguments

`col`	numeric vector

Value

normalized numeric vector

`.model_ids` *Get Model Ids*

Description

When provided with list of models it will extract model ids. When provided with model ids it won't change anything. Works for mixed list as well.

Usage

```
.model_ids(models)
```

Arguments

`models`	list or vector of models/model_ids

Value

a vector of `model_ids`

`.pkg.env` *The H2O Package Environment*

Description

The H2O Package Environment

Usage

```
.pkg.env
```

Format

An object of class `environment` of length 4.

`.plot_varimp` *Plot variable importances with ggplot2*

Description

Plot variable importances with ggplot2

Usage

```
.plot_varimp(model, top_n = 10)
```

Arguments

`model`	H2OModel
`top_n`	Plot just top_n features

Value

list of variable importance, groupped variable importance, and variable importance plot

.process_models_or_automl

Do basic validation and transform `object` *to a "standardized" list containing models, and their properties such as* `x`, `y`, *whether it is a (multinomial) clasification or not etc.*

Description

Do basic validation and transform `object` to a "standardized" list containing models, and their properties such as `x`, `y`, whether it is a (multinomial) clasification or not etc.

Usage

```
.process_models_or_automl(
  object,
  newdata,
  require_single_model = FALSE,
  require_multiple_models = FALSE,
  top_n_from_AutoML = NA,
  only_with_varimp = FALSE,
  best_of_family = FALSE,
  require_newdata = TRUE,
  check_x_y_consistency = TRUE
)
```

Arguments

`object`	Can be a single model/model_id, vector of model_id, list of models, H2OAutoML object
`newdata`	An H2OFrame with the same format as training frame
`require_single_model`	If true, make sure we were provided only one model
`require_multiple_models`	If true, make sure we were provided at least two models
`top_n_from_AutoML`	If set, don't return more than top_n models (applies only for AutoML object)
`only_with_varimp`	If TRUE, return only models that have variable importance
`best_of_family`	If TRUE, return only the best of family models; if FALSE return all models in `object`
`require_newdata`	If TRUE, require newdata to be specified; otherwise allow NULL instead, this can be used when there is no need to know if the problem is (multinomial) classification.
`check_x_y_consistency`	If TRUE, make sure that when given a list of models all models have the same X and y. Defaults to TRUE.

Value

a list with the following names `leader`, `is_automl`, `models`, `is_classification`, `is_multinomial_classification`, `x`, `y`, `model`

`.shorten_model_ids` *Shortens model ids if possible (iff there will be same amount of unique model_ids as before)*

Description

Shortens model ids if possible (iff there will be same amount of unique model_ids as before)

Usage

```
.shorten_model_ids(model_ids)
```

Arguments

`model_ids`	character vector

Value

character vector

`.skip_if_not_developer` *H2O <-> R Communication and Utility Methods*

Description

Collected here are the various methods used by the h2o-R package to communicate with the H2O backend. There are methods for checking cluster health, polling, and inspecting objects in the H2O store.

Usage

```
.skip_if_not_developer()
```

.uniformize — *Convert to quantiles when provided with numeric vector. When col is a factor vector assign uniformly value between 0 and 1 to each level.*

Description

Convert to quantiles when provided with numeric vector. When col is a factor vector assign uniformly value between 0 and 1 to each level.

Usage

```
.uniformize(col)
```

Arguments

`col`	vector

Value

vector with values between 0 and 1

.varimp — *Get variable importance in a standardized way.*

Description

Get variable importance in a standardized way.

Usage

```
.varimp(model)
```

Arguments

`model`	H2OModel

Value

A named vector

.verify_dataxy — *Used to verify data, x, y and turn into the appropriate things*

Description

Used to verify data, x, y and turn into the appropriate things

Usage

```
.verify_dataxy(data, x, y, autoencoder = FALSE)
```

Arguments

data	H2OFrame
x	features
y	response
autoencoder	autoencoder flag

aaa — *Starting H2O For examples*

Description

Starting H2O For examples

Examples

```
## Not run:
if (Sys.info()['sysname'] == "Darwin" && Sys.info()['release'] == '13.4.0') {
  quit(save = "no")
} else {
 h2o.init(nthreads = 2)
}

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
summary(apply(iris_hf, 2, sum))

## End(Not run)
```

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.

Examples

```
## Not run:
library(h2o)
h2o.init()
pretrained <- as.h2o(data.frame(
       C1 = c("a", "b"), C2 = c(0, 1), C3 = c(1, 0), C4 = c(0.2, 0.8),
       stringsAsFactors = FALSE))
pretrained_w2v <- h2o.word2vec(pre_trained = pretrained, vec_size = 3)
words <- as.character(as.h2o(c("b", "a", "c", NA, "a")))
vecs <- h2o.transform(pretrained_w2v, words = words)

## End(Not run)
```

`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.

Details

Method `as.data.frame.H2OFrame` will use `fread` if data.table package is installed in required version.

See Also

`use.package`

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
as.data.frame(prostate)

## End(Not run)
```

---

as.data.frame.H2OSegmentModels

Converts a collection of Segment Models to a data.frame

Description

Converts a collection of Segment Models to a data.frame

Usage

```
## S3 method for class 'H2OSegmentModels'
as.data.frame(x, ...)
```

Arguments

x	Object of class H2OSegmentModels.
...	Further arguments to be passed down from other methods.

Value

Returns data.frame with result of segment model training.

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
models <- h2o.train_segments(algorithm = "gbm",
                             segment_columns = "Species",
                             x = c(1:3), y = 4,
                             training_frame = iris_hf,
                             ntrees = 5,
                             max_depth = 4)
as.data.frame(models)

## End(Not run)
```

`as.factor` *Convert H2O Data to Factors*

Description

Convert column/columns in the current frame to categoricals.

Usage

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

Arguments

`x`	a column from an H2OFrame data set.

See Also

`as.factor`.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Single column
cars <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
df <- h2o.importFile(cars)
df["cylinders"] <- as.factor(df["cylinders"])
h2o.describe(df["cylinders"])

# Multiple columns
cars <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
df <- h2o.importFile(cars)
df[c("cylinders","economy_20mpg")] <- as.factor(df[c("cylinders","economy_20mpg")])
h2o.describe(df[c("cylinders","economy_20mpg")])

## End(Not run)
```

`as.h2o` *Create H2OFrame*

Description

Import R object to the H2O cluster.

Usage

```
as.h2o(x, destination_frame = "", ...)

## Default S3 method:
as.h2o(x, destination_frame = "", ...)

## S3 method for class 'H2OFrame'
as.h2o(x, destination_frame = "", ...)

## S3 method for class 'data.frame'
as.h2o(x, destination_frame = "", use_datatable = TRUE, ...)

## S3 method for class 'Matrix'
as.h2o(x, destination_frame = "", use_datatable = TRUE, ...)
```

Arguments

`x`	An R object.
`destination_frame`	A string with the desired name for the H2OFrame
`...`	arguments passed to method arguments.
`use_datatable`	allow usage of data.table

Details

Method `as.h2o.data.frame` will use `fwrite` if data.table package is installed in required version.

To speedup execution time for large sparse matrices, use h2o datatable. Make sure you have installed and imported data.table and slam packages. Turn on h2o datatable by options("h2o.use.data.table"=TRUE)

References

https://h2o.ai/blog/fast-csv-writing-for-r/

See Also

`use.package`

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
euro_hf <- as.h2o(euro)
letters_hf <- as.h2o(letters)
state_hf <- as.h2o(state.x77)
iris_hf_2 <- as.h2o(iris_hf)
stopifnot(is.h2o(iris_hf), dim(iris_hf) == dim(iris),
          is.h2o(euro_hf), dim(euro_hf) == c(length(euro), 1L),
          is.h2o(letters_hf), dim(letters_hf) == c(length(letters), 1L),
          is.h2o(state_hf), dim(state_hf) == dim(state.x77),
          is.h2o(iris_hf_2), dim(iris_hf_2) == dim(iris_hf))
if (requireNamespace("Matrix", quietly=TRUE)) {
  data <- rep(0, 100)
  data[(1:10) ^ 2] <- 1:10 * pi
  m <- matrix(data, ncol = 20, byrow = TRUE)
  m <- Matrix::Matrix(m, sparse = TRUE)
  m_hf <- as.h2o(m)
  stopifnot(is.h2o(m_hf), dim(m_hf) == dim(m))
}

## End(Not run)
```

`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.

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
describe <- h2o.describe(iris_hf)
mins = as.matrix(apply(iris_hf, 2, min))
print(mins)

## End(Not run)
```

`as.numeric` *Convert H2O Data to Numeric*

Description

Converts an H2O column into a numeric value column. If the column type is enum and you want to convert it to numeric, you should first convert it to character then convert it to numeric. Otherwise, the values may be converted to underlying factor values, not the expected mapped values.

Usage

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

Arguments

x	a column from an H2OFrame data set.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate[, 2] <- as.factor (prostate[, 2])
prostate[, 2] <- as.numeric(prostate[, 2])

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
cor_R <- cor(as.matrix(iris[, 1]))
cor_h2o <- cor(iris_hf[, 1])
iris_R_cor <- cor(iris[, 1:4])
iris_H2O_cor <- as.data.frame(cor(iris_hf[, 1:4]))
h2o_vec <- as.vector(unlist(iris_H2O_cor))
r_vec <- as.vector(unlist(iris_R_cor))

## End(Not run)
```

`australia` *Australia Coastal Data*

Description

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

Format

A data frame with 251 rows and 8 columns

`case_insensitive_match_arg` *Works like match.arg but ignores case*

Description

Works like match.arg but ignores case

Usage

```
case_insensitive_match_arg(arg, choices)
```

Arguments

`arg`	argument to match that should be declared as a character vector containing possible values
`choices`	argument to choose from (OPTIONAL)

Value

matched arg

`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.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
colnames(iris_hf)  # Returns "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

## End(Not run)
```

---

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

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
dim(iris_hf)

## End(Not run)
```

`dimnames.H2OFrame` *Column names of an H2OFrame*

Description

Set column names of an H2O Frame

Usage

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

Arguments

x	An H2OFrame

Examples

```
## Not run:
library(h2o)
h2o.init()

n <- 2000
#  Generate variables V1, ... V10
X <- matrix(rnorm(10 * n), n, 10)
#  y = +1 if sum_i x_{ij}^2 > chisq median on 10 df
y <- rep(-1, n)
y[apply(X*X, 1, sum) > qchisq(.5, 10)] <- 1
#  Assign names to the columns of X:
dimnames(X)[[2]] <- c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10")

## End(Not run)
```

`feature_frequencies.H2OModel` *Retrieve the number of occurrences of each feature for given observations Available for GBM, Random Forest and Isolation Forest models.*

Description

Retrieve the number of occurrences of each feature for given observations Available for GBM, Random Forest and Isolation Forest models.

Usage

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

h2o.feature_frequencies(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.

Value

Returns an H2OFrame contain per-feature frequencies on the predict path for each input row.

See Also

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

`generate_col_ind` *CHeck to see if the column names/indices entered is valid for the dataframe given. This is an internal function*

Description

CHeck to see if the column names/indices entered is valid for the dataframe given. This is an internal function

Usage

```
generate_col_ind(data, by)
```

Arguments

`data`	The H2OFrame whose column names or indices are entered as a list
`by`	The column names/indices in a list.

`get_seed.H2OModel` *Get the seed from H2OModel which was used during training. If a user does not set the seed parameter before training, the seed is autogenerated. It returns seed as the string if the value is bigger than the integer. For example, an autogenerated seed is always long so that the seed in R is a string.*

Description

Get the seed from H2OModel which was used during training. If a user does not set the seed parameter before training, the seed is autogenerated. It returns seed as the string if the value is bigger than the integer. For example, an autogenerated seed is always long so that the seed in R is a string.

Usage

```
get_seed.H2OModel(object)

h2o.get_seed(object)
```

Arguments

`object`	a fitted H2OModel object.

Value

Returns seed to be used during training a model. Could be numeric or string.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
seed <- h2o.get_seed(prostate_gbm)

## End(Not run)
```

`h2o.abs` *Compute the absolute value of x*

Description

Compute the absolute value of x

Usage

```
h2o.abs(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`MathFun` for the base R implementation, `abs()`.

Examples

```
## Not run:
library(h2o)
h2o.init()
url <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/smtrees.csv"
smtrees_hf <- h2o.importFile(url)
smtrees_df <- read.csv(
  "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/smtrees.csv")
model <- h2o.gbm(x = c("girth", "height"), y = "vol", ntrees = 3, max_depth = 1,
                 distribution = "gaussian", min_rows = 2, learn_rate = .1,
                 training_frame = smtrees_hf)
pred <- as.data.frame(predict(model, newdata = smtrees_hf))
diff <- pred - smtrees_df[, 4]
```

```
diff_abs <- abs(diff)
print(diff_abs)

## End(Not run)
```

---

h2o.acos — *Compute the arc cosine of x*

Description

Compute the arc cosine of x

Usage

```
h2o.acos(x)
```

Arguments

x	An H2OFrame object.

See Also

Trig for the base R implementation, acos().

Examples

```
## Not run:
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.acos(prostate[, 2])

## End(Not run)
```

---

h2o.aecu — *Retrieve the default AECU (Average Excess Cumulative Uplift = area between AUUC and random AUUC)*

Description

Retrieves the AECU value from an H2OBinomialUpliftMetrics. You need to specificy the type of AECU using metric parameter. Defaults "qini". Qini AECU equals the Qini value. If "train" and "valid" parameters are FALSE (default), then the training AECU value is returned. If more than one parameter is set to TRUE, then a named vector of AECUs are returned, where the names are "train", "valid".

Usage

```
h2o.aecu(object, train = FALSE, valid = FALSE, metric = "qini")
```

Arguments

`object`	An H2OBinomialUpliftMetrics
`train`	Retrieve the training AECU
`valid`	Retrieve the validation AECU
`metric`	Specify metric of AECU. Posibilities are "qini", "lift", "gain", defaults "qini".

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.aecu(perf)

## End(Not run)
```

`h2o.aecu_table` *Retrieve the all types of AECU (average excess cumulative uplift) value in a table*

Description

Retrieves the all types of AECU value in a table from an H2OBinomialUpliftMetrics. If "train" and "valid" parameters are FALSE (default), then the training AECU values are returned. If more than one parameter is set to TRUE, then a named vector of AECU values are returned, where the names are "train", "valid".

Usage

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

Arguments

`object`	An H2OBinomialUpliftMetrics
`train`	Retrieve the training AECU values table
`valid`	Retrieve the validation AECU values table

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.aecu_table(perf)

## End(Not run)
```

`h2o.aggregated_frame` *Retrieve an aggregated frame from an Aggregator model*

Description

Retrieve an aggregated frame from the Aggregator model and use it to create a new frame.

Usage

```
h2o.aggregated_frame(model)
```

Arguments

`model`	an H2OClusteringModel corresponding from a `h2o.aggregator` call.

Examples

```
## Not run:
library(h2o)
h2o.init()
df <- h2o.createFrame(rows = 100,
                      cols = 5,
                      categorical_fraction = 0.6,
                      integer_fraction = 0,
                      binary_fraction = 0,
                      real_range = 100,
                      integer_range = 100,
                      missing_fraction = 0)
target_num_exemplars = 1000
rel_tol_num_exemplars = 0.5
encoding = "Eigen"
agg <- h2o.aggregator(training_frame = df,
                     target_num_exemplars = target_num_exemplars,
                     rel_tol_num_exemplars = rel_tol_num_exemplars,
                     categorical_encoding = encoding)
# Use the aggregated frame to create a new dataframe
new_df <- h2o.aggregated_frame(agg)
```

```
## End(Not run)
```

`h2o.aggregator` *Build an Aggregated Frame*

Description

Builds an Aggregated Frame of an H2OFrame.

Usage

```
h2o.aggregator(
  training_frame,
  x,
  model_id = NULL,
  ignore_const_cols = TRUE,
  target_num_exemplars = 5000,
  rel_tol_num_exemplars = 0.5,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  save_mapping_frame = FALSE,
  num_iteration_without_new_exemplar = 500,
  export_checkpoints_dir = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`model_id`	Destination id for this model; auto-generated if not specified.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`target_num_exemplars`	Targeted number of exemplars Defaults to 5000.
`rel_tol_num_exemplars`	Relative tolerance for number of exemplars (e.g, 0.5 is +/- 50 percents) Defaults to 0.5.
`transform`	Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NORMALIZE.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`save_mapping_frame`	`Logical`. Whether to export the mapping of the aggregated frame Defaults to FALSE.

`num_iteration_without_new_exemplar`	The number of iterations to run before aggregator exits if the number of exemplars collected didn't change Defaults to 500.
`export_checkpoints_dir`	Automatically export generated models to this directory.

Examples

```
## Not run:
library(h2o)
h2o.init()
df <- h2o.createFrame(rows = 100,
                      cols = 5,
                      categorical_fraction = 0.6,
                      integer_fraction = 0,
                      binary_fraction = 0,
                      real_range = 100,
                      integer_range = 100,
                      missing_fraction = 0)
target_num_exemplars = 1000
rel_tol_num_exemplars = 0.5
encoding = "Eigen"
agg <- h2o.aggregator(training_frame = df,
                     target_num_exemplars = target_num_exemplars,
                     rel_tol_num_exemplars = rel_tol_num_exemplars,
                     categorical_encoding = encoding)

## End(Not run)
```

h2o.aic *Retrieve the Akaike information criterion (AIC) value*

Description

Retrieves the AIC value. 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

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
p_sid <- h2o.runif(prostate)
prostate_train <- prostate[p_sid > .2,]
prostate_glm <- h2o.glm(x = 3:7, y = 2, training_frame = prostate_train)
aic_basic <- h2o.aic(prostate_glm)
print(aic_basic)

## End(Not run)
```

`h2o.all` *Given a set of logical vectors, are all of the values true?*

Description

Given a set of logical vectors, are all of the values true?

Usage

```
h2o.all(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`all` for the base R implementation.

`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

```
## Not run:
library(h2o)
h2o.init()
prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
prostate_dl = h2o.deeplearning(x = 3:9, training_frame = prostate, autoencoder = TRUE,
                               hidden = c(10, 10), epochs = 5, seed = 1)
prostate_anon = h2o.anomaly(prostate_dl, prostate)
head(prostate_anon)
prostate_anon_per_feature = h2o.anomaly(prostate_dl, prostate, per_feature = TRUE)
head(prostate_anon_per_feature)

## End(Not run)
```

`h2o.anovaglm` *H2O ANOVAGLM is used to calculate Type III SS which is used to evaluate the contributions of individual predictors and their interactions to a model. Predictors or interactions with negligible contributions to the model will have high p-values while those with more contributions will have low p-values.*

Description

H2O ANOVAGLM is used to calculate Type III SS which is used to evaluate the contributions of individual predictors and their interactions to a model. Predictors or interactions with negligible contributions to the model will have high p-values while those with more contributions will have low p-values.

Usage

```
h2o.anovaglm(
  x,
  y,
  training_frame,
  model_id = NULL,
  seed = -1,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  offset_column = NULL,
  weights_column = NULL,
 family = c("AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial",
   "poisson", "gamma", "tweedie", "negativebinomial"),
  tweedie_variance_power = 0,
  tweedie_link_power = 1,
  theta = 0,
  solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
    "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
  missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
  plug_values = NULL,
  compute_p_values = TRUE,
  standardize = TRUE,
  non_negative = FALSE,
  max_iterations = 0,
  link = c("family_default", "identity", "logit", "log", "inverse", "tweedie",
    "ologit"),
  prior = 0,
  alpha = NULL,
  lambda = c(0),
  lambda_search = FALSE,
  stopping_rounds = 0,
 stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
   "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  early_stopping = FALSE,
  stopping_tolerance = 0.001,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = 0,
  save_transformed_framekeys = FALSE,
  highest_interaction_term = 0,
  nparallelism = 4,
  type = 0
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.

`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`family`	Family. Use binomial for classification with logistic regression, others are for regression problems. Must be one of: "AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial", "poisson", "gamma", "tweedie", "negativebinomial". Defaults to AUTO.
`tweedie_variance_power`	Tweedie variance power Defaults to 0.
`tweedie_link_power`	Tweedie link power Defaults to 1.
`theta`	Theta Defaults to 0.
`solver`	AUTO will set the solver based on given data and the other parameters. IRLSM is fast on on problems with small number of predictors and for lambda-search with L1 penalty, L_BFGS scales better for datasets with many columns. Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR Defaults to IRLSM.
`missing_values_handling`	Handling of missing values. Either MeanImputation, Skip or PlugValues. Must be one of: "MeanImputation", "Skip", "PlugValues". Defaults to MeanImputation.
`plug_values`	Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues)
`compute_p_values`	`Logical`. Request p-values computation, p-values work only with IRLSM solver and no regularization Defaults to TRUE.
`standardize`	`Logical`. Standardize numeric columns to have zero mean and unit variance Defaults to TRUE.
`non_negative`	`Logical`. Restrict coefficients (not intercept) to be non-negative Defaults to FALSE.
`max_iterations`	Maximum number of iterations Defaults to 0.
`link`	Link function. Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit". Defaults to family_default.
`prior`	Prior probability for y==1. To be used only for logistic regression iff the data has been sampled and the mean of response does not reflect reality. Defaults to 0.
`alpha`	Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.

`lambda`	Regularization strength Defaults to c(0.0).
`lambda_search`	`Logical`. Use lambda search starting at lambda max, given lambda is then interpreted as lambda min Defaults to FALSE.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`early_stopping`	`Logical`. Stop early when there is no more relative improvement on train or validation (if provided). Defaults to FALSE.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
`balance_classes`	`Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
`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). Requires balance_classes. Defaults to 5.0.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`save_transformed_framekeys`	`Logical`. true to save the keys of transformed predictors and interaction column. Defaults to FALSE.
`highest_interaction_term`	Limit the number of interaction terms, if 2 means interaction between 2 columns only, 3 for three columns and so on... Default to 2. Defaults to 0.
`nparallelism`	Number of models to build in parallel. Default to 4. Adjust according to your system. Defaults to 4.
`type`	Refer to the SS type 1, 2, 3, or 4. We are currently only supporting 3 Defaults to 0.

Examples

```
## Not run:
h2o.init()

# Run ANOVA GLM of VOL ~ CAPSULE + RACE
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
```

```
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
model <- h2o.anovaglm(y = "VOL", x = c("CAPSULE","RACE"), training_frame = prostate)


## End(Not run)
```

h2o.any — *Given a set of logical vectors, is at least one of the values true?*

Description

Given a set of logical vectors, is at least one of the values true?

Usage

```
h2o.any(x)
```

Arguments

x	An H2OFrame object.

See Also

`all` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.any(iris[, 1] < 1000)

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
h2o.anyFactor(iris_hf)

## End(Not run)
```

`h2o.api` *Perform a REST API request to a previously connected server.*

Description

This function is mostly for internal purposes, but may occasionally be useful for direct access to the backend H2O server. It has same parameters as :meth:H2OConnection.request <h2o.backend.H2OConnection.request>.

Usage

```
h2o.api(endpoint, params = NULL, json = NULL)
```

Arguments

`endpoint`	A H2O REST API endpoint.
`params`	A list of params passed in the url.
`json`	A list of params passed as a json payload.

Details

REST API endpoints can be obtained using:

```
endpoints <- sapply(h2o.api("GET /3/Metadata/endpoints")$routes, function(r) paste(r$http_method,
```

For a given route, the supported params can be otained using:

```
parameters <- sapply(h2o.api("GET /3/Metadata/schemas/{route$input_schema}")$schemas[[1]]$fields
```

Value

The parsed response.

Examples

```
## Not run:
res <- h2o.api("GET /3/NetworkTest")
res$table

## End(Not run)
```

h2o.arrange *Sorts an H2O frame by columns*

Description

Sorts H2OFrame by the columns specified. H2OFrame can contain String columns but should not sort on any String columns. Otherwise, an error will be thrown. To sort column c1 in descending order, do desc(c1). Returns a new H2OFrame, like dplyr::arrange.

Usage

```
h2o.arrange(x, ...)
```

Arguments

x	The H2OFrame input to be sorted.
...	The column names to sort by.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.arrange(iris, "species","petal_len","petal_wid")

## End(Not run)
```

h2o.ascharacter *Convert H2O Data to Characters*

Description

Convert H2O Data to Characters

Usage

```
h2o.ascharacter(x)
```

Arguments

x	An H2OFrame object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.ascharacter(iris["species"])

## End(Not run)
```

`h2o.asfactor` *Convert H2O Data to Factors*

Description

Convert H2O Data to Factors

Usage

```
h2o.asfactor(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`factor` for the base R implementation, `as.factor()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.asfactor(cars["cylinders"])

## End(Not run)
```

h2o.asnumeric *Convert H2O Data to Numerics*

Description

If the column type is enum and you want to convert it to numeric, you should first convert it to character then convert it to numeric. Otherwise, the values may be converted to underlying factor values, not the expected mapped values.

Usage

```
h2o.asnumeric(x)
```

Arguments

x	An H2OFrame object.

See Also

numeric for the base R implementation, as.numeric().

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.ascharacter(cars)
h2o.asnumeric(cars)

## End(Not run)
```

h2o.assign *Rename an H2O object.*

Description

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

Usage

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

Arguments

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
dim(cars)
split <- h2o.splitFrame(data = cars, ratios = 0.8)
train <- h2o.assign(split[[1]], key = "train")
test <- h2o.assign(split[[2]], key = "test")
dim(train)
dim(test)

## End(Not run)
```

---

h2o.as_date *Convert between character representations and objects of Date class*

Description

Functions to convert between character representations and objects of class "Date" representing calendar dates.

Usage

```
h2o.as_date(x, format, ...)
```

Arguments

x	H2OFrame column of strings or factors to be converted
format	A character string indicating date pattern
...	Further arguments to be passed from or to other methods.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.as_date(hdf["ds5"], "%d.%m.%y %H:%M")

## End(Not run)
```

`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 or H2OMultinomialMetrics 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

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

`h2o.aucpr` *Retrieve the AUCPR (Area Under Precision Recall Curve)*

Description

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

Usage

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

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

Arguments

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

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

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

`h2o.automl` *Automatic Machine Learning*

Description

The Automatic Machine Learning (AutoML) function automates the supervised machine learning model training process. AutoML finds the best model, given a training frame and response, and returns an H2OAutoML object, which contains a leaderboard of all the models that were trained in the process, ranked by a default model performance metric.

Usage

```
h2o.automl(
  x,
  y,
  training_frame,
  validation_frame = NULL,
  leaderboard_frame = NULL,
  blending_frame = NULL,
  nfolds = -1,
  fold_column = NULL,
  weights_column = NULL,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = NULL,
  max_runtime_secs_per_model = NULL,
  max_models = NULL,
  distribution = c("AUTO", "bernoulli", "ordinal", "multinomial", "gaussian",
    "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom"),
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
    "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error"),
  stopping_tolerance = NULL,
  stopping_rounds = 3,
  seed = NULL,
  project_name = NULL,
  exclude_algos = NULL,
  include_algos = NULL,
  modeling_plan = NULL,
  preprocessing = NULL,
  exploitation_ratio = -1,
  monotone_constraints = NULL,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_models = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  sort_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC",
    "AUCPR", "mean_per_class_error"),
  export_checkpoints_dir = NULL,
  verbosity = "warn",
  ...
)
```

Arguments

`x`	A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or index of the response variable in the model. For classification, the y column must be a factor, otherwise regression will be performed. Indexes are 1-based in R.
`training_frame`	Training frame (H2OFrame or ID).
`validation_frame`	Validation frame (H2OFrame or ID); Optional. This argument is ignored unless the user sets nfolds = 0. If cross-validation is turned off, then a validation frame can be specified and used for early stopping of individual models and early stopping of the grid searches. By default and when nfolds > 1, cross-validation metrics will be used for early stopping and thus validation_frame will be ignored.
`leaderboard_frame`	Leaderboard frame (H2OFrame or ID); Optional. If provided, the Leaderboard will be scored using this data frame intead of using cross-validation metrics,

which is the default.

`blending_frame` Blending frame (H2OFrame or ID) used to train the the metalearning algorithm in Stacked Ensembles (instead of relying on cross-validated predicted values); Optional. When provided, it also is recommended to disable cross validation by setting `nfolds=0` and to provide a leaderboard frame for scoring purposes.

`nfolds` Number of folds for k-fold cross-validation. Must be >= 2; defaults to 5. Use 0 to disable cross-validation; this will also disable Stacked Ensemble (thus decreasing the overall model performance).

`fold_column` Column with cross-validation fold index assignment per observation; used to override the default, randomized, 5-fold cross-validation scheme for individual models in the AutoML run.

`weights_column` Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed.

`balance_classes`
: `Logical`. Specify whether to oversample the minority classes to balance the class distribution; only applicable to classification. If the oversampled size of the dataset exceeds the maximum size calculated during `max_after_balance_size` parameter, then the majority class will be undersampled to satisfy the size limit. Defaults to FALSE.

`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). Requires `balance_classes`. Defaults to 5.0.

`max_runtime_secs`
: This argument specifies the maximum time that the AutoML process will run for. If both `max_runtime_secs` and `max_models` are specified, then the AutoML run will stop as soon as it hits either of these limits. If neither `max_runtime_secs` nor `max_models` are specified by the user, then `max_runtime_secs` defaults to 3600 seconds (1 hour).

`max_runtime_secs_per_model`
: Maximum runtime in seconds dedicated to each individual model training process. Use 0 to disable. Defaults to 0. Note that models constrained by a time budget are not guaranteed reproducible.

`max_models` Maximum number of models to build in the AutoML process (does not include Stacked Ensembles). Defaults to NULL (no strict limit). Always set this parameter to ensure AutoML reproducibility: all models are then trained until convergence and none is constrained by a time budget.

`distribution` Distribution function used by algorithms that support it; other algorithms use their defaults. Possible values: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom", and for parameterized distributions list form is used to specify the parameter, e.g., `list(type = "tweedie", tweedie_power = 1.5)`. Defaults to "AUTO".

`stopping_metric`
: Metric to use for early stopping ("AUTO" is logloss for classification, deviance for regression). Must be one of "AUTO", "deviance", "logloss", "MSE", "RMSE",

"MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to "AUTO".

`stopping_tolerance`
: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much). This value defaults to 0.001 if the dataset is at least 1 million rows; otherwise it defaults to a bigger value determined by the size of the dataset and the non-NA-rate. In that case, the value is computed as 1/sqrt(nrows * non-NA-rate).

`stopping_rounds`
: Integer. Early stopping based on convergence of `stopping_metric`. Stop if simple moving average of length k of the `stopping_metric` does not improve for k (`stopping_rounds`) scoring events. Defaults to 3 and must be an non-zero integer. Use 0 to disable early stopping.

`seed`
: Integer. Set a seed for reproducibility. AutoML can only guarantee reproducibility if `max_models` or early stopping is used because `max_runtime_secs` is resource limited, meaning that if the resources are not the same between runs, AutoML may be able to train more models on one run vs another. In addition, H2O Deep Learning models are not reproducible by default for performance reasons, so if the user requires reproducibility, then `exclude_algos` must contain "DeepLearning".

`project_name`
: Character string to identify an AutoML project. Defaults to NULL, which means a project name will be auto-generated. More models can be trained and added to an existing AutoML project by specifying the same project name in multiple calls to the AutoML function (as long as the same training frame is used in subsequent runs).

`exclude_algos`
: Vector of character strings naming the algorithms to skip during the model-building phase. An example use is `exclude_algos = c("GLM", "DeepLearning", "DRF")`, and the full list of options is: "DRF" (Random Forest and Extremely-Randomized Trees), "GLM", "XGBoost", "GBM", "DeepLearning" and "StackedEnsemble". Defaults to NULL, which means that all appropriate H2O algorithms will be used, if the search stopping criteria allow. Optional.

`include_algos`
: Vector of character strings naming the algorithms to restrict to during the model-building phase. This can't be used in combination with `exclude_algos` param. Defaults to NULL, which means that all appropriate H2O algorithms will be used, if the search stopping criteria allow. Optional.

`modeling_plan`
: List. The list of modeling steps to be used by the AutoML engine (they may not all get executed, depending on other constraints). Optional (Expert usage only).

`preprocessing`
: List. The list of preprocessing steps to run. Only 'target_encoding' is currently supported.

`exploitation_ratio`
: The budget ratio (between 0 and 1) dedicated to the exploitation (vs exploration) phase. By default, this is set to AUTO (exploitation_ratio=-1) as this is still experimental; to activate it, it is recommended to try a ratio around 0.1. Note that the current exploitation phase only tries to fine-tune the best XGBoost and the best GBM found during exploration.

`monotone_constraints`
: List. A mapping representing monotonic constraints. Use +1 to enforce an increasing constraint and -1 to specify a decreasing constraint.

`keep_cross_validation_predictions`
: `Logical`. Whether to keep the predictions of the cross-validation predictions. This needs to be set to TRUE if running the same AutoML object for repeated

runs because CV predictions are required to build additional Stacked Ensemble models in AutoML. This option defaults to FALSE.

`keep_cross_validation_models`
: `Logical`. Whether to keep the cross-validated models. Keeping cross-validation models may consume significantly more memory in the H2O cluster. This option defaults to FALSE.

`keep_cross_validation_fold_assignment`
: `Logical`. Whether to keep fold assignments in the models. Deleting them will save memory in the H2O cluster. Defaults to FALSE.

`sort_metric`
: Metric to sort the leaderboard by. For binomial classification choose between "AUC", "AUCPR", "logloss", "mean_per_class_error", "RMSE", "MSE". For regression choose between "mean_residual_deviance", "RMSE", "MSE", "MAE", and "RMSLE". For multinomial classification choose between "mean_per_class_error", "logloss", "RMSE", "MSE". Default is "AUTO". If set to "AUTO", then "AUC" will be used for binomial classification, "mean_per_class_error" for multinomial classification, and "mean_residual_deviance" for regression.

`export_checkpoints_dir`
: (Optional) Path to a directory where every model will be stored in binary form.

`verbosity`
: Verbosity of the backend messages printed during training; Optional. Must be one of NULL (live log disabled), "debug", "info", "warn", "error". Defaults to "warn".

`...`
: Additional (experimental) arguments to be passed through; Optional.

Details

AutoML trains several models, cross-validated by default, by using the following available algorithms:

- XGBoost
- GBM (Gradient Boosting Machine)
- GLM (Generalized Linear Model)
- DRF (Distributed Random Forest)
- XRT (eXtremely Randomized Trees)
- DeepLearning (Fully Connected Deep Neural Network)

It also applies HPO on the following algorithms:

- XGBoost
- GBM
- DeepLearning

In some cases, there will not be enough time to complete all the algorithms, so some may be missing from the leaderboard.

Finally, AutoML also trains several Stacked Ensemble models at various stages during the run. Mainly two kinds of Stacked Ensemble models are trained:

- one of all available models at time t.
- one of only the best models of each kind at time t.

Note that Stacked Ensemble models are trained only if there isn't another stacked ensemble with the same base models.

Value

An H2OAutoML object.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path, header = TRUE)
y <- "CAPSULE"
prostate[,y] <- as.factor(prostate[,y])  #convert to factor for classification
aml <- h2o.automl(y = y, training_frame = prostate, max_runtime_secs = 30)
lb <- h2o.get_leaderboard(aml)
head(lb)

## End(Not run)
```

h2o.auuc *Retrieve AUUC*

Description

Retrieves the AUUC value from an H2OBinomialUpliftMetrics. If the metric parameter is "AUTO", the type of AUUC depends on auuc_type which was set before training. If you need specific AUUC, set metric parameter. If "train" and "valid" parameters are FALSE (default), then the training AUUC value is returned. If more than one parameter is set to TRUE, then a named vector of AUUCs are returned, where the names are "train", "valid".

Usage

```
h2o.auuc(object, train = FALSE, valid = FALSE, metric = NULL)
```

Arguments

`object`	An H2OBinomialUpliftMetrics
`train`	Retrieve the training AUUC
`valid`	Retrieve the validation AUUC
`metric`	Specify the AUUC metric to get specific AUUC. Possibilities are NULL, "qini", "lift", "gain".

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.auuc(perf)

## End(Not run)
```

`h2o.auuc_normalized` *Retrieve normalized AUUC*

Description

Retrieves the AUUC value from an H2OBinomialUpliftMetrics. If the metric parameter is "AUTO", the type of AUUC depends on auuc_type which was set before training. If you need specific normalized AUUC, set metric parameter. If "train" and "valid" parameters are FALSE (default), then the training normalized AUUC value is returned. If more than one parameter is set to TRUE, then a named vector of normalized AUUCs are returned, where the names are "train", "valid".

Usage

```
h2o.auuc_normalized(object, train = FALSE, valid = FALSE, metric = NULL)
```

Arguments

`object`	An H2OBinomialUpliftMetrics
`train`	Retrieve the training AUUC
`valid`	Retrieve the validation AUUC
`metric`	Specify the AUUC metric to get specific AUUC. Possibilities are NULL, "qini", "lift", "gain".

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.auuc_normalized(perf)

## End(Not run)
```

h2o.auuc_table *Retrieve the all types of AUUC in a table*

Description

Retrieves the all types of AUUC in a table from an H2OBinomialUpliftMetrics. If "train" and "valid" parameters are FALSE (default), then the training AUUC values are returned. If more than one parameter is set to TRUE, then a named vector of AUUCs are returned, where the names are "train", "valid".

Usage

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

Arguments

object	An H2OBinomialUpliftMetrics
train	Retrieve the training AUUC table
valid	Retrieve the validation AUUC table

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                   ntrees=10, max_depth=5, treatment_column="treatment",
                                       auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.auuc_table(perf)

## End(Not run)
```

h2o.average_objective *Extracts the final training average objective function of a GLM model.*

Description

Extracts the final training average objective function of a GLM model.

Usage

```
h2o.average_objective(model)
```

Arguments

`model`	an H2OModel object.

Value

The final training average objective of a GLM model.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "acceleration"
cars_model <- h2o.glm(y=response,
                       x=predictors,
                       training_frame = cars,
                       family="gaussian",
                       generate_scoring_history=TRUE)
objValue <- h2o.average_objective(cars_model)

## End(Not run)
```

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

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

Examples

```
## Not run:
library(h2o)
h2o.init()
fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.betweenss(km, train = TRUE)

## End(Not run)
```

---

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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/chicago/chicagoCensus.csv"
census <- h2o.importFile(f)
census[, 1] <- as.factor(census[, 1])

dl_model <- h2o.deeplearning(x = c(1:3), y = 4, training_frame = census,
                            hidden = c(17, 191),
                            epochs = 1,
                            balance_classes = FALSE,
                            export_weights_and_biases = TRUE)
h2o.biases(dl_model, vector_id = 1)

## End(Not run)
```

`h2o.bottomN` *H2O bottomN*

Description

bottomN function will will grab the bottom N percent of values of a column and return it in a H2OFrame. Extract the top N percent of values of a column and return it in a H2OFrame.

Usage

```
h2o.bottomN(x, column, nPercent)
```

Arguments

`x`	an H2OFrame
`column`	is a column name or column index to grab the top N percent value from
`nPercent`	is a bottom percentage value to grab

Value

An H2OFrame with 2 columns. The first column is the original row indices, second column contains the bottomN values

Examples

```
## Not run:
library(h2o)
h2o.init()

f1 <- "https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/jira/TopBottomNRep4.csv.zip"
f2 <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/Bottom20Per.csv.zip"
data_Frame <- h2o.importFile(f1)
bottom_Answer <- h2o.importFile(f2)
nPercent <- c(1, 2, 3, 4)
frame_Names <- names(data_Frame)
nP <- nPercent[sample(1:length(nPercent), 1, replace = FALSE)]
col_Index <- sample(1:length(frame_Names), 1, replace = FALSE)
h2o.bottomN(data_Frame, frame_Names[col_Index], nP)

## End(Not run)
```

`h2o.calculate_fairness_metrics` *Calculate intersectional fairness metrics.*

Description

Calculate intersectional fairness metrics.

Usage

```
h2o.calculate_fairness_metrics(
  model,
  frame,
  protected_columns,
  reference,
  favorable_class
)
```

Arguments

model	H2O Model
frame	Frame used to calculate the metrics.
protected_columns	List of categorical columns that contain sensitive information such as race, gender, age etc.
reference	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
favorable_class	Positive/favorable outcome class of the response.

Value

Dictionary of frames. One frame is the overview, other frames contain dependence of performance on threshold for each protected group.

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
      'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
       'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.calculate_fairness_metrics(gbm, test, protected_columns = protected_columns,
                               reference = reference, favorable_class = favorable_class)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_cbind <- h2o.cbind(prostate, prostate)
head(prostate_cbind)

## End(Not run)
```

`h2o.ceiling` *Take a single numeric argument and return a numeric vector with the smallest integers*

Description

ceiling takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of x.

Usage

```
h2o.ceiling(x)
```

Arguments

`x`	An H2OFrame object.

See Also

Round for the base R implementation, `ceiling()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.ceiling(iris[, 1])

## End(Not run)
```

---

h2o.centers *Retrieve the Model Centers*

Description

Retrieve the Model Centers

Usage

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

Arguments

object	An H2OClusteringModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()
fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
h2o.ceiling(fr[, 1])

## End(Not run)
```

---

h2o.centersSTD *Retrieve the Model Centers STD*

Description

Retrieve the Model Centers STD

Usage

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

Arguments

object	An H2OClusteringModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()
fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.centersSTD(km)

## End(Not run)
```

---

h2o.centroid_stats *Retrieve centroid statistics*

Description

Retrieve the centroid statistics. If "train" and "valid" 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" or "valid" For cross validation metrics this statistics are not available.

Usage

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

Arguments

object	An H2OClusteringModel object.
train	Retrieve the training centroid statistics
valid	Retrieve the validation centroid statistics

Examples

```
## Not run:
library(h2o)
h2o.init()
fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.centroid_stats(km, train = TRUE)

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
h2o.startLogging()
australia_path = system.file("extdata", "australia.csv", package = "h2o")
australia = h2o.importFile(path = australia_path)
h2o.stopLogging()
h2o.clearLog()

## End(Not run)
```

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

```
## Not run:
h2o.init()
h2o.clusterStatus()

## End(Not run)
```

h2o.cluster_sizes *Retrieve the cluster sizes*

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

Examples

```
## Not run:
library(h2o)
h2o.init()
fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.cluster_sizes(km, train = TRUE)

## End(Not run)
```

h2o.coef *Return the coefficients that can be applied to the non-standardized data.*

Description

Note: standardize = True by default. If set to False, then coef() returns the coefficients that are fit directly.

Usage

```
h2o.coef(object, predictorSize = -1)
```

Arguments

`object`	an H2OModel object.
`predictorSize`	predictor subset size. If specified, will only return model coefficients of that subset size. If not specified will return a lists of model coefficient dicts for all predictor subset size.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "cylinders"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_glm <- h2o.glm(balance_classes = TRUE,
                    seed = 1234,
                    x = predictors,
                    y = response,
                    training_frame = train,
                    validation_frame = valid)
h2o.coef(cars_glm)

## End(Not run)
```

h2o.coef_norm *Return coefficients fitted on the standardized data (requires standardize = True, which is on by default). These coefficients can be used to evaluate variable importance.*

Description

Return coefficients fitted on the standardized data (requires standardize = True, which is on by default). These coefficients can be used to evaluate variable importance.

Usage

```
h2o.coef_norm(object, predictorSize = -1)
```

Arguments

object	an H2OModel object.
predictorSize	predictor subset size. If specified, will only return model coefficients of that subset size. If not specified will return a lists of model coefficient dicts for all predictor subset size.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "cylinders"
```

```
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_glm <- h2o.glm(balance_classes = TRUE,
                    seed = 1234,
                    x = predictors,
                    y = response,
                    training_frame = train,
                    validation_frame = valid)
h2o.coef_norm(cars_glm)

## End(Not run)
```

`h2o.coef_with_p_values` *Return the coefficients table with coefficients, standardized coefficients, p-values, z-values and std-error for GLM models*

Description

Return the coefficients table with coefficients, standardized coefficients, p-values, z-values and std-error for GLM models

Usage

```
h2o.coef_with_p_values(object)
```

Arguments

`object`	An H2OModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "cylinders"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_glm <- h2o.glm(seed = 1234,
                    lambda=0.0,
                    compute_p_values=TRUE,
                    x = predictors,
                    y = response,
                    training_frame = train,
                    validation_frame = valid)
h2o.coef_with_p_values(cars_glm)

## End(Not run)
```

`h2o.colnames` *Return column names of an H2OFrame*

Description

Return column names of an H2OFrame

Usage

```
h2o.colnames(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`colnames` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.colnames(frame)

## End(Not run)
```

`h2o.columns_by_type` *Obtain a list of columns that are specified by ‘coltype‘*

Description

Obtain a list of columns that are specified by ‘coltype‘

Usage

```
h2o.columns_by_type(object, coltype = "numeric", ...)
```

Arguments

`object`	H2OFrame object
`coltype`	A character string indicating which column type to filter by. This must be one of the following: "numeric" - Numeric, but not categorical or time "categorical" - Integer, with a categorical/factor String mapping "string" - String column "time" - Long msec since the Unix Epoch - with a variety of display/parse options "uuid" - UUID "bad" - No none-NA rows (triple negative! all NAs or zero rows)
`...`	Ignored

Value

A list of column indices that correspond to "type"

Examples

```
## Not run:
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.columns_by_type(prostate, coltype = "numeric")

## End(Not run)
```

---

h2o.computeGram *Compute weighted gram matrix.*

Description

Compute weighted gram matrix.

Usage

```
h2o.computeGram(
  X,
  weights = "",
  use_all_factor_levels = FALSE,
  standardize = TRUE,
  skip_missing = FALSE
)
```

Arguments

X	an H2OModel corresponding to H2O framel.
weights	character corresponding to name of weight vector in frame.
use_all_factor_levels	logical flag telling h2o whether or not to skip first level of categorical variables during one-hot encoding.
standardize	logical flag telling h2o whether or not to standardize data
skip_missing	logical flag telling h2o whether skip rows with missing data or impute them with mean

`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, xval = 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.
`xval`	Retrieve the cross-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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
h2o.confusionMatrix(model, prostate)
# Generating a ModelMetrics object
perf <- h2o.performance(model, prostate)
h2o.confusionMatrix(perf)

## End(Not run)
```

---

h2o.connect *Connect to a running H2O instance.*

Description

Connect to a running H2O instance.

Usage

```
h2o.connect(
  ip = "localhost",
  port = 54321,
  strict_version_check = TRUE,
  proxy = NA_character_,
  https = FALSE,
  cacert = NA_character_,
  insecure = FALSE,
  username = NA_character_,
  password = NA_character_,
  use_spnego = FALSE,
  cookies = NA_character_,
  context_path = NA_character_,
  config = NULL
)
```

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.
`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.

cacert	Path to a CA bundle file with root and intermediate certificates of trusted CAs.
insecure	(Optional) Set this to TRUE to disable SSL certificate checking.
username	(Optional) Username to login with.
password	(Optional) Password to login with.
use_spnego	(Optional) Set this to TRUE to enable SPNEGO authentication.
cookies	(Optional) Vector(or list) of cookies to add to request.
context_path	(Optional) The last part of connection URL: http://<ip>:<port>/<context_path>
config	(Optional) A `list` describing connection parameters. Using `config` makes `h2o.connect` ignore other parameters and collect named list members instead (see examples).

Value

an instance of `H2OConnection` object representing a connection to the running H2O instance.

Examples

```
## Not run:
library(h2o)
# Try to connect to a H2O instance running at http://localhost:54321/cluster_X
#h2o.connect(ip = "localhost", port = 54321, context_path = "cluster_X")
# Or
#config = list(ip = "localhost", port = 54321, context_path = "cluster_X")
#h2o.connect(config = config)

# Skip strict version check during connecting to the instance
#h2o.connect(config = c(strict_version_check = FALSE, config))

## End(Not run)
```

`h2o.cor` *Correlation of columns.*

Description

Compute the correlation matrix of one or two H2OFrames.

Usage

```
h2o.cor(x, y = NULL, na.rm = FALSE, use, method = "Pearson")

cor(x, ...)
```

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: "everything" - outputs NaNs whenever one of its contributing observations is missing "all.obs" - presence of missing observations will throw an error "complete.obs" - discards missing values along with all observations in their rows so that only complete observations are used
method	str Method of correlation computation. Allowed values are: "Pearson" - Pearson's correlation coefficient "Spearman" - Spearman's correlation coefficient (Spearman's Rho) Defaults to "Pearson"
...	Further arguments to be passed down from other methods.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
cor(prostate$AGE)

## End(Not run)
```

---

h2o.cos *Compute the cosine of x*

Description

Compute the cosine of x

Usage

```
h2o.cos(x)
```

Arguments

x	An H2OFrame object.

See Also

Trig for the base R implementation, cos().

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cos(frame["C1"])

## End(Not run)
```

h2o.cosh — *Compute the hyperbolic cosine of x*

Description

Compute the hyperbolic cosine of x

Usage

```
h2o.cosh(x)
```

Arguments

x	An H2OFrame object.

See Also

Hyperbolic for the base R implementation, cosh().

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cosh(frame["C1"])

## End(Not run)
```

h2o.coxph — *Trains a Cox Proportional Hazards Model (CoxPH) on an H2O dataset*

Description

Trains a Cox Proportional Hazards Model (CoxPH) on an H2O dataset

Usage

```
h2o.coxph(
  x,
  event_column,
  training_frame,
  model_id = NULL,
  start_column = NULL,
  stop_column = NULL,
  weights_column = NULL,
  offset_column = NULL,
  stratify_by = NULL,
  ties = c("efron", "breslow"),
  init = 0,
  lre_min = 9,
  max_iterations = 20,
  interactions = NULL,
  interaction_pairs = NULL,
  interactions_only = NULL,
  use_all_factor_levels = FALSE,
  export_checkpoints_dir = NULL,
  single_node_mode = FALSE
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except event_column, start_column and stop_column are used.
`event_column`	The name of binary data column in the training frame indicating the occurrence of an event.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`start_column`	Start Time Column.
`stop_column`	Stop Time Column.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`stratify_by`	List of columns to use for stratification.
`ties`	Method for Handling Ties. Must be one of: "efron", "breslow". Defaults to efron.
`init`	Coefficient starting value. Defaults to 0.
`lre_min`	Minimum log-relative error. Defaults to 9.
`max_iterations`	Maximum number of iterations. Defaults to 20.
`interactions`	A list of predictor column indices to interact. All pairwise combinations will be computed for the list.
`interaction_pairs`	A list of pairwise (first order) column interactions.
`interactions_only`	A list of columns that should only be used to create interactions but should not itself participate in model training.

`use_all_factor_levels`
: `Logical`. (Internal. For development only!) Indicates whether to use all factor levels. Defaults to FALSE.

`export_checkpoints_dir`
: Automatically export generated models to this directory.

`single_node_mode`
: `Logical`. Run on a single node to reduce the effect of network overhead (for smaller datasets) Defaults to FALSE.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the heart dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

# Set the predictor and response
predictor <- "age"
response <- "event"

# Train a Cox Proportional Hazards model
heart_coxph <- h2o.coxph(x = predictor, training_frame = heart,
                         event_column = "event",
                         start_column = "start",
                         stop_column = "stop")

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
hf <- h2o.createFrame(rows = 1000, cols = 100, categorical_fraction = 0.1,
                      factors = 5, integer_fraction = 0.5, integer_range = 1,
                      has_response = TRUE)
head(hf)
summary(hf)

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

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                  nfolds = 5,  keep_cross_validation_fold_assignment = TRUE, seed = 1234)
h2o.cross_validation_fold_assignment(cars_gbm)

## End(Not run)
```

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                    nfolds = 5,  keep_cross_validation_predictions = TRUE, seed = 1234)
h2o.cross_validation_holdout_predictions(cars_gbm)

## End(Not run)
```

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                    nfolds = 5,  keep_cross_validation_models = TRUE, seed = 1234)
h2o.cross_validation_models(cars_gbm)

## End(Not run)
```

`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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                    nfolds = 5,  keep_cross_validation_predictions = TRUE, seed = 1234)
h2o.cross_validation_predictions(cars_gbm)

## End(Not run)
```

---

h2o.cummax   *Return the cumulative max over a column or across a row*

Description

Return the cumulative max over a column or across a row

Usage

```
h2o.cummax(x, axis = 0)
```

Arguments

x	An H2OFrame object.
axis	An int that indicates whether to do down a column (0) or across a row (1).

See Also

cumsum for the base R implementation, cummax().

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cummax(frame, 1)

## End(Not run)
```

h2o.cummin — *Return the cumulative min over a column or across a row*

Description

Return the cumulative min over a column or across a row

Usage

```
h2o.cummin(x, axis = 0)
```

Arguments

x	An H2OFrame object.
axis	An int that indicates whether to do down a column (0) or across a row (1).

See Also

cumsum for the base R implementation, cummin().

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cummin(frame, 1)

## End(Not run)
```

h2o.cumprod — *Return the cumulative product over a column or across a row*

Description

Return the cumulative product over a column or across a row

Usage

```
h2o.cumprod(x, axis = 0)
```

Arguments

x	An H2OFrame object.
axis	An int that indicates whether to do down a column (0) or across a row (1).

See Also

`cumsum` for the base R implementation, `cumprod()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cumprod(frame, 1)

## End(Not run)
```

---

h2o.cumsum *Return the cumulative sum over a column or across a row*

Description

Return the cumulative sum over a column or across a row

Usage

```
h2o.cumsum(x, axis = 0)
```

Arguments

x	An H2OFrame object.
axis	An int that indicates whether to do down a column (0) or across a row (1).

See Also

`cumsum` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.cumsum(frame, 1)

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
summary(iris_hf)

# Cut sepal length column into intervals determined by min/max/quantiles
sepal_len_cut <- cut(iris_hf$Sepal.Length, c(4.2, 4.8, 5.8, 6, 8))
head(sepal_len_cut)
summary(sepal_len_cut)

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.day(hdf["ds9"])

## End(Not run)
```

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.dayOfWeek(hdf["ds9"])

## End(Not run)
```

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

Value

Returns an H2OFrame object.

Examples

```
## Not run:
  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))

## End(Not run)
```

`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 based on plyr::ddply.

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

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import iris dataset to H2O
iris_hf <- as.h2o(iris)
# Add function taking mean of Sepal.Length column
fun <- function(df) { sum(df[, 1], na.rm = TRUE) / nrow(df) }
# Apply function to groups by flower specie
# uses h2o's ddply, since iris_hf is an H2OFrame object
res <- h2o.ddply(iris_hf, "Species", fun)
head(res)

## End(Not run)
```

`h2o.decryptionSetup` *Setup a Decryption Tool*

Description

If your source file is encrypted - setup a Decryption Tool and then provide the reference (result of this function) to the import functions.

Usage

```
h2o.decryptionSetup(
  keystore,
  keystore_type = "JCEKS",
  key_alias = NA_character_,
  password = NA_character_,
  decrypt_tool = "",
  decrypt_impl = "water.parser.GenericDecryptionTool",
  cipher_spec = NA_character_
)
```

Arguments

`keystore`	An H2OFrame object referencing a loaded Java Keystore (see example).
`keystore_type`	(Optional) Specification of Keystore type, defaults to JCEKS.
`key_alias`	Which key from the keystore to use for decryption.
`password`	Password to the keystore and the key.
`decrypt_tool`	(Optional) Name of the decryption tool.
`decrypt_impl`	(Optional) Java class name implementing the Decryption Tool.
`cipher_spec`	Specification of a cipher (eg.: AES/ECB/PKCS5Padding).

See Also

h2o.importFile, h2o.parseSetup

Examples

```
## Not run:
library(h2o)
h2o.init()
ks_path <- system.file("extdata", "keystore.jks", package = "h2o")
keystore <- h2o.importFile(path = ks_path, parse = FALSE) # don't parse, keep as a binary file
cipher <- "AES/ECB/PKCS5Padding"
pwd <- "Password123"
alias <- "secretKeyAlias"
dt <- h2o.decryptionSetup(keystore, key_alias = alias, password = pwd, cipher_spec = cipher)
data_path <- system.file("extdata", "prostate.csv.aes", package = "h2o")
data <- h2o.importFile(data_path, decrypt_tool = dt)
summary(data)

## End(Not run)
```

---

h2o.deepfeatures *Feature Generation via H2O Deep Learning*

Description

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

Usage

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

Arguments

`object`	An H2OModel object that represents the deep learning model to be used for feature extraction.
`data`	An H2OFrame object.
`layer`	Index (integer) 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

`h2o.deeplearning` for making H2O Deep Learning models.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
prostate_dl = h2o.deeplearning(x = 3:9, y = 2, training_frame = prostate,
                               hidden = c(100, 200), epochs = 5)
prostate_deepfeatures_layer1 = h2o.deepfeatures(prostate_dl, prostate, layer = 1)
prostate_deepfeatures_layer2 = h2o.deepfeatures(prostate_dl, prostate, layer = 2)
head(prostate_deepfeatures_layer1)
head(prostate_deepfeatures_layer2)


## End(Not run)
```

`h2o.deeplearning` *Build a Deep Neural Network model using CPUs*

Description

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

Usage

```
h2o.deeplearning(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  weights_column = NULL,
  offset_column = NULL,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  checkpoint = NULL,
  pretrained_autoencoder = NULL,
  overwrite_with_best_model = TRUE,
  use_all_factor_levels = TRUE,
  standardize = TRUE,
  activation = c("Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout",
    "Maxout", "MaxoutWithDropout"),
  hidden = c(200, 200),
  epochs = 10,
  train_samples_per_iteration = -2,
  target_ratio_comm_to_comp = 0.05,
  seed = -1,
  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 = NULL,
  l1 = 0,
  l2 = 0,
  max_w2 = 3.4028235e+38,
  initial_weight_distribution = c("UniformAdaptive", "Uniform", "Normal"),
  initial_weight_scale = 1,
  initial_weights = NULL,
  initial_biases = NULL,
 loss = c("Automatic", "CrossEntropy", "Quadratic", "Huber", "Absolute", "Quantile"),
 distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
   "tweedie", "laplace", "quantile", "huber"),
  quantile_alpha = 0.5,
  tweedie_power = 1.5,
  huber_alpha = 0.9,
  score_interval = 5,
  score_training_samples = 10000,
  score_validation_samples = 0,
  score_duty_cycle = 0.1,
  classification_stop = 0,
  regression_stop = 1e-06,
  stopping_rounds = 5,
 stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
  "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
   "custom", "custom_increasing"),
  stopping_tolerance = 0,
  max_runtime_secs = 0,
  score_validation_sampling = c("Uniform", "Stratified"),
  diagnostics = TRUE,
  fast_mode = TRUE,
  force_load_balance = TRUE,
  variable_importances = TRUE,
  replicate_training_data = TRUE,
  single_node_mode = FALSE,
  shuffle_training_data = FALSE,
  missing_values_handling = c("MeanImputation", "Skip"),
  quiet_mode = FALSE,
  autoencoder = FALSE,
  sparse = FALSE,
  col_major = FALSE,
  average_activation = 0,
  sparsity_beta = 0,
  max_categorical_features = 2147483647,
  reproducible = FALSE,
  export_weights_and_biases = FALSE,
  mini_batch_size = 1,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  elastic_averaging = FALSE,
  elastic_averaging_moving_rate = 0.9,
  elastic_averaging_regularization = 0.001,
  export_checkpoints_dir = NULL,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  verbose = FALSE
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.

`score_each_iteration`
: `Logical`. Whether to score during each iteration of model training. Defaults to FALSE.

`weights_column`
: Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.

`offset_column`
: Offset column. This will be added to the combination of columns before applying the link function.

`balance_classes`
: `Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.

`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). Requires balance_classes. Defaults to 5.0.

`checkpoint`
: Model checkpoint to resume training with.

`pretrained_autoencoder`
: Pretrained autoencoder model to initialize this model with.

`overwrite_with_best_model`
: `Logical`. If enabled, override the final model with the best model found during training. Defaults to TRUE.

`use_all_factor_levels`
: `Logical`. Use all factor levels of categorical variables. Otherwise, the first factor level is omitted (without loss of accuracy). Useful for variable importances and auto-enabled for autoencoder. Defaults to TRUE.

`standardize`
: `Logical`. If enabled, automatically standardize the data. If disabled, the user must provide properly scaled input data. Defaults to TRUE.

`activation`
: Activation function. Must be one of: "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", "MaxoutWithDropout". Defaults to Rectifier.

`hidden`
: Hidden layer sizes (e.g. [100, 100]). Defaults to c(200, 200).

`epochs`
: How many times the dataset should be iterated (streamed), can be fractional. Defaults to 10.

`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), -2: automatic. Defaults to -2.

`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). Defaults to 0.05.

`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Note: only reproducible when running single threaded. Defaults to -1 (time-based random number).
`adaptive_rate`	`Logical`. Adaptive learning rate. Defaults to TRUE.
`rho`	Adaptive learning rate time decay factor (similarity to prior updates). Defaults to 0.99.
`epsilon`	Adaptive learning rate smoothing factor (to avoid divisions by zero and allow progress). Defaults to 1e-08.
`rate`	Learning rate (higher => less stable, lower => slower convergence). Defaults to 0.005.
`rate_annealing`	Learning rate annealing: rate / (1 + rate_annealing * samples). Defaults to 1e-06.
`rate_decay`	Learning rate decay factor between layers (N-th layer: rate * rate_decay ^ (n - 1). Defaults to 1.
`momentum_start`	Initial momentum at the beginning of training (try 0.5). Defaults to 0.
`momentum_ramp`	Number of training samples for which momentum increases. Defaults to 1000000.
`momentum_stable`	Final momentum after the ramp is over (try 0.99). Defaults to 0.
`nesterov_accelerated_gradient`	`Logical`. Use Nesterov accelerated gradient (recommended). Defaults to TRUE.
`input_dropout_ratio`	Input layer dropout ratio (can improve generalization, try 0.1 or 0.2). Defaults to 0.
`hidden_dropout_ratios`	Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5.
`l1`	L1 regularization (can add stability and improve generalization, causes many weights to become 0). Defaults to 0.
`l2`	L2 regularization (can add stability and improve generalization, causes many weights to be small. Defaults to 0.
`max_w2`	Constraint for squared sum of incoming weights per unit (e.g. for Rectifier). Defaults to 3.4028235e+38.
`initial_weight_distribution`	Initial weight distribution. Must be one of: "UniformAdaptive", "Uniform", "Normal". Defaults to UniformAdaptive.
`initial_weight_scale`	Uniform: -value...value, Normal: stddev. Defaults to 1.
`initial_weights`	A list of H2OFrame ids to initialize the weight matrices of this model with.
`initial_biases`	A list of H2OFrame ids to initialize the bias vectors of this model with.
`loss`	Loss function. Must be one of: "Automatic", "CrossEntropy", "Quadratic", "Huber", "Absolute", "Quantile". Defaults to Automatic.
`distribution`	Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.
`quantile_alpha`	Desired quantile for Quantile regression, must be between 0 and 1. Defaults to 0.5.

`tweedie_power`	Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to 1.5.
`huber_alpha`	Desired quantile for Huber/M-regression (threshold between quadratic and linear loss, must be between 0 and 1). Defaults to 0.9.
`score_interval`	Shortest time interval (in seconds) between model scoring. Defaults to 5.
`score_training_samples`	Number of training set samples for scoring (0 for all). Defaults to 10000.
`score_validation_samples`	Number of validation set samples for scoring (0 for all). Defaults to 0.
`score_duty_cycle`	Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring). Defaults to 0.1.
`classification_stop`	Stopping criterion for classification error fraction on training data (-1 to disable). Defaults to 0.
`regression_stop`	Stopping criterion for regression error (MSE) on training data (-1 to disable). Defaults to 1e-06.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 5.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`score_validation_sampling`	Method used to sample validation dataset for scoring. Must be one of: "Uniform", "Stratified". Defaults to Uniform.
`diagnostics`	`Logical`. Enable diagnostics for hidden layers. Defaults to TRUE.
`fast_mode`	`Logical`. Enable fast mode (minor approximation in back-propagation). Defaults to TRUE.
`force_load_balance`	`Logical`. Force extra load balancing to increase training speed for small datasets (to keep all cores busy). Defaults to TRUE.
`variable_importances`	`Logical`. Compute variable importances for input features (Gedeon method) - can be slow for large networks. Defaults to TRUE.
`replicate_training_data`	`Logical`. Replicate the entire training dataset onto every node for faster training on small datasets. Defaults to TRUE.

`single_node_mode`	`Logical`. Run on a single node for fine-tuning of model parameters. Defaults to FALSE.
`shuffle_training_data`	`Logical`. Enable shuffling of training data (recommended if training data is replicated and train_samples_per_iteration is close to #nodes x #rows, of if using balance_classes). Defaults to FALSE.
`missing_values_handling`	Handling of missing values. Either MeanImputation or Skip. Must be one of: "MeanImputation", "Skip". Defaults to MeanImputation.
`quiet_mode`	`Logical`. Enable quiet mode for less output to standard output. Defaults to FALSE.
`autoencoder`	`Logical`. Auto-Encoder. Defaults to FALSE.
`sparse`	`Logical`. Sparse data handling (more efficient for data with lots of 0 values). Defaults to FALSE.
`col_major`	`Logical`. #DEPRECATED Use a column major weight matrix for input layer. Can speed up forward propagation, but might slow down backpropagation. Defaults to FALSE.
`average_activation`	Average activation for sparse auto-encoder. #Experimental Defaults to 0.
`sparsity_beta`	Sparsity regularization. #Experimental Defaults to 0.
`max_categorical_features`	Max. number of categorical features, enforced via hashing. #Experimental Defaults to 2147483647.
`reproducible`	`Logical`. Force reproducibility on small data (will be slow - only uses 1 thread). Defaults to FALSE.
`export_weights_and_biases`	`Logical`. Whether to export Neural Network weights and biases to H2O Frames. Defaults to FALSE.
`mini_batch_size`	Mini-batch size (smaller leads to better fit, larger can speed up and generalize better). Defaults to 1.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`elastic_averaging`	`Logical`. Elastic averaging between compute nodes can improve distributed model convergence. #Experimental Defaults to FALSE.
`elastic_averaging_moving_rate`	Elastic averaging moving rate (only if elastic averaging is enabled). Defaults to 0.9.
`elastic_averaging_regularization`	Elastic averaging regularization strength (only if elastic averaging is enabled). Defaults to 0.001.
`export_checkpoints_dir`	Automatically export generated models to this directory.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
`verbose`	`Logical`. Print scoring history to the console (Metrics per epoch). Defaults to FALSE.

See Also

predict.H2OModel for prediction

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris_hf, seed=123456)

# now make a prediction
predictions <- h2o.predict(iris_dl, iris_hf)

## End(Not run)
```

---

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path)
h2o.describe(prostate)

## End(Not run)
```

h2o.difflag1 — *Conduct a lag 1 transform on a numeric H2OFrame column*

Description

Conduct a lag 1 transform on a numeric H2OFrame column

Usage

```
h2o.difflag1(object)
```

Arguments

`object`	H2OFrame object

Value

Returns an H2OFrame object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "cylinders"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                    validation_frame = valid, nfolds = 5, seed = 1234)
h2o.difflag1(cars["cylinders"])

## End(Not run)
```

h2o.dim — *Returns the number of rows and columns for an H2OFrame object.*

Description

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

Usage

```
h2o.dim(x)
```

Arguments

`x`	An H2OFrame object.

See Also

dim for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.dim(cars)

## End(Not run)
```

---

h2o.dimnames *Column names of an H2OFrame*

Description

Column names of an H2OFrame

Usage

```
h2o.dimnames(x)
```

Arguments

x	An H2OFrame object.

See Also

dimnames for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.dimnames(cars)

## End(Not run)
```

h2o.disparate_analysis

Create a frame containing aggregations of intersectional fairness across the models.

Description

Create a frame containing aggregations of intersectional fairness across the models.

Usage

```
h2o.disparate_analysis(
  models,
  newdata,
  protected_columns,
  reference,
  favorable_class,
  air_metric = "selectedRatio",
  alpha = 0.05
)
```

Arguments

`models`	List of H2O Models
`newdata`	H2OFrame
`protected_columns`	List of categorical columns that contain sensitive information such as race, gender, age etc.
`reference`	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
`favorable_class`	Positive/favorable outcome class of the response.
`air_metric`	Metric used for Adverse Impact Ratio calculation. Defaults to "selectedRatio".
`alpha`	The alpha level is the probability of rejecting the null hypothesis that the protected group and the reference came from the same population when the null hypothesis is true.

Value

frame containing aggregations of intersectional fairness across the models

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
       'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
        'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
```

```
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

aml <- h2o.automl(x, y, training_frame = train, max_models = 3)

h2o.disparate_analysis(aml, test, protected_columns = protected_columns,
                       reference = reference, favorable_class = favorable_class)

## End(Not run)
```

h2o.distance — *Compute a pairwise distance measure between all rows of two numeric H2OFrames.*

Description

Compute a pairwise distance measure between all rows of two numeric H2OFrames.

Usage

```
h2o.distance(x, y, measure)
```

Arguments

x	An H2OFrame object (large, references).
y	An H2OFrame object (small, queries).
measure	An optional string indicating what distance measure to use. Must be one of: "l1" - Absolute distance (L1-norm, >=0) "l2" - Euclidean distance (L2-norm, >=0) "cosine" - Cosine similarity (-1...1) "cosine_sq" - Squared Cosine similarity (0...1)

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.distance(prostate[11:30, ], prostate[1:10, ], "cosine")

## End(Not run)
```

`h2o.downloadAllLogs` *Download H2O Log Files to Disk*

Description

`h2o.downloadAllLogs` downloads all H2O log files to local disk in .zip format. 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. Note that the saved format is .zip, so the file name must include the .zip extension.

Examples

```
## Not run:
h2o.downloadAllLogs(dirname='./your_directory_name/', filename = 'autoh2o_log.zip')

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)

file_path <- paste(getwd(), "my_iris_file.csv", sep = .Platform$file.sep)
h2o.downloadCSV(iris_hf, file_path)
file.info(file_path)
file.remove(file_path)

## End(Not run)
```

`h2o.download_model` *Download the model in binary format. The owner of the file saved is the user by which python session was executed.*

Description

Download the model in binary format. The owner of the file saved is the user by which python session was executed.

Usage

```
h2o.download_model(
  model,
  path = NULL,
  export_cross_validation_predictions = FALSE,
  filename = ""
)
```

Arguments

`model`	An H2OModel
`path`	The path where binary file should be downloaded. Downloaded to current directory by default.
`export_cross_validation_predictions`	A boolean flag indicating whether the download model should be saved with CV Holdout Frame predictions. Default is not to export the predictions.
`filename`	string indicating the file name.

Examples

```
## Not run:
library(h2o)
h <- h2o.init()
fr <- as.h2o(iris)
my_model <- h2o.gbm(x = 1:4, y = 5, training_frame = fr)
h2o.download_model(my_model)  # save to the current working directory

## End(Not run)
```

`h2o.download_mojo` *Download the model in MOJO format.*

Description

Download the model in MOJO format.

Usage

```
h2o.download_mojo(
  model,
  path = getwd(),
  get_genmodel_jar = FALSE,
  genmodel_name = "",
  genmodel_path = "",
  filename = ""
)
```

Arguments

`model`	An H2OModel
`path`	The path where MOJO file should be saved. Saved to current directory by default.
`get_genmodel_jar`	If TRUE, then also download h2o-genmodel.jar and store it in either in the same folder as the MOJO or in "genmodel_path" if specified.
`genmodel_name`	Custom name of genmodel jar.
`genmodel_path`	Path to store h2o-genmodel.jar. If left blank and "get_genmodel_jar" is TRUE, then the h2o-genmodel.jar is saved to "path".
`filename`	string indicating the file name. (Type of file is always .zip)

Value

Name of the MOJO file written to the path.

Examples

```
## Not run:
library(h2o)
h <- h2o.init()
fr <- as.h2o(iris)
my_model <- h2o.gbm(x = 1:4, y = 5, training_frame = fr)
h2o.download_mojo(my_model)  # save to the current working directory

## End(Not run)
```

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 = NULL,
  getjar = NULL,
  get_jar = TRUE,
  jar_name = ""
)
```

Arguments

model	An H2OModel
path	The path to the directory to store the POJO (no trailing slash). If NULL, then print to to console. The file name will be a compilable java file name.
getjar	(DEPRECATED) Whether to also download the h2o-genmodel.jar file needed to compile the POJO. This argument is now called 'get_jar'.
get_jar	Whether to also download the h2o-genmodel.jar file needed to compile the POJO
jar_name	Custom name of genmodel jar.

Value

If path is NULL, then pretty print the POJO to the console. Otherwise save it to the specified directory and return POJO file name.

Examples

```
## Not run:
library(h2o)
h <- h2o.init()
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(), get_jar = FALSE )  # save only the POJO to the current
#                                                            working directory, NOT RUN
h2o.download_pojo(my_model, getwd())  # save to the current working directory

## End(Not run)
```

h2o.drop_duplicates *Drops duplicated rows.*

Description

Drops duplicated rows across specified columns.

Usage

```
h2o.drop_duplicates(frame, columns, keep = "first")
```

Arguments

frame	An H2OFrame object to drop duplicates on.
columns	Columns to compare during the duplicate detection process.
keep	Which rows to keep. The "first" value (default) keeps the first row and deletes the rest. The "last" keeps the last row.

Examples

```
## Not run:
library(h2o)
h2o.init()

data <- as.h2o(iris)
deduplicated_data <- h2o.drop_duplicates(data, c("Species", "Sepal.Length"), keep = "first")

## End(Not run)
```

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.

Examples

```
## Not run:
library(h2o)
h2o.init()
buys <- as.h2o(c("no", "no", "yes", "yes", "yes", "no", "yes", "no", "yes", "yes","no"))
buys_entropy <- h2o.entropy(buys)

## End(Not run)
```

h2o.exp *Compute the exponential function of x*

Description

Compute the exponential function of x

Usage

```
h2o.exp(x)
```

Arguments

x	An H2OFrame object.

See Also

Log for the base R implementation, exp().

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.exp(frame["C1"])

## End(Not run)
```

h2o.explain *Generate Model Explanations*

Description

The H2O Explainability Interface is a convenient wrapper to a number of explainabilty methods and visualizations in H2O. The function can be applied to a single model or group of models and returns a list of explanations, which are individual units of explanation such as a partial dependence plot or a variable importance plot. Most of the explanations are visual (ggplot plots). These plots can also be created by individual utility functions as well.

Usage

```
h2o.explain(
  object,
  newdata,
  columns = NULL,
  top_n_features = 5,
  include_explanations = "ALL",
  exclude_explanations = NULL,
  plot_overrides = NULL
)
```

Arguments

`object`	A list of H2O models, an H2O AutoML instance, or an H2OFrame with a 'model_id' column (e.g. H2OAutoML leaderboard).
`newdata`	An H2OFrame.
`columns`	A vector of column names or column indices to create plots with. If specified parameter top_n_features will be ignored.
`top_n_features`	An integer specifying the number of columns to use, ranked by variable importance (where applicable).
`include_explanations`	If specified, return only the specified model explanations. (Mutually exclusive with exclude_explanations)
`exclude_explanations`	Exclude specified model explanations.
`plot_overrides`	Overrides for individual model explanations, e.g. `list(shap_summary_plot = list(columns = 50`

Value

List of outputs with class "H2OExplanation"

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                   seed = 1)

# Create the explanation for whole H2OAutoML object
exa <- h2o.explain(aml, test)
print(exa)

# Create the explanation for the leader model
exm <- h2o.explain(aml@leader, test)
print(exm)

## End(Not run)
```

`h2o.explain_row` *Generate Model Explanations for a single row*

Description

Explain the behavior of a model or group of models with respect to a single row of data. The function returns a list of explanations, which are individual units of explanation such as a partial dependence plot or a variable importance plot. Most of the explanations are visual (ggplot plots). These plots can also be created by individual utility functions as well.

Usage

```
h2o.explain_row(
  object,
  newdata,
  row_index,
  columns = NULL,
  top_n_features = 5,
  include_explanations = "ALL",
  exclude_explanations = NULL,
  plot_overrides = NULL
)
```

Arguments

`object`	A list of H2O models, an H2O AutoML instance, or an H2OFrame with a 'model_id' column (e.g. H2OAutoML leaderboard).
`newdata`	An H2OFrame.
`row_index`	A row index of the instance to explain.
`columns`	A vector of column names or column indices to create plots with. If specified parameter top_n_features will be ignored.
`top_n_features`	An integer specifying the number of columns to use, ranked by variable importance (where applicable).
`include_explanations`	If specified, return only the specified model explanations. (Mutually exclusive with exclude_explanations)
`exclude_explanations`	Exclude specified model explanations.
`plot_overrides`	Overrides for individual model explanations, e.g., `list(shap_explain_row = list(columns = 5))`

Value

List of outputs with class "H2OExplanation"

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)

# Create the explanation for whole H2OAutoML object
exa <- h2o.explain_row(aml, test, row_index = 1)
print(exa)

# Create the explanation for the leader model
exm <- h2o.explain_row(aml@leader, test, row_index = 1)
print(exm)

## End(Not run)
```

`h2o.exportFile` *Export an H2O Data Frame (H2OFrame) to a File or to a collection of Files.*

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,
  sep = ",",
  compression = NULL,
  parts = 1,
  header = TRUE,
  quote_header = TRUE,
  format = "csv"
)
```

Arguments

data	An H2OFrame object.
path	The path to write the file to. Must include the directory and also filename if exporting to a single file. 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.
sep	The field separator character. Values on each line of the file will be separated by this character (default ",").
compression	How to compress the exported dataset (default none; gzip, bzip2 and snappy available)
parts	integer, number of part files to export to. Default is to write to a single file. Large data can be exported to multiple 'part' files, where each part file contains subset of the data. User can specify the maximum number of part files or use value -1 to indicate that H2O should itself determine the optimal number of files. Parameter path will be considered to be a path to a directory if export to multiple part files is desired. Part files conform to naming scheme 'part-m-?????'.
header	logical, indicates whether to write the header line. Default is to include the header in the output file.
quote_header	logical, indicates whether column names should be quoted. Default is to use quotes.
format	string, one of "csv" or "parquet". Default is "csv". Export to parquet is multipart and H2O itself determines the optimal number of files (1 file per chunk).

Details

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

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)

# These aren't real paths
# h2o.exportFile(iris_hf, path = "/path/on/h2o/server/filesystem/iris.csv")
# h2o.exportFile(iris_hf, path = "hdfs://path/in/hdfs/iris.csv")
# h2o.exportFile(iris_hf, 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.

Examples

```
## Not run:
library(h2o)
h2o.init

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/iris/iris_train.csv"
train <- h2o.importFile(f)
h2o.exportHDFS(train, path = " ", force = FALSE)

## End(Not run)
```

h2o.extendedIsolationForest — *Trains an Extended Isolation Forest model*

Description

Trains an Extended Isolation Forest model

Usage

```
h2o.extendedIsolationForest(
  training_frame,
  x,
  model_id = NULL,
  ignore_const_cols = TRUE,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  ntrees = 100,
  sample_size = 256,
  extension_level = 0,
  seed = -1
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`model_id`	Destination id for this model; auto-generated if not specified.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`ntrees`	Number of Extended Isolation Forest trees. Defaults to 100.
`sample_size`	Number of randomly sampled observations used to train each Extended Isolation Forest tree. Defaults to 256.
`extension_level`	Maximum is N - 1 (N = numCols). Minimum is 0. Extended Isolation Forest with extension_Level = 0 behaves like Isolation Forest. Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the prostate dataset
p <- h2o.importFile(path="https://raw.github.com/h2oai/h2o/master/smalldata/logreg/prostate.csv")

# Set the predictors
predictors <- c("AGE","RACE","DPROS","DCAPS","PSA","VOL","GLEASON")

# Build an Extended Isolation forest model
model <- h2o.extendedIsolationForest(x = predictors,
                                     training_frame = p,
                                     model_id = "eif.hex",
                                     ntrees = 100,
                                     sample_size = 256,
                                     extension_level = length(predictors) - 1)

# Calculate score
score <- h2o.predict(model, p)
anomaly_score <- score$anomaly_score

# Number in [0, 1] explicitly defined in Equation (1) from Extended Isolation Forest paper
# or in paragraph '2 Isolation and Isolation Trees' of Isolation Forest paper
anomaly_score <- score$anomaly_score

# Average path length of the point in Isolation Trees from root to the leaf
mean_length <- score$mean_length

## End(Not run)
```

h2o.fair_pd_plot *Partial dependence plot per protected group.*

Description

Partial dependence plot per protected group.

Usage

```
h2o.fair_pd_plot(model, newdata, protected_columns, column, autoscale = TRUE)
```

Arguments

model	H2O Model Object
newdata	H2OFrame
protected_columns	List of categorical columns that contain sensitive information such as race, gender, age etc.
column	String containing column name.
autoscale	If "True", try to guess when to use log transformation on X axis.

Value

ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
       'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
        'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.fair_pd_plot(gbm, test, protected_columns, "AGE")

## End(Not run)
```

h2o.fair_pr_plot *Plot PR curve per protected group.*

Description

Plot PR curve per protected group.

Usage

```
h2o.fair_pr_plot(model, newdata, protected_columns, reference, favorable_class)
```

Arguments

model	H2O Model Object
newdata	H2OFrame
protected_columns	List of categorical columns that contain sensitive information such as race, gender, age etc.
reference	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
favorable_class	Positive/favorable outcome class of the response.

Value

ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
      'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
       'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.fair_pr_plot(gbm, test, protected_columns = protected_columns,
                  reference = reference, favorable_class = favorable_class)

## End(Not run)
```

h2o.fair_roc_plot *Plot ROC curve per protected group.*

Description

Plot ROC curve per protected group.

Usage

```
h2o.fair_roc_plot(
  model,
  newdata,
  protected_columns,
  reference,
  favorable_class
)
```

Arguments

`model`	H2O Model Object
`newdata`	H2OFrame
`protected_columns`	List of categorical columns that contain sensitive information such as race, gender, age etc.
`reference`	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
`favorable_class`	Positive/favorable outcome class of the response.

Value

ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
     'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
      'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])
```

```
splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.fair_roc_plot(gbm, test, protected_columns = protected_columns,
                  reference = reference, favorable_class = favorable_class)

## End(Not run)
```

`h2o.fair_shap_plot` *SHAP summary plot for one feature with protected groups on y-axis.*

Description

SHAP summary plot for one feature with protected groups on y-axis.

Usage

```
h2o.fair_shap_plot(model, newdata, protected_columns, column, autoscale = TRUE)
```

Arguments

model	H2O Model Object
newdata	H2OFrame
protected_columns	List of categorical columns that contain sensitive information such as race, gender, age etc.
column	String containing column name.
autoscale	If TRUE, try to guess when to use log transformation on X axis.

Value

list of ggplot2 objects

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
     'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
      'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')
```

```
for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.fair_shap_plot(gbm, test, protected_columns, "AGE")

## End(Not run)
```

`h2o.feature_interaction` *Feature interactions and importance, leaf statistics and split value histograms in a tabular form. Available for XGBoost and GBM.*

Description

Metrics: Gain - Total gain of each feature or feature interaction. FScore - Amount of possible splits taken on a feature or feature interaction. wFScore - Amount of possible splits taken on a feature or feature interaction weighed by the probability of the splits to take place. Average wFScore - wFScore divided by FScore. Average Gain - Gain divided by FScore. Expected Gain - Total gain of each feature or feature interaction weighed by the probability to gather the gain. Average Tree Index Average Tree Depth

Usage

```
h2o.feature_interaction(
  model,
  max_interaction_depth = 100,
  max_tree_depth = 100,
  max_deepening = -1
)
```

Arguments

`model`	A trained xgboost model.
`max_interaction_depth`	Upper bound for extracted feature interactions depth. Defaults to 100.
`max_tree_depth`	Upper bound for tree depth. Defaults to 100.
`max_deepening`	Upper bound for interaction start deepening (zero deepening => interactions starting at root only). Defaults to -1.

Examples

```
## Not run:
library(h2o)
h2o.init()
boston <- h2o.importFile(
      "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv",
        destination_frame="boston"
        )
boston_xgb <- h2o.xgboost(training_frame = boston, y = "medv", seed = 1234)
feature_interactions <- h2o.feature_interaction(boston_xgb)

## End(Not run)
```

---

h2o.fillna *fillNA*

Description

Fill NA's in a sequential manner up to a specified limit

Usage

```
h2o.fillna(x, method = "forward", axis = 1, maxlen = 1L)
```

Arguments

x	an H2OFrame
method	A String: "forward" or "backward"
axis	An Integer 1 for row-wise fill (default), 2 for column-wise fill
maxlen	An Integer for maximum number of consecutive NA's to fill

Value

An H2OFrame after filling missing values

Examples

```
## Not run:
library(h2o)
h2o.init()

frame_with_nas <- h2o.createFrame(rows = 6, cols = 2,
                                  categorical_fraction = 0.0,
                                  missing_fraction = 0.7,
                                  seed = 123)
frame <- h2o.fillna(frame_with_nas, "forward", axis = 1, maxlen = 2L)

## End(Not run)
```

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)

Value

Returns a numeric vector of indexes that pertain to non-NA columns

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.filterNACols(frame, frac = 0.5)
h2o.filterNACols(frame, frac = 0.6)

## End(Not run)
```

h2o.findSynonyms *Find synonyms using a word2vec model.*

Description

Find synonyms using a word2vec model.

Usage

```
h2o.findSynonyms(word2vec, word, count = 20)
```

Arguments

word2vec	A word2vec model.
word	A single word to find synonyms for.
count	The top ‘count‘ synonyms will be returned.

Examples

```
## Not run:
library(h2o)
h2o.init()

job_titles <- h2o.importFile(
    "https://s3.amazonaws.com/h2o-public-test-data/smalldata/craigslistJobTitles.csv",
   col.names = c("category", "jobtitle"), col.types = c("String", "String"), header = TRUE
)
words <- h2o.tokenize(job_titles, " ")
vec <- h2o.word2vec(training_frame = words)
h2o.findSynonyms(vec, "teacher", count = 20)

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
                    training_frame = train, validation_frame = valid,
                    build_tree_one_node = TRUE , seed = 1234)
perf <- h2o.performance(cars_gbm, cars)
h2o.find_row_by_threshold(perf, 0.5)

## End(Not run)
```

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
                    training_frame = train, validation_frame = valid,
                    build_tree_one_node = TRUE , seed = 1234)
perf <- h2o.performance(cars_gbm, cars)
h2o.find_threshold_by_max_metric(perf, "fnr")

## End(Not run)
```

h2o.floor — *Take a single numeric argument and return a numeric vector with the largest integers*

Description

floor takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.

Usage

```
h2o.floor(x)
```

Arguments

x	An H2OFrame object.

See Also

Round for the base R implementation, `floor()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.floor(frame["C2"])

## End(Not run)
```

`h2o.flow` *Open H2O Flow*

Description

Open H2O Flow in your browser

Usage

```
h2o.flow()
```

`h2o.gainsLift` *Access H2O Gains/Lift Tables*

Description

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

Usage

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

h2o.gains_lift(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.
`...`	further arguments to be passed to/from this method.
`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.

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, distribution = "bernoulli",
                 training_frame = prostate, validation_frame = prostate, 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 = prostate) ## score on new data (here: the same)
# Generating a ModelMetrics object
perf <- h2o.performance(model, prostate)
h2o.gainsLift(perf)               ## extract from existing metrics object

## End(Not run)
```

`h2o.gains_lift_plot` *Plot Gains/Lift curves*

Description

Plot Gains/Lift curves

Usage

```
h2o.gains_lift_plot(object, type = c("both", "gains", "lift"), ...)
```

Arguments

`object`	Either an H2OModel or H2OModelMetrics
`type`	What curve to plot. One of "both", "gains", "lift".
`...`	Optional arguments

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(
path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip")
model <- h2o.gbm(x = c("Origin", "Distance"), y = "IsDepDelayed", training_frame = data, ntrees = 1)
h2o.gains_lift_plot(model)

## End(Not run)
```

`h2o.gains_lift_plot,H2OModel-method` *Plot Gains/Lift curves*

Description

Plot Gains/Lift curves

Usage

```
## S4 method for signature 'H2OModel'
h2o.gains_lift_plot(object, type = c("both", "gains", "lift"), xval = FALSE)
```

Arguments

`object`	H2OModel object
`type`	What curve to plot. One of "both", "gains", "lift".
`xval`	if TRUE, use cross-validation metrics

`h2o.gains_lift_plot,H2OModelMetrics-method` *Plot Gains/Lift curves*

Description

Plot Gains/Lift curves

Usage

```
## S4 method for signature 'H2OModelMetrics'
h2o.gains_lift_plot(object, type = c("both", "gains", "lift"))
```

Arguments

`object`	H2OModelMetrics object
`type`	What curve to plot. One of "both", "gains", "lift".

`h2o.gam` *Fit a General Additive Model*

Description

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

Usage

```
h2o.gam(
  x,
  y,
  training_frame,
  gam_columns,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  seed = -1,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  offset_column = NULL,
  weights_column = NULL,
 family = c("AUTO", "gaussian", "binomial", "quasibinomial", "ordinal", "multinomial",
    "poisson", "gamma", "tweedie", "negativebinomial", "fractionalbinomial"),
  tweedie_variance_power = 0,
  tweedie_link_power = 0,
  theta = 0,
  solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
    "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
  alpha = NULL,
  lambda = NULL,
  lambda_search = FALSE,
  early_stopping = TRUE,
  nlambdas = -1,
  standardize = FALSE,
  missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
  plug_values = NULL,
  compute_p_values = FALSE,
  remove_collinear_columns = FALSE,
  splines_non_negative = NULL,
  intercept = TRUE,
  non_negative = FALSE,
  max_iterations = -1,
  objective_epsilon = -1,
  beta_epsilon = 1e-04,
  gradient_epsilon = -1,
  link = c("family_default", "identity", "logit", "log", "inverse", "tweedie",
    "ologit"),
  startval = NULL,
  prior = -1,
  cold_start = FALSE,
  lambda_min_ratio = -1,
  beta_constraints = NULL,
  max_active_predictors = -1,
  interactions = NULL,
  interaction_pairs = NULL,
  obj_reg = -1,
  export_checkpoints_dir = NULL,
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
    "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = 0,
  custom_metric_func = NULL,
  num_knots = NULL,
  spline_orders = NULL,
  knot_ids = NULL,
  standardize_tp_gam_cols = FALSE,
  scale_tp_penalty_mat = FALSE,
  bs = NULL,
  scale = NULL,
  keep_gam_cols = FALSE,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO")
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`gam_columns`	Arrays of predictor column names for gam for smoothers using single or multiple predictors like 'c1','c2','c3','c4',...
`model_id`	Destination id for this model; auto-generated if not specified.

`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`family`	Family. Use binomial for classification with logistic regression, others are for regression problems. Must be one of: "AUTO", "gaussian", "binomial", "quasibinomial", "ordinal", "multinomial", "poisson", "gamma", "tweedie", "negativebinomial", "fractionalbinomial". Defaults to AUTO.
`tweedie_variance_power`	Tweedie variance power Defaults to 0.
`tweedie_link_power`	Tweedie link power Defaults to 0.
`theta`	Theta Defaults to 0.
`solver`	AUTO will set the solver based on given data and the other parameters. IRLSM is fast on on problems with small number of predictors and for lambda-search with L1 penalty, L_BFGS scales better for datasets with many columns. Must be

	one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR Defaults to AUTO.
`alpha`	Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.
`lambda`	Regularization strength
`lambda_search`	`Logical`. Use lambda search starting at lambda max, given lambda is then interpreted as lambda min Defaults to FALSE.
`early_stopping`	`Logical`. Stop early when there is no more relative improvement on train or validation (if provided) Defaults to TRUE.
`nlambdas`	Number of lambdas to be used in a search. Default indicates: If alpha is zero, with lambda search set to True, the value of nlamdas is set to 30 (fewer lambdas are needed for ridge regression) otherwise it is set to 100. Defaults to -1.
`standardize`	`Logical`. Standardize numeric columns to have zero mean and unit variance Defaults to FALSE.
`missing_values_handling`	Handling of missing values. Either MeanImputation, Skip or PlugValues. Must be one of: "MeanImputation", "Skip", "PlugValues". Defaults to MeanImputation.
`plug_values`	Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues)
`compute_p_values`	`Logical`. Request p-values computation, p-values work only with IRLSM solver and no regularization Defaults to FALSE.
`remove_collinear_columns`	`Logical`. In case of linearly dependent columns, remove some of the dependent columns Defaults to FALSE.
`splines_non_negative`	Valid for I-spline (bs=2) only. True if the I-splines are monotonically increasing (and monotonically non- decreasing) and False if the I-splines are monotonically decreasing (and monotonically non-increasing). If specified, must be the same size as gam_columns. Values for other spline types will be ignored. Default to true.
`intercept`	`Logical`. Include constant term in the model Defaults to TRUE.
`non_negative`	`Logical`. Restrict coefficients (not intercept) to be non-negative Defaults to FALSE.
`max_iterations`	Maximum number of iterations Defaults to -1.
`objective_epsilon`	Converge if objective value changes less than this. Default indicates: If lambda_search is set to True the value of objective_epsilon is set to .0001. If the lambda_search is set to False and lambda is equal to zero, the value of objective_epsilon is set to .000001, for any other value of lambda the default value of objective_epsilon is set to .0001. Defaults to -1.
`beta_epsilon`	Converge if beta changes less (using L-infinity norm) than beta esilon, ONLY applies to IRLSM solver Defaults to 0.0001.

`gradient_epsilon`	Converge if objective changes less (using L-infinity norm) than this, ONLY applies to L-BFGS solver. Default indicates: If lambda_search is set to False and lambda is equal to zero, the default value of gradient_epsilon is equal to .000001, otherwise the default value is .0001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively. Defaults to -1.
`link`	Link function. Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit". Defaults to family_default.
`startval`	double array to initialize coefficients for GAM.
`prior`	Prior probability for y==1. To be used only for logistic regression iff the data has been sampled and the mean of response does not reflect reality. Defaults to -1.
`cold_start`	`Logical`. Only applicable to multiple alpha/lambda values when calling GLM from GAM. If false, build the next model for next set of alpha/lambda values starting from the values provided by current model. If true will start GLM model from scratch. Defaults to FALSE.
`lambda_min_ratio`	Minimum lambda used in lambda search, specified as a ratio of lambda_max (the smallest lambda that drives all coefficients to zero). Default indicates: if the number of observations is greater than the number of variables, then lambda_min_ratio is set to 0.0001; if the number of observations is less than the number of variables, then lambda_min_ratio is set to 0.01. Defaults to -1.
`beta_constraints`	Beta constraints
`max_active_predictors`	Maximum number of active predictors during computation. Use as a stopping criterion to prevent expensive model building with many predictors. Default indicates: If the IRLSM solver is used, the value of max_active_predictors is set to 5000 otherwise it is set to 100000000. Defaults to -1.
`interactions`	A list of predictor column indices to interact. All pairwise combinations will be computed for the list.
`interaction_pairs`	A list of pairwise (first order) column interactions.
`obj_reg`	Likelihood divider in objective value computation, default is 1/nobs Defaults to -1.
`export_checkpoints_dir`	Automatically export generated models to this directory.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

`balance_classes`	`Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
`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). Requires balance_classes. Defaults to 5.0.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`custom_metric_func`	Reference to custom evaluation function, format: ‘language:keyName=funcName‘
`num_knots`	Number of knots for gam predictors. If specified, must specify one for each gam predictor. For monotone I-splines, mininum = 2, for cs spline, minimum = 3. For thin plate, minimum is size of polynomial basis + 2.
`spline_orders`	Order of I-splines or NBSplineTypeI M-splines used for gam predictors. If specified, must be the same size as gam_columns. For I-splines, the spline_orders will be the same as the polynomials used to generate the splines. For M-splines, the polynomials used to generate the splines will be spline_order-1. Values for bs=0 or 1 will be ignored.
`knot_ids`	Array storing frame keys of knots. One for each gam column set specified in gam_columns
`standardize_tp_gam_cols`	`Logical`. standardize tp (thin plate) predictor columns Defaults to FALSE.
`scale_tp_penalty_mat`	`Logical`. Scale penalty matrix for tp (thin plate) smoothers as in R Defaults to FALSE.
`bs`	Basis function type for each gam predictors, 0 for cr, 1 for thin plate regression with knots, 2 for monotone I-splines, 3 for NBSplineTypeI M-splines (refer to doc here: https://h2oai.atlassian.net/browse/PUBDEV-8835). If specified, must be the same size as gam_columns
`scale`	Smoothing parameter for gam predictors. If specified, must be of the same length as gam_columns
`keep_gam_cols`	`Logical`. Save keys of model matrix Defaults to FALSE.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.

Examples

```
## Not run:
h2o.init()

# Run GAM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
h2o.gam(y = "CAPSULE", x = c("RACE"), gam_columns = c("PSA"),
     training_frame = prostate, family = "binomial")


## End(Not run)
```

---

h2o.gbm *Build gradient boosted classification or regression trees*

Description

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

Usage

```
h2o.gbm(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  score_each_iteration = FALSE,
  score_tree_interval = 0,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  offset_column = NULL,
  weights_column = NULL,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  ntrees = 50,
  max_depth = 5,
  min_rows = 10,
  nbins = 20,
  nbins_top_level = 1024,
  nbins_cats = 1024,
  r2_stopping = Inf,
  stopping_rounds = 0,
 stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
   "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
   "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  max_runtime_secs = 0,
  seed = -1,
  build_tree_one_node = FALSE,
  learn_rate = 0.1,
  learn_rate_annealing = 1,
  distribution = c("AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian",
    "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom"),
  quantile_alpha = 0.5,
  tweedie_power = 1.5,
  huber_alpha = 0.9,
  checkpoint = NULL,
  sample_rate = 1,
  sample_rate_per_class = NULL,
  col_sample_rate = 1,
  col_sample_rate_change_per_level = 1,
  col_sample_rate_per_tree = 1,
  min_split_improvement = 1e-05,
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal",
    "RoundRobin", "UniformRobust"),
  max_abs_leafnode_pred = Inf,
  pred_noise_bandwidth = 0,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  calibrate_model = FALSE,
  calibration_frame = NULL,
  calibration_method = c("AUTO", "PlattScaling", "IsotonicRegression"),
  custom_metric_func = NULL,
  custom_distribution_func = NULL,
  export_checkpoints_dir = NULL,
  in_training_checkpoints_dir = NULL,
  in_training_checkpoints_tree_interval = 1,
  monotone_constraints = NULL,
  check_constant_response = TRUE,
  gainslift_bins = -1,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  interaction_constraints = NULL,
  verbose = FALSE
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

keep_cross_validation_models	Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions	Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment	Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
score_each_iteration	Logical. Whether to score during each iteration of model training. Defaults to FALSE.
score_tree_interval	Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
fold_assignment	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
fold_column	Column with cross-validation fold index assignment per observation.
ignore_const_cols	Logical. Ignore constant columns. Defaults to TRUE.
offset_column	Offset column. This will be added to the combination of columns before applying the link function.
weights_column	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
balance_classes	Logical. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
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). Requires balance_classes. Defaults to 5.0.
ntrees	Number of trees. Defaults to 50.
max_depth	Maximum tree depth (0 for unlimited). Defaults to 5.
min_rows	Fewest allowed (weighted) observations in a leaf. Defaults to 10.
nbins	For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.
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 Defaults to 1024.

`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. Defaults to 1024.
`r2_stopping`	r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this Defaults to 1.797693135e+308.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`build_tree_one_node`	`Logical`. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
`learn_rate`	Learning rate (from 0.0 to 1.0) Defaults to 0.1.
`learn_rate_annealing`	Scale the learning rate by this factor after each tree (e.g., 0.99 or 0.999) Defaults to 1.
`distribution`	Distribution function Must be one of: "AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom". Defaults to AUTO.
`quantile_alpha`	Desired quantile for Quantile regression, must be between 0 and 1. Defaults to 0.5.
`tweedie_power`	Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to 1.5.
`huber_alpha`	Desired quantile for Huber/M-regression (threshold between quadratic and linear loss, must be between 0 and 1). Defaults to 0.9.
`checkpoint`	Model checkpoint to resume training with.
`sample_rate`	Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`sample_rate_per_class`	A list of row sample rates per class (relative fraction for each class, from 0.0 to 1.0), for each tree
`col_sample_rate`	Column sample rate (from 0.0 to 1.0) Defaults to 1.

`col_sample_rate_change_per_level`
: Relative change of the column sampling rate for every level (must be > 0.0 and <= 2.0) Defaults to 1.

`col_sample_rate_per_tree`
: Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.

`min_split_improvement`
: Minimum relative improvement in squared error reduction for a split to happen Defaults to 1e-05.

`histogram_type`
: What type of histogram to use for finding optimal split points Must be one of: "AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin", "UniformRobust". Defaults to AUTO.

`max_abs_leafnode_pred`
: Maximum absolute value of a leaf node prediction Defaults to 1.797693135e+308.

`pred_noise_bandwidth`
: Bandwidth (sigma) of Gaussian multiplicative noise ~N(1,sigma) for tree node predictions Defaults to 0.

`categorical_encoding`
: Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.

`calibrate_model`
: `Logical`. Use Platt Scaling (default) or Isotonic Regression to calculate calibrated class probabilities. Calibration can provide more accurate estimates of class probabilities. Defaults to FALSE.

`calibration_frame`
: Data for model calibration

`calibration_method`
: Calibration method to use Must be one of: "AUTO", "PlattScaling", "IsotonicRegression". Defaults to AUTO.

`custom_metric_func`
: Reference to custom evaluation function, format: ‘language:keyName=funcName‘

`custom_distribution_func`
: Reference to custom distribution, format: ‘language:keyName=funcName‘

`export_checkpoints_dir`
: Automatically export generated models to this directory.

`in_training_checkpoints_dir`
: Create checkpoints into defined directory while training process is still running. In case of cluster shutdown, this checkpoint can be used to restart training.

`in_training_checkpoints_tree_interval`
: Checkpoint the model after every so many trees. Parameter is used only when in_training_checkpoints_dir is defined Defaults to 1.

`monotone_constraints`
: A mapping representing monotonic constraints. Use +1 to enforce an increasing constraint and -1 to specify a decreasing constraint.

`check_constant_response`
: `Logical`. Check if response column is constant. If enabled, then an exception is thrown if the response column is a constant value.If disabled, then model will train regardless of the response column being a constant value or not. Defaults to TRUE.

gainslift_bins	Gains/Lift table number of bins. 0 means disabled.. Default value -1 means automatic binning. Defaults to -1.
auc_type	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
interaction_constraints	A set of allowed column interactions.
verbose	Logical. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

See Also

predict.H2OModel for prediction

Examples

```
## Not run:
library(h2o)
h2o.init()

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

## End(Not run)
```

h2o.generic — *Imports a generic model into H2O. Such model can be used then used for scoring and obtaining additional information about the model. The imported model has to be supported by H2O.*

Description

Imports a generic model into H2O. Such model can be used then used for scoring and obtaining additional information about the model. The imported model has to be supported by H2O.

Usage

```
h2o.generic(model_id = NULL, model_key = NULL, path = NULL)
```

Arguments

model_id	Destination id for this model; auto-generated if not specified.
model_key	Key to the self-contained model archive already uploaded to H2O.
path	Path to file with self-contained model archive.

Examples

```
## Not run:
# library(h2o)
# h2o.init()

# generic_model <- h2o.genericModel(path="/path/to/model.zip", model_id="my_model")
# predictions <- h2o.predict(generic_model, dataset)

## End(Not run)
```

---

h2o.genericModel — *Imports a model under given path, creating a Generic model with it.*

Description

Usage example: generic_model <- h2o.genericModel(model_file_path = "/path/to/mojo.zip") predictions <- h2o.predict(generic_model, dataset)

Usage

```
h2o.genericModel(mojo_file_path, model_id = NULL)
```

Arguments

`mojo_file_path`	Filesystem path to the model imported
`model_id`	Model ID, default is NULL

Value

Returns H2O Generic Model based on given embedded model

Examples

```
## Not run:

# Import default Iris dataset as H2O frame
data <- as.h2o(iris)

# Train a very simple GBM model
features <- c("Sepal.Length", "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
original_model <- h2o.gbm(x = features, y = "Species", training_frame = data)

# Download the trained GBM model as MOJO (temporary directory used in this example)
mojo_original_name <- h2o.download_mojo(model = original_model, path = tempdir())
mojo_original_path <- paste0(tempdir(), "/", mojo_original_name)

# Import the MOJO as Generic model
generic_model <- h2o.genericModel(mojo_original_path)

# Perform scoring with the generic model
generic_model_predictions  <- h2o.predict(generic_model, data)

## End(Not run)
```

`h2o.getAlphaBest` *Extract best alpha value found from glm model.*

Description

This function allows setting betas of an existing glm model.

Usage

```
h2o.getAlphaBest(model)
```

Arguments

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

`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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
train <- h2o.importFile(f)
y <- "species"
x <- setdiff(names(train), y)
train[, y] <- as.factor(train[, y])
nfolds <- 5
num_base_models <- 2
my_gbm <- h2o.gbm(x = x, y = y, training_frame = train,
                  distribution = "multinomial", ntrees = 10,
                  max_depth = 3, min_rows = 2, learn_rate = 0.2,
                  nfolds = nfolds, fold_assignment = "Modulo",
                  keep_cross_validation_predictions = TRUE, seed = 1)
my_rf <- h2o.randomForest(x = x, y = y, training_frame = train,
                          ntrees = 50, nfolds = nfolds, fold_assignment = "Modulo",
                          keep_cross_validation_predictions = TRUE, seed = 1)
stack <- h2o.stackedEnsemble(x = x, y = y, training_frame = train,
                             model_id = "my_ensemble_l1",
                             base_models = list(my_gbm@model_id, my_rf@model_id),
                             keep_levelone_frame = TRUE)
h2o.getFrame(stack@model$levelone_frame_id$name)

## End(Not run)
```

`h2o.getGLMFullRegularizationPath`

Extract full regularization path from a GLM model

Description

Extract the full regularization path from a GLM model (assuming it was run with the 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

Note that if neither cross-validation nor a validation frame is used in the grid search, then the training metrics will display in the "get grid" output. If a validation frame is passed to the grid, and nfolds = 0, then the validation metrics will display. However, if nfolds > 1, then cross-validation metrics will display even if a validation frame is provided.

Usage

```
h2o.getGrid(grid_id, sort_by, decreasing, verbose = FALSE)
```

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", "accuracy", "precision", "recall", "f1", etc.
decreasing	Specify whether sort order should be decreasing
verbose	Controls verbosity of the output, if enabled prints out error messages for failed models (default: FALSE)

Examples

```
## Not run:
library(h2o)
library(jsonlite)
h2o.init()
iris_hf <- as.h2o(iris)
h2o.grid("gbm", grid_id = "gbm_grid_id", x = c(1:4), y = 5,
         training_frame = iris_hf, 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)})

## End(Not run)
```

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 of the H2OFrame

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.getId(iris)

## End(Not run)
```

`h2o.getLambdaBest` *Extract best lambda value found from glm model.*

Description

This function allows setting betas of an existing glm model.

Usage

```
h2o.getLambdaBest(model)
```

Arguments

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

`h2o.getLambdaMax` *Extract the maximum lambda value used during lambda search from glm model.*

Description

This function allows setting betas of an existing glm model.

Usage

```
h2o.getLambdaMax(model)
```

Arguments

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

`h2o.getLambdaMin` *Extract the minimum lambda value calculated during lambda search from glm model. Note that due to early stop, this minimum lambda value may not be used in the actual lambda search.*

Description

This function allows setting betas of an existing glm model.

Usage

```
h2o.getLambdaMin(model)
```

Arguments

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

`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

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
model_id <- h2o.gbm(x = 1:4, y = 5, training_frame = iris_hf)@model_id
model_retrieved <- h2o.getModel(model_id)

## End(Not run)
```

`h2o.getModelTree` *Fetchces a single tree of a H2O model. This function is intended to be used on Gradient Boosting Machine models or Distributed Random Forest models.*

Description

Fetchces a single tree of a H2O model. This function is intended to be used on Gradient Boosting Machine models or Distributed Random Forest models.

Usage

```
h2o.getModelTree(
  model,
  tree_number,
  tree_class = NA,
  plain_language_rules = "AUTO"
)
```

Arguments

`model`	Model with trees
`tree_number`	Number of the tree in the model to fetch, starting with 1
`tree_class`	Name of the class of the tree (if applicable). This value is ignored for regression and binomial response column, as there is only one tree built. As there is exactly one class per categorical level, name of tree's class equals to the corresponding categorical level of response column.
`plain_language_rules`	(Optional) Whether to generate plain language rules. AUTO by default, meaning FALSE for big trees and TRUE for small trees.

Value

Returns an H2OTree object with detailed information about a tree.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
gbm_model <- h2o.gbm(y = "species", training_frame = iris)
tree <- h2o.getModelTree(gbm_model, 1, "Iris-setosa")

## End(Not run)
```

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

Description

Get the Time Zone on the H2O cluster 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 per column

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.getTypes(iris)

## End(Not run)
```

h2o.getVersion *Get h2o version*

Description

Get h2o version

Usage

```
h2o.getVersion()
```

`h2o.get_automl` *Get an R object that is a subclass of H2OAutoML*

Description

Get an R object that is a subclass of H2OAutoML

Usage

```
h2o.get_automl(project_name)

h2o.getAutoML(project_name)
```

Arguments

`project_name`	A string indicating the project_name of the automl instance to retrieve.

Value

Returns an object that is a subclass of H2OAutoML.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path, header = TRUE)
y <- "CAPSULE"
prostate[,y] <- as.factor(prostate[,y])  #convert to factor for classification
aml <- h2o.automl(y = y, training_frame = prostate,
                  max_runtime_secs = 30, project_name = "prostate")
aml2 <- h2o.get_automl("prostate")

## End(Not run)
```

`h2o.get_best_model` *Get best model of a given family/algorithm for a given criterion from an AutoML object.*

Description

Get best model of a given family/algorithm for a given criterion from an AutoML object.

Usage

```
h2o.get_best_model(
  object,
  algorithm = c("any", "basemodel", "deeplearning", "drf", "gbm", "glm",
    "stackedensemble", "xgboost"),
 criterion = c("AUTO", "AUC", "AUCPR", "logloss", "MAE", "mean_per_class_error",
   "deviance", "MSE", "predict_time_per_row_ms", "RMSE", "RMSLE", "training_time_ms")
)
```

Arguments

`object`	H2OAutoML object
`algorithm`	One of "any", "basemodel", "deeplearning", "drf", "gbm", "glm", "stackedensemble", "xgboost"
`criterion`	Criterion can be one of the metrics reported in the leaderboard. If set to NULL, the same ordering as in the leaderboard will be used. Avaliable criteria: • Regression metrics: deviance, RMSE, MSE, MAE, RMSLE • Binomial metrics: AUC, logloss, AUCPR, mean_per_class_error, RMSE, MSE • Multinomial metrics: mean_per_class_error, logloss, RMSE, MSE The following additional leaderboard information can be also used as a criterion: • 'training_time_ms': column providing the training time of each model in milliseconds (doesn't include the training of cross validation models). • 'predict_time_per_row_ms': column providing the average prediction time by the model for a single row.

Value

An H2OModel or NULL if no model of a given family is present

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path, header = TRUE)
y <- "CAPSULE"
prostate[,y] <- as.factor(prostate[,y])  #convert to factor for classification
aml <- h2o.automl(y = y, training_frame = prostate, max_runtime_secs = 30)
gbm <- h2o.get_best_model(aml, "gbm")

## End(Not run)
```

`h2o.get_best_model_predictors`

Extracts the subset of predictor names that yield the best R2 value for each predictor subset size.

Description

Extracts the subset of predictor names that yield the best R2 value for each predictor subset size.

Usage

```
h2o.get_best_model_predictors(model)
```

Arguments

`model`	is a H2OModel with algorithm name of modelselection

`h2o.get_best_r2_values` *Extracts the best R2 values for all predictor subset size.*

Description

Extracts the best R2 values for all predictor subset size.

Usage

```
h2o.get_best_r2_values(model)
```

Arguments

`model`	is a H2OModel with algorithm name of modelselection

`h2o.get_leaderboard` *Retrieve the leaderboard from the AutoML instance.*

Description

Contrary to the default leaderboard attached to the automl instance, this one can return columns other than the metrics.

Usage

```
h2o.get_leaderboard(object, extra_columns = NULL)
```

Arguments

`object`	The object for which to return the leaderboard. Currently, only H2OAutoML instances are supported.
`extra_columns`	A string or a list of string specifying which optional columns should be added to the leaderboard. Defaults to None. Currently supported extensions are: • 'ALL': adds all columns below. • 'training_time_ms': column providing the training time of each model in milliseconds (doesn't include the training of cross validation models). • 'predict_time_per_row_ms': column providing the average prediction time by the model for a single row. • 'algo': column providing the algorithm name for each model.

Value

An H2OFrame representing the leaderboard.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path, header = TRUE)
y <- "CAPSULE"
prostate[,y] <- as.factor(prostate[,y])  #convert to factor for classification
aml <- h2o.automl(y = y, training_frame = prostate, max_runtime_secs = 30)
lb <- h2o.get_leaderboard(aml)
head(lb)

## End(Not run)
```

`h2o.get_ntrees_actual` *Retrieve actual number of trees for tree algorithms*

Description

Retrieve actual number of trees for tree algorithms

Usage

```
h2o.get_ntrees_actual(object)
```

Arguments

`object`	An H2OModel object.

`h2o.get_predictors_added_per_step` *Extracts the predictor added to model at each step.*

Description

Extracts the predictor added to model at each step.

Usage

```
h2o.get_predictors_added_per_step(model)
```

Arguments

`model`	is a H2OModel with algorithm name of modelselection

h2o.get_predictors_removed_per_step

Extracts the predictor removed to model at each step.

Description

Extracts the predictor removed to model at each step.

Usage

```
h2o.get_predictors_removed_per_step(model)
```

Arguments

model	is a H2OModel with algorithm name of modelselection

h2o.get_regression_influence_diagnostics

Extracts a list of H2OFrames containing regression influence diagnostics for predictor subsets of various sizes or just one H2OFrame containing regression influence diagnostics for predictor subsets of one fixed size

Description

Extracts a list of H2OFrames containing regression influence diagnostics for predictor subsets of various sizes or just one H2OFrame containing regression influence diagnostics for predictor subsets of one fixed size

Usage

```
h2o.get_regression_influence_diagnostics(model, predictorSize = -1)
```

Arguments

model	an H2OModel object.
predictorSize	predictor subset size. If specified, will only return model coefficients of that subset size. If not specified will return a lists of model coefficient dicts for all predictor subset size.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "acceleration"
```

```
cars_model <- h2o.modelSelection(y=response,
                                 x=predictors,
                                 training_frame = cars,
                                 min_predictor_number=2,
                                 mode="backward",
                                 influence="dfbetas",
                                 lambda=0.0,
                                 family="gaussian")
rid_frame <- h2o.get_regression_influence_diagnostics(cars_model, predictorSize=3)

## End(Not run)
```

---

h2o.get_segment_models

Retrieves an instance of H2OSegmentModels for a given id.

Description

Retrieves an instance of H2OSegmentModels for a given id.

Usage

```
h2o.get_segment_models(segment_models_id)
```

Arguments

`segment_models_id`	A string indicating the unique segment_models_id

Value

Returns an object that is a subclass of H2OSegmentModels.

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
h2o.train_segments(algorithm = "gbm",
                   segment_columns = "Species", segment_models_id="models_by_species",
                  x = c(1:3), y = 4, training_frame = iris_hf, ntrees = 5, max_depth = 4)
models <- h2o.get_segment_models("models_by_species")
as.data.frame(models)

## End(Not run)
```

`h2o.get_variable_inflation_factors`
Return the variable inflation factors associated with numerical predictors for GLM models.

Description

Return the variable inflation factors associated with numerical predictors for GLM models.

Usage

```
h2o.get_variable_inflation_factors(object)
```

Arguments

`object`	An H2OModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "cylinders"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_glm <- h2o.glm(seed = 1234,
                    lambda=0.0,
                    compute_p_values=TRUE,
                    generate_variable_inflation_factors=TRUE,
                    x = predictors,
                    y = response,
                    training_frame = train,
                    validation_frame = valid)
h2o.get_variable_inflation_factors(cars_glm)

## End(Not run)
```

`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 threshold metrics. See `h2o.performance` for creating H2OModelMetrics objects.

Examples

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

h2o.glm   *Fit a generalized linear model*

Description

Fits 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 = NULL,
  validation_frame = NULL,
  nfolds = 0,
  checkpoint = NULL,
  export_checkpoints_dir = NULL,
  seed = -1,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  random_columns = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  score_iteration_interval = -1,
  offset_column = NULL,
  weights_column = NULL,
  family = c("AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial",
   "ordinal", "multinomial", "poisson", "gamma", "tweedie", "negativebinomial"),
  rand_family = c("[gaussian]"),
  tweedie_variance_power = 0,
  tweedie_link_power = 1,
  theta = 1e-10,
  solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
    "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
  alpha = NULL,
  lambda = NULL,
  lambda_search = FALSE,
  early_stopping = TRUE,
  nlambdas = -1,
  standardize = TRUE,
  missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
  plug_values = NULL,
  compute_p_values = FALSE,
  dispersion_parameter_method = c("deviance", "pearson", "ml"),
  init_dispersion_parameter = 1,
  remove_collinear_columns = FALSE,
  intercept = TRUE,
  non_negative = FALSE,
  max_iterations = -1,
  objective_epsilon = -1,
  beta_epsilon = 1e-04,
  gradient_epsilon = -1,
  link = c("family_default", "identity", "logit", "log", "inverse", "tweedie",
    "ologit"),
  rand_link = c("[identity]", "[family_default]"),
  startval = NULL,
  calc_like = FALSE,
  HGLM = FALSE,
  prior = -1,
  cold_start = FALSE,
  lambda_min_ratio = -1,
  beta_constraints = NULL,
  max_active_predictors = -1,
  interactions = NULL,
  interaction_pairs = NULL,
  obj_reg = -1,
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
   "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = 0,
  custom_metric_func = NULL,
  generate_scoring_history = FALSE,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  dispersion_epsilon = 1e-04,
  tweedie_epsilon = 8e-17,
  max_iterations_dispersion = 3000,
  build_null_model = FALSE,
  fix_dispersion_parameter = FALSE,
  generate_variable_inflation_factors = FALSE,
  fix_tweedie_variance_power = TRUE,
  dispersion_learning_rate = 0.5,
  influence = c("dfbetas")
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
y	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
training_frame	Id of the training data frame.
model_id	Destination id for this model; auto-generated if not specified.
validation_frame	Id of the validation data frame.
nfolds	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
checkpoint	Model checkpoint to resume training with.
export_checkpoints_dir	Automatically export generated models to this directory.
seed	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
keep_cross_validation_models	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

`random_columns`
: random columns indices for HGLM.

`ignore_const_cols`
: `Logical`. Ignore constant columns. Defaults to TRUE.

`score_each_iteration`
: `Logical`. Whether to score during each iteration of model training. Defaults to FALSE.

`score_iteration_interval`
: Perform scoring for every score_iteration_interval iterations Defaults to -1.

`offset_column`
: Offset column. This will be added to the combination of columns before applying the link function.

`weights_column`
: Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.

`family`
: Family. Use binomial for classification with logistic regression, others are for regression problems. Must be one of: "AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial", "ordinal", "multinomial", "poisson", "gamma", "tweedie", "negativebinomial". Defaults to AUTO.

`rand_family`
: Random Component Family array. One for each random component. Only support gaussian for now. Must be one of: "[gaussian]".

`tweedie_variance_power`
: Tweedie variance power Defaults to 0.

`tweedie_link_power`
: Tweedie link power Defaults to 1.

`theta`
: Theta Defaults to 1e-10.

`solver`
: AUTO will set the solver based on given data and the other parameters. IRLSM is fast on on problems with small number of predictors and for lambda-search with L1 penalty, L_BFGS scales better for datasets with many columns. Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR Defaults to AUTO.

`alpha`
: Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.

`lambda`
: Regularization strength

`lambda_search`
: `Logical`. Use lambda search starting at lambda max, given lambda is then interpreted as lambda min Defaults to FALSE.

`early_stopping`	`Logical`. Stop early when there is no more relative improvement on train or validation (if provided) Defaults to TRUE.
`nlambdas`	Number of lambdas to be used in a search. Default indicates: If alpha is zero, with lambda search set to True, the value of nlamdas is set to 30 (fewer lambdas are needed for ridge regression) otherwise it is set to 100. Defaults to -1.
`standardize`	`Logical`. Standardize numeric columns to have zero mean and unit variance Defaults to TRUE.
`missing_values_handling`	Handling of missing values. Either MeanImputation, Skip or PlugValues. Must be one of: "MeanImputation", "Skip", "PlugValues". Defaults to MeanImputation.
`plug_values`	Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues)
`compute_p_values`	`Logical`. Request p-values computation, p-values work only with IRLSM solver and no regularization Defaults to FALSE.
`dispersion_parameter_method`	Method used to estimate the dispersion parameter for Tweedie, Gamma and Negative Binomial only. Must be one of: "deviance", "pearson", "ml". Defaults to pearson.
`init_dispersion_parameter`	Only used for Tweedie, Gamma and Negative Binomial GLM. Store the initial value of dispersion parameter. If fix_dispersion_parameter is set, this value will be used in the calculation of p-values.Default to 1.0. Defaults to 1.
`remove_collinear_columns`	`Logical`. In case of linearly dependent columns, remove some of the dependent columns Defaults to FALSE.
`intercept`	`Logical`. Include constant term in the model Defaults to TRUE.
`non_negative`	`Logical`. Restrict coefficients (not intercept) to be non-negative Defaults to FALSE.
`max_iterations`	Maximum number of iterations Defaults to -1.
`objective_epsilon`	Converge if objective value changes less than this. Default (of -1.0) indicates: If lambda_search is set to True the value of objective_epsilon is set to .0001. If the lambda_search is set to False and lambda is equal to zero, the value of objective_epsilon is set to .000001, for any other value of lambda the default value of objective_epsilon is set to .0001. Defaults to -1.
`beta_epsilon`	Converge if beta changes less (using L-infinity norm) than beta esilon, ONLY applies to IRLSM solver Defaults to 0.0001.
`gradient_epsilon`	Converge if objective changes less (using L-infinity norm) than this, ONLY applies to L-BFGS solver. Default (of -1.0) indicates: If lambda_search is set to False and lambda is equal to zero, the default value of gradient_epsilon is equal to .000001, otherwise the default value is .0001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively. Defaults to -1.
`link`	Link function. Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit". Defaults to family_default.

`rand_link`	Link function array for random component in HGLM. Must be one of: "[identity]", "[family_default]".
`startval`	double array to initialize fixed and random coefficients for HGLM, coefficients for GLM.
`calc_like`	`Logical`. if true, will return likelihood function value for HGLM. Defaults to FALSE.
`HGLM`	`Logical`. If set to true, will return HGLM model. Otherwise, normal GLM model will be returned Defaults to FALSE.
`prior`	Prior probability for y==1. To be used only for logistic regression iff the data has been sampled and the mean of response does not reflect reality. Defaults to -1.
`cold_start`	`Logical`. Only applicable to multiple alpha/lambda values. If false, build the next model for next set of alpha/lambda values starting from the values provided by current model. If true will start GLM model from scratch. Defaults to FALSE.
`lambda_min_ratio`	Minimum lambda used in lambda search, specified as a ratio of lambda_max (the smallest lambda that drives all coefficients to zero). Default indicates: if the number of observations is greater than the number of variables, then lambda_min_ratio is set to 0.0001; if the number of observations is less than the number of variables, then lambda_min_ratio is set to 0.01. Defaults to -1.
`beta_constraints`	Beta constraints
`max_active_predictors`	Maximum number of active predictors during computation. Use as a stopping criterion to prevent expensive model building with many predictors. Default indicates: If the IRLSM solver is used, the value of max_active_predictors is set to 5000 otherwise it is set to 100000000. Defaults to -1.
`interactions`	A list of predictor column indices to interact. All pairwise combinations will be computed for the list.
`interaction_pairs`	A list of pairwise (first order) column interactions.
`obj_reg`	Likelihood divider in objective value computation, default (of -1.0) will set it to 1/nobs Defaults to -1.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

`balance_classes`
: `Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.

`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). Requires balance_classes. Defaults to 5.0.

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

`custom_metric_func`
: Reference to custom evaluation function, format: ‘language:keyName=funcName‘

`generate_scoring_history`
: `Logical`. If set to true, will generate scoring history for GLM. This may significantly slow down the algo. Defaults to FALSE.

`auc_type`
: Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.

`dispersion_epsilon`
: If changes in dispersion parameter estimation or loglikelihood value is smaller than dispersion_epsilon, will break out of the dispersion parameter estimation loop using maximum likelihood. Defaults to 0.0001.

`tweedie_epsilon`
: In estimating tweedie dispersion parameter using maximum likelihood, this is used to choose the lower and upper indices in the approximating of the infinite series summation. Defaults to 8e-17.

`max_iterations_dispersion`
: Control the maximum number of iterations in the dispersion parameter estimation loop using maximum likelihood. Defaults to 3000.

`build_null_model`
: `Logical`. If set, will build a model with only the intercept. Default to false. Defaults to FALSE.

`fix_dispersion_parameter`
: `Logical`. Only used for Tweedie, Gamma and Negative Binomial GLM. If set, will use the dispsersion parameter in init_dispersion_parameter as the standard error and use it to calculate the p-values. Default to false. Defaults to FALSE.

`generate_variable_inflation_factors`
: `Logical`. if true, will generate variable inflation factors for numerical predictors. Default to false. Defaults to FALSE.

`fix_tweedie_variance_power`
: `Logical`. If true, will fix tweedie variance power value to the value set in tweedie_variance_power. Defaults to TRUE.

`dispersion_learning_rate`
: Dispersion learning rate is only valid for tweedie family dispersion parameter estimation using ml. It must be > 0. This controls how much the dispersion parameter estimate is to be changed when the calculated loglikelihood actually decreases with the new dispersion. In this case, instead of setting new dispersion = dispersion - change, we set new dispersion = dispersion + dispersion_learning_rate * change. Defaults to 0.5. Defaults to 0.5.

`influence`	If set to dfbetas will calculate the difference in beta when a datarow is included and excluded in the dataset. Must be one of: "dfbetas".

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: `https://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

```
## Not run:
h2o.init()

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

# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
predictors = setdiff(colnames(prostate), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.glm(y = "VOL", x = predictors, training_frame = prostate, 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
bank = h2o.importFile(
  path="https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv"
)
predictors = 1:20
target = "y"
glm = h2o.glm(x = predictors,
              y = target,
              training_frame = bank,
              family = "binomial",
              standardize = TRUE,
              lambda_search = TRUE)
h2o.std_coef_plot(glm, num_of_features = 20)

## End(Not run)
```

h2o.glrm *Generalized low rank decomposition of an H2O data frame*

Description

Builds a generalized low rank decomposition of an H2O data frame

Usage

```
h2o.glrm(
  training_frame,
  cols = NULL,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  representation_name = NULL,
  loading_name = NULL,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  k = 1,
  loss = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic",
    "Periodic"),
 loss_by_col = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic",
    "Periodic", "Categorical", "Ordinal"),
  loss_by_col_idx = NULL,
  multi_loss = c("Categorical", "Ordinal"),
  period = 1,
 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 = 2000,
  init_step_size = 1,
  min_step_size = 1e-04,
  seed = -1,
  init = c("Random", "SVD", "PlusPlus", "User"),
  svd_method = c("GramSVD", "Power", "Randomized"),
  user_y = NULL,
  user_x = NULL,
  expand_user_y = TRUE,
  impute_original = FALSE,
  recover_svd = FALSE,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.

`cols`	(Optional) A vector containing the data columns on which k-means operates.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`representation_name`	Frame key to save resulting X
`loading_name`	[Deprecated] Use representation_name instead. Frame key to save resulting X.
`transform`	Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
`k`	Rank of matrix approximation Defaults to 1.
`loss`	Numeric loss function Must be one of: "Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic". Defaults to Quadratic.
`loss_by_col`	Loss function by column (override) Must be one of: "Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic", "Categorical", "Ordinal".
`loss_by_col_idx`	Loss function by column index (override)
`multi_loss`	Categorical loss function Must be one of: "Categorical", "Ordinal". Defaults to Categorical.
`period`	Length of period (only used with periodic loss function) Defaults to 1.
`regularization_x`	Regularization function for X matrix Must be one of: "None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex". Defaults to None.
`regularization_y`	Regularization function for Y matrix Must be one of: "None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex". Defaults to None.
`gamma_x`	Regularization weight on X matrix Defaults to 0.
`gamma_y`	Regularization weight on Y matrix Defaults to 0.
`max_iterations`	Maximum number of iterations Defaults to 1000.
`max_updates`	Maximum number of updates, defaults to 2*max_iterations Defaults to 2000.
`init_step_size`	Initial step size Defaults to 1.
`min_step_size`	Minimum step size Defaults to 0.0001.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`init`	Initialization mode Must be one of: "Random", "SVD", "PlusPlus", "User". Defaults to PlusPlus.
`svd_method`	Method for computing SVD during initialization (Caution: Randomized is currently experimental and unstable) Must be one of: "GramSVD", "Power", "Randomized". Defaults to Randomized.

`user_y`	User-specified initial Y
`user_x`	User-specified initial X
`expand_user_y`	`Logical`. Expand categorical columns in user-specified initial Y Defaults to TRUE.
`impute_original`	`Logical`. Reconstruct original training data by reversing transform Defaults to FALSE.
`recover_svd`	`Logical`. Recover singular values and eigenvectors of XY Defaults to FALSE.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`export_checkpoints_dir`	Automatically export generated models to this directory.

Value

an object of class H2ODimReductionModel.

References

M. Udell, C. Horn, R. Zadeh, S. Boyd (2014). Generalized Low Rank Models[https://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[https://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

```
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
h2o.glrm(training_frame = australia, k = 5, loss = "Quadratic", regularization_x = "L1",
         gamma_x = 0.5, gamma_y = 0, max_iterations = 1000)

## End(Not run)
```

`h2o.grep` *Search for matches to an argument pattern*

Description

Searches for matches to argument ‘pattern‘ within each element of a string column.

Usage

```
h2o.grep(
  pattern,
  x,
  ignore.case = FALSE,
  invert = FALSE,
  output.logical = FALSE
)
```

Arguments

`pattern`	A character string containing a regular expression.
`x`	An H2O frame that wraps a single string column.
`ignore.case`	If TRUE case is ignored during matching.
`invert`	Identify elements that do not match the pattern.
`output.logical`	If TRUE returns logical vector of indicators instead of list of matching positions

Details

This function has similar semantics as R's native grep function and it supports a subset of its parameters. Default behavior is to return indices of the elements matching the pattern. Parameter 'output.logical' can be used to return a logical vector indicating if the element matches the pattern (1) or not (0).

Value

H2OFrame holding the matching positions or a logical vector if 'output.logical' is enabled.

Examples

```
## Not run:
library(h2o)
h2o.init()
addresses <- as.h2o(c("2307", "Leghorn St", "Mountain View", "CA", "94043"))
zip_codes <- addresses[h2o.grep("[0-9]{5}", addresses, output.logical = TRUE),]

## End(Not run)
```

`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,
  x,
  y,
  training_frame,
  ...,
  hyper_params = list(),
  is_supervised = NULL,
  do_hyper_params_check = FALSE,
  search_criteria = NULL,
  export_checkpoints_dir = NULL,
  recovery_dir = NULL,
  parallelism = 1
)
```

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.
`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`...`	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`	[Deprecated] It is not possible to override default behaviour. (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 list can include values for: strategy, max_models, max_runtime_secs, stopping_metric, stopping_tolerance, stopping_rounds and seed. The default strategy 'Cartesian' covers the entire space of hyperparameter combinations. If you want to use cartesian grid search, you can leave the search_criteria argument unspecified. Specify the "RandomDiscrete" strategy to get random search of all the combinations of your hyperparameters with three ways of specifying when to

stop the search: max number of models, max time, and metric-based early stopping (e.g., stop if MSE has not improved by 0.0001 over the 5 best models). Examples below: `list(strategy = "RandomDiscrete", max_runtime_secs = 600, max_models =` or `list(strategy = "RandomDiscrete",max_models = 42, max_runtime_secs = 28800)` or `list(strategy = "RandomDiscrete", stopping_metric = "AUTO",stopping_tolerance =` or `list(strategy = "RandomDiscrete", stopping_metric = "misclassification", stoppi`

`export_checkpoints_dir`
: Directory to automatically export grid and its models to.

`recovery_dir`
: When specified the grid and all necessary data (frames, models) will be saved to this directory (use HDFS or other distributed file-system). Should the cluster crash during training, the grid can be reloaded from this directory via `h2o.loadGrid` and training can be resumed

`parallelism`
: Level of Parallelism during grid model building. 1 = sequential building (default). Use the value of 0 for adaptive parallelism - decided by H2O. Any number > 1 sets the exact number of models built in parallel.

Details

Launch grid search with given algorithm and parameters.

Examples

```
## Not run:
library(h2o)
library(jsonlite)
h2o.init()
iris_hf <- as.h2o(iris)
grid <- h2o.grid("gbm", x = c(1:4), y = 5, training_frame = iris_hf,
                 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)})

## End(Not run)
```

`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
`...`	any supported aggregate function. See `Details:` for more help.
`gb.control`	a list of how to handle `NA` values in the dataset as well as how to name output columns. The method is specified using the `rm.method` argument. See `Details:` for more help.

Details

In the case of `na.methods` within `gb.control`, there are three possible settings. `"all"` will include `NA`s in computation of functions. `"rm"` will completely remove all `NA` fields. `"ignore"` will remove `NA`s 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"`.

Note that to specify a list of column names in the `gb.control` list, you must add the `col.names` argument. 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.

Supported functions include `nrow`. This function is required and accepts a string for the name of the generated column. Other supported aggregate functions accept `col` and `na` arguments for specifying columns and the handling of NAs (`"all"`, `"ignore"`, and GroupBy object; `max` calculates the maximum of each column specified in `col` for each group of a GroupBy object; `mean` calculates the mean of each column specified in `col` for each group of a GroupBy object; `min` calculates the minimum of each column specified in `col` for each group of a GroupBy object; `mode` calculates the mode of each column specified in `col` for each group of a GroupBy object; `sd` calculates the standard deviation of each column specified in `col` for each group of a GroupBy object; `ss` calculates the sum of squares of each column specified in `col` for each group of a GroupBy object; `sum` calculates the sum of each column specified in `col` for each group of a GroupBy object; and `var` calculates the variance of each column specified in `col` for each group of a GroupBy object. If an aggregate is provided without a value (for example, as `max` in `sum(col="X1", na="all").mean(col="X5", na="all").max()`), then it is assumed that the aggregation should apply to all columns except the GroupBy columns. However, operations will not be performed on String columns. They will be skipped. Note again that `nrow` is required and cannot be empty.

Value

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

Examples

```
## Not run:
library(h2o)
h2o.init()
df <- h2o.importFile(paste("https://s3.amazonaws.com/h2o-public-test-data",
                           "/smalldata/prostate/prostate.csv",
                           sep=""))
h2o.group_by(data = df, by = "RACE", nrow("VOL"))

## End(Not run)
```

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

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_gsub <- as.h2o("r tutorial")
sub_string <- h2o.gsub("r ", "H2O ", string_to_gsub)

## End(Not run)
```

h2o.h — *Calculates Friedman and Popescu's H statistics, in order to test for the presence of an interaction between specified variables in h2o gbm and xgb models. H varies from 0 to 1. It will have a value of 0 if the model exhibits no interaction between specified variables and a correspondingly larger value for a stronger interaction effect between them. NaN is returned if a computation is spoiled by weak main effects and rounding errors.*

Description

See Jerome H. Friedman and Bogdan E. Popescu, 2008, "Predictive learning via rule ensembles", *Ann. Appl. Stat.* **2**:916-954, http://projecteuclid.org/download/pdfview_1/euclid.aoas/1223908046, s. 8.1.

Usage

```
h2o.h(model, frame, variables)
```

Arguments

model	A trained gradient-boosting model.
frame	A frame that current model has been fitted to.
variables	Variables of the interest.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate.hex <- h2o.importFile(
       "https://s3.amazonaws.com/h2o-public-test-data/smalldata/logreg/prostate.csv",
        destination_frame="prostate.hex"
        )
prostate.hex$CAPSULE <- as.factor(prostate.hex$CAPSULE)
prostate.hex$RACE <- as.factor(prostate.hex$RACE)
prostate.h2o <- h2o.gbm(x = 3:9, y = "CAPSULE", training_frame = prostate.hex,
distribution = "bernoulli", ntrees = 100, max_depth = 5, min_rows = 10, learn_rate = 0.1)
h_val <- h2o.h(prostate.h2o, prostate.hex, c('DPROS','DCAPS'))

## End(Not run)
```

---

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, m = 200L, ...)

## S3 method for class 'H2OFrame'
head(x, n = 6L, m = 200L, ...)

h2o.tail(x, n = 6L, m = 200L, ...)

## S3 method for class 'H2OFrame'
tail(x, n = 6L, m = 200L, ...)
```

Arguments

x	An H2OFrame object.
n	(Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x.
m	(Optional) A single integer. If positive, number of columns in x to return. If negative, all but the m first/last number of columns in x.
...	Ignored.

Value

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

Examples

```
## Not run:
library(h2o)
h2o.init(ip <- "localhost", port = 54321, startH2O = TRUE)
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
# Return the first 10 rows and 6 columns
h2o.head(australia, n = 10L, m = 6L)
# Return the last 10 rows and 6 columns
h2o.tail(australia, n = 10L, m = 6L)

# For Jupyter notebook with an R kernel,
# view all rows of a data frame
options(repr.matrix.max.rows = 600, repr.matrix.max.cols = 200)

## End(Not run)
```

`h2o.HGLMMetrics` *Retrieve HGLM ModelMetrics*

Description

Retrieve HGLM ModelMetrics

Usage

```
h2o.HGLMMetrics(object)
```

Arguments

`object`	an H2OModel object or H2OModelMetrics.

`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).

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.asnumeric(iris["petal_len"])
h2o.hist(iris["petal_len"], breaks = "Sturges", plot = TRUE)

## End(Not run)
```

---

h2o.hit_ratio_table *Retrieve the Hit Ratios*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
iris_split <- h2o.splitFrame(data = iris, ratios = 0.8, seed = 1234)
```

```
train <- iris_split[[1]]
valid <- iris_split[[2]]

iris_xgb <- h2o.xgboost(x = 1:4, y = 5, training_frame = train, validation_frame = valid)
hrt_iris <- h2o.hit_ratio_table(iris_xgb, valid = TRUE)
hrt_iris

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.hour(hdf["ds9"])

## End(Not run)
```

h2o.ice_plot *Plot Individual Conditional Expectation (ICE) for each decile*

Description

Individual Conditional Expectation (ICE) plot gives a graphical depiction of the marginal effect of a variable on the response. ICE plots are similar to partial dependence plots (PDP); PDP shows the average effect of a feature while ICE plot shows the effect for a single instance. This function will plot the effect for each decile. In contrast to the PDP, ICE plots can provide more insight, especially when there is stronger feature interaction. Also, the plot shows the original observation values marked by semi-transparent circle on each ICE line. Please note, that the score of the original observation value may differ from score value of underlying ICE line at original observation point as ICE line is drawn as an interpolation of several points.

Usage

```
h2o.ice_plot(
  model,
  newdata,
  column,
  target = NULL,
  max_levels = 30,
  show_pdp = TRUE,
  binary_response_scale = c("response", "logodds"),
  centered = FALSE,
  grouping_column = NULL,
  output_graphing_data = FALSE,
  nbins = 100,
  show_rug = TRUE,
  ...
)
```

Arguments

`model`	An H2OModel.
`newdata`	An H2OFrame.
`column`	A feature column name to inspect.
`target`	If multinomial, plot PDP just for `target` category. Character string.
`max_levels`	An integer specifying the maximum number of factor levels to show. Defaults to 30.
`show_pdp`	Option to turn on/off PDP line. Defaults to TRUE.
`binary_response_scale`	Option for binary model to display (on the y-axis) the logodds instead of the actual score. Can be one of: "response", "logodds". Defaults to "response".
`centered`	A boolean whether to center curves around 0 at the first valid x value or not. Defaults to FALSE.
`grouping_column`	A feature column name to group the data and provide separate sets of plots by grouping feature values

`output_graphing_data`	A bool whether to output final graphing data to a frame. Defaults to FALSE.
`nbins`	A number of bins used. Defaults to 100.
`show_rug`	Show rug to visualize the density of the column. Defaults to TRUE.
`...`	Custom parameters.

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the individual conditional expectations plot
ice <- h2o.ice_plot(gbm, test, column = "alcohol")
print(ice)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.importFile(path = australia_path)
australia[, 9] <- ifelse(australia[, 3] < 279.9, 1, 0)
summary(australia)

## End(Not run)
```

---

h2o.importFile *Import Files into H2O*

Description

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

Usage

```
h2o.importFile(
  path,
  destination_frame = "",
  parse = TRUE,
  header = NA,
  sep = "",
  col.names = NULL,
  col.types = NULL,
  na.strings = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = NULL,
  partition_by = NULL,
  quotechar = NULL,
  escapechar = ""
)

h2o.importFolder(
  path,
  pattern = "",
  destination_frame = "",
  parse = TRUE,
  header = NA,
  sep = "",
  col.names = NULL,
  col.types = NULL,
  na.strings = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = NULL,
  partition_by = NULL,
  quotechar = NULL,
  escapechar = "\\"
)

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,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  quotechar = NULL,
  escapechar = "\\"
)
```

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.

`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, for details see h2o.parseRaw.
`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.
`decrypt_tool`	(Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.
`skipped_columns`	a list of column indices to be skipped during parsing.
`custom_non_data_line_markers`	(Optional) If a line in imported file starts with any character in given string it will NOT be imported. Empty string means all lines are imported, NULL means that default behaviour for given format will be used
`partition_by`	names of the columns the persisted dataset has been partitioned by.
`quotechar`	A hint for the parser which character to expect as quoting character. None (default) means autodetection.
`escapechar`	(Optional) One ASCII character used to escape other characters.
`pattern`	(Optional) Character string containing a regular expression to match file(s) in the folder.
`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

`h2o.importFile` is a parallelized reader and pulls information from the server from a location specified by the client. The path is a server-side path. This is a fast, scalable, highly optimized way to read data. H2O pulls the data from a data store and initiates the data transfer as a read operation.

Unlike the import function, which is a parallelized reader, `h2o.uploadFile` is a push from the client to the server. The specified path must be a client-side path. This is not scalable and is only intended for smaller data sizes. The client pushes the data from a local filesystem (for example, on your machine where R is running) to H2O. For big-data operations, you don't want the data stored on or flowing through the client.

`h2o.importFolder` imports an entire directory of files. If the given path is relative, then it will be relative to the start location of the H2O instance. The default behavior is to pass-through to the parse phase automatically.

`h2o.importHDFS` is deprecated. Instead, use `h2o.importFile`.

See Also

h2o.import_sql_select, h2o.import_sql_table, h2o.parseRaw

Examples

```
## Not run:
h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
class(prostate)
summary(prostate)

#Import files with a certain regex pattern by utilizing h2o.importFolder()
#In this example we import all .csv files in the directory prostate_folder
prostate_path = system.file("extdata", "prostate_folder", package = "h2o")
prostate_pattern = h2o.importFolder(path = prostate_path, pattern = ".*.csv")
class(prostate_pattern)
summary(prostate_pattern)

## End(Not run)
```

h2o.import_hive_table *Import Hive Table into H2O*

Description

Import Hive table to H2OFrame in memory. Make sure to start H2O with Hive on classpath. Uses hive-site.xml on classpath to connect to Hive. When database is specified as jdbc URL uses Hive JDBC driver to obtain table metadata. then uses direct HDFS access to import data.

Usage

```
h2o.import_hive_table(
  database,
  table,
  partitions = NULL,
  allow_multi_format = FALSE
)
```

Arguments

`database`	Name of Hive database (default database will be used by default), can be also a JDBC URL
`table`	name of Hive table to import
`partitions`	a list of lists of strings - partition key column values of partitions you want to import.
`allow_multi_format`	enable import of partitioned tables with different storage formats used. WARNING: this may fail on out-of-memory for tables with a large number of small partitions.

Details

For example, my_citibike_data = h2o.import_hive_table("default", "citibike20k", partitions = list(c("2017", "01"), c("2017", "02"))) my_citibike_data = h2o.import_hive_table("jdbc:hive2://hive-server:10000/default", "citibike20k", allow_multi_format = TRUE)

`h2o.import_mojo` *Imports a MOJO under given path, creating a Generic model with it.*

Description

Usage example: mojo_model <- h2o.import_mojo(model_file_path = "/path/to/mojo.zip") predictions <- h2o.predict(mojo_model, dataset)

Usage

```
h2o.import_mojo(mojo_file_path, model_id = NULL)
```

Arguments

`mojo_file_path`	Filesystem path to the model imported
`model_id`	Model ID, default is NULL

Value

Returns H2O Generic Model embedding given MOJO model

Examples

```
## Not run:

# Import default Iris dataset as H2O frame
data <- as.h2o(iris)

# Train a very simple GBM model
features <- c("Sepal.Length", "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
original_model <- h2o.gbm(x = features, y = "Species", training_frame = data)

# Download the trained GBM model as MOJO (temporary directory used in this example)
mojo_original_path <- h2o.save_mojo(original_model, path = tempdir())

# Import the MOJO and obtain a Generic model
mojo_model <- h2o.import_mojo(mojo_original_path)

# Perform scoring with the generic model
predictions  <- h2o.predict(mojo_model, data)

## End(Not run)
```

`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_c water.H2OApp' Also see h2o.import_sql_table. Currently supported SQL databases are MySQL, PostgreSQL, MariaDB, Hive, Oracle and Microsoft SQL Server.

Usage

```
h2o.import_sql_select(
  connection_url,
  select_query,
  username,
  password,
  use_temp_table = NULL,
  temp_table_name = NULL,
  optimize = NULL,
  fetch_mode = 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
`use_temp_table`	Whether a temporary table should be created from select_query
`temp_table_name`	Name of temporary table to be created from select_query
`optimize`	(Optional) Optimize import of SQL table for faster imports. Experimental. Default is true.
`fetch_mode`	(Optional) Set to DISTRIBUTED to enable distributed import. Set to SINGLE to force a sequential read from the database Can be used for databases that do not support OFFSET-like clauses in SQL statements.

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 cluster. 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, MariaDB, Hive, Oracle and Microsoft SQL Server.

Usage

```
h2o.import_sql_table(
  connection_url,
  table,
  username,
  password,
  columns = NULL,
  optimize = NULL,
  fetch_mode = 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. Default is true. Ignored - use fetch_mode instead.
`fetch_mode`	(Optional) Set to DISTRIBUTED to enable distributed import. Set to SINGLE to force a sequential read from the database Can be used for databases that do not support OFFSET-like clauses in SQL statements.

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 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

```
## Not run:
 h2o.init()
 iris_hf <- as.h2o(iris)
 iris_hf[sample(nrow(iris_hf), 40), 5] <- NA  # randomly replace 50 values with NA
 # impute with a group by
 iris_hf <- h2o.impute(iris_hf, "Species", "mode", by = c("Sepal.Length", "Sepal.Width"))

## End(Not run)
```

`h2o.infogram` *H2O Infogram*

Description

The infogram is a graphical information-theoretic interpretability tool which allows the user to quickly spot the core, decision-making variables that uniquely and safely drive the response, in supervised classification problems. The infogram can significantly cut down the number of predictors needed to build a model by identifying only the most valuable, admissible features. When protected variables such as race or gender are present in the data, the admissibility of a variable is determined by a safety and relevancy index, and thus serves as a diagnostic tool for fairness. The safety of each feature can be quantified and variables that are unsafe will be considered inadmissible. Models built using only admissible features will naturally be more interpretable, given the reduced feature set. Admissible models are also less susceptible to overfitting and train faster, while providing similar accuracy as models built using all available features.

Usage

```
h2o.infogram(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  seed = -1,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  nfolds = 0,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  offset_column = NULL,
  weights_column = NULL,
  standardize = FALSE,
 distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
    "tweedie", "laplace", "quantile", "huber"),
  plug_values = NULL,
  max_iterations = 0,
  stopping_rounds = 0,
   stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
     "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
     "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = 0,
  custom_metric_func = NULL,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  algorithm = c("AUTO", "deeplearning", "drf", "gbm", "glm", "xgboost"),
  algorithm_params = NULL,
  protected_columns = NULL,
  total_information_threshold = -1,
  net_information_threshold = -1,
  relevance_index_threshold = -1,
  safety_index_threshold = -1,
  data_fraction = 1,
  top_n_features = 50
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.

`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`standardize`	`Logical`. Standardize numeric columns to have zero mean and unit variance. Defaults to FALSE.
`distribution`	Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.
`plug_values`	Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues).
`max_iterations`	Maximum number of iterations. Defaults 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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
`balance_classes`	`Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
`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). Requires balance_classes. Defaults to 5.0.

max_runtime_secs	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
custom_metric_func	Reference to custom evaluation function, format: ‘language:keyName=funcName‘
auc_type	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
algorithm	Type of machine learning algorithm used to build the infogram. Options include ’AUTO’ (gbm), ’deeplearning’ (Deep Learning with default parameters), ’drf’ (Random Forest with default parameters), ’gbm’ (GBM with default parameters), ’glm’ (GLM with default parameters), or ’xgboost’ (if available, XGBoost with default parameters). Must be one of: "AUTO", "deeplearning", "drf", "gbm", "glm", "xgboost". Defaults to AUTO.
algorithm_params	Customized parameters for the machine learning algorithm specified in the algorithm parameter.
protected_columns	Columns that contain features that are sensitive and need to be protected (legally, or otherwise), if applicable. These features (e.g. race, gender, etc) should not drive the prediction of the response.
total_information_threshold	A number between 0 and 1 representing a threshold for total information, defaulting to 0.1. For a specific feature, if the total information is higher than this threshold, and the corresponding net information is also higher than the threshold “net_information_threshold“, that feature will be considered admissible. The total information is the x-axis of the Core Infogram. Default is -1 which gets set to 0.1. Defaults to -1.
net_information_threshold	A number between 0 and 1 representing a threshold for net information, defaulting to 0.1. For a specific feature, if the net information is higher than this threshold, and the corresponding total information is also higher than the total_information_threshold, that feature will be considered admissible. The net information is the y-axis of the Core Infogram. Default is -1 which gets set to 0.1. Defaults to -1.
relevance_index_threshold	A number between 0 and 1 representing a threshold for the relevance index, defaulting to 0.1. This is only used when “protected_columns“ is set by the user. For a specific feature, if the relevance index value is higher than this threshold, and the corresponding safety index is also higher than the safety_index_threshold“, that feature will be considered admissible. The relevance index is the x-axis of the Fair Infogram. Default is -1 which gets set to 0.1. Defaults to -1.
safety_index_threshold	A number between 0 and 1 representing a threshold for the safety index, defaulting to 0.1. This is only used when protected_columns is set by the user. For a specific feature, if the safety index value is higher than this threshold, and the corresponding relevance index is also higher than the relevance_index_threshold, that feature will be considered admissible. The safety index is the y-axis of the Fair Infogram. Default is -1 which gets set to 0.1. Defaults to -1.
data_fraction	The fraction of training frame to use to build the infogram model. Defaults to 1.0, and any value greater than 0 and less than or equal to 1.0 is acceptable. Defaults to 1.

`top_n_features`	An integer specifying the number of columns to evaluate in the infogram. The columns are ranked by variable importance, and the top N are evaluated. Defaults to 50. Defaults to 50.

Details

The infogram allows the user to quickly spot the admissible decision-making variables that are driving the response. There are two types of infogram plots: Core and Fair Infogram.

The Core Infogram plots all the variables as points on two-dimensional grid of total vs net information. The x-axis is total information, a measure of how much the variable drives the response (the more predictive, the higher the total information). The y-axis is net information, a measure of how unique the variable is. The top right quadrant of the infogram plot is the admissible section; the variables located in this quadrant are the admissible features. In the Core Infogram, the admissible features are the strongest, unique drivers of the response.

If sensitive or protected variables are present in data, the user can specify which attributes should be protected while training using the `protected_columns` argument. All non-protected predictor variables will be checked to make sure that there's no information pathway to the response through a protected feature, and deemed inadmissible if they possess little or no informational value beyond their use as a dummy for protected attributes. The Fair Infogram plots all the features as points on two-dimensional grid of relevance vs safety. The x-axis is relevance index, a measure of how much the variable drives the response (the more predictive, the higher the relevance). The y-axis is safety index, a measure of how much extra information the variable has that is not acquired through the protected variables. In the Fair Infogram, the admissible features are the strongest, safest drivers of the response.

Examples

```
## Not run:
h2o.init()

# Convert iris dataset to an H2OFrame
df <- as.h2o(iris)

# Infogram
ig <- h2o.infogram(y = "Species", training_frame = df)
plot(ig)


## End(Not run)
```

`h2o.infogram_train_subset_models`
Train models over subsets selected using infogram

Description

Train models over subsets selected using infogram

Usage

```
h2o.infogram_train_subset_models(
  ig,
  model_fun,
  training_frame,
  test_frame,
  y,
  protected_columns,
  reference,
  favorable_class,
  feature_selection_metrics = c("safety_index"),
  metric = "euclidean",
  air_metric = "selectedRatio",
  alpha = 0.05,
  ...
)
```

Arguments

`ig`	Infogram object trained with the same protected columns
`model_fun`	Function that creates models. This can be something like h2o.automl, h2o.gbm, etc.
`training_frame`	Training frame
`test_frame`	Test frame
`y`	Response column
`protected_columns`	Protected columns
`reference`	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
`favorable_class`	Positive/favorable outcome class of the response.
`feature_selection_metrics`	One or more columns from the infogram@admissible_score.
`metric`	Metric supported by stats::dist which is used to sort the features.
`air_metric`	Metric used for Adverse Impact Ratio calculation. Defaults to “selectedRatio“.
`alpha`	The alpha level is the probability of rejecting the null hypothesis that the protected group and the reference came from the same population when the null hypothesis is true.
`...`	Parameters that are passed to the model_fun.

Value

frame containing aggregations of intersectional fairness across the models

Examples

```
## Not run:
library(h2o)
h2o.connect()
```

```
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
      'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
       'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

ig <- h2o.infogram(x, y, train, protected_columns = protected_columns)
print(ig@admissible_score)
plot(ig)

infogram_models <- h2o.infogram_train_subset_models(ig, h2o.gbm, train, test, y,
                                                    protected_columns, reference,
                                                    favorable_class)

pf <- h2o.pareto_front(infogram_models, x_metric = "air_min",
                       y_metric = "AUC", optimum = "top right")
plot(pf)
pf@pareto_front

## End(Not run)
```

---

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,
  name = NA_character_,
  startH2O = TRUE,
  forceDL = FALSE,
  enable_assertions = TRUE,
  license = NULL,
  nthreads = -1,
  max_mem_size = NULL,
  min_mem_size = NULL,
  ice_root = tempdir(),
  log_dir = NA_character_,
  log_level = NA_character_,
  strict_version_check = TRUE,
  proxy = NA_character_,
  https = FALSE,
  cacert = NA_character_,
  insecure = FALSE,
  username = NA_character_,
  password = NA_character_,
  use_spnego = FALSE,
  cookies = NA_character_,
  context_path = NA_character_,
  ignore_config = FALSE,
  extra_classpath = NULL,
  jvm_custom_args = NULL,
  bind_to_localhost = TRUE
)
```

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.
`name`	(Optional) A `character` string representing the H2O cluster name.
`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. -1 means use all CPUs on the host (Default). 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. If max_mem_size is not defined, then the amount of memory that H2O allocates will be determined by the default memory of Java Virtual Machine. This amount is dependent on the Java version, but it will generally be 25 percent of the machine's physical memory.

`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.
`log_dir`	(Optional) A directory where H2O server logs are stored. The default varies by OS.
`log_level`	(Optional) The level of logging of H2O server. The default is INFO.
`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.
`cacert`	(Optional) Path to a CA bundle file with root and intermediate certificates of trusted CAs.
`insecure`	(Optional) Set this to TRUE to disable SSL certificate checking.
`username`	(Optional) Username to login with.
`password`	(Optional) Password to login with.
`use_spnego`	(Optional) Set this to TRUE to enable SPNEGO authentication.
`cookies`	(Optional) Vector(or list) of cookies to add to request.
`context_path`	(Optional) The last part of connection URL: http://<ip>:<port>/<context_path>
`ignore_config`	(Optional) A `logical` value indicating whether a search for a .h2oconfig file should be conducted or not. Default value is FALSE.
`extra_classpath`	(Optional) A vector of paths to libraries to be added to the Java classpath when H2O is started from R.
`jvm_custom_args`	(Optional) A `character` list of custom arguments for the JVM where new H2O instance is going to run, if started. Ignored when connecting to an existing instance.
`bind_to_localhost`	(Optional) A `logical` flag indicating whether access to the H2O instance should be restricted to the local machine (default) or if it can be reached from other computers on the network. Only applicable when H2O is started from R.

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. If an open ip and port of your choice are passed in, then this method will attempt to start an H2O instance at that specified ip port.

When initializing H2O locally, this method searches for h2o.jar in the R library resources (`system.file("java", "h2o.` 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

Insert Missing Values into an H2OFrame

Description

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.

Value

Returns an H2OFrame object.

WARNING

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

Examples

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
summary(iris_hf)

iris_miss <- h2o.insertMissingValues(iris_hf, fraction = 0.25)
head(iris_miss)
summary(iris_miss)

## End(Not run)
```

`h2o.inspect_model_fairness`

Produce plots and dataframes related to a single model fairness.

Description

Produce plots and dataframes related to a single model fairness.

Usage

```
h2o.inspect_model_fairness(
  model,
  newdata,
  protected_columns,
  reference,
  favorable_class,
  metrics = c("auc", "aucpr", "f1", "p.value", "selectedRatio", "total")
)
```

Arguments

`model`	H2O Model Object
`newdata`	H2OFrame
`protected_columns`	List of categorical columns that contain sensitive information such as race, gender, age etc.
`reference`	List of values corresponding to a reference for each protected columns. If set to NULL, it will use the biggest group as the reference.
`favorable_class`	Positive/favorable outcome class of the response.
`metrics`	Character vector of metrics to show.

Value

H2OExplanation object

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(paste0("https://s3.amazonaws.com/h2o-public-test-data/smalldata/",
                              "admissibleml_test/taiwan_credit_card_uci.csv"))
x <- c('LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
      'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
       'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6')
y <- "default payment next month"
protected_columns <- c('SEX', 'EDUCATION')

for (col in c(y, protected_columns))
  data[[col]] <- as.factor(data[[col]])

splits <- h2o.splitFrame(data, 0.8)
train <- splits[[1]]
test <- splits[[2]]
reference <- c(SEX = "1", EDUCATION = "2")  # university educated man
favorable_class <- "0" # no default next month

gbm <- h2o.gbm(x, y, training_frame = train)

h2o.inspect_model_fairness(gbm, test, protected_columns = protected_columns,
                           reference = reference, favorable_class = favorable_class)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()

# Create some random data
my_frame <- 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
my_frame[,5] <- as.factor(my_frame[,5])
head(my_frame, 20)

# Create pairwise interactions
pairwise <- h2o.interaction(my_frame,
                             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(my_frame, 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(my_frame[,5], 20)
trim_integer_levels <- h2o.interaction(my_frame, factors = "C5", pairwise = FALSE, max_factors = 3,
                                       min_occurrence = 2)
head(trim_integer_levels, 20)

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

## End(Not run)
```

h2o.isax *iSAX*

Description

Compute the iSAX index for a DataFrame which is assumed to be numeric time series data

Usage

```
h2o.isax(x, num_words, max_cardinality, optimize_card = FALSE)
```

Arguments

`x`	an H2OFrame
`num_words`	Number of iSAX words for the timeseries. ie granularity along the time series
`max_cardinality`	Maximum cardinality of the iSAX word. Each word can have less than the max
`optimize_card`	An optimization flag that will find the max cardinality regardless of what is passed in for max_cardinality.

Value

An H2OFrame with the name of time series, string representation of iSAX word, followed by binary representation

References

https://www.cs.ucr.edu/~eamonn/iSAX_2.0.pdf

https://www.cs.ucr.edu/~eamonn/SAX.pdf

Examples

```
## Not run:
library(h2o)
h2o.init()
df <- h2o.createFrame(rows = 1, cols = 256, randomize = TRUE, value = 0,
                      real_range = 100, categorical_fraction = 0, factors = 0,
                      integer_fraction = 0, integer_range = 100, binary_fraction = 0,
                      binary_ones_fraction = 0, time_fraction = 0, string_fraction = 0,
                      missing_fraction = 0, has_response = FALSE, seed = 123)
df2 <- h2o.cumsum(df, axis = 1)
h2o.isax(df2, num_words = 10, max_cardinality = 10)

## End(Not run)
```

`h2o.ischaracter` *Check if character*

Description

Check if character

Usage

```
h2o.ischaracter(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`character` for the base R implementation, `is.character()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
iris_char <- h2o.ascharacter(iris["class"])
h2o.ischaracter(iris_char)

## End(Not run)
```

h2o.isfactor *Check if factor*

Description

Check if factor

Usage

```
h2o.isfactor(x)
```

Arguments

x	An H2OFrame object.

See Also

`factor` for the base R implementation, `is.factor()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
h2o.isfactor(cars["economy_20mpg"])

## End(Not run)
```

h2o.isnumeric *Check if numeric*

Description

Check if numeric

Usage

```
h2o.isnumeric(x)
```

Arguments

x	An H2OFrame object.

See Also

`numeric` for the base R implementation, `is.numeric()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
h2o.isnumeric(iris["sepal_len"])

## End(Not run)
```

`h2o.isolationForest` *Trains an Isolation Forest model*

Description

Trains an Isolation Forest model

Usage

```
h2o.isolationForest(
  training_frame,
  x,
  model_id = NULL,
  score_each_iteration = FALSE,
  score_tree_interval = 0,
  ignore_const_cols = TRUE,
  ntrees = 50,
  max_depth = 8,
  min_rows = 1,
  max_runtime_secs = 0,
  seed = -1,
  build_tree_one_node = FALSE,
  mtries = -1,
  sample_size = 256,
  sample_rate = -1,
  col_sample_rate_change_per_level = 1,
  col_sample_rate_per_tree = 1,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "anomaly_score"),
  stopping_tolerance = 0.01,
  export_checkpoints_dir = NULL,
  contamination = -1,
  validation_frame = NULL,
  validation_response_column = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`model_id`	Destination id for this model; auto-generated if not specified.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`score_tree_interval`	Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`ntrees`	Number of trees. Defaults to 50.
`max_depth`	Maximum tree depth (0 for unlimited). Defaults to 8.
`min_rows`	Fewest allowed (weighted) observations in a leaf. Defaults to 1.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`build_tree_one_node`	`Logical`. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
`mtries`	Number of variables randomly sampled as candidates at each split. If set to -1, defaults (number of predictors)/3. Defaults to -1.
`sample_size`	Number of randomly sampled observations used to train each Isolation Forest tree. Only one of parameters sample_size and sample_rate should be defined. If sample_rate is defined, sample_size will be ignored. Defaults to 256.
`sample_rate`	Rate of randomly sampled observations used to train each Isolation Forest tree. Needs to be in range from 0.0 to 1.0. If set to -1, sample_rate is disabled and sample_size will be used instead. Defaults to -1.
`col_sample_rate_change_per_level`	Relative change of the column sampling rate for every level (must be > 0.0 and <= 2.0) Defaults to 1.
`col_sample_rate_per_tree`	Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "anomaly_score". Defaults to AUTO.

`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.01.
`export_checkpoints_dir`	Automatically export generated models to this directory.
`contamination`	Contamination ratio - the proportion of anomalies in the input dataset. If undefined (-1) the predict function will not mark observations as anomalies and only anomaly score will be returned. Defaults to -1 (undefined). Defaults to -1.
`validation_frame`	Id of the validation data frame.
`validation_response_column`	(experimental) Name of the response column in the validation frame. Response column should be binary and indicate not anomaly/anomaly.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the cars dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)

# Set the predictors
predictors <- c("displacement", "power", "weight", "acceleration", "year")

# Train the IF model
cars_if <- h2o.isolationForest(x = predictors, training_frame = cars,
                               seed = 1234, stopping_metric = "anomaly_score",
                               stopping_rounds = 3, stopping_tolerance = 0.1)

## End(Not run)
```

`h2o.isotonicregression` *Build an Isotonic Regression model*

Description

Builds an Isotonic Regression model on an H2OFrame with a single feature (univariate regression).

Usage

```
h2o.isotonicregression(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  weights_column = NULL,
  out_of_bounds = c("NA", "clip"),
  custom_metric_func = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
y	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
training_frame	Id of the training data frame.
model_id	Destination id for this model; auto-generated if not specified.
validation_frame	Id of the validation data frame.
weights_column	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
out_of_bounds	Method of handling values of X predictor that are outside of the bounds seen in training. Must be one of: "NA", "clip". Defaults to NA.
custom_metric_func	Reference to custom evaluation function, format: 'language:keyName=funcName'
nfolds	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
keep_cross_validation_models	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
fold_assignment	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
fold_column	Column with cross-validation fold index assignment per observation.

Value

Creates a H2OModel object of the right type.

See Also

`predict.H2OModel` for prediction

Examples

```
## Not run:
library(h2o)
h2o.init()

N <- 100
x <- seq(N)
y <- sample(-50:50, N, replace=TRUE) + 50 * log1p(x)

train <- as.h2o(data.frame(x = x, y = y))
isotonic <- h2o.isotonicregression(x = "x", y = "y", training_frame = train)

## End(Not run)
```

`h2o.is_client` *Check Client Mode Connection*

Description

Check Client Mode Connection

Usage

```
h2o.is_client()
```

`h2o.keyof` *Method on* `Keyed` *objects allowing to obtain their key.*

Description

Method on `Keyed` objects allowing to obtain their key.

Usage

```
h2o.keyof(object)

## S4 method for signature 'Keyed'
h2o.keyof(object)

## S4 method for signature 'H2OModel'
h2o.keyof(object)
```

```
## S4 method for signature 'H2OGrid'
h2o.keyof(object)

## S4 method for signature 'H2OFrame'
h2o.keyof(object)

## S4 method for signature 'H2OAutoML'
h2o.keyof(object)
```

Arguments

`object`	A Keyed object

Value

the string key holding the persistent object.

`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.

Value

Returns an H2OFrame object with fold assignments.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
kfolds <- h2o.kfold_column(iris, nfolds = 5, seed = 1234)

## End(Not run)
```

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

Description

A poor man's profiler, but effective.

Usage

```
h2o.killMinus3()
```

`h2o.kmeans` *Performs k-means clustering on an H2O dataset*

Description

Performs k-means clustering on an H2O dataset

Usage

```
h2o.kmeans(
  training_frame,
  x,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  k = 1,
  estimate_k = FALSE,
  user_points = NULL,
  max_iterations = 10,
  standardize = TRUE,
  seed = -1,
  init = c("Random", "PlusPlus", "Furthest", "User"),
  max_runtime_secs = 0,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  export_checkpoints_dir = NULL,
  cluster_size_constraints = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`k`	The max. number of clusters. If estimate_k is disabled, the model will find k centroids, otherwise it will find up to k centroids. Defaults to 1.
`estimate_k`	`Logical`. Whether to estimate the number of clusters (<=k) iteratively and deterministically. Defaults to FALSE.
`user_points`	This option allows you to specify a dataframe, where each row represents an initial cluster center. The user- specified points must have the same number of columns as the training observations. The number of rows must equal the number of clusters
`max_iterations`	Maximum training iterations (if estimate_k is enabled, then this is for each inner Lloyds iteration) Defaults to 10.
`standardize`	`Logical`. Standardize columns before computing distances Defaults to TRUE.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`init`	Initialization mode Must be one of: "Random", "PlusPlus", "Furthest", "User". Defaults to Furthest.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.

`categorical_encoding`
: Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.

`export_checkpoints_dir`
: Automatically export generated models to this directory.

`cluster_size_constraints`
: An array specifying the minimum number of points that should be in each cluster. The length of the constraints array has to be the same as the number of clusters.

Value

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.kmeans(training_frame = prostate, k = 10, x = c("AGE", "RACE", "VOL", "GLEASON"))

## End(Not run)
```

`h2o.kolmogorov_smirnov` *Kolmogorov-Smirnov metric for binomial models*

Description

Retrieves a Kolmogorov-Smirnov metric for given binomial model. The number returned is in range between 0 and 1. K-S metric represents the degree of separation between the positive (1) and negative (0) cumulative distribution functions. Detailed metrics per each group are to be found in the gains-lift table.

Usage

```
h2o.kolmogorov_smirnov(object)

## S4 method for signature 'H2OModelMetrics'
h2o.kolmogorov_smirnov(object)

## S4 method for signature 'H2OModel'
h2o.kolmogorov_smirnov(object)
```

Arguments

`object`	Either an H2OModel object or an H2OModelMetrics object.

Details

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

Value

Kolmogorov-Smirnov metric, a number between 0 and 1.

See Also

`h2o.gainsLift` to see detailed K-S metrics per group

Examples

```
## Not run:
library(h2o)
h2o.init()
data <- h2o.importFile(
path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip")
model <- h2o.gbm(x = c("Origin", "Distance"), y = "IsDepDelayed",
                       training_frame = data, ntrees = 1)
h2o.kolmogorov_smirnov(model)

## End(Not run)
```

`h2o.kurtosis` *Kurtosis of a column*

Description

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

Usage

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

kurtosis.H2OFrame(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.

Value

Returns a list containing the kurtosis for each column (NaN for non-numeric columns).

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.kurtosis(prostate$AGE)

## End(Not run)
```

`h2o.learning_curve_plot` *Learning Curve Plot*

Description

Create learning curve plot for an H2O Model. Learning curves show error metric dependence on learning progress, e.g., RMSE vs number of trees trained so far in GBM. There can be up to 4 curves showing Training, Validation, Training on CV Models, and Cross-validation error.

Usage

```
h2o.learning_curve_plot(
  model,
  metric = c("AUTO", "auc", "aucpr", "mae", "rmse", "anomaly_score", "convergence",
    "custom", "custom_increasing", "deviance", "lift_top_group", "logloss",
   "misclassification", "negative_log_likelihood", "objective", "sumetaieta02"),
  cv_ribbon = NULL,
  cv_lines = NULL
)
```

Arguments

`model`	an H2O model
`metric`	Metric to be used for the learning curve plot. These should mostly correspond with stopping metric.
`cv_ribbon`	if True, plot the CV mean as a and CV standard deviation as a ribbon around the mean, if NULL, it will attempt to automatically determine if this is suitable visualisation
`cv_lines`	if True, plot scoring history for individual CV models, if NULL, it will attempt to automatically determine if this is suitable visualisation

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the learning curve plot
learning_curve <- h2o.learning_curve_plot(gbm)
print(learning_curve)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
```

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

## End(Not run)
```

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

Description

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

Usage

```
h2o.listTimezones()
```

`h2o.list_all_extensions` *List all H2O registered extensions*

Description

List all H2O registered extensions

Usage

```
h2o.list_all_extensions()
```

`h2o.list_api_extensions` *List registered API extensions*

Description

List registered API extensions

Usage

```
h2o.list_api_extensions()
```

h2o.list_core_extensions
List registered core extensions

Description

List registered core extensions

Usage

```
h2o.list_core_extensions()
```

h2o.list_jobs
Return list of jobs performed by the H2O cluster

Description

Return list of jobs performed by the H2O cluster

Usage

```
h2o.list_jobs()
```

h2o.list_models
Get an list of all model ids present in the cluster

Description

Get an list of all model ids present in the cluster

Usage

```
h2o.list_models()
```

Value

Returns a vector of model ids.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
model_id <- h2o.gbm(x = 1:4, y = 5, training_frame = iris_hf)@model_id
model_id_list <- h2o.list_models()

## End(Not run)
```

h2o.loadGrid *Loads previously saved grid with all it's models from the same folder*

Description

Returns a reference to the loaded Grid.

Usage

```
h2o.loadGrid(grid_path, load_params_references = FALSE)
```

Arguments

grid_path	A character string containing the path to the file with the grid saved.
load_params_references	A logical which if true will attemt to reload saved objects referenced by grid parameters (e.g. training frame, calibration frame), will fail if grid was saved without referenced objects.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris <- as.h2o(iris)

ntrees_opts = c(1, 5)
learn_rate_opts = c(0.1, 0.01)
size_of_hyper_space = length(ntrees_opts) * length(learn_rate_opts)

hyper_parameters = list(ntrees = ntrees_opts, learn_rate = learn_rate_opts)
# Tempdir is chosen arbitrarily. May be any valid folder on an H2O-supported filesystem.
baseline_grid <- h2o.grid("gbm", grid_id="gbm_grid_test", x=1:4, y=5, training_frame=iris,
hyper_params = hyper_parameters, export_checkpoints_dir = tempdir())
# Remove everything from the cluster or restart it
h2o.removeAll()
grid <- h2o.loadGrid(paste0(tempdir(),"/",baseline_grid@grid_id))

## End(Not run)
```

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

Description

Load a saved H2O model from disk. (Note that ensemble binary models can now be loaded using this method.)

Usage

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

Arguments

`path`	The path of the H2O Model to be imported.

Value

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

See Also

`h2o.saveModel`, `H2OModel`

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
# prostate = h2o.importFile(path = prostate_path)
# prostate_glm = h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#   training_frame = prostate, 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.load_frame` *Load frame previously stored in H2O's native format.*

Description

Load frame previously stored in H2O's native format.

Usage

```
h2o.load_frame(frame_id, dir, force = TRUE)
```

Arguments

`frame_id`	the frame ID of the original frame
`dir`	a filesystem location where to look for frame data
`force`	`logical`. overwrite an already existing frame (defaults to true)

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
h2o.save_frame(prostate, "/tmp/prostate")
prostate.key <- h2o.getId(prostate)
```

```
h2o.rm(prostate)
prostate <- h2o.load_frame(prostate.key, "/tmp/prostate")

## End(Not run)
```

`h2o.log` *Compute the logarithm of x*

Description

Compute the logarithm of x

Usage

```
h2o.log(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`Log` for the base R implementation, `log`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.log(frame)

## End(Not run)
```

`h2o.log10` *Compute the log10 of x*

Description

Compute the log10 of x

Usage

```
h2o.log10(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`Log` for the base R implementation, `log10()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.log10(frame)

## End(Not run)
```

`h2o.log1p` *Compute the log1p of x*

Description

Compute the log1p of x

Usage

```
h2o.log1p(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`Log` for the base R implementation, `log1p()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.log1p(frame)

## End(Not run)
```

`h2o.log2` *Compute the log2 of x*

Description

Compute the log2 of x

Usage

```
h2o.log2(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`Log` for the base R implementation, `log2()`

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.log2(frame)

## 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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_splits <- h2o.splitFrame(data =  cars, ratios = .8, seed = 1234)
train <- cars_splits[[1]]
valid <- cars_splits[[2]]
car_drf <- h2o.randomForest(x = predictors,
                            y = response,
                            training_frame = train,
                            validation_frame = valid)
h2o.logloss(car_drf, train = TRUE, valid = TRUE)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.ls()

## End(Not run)
```

`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

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_lstrip <- as.h2o("1234567890")
lstrip_string <- h2o.lstrip(string_to_lstrip, "123") #Remove "123"

## End(Not run)
```

h2o.mae — *Retrieve the Mean Absolute Error Value*

Description

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

Usage

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

Arguments

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

Examples

```
## Not run:
library(h2o)

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

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

h2o.mae(m)

## End(Not run)
```

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.make_leaderboard *Create a leaderboard from a list of models, grids and/or automls.*

Description

Create a leaderboard from a list of models, grids and/or automls.

Usage

```
h2o.make_leaderboard(
  object,
  leaderboard_frame,
  sort_metric = "AUTO",
  extra_columns = c(),
  scoring_data = c("AUTO", "train", "valid", "xval")
)
```

Arguments

object	List of models, automls, or grids; or just single automl/grid object.
leaderboard_frame	Frame used for generating the metrics (optional).
sort_metric	Metric used for sorting the leaderboard.
extra_columns	What extra columns should be calculated (might require leaderboard_frame). Use "ALL" for all available or list of extra columns.
scoring_data	Metrics to be reported in the leaderboard ("xval", "train", or "valid"). Used if no leaderboard_frame is provided.

Value

data.frame

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
grid <- h2o.grid("gbm", x = c(1:4), y = 5, training_frame = iris_hf,
                 hyper_params = list(ntrees = c(1, 2, 3)))
h2o.make_leaderboard(grid, iris_hf)

## End(Not run)
```

h2o.make_metrics *Create Model Metrics from predicted and actual values in H2O*

Description

Given predicted values (target for regression, class-1 probabilities or binomial or per-class probabilities for multinomial), compute a model metrics object

Usage

```
h2o.make_metrics(
  predicted,
  actuals,
  domain = NULL,
  distribution = NULL,
  weights = NULL,
  treatment = NULL,
  auc_type = "NONE",
  auuc_type = "AUTO",
  auuc_nbins = -1
)
```

Arguments

`predicted`	An H2OFrame containing predictions
`actuals`	An H2OFrame containing actual values
`domain`	Vector with response factors for classification.
`distribution`	Distribution for regression.
`weights`	(optional) An H2OFrame containing observation weights.
`treatment`	(optional, for uplift models only) An H2OFrame containing treatment column for uplift classification.
`auc_type`	(optional) For multinomial classification you have to specify which type of agregated AUC/AUCPR will be used to calculate this metric.
`auuc_type`	(optional) For uplift binomial classification you have to specify which type of AUUC will be used to calculate this metric. Possibilities are gini, lift, gain, AUTO. Default is AUTO which means qini.
`auuc_nbins`	(optional) For uplift binomial classification you have to specify number of bins to be used for calculation the AUUC. Default is -1, which means 1000.

Value

Returns an object of the H2OModelMetrics subclass.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
```

```
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
pred <- h2o.predict(prostate_gbm, prostate)[, 3] ## class-1 probability
h2o.make_metrics(pred, prostate$CAPSULE)

## End(Not run)
```

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.

Value

Returns a vector of the positions of (first) matches of its first argument in its second

See Also

`match` for base R implementation.

Examples

```
## Not run:
h2o.init()
iris_hf <- as.h2o(iris)
h2o.match(iris_hf[, 5], c("setosa", "versicolor"))

## End(Not run)
```

`h2o.max` *Returns the maxima of the input values.*

Description

Returns the maxima of the input values.

Usage

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

Arguments

`x`	An H2OFrame object.
`na.rm`	`logical`. indicating whether missing values should be removed.

See Also

`Extremes` for the base R implementation, `max()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.max(iris["petal_len"], na.rm = TRUE)

## End(Not run)
```

`h2o.mean` *Compute the frame's mean by-column (or by-row).*

Description

Compute the frame's mean by-column (or by-row).

Usage

```
h2o.mean(x, na.rm = FALSE, axis = 0, return_frame = FALSE, ...)

## S3 method for class 'H2OFrame'
mean(x, na.rm = FALSE, axis = 0, return_frame = FALSE, ...)
```

Arguments

x	An H2OFrame object.
na.rm	logical. Indicate whether missing values should be removed.
axis	integer. Indicate whether to calculate the mean down a column (0) or across a row (1). NOTE: This is only applied when return_frame is set to TRUE. Otherwise, this parameter is ignored.
return_frame	logical. Indicate whether to return an H2O frame or a list. Default is FALSE (returns a list).
...	Further arguments to be passed from or to other methods.

Value

Returns a list containing the mean for each column (NaN for non-numeric columns) if return_frame is set to FALSE. If return_frame is set to TRUE, then it will return an H2O frame with means per column or row (depends on axis argument).

See Also

Round for base R implementation, `mean()` and colSums for the base R implementation, `colMeans()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
# Default behavior. Will return list of means per column.
h2o.mean(prostate$AGE)
# return_frame set to TRUE. This will return an H2O Frame
# with mean per row or column (depends on axis argument)
h2o.mean(prostate, na.rm = TRUE, axis = 1, return_frame = TRUE)

## End(Not run)
```

`h2o.mean_per_class_error` *Retrieve the mean per class error*

Description

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

Usage

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

Arguments

`object`	An H2OBinomialMetrics object.
`train`	Retrieve the training mean per class error
`valid`	Retrieve the validation mean per class error
`xval`	Retrieve the cross-validation mean per class error

See Also

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

Examples

```
## Not run:
library(h2o)
h2o.init()

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

prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
perf <- h2o.performance(model, prostate)
h2o.mean_per_class_error(perf)
h2o.mean_per_class_error(model, train=TRUE)

## End(Not run)
```

---

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

```
## Not run:
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)

## End(Not run)
```

---

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.

Value

Returns a list containing the median for each column (NaN for non-numeric columns)

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.median(prostate)

## End(Not run)
```

h2o.melt — *Converts a frame to key-value representation while optionally skipping NA values. Inverse operation to h2o.pivot.*

Description

Pivot the frame designated by the three columns: index, column, and value. Index and column should be of type enum, int, or time. For cases of multiple indexes for a column label, the aggregation method is to pick the first occurrence in the data frame

Usage

```
h2o.melt(
  x,
  id_vars,
  value_vars = NULL,
  var_name = "variable",
  value_name = "value",
  skipna = FALSE
)
```

Arguments

`x`	an H2OFrame
`id_vars`	the columns used as identifiers
`value_vars`	what columns will be converted to key-value pairs (optional, if not specified complement to id_vars will be used)
`var_name`	name of the key-column (default: "variable")
`value_name`	name of the value-column (default: "value")
`skipna`	if enabled, do not include NAs in the result (default: FALSE)

Value

an unpivoted H2OFrame

h2o.merge — *Merge Two H2O Data Frames*

Description

Merges two H2OFrame objects with the same arguments and meanings as merge() in base R. However, we do not support all=TRUE, all.x=TRUE and all.y=TRUE. The default method is auto and it will default to the radix method. The radix method will return the correct merge result regardless of duplicated rows in the right frame. In addition, the radix method can perform merge even if you have string columns in your frames. If there are duplicated rows in your rite frame, they will not be included if you use the hash method. The hash method cannot perform merge if you have string columns in your left frame. Hence, we consider the radix method superior to the hash method and is the default method to use.

Usage

```
h2o.merge(
  x,
  y,
  by = intersect(names(x), names(y)),
  by.x = by,
  by.y = by,
  all = FALSE,
  all.x = all,
  all.y = all,
  method = "auto"
)
```

Arguments

`x, y`	H2OFrame objects
`by`	columns used for merging by default the common names
`by.x`	x columns used for merging by name or number
`by.y`	y columns used for merging by name or number
`all`	TRUE includes all rows in x and all rows in y even if there is no match to the other
`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
`method`	auto(default), radix, hash

Examples

```
## Not run:
library(h2o)
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))
left_hf <- as.h2o(left)
right_hf <- as.h2o(right)
merged <- h2o.merge(left_hf, right_hf, all.x = TRUE)

## End(Not run)
```

`h2o.metric` *H2O Model Metric Accessor Functions*

Description

A series of functions that retrieve model metric details.

Usage

```
h2o.metric(object, thresholds, metric, transform = NULL)

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.mean_per_class_accuracy(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. If not set, then all thresholds will be returned. If "max", then the threshold maximizing the metric will be used.
`metric`	(Optional) the metric to retrieve. If not set, then all metrics will be returned.
`transform`	(Optional) a list describing a transformer for the given metric, if any. e.g. transform=list(op=foo_fn, name="foo") will rename the given metric to "foo" and apply function foo_fn to the metric values.

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

```
## Not run:
library(h2o)
h2o.init()

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

prostate$CAPSULE <- as.factor(prostate$CAPSULE)
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
perf <- h2o.performance(model, prostate)
h2o.F1(perf)

## End(Not run)
```

`h2o.min` *Returns the minima of the input values.*

Description

Returns the minima of the input values.

Usage

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

Arguments

x	An H2OFrame object.
na.rm	`logical`. indicating whether missing values should be removed.

See Also

`Extremes` for the base R implementation, `min()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.min(iris["sepal_len"], na.rm = TRUE)

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

x = as.h2o(c(2018, 3, 2, 6, 32, 0, 0))
h2o.mktime(x)

## End(Not run)
```

`h2o.modelSelection` *H2O ModelSelection is used to build the best model with one predictor, two predictors, ... up to max_predictor_number specified in the algorithm parameters when mode=allsubsets. The best model is the one with the highest R2 value. When mode=maxr, the model returned is no longer guaranteed to have the best R2 value.*

Description

H2O ModelSelection is used to build the best model with one predictor, two predictors, ... up to max_predictor_number specified in the algorithm parameters when mode=allsubsets. The best model is the one with the highest R2 value. When mode=maxr, the model returned is no longer guaranteed to have the best R2 value.

Usage

```
h2o.modelSelection(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  seed = -1,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  score_iteration_interval = 0,
  offset_column = NULL,
  weights_column = NULL,
  family = c("AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial",
    "poisson", "gamma", "tweedie", "negativebinomial"),
  link = c("family_default", "identity", "logit", "log", "inverse", "tweedie",
    "ologit"),
  tweedie_variance_power = 0,
  tweedie_link_power = 0,
  theta = 0,
  solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
    "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
  alpha = NULL,
  lambda = c(0),
  lambda_search = FALSE,
  early_stopping = FALSE,
  nlambdas = 0,
  standardize = TRUE,
  missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
  plug_values = NULL,
  compute_p_values = FALSE,
  remove_collinear_columns = FALSE,
  intercept = TRUE,
  non_negative = FALSE,
  max_iterations = 0,
  objective_epsilon = -1,
  beta_epsilon = 1e-04,
  gradient_epsilon = -1,
  startval = NULL,
  prior = 0,
  cold_start = FALSE,
  lambda_min_ratio = 0,
  beta_constraints = NULL,
  max_active_predictors = -1,
  obj_reg = -1,
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
    "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  max_runtime_secs = 0,
  custom_metric_func = NULL,
  nparallelism = 0,
  max_predictor_number = 1,
  min_predictor_number = 1,
  mode = c("allsubsets", "maxr", "maxrsweep", "backward"),
  build_glm_model = TRUE,
  p_values_threshold = 0,
  influence = c("dfbetas")
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for clas-

	sification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`score_iteration_interval`	Perform scoring for every score_iteration_interval iterations Defaults to 0.
`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`family`	Family. For maxr/maxrsweep, only gaussian. For backward, ordinal and multinomial families are not supported Must be one of: "AUTO", "gaussian", "binomial", "fractionalbinomial", "quasibinomial", "poisson", "gamma", "tweedie", "negativebinomial". Defaults to AUTO.
`link`	Link function. Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit". Defaults to family_default.
`tweedie_variance_power`	Tweedie variance power Defaults to 0.
`tweedie_link_power`	Tweedie link power Defaults to 0.
`theta`	Theta Defaults to 0.
`solver`	AUTO will set the solver based on given data and the other parameters. IRLSM is fast on on problems with small number of predictors and for lambda-search with L1 penalty, L_BFGS scales better for datasets with many columns. Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR Defaults to IRLSM.
`alpha`	Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.
`lambda`	Regularization strength Defaults to c(0.0).
`lambda_search`	`Logical`. Use lambda search starting at lambda max, given lambda is then interpreted as lambda min Defaults to FALSE.
`early_stopping`	`Logical`. Stop early when there is no more relative improvement on train or validation (if provided) Defaults to FALSE.

`nlambdas`	Number of lambdas to be used in a search. Default indicates: If alpha is zero, with lambda search set to True, the value of nlamdas is set to 30 (fewer lambdas are needed for ridge regression) otherwise it is set to 100. Defaults to 0.
`standardize`	`Logical`. Standardize numeric columns to have zero mean and unit variance Defaults to TRUE.
`missing_values_handling`	Handling of missing values. Either MeanImputation, Skip or PlugValues. Must be one of: "MeanImputation", "Skip", "PlugValues". Defaults to MeanImputation.
`plug_values`	Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues)
`compute_p_values`	`Logical`. Request p-values computation, p-values work only with IRLSM solver and no regularization Defaults to FALSE.
`remove_collinear_columns`	`Logical`. In case of linearly dependent columns, remove some of the dependent columns Defaults to FALSE.
`intercept`	`Logical`. Include constant term in the model Defaults to TRUE.
`non_negative`	`Logical`. Restrict coefficients (not intercept) to be non-negative Defaults to FALSE.
`max_iterations`	Maximum number of iterations Defaults to 0.
`objective_epsilon`	Converge if objective value changes less than this. Default (of -1.0) indicates: If lambda_search is set to True the value of objective_epsilon is set to .0001. If the lambda_search is set to False and lambda is equal to zero, the value of objective_epsilon is set to .000001, for any other value of lambda the default value of objective_epsilon is set to .0001. Defaults to -1.
`beta_epsilon`	Converge if beta changes less (using L-infinity norm) than beta esilon, ONLY applies to IRLSM solver Defaults to 0.0001.
`gradient_epsilon`	Converge if objective changes less (using L-infinity norm) than this, ONLY applies to L-BFGS solver. Default (of -1.0) indicates: If lambda_search is set to False and lambda is equal to zero, the default value of gradient_epsilon is equal to .000001, otherwise the default value is .0001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively. Defaults to -1.
`startval`	double array to initialize fixed and random coefficients for HGLM, coefficients for GLM.
`prior`	Prior probability for y==1. To be used only for logistic regression iff the data has been sampled and the mean of response does not reflect reality. Defaults to 0.
`cold_start`	`Logical`. Only applicable to multiple alpha/lambda values. If false, build the next model for next set of alpha/lambda values starting from the values provided by current model. If true will start GLM model from scratch. Defaults to FALSE.
`lambda_min_ratio`	Minimum lambda used in lambda search, specified as a ratio of lambda_max (the smallest lambda that drives all coefficients to zero). Default indicates: if the number of observations is greater than the number of variables, then

lambda_min_ratio is set to 0.0001; if the number of observations is less than the number of variables, then lambda_min_ratio is set to 0.01. Defaults to 0.

`beta_constraints`
: Beta constraints

`max_active_predictors`
: Maximum number of active predictors during computation. Use as a stopping criterion to prevent expensive model building with many predictors. Default indicates: If the IRLSM solver is used, the value of max_active_predictors is set to 5000 otherwise it is set to 100000000. Defaults to -1.

`obj_reg`
: Likelihood divider in objective value computation, default (of -1.0) will set it to 1/nobs Defaults to -1.

`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.

`stopping_metric`
: Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

`stopping_tolerance`
: Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

`balance_classes`
: `Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.

`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). Requires balance_classes. Defaults to 5.0.

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

`custom_metric_func`
: Reference to custom evaluation function, format: ‘language:keyName=funcName‘

`nparallelism`
: number of models to build in parallel. Defaults to 0.0 which is adaptive to the system capability Defaults to 0.

`max_predictor_number`
: Maximum number of predictors to be considered when building GLM models. Defaults to 1. Defaults to 1.

`min_predictor_number`
: For mode = ’backward’ only. Minimum number of predictors to be considered when building GLM models starting with all predictors to be included. Defaults to 1. Defaults to 1.

mode	Mode: Used to choose model selection algorithms to use. Options include 'allsubsets' for all subsets, 'maxr' that uses sequential replacement and GLM to build all models, slow but works with cross-validation, validation frames for more robust results, 'maxrsweep' that uses sequential replacement and sweeping action, much faster than 'maxr', 'backward' for backward selection. Must be one of: "allsubsets", "maxr", "maxrsweep", "backward". Defaults to maxr.
build_glm_model	Logical. For maxrsweep mode only. If true, will return full blown GLM models with the desired predictorsubsets. If false, only the predictor subsets, predictor coefficients are returned. This is forspeeding up the model selection process. The users can choose to build the GLM models themselvesby using the predictor subsets themselves. Default to true. Defaults to TRUE.
p_values_threshold	For mode='backward' only. If specified, will stop the model building process when all coefficientsp-values drop below this threshold Defaults to 0.
influence	If set to dfbetas will calculate the difference in beta when a datarow is included and excluded in the dataset. Must be one of: "dfbetas".

Examples

```
## Not run:
library(h2o)
h2o.init()
# Run ModelSelection of VOL ~ all predictors
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
model <- h2o.modelSelection(y="VOL", x=c("RACE","AGE","RACE","DPROS"), training_frame=prostate)

## End(Not run)
```

h2o.model_correlation *Model Prediction Correlation*

Description

Get a data.frame containing the correlation between the predictions of the models. For classification, frequency of identical predictions is used. By default, models are ordered by their similarity (as computed by hierarchical clustering).

Usage

```
h2o.model_correlation(object, newdata, top_n = 20, cluster_models = TRUE)
```

Arguments

object	A list of H2O models, an H2O AutoML instance, or an H2OFrame with a 'model_id' column (e.g. H2OAutoML leaderboard)..
newdata	An H2O Frame. Predictions from the models will be generated using this frame, so this should be a holdout set.

top_n	(DEPRECATED) Integer specifying the number models shown in the heatmap (used only with an AutoML object, and based on the leaderboard ranking. Defaults to 20.
cluster_models	Logical. Order models based on their similarity. Defaults to TRUE.

Value

A data.frame containing variable importance.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)

# Create the model correlation
model_correlation <- h2o.model_correlation(aml, test)
print(model_correlation)

## End(Not run)
```

h2o.model_correlation_heatmap

Model Prediction Correlation Heatmap

Description

This plot shows the correlation between the predictions of the models. For classification, frequency of identical predictions is used. By default, models are ordered by their similarity (as computed by hierarchical clustering).

Usage

```
h2o.model_correlation_heatmap(
  object,
  newdata,
  top_n = 20,
  cluster_models = TRUE,
  triangular = TRUE
)
```

Arguments

`object`	A list of H2O models, an H2O AutoML instance, or an H2OFrame with a 'model_id' column (e.g. H2OAutoML leaderboard).
`newdata`	An H2O Frame. Predictions from the models will be generated using this frame, so this should be a holdout set.
`top_n`	Integer specifying the number models shown in the heatmap (used only with an AutoML object, and based on the leaderboard ranking. Defaults to 20.
`cluster_models`	Logical. Order models based on their similarity. Defaults to TRUE.
`triangular`	Print just the lower triangular part of correlation matrix. Defaults to TRUE.

Value

A ggplot2 object.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)

# Create the model correlation heatmap
model_correlation_heatmap <- h2o.model_correlation_heatmap(aml, test)
print(model_correlation_heatmap)

## End(Not run)
```

h2o.mojo_predict_csv *H2O Prediction from R without having H2O running*

Description

Provides the method h2o.mojo_predict_csv with which you can predict a MOJO model from R.

Usage

```
h2o.mojo_predict_csv(
  input_csv_path,
  mojo_zip_path,
  output_csv_path = NULL,
  genmodel_jar_path = NULL,
  classpath = NULL,
  java_options = NULL,
  verbose = F,
  setInvNumNA = F
)
```

Arguments

`input_csv_path`	Path to input CSV file.
`mojo_zip_path`	Path to MOJO zip downloaded from H2O.
`output_csv_path`	Optional, path to the output CSV file with computed predictions. If NULL (default), then predictions will be saved as prediction.csv in the same folder as the MOJO zip.
`genmodel_jar_path`	Optional, path to genmodel jar file. If NULL (default) then the h2o-genmodel.jar in the same folder as the MOJO zip will be used.
`classpath`	Optional, specifies custom user defined classpath which will be used when scoring. If NULL (default) then the default classpath for this MOJO model will be used.
`java_options`	Optional, custom user defined options for Java. By default '-Xmx4g -XX:ReservedCodeCacheSize=2 is used.
`verbose`	Optional, if TRUE, then additional debug information will be printed. FALSE by default.
`setInvNumNA`	Optional, if TRUE, then then for an string that cannot be parsed into a number an N/A value will be produced, if false the command will fail. FALSE by default.

Value

Returns a data.frame containing computed predictions

`h2o.mojo_predict_df` *H2O Prediction from R without having H2O running*

Description

Provides the method h2o.mojo_predict_df with which you can predict a MOJO model from R.

Usage

```
h2o.mojo_predict_df(
  frame,
  mojo_zip_path,
  genmodel_jar_path = NULL,
  classpath = NULL,
  java_options = NULL,
  verbose = F,
  setInvNumNA = F
)
```

Arguments

`frame`	data.frame to score.
`mojo_zip_path`	Path to MOJO zip downloaded from H2O.
`genmodel_jar_path`	Optional, path to genmodel jar file. If NULL (default) then the h2o-genmodel.jar in the same folder as the MOJO zip will be used.
`classpath`	Optional, specifies custom user defined classpath which will be used when scoring. If NULL (default) then the default classpath for this MOJO model will be used.
`java_options`	Optional, custom user defined options for Java. By default '-Xmx4g -XX:ReservedCodeCacheSize=2 is used.
`verbose`	Optional, if TRUE, then additional debug information will be printed. FALSE by default.
`setInvNumNA`	Optional, if TRUE, then then for an string that cannot be parsed into a number an N/A value will be produced, if false the command will fail. FALSE by default.

Value

Returns a data.frame containing computed predictions

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.month(hdf["ds9"])

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

`h2o.multinomial_aucpr_table`

Retrieve the all PR AUC values in a table (One to Rest, One to One, macro and weighted average) for mutlinomial classification.

Description

Retrieves the PR AUC table from an H2OMultinomialMetrics. If "train", "valid", and "xval" parameters are FALSE (default), then the training PR AUC table is returned. If more than one parameter is set to TRUE, then a named vector of PR AUC tables are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

`object`	An H2OMultinomialMetrics object.
`train`	Retrieve the training PR AUC table
`valid`	Retrieve the validation PR AUC table
`xval`	Retrieve the cross-validation PR AUC table

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

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

---

h2o.multinomial_auc_table

Retrieve the all AUC values in a table (One to Rest, One to One, macro and weighted average) for mutlinomial classification.

Description

Retrieves the AUC table from an H2OMultinomialMetrics. If "train", "valid", and "xval" parameters are FALSE (default), then the training AUC table is returned. If more than one parameter is set to TRUE, then a named vector of AUC tables are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

`object`	An H2OMultinomialMetrics object.
`train`	Retrieve the training AUC table
`valid`	Retrieve the validation AUC table
`xval`	Retrieve the cross-validation AUC table

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

```
## Not run:
library(h2o)
h2o.init()

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

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

## End(Not run)
```

h2o.nacnt   *Count of NAs per column*

Description

Gives the count of NAs per column.

Usage

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

Arguments

x	An H2OFrame object.

Value

Returns a list containing the count of NAs per column

Examples

```
## Not run:
library(h2o)
h2o.init()

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

## End(Not run)
```

h2o.naiveBayes *Compute naive Bayes probabilities on an H2O dataset.*

Description

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.

Usage

```
h2o.naiveBayes(
  x,
  y,
  training_frame,
  model_id = NULL,
  nfolds = 0,
  seed = -1,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  laplace = 0,
  threshold = 0.001,
  min_sdev = 0.001,
  eps = 0,
  eps_sdev = 0,
  min_prob = 0.001,
  eps_prob = 0,
  compute_metrics = TRUE,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL,
  gainslift_bins = -1,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO")
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`validation_frame`	Id of the validation data frame.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`balance_classes`	`Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
`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). Requires balance_classes. Defaults to 5.0.
`laplace`	Laplace smoothing parameter Defaults to 0.
`threshold`	This argument is deprecated, use 'min_sdev' instead. The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.
`min_sdev`	The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.
`eps`	This argument is deprecated, use 'eps_sdev' instead. A threshold cutoff to deal with numeric instability, must be positive.

`eps_sdev`	A threshold cutoff to deal with numeric instability, must be positive.
`min_prob`	Min. probability to use for observations with not enough data.
`eps_prob`	Cutoff below which probability is replaced with min_prob.
`compute_metrics`	`Logical`. Compute metrics on training data Defaults to TRUE.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`export_checkpoints_dir`	Automatically export generated models to this directory.
`gainslift_bins`	Gains/Lift table number of bins. 0 means disabled.. Default value -1 means automatic binning. Defaults to -1.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.

Value

an object of class H2OBinomialModel if the response has two categorical levels, and H2OMultinomialModel otherwise.

Examples

```
## Not run:
h2o.init()
votes_path <- system.file("extdata", "housevotes.csv", package = "h2o")
votes <- h2o.uploadFile(path = votes_path, header = TRUE)
h2o.naiveBayes(x = 2:17, y = 1, training_frame = votes, laplace = 3)

## End(Not run)
```

`h2o.names` *Column names of an H2OFrame*

Description

Column names of an H2OFrame

Usage

```
h2o.names(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`names` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.names(iris)

## End(Not run)
```

---

h2o.na_omit *Remove Rows With NAs*

Description

Remove Rows With NAs

Usage

```
h2o.na_omit(object, ...)
```

Arguments

object	H2OFrame object
...	Ignored

Value

Returns an H2OFrame object containing non-NA rows.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.na_omit(frame)

## End(Not run)
```

h2o.nchar *String length*

Description

String length

Usage

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

Arguments

x	The column whose string lengths will be returned.

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_nchar <- as.h2o("r tutorial")
nchar_string <- h2o.nchar(string_to_nchar)

## End(Not run)
```

h2o.ncol *Return the number of columns present in x.*

Description

Return the number of columns present in x.

Usage

```
h2o.ncol(x)
```

Arguments

x	An H2OFrame object.

See Also

nrow for the base R implementation, `ncol()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.ncol(iris)

## End(Not run)
```

---

h2o.negative_log_likelihood

Extracts the final training negative log likelihood of a GLM model.

Description

Extracts the final training negative log likelihood of a GLM model.

Usage

```
h2o.negative_log_likelihood(model)
```

Arguments

`model`	an H2OModel object.

Value

The final training negative log likelihood of a GLM model.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "acceleration"
cars_model <- h2o.glm(y=response,
                       x=predictors,
                       training_frame = cars,
                       family="gaussian",
                       generate_scoring_history=TRUE)
nllValue <- h2o.negative_log_likelihood(cars_model)

## End(Not run)
```

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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.nlevels(cars)

## End(Not run)
```

`h2o.no_progress` *Disable Progress Bar*

Description

Disable Progress Bar

Usage

```
h2o.no_progress(expr)
```

Arguments

`expr`	When specified, disable progress bar only for the evaluation of the expr and after the evaluation return to the previous setting (default is to show the progress bar), otherwise disable it globally.

Value

Value of expr if specified, otherwise NULL.

See Also

h2o.show_progress

Examples

```
## Not run:
library(h2o)
h2o.init()
h2o.no_progress()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
iris["class"] <- as.factor(iris["class"])
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
splits <- h2o.splitFrame(iris, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]

iris_km <- h2o.kmeans(x = predictors,
                      training_frame = train,
                      validation_frame = valid,
                      k = 10, estimate_k = TRUE,
                      standardize = FALSE, seed = 1234)

## End(Not run)
```

`h2o.nrow` *Return the number of rows present in x.*

Description

Return the number of rows present in x.

Usage

```
h2o.nrow(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`nrow` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
h2o.nrow(cars)

## End(Not run)
```

`h2o.null_deviance` *Retrieve the null deviance*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial", nfolds = 0,
                        alpha = 0.5, lambda_search = FALSE)
h2o.null_deviance(prostate_glm, train = TRUE)

## End(Not run)
```

---

h2o.null_dof *Retrieve the null degrees of freedom*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial", nfolds = 0,
                        alpha = 0.5, lambda_search = FALSE)
h2o.null_dof(prostate_glm, train = TRUE)

## End(Not run)
```

h2o.num_iterations *Retrieve the number of iterations.*

Description

Retrieve the number of iterations.

Usage

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

Arguments

`object`	An H2OClusteringModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.num_iterations(prostate_glm)

## End(Not run)
```

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

# Not run to avoid windows being opened during R CMD check
# h2o.openLog("Command")
# h2o.openLog("Error")

## End(Not run)
```

h2o.pareto_front *Plot Pareto front*

Description

Create Pareto front and plot it. Pareto front contains models that are optimal in a sense that for each model in the Pareto front there isn't a model that would be better in both criteria. For example, this can be useful in picking models that are fast to predict and at the same time have high accuracy. For generic data.frames/H2OFrames input the task is assumed to be minimization for both metrics.

Usage

```
h2o.pareto_front(
  object,
  leaderboard_frame = NULL,
  x_metric = c("AUTO", "AUC", "AUCPR", "logloss", "MAE", "mean_per_class_error",
    "mean_residual_deviance", "MSE", "predict_time_per_row_ms", "RMSE", "RMSLE",
    "training_time_ms"),
  y_metric = c("AUTO", "AUC", "AUCPR", "logloss", "MAE", "mean_per_class_error",
    "mean_residual_deviance", "MSE", "predict_time_per_row_ms", "RMSE", "RMSLE",
    "training_time_ms"),
  optimum = c("AUTO", "top left", "top right", "bottom left", "bottom right"),
  title = NULL,
  color_col = "algo"
)
```

Arguments

`object`	H2OAutoML or H2OGrid or a data.frame
`leaderboard_frame`	a frame used for generating the leaderboard (used when `object` is not a frame)
`x_metric`	one of the metrics present in the leaderboard
`y_metric`	one of the metrics present in the leaderboard
`optimum`	location of the optimum on XY plane
`title`	title used for plotting
`color_col`	categorical column in the leaderboard that should be used for coloring the points

Value

An H2OParetoFront S4 object with plot method and ‘pareto_front“ slot

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
df <-  h2o.importFile("h2o://prostate.csv")

# Set the response
response <- "CAPSULE"
df[[response]] <- as.factor(df[[response]])

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)
```

```
# Create the Pareto front
pf <- h2o.pareto_front(aml)
plot(pf)
pf@pareto_front # to retrieve the Pareto front subset of the leaderboard

aml2 <- h2o.automl(y = response,
                   training_frame = train,
                   max_models = 10,
                   seed = 42)

combined_leaderboard <- h2o.make_leaderboard(list(aml, aml2), test, extra_columns = "ALL")
pf_combined <- h2o.pareto_front(combined_leaderboard, x_metric = "predict_time_per_row_ms",
                                y_metric = "rmse", optimum = "bottom left")
plot(pf_combined)
pf_combined@pareto_front

## End(Not run)
```

`h2o.parseRaw` *H2O Data Parsing*

Description

The second phase in the data ingestion step.

Usage

```
h2o.parseRaw(
  data,
  pattern = "",
  destination_frame = "",
  header = NA,
  sep = "",
  col.names = NULL,
  col.types = NULL,
  na.strings = NULL,
  blocking = FALSE,
  parse_type = NULL,
  chunk_size = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = NULL,
  partition_by = NULL,
  quotechar = NULL,
  escapechar = ""
)
```

Arguments

data	An H2OFrame object to be parsed.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.

`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. If skipped_columns are specified, only list column names of columns that are not skipped.
`col.types`	(Optional) A vector specifying the types to attempt to force over columns. If skipped_columns are specified, only list column types of columns that are not skipped.
`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
`decrypt_tool`	(Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.
`skipped_columns`	a list of column indices to be excluded from parsing
`custom_non_data_line_markers`	(Optional) If a line in imported file starts with any character in given string it will NOT be imported. Empty string means all lines are imported, NULL means that default behaviour for given format will be used
`partition_by`	(Optional) Names of the columns the persisted dataset has been partitioned by.
`quotechar`	A hint for the parser which character to expect as quoting character. None (default) means autodetection.
`escapechar`	(Optional) One ASCII character used to escape other characters.

Details

Parse the Raw Data produced by the import phase.

See Also

h2o.importFile, h2o.parseSetup

`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,
  pattern = "",
  destination_frame = "",
  header = NA,
  sep = "",
  col.names = NULL,
  col.types = NULL,
  na.strings = NULL,
  parse_type = NULL,
  chunk_size = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = NULL,
  partition_by = NULL,
  single_quotes = FALSE,
  escapechar = ""
)
```

Arguments

`data`	An H2OFrame object to be parsed.
`pattern`	(Optional) Character string containing a regular expression to match file(s) in the folder.
`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. If skipped_columns are specified, only list column names of columns that are not skipped.
`col.types`	(Optional) A vector specifying the types to attempt to force over columns. If skipped_columns are specified, only list column types of columns that are not skipped.
`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"
`chunk_size`	size of chunk of (input) data in bytes
`decrypt_tool`	(Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.
`skipped_columns`	a list of column indices to be excluded from parsing
`custom_non_data_line_markers`	(Optional) If a line in imported file starts with any character in given string it will NOT be imported. Empty string means all lines are imported, NULL means that default behaviour for given format will be used

`partition_by`	(Optional) Names of the columns the persisted dataset has been partitioned by.
`single_quotes`	If set to true, the parser expects single quotes. False for double quotes (default).
`escapechar`	(Optional) One ASCII character used to escape other characters.

See Also

h2o.parseRaw

`h2o.partialPlot` *Partial Dependence Plots*

Description

Partial dependence plot gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. Note: Unlike randomForest's partialPlot when plotting partial dependence the mean response (probabilities) is returned rather than the mean of the log class probability.

Usage

```
h2o.partialPlot(
  object,
  data,
  cols,
  destination_key,
  nbins = 20,
  plot = TRUE,
  plot_stddev = TRUE,
  weight_column = -1,
  include_na = FALSE,
  user_splits = NULL,
  col_pairs_2dpdp = NULL,
  save_to = NULL,
  row_index = -1,
  targets = NULL
)
```

Arguments

`object`	An H2OModel object.
`data`	An H2OFrame object used for scoring and constructing the plot.
`cols`	Feature(s) for which partial dependence will be calculated.
`destination_key`	An key reference to the created partial dependence tables in H2O.
`nbins`	Number of bins used. For categorical columns make sure the number of bins exceeds the level count. If you enable add_missing_NA, the returned length will be nbin+1.
`plot`	A logical specifying whether to plot partial dependence table.
`plot_stddev`	A logical specifying whether to add std err to partial dependence plot.

`weight_column`	A string denoting which column of data should be used as the weight column.
`include_na`	A logical specifying whether missing value should be included in the Feature values.
`user_splits`	A two-level nested list containing user defined split points for pdp plots for each column. If there are two columns using user defined split points, there should be two lists in the nested list. Inside each list, the first element is the column name followed by values defined by the user.
`col_pairs_2dpdp`	A two-level nested list like this: col_pairs_2dpdp = list(c("col1_name", "col2_name"), c("col1_name","col3_name"), ...,) where a 2D partial plots will be generated for col1_name, col2_name pair, for col1_name, col3_name pair and whatever other pairs that are specified in the nested list.
`save_to`	Fully qualified prefix of the image files the resulting plots should be saved to, e.g. '/home/user/pdp'. Plots for each feature are saved separately in PNG format, each file receives a suffix equal to the corresponding feature name, e.g. '/home/user/pdp_AGE.png'. If the files already exists, they will be overridden. Files are only saves if plot = TRUE (default).
`row_index`	Row for which partial dependence will be calculated instead of the whole input frame.
`targets`	Target classes for multinomial model.

Value

Plot and list of calculated mean response tables for each feature requested.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate[, "CAPSULE"] <- as.factor(prostate[, "CAPSULE"] )
prostate[, "RACE"] <- as.factor(prostate[, "RACE"] )
prostate_gbm <- h2o.gbm(x = c("AGE", "RACE"),
                        y = "CAPSULE",
                        training_frame = prostate,
                        ntrees = 10,
                        max_depth = 5,
                        learn_rate = 0.1)
h2o.partialPlot(object = prostate_gbm, data = prostate, cols = c("AGE", "RACE"))

iris_hex <- as.h2o(iris)
iris_gbm <- h2o.gbm(x = c(1:4), y = 5, training_frame = iris_hex)

# one target class
h2o.partialPlot(object = iris_gbm, data = iris_hex, cols="Petal.Length", targets=c("setosa"))
# three target classes
h2o.partialPlot(object = iris_gbm, data = iris_hex, cols="Petal.Length",
                 targets=c("setosa", "virginica", "versicolor"))

## End(Not run)
```

h2o.pd_multi_plot — *Plot partial dependencies for a variable across multiple models*

Description

Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

Usage

```
h2o.pd_multi_plot(
  object,
  newdata,
  column,
  best_of_family = TRUE,
  target = NULL,
  row_index = NULL,
  max_levels = 30,
  show_rug = TRUE
)
```

Arguments

`object`	Either a list of H2O models/model_ids or an H2OAutoML object.
`newdata`	An H2OFrame.
`column`	A feature column name to inspect. Character string.
`best_of_family`	If TRUE, plot only the best model of each algorithm family; if FALSE, plot all models. Defaults to TRUE.
`target`	If multinomial, plot PDP just for `target` category.
`row_index`	Optional. Calculate Individual Conditional Expectation (ICE) for row, `row_index`. Integer.
`max_levels`	An integer specifying the maximum number of factor levels to show. Defaults to 30.
`show_rug`	Show rug to visualize the density of the column. Defaults to TRUE.

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)
```

```
# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)

# Create the partial dependence plot
pdp <- h2o.pd_multi_plot(aml, test, column = "alcohol")
print(pdp)

## End(Not run)
```

h2o.pd_plot *Plot partial dependence for a variable*

Description

Partial dependence plot (PDP) gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. PDP assumes independence between the feature for which is the PDP computed and the rest.

Usage

```
h2o.pd_plot(
  object,
  newdata,
  column,
  target = NULL,
  row_index = NULL,
  max_levels = 30,
  binary_response_scale = c("response", "logodds"),
  grouping_column = NULL,
  nbins = 100,
  show_rug = TRUE
)
```

Arguments

`object`	An H2O model.
`newdata`	An H2OFrame. Used to generate predictions used in Partial Dependence calculations.
`column`	A feature column name to inspect. Character string.
`target`	If multinomial, plot PDP just for `target` category. Character string.

row_index	Optional. Calculate Individual Conditional Expectation (ICE) for row, row_index. Integer.
max_levels	An integer specifying the maximum number of factor levels to show. Defaults to 30.
binary_response_scale	Option for binary model to display (on the y-axis) the logodds instead of the actual score. Can be one of: "response", "logodds". Defaults to "response".
grouping_column	A feature column name to group the data and provide separate sets of plots by grouping feature values
nbins	A number of bins used. Defaults to 100.
show_rug	Show rug to visualize the density of the column. Defaults to TRUE.

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the partial dependence plot
pdp <- h2o.pd_plot(gbm, test, column = "alcohol")
print(pdp)

## End(Not run)
```

---

h2o.performance *Model Performance Metrics in H2O*

Description

Given a trained h2o model, compute its performance on the given dataset. However, if the dataset does not contain the response/target column, no performance will be returned. Instead, a warning message will be printed.

Usage

```
h2o.performance(
  model,
  newdata = NULL,
  train = FALSE,
  valid = FALSE,
  xval = FALSE,
  data = NULL,
  auc_type = "NONE"
)
```

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). Note: when the trained h2o model uses balance_classes, the training metrics constructed during training will be from the balanced training dataset. For more information visit: `https://0xdata.atlassian.net/browse/TN-9`
`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‘.
`auc_type`	For multinomila model only. Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Default is "NONE"

Value

Returns an object of the H2OModelMetrics subclass.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
h2o.performance(model = prostate_gbm, newdata=prostate)

## If model uses balance_classes
## the results from train = TRUE will not match the results from newdata = prostate
prostate_gbm_balanced <- h2o.gbm(3:9, "CAPSULE", prostate, balance_classes = TRUE)
h2o.performance(model = prostate_gbm_balanced, newdata = prostate)
h2o.performance(model = prostate_gbm_balanced, train = TRUE)
```

```
## End(Not run)
```

h2o.permutation_importance

Calculate Permutation Feature Importance.

Description

When n_repeats == 1, the result is similar to the one from h2o.varimp(), i.e., it contains the following columns "Relative Importance", "Scaled Importance", and "Percentage".

Usage

```
h2o.permutation_importance(
  object,
  newdata,
  metric = c("AUTO", "AUC", "MAE", "MSE", "RMSE", "logloss", "mean_per_class_error",
    "PR_AUC"),
  n_samples = 10000,
  n_repeats = 1,
  features = NULL,
  seed = -1
)
```

Arguments

object	A trained supervised H2O model.
newdata	Training frame of the model which is going to be permuted
metric	Metric to be used. One of "AUTO", "AUC", "MAE", "MSE", "RMSE", "logloss", "mean_per_class_error", "PR_AUC". Defaults to "AUTO".
n_samples	Number of samples to be evaluated. Use -1 to use the whole dataset. Defaults to 10 000.
n_repeats	Number of repeated evaluations. Defaults to 1.
features	Character vector of features to include in the permutation importance. Use NULL to include all.
seed	Seed for the random generator. Use -1 to pick a random seed. Defaults to -1.

Details

When n_repeats > 1, the individual columns correspond to the permutation variable importance values from individual runs which corresponds to the "Relative Importance" and also to the distance between the original prediction error and prediction error using a frame with a given feature permuted.

Value

H2OTable with variable importance.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
h2o.permutation_importance(model, prostate)

## End(Not run)
```

`h2o.permutation_importance_plot`
Plot Permutation Variable Importances.

Description

This method plots either a bar plot or if n_repeats > 1 a box plot and returns the variable importance table.

Usage

```
h2o.permutation_importance_plot(
  object,
  newdata,
  metric = c("AUTO", "AUC", "MAE", "MSE", "RMSE", "logloss", "mean_per_class_error",
    "PR_AUC"),
  n_samples = 10000,
  n_repeats = 1,
  features = NULL,
  seed = -1,
  num_of_features = NULL
)
```

Arguments

`object`	A trained supervised H2O model.
`newdata`	Training frame of the model which is going to be permuted
`metric`	Metric to be used. One of "AUTO", "AUC", "MAE", "MSE", "RMSE", "logloss", "mean_per_class_error", "PR_AUC". Defaults to "AUTO".
`n_samples`	Number of samples to be evaluated. Use -1 to use the whole dataset. Defaults to 10 000.
`n_repeats`	Number of repeated evaluations. Defaults to 1.
`features`	Character vector of features to include in the permutation importance. Use NULL to include all.
`seed`	Seed for the random generator. Use -1 to pick a random seed. Defaults to -1.
`num_of_features`	The number of features shown in the plot (default is 10 or all if less than 10).

Value

H2OTable with variable importance.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
h2o.permutation_importance_plot(model, prostate)

## End(Not run)
```

`h2o.pivot` *Pivot a frame*

Description

Pivot the frame designated by the three columns: index, column, and value. Index and column should be of type enum, int, or time. For cases of multiple indexes for a column label, the aggregation method is to pick the first occurrence in the data frame

Usage

```
h2o.pivot(x, index, column, value)
```

Arguments

`x`	an H2OFrame
`index`	the column where pivoted rows should be aligned on
`column`	the column to pivot
`value`	values of the pivoted table

Value

An H2OFrame with columns from the columns arg, aligned on the index arg, with values from values arg

Examples

```
## Not run:
library(h2o)
h2o.init()

df = h2o.createFrame(rows = 1000, cols = 3, factors = 10, integer_fraction = 1.0/3,
                     categorical_fraction = 1.0/3, missing_fraction = 0.0, seed = 123)
df$C3 = h2o.abs(df$C3)
h2o.pivot(df, index="C3", column="C2", value="C1")

## End(Not run)
```

h2o.prcomp *Principal component analysis of an H2O data frame*

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,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  pca_method = c("GramSVD", "Power", "Randomized", "GLRM"),
 pca_impl = c("MTJ_EVD_DENSEMATRIX", "MTJ_EVD_SYMMMATRIX", "MTJ_SVD_DENSEMATRIX",
    "JAMA"),
 k = 1,
  max_iterations = 1000,
  use_all_factor_levels = FALSE,
  compute_metrics = TRUE,
  impute_missing = FALSE,
  seed = -1,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`transform`	Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
`pca_method`	Specify the algorithm to use for computing the principal components: GramSVD - uses a distributed computation of the Gram matrix, followed by a local SVD; Power - computes the SVD using the power iteration method (experimental); Randomized - uses randomized subspace iteration method; GLRM - fits a generalized low-rank model with L2 loss function and no regularization and solves for

the SVD using local matrix algebra (experimental) Must be one of: "GramSVD", "Power", "Randomized", "GLRM". Defaults to GramSVD.

`pca_impl` Specify the implementation to use for computing PCA (via SVD or EVD): MTJ_EVD_DENSEMATRIX - eigenvalue decompositions for dense matrix using MTJ; MTJ_EVD_SYMMMATRIX - eigenvalue decompositions for symmetric matrix using MTJ; MTJ_SVD_DENSEMATRIX - singular-value decompositions for dense matrix using MTJ; JAMA - eigenvalue decompositions for dense matrix using JAMA. References: JAMA - http://math.nist.gov/javanumerics/jama/; MTJ - https://github.com/fommil/matrix-toolkits-java/ Must be one of: "MTJ_EVD_DENSEMATRI "MTJ_EVD_SYMMMATRIX", "MTJ_SVD_DENSEMATRIX", "JAMA".

`k` Rank of matrix approximation Defaults to 1.

`max_iterations` Maximum training iterations Defaults to 1000.

`use_all_factor_levels` `Logical`. Whether first factor level is included in each categorical expansion Defaults to FALSE.

`compute_metrics` `Logical`. Whether to compute metrics on the training data Defaults to TRUE.

`impute_missing` `Logical`. Whether to impute missing entries with the column mean Defaults to FALSE.

`seed` Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

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

`export_checkpoints_dir` Automatically export generated models to this directory.

Value

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

```
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
h2o.prcomp(training_frame = australia, k = 8, transform = "STANDARDIZE")

## End(Not run)
```

`h2o.predict` *Predict on an H2O Model*

Description

Predict on an H2O Model

Usage

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

Arguments

`object`	a fitted model 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.

Value

Returns an H2OFrame object with probabilites and default predictions.

`h2o.predicted_vs_actual_by_variable` *Calculates per-level mean of predicted value vs actual value for a given variable.*

Description

In the basic setting, this function is equivalent to doing group-by on variable and calculating mean on predicted and actual. In addition to that it also handles NAs in response and weights automatically.

Usage

```
h2o.predicted_vs_actual_by_variable(object, newdata, predicted, variable)
```

Arguments

`object`	A trained supervised H2O model.
`newdata`	Input frame (can be training/test/.. frame).
`predicted`	Frame of predictions for the given input frame.
`variable`	Name of variable to inspect.

Value

H2OTable

h2o.predict_json *H2O Prediction from R without having H2O running*

Description

Provides the method h2o.predict with which you can predict a MOJO or POJO Jar model from R.

Usage

```
h2o.predict_json(model, json, genmodelpath, labels, classpath, javaoptions)
```

Arguments

model	String with file name of MOJO or POJO Jar
json	JSON String with inputs to model
genmodelpath	(Optional) path name to h2o-genmodel.jar, if not set defaults to same dir as MOJO
labels	(Optional) if TRUE then show output labels in result
classpath	(Optional) Extra items for the class path of where to look for Java classes, e.g., h2o-genmodel.jar
javaoptions	(Optional) Java options string, default if "-Xmx4g"

Value

Returns an object with the prediction result

Examples

```
## Not run:
library(h2o)
h2o.predict_json('~/GBM_model_python_1473313897851_6.zip', '{"C7":1}')
h2o.predict_json('~/GBM_model_python_1473313897851_6.zip', '{"C7":1}', c(".", "lib"))

## End(Not run)
```

h2o.predict_rules *Evaluates validity of the given rules on the given data. Returns a frame with a column per each input rule id, representing a flag whether given rule is applied to the observation or not.*

Description

Evaluates validity of the given rules on the given data. Returns a frame with a column per each input rule id, representing a flag whether given rule is applied to the observation or not.

Usage

```
h2o.predict_rules(model, frame, rule_ids)
```

Arguments

model	A trained rulefit model.
frame	A frame on which rule validity is to be evaluated
rule_ids	Rule ids to be evaluated against the frame

Examples

```
## Not run:
library(h2o)
h2o.init()
titanic <- h2o.importFile(
 "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/titanic.csv"
)
response = "survived"
predictors <- c("age", "sibsp", "parch", "fare", "sex", "pclass")
titanic[,response] <- as.factor(titanic[,response])
titanic[,"pclass"] <- as.factor(titanic[,"pclass"])

splits <- h2o.splitFrame(data = titanic, ratios = .8, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]

rfit <- h2o.rulefit(y = response, x = predictors, training_frame = train, validation_frame = test,
min_rule_length = 1, max_rule_length = 10, max_num_rules = 100, seed = 1, model_type="rules")
h2o.predict_rules(rfit, train, c("M1T0N7, M1T49N7, M1T16N7", "M1T36N7", "M2T19N19"))

## End(Not run)
```

h2o.print *Print An H2OFrame*

Description

Print An H2OFrame

Usage

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

Arguments

x	An H2OFrame object
n	An (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. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).

Examples

```
## Not run:
library()
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.print(iris["species"], n = 15)

## End(Not run)
```

h2o.prod *Return the product of all the values present in its arguments.*

Description

Return the product of all the values present in its arguments.

Usage

```
h2o.prod(x)
```

Arguments

x	An H2OFrame object.

See Also

prod for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.prod(iris["petal_len"])

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_glrm <- h2o.glrm(training_frame = iris_hf, k = 4, loss = "Quadratic",
                      multi_loss = "Categorical", max_iterations = 1000)
iris_parch <- h2o.proj_archetypes(iris_glrm, iris_hf)
head(iris_parch)

## End(Not run)
```

h2o.psvm *Trains a Support Vector Machine model on an H2O dataset*

Description

Alpha version. Supports only binomial classification problems.

Usage

```
h2o.psvm(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  hyper_param = 1,
  kernel_type = c("gaussian"),
  gamma = -1,
  rank_ratio = -1,
  positive_weight = 1,
  negative_weight = 1,
  disable_training_metrics = TRUE,
  sv_threshold = 1e-04,
  fact_threshold = 1e-05,
  feasible_threshold = 0.001,
  surrogate_gap_threshold = 0.001,
  mu_factor = 10,
  max_iterations = 200,
  seed = -1
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a binary categorical/factor variable or a numeric variable with values -1/1 (for compatibility with SVMlight format).
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`hyper_param`	Penalty parameter C of the error term Defaults to 1.
`kernel_type`	Type of used kernel Must be one of: "gaussian". Defaults to gaussian.
`gamma`	Coefficient of the kernel (currently RBF gamma for gaussian kernel, -1 means 1/#features) Defaults to -1.

`rank_ratio`	Desired rank of the ICF matrix expressed as an ration of number of input rows (-1 means use sqrt(#rows)). Defaults to -1.
`positive_weight`	Weight of positive (+1) class of observations Defaults to 1.
`negative_weight`	Weight of positive (-1) class of observations Defaults to 1.
`disable_training_metrics`	`Logical`. Disable calculating training metrics (expensive on large datasets) Defaults to TRUE.
`sv_threshold`	Threshold for accepting a candidate observation into the set of support vectors Defaults to 0.0001.
`fact_threshold`	Convergence threshold of the Incomplete Cholesky Factorization (ICF) Defaults to 1e-05.
`feasible_threshold`	Convergence threshold for primal-dual residuals in the IPM iteration Defaults to 0.001.
`surrogate_gap_threshold`	Feasibility criterion of the surrogate duality gap (eta) Defaults to 0.001.
`mu_factor`	Increasing factor mu Defaults to 10.
`max_iterations`	Maximum number of iteration of the algorithm Defaults to 200.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the splice dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/splice/splice.svm"
splice <- h2o.importFile(f)

# Train the Support Vector Machine model
svm_model <- h2o.psvm(gamma = 0.01, rank_ratio = 0.1,
                      y = "C1", training_frame = splice,
                      disable_training_metrics = FALSE)

## End(Not run)
```

`h2o.qini` *Retrieve the default Qini value*

Description

Retrieves the Qini value from an H2OBinomialUpliftMetrics. If "train" and "valid" parameters are FALSE (default), then the training Qini value is returned. If more than one parameter is set to TRUE, then a named vector of Qini values are returned, where the names are "train", "valid".

Usage

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

Arguments

`object`	An H2OBinomialUpliftMetrics or
`train`	Retrieve the training Qini value
`valid`	Retrieve the validation Qini

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.qini(perf)

## End(Not run)
```

---

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

```
## Not run:
# Request quantiles for an H2O parsed data set:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
# Request quantiles for a subset of columns in an H2O parsed data set
quantile(prostate[, 3])
for(i in 1:ncol(prostate))
   quantile(prostate[, i])

## End(Not run)
```

`h2o.r2` *Retrieve the R2 value*

Description

Retrieves the R2 value from an H2O model. Will return R^2 for GLM Models. 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

```
## Not run:
library(h2o)

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

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

h2o.r2(m)

## End(Not run)
```

`h2o.randomForest` *Build a Random Forest model*

Description

Builds a Random Forest model on an H2OFrame.

Usage

```
h2o.randomForest(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  score_each_iteration = FALSE,
  score_tree_interval = 0,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  offset_column = NULL,
  weights_column = NULL,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  ntrees = 50,
  max_depth = 20,
  min_rows = 1,
  nbins = 20,
  nbins_top_level = 1024,
  nbins_cats = 1024,
  r2_stopping = Inf,
  stopping_rounds = 0,
 stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
   "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  max_runtime_secs = 0,
  seed = -1,
  build_tree_one_node = FALSE,
  mtries = -1,
  sample_rate = 0.632,
  sample_rate_per_class = NULL,
  binomial_double_trees = FALSE,
  checkpoint = NULL,
  col_sample_rate_change_per_level = 1,
  col_sample_rate_per_tree = 1,
  min_split_improvement = 1e-05,
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal",
    "RoundRobin", "UniformRobust"),
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  calibrate_model = FALSE,
  calibration_frame = NULL,
  calibration_method = c("AUTO", "PlattScaling", "IsotonicRegression"),
 distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
    "tweedie", "laplace", "quantile", "huber"),
  custom_metric_func = NULL,
  export_checkpoints_dir = NULL,
  check_constant_response = TRUE,
  gainslift_bins = -1,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  verbose = FALSE
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
y	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
training_frame	Id of the training data frame.
model_id	Destination id for this model; auto-generated if not specified.
validation_frame	Id of the validation data frame.

`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`score_tree_interval`	Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
`fold_column`	Column with cross-validation fold index assignment per observation.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`offset_column`	Offset column. This argument is deprecated and has no use for Random Forest.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`balance_classes`	`Logical`. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
`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). Requires balance_classes. Defaults to 5.0.
`ntrees`	Number of trees. Defaults to 50.
`max_depth`	Maximum tree depth (0 for unlimited). Defaults to 20.
`min_rows`	Fewest allowed (weighted) observations in a leaf. Defaults to 1.
`nbins`	For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.

`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 Defaults to 1024.
`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. Defaults to 1024.
`r2_stopping`	r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this Defaults to 1.797693135e+308.
`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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.
`stopping_metric`	Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.
`stopping_tolerance`	Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`build_tree_one_node`	`Logical`. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
`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 # of predictors Defaults to -1.
`sample_rate`	Row sample rate per tree (from 0.0 to 1.0) Defaults to 0.632.
`sample_rate_per_class`	A list of row sample rates per class (relative fraction for each class, from 0.0 to 1.0), for each tree
`binomial_double_trees`	`Logical`. For binary classification: Build 2x as many trees (one per class) - can lead to higher accuracy. Defaults to FALSE.
`checkpoint`	Model checkpoint to resume training with.
`col_sample_rate_change_per_level`	Relative change of the column sampling rate for every level (must be > 0.0 and <= 2.0) Defaults to 1.
`col_sample_rate_per_tree`	Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.

`min_split_improvement`	Minimum relative improvement in squared error reduction for a split to happen Defaults to 1e-05.
`histogram_type`	What type of histogram to use for finding optimal split points Must be one of: "AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin", "UniformRobust". Defaults to AUTO.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`calibrate_model`	`Logical`. Use Platt Scaling (default) or Isotonic Regression to calculate calibrated class probabilities. Calibration can provide more accurate estimates of class probabilities. Defaults to FALSE.
`calibration_frame`	Data for model calibration
`calibration_method`	Calibration method to use Must be one of: "AUTO", "PlattScaling", "IsotonicRegression". Defaults to AUTO.
`distribution`	Distribution. This argument is deprecated and has no use for Random Forest.
`custom_metric_func`	Reference to custom evaluation function, format: ‘language:keyName=funcName‘
`export_checkpoints_dir`	Automatically export generated models to this directory.
`check_constant_response`	`Logical`. Check if response column is constant. If enabled, then an exception is thrown if the response column is a constant value.If disabled, then model will train regardless of the response column being a constant value or not. Defaults to TRUE.
`gainslift_bins`	Gains/Lift table number of bins. 0 means disabled.. Default value -1 means automatic binning. Defaults to -1.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
`verbose`	`Logical`. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

Value

Creates a H2OModel object of the right type.

See Also

`predict.H2OModel` for prediction

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the cars dataset
```

```
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)

# Set predictors and response; set response as a factor
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"

# Train the DRF model
cars_drf <- h2o.randomForest(x = predictors, y = response,
                            training_frame = cars, nfolds = 5,
                            seed = 1234)

## End(Not run)
```

h2o.range — *Returns a vector containing the minimum and maximum of all the given arguments.*

Description

Returns a vector containing the minimum and maximum of all the given arguments.

Usage

```
h2o.range(x, na.rm = FALSE, finite = FALSE)
```

Arguments

x	An H2OFrame object.
na.rm	`logical`. indicating whether missing values should be removed.
finite	`logical`. indicating if all non-finite elements should be omitted.

See Also

`range` for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.range(iris["petal_len"], na.rm = TRUE, finite = TRUE)

## End(Not run)
```

h2o.rank_within_group_by

This function will add a new column rank where the ranking is produced as follows: 1. sorts the H2OFrame by columns sorted in by columns specified in group_by_cols and sort_cols in the directions specified by the ascending for the sort_cols. The sort directions for the group_by_cols are ascending only. 2. A new rank column is added to the frame which will contain a rank assignment performed next. The user can choose to assign a name to this new column. The default name is New_Rank_column. 3. For each groupby groups, a rank is assigned to the row starting from 1, 2, ... to the end of that group. 4. If sort_cols_sorted is TRUE, a final sort on the frame will be performed frame according to the sort_cols and the sort directions in ascending. If sort_cols_sorted is FALSE (by default), the frame from step 3 will be returned as is with no extra sort. This may provide a small speedup if desired.

Description

This function will add a new column rank where the ranking is produced as follows: 1. sorts the H2OFrame by columns sorted in by columns specified in group_by_cols and sort_cols in the directions specified by the ascending for the sort_cols. The sort directions for the group_by_cols are ascending only. 2. A new rank column is added to the frame which will contain a rank assignment performed next. The user can choose to assign a name to this new column. The default name is New_Rank_column. 3. For each groupby groups, a rank is assigned to the row starting from 1, 2, ... to the end of that group. 4. If sort_cols_sorted is TRUE, a final sort on the frame will be performed frame according to the sort_cols and the sort directions in ascending. If sort_cols_sorted is FALSE (by default), the frame from step 3 will be returned as is with no extra sort. This may provide a small speedup if desired.

Usage

```
h2o.rank_within_group_by(
  x,
  group_by_cols,
  sort_cols,
  ascending = NULL,
  new_col_name = "New_Rank_column",
  sort_cols_sorted = FALSE
)
```

Arguments

x	The H2OFrame input to be sorted.
group_by_cols	a list of column names or indices to form the groupby groups
sort_cols	a list of column names or indices for sorting
ascending	a list of Boolean to determine if ascending sort (set to TRUE) is needed for each column in sort_cols (optional). Default is ascending sort for all. To perform descending sort, set value to FALSE

`new_col_name`	new column name for the newly added rank column if specified (optional). Default name is New_Rank_column.
`sort_cols_sorted`	Boolean to determine if the final returned frame is to be sorted according to the sort_cols and sort directions in ascending. Default is FALSE. The following example is generated by Nidhi Mehta. If the input frame is train: ID Group_by_column num data Column_to_arrange_by num_1 fdata 12 1 2941.552 1 3 -3177.9077 1 12 1 2941.552 1 5 -13311.8247 1 12 2 -22722.174 1 3 -3177.9077 1 12 2 -22722.174 1 5 -13311.8247 1 13 3 -12776.884 1 5 -18421.6171 0 13 3 -12776.884 1 4 28080.1607 0 13 1 -6049.830 1 5 -18421.6171 0 13 1 -6049.830 1 4 28080.1607 0 15 3 -16995.346 1 1 -9781.6373 0 16 1 -10003.593 0 3 -61284.6900 0 16 3 26052.495 1 3 -61284.6900 0 16 3 -22905.288 0 3 -61284.6900 0 17 2 -13465.496 1 2 12094.4851 1 17 2 -13465.496 1 3 -11772.1338 1 17 2 -13465.496 1 3 -415.1114 0 17 2 -3329.619 1 2 12094.4851 1 17 2 -3329.619 1 3 -11772.1338 1 17 2 -3329.619 1 3 -415.1114 0 If the following commands are issued: rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE)) h2o.summary(rankedF1) The returned frame rankedF1 will look like this: ID Group_by_column num fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column 12 1 2941.552 1 3 -3177.9077 1 1 16 1 -10003.593 0 3 -61284.6900 0 2 13 1 -6049.830 0 4 28080.1607 0 3 12 1 2941.552 1 5 -13311.8247 1 4 13 1 -6049.830 0 5 -18421.6171 0 5 17 2 -13465.496 0 2 12094.4851 1 1 17 2 -3329.619 0 2 12094.4851 1 2 12 2 -22722.174 1 3 -3177.9077 1 3 17 2 -13465.496 0 3 -11772.1338 1 4 17 2 -13465.496 0 3 -415.1114 0 5 17 2 -3329.619 0 3 -11772.1338 1 6 17 2 -3329.619 0 3 -415.1114 0 7 12 2 -22722.174 1 5 -13311.8247 1 8 15 3 -16995.346 1 1 -9781.6373 0 1 16 3 26052.495 0 3 -61284.6900 0 2 16 3 -22905.288 1 3 -61284.6900 0 3 13 3 -12776.884 1 4 28080.1607 0 4 13 3 -12776.884 1 5 -18421.6171 0 5 If the following commands are issued: rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE), sort_cols_sorted=TRUE) h2o.summary(rankedF1) The returned frame will be sorted according to sortCols and hence look like this instead: ID Group_by_column num fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column 15 3 -16995.346 1 1 -9781.6373 0 1 17 2 -13465.496 0 2 12094.4851 1 1 17 2 -3329.619 0 2 12094.4851 1 2 12 1 2941.552 1 3 -3177.9077 1 1 12 2 -22722.174 1 3 -3177.9077 1 3 16 1 -10003.593 0 3 -61284.6900 0 2 16 3 26052.495 0 3 -61284.6900 0 2 16 3 -22905.288 1 3 -61284.6900 0 3 17 2 -13465.496 0 3 -11772.1338 1 4 17 2 -13465.496 0 3 -415.1114 0 5 17 2 -3329.619 0 3 -11772.1338 1 6 17 2 -3329.619 0 3 -415.1114 0 7 13 3 -12776.884 1 4 28080.1607 0 4 13 1 -6049.830 0 4 28080.1607 0 3 12 1 2941.552 1 5 -13311.8247 1 4 12 2 -22722.174 1 5 -13311.8247 1 8 13 3 -12776.884 1 5 -18421.6171 0 5 13 1 -6049.830 0 5 -18421.6171 0 5

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip"
air <- h2o.importFile(f)
```

```
group_cols <- c("Distance")
sort_cols <- c("IsArrDelayed", "IsDepDelayed")
sort_directions <- c(TRUE, FALSE)
h2o.rank_within_group_by(x = air, group_by_cols = group_cols,
                         sort_cols = sort_cols,
                         ascending = sort_directions,
                         new_col_name = "New_Rank",
                         sort_cols_sorted = TRUE)

## End(Not run)
```

`h2o.rapids` *Execute a Rapids expression.*

Description

Execute a Rapids expression.

Usage

```
h2o.rapids(expr)
```

Arguments

`expr`	The rapids expression (ascii string)

Examples

```
## Not run:
h2o.rapids('(setproperty "sys.ai.h2o.algos.evaluate_auto_model_parameters" "true")')

## End(Not run)
```

`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

`cbind` for the base `R` method, `rbind()`.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_rbind <- h2o.rbind(prostate, prostate)
head(prostate_rbind)
dim(prostate)
dim(prostate_rbind)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_glrm <- h2o.glrm(training_frame = iris_hf, k = 4, transform = "STANDARDIZE",
                      loss = "Quadratic", multi_loss = "Categorical", max_iterations = 1000)
iris_rec <- h2o.reconstruct(iris_glrm, iris_hf, reverse_transform = TRUE)
head(iris_rec)

## End(Not run)
```

---

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

Examples

```
## Not run:
library(h2o)
h2o.init()

# Convert iris dataset to an H2OFrame
iris_hf <- as.h2o(iris)
# Look at current ordering of the Species column levels
h2o.levels(iris_hf["Species"])
# "setosa"     "versicolor" "virginica"
# Change the reference level to "virginica"
iris_hf["Species"] <- h2o.relevel(x = iris_hf["Species"], y = "virginica")
# Observe new ordering
h2o.levels(iris_hf["Species"])
# "virginica"  "setosa"     "versicolor"

## End(Not run)
```

`h2o.relevel_by_frequency` *Reorders levels of factor columns by the frequencies for the individual levels.*

Description

The levels of a factor are reordered so that the most frequency level is at level 0, remaining levels are ordered from the second most frequent to the least frequent.

Usage

```
h2o.relevel_by_frequency(x, weights_column = NULL, top_n = -1)
```

Arguments

`x`	H2O frame with some factor columns
`weights_column`	optional name of weights column
`top_n`	optional number of most frequent levels to move to the top (eg.: for top_n=1 move only the most frequent level)

Value

new reordered frame

Examples

```
## Not run:
library(h2o)
h2o.init()

# Convert iris dataset to an H2OFrame
iris_hf <- as.h2o(iris)
# Look at current ordering of the Species column levels
h2o.levels(iris_hf["Species"])
# "setosa"     "versicolor" "virginica"
# Change the reference level to "virginica"
iris_hf["Species"] <- h2o.relevel_by_frequency(x = iris_hf["Species"])
# Observe new ordering
h2o.levels(iris_hf["Species"])
# "virginica"  "versicolor" "setosa"

## End(Not run)
```

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

Description

Removes the data from the h2o cluster, but does not remove the local references. Retains models, frames and vectors specified in retained_elements argument. Retained elements must be instances/ids of models and frames only. For models retained, training and validation frames are retained as well. Cross validation models of a retained model are NOT retained automatically, those must be specified explicitely.

Usage

```
h2o.removeAll(timeout_secs = 0, retained_elements = c())
```

Arguments

`timeout_secs`	Timeout in seconds. Default is no timeout.
`retained_elements`	Instances or ids of models and frames to be retained. Combination of instances and ids in the same list is also a valid input.

See Also

h2o.rm

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.ls()
h2o.removeAll()
h2o.ls()

## End(Not run)
```

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

Usage

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

Arguments

x	an H2O frame
length.out	non negative integer. The desired length of the output vector.

Value

Creates an H2OFrame of the same type as x

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv"
iris <- h2o.importFile(f)
h2o.rep_len(iris, length.out = 3)

## End(Not run)
```

---

h2o.reset_threshold *Reset model threshold and return old threshold value.*

Description

Reset model threshold and return old threshold value.

Usage

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

Arguments

object	An H2OModel object.
threshold	A threshold value from 0 to 1 included.

Value

Returns the previous threshold used in the model.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
old_threshold <- h2o.reset_threshold(prostate_glm, 0.9)

## End(Not run)
```

`h2o.residual_analysis_plot`
Residual Analysis

Description

Do Residual Analysis and plot the fitted values vs residuals on a test dataset. Ideally, residuals should be randomly distributed. Patterns in this plot can indicate potential problems with the model selection, e.g., using simpler model than necessary, not accounting for heteroscedasticity, autocorrelation, etc. If you notice "striped" lines of residuals, that is just an indication that your response variable was integer valued instead of real valued.

Usage

```
h2o.residual_analysis_plot(model, newdata)
```

Arguments

`model`	An H2OModel.
`newdata`	An H2OFrame. Used to calculate residuals.

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)
```

```
# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the residual analysis plot
residual_analysis_plot <- h2o.residual_analysis_plot(gbm, test)
print(residual_analysis_plot)

## End(Not run)
```

h2o.residual_deviance *Retrieve the residual deviance*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.residual_deviance(prostate_glm, train = TRUE)

## End(Not run)
```

`h2o.residual_dof` *Retrieve the residual degrees of freedom*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.residual_dof(prostate_glm, train = TRUE)

## End(Not run)
```

`h2o.resume` *Triggers auto-recovery resume - this will look into configured recovery dir and resume and tasks that were interrupted by unexpected cluster stopping.*

Description

Triggers auto-recovery resume - this will look into configured recovery dir and resume and tasks that were interrupted by unexpected cluster stopping.

Usage

```
h2o.resume(recovery_dir = NULL)
```

Arguments

`recovery_dir`	A `character` path to where cluster recovery data is stored, if blank, will use cluster's configuration.

`h2o.resumeGrid` *Resume previously stopped grid training.*

Description

Resume previously stopped grid training.

Usage

```
h2o.resumeGrid(grid_id, recovery_dir = NULL, ...)
```

Arguments

`grid_id`	ID of existing grid search
`recovery_dir`	When specified the grid and all necessary data (frames, models) will be saved to this directory (use HDFS or other distributed file-system). Should the cluster crash during training, the grid can be reloaded from this directory via `h2o.loadGrid` and training can be resumed
`...`	Additional parameters to modify the resumed Grid.

`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, cascade = TRUE)
```

Arguments

`ids`	The object or hex key associated with the object to be removed or a vector/list of those things.
`cascade`	Boolean, if set to TRUE (default), the object dependencies (e.g. submodels) are also removed.

See Also

`h2o.assign`, `h2o.ls`

Examples

```
## Not run:
library(h2o)
h2o.init()
iris <- as.h2o(iris)
model <- h2o.glm(1:4,5,training = iris, family = "multinomial")
h2o.rm(iris)

## End(Not run)
```

`h2o.rmse` *Retrieves Root Mean Squared Error Value*

Description

Retrieves the root mean squared error value from an H2OModelMetrics object. If "train", "valid", and "xval" parameters are FALSE (default), then the training RMSEvalue is returned. If more than one parameter is set to TRUE, then a named vector of RMSEs are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

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

Details

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

See Also

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

Examples

```
## Not run:
library(h2o)
h2o.init()

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

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

```
perf <- h2o.performance(model, prostate)
h2o.rmse(perf)

## End(Not run)
```

`h2o.rmsle` *Retrieve the Root Mean Squared Log Error*

Description

Retrieves the root mean squared log error (RMSLE) value from an H2O model. If "train", "valid", and "xval" parameters are FALSE (default), then the training rmsle value is returned. If more than one parameter is set to TRUE, then a named vector of rmsles are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

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

Examples

```
## Not run:
library(h2o)

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

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

h2o.rmsle(m)

## End(Not run)
```

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, `round()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

h2o.round(heart["age"], digits = 3)

## End(Not run)
```

h2o.rstrip — *Strip set from right*

Description

Return a copy of the target column with trailing 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

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_rstrip <- as.h2o("1234567890")
rstrip_string <- h2o.rstrip(string_to_rstrip, "890") #Remove "890"

## End(Not run)
```

---

h2o.rulefit *Build a RuleFit Model*

Description

Builds a Distributed RuleFit model on a parsed dataset, for regression or classification.

Usage

```
h2o.rulefit(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  seed = -1,
  algorithm = c("AUTO", "DRF", "GBM"),
  min_rule_length = 3,
  max_rule_length = 3,
  max_num_rules = -1,
  model_type = c("rules_and_linear", "rules", "linear"),
  weights_column = NULL,
 distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
    "tweedie", "laplace", "quantile", "huber"),
  rule_generation_ntrees = 50,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  remove_duplicates = TRUE,
  lambda = NULL,
  max_categorical_levels = 10
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`algorithm`	The algorithm to use to generate rules. Must be one of: "AUTO", "DRF", "GBM". Defaults to AUTO.
`min_rule_length`	Minimum length of rules. Defaults to 3.
`max_rule_length`	Maximum length of rules. Defaults to 3.
`max_num_rules`	The maximum number of rules to return. defaults to -1 which means the number of rules is selected by diminishing returns in model deviance. Defaults to -1.
`model_type`	Specifies type of base learners in the ensemble. Must be one of: "rules_and_linear", "rules", "linear". Defaults to rules_and_linear.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.
`distribution`	Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.
`rule_generation_ntrees`	Specifies the number of trees to build in the tree model. Defaults to 50. Defaults to 50.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
`remove_duplicates`	`Logical`. Whether to remove rules which are identical to an earlier rule. Defaults to true. Defaults to TRUE.
`lambda`	Lambda for LASSO regressor.
`max_categorical_levels`	For every categorical feature, only use this many most frequent categorical levels for model training. Only used for categorical_encoding == EnumLimited. Defaults to 10.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the titanic dataset:
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/titanic.csv"
coltypes <- list(by.col.name = c("pclass", "survived"), types=c("Enum", "Enum"))
df <- h2o.importFile(f, col.types = coltypes)

# Split the dataset into train and test
splits <- h2o.splitFrame(data = df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Set the predictors and response; set the factors:
response <- "survived"
predictors <- c("age", "sibsp", "parch", "fare", "sex", "pclass")

# Build and train the model:
rfit <- h2o.rulefit(y = response,
                    x = predictors,
                    training_frame = train,
                    max_rule_length = 10,
                    max_num_rules = 100,
                    seed = 1)

# Retrieve the rule importance:
print(rfit@model$rule_importance)

# Predict on the test data:
h2o.predict(rfit, newdata = test)

## End(Not run)
```

`h2o.rule_importance` *This function returns the table with estimated coefficients and language representations (in case it is a rule) for each of the significant baselearners.*

Description

This function returns the table with estimated coefficients and language representations (in case it is a rule) for each of the significant baselearners.

Usage

```
h2o.rule_importance(model)
```

Arguments

`model`	of the interest

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path)
s <- h2o.runif(prostate)
summary(s)

prostate_train <- prostate[s <= 0.8,]
prostate_test <- prostate[s > 0.8,]
nrow(prostate_train) + nrow(prostate_test)

## End(Not run)
```

h2o.saveGrid *Saves an existing Grid of models into a given folder.*

Description

Returns a reference to the saved Grid.

Usage

```
h2o.saveGrid(
  grid_directory,
  grid_id,
  save_params_references = FALSE,
  export_cross_validation_predictions = FALSE
)
```

Arguments

grid_directory	A character string containing the path to the folder for the grid to be saved to.
grid_id	A chracter string with identification of the grid to be saved.
save_params_references	A logical indicating if objects referenced by grid parameters (e.g. training frame, calibration frame) should also be saved.
export_cross_validation_predictions	A logical indicating whether exported model artifacts should also include CV holdout Frame predictions.

Value

Returns an object that is a subclass of H2OGrid.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris <- as.h2o(iris)

ntrees_opts = c(1, 5)
learn_rate_opts = c(0.1, 0.01)
size_of_hyper_space = length(ntrees_opts) * length(learn_rate_opts)

hyper_parameters = list(ntrees = ntrees_opts, learn_rate = learn_rate_opts)
# Tempdir is chosen arbitrarily. May be any valid folder on an H2O-supported filesystem.
baseline_grid <- h2o.grid(algorithm = "gbm",
                         grid_id = "gbm_grid_test",
                         x = 1:4,
                         y = 5,
                         training_frame = iris,
                         hyper_params = hyper_parameters)

grid_path <- h2o.saveGrid(grid_directory = tempdir(), grid_id = baseline_grid@grid_id)
# Remove everything from the cluster or restart it
h2o.removeAll()
grid <- h2o.loadGrid(grid_path)

## End(Not run)
```

---

h2o.saveModel *Save an H2O Model Object to Disk*

Description

Save an H2OModel to disk. (Note that ensemble binary models can be saved.)

Usage

```
h2o.saveModel(
  object,
  path = "",
  force = FALSE,
  export_cross_validation_predictions = FALSE,
  filename = ""
)
```

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.
`export_cross_validation_predictions`	logical, indicates whether the exported model artifacts should also include CV Holdout Frame predictions. Default is not to export the predictions.
`filename`	string indicating the file name.

Details

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

The owner of the file saved is the user by which H2O cluster was executed.

See Also

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

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.importFile(path = paste("https://raw.github.com",
#    "h2oai/h2o-2/master/smalldata/logreg/prostate.csv", sep = "/"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#    training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveModel(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)
```

`h2o.saveModelDetails` *Save an H2O Model Details*

Description

Save Model Details of an H2O Model in JSON Format

Usage

```
h2o.saveModelDetails(object, path = "", force = FALSE, filename = "")
```

Arguments

`object`	an H2OModel object.
`path`	string indicating the directory the model details will be written to.
`force`	logical, indicates how to deal with files that already exist.
`filename`	string indicating the file name. (Type of file is always .json)

Details

Model Details will download as a JSON file. In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package = "h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#                         training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveModelDetails(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)
```

`h2o.saveMojo` *Deprecated - use h2o.save_mojo instead. Save an H2O Model Object as Mojo to Disk*

Description

Save an MOJO (Model Object, Optimized) to disk.

Usage

```
h2o.saveMojo(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

MOJO will download as a zip file. In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

See Also

`h2o.saveModel` for saving a model to disk as a binary object.

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#                          training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveMojo(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)
```

`h2o.save_frame` *Store frame data in H2O's native format.*

Description

Store frame data in H2O's native format.

Usage

```
h2o.save_frame(x, dir, force = TRUE)
```

Arguments

`x`	An H2OFrame object
`dir`	a filesystem location where to write frame data (hdfs, nfs)
`force`	`logical`. overwrite already existing files (defaults to true)

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
h2o.save_frame(prostate, "/tmp/prostate")

## End(Not run)
```

`h2o.save_mojo` *Save an H2O Model Object as Mojo to Disk*

Description

Save an MOJO (Model Object, Optimized) to disk.

Usage

```
h2o.save_mojo(object, path = "", force = FALSE, filename = "")
```

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.
`filename`	string indicating the file name. (Type of file is always .zip)

Details

MOJO will download as a zip file. In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

See Also

`h2o.saveModel` for saving a model to disk as a binary object.

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
#                         training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.save_mojo(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)
```

`h2o.save_to_hive` *Save contents of this data frame into a Hive table*

Description

For example, h2o.save_to_hive(data_frame, "jdbc:hive2://host:10000/database", "table_name") h2o.save_to_hive(data_f "jdbc:hive2://host:10000/", "database.table_name", format = "parquet")

Usage

```
h2o.save_to_hive(
  data,
  jdbc_url,
  table_name,
  format = "csv",
  table_path = NULL,
  tmp_path = NULL
)
```

Arguments

`data`	A H2O Frame object to be saved.
`jdbc_url`	Hive JDBC connection URL.
`table_name`	Table name into which to store the data. The table must not exist as it will be created
`format`	Storage format of created Hive table. (default csv, can be csv or parquet)
`table_path`	If specified, the table will be created as an external table and this is where the data
`tmp_path`	Path where to store temporary data.

`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, inplace = FALSE)
```

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.
`inplace`	a `logical` values indicating whether directly overwrite original data (disabled by default). Exposed for backwards compatibility (prior versions of this functions were always doing an inplace update).

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
summary(iris_hf)

# Scale and center all the numeric columns in iris data set
iris_scaled <- h2o.scale(iris_hf[, 1:4])

## End(Not run)
```

h2o.scoreHistory *Retrieve Model Score History*

Description

Retrieve Model Score History

Usage

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

Arguments

object	An H2OModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement", "power", "weight", "acceleration", "year")
response <- "economy_20mpg"
cars_split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars_split[[1]]
valid <- cars_split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
                    training_frame = train,
                    validation_frame = valid,
                    seed = 1234)
h2o.scoreHistory(cars_gbm)

## End(Not run)
```

h2o.scoreHistoryGAM *Retrieve GLM Model Score History buried in GAM model*

Description

Retrieve GLM Model Score History buried in GAM model

Usage

```
h2o.scoreHistoryGAM(object)
```

Arguments

object	An H2OModel object.

`h2o.screeplot` *Scree Plot*

Description

Scree Plot

Usage

```
h2o.screeplot(model, type = c("barplot", "lines"))
```

Arguments

`model`	A PCA model
`type`	Type of the plot. Either "barplot" or "lines".

`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

```
## Not run:
library(h2o)
h2o.init()

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

## End(Not run)
```

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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
predictors <- c("displacement", "power", "weight", "acceleration", "year")
cars_pca <- h2o.prcomp(cars, transform = "STANDARDIZE",
                       k = 3, x = predictors, seed = 12345)
h2o.sdev(cars_pca)

## End(Not run)
```

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 (change of the levels will also affect all the frames that are referencing this column). If you want to make a copy of the column instead, use parameter in.place = FALSE.

Usage

```
h2o.setLevels(x, levels, in.place = TRUE)
```

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.
in.place	Indicates whether new domain will be directly applied to the column (in place change) or if a copy of the column will be created with the given domain levels.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
new_levels <- c("setosa", "versicolor", "caroliniana")
iris_hf$Species <- h2o.setLevels(iris_hf$Species, new_levels, in.place = FALSE)
h2o.levels(iris_hf$Species)

## End(Not run)
```

---

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

Description

Set the Time Zone on the H2O cluster

Usage

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

Arguments

tz	The desired timezone.

Examples

```
## Not run:
library(h2o)
h2o.init()

h2o.setTimezone("America/Juneau")
h2o.getTimezone()

## End(Not run)
```

---

h2o.set_s3_credentials *Creates a new Amazon S3 client internally with specified credentials.*

Description

There are no validations done to the credentials. Incorrect credentials are thus revealed with first S3 import call.

Usage

```
h2o.set_s3_credentials(secretKeyId, secretAccessKey, sessionToken = NULL)
```

Arguments

`secretKeyId`	Amazon S3 Secret Key ID (provided by Amazon)
`secretAccessKey`	Amazon S3 Secret Access Key (provided by Amazon)
`sessionToken`	Amazon Session Token (optional, only when using AWS Temporary Credentials)

`h2o.shap_explain_row_plot`
SHAP Local Explanation

Description

SHAP explanation shows contribution of features for a given instance. The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function. H2O implements TreeSHAP which when the features are correlated, can increase contribution of a feature that had no influence on the prediction.

Usage

```
h2o.shap_explain_row_plot(
  model,
  newdata,
  row_index,
  columns = NULL,
  top_n_features = 10,
  plot_type = c("barplot", "breakdown"),
  contribution_type = c("both", "positive", "negative")
)
```

Arguments

`model`	An H2O tree-based model. This includes Random Forest, GBM and XGboost only. Must be a binary classification or regression model.
`newdata`	An H2O Frame, used to determine feature contributions.
`row_index`	Instance row index.
`columns`	List of columns or list of indices of columns to show. If specified, then the `top_n_features` parameter will be ignored.
`top_n_features`	Integer specifying the maximum number of columns to show (ranked by their contributions). When `plot_type = "barplot"`, then `top_n_features` features will be chosen for each contribution_type.
`plot_type`	Either "barplot" or "breakdown". Defaults to "barplot".
`contribution_type`	When `plot_type == "barplot"`, plot one of "negative", "positive", or "both" contributions. Defaults to "both".

Value

A ggplot2 object.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the SHAP row explanation plot
shap_explain_row_plot <- h2o.shap_explain_row_plot(gbm, test, row_index = 1)
print(shap_explain_row_plot)

## End(Not run)
```

---

h2o.shap_summary_plot *SHAP Summary Plot*

Description

SHAP summary plot shows the contribution of the features for each instance (row of data). The sum of the feature contributions and the bias term is equal to the raw prediction of the model, i.e., prediction before applying inverse link function.

Usage

```
h2o.shap_summary_plot(
  model,
  newdata,
  columns = NULL,
  top_n_features = 20,
  sample_size = 1000
)
```

Arguments

model	An H2O tree-based model. This includes Random Forest, GBM and XGboost only. Must be a binary classification or regression model.
newdata	An H2O Frame, used to determine feature contributions.

`columns`	List of columns or list of indices of columns to show. If specified, then the `top_n_features` parameter will be ignored.
`top_n_features`	Integer specifying the maximum number of columns to show (ranked by variable importance).
`sample_size`	Integer specifying the maximum number of observations to be plotted.

Value

A ggplot2 object

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
gbm <- h2o.gbm(y = response,
               training_frame = train)

# Create the SHAP summary plot
shap_summary_plot <- h2o.shap_summary_plot(gbm, test)
print(shap_summary_plot)

## End(Not run)
```

`h2o.show_progress` *Enable Progress Bar*

Description

Enable Progress Bar

Usage

```
h2o.show_progress(expr)
```

Arguments

`expr`	When specified enable progress only for the evaluation of the expr and after the evaluation return to the previous setting (default is to show the progress bar), otherwise enable it globally.

Value

Value of expr if specified, otherwise NULL.

See Also

h2o.no_progress

Examples

```
## Not run:
library(h2o)
h2o.init()
h2o.no_progress()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
iris["class"] <- as.factor(iris["class"])
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
splits <- h2o.splitFrame(iris, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]
h2o.show_progress()

iris_km <- h2o.kmeans(x = predictors,
                      training_frame = train,
                      validation_frame = valid,
                      k = 10, estimate_k = TRUE,
                      standardize = FALSE, seed = 1234)

## End(Not run)
```

`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

```
# Don't run automatically to prevent accidentally shutting down a cluster
## 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

Round for the base R implementation, `signif()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

h2o.signif(heart["age"], digits = 3)

## End(Not run)
```

`h2o.sin` *Compute the sine of x*

Description

Compute the sine of x

Usage

```
h2o.sin(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`Trig` for the base R implementation, `sin()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.sin(frame)

## End(Not run)
```

h2o.skewness *Skewness of a column*

Description

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

Usage

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

skewness.H2OFrame(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.

Value

Returns a list containing the skewness for each column (NaN for non-numeric columns).

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.skewness(prostate$AGE)

## End(Not run)
```

h2o.splitFrame *Split an H2O Data Set*

Description

Split an existing H2O data set according to user-specified ratios. The number of subsets is always 1 more than the number of given ratios. Note that this does not give an exact split. H2O is designed to be efficient on big data using a probabilistic splitting method rather than an exact split. For example, when specifying a split of 0.75/0.25, H2O will produce a test/train split with an expected value of 0.75/0.25 rather than exactly 0.75/0.25. On small datasets, the sizes of the resulting splits will deviate from the expected value more than on big data, where they will be very close to exact.

Usage

```
h2o.splitFrame(data, ratios = 0.75, destination_frames, seed = -1)
```

Arguments

`data`	An H2OFrame object, to be split.
`ratios`	A numeric value or array indicating the ratio of total rows contained in each split. Must total up to less than 1. e.g. c(0.8) for 80/20 split.
`destination_frames`	An array of frame IDs equal to the number of values specified in the ratios array, plus one.
`seed`	Random seed.

Value

Returns a list of split H2OFrames

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_split <- h2o.splitFrame(iris_hf, ratios = c(0.2, 0.5))
head(iris_split[[1]])
summary(iris_split[[1]])

## End(Not run)
```

`h2o.sqrt` *Compute the square root of x*

Description

Compute the square root of x

Usage

```
h2o.sqrt(x)
```

Arguments

`x`	An H2OFrame object.

See Also

`MathFun` for the base R implementation, `sqrt()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.sqrt(frame)

## End(Not run)
```

`h2o.stackedEnsemble` *Builds a Stacked Ensemble*

Description

Build a stacked ensemble (aka. Super Learner) using the H2O base learning algorithms specified by the user.

Usage

```
h2o.stackedEnsemble(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  blending_frame = NULL,
  base_models = list(),
 metalearner_algorithm = c("AUTO", "deeplearning", "drf", "gbm", "glm", "naivebayes",
    "xgboost"),
  metalearner_nfolds = 0,
  metalearner_fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  metalearner_fold_column = NULL,
  metalearner_params = NULL,
  metalearner_transform = c("NONE", "Logit"),
  max_runtime_secs = 0,
  weights_column = NULL,
  offset_column = NULL,
  seed = -1,
  score_training_samples = 10000,
  keep_levelone_frame = FALSE,
  export_checkpoints_dir = NULL,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO")
)
```

Arguments

`x`	(Optional). A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used. Training frame is used only to compute ensemble training metrics.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`blending_frame`	Frame used to compute the predictions that serve as the training frame for the metalearner (triggers blending mode if provided)
`base_models`	List of models or grids (or their ids) to ensemble/stack together. Grids are expanded to individual models. If not using blending frame, then models must have been cross-validated using nfolds > 1, and folds must be identical across models.
`metalearner_algorithm`	Type of algorithm to use as the metalearner. Options include 'AUTO' (GLM with non negative weights; if validation_frame is present, a lambda search is performed), 'deeplearning' (Deep Learning with default parameters), 'drf' (Random Forest with default parameters), 'gbm' (GBM with default parameters), 'glm' (GLM with default parameters), 'naivebayes' (NaiveBayes with default parameters), or 'xgboost' (if available, XGBoost with default parameters). Must be one of: "AUTO", "deeplearning", "drf", "gbm", "glm", "naivebayes", "xgboost". Defaults to AUTO.
`metalearner_nfolds`	Number of folds for K-fold cross-validation of the metalearner algorithm (0 to disable or >= 2). Defaults to 0.
`metalearner_fold_assignment`	Cross-validation fold assignment scheme for metalearner cross-validation. Defaults to AUTO (which is currently set to Random). The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified".
`metalearner_fold_column`	Column with cross-validation fold index assignment per observation for cross-validation of the metalearner.
`metalearner_params`	Parameters for metalearner algorithm
`metalearner_transform`	Transformation used for the level one frame. Must be one of: "NONE", "Logit". Defaults to NONE.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`weights_column`	Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not

allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 0.

`offset_column`	Offset column. This will be added to the combination of columns before applying the link function.
`seed`	Seed for random numbers; passed through to the metalearner algorithm. Defaults to -1 (time-based random number).
`score_training_samples`	Specify the number of training set samples for scoring. The value must be >= 0. To use all training samples, enter 0. Defaults to 10000.
`keep_levelone_frame`	`Logical`. Keep level one frame used for metalearner training. Defaults to FALSE.
`export_checkpoints_dir`	Automatically export generated models to this directory.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import a sample binary outcome train/test set
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

# Number of CV folds
nfolds <- 5

# Train & Cross-validate a GBM
my_gbm <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train,
                  distribution = "bernoulli",
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
                  nfolds = nfolds,
                  fold_assignment = "Modulo",
                  keep_cross_validation_predictions = TRUE,
                      seed = 1)

# Train & Cross-validate a RF
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 50,
                          nfolds = nfolds,
                          fold_assignment = "Modulo",
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)

# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                model_id = "my_ensemble_binomial",
                                base_models = list(my_gbm, my_rf))

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
h2o.startLogging()
australia_path = system.file("extdata", "australia.csv", package = "h2o")
australia = h2o.importFile(path = australia_path)
h2o.stopLogging()

## End(Not run)
```

`h2o.std_coef_plot` *Plot Standardized Coefficient Magnitudes*

Description

Plot a GLM model's standardized coefficient magnitudes.

Usage

```
h2o.std_coef_plot(model, num_of_features = NULL)
```

Arguments

`model`	A trained generalized linear model
`num_of_features`	The number of features to be shown in the plot

See Also

`h2o.varimp_plot` for variable importances plot of random forest, GBM, deep learning.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                         training_frame = prostate, family = "binomial",
                         nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.std_coef_plot(prostate_glm)

## End(Not run)
```

`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

```
## Not run:
library(h2o)
h2o.init()
h2o.startLogging()
australia_path = system.file("extdata", "australia.csv", package = "h2o")
australia = h2o.importFile(path = australia_path)
h2o.stopLogging()

## End(Not run)
```

`h2o.str` *Display the structure of an H2OFrame object*

Description

Display the structure of an H2OFrame object

Usage

```
h2o.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

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.str(frame, cols = FALSE)

## End(Not run)
```

h2o.stringdist *Compute element-wise string distances between two H2OFrames*

Description

Compute element-wise string distances between two H2OFrames. Both frames need to have the same shape (N x M) and only contain string/factor columns. Return a matrix (H2OFrame) of shape N x M.

Usage

```
h2o.stringdist(
  x,
  y,
  method = c("lv", "lcs", "qgram", "jaccard", "jw", "soundex"),
  compare_empty = TRUE
)
```

Arguments

x	An H2OFrame
y	A comparison H2OFrame
method	A string identifier indicating what string distance measure to use. Must be one of: "lv" - Levenshtein distance "lcs" - Longest common substring distance "qgram" - q-gram distance "jaccard" - Jaccard distance between q-gram profiles "jw" - Jaro, or Jaro-Winker distance "soundex" - Distance based on soundex encoding
compare_empty	if set to FALSE, empty strings will be handled as NaNs

Examples

```
## Not run:
h2o.init()
x <- as.h2o(c("Martha", "Dwayne", "Dixon"))
y <- as.character(as.h2o(c("Marhta", "Duane", "Dicksonx")))
h2o.stringdist(x, y, method = "jw")

## End(Not run)
```

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.

Value

An H2OFrame where each column is the outcome of the string split.

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_split <- as.h2o("Split at every character.")
split_string <- h2o.strsplit(string_to_split, "")

## End(Not run)
```

`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

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_sub <- as.h2o("r tutorial")
sub_string <- h2o.sub("r ", "H2O ", string_to_sub)

## End(Not run)
```

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.

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_substring <- as.h2o("1234567890")
substr <- h2o.substring(string_to_substring, 2) #Get substring from second index onwards

## End(Not run)
```

h2o.sum *Compute the frame's sum by-column (or by-row).*

Description

Compute the frame's sum by-column (or by-row).

Usage

```
h2o.sum(x, na.rm = FALSE, axis = 0, return_frame = FALSE)
```

Arguments

`x`	An H2OFrame object.
`na.rm`	`logical`. indicating whether missing values should be removed.
`axis`	An int that indicates whether to do down a column (0) or across a row (1). For row or column sums, the `return_frame` parameter must be TRUE.
`return_frame`	A boolean that indicates whether to return an H2O frame or one single aggregated value. Default is FALSE.

See Also

sum for the base R implementation.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.sum(frame["C1"], na.rm = TRUE, axis = 0, return_frame = TRUE)

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path)
summary(prostate)
summary(prostate$GLEASON)
summary(prostate[, 4:6])
summary(prostate, exact_quantiles = TRUE)

## End(Not run)
```

---

h2o.svd	*Singular value decomposition of an H2O data frame using the power method*

Description

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

Usage

```
h2o.svd(
  training_frame,
  x,
  destination_key,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  svd_method = c("GramSVD", "Power", "Randomized"),
  nv = 1,
  max_iterations = 1000,
  seed = -1,
  keep_u = TRUE,
  u_name = NULL,
  use_all_factor_levels = TRUE,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```

Arguments

`training_frame`	Id of the training data frame.
`x`	A vector containing the `character` names of the predictors in the model.
`destination_key`	(Optional) The unique key assigned to the resulting model. Automatically generated if none is provided.
`model_id`	Destination id for this model; auto-generated if not specified.

`validation_frame`	Id of the validation data frame.
`ignore_const_cols`	`Logical`. Ignore constant columns. Defaults to TRUE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`transform`	Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
`svd_method`	Method for computing SVD (Caution: Randomized is currently experimental and unstable) Must be one of: "GramSVD", "Power", "Randomized". Defaults to GramSVD.
`nv`	Number of right singular vectors Defaults to 1.
`max_iterations`	Maximum iterations Defaults to 1000.
`seed`	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
`keep_u`	`Logical`. Save left singular vectors? Defaults to TRUE.
`u_name`	Frame key to save left singular vectors
`use_all_factor_levels`	`Logical`. Whether first factor level is included in each categorical expansion Defaults to TRUE.
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`export_checkpoints_dir`	Automatically export generated models to this directory.

Value

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[https://arxiv.org/abs/0909.4061]. SIAM Rev., Survey and Review section, Vol. 53, num. 2, pp. 217-288, June 2011.

Examples

```
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
h2o.svd(training_frame = australia, nv = 8)

## End(Not run)
```

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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
summary(prostate)

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

# Two-way table of ages (rows) and race (cols) of all patients
head(h2o.table(prostate[, c(3, 4)]))
h2o.table(prostate[, c(3, 4)])

## End(Not run)
```

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

```
## Not run:
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)

## End(Not run)
```

h2o.tan *Compute the tangent of x*

Description

Compute the tangent of x

Usage

```
h2o.tan(x)
```

Arguments

x	An H2OFrame object.

See Also

Trig for the base R implementation, `tan()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.tan(frame)

## End(Not run)
```

h2o.tanh *Compute the hyperbolic tangent of x*

Description

Compute the hyperbolic tangent of x

Usage

```
h2o.tanh(x)
```

Arguments

x	An H2OFrame object.

See Also

Hyperbolic for the base R implementation, `tanh()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.tanh(frame)

## End(Not run)
```

h2o.targetencoder — *Transformation of a categorical variable with a mean value of the target variable*

Description

Transformation of a categorical variable with a mean value of the target variable

Usage

```
h2o.targetencoder(
  x,
  y,
  training_frame,
  model_id = NULL,
  fold_column = NULL,
  columns_to_encode = NULL,
  keep_original_categorical_columns = TRUE,
  blending = FALSE,
  inflection_point = 10,
  smoothing = 20,
 data_leakage_handling = c("leave_one_out", "k_fold", "none", "LeaveOneOut", "KFold",
    "None"),
  noise = 0.01,
  seed = -1,
  ...
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`fold_column`	Column with cross-validation fold index assignment per observation.
`columns_to_encode`	List of categorical columns or groups of categorical columns to encode. When groups of columns are specified, each group is encoded as a single column (interactions are created internally).
`keep_original_categorical_columns`	`Logical`. If true, the original non-encoded categorical features will remain in the result frame. Defaults to TRUE.
`blending`	`Logical`. If true, enables blending of posterior probabilities (computed for a given categorical value) with prior probabilities (computed on the entire set). This allows to mitigate the effect of categorical values with small cardinality.

The blending effect can be tuned using the ‘inflection_point‘ and ‘smoothing‘ parameters. Defaults to FALSE.

`inflection_point`
: Inflection point of the sigmoid used to blend probabilities (see ‘blending‘ parameter). For a given categorical value, if it appears less that ‘inflection_point‘ in a data sample, then the influence of the posterior probability will be smaller than the prior. Defaults to 10.

`smoothing`
: Smoothing factor corresponds to the inverse of the slope at the inflection point on the sigmoid used to blend probabilities (see ‘blending‘ parameter). If smoothing tends towards 0, then the sigmoid used for blending turns into a Heaviside step function. Defaults to 20.

`data_leakage_handling`
: Data leakage handling strategy used to generate the encoding. Supported options are: 1) "none" (default) - no holdout, using the entire training frame. 2) "leave_one_out" - current row's response value is subtracted from the per-level frequencies pre-calculated on the entire training frame. 3) "k_fold" - encodings for a fold are generated based on out-of-fold data. Must be one of: "leave_one_out", "k_fold", "none", "LeaveOneOut", "KFold", "None". Defaults to None.

`noise`
: The amount of noise to add to the encoded column. Use 0 to disable noise, and -1 (=AUTO) to let the algorithm determine a reasonable amount of noise. Defaults to 0.01.

`seed`
: Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

`...`
: Mainly used for backwards compatibility, to allow deprecated parameters.

Examples

```
## Not run:
library(h2o)
h2o.init()
#Import the titanic dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/titanic.csv"
titanic <- h2o.importFile(f)

# Set response as a factor
response <- "survived"
titanic[response] <- as.factor(titanic[response])

# Split the dataset into train and test
splits <- h2o.splitFrame(data = titanic, ratios = .8, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]

# Choose which columns to encode
encode_columns <- c("home.dest", "cabin", "embarked")

# Train a TE model
te_model <- h2o.targetencoder(x = encode_columns,
                              y = response,
                              training_frame = train,
                              fold_column = "pclass",
                                data_leakage_handling = "KFold")

# New target encoded train and test sets
train_te <- h2o.transform(te_model, train)
test_te <- h2o.transform(te_model, test)

## End(Not run)
```

`h2o.target_encode_apply` *Apply Target Encoding Map to Frame*

Description

Applies a target encoding map to an H2OFrame object. Computing target encoding for high cardinality categorical columns can improve performance of supervised learning models. A Target Encoding tutorial is available here: https://github.com/h2oai/h2o-tutorials/blob/master/best-practices/categorical-predictors/target_encoding.md.

Usage

```
h2o.target_encode_apply(
  data,
  x,
  y,
  target_encode_map,
  holdout_type,
  fold_column = NULL,
  blended_avg = TRUE,
  noise_level = NULL,
  seed = -1
)
```

Arguments

`data`	An H2OFrame object with which to apply the target encoding map.
`x`	A list containing the names or indices of the variables to encode. A target encoding column will be created for each element in the list. Items in the list can be multiple columns. For example, if ‘x = list(c("A"), c("B", "C"))‘, then the resulting frame will have a target encoding column for A and a target encoding column for B & C (in this case, we group by two columns).
`y`	The name or column index of the response variable in the data. The response variable can be either numeric or binary.
`target_encode_map`	A list of H2OFrame objects that is the results of the `h2o.target_encode_create` function.
`holdout_type`	The holdout type used. Must be one of: "LeaveOneOut", "KFold", "None".
`fold_column`	(Optional) The name or column index of the fold column in the data. Defaults to NULL (no ‘fold_column‘). Only required if ‘holdout_type‘ = "KFold".
`blended_avg`	`Logical`. (Optional) Whether to perform blended average.

`noise_level`	(Optional) The amount of random noise added to the target encoding. This helps prevent overfitting. Defaults to 0.01 * range of y.
`seed`	(Optional) A random seed used to generate draws from the uniform distribution for random noise. Defaults to -1.

Value

Returns an H2OFrame object containing the target encoding per record.

See Also

`h2o.target_encode_create` for creating the target encoding map

Examples

```
## Not run:
library(h2o)
h2o.init()

# Get Target Encoding Frame on bank-additional-full data with numeric `y`
data <- h2o.importFile(
  path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv")
splits <- h2o.splitFrame(data, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]
mapping <- h2o.target_encode_create(data = train, x = list(c("job"), c("job", "marital")),
                                    y = "age")

# Apply mapping to the training dataset
train_encode <- h2o.target_encode_apply(data = train, x = list(c("job"), c("job", "marital")),
                                       y = "age", mapping, holdout_type = "LeaveOneOut")
# Apply mapping to a test dataset
test_encode <- h2o.target_encode_apply(data = test, x = list(c("job"), c("job", "marital")),
                                       y = "age", target_encode_map = mapping,
                                       holdout_type = "None")


## End(Not run)
```

`h2o.target_encode_create` *Create Target Encoding Map*

Description

Creates a target encoding map based on group-by columns (‘x‘) and a numeric or binary target column (‘y‘). Computing target encoding for high cardinality categorical columns can improve performance of supervised learning models. A Target Encoding tutorial is available here: https://github.com/h2oai/h2o-tutorials/blob/master/best-practices/categorical-predictors/target_encoding.md.

Usage

```
h2o.target_encode_create(data, x, y, fold_column = NULL)
```

Arguments

`data`	An H2OFrame object with which to create the target encoding map.
`x`	A list containing the names or indices of the variables to encode. A target encoding map will be created for each element in the list. Items in the list can be multiple columns. For example, if ‘x = list(c("A"), c("B", "C"))‘, then there will be one mapping frame for A and one mapping frame for B & C (in this case, we group by two columns).
`y`	The name or column index of the response variable in the data. The response variable can be either numeric or binary.
`fold_column`	(Optional) The name or column index of the fold column in the data. Defaults to NULL (no ‘fold_column‘).

Value

Returns a list of H2OFrame objects containing the target encoding mapping for each column in ‘x‘.

See Also

`h2o.target_encode_apply` for applying the target encoding mapping to a frame.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Get Target Encoding Map on bank-additional-full data with numeric response
data <- h2o.importFile(
path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv")
mapping_age <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")),
                                        y = "age")
head(mapping_age)

# Get Target Encoding Map on bank-additional-full data with binary response
mapping_y <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")),
                                      y = "y")
head(mapping_y)


## End(Not run)
```

`h2o.tf_idf` *Computes TF-IDF values for each word in given documents.*

Description

Computes TF-IDF values for each word in given documents.

Usage

```
h2o.tf_idf(
  frame,
  document_id_col,
  text_col,
  preprocess = TRUE,
  case_sensitive = TRUE
)
```

Arguments

`frame`	documents or words frame for which TF-IDF values should be computed.
`document_id_col`	index or name of a column containing document IDs.
`text_col`	index or name of a column containing documents if 'preprocess = TRUE' or words if 'preprocess = FALSE'.
`preprocess`	whether input text data should be pre-processed. Defaults to 'TRUE'.
`case_sensitive`	whether input data should be treated as case sensitive. Defaults to 'TRUE'.

Value

resulting frame with TF-IDF values. Row format: documentID, word, TF, IDF, TF-IDF

`h2o.thresholds_and_metric_scores` *Retrieve the thresholds and metric scores table*

Description

Retrieves the thresholds and metric scores table from an H2OBinomialUpliftMetrics. The table contains indices, thresholds, all cumulative uplift values and cumulative number of observations. If "train" and "valid" parameters are FALSE (default), then the training table is returned. If more than one parameter is set to TRUE, then a named vector of tables is returned, where the names are "train", "valid".

Usage

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

Arguments

`object`	An H2OBinomialUpliftMetrics
`train`	Retrieve the training thresholds and metric scores table
`valid`	Retrieve the validation thresholds and metric scores table

Examples

```
## Not run:
library(h2o)
h2o.init()
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/uplift/criteo_uplift_13k.csv"
train <- h2o.importFile(f)
train$treatment <- as.factor(train$treatment)
train$conversion <- as.factor(train$conversion)

model <- h2o.upliftRandomForest(training_frame=train, x=sprintf("f%s",seq(0:10)), y="conversion",
                                ntrees=10, max_depth=5, treatment_column="treatment",
                                auuc_type="AUTO")
perf <- h2o.performance(model, train=TRUE)
h2o.thresholds_and_metric_scores(perf)

## End(Not run)
```

h2o.toFrame *Convert a word2vec model into an H2OFrame*

Description

Converts a given word2vec model into an H2OFrame. The frame represents learned word embeddings

Usage

```
h2o.toFrame(word2vec)
```

Arguments

word2vec	A word2vec model.

Examples

```
## Not run:
h2o.init()

# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors and return average vector for each sentence
h2o.toFrame(w2v_model) # -> Frame made of 2 rows and 2 columns

## End(Not run)
```

h2o.tokenize *Tokenize String*

Description

h2o.tokenize is similar to h2o.strsplit, the difference between them is that h2o.tokenize will store the tokenized text into a single column making it easier for additional processing (filtering stop words, word2vec algo, ...).

Usage

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

Arguments

x	The column or columns whose strings to tokenize.
split	The regular expression to split on.

Value

An H2OFrame with a single column representing the tokenized Strings. Original rows of the input DF are separated by NA.

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_tokenize <- as.h2o("Split at every character and tokenize.")
tokenize_string <- h2o.tokenize(as.character(string_to_tokenize), "")

## End(Not run)
```

h2o.tolower *Convert strings to lowercase*

Description

Convert strings to lowercase

Usage

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

Arguments

x	An H2OFrame object whose strings should be lower cased

Value

An H2OFrame with all entries in lowercase format

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_lower <- as.h2o("ABCDE")
lowered_string <- h2o.tolower(string_to_lower)

## End(Not run)
```

---

h2o.topBottomN *H2O topBottomN*

Description

topBottomN function will will grab the top N percent or botom N percent of values of a column and return it in a H2OFrame.

Usage

```
h2o.topBottomN(x, column, nPercent, grabTopN)
```

Arguments

`x`	an H2OFrame
`column`	is a column name or column index to grab the top N percent value from
`nPercent`	a top percentage values to grab
`grabTopN`	if -1 grab bottom percentage, 1 grab top percentage

Value

An H2OFrame with 2 columns: first column is the original row indices, second column contains the values

---

h2o.topN *H2O topN*

Description

Extract the top N percent of values of a column and return it in a H2OFrame.

Usage

```
h2o.topN(x, column, nPercent)
```

Arguments

`x`	an H2OFrame
`column`	is a column name or column index to grab the top N percent value from
`nPercent`	is a top percentage value to grab

Value

An H2OFrame with 2 columns. The first column is the original row indices, second column contains the topN values

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/bigdata/laptop/jira/TopBottomNRep4.csv.zip"
dataset <- h2o.importFile(f)
frameNames <- names(dataset)
nPercent <- c(1, 2, 3, 4)
nP <- nPercent[sample(1:length(nPercent), 1, replace = FALSE)]
colIndex <- sample(1:length(frameNames), 1, replace = FALSE)
h2o.topN(dataset, frameNames[colIndex], nP)

## End(Not run)
```

`h2o.totss` *Get the total sum of squares.*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.totss(km, train = TRUE)

## End(Not run)
```

h2o.tot_withinss *Get the total within cluster sum of squares.*

Description

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

Examples

```
## Not run:
library(h2o)
h2o.init()

fr <- h2o.importFile("https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_train.csv")
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.tot_withinss(km, train = TRUE)

## End(Not run)
```

h2o.toupper *Convert strings to uppercase*

Description

Convert strings to uppercase

Usage

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

Arguments

`x`	An H2OFrame object whose strings should be upper cased

Value

An H2OFrame with all entries in uppercase format

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_upper <- as.h2o("abcde")
upper_string <- h2o.toupper(string_to_upper)

## End(Not run)
```

---

h2o.train_segments *H2O Segmented-Data Bulk Model Training*

Description

Provides a set of functions to train a group of models on different segments (subpopulations) of the training set.

Usage

```
h2o.train_segments(
  algorithm,
  segment_columns,
  segment_models_id,
  parallelism = 1,
  ...
)
```

Arguments

`algorithm`	Name of algorithm to use in training segment models (gbm, randomForest, kmeans, glm, deeplearning, naivebayes, psvm, xgboost, pca, svd, targetencoder, aggregator, word2vec, coxph, isolationforest, kmeans, stackedensemble, glrm, gam, anovaglm, modelselection).
`segment_columns`	A list of columns to segment-by. H2O will group the training (and validation) dataset by the segment-by columns and train a separate model for each segment (group of rows).
`segment_models_id`	Identifier for the returned collection of Segment Models. If not specified it will be automatically generated.
`parallelism`	Level of parallelism of bulk model building, it is the maximum number of models each H2O node will be building in parallel, defaults to 1.
`...`	Use to pass along training_frame parameter, x, y, and all non-default parameter values to the algorithm Look at the specific algorithm - h2o.gbm, h2o.glm, h2o.kmeans, h2o.deepLearning - for available parameters.

Details

Start Segmented-Data bulk Model Training for a given algorithm and parameters.

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
models <- h2o.train_segments(algorithm = "gbm",
                             segment_columns = "Species",
                             x = c(1:3), y = 4,
                             training_frame = iris_hf,
                             ntrees = 5,
                             max_depth = 4)
as.data.frame(models)

## End(Not run)
```

`h2o.transform` *Use H2O Transformation model and apply the underlying transformation*

Description

Use H2O Transformation model and apply the underlying transformation

Usage

```
h2o.transform(model, ...)
```

Arguments

`model`	A trained model representing the transformation strategy
`...`	Transformation model-specific parameters

Value

Returns an H2OFrame object with data transformed.

`h2o.transform,H2OTargetEncoderModel-method`
Applies target encoding to a given dataset

Description

Applies target encoding to a given dataset

Usage

```
## S4 method for signature 'H2OTargetEncoderModel'
h2o.transform(
  model,
  data,
  blending = NULL,
  inflection_point = -1,
  smoothing = -1,
  noise = NULL,
  as_training = FALSE,
  ...
)
```

Arguments

`model`	A trained model representing the transformation strategy
`data`	An H2OFrame with data to be transformed
`blending`	Use blending during the transformation. Respects model settings when not set.
`inflection_point`	Blending parameter. Only effective when blending is enabled. By default, model settings are respected, if not overridden by this setting.
`smoothing`	Blending parameter. Only effective when blending is enabled. By default, model settings are respected, if not overridden by this setting.
`noise`	An amount of random noise added to the encoding, this helps prevent overfitting. By default, model settings are respected, if not overridden by this setting.
`as_training`	Must be set to True when encoding the training frame. Defaults to False.
`...`	Mainly used for backwards compatibility, to allow deprecated parameters.

Value

Returns an H2OFrame object with data transformed.

`h2o.transform,H2OWordEmbeddingModel-method` *Transform words (or sequences of words) to vectors using a word2vec model.*

Description

Transform words (or sequences of words) to vectors using a word2vec model.

Usage

```
## S4 method for signature 'H2OWordEmbeddingModel'
h2o.transform(model, words, aggregate_method = c("NONE", "AVERAGE"))
```

Arguments

`model`	A word2vec model.
`words`	An H2OFrame made of a single column containing source words.
`aggregate_method`	Specifies how to aggregate sequences of words. If method is 'NONE' then no aggregation is performed and each input word is mapped to a single word-vector. If method is 'AVERAGE' then input is treated as sequences of words delimited by NA. Each word of a sequences is internally mapped to a vector and vectors belonging to the same sentence are averaged and returned in the result.

Examples

```
## Not run:
h2o.init()

# Build a simple word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors without aggregation
sentences <- as.character(as.h2o(c("b", "c", "a", NA, "b")))
h2o.transform(w2v_model, sentences) # -> 5 rows total, 2 rows NA ("c" is not in the vocabulary)

# Transform words to vectors and return average vector for each sentence
h2o.transform(w2v_model, sentences, aggregate_method = "AVERAGE") # -> 2 rows

## End(Not run)
```

`h2o.transform_word2vec` *Transform words (or sequences of words) to vectors using a word2vec model.*

Description

Transform words (or sequences of words) to vectors using a word2vec model.

Usage

```
h2o.transform_word2vec(
  word2vec,
  words,
  aggregate_method = c("NONE", "AVERAGE")
)
```

Arguments

`word2vec`	A word2vec model.
`words`	An H2OFrame made of a single column containing source words.

`aggregate_method`
: Specifies how to aggregate sequences of words. If method is 'NONE' then no aggregation is performed and each input word is mapped to a single word-vector. If method is 'AVERAGE' then input is treated as sequences of words delimited by NA. Each word of a sequences is internally mapped to a vector and vectors belonging to the same sentence are averaged and returned in the result.

Examples

```
## Not run:
h2o.init()

# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors without aggregation
sentences <- as.character(as.h2o(c("b", "c", "a", NA, "b")))
h2o.transform(w2v_model, sentences) # -> 5 rows total, 2 rows NA ("c" is not in the vocabulary)

# Transform words to vectors and return average vector for each sentence
h2o.transform(w2v_model, sentences, aggregate_method = "AVERAGE") # -> 2 rows

## End(Not run)
```

`h2o.trim` *Trim Space*

Description

Trim Space

Usage

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

Arguments

`x`	The column whose strings should be trimmed.

Examples

```
## Not run:
library(h2o)
h2o.init()
string_to_trim <- as.h2o("r tutorial")
trim_string <- h2o.trim(string_to_trim)

## End(Not run)
```

`h2o.trunc` *Truncate values in x toward 0*

Description

trunc takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0.

Usage

```
h2o.trunc(x)
```

Arguments

`x`	An H2OFrame object.

See Also

Round for the base R implementation, `trunc()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
h2o.trunc(frame["C1"])

## End(Not run)
```

`h2o.unique` *H2O Unique*

Description

Extract unique values in the column.

Usage

```
h2o.unique(x, include_nas = FALSE)
```

Arguments

`x`	An H2OFrame object.
`include_nas`	If set to TRUE, NAs are included. FALSE (turned off) by default.

Value

Returns an H2OFrame object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/iris/iris_wheader.csv"
iris <- h2o.importFile(f)
h2o.unique(iris["class"])

## End(Not run)
```

`h2o.upliftRandomForest`
Build a Uplift Random Forest model

Description

Builds a Uplift Random Forest model on an H2OFrame.

Usage

```
h2o.upliftRandomForest(
  x,
  y,
  training_frame,
  treatment_column,
  model_id = NULL,
  validation_frame = NULL,
  score_each_iteration = FALSE,
  score_tree_interval = 0,
  ignore_const_cols = TRUE,
  ntrees = 50,
  max_depth = 20,
  min_rows = 1,
  nbins = 20,
  nbins_top_level = 1024,
  nbins_cats = 1024,
  max_runtime_secs = 0,
  seed = -1,
  mtries = -2,
  sample_rate = 0.632,
  sample_rate_per_class = NULL,
  col_sample_rate_change_per_level = 1,
  col_sample_rate_per_tree = 1,
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal",
    "RoundRobin", "UniformRobust"),
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
    "tweedie", "laplace", "quantile", "huber"),
  check_constant_response = TRUE,
  uplift_metric = c("AUTO", "KL", "Euclidean", "ChiSquared"),
  auuc_type = c("AUTO", "qini", "lift", "gain"),
  auuc_nbins = -1,
  verbose = FALSE
)
```

Arguments

x	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
y	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
training_frame	Id of the training data frame.
treatment_column	Define the column which will be used for computing uplift gain to select best split for a tree. The column has to divide the dataset into treatment (value 1) and control (value 0) groups. Defaults to treatment.
model_id	Destination id for this model; auto-generated if not specified.
validation_frame	Id of the validation data frame.
score_each_iteration	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
score_tree_interval	Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
ignore_const_cols	`Logical`. Ignore constant columns. Defaults to TRUE.
ntrees	Number of trees. Defaults to 50.
max_depth	Maximum tree depth (0 for unlimited). Defaults to 20.
min_rows	Fewest allowed (weighted) observations in a leaf. Defaults to 1.
nbins	For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.
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 Defaults to 1024.
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. Defaults to 1024.
max_runtime_secs	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
seed	Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

`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 # of predictors Defaults to -2.
`sample_rate`	Row sample rate per tree (from 0.0 to 1.0) Defaults to 0.632.
`sample_rate_per_class`	A list of row sample rates per class (relative fraction for each class, from 0.0 to 1.0), for each tree
`col_sample_rate_change_per_level`	Relative change of the column sampling rate for every level (must be > 0.0 and <= 2.0) Defaults to 1.
`col_sample_rate_per_tree`	Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`histogram_type`	What type of histogram to use for finding optimal split points Must be one of: "AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin", "UniformRobust". Defaults to AUTO.
`categorical_encoding`	Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.
`distribution`	Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.
`check_constant_response`	`Logical`. Check if response column is constant. If enabled, then an exception is thrown if the response column is a constant value.If disabled, then model will train regardless of the response column being a constant value or not. Defaults to TRUE.
`uplift_metric`	Divergence metric used to find best split when building an uplift tree. Must be one of: "AUTO", "KL", "Euclidean", "ChiSquared". Defaults to AUTO.
`auuc_type`	Metric used to calculate Area Under Uplift Curve. Must be one of: "AUTO", "qini", "lift", "gain". Defaults to AUTO.
`auuc_nbins`	Number of bins to calculate Area Under Uplift Curve. Defaults to -1.
`verbose`	`Logical`. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

Value

Creates a H2OModel object of the right type.

See Also

`predict.H2OModel` for prediction

h2o.upload_model

Upload a binary model from the provided local path to the H2O cluster. (H2O model can be saved in a binary form either by saveModel() or by download_model() function.)

Description

Upload a binary model from the provided local path to the H2O cluster. (H2O model can be saved in a binary form either by saveModel() or by download_model() function.)

Usage

```
h2o.upload_model(path)
```

Arguments

`path`	A path on the machine this python session is currently connected to, specifying the location of the model to upload.

Value

Returns a new H2OModel object.

See Also

`h2o.saveModel`, `h2o.download_model`

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
# prostate = h2o.importFile(path = prostate_path)
# prostate_glm = h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#   training_frame = prostate, family = "binomial", alpha = 0.5)
# glmmodel_path = h2o.download_model(prostate_glm, dir = "/Users/UserName/Desktop")
# glmmodel_load = h2o.upload_model(glmmodel_path)

## End(Not run)
```

h2o.upload_mojo

Imports a MOJO from a local filesystem, creating a Generic model with it.

Description

Usage example: mojo_model <- h2o.upload_mojo(model_file_path = "/path/to/local/mojo.zip") predictions <- h2o.predict(mojo_model, dataset)

Usage

```
h2o.upload_mojo(mojo_local_file_path, model_id = NULL)
```

Arguments

mojo_local_file_path	Filesystem path to the model imported
model_id	Model ID, default is NULL

Value

Returns H2O Generic Model embedding given MOJO model

Examples

```
## Not run:

# Import default Iris dataset as H2O frame
data <- as.h2o(iris)

# Train a very simple GBM model
features <- c("Sepal.Length", "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
original_model <- h2o.gbm(x = features, y = "Species", training_frame = data)

# Download the trained GBM model as MOJO (temporary directory used in this example)
mojo_original_name <- h2o.download_mojo(model = original_model, path = tempdir())
mojo_original_path <- paste0(tempdir(), "/", mojo_original_name)

# Upload the MOJO from local filesystem and obtain a Generic model
mojo_model <- h2o.upload_mojo(mojo_original_path)

# Perform scoring with the generic model
predictions  <- h2o.predict(mojo_model, data)

## End(Not run)
```

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: "everything" - outputs NaNs whenever one of its contributing observations is missing "all.obs" - presence of missing observations will throw an error "complete.obs" - discards missing values along with all observations in their rows so that only complete observations are used

See Also

cor for the base R implementation, var(). h2o.sd for standard deviation.

Examples

```
## Not run:
library(h2o)
h2o.init()

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
var(prostate$AGE)

## End(Not run)
```

---

h2o.varimp   *Retrieve the variable importance.*

Description

Retrieve the variable importance.

Usage

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

Arguments

object	An H2O object.
...	Additional arguments for specific use-cases.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/prostate/prostate_complete.csv.zip"
pros <- h2o.importFile(f)
response <- "GLEASON"
```

```
predictors <- c("ID", "AGE", "CAPSULE", "DCAPS", "PSA", "VOL", "DPROS")
aml <- h2o.automl(x = predictors, y = response, training_frame = pros, max_runtime_secs = 60)

h2o.varimp(aml, top_n = 20)  # get variable importance matrix for the top 20 models

h2o.varimp(aml@leader)  # get variable importance for the leader model

## End(Not run)
```

---

h2o.varimp,H2OAutoML-method
Retrieve the variable importance.

Description

Retrieve the variable importance.

Usage

```
## S4 method for signature 'H2OAutoML'
h2o.varimp(object, top_n = 20, num_of_features = NULL)
```

Arguments

object	An H2OAutoML object.
top_n	Show at most top_n models
num_of_features	Integer specifying the number of features returned based on the maximum importance across the models. Use NULL for unlimited. Defaults to NULL.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/prostate/prostate_complete.csv.zip"
pros <- h2o.importFile(f)
response <- "GLEASON"
predictors <- c("ID", "AGE", "CAPSULE", "DCAPS", "PSA", "VOL", "DPROS")
aml <- h2o.automl(x = predictors, y = response, training_frame = pros, max_runtime_secs = 60)
h2o.varimp(aml)

## End(Not run)
```

h2o.varimp,H2OFrame-method

Retrieve the variable importance.

Description

Retrieve the variable importance.

Usage

```
## S4 method for signature 'H2OFrame'
h2o.varimp(object, num_of_features = NULL)
```

Arguments

object	A leaderboard frame.
num_of_features	Integer specifying the number of features returned based on the maximum importance across the models. Use NULL for unlimited. Defaults to NULL.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/prostate/prostate_complete.csv.zip"
pros <- h2o.importFile(f)
response <- "GLEASON"
predictors <- c("ID", "AGE", "CAPSULE", "DCAPS", "PSA", "VOL", "DPROS")
aml <- h2o.automl(x = predictors, y = response, training_frame = pros, max_runtime_secs = 60)
h2o.varimp(aml@leaderboard[1:5,])

## End(Not run)
```

h2o.varimp,H2OModel-method

Retrieve the variable importance.

Description

Retrieve the variable importance.

Usage

```
## S4 method for signature 'H2OModel'
h2o.varimp(object)
```

Arguments

object	An H2OModel object.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/prostate/prostate_complete.csv.zip"
pros <- h2o.importFile(f)
response <- "GLEASON"
predictors <- c("ID", "AGE", "CAPSULE", "DCAPS", "PSA", "VOL", "DPROS")
model <- h2o.glm(x = predictors, y = response, training_frame = pros)
h2o.varimp(model)

## End(Not run)
```

`h2o.varimp_heatmap` *Variable Importance Heatmap across multiple models*

Description

Variable importance heatmap shows variable importance across multiple models. Some models in H2O return variable importance for one-hot (binary indicator) encoded versions of categorical columns (e.g. Deep Learning, XGBoost). In order for the variable importance of categorical columns to be compared across all model types we compute a summarization of the the variable importance across all one-hot encoded features and return a single variable importance for the original categorical feature. By default, the models and variables are ordered by their similarity.

Usage

```
h2o.varimp_heatmap(object, top_n = 20, num_of_features = 20)
```

Arguments

`object`	A list of H2O models, an H2O AutoML instance, or an H2OFrame with a 'model_id' column (e.g. H2OAutoML leaderboard).
`top_n`	Integer specifying the number models shown in the heatmap (based on leaderboard ranking). Defaults to 20.
`num_of_features`	Integer specifying the number of features shown in the heatmap based on the maximum variable importance across the models. Use NULL for unlimited. Defaults to 20.

Value

A ggplot2 object.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the wine dataset into H2O:
```

```
f <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/wine/winequality-redwhite-no-BOM.csv"
df <-  h2o.importFile(f)

# Set the response
response <- "quality"

# Split the dataset into a train and test set:
splits <- h2o.splitFrame(df, ratios = 0.8, seed = 1)
train <- splits[[1]]
test <- splits[[2]]

# Build and train the model:
aml <- h2o.automl(y = response,
                  training_frame = train,
                  max_models = 10,
                  seed = 1)

# Create the variable importance heatmap
varimp_heatmap <- h2o.varimp_heatmap(aml)
print(varimp_heatmap)

## End(Not run)
```

`h2o.varimp_plot` *Plot Variable Importances*

Description

Plot Variable Importances

Usage

```
h2o.varimp_plot(model, num_of_features = NULL)
```

Arguments

`model`	A trained model (accepts a trained random forest, GBM, or deep learning model, will use `h2o.std_coef_plot` for a trained GLM
`num_of_features`	The number of features shown in the plot (default is 10 or all if less than 10).

See Also

`h2o.std_coef_plot` for GLM.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[, 2] <- as.factor(prostate[, 2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
```

```
h2o.varimp_plot(model)

# for deep learning set the variable_importance parameter to TRUE
iris_hf <- as.h2o(iris)
iris_dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris_hf,
variable_importances = TRUE)
h2o.varimp_plot(iris_dl)

## End(Not run)
```

`h2o.varsplits` *Retrieve per-variable split information for a given Isolation Forest model. Output will include: - count - The number of times a variable was used to make a split. - aggregated_split_ratios - The split ratio is defined as "abs(#left_observations - #right_observations) / #before_split". Even splits (#left_observations approx the same as #right_observations) contribute less to the total aggregated split ratio value for the given feature; highly imbalanced splits (eg. #left_observations » #right_observations) contribute more. - aggregated_split_depths - The sum of all depths of a variable used to make a split. (If a variable is used on level N of a tree, then it contributes with N to the total aggregate.)*

Description

Retrieve per-variable split information for a given Isolation Forest model. Output will include: - count - The number of times a variable was used to make a split. - aggregated_split_ratios - The split ratio is defined as "abs(#left_observations - #right_observations) / #before_split". Even splits (#left_observations approx the same as #right_observations) contribute less to the total aggregated split ratio value for the given feature; highly imbalanced splits (eg. #left_observations » #right_observations) contribute more. - aggregated_split_depths - The sum of all depths of a variable used to make a split. (If a variable is used on level N of a tree, then it contributes with N to the total aggregate.)

Usage

```
h2o.varsplits(object)
```

Arguments

`object`	An Isolation Forest model represented by 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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.week(hdf["ds9"])

## End(Not run)
```

---

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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/chicago/chicagoCensus.csv"
census <- h2o.importFile(f)
census[, 1] <- as.factor(census[, 1])
dl_model <- h2o.deeplearning(x = c(1:3), y = 4, training_frame = census,
                            hidden = c(17, 191),
                            epochs = 1,
                            balance_classes = FALSE,
                            export_weights_and_biases = TRUE)
h2o.weights(dl_model, matrix_id = 1)

## End(Not run)
```

`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.

Value

Returns an H2OFrame object.

See Also

`which` for the base R method.

Examples

```
## Not run:
library(h2o)
h2o.init()

iris_hf <- as.h2o(iris)
h2o.which(iris_hf[, 1] == 4.4)

## End(Not run)
```

h2o.which_max *Which indice contains the max value?*

Description

Get the index of the max value in a column or row

Usage

```
h2o.which_max(x, na.rm = TRUE, axis = 0)

which.max.H2OFrame(x, na.rm = TRUE, axis = 0)

which.min.H2OFrame(x, na.rm = TRUE, axis = 0)
```

Arguments

`x`	An H2OFrame object.
`na.rm`	`logical`. Indicate whether missing values should be removed.
`axis`	`integer`. Indicate whether to calculate the mean down a column (0) or across a row (1).

Value

Returns an H2OFrame object.

See Also

`which.min` for the base R method, `which.max()`.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/chicago/chicagoCensus.csv"
census <- h2o.importFile(f)
census[, 1] <- as.factor(census[, 1])
dl_model <- h2o.deeplearning(x = c(1:3), y = 4, hidden = c(17, 191),
                            epochs = 1, training_frame = census,
                            balance_classes = FALSE,
                            export_weights_and_biases = TRUE)
h2o.which_max(census["PER CAPITA INCOME "], na.rm = FALSE, axis = 0)

## End(Not run)
```

`h2o.which_min` *Which index contains the min value?*

Description

Get the index of the min value in a column or row

Usage

```
h2o.which_min(x, na.rm = TRUE, axis = 0)
```

Arguments

`x`	An H2OFrame object.
`na.rm`	`logical`. Indicate whether missing values should be removed.
`axis`	`integer`. Indicate whether to calculate the mean down a column (0) or across a row (1).

Value

Returns an H2OFrame object.

See Also

`which.min` for the base R method.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/chicago/chicagoCensus.csv"
census <- h2o.importFile(f)
dl_model <- h2o.deeplearning(x = c(1:3), y = 4, hidden = c(17, 191),
                            epochs = 1, training_frame = census,
                            balance_classes = FALSE,
                            export_weights_and_biases = TRUE)
h2o.which_min(census["PER CAPITA INCOME "], na.rm = FALSE, axis = 0)

## End(Not run)
```

h2o.withinss *Get the Within SS*

Description

Get the Within SS

Usage

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

Arguments

object	An H2OClusteringModel object.

h2o.word2vec *Trains a word2vec model on a String column of an H2O data frame*

Description

Trains a word2vec model on a String column of an H2O data frame

Usage

```
h2o.word2vec(
  training_frame = NULL,
  model_id = NULL,
  min_word_freq = 5,
  word_model = c("SkipGram", "CBOW"),
  norm_model = c("HSM"),
  vec_size = 100,
  window_size = 5,
  sent_sample_rate = 0.001,
  init_learning_rate = 0.025,
  epochs = 5,
  pre_trained = NULL,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```

Arguments

training_frame	Id of the training data frame.
model_id	Destination id for this model; auto-generated if not specified.
min_word_freq	This will discard words that appear less than <int> times Defaults to 5.
word_model	The word model to use (SkipGram or CBOW) Must be one of: "SkipGram", "CBOW". Defaults to SkipGram.
norm_model	Use Hierarchical Softmax Must be one of: "HSM". Defaults to HSM.

`vec_size`	Set size of word vectors Defaults to 100.
`window_size`	Set max skip length between words Defaults to 5.
`sent_sample_rate`	Set threshold for occurrence of words. Those that appear with higher frequency in the training data will be randomly down-sampled; useful range is (0, 1e-5) Defaults to 0.001.
`init_learning_rate`	Set the starting learning rate Defaults to 0.025.
`epochs`	Number of training iterations to run Defaults to 5.
`pre_trained`	Id of a data frame that contains a pre-trained (external) word2vec model
`max_runtime_secs`	Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
`export_checkpoints_dir`	Automatically export generated models to this directory.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the CraigslistJobTitles dataset
job_titles <- h2o.importFile(
    "https://s3.amazonaws.com/h2o-public-test-data/smalldata/craigslistJobTitles.csv",
   col.names = c("category", "jobtitle"), col.types = c("String", "String"), header = TRUE
)

# Build and train the Word2Vec model
words <- h2o.tokenize(job_titles, " ")
vec <- h2o.word2vec(training_frame = words)
h2o.findSynonyms(vec, "teacher", count = 20)

## End(Not run)
```

`h2o.xgboost` *Build an eXtreme Gradient Boosting model*

Description

Builds a eXtreme Gradient Boosting model using the native XGBoost backend.

Usage

```
h2o.xgboost(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  score_each_iteration = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  offset_column = NULL,
  weights_column = NULL,
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE",
    "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error",
    "custom", "custom_increasing"),
  stopping_tolerance = 0.001,
  max_runtime_secs = 0,
  seed = -1,
  distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma",
    "tweedie", "laplace", "quantile", "huber"),
  tweedie_power = 1.5,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  quiet_mode = TRUE,
  checkpoint = NULL,
  export_checkpoints_dir = NULL,
  ntrees = 50,
  max_depth = 6,
  min_rows = 1,
  min_child_weight = 1,
  learn_rate = 0.3,
  eta = 0.3,
  sample_rate = 1,
  subsample = 1,
  col_sample_rate = 1,
  colsample_bylevel = 1,
  col_sample_rate_per_tree = 1,
  colsample_bytree = 1,
  colsample_bynode = 1,
  max_abs_leafnode_pred = 0,
  max_delta_step = 0,
  monotone_constraints = NULL,
  interaction_constraints = NULL,
  score_tree_interval = 0,
  min_split_improvement = 0,
  gamma = 0,
  nthread = -1,
  save_matrix_directory = NULL,
  build_tree_one_node = FALSE,
  parallelize_cross_validation = TRUE,
  calibrate_model = FALSE,
  calibration_frame = NULL,
  calibration_method = c("AUTO", "PlattScaling", "IsotonicRegression"),
  max_bins = 256,
  max_leaves = 0,
  sample_type = c("uniform", "weighted"),
  normalize_type = c("tree", "forest"),
  rate_drop = 0,
  one_drop = FALSE,
  skip_drop = 0,
  tree_method = c("auto", "exact", "approx", "hist"),
  grow_policy = c("depthwise", "lossguide"),
  booster = c("gbtree", "gblinear", "dart"),
  reg_lambda = 1,
  reg_alpha = 0,
  dmatrix_type = c("auto", "dense", "sparse"),
  backend = c("auto", "gpu", "cpu"),
  gpu_id = NULL,
  gainslift_bins = -1,
  auc_type = c("AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO",
    "WEIGHTED_OVO"),
  scale_pos_weight = 1,
  eval_metric = NULL,
  score_eval_metric_only = FALSE,
  verbose = FALSE
)
```

Arguments

`x`	(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
`y`	The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
`training_frame`	Id of the training data frame.
`model_id`	Destination id for this model; auto-generated if not specified.
`validation_frame`	Id of the validation data frame.
`nfolds`	Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
`keep_cross_validation_models`	`Logical`. Whether to keep the cross-validation models. Defaults to TRUE.
`keep_cross_validation_predictions`	`Logical`. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
`keep_cross_validation_fold_assignment`	`Logical`. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
`score_each_iteration`	`Logical`. Whether to score during each iteration of model training. Defaults to FALSE.
`fold_assignment`	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for clas-

sification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.

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

`ignore_const_cols` `Logical`. Ignore constant columns. Defaults to TRUE.

`offset_column` Offset column. This will be added to the combination of columns before applying the link function.

`weights_column` Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor. If you set weight = 0 for a row, the returned prediction frame at that row is zero and this is incorrect. To get an accurate prediction, remove all rows with weight == 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 for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.

`stopping_metric` Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

`stopping_tolerance` Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

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

`seed` Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

`distribution` Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.

`tweedie_power` Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to 1.5.

`categorical_encoding` Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited". Defaults to AUTO.

`quiet_mode` `Logical`. Enable quiet mode Defaults to TRUE.

`checkpoint` Model checkpoint to resume training with.

`export_checkpoints_dir` Automatically export generated models to this directory.

`ntrees`	(same as n_estimators) Number of trees. Defaults to 50.
`max_depth`	Maximum tree depth (0 for unlimited). Defaults to 6.
`min_rows`	(same as min_child_weight) Fewest allowed (weighted) observations in a leaf. Defaults to 1.
`min_child_weight`	(same as min_rows) Fewest allowed (weighted) observations in a leaf. Defaults to 1.
`learn_rate`	(same as eta) Learning rate (from 0.0 to 1.0) Defaults to 0.3.
`eta`	(same as learn_rate) Learning rate (from 0.0 to 1.0) Defaults to 0.3.
`sample_rate`	(same as subsample) Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`subsample`	(same as sample_rate) Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`col_sample_rate`	(same as colsample_bylevel) Column sample rate (from 0.0 to 1.0) Defaults to 1.
`colsample_bylevel`	(same as col_sample_rate) Column sample rate (from 0.0 to 1.0) Defaults to 1.
`col_sample_rate_per_tree`	(same as colsample_bytree) Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`colsample_bytree`	(same as col_sample_rate_per_tree) Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
`colsample_bynode`	Column sample rate per tree node (from 0.0 to 1.0) Defaults to 1.
`max_abs_leafnode_pred`	(same as max_delta_step) Maximum absolute value of a leaf node prediction Defaults to 0.0.
`max_delta_step`	(same as max_abs_leafnode_pred) Maximum absolute value of a leaf node prediction Defaults to 0.0.
`monotone_constraints`	A mapping representing monotonic constraints. Use +1 to enforce an increasing constraint and -1 to specify a decreasing constraint.
`interaction_constraints`	A set of allowed column interactions.
`score_tree_interval`	Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
`min_split_improvement`	(same as gamma) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
`gamma`	(same as min_split_improvement) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
`nthread`	Number of parallel threads that can be used to run XGBoost. Cannot exceed H2O cluster limits (-nthreads parameter). Defaults to maximum available Defaults to -1.
`save_matrix_directory`	Directory where to save matrices passed to XGBoost library. Useful for debugging.

`build_tree_one_node`	`Logical`. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
`parallelize_cross_validation`	`Logical`. Allow parallel training of cross-validation models Defaults to TRUE.
`calibrate_model`	`Logical`. Use Platt Scaling (default) or Isotonic Regression to calculate calibrated class probabilities. Calibration can provide more accurate estimates of class probabilities. Defaults to FALSE.
`calibration_frame`	Data for model calibration
`calibration_method`	Calibration method to use Must be one of: "AUTO", "PlattScaling", "IsotonicRegression". Defaults to AUTO.
`max_bins`	For tree_method=hist only: maximum number of bins Defaults to 256.
`max_leaves`	For tree_method=hist only: maximum number of leaves Defaults to 0.
`sample_type`	For booster=dart only: sample_type Must be one of: "uniform", "weighted". Defaults to uniform.
`normalize_type`	For booster=dart only: normalize_type Must be one of: "tree", "forest". Defaults to tree.
`rate_drop`	For booster=dart only: rate_drop (0..1) Defaults to 0.0.
`one_drop`	`Logical`. For booster=dart only: one_drop Defaults to FALSE.
`skip_drop`	For booster=dart only: skip_drop (0..1) Defaults to 0.0.
`tree_method`	Tree method Must be one of: "auto", "exact", "approx", "hist". Defaults to auto.
`grow_policy`	Grow policy - depthwise is standard GBM, lossguide is LightGBM Must be one of: "depthwise", "lossguide". Defaults to depthwise.
`booster`	Booster type Must be one of: "gbtree", "gblinear", "dart". Defaults to gbtree.
`reg_lambda`	L2 regularization Defaults to 1.0.
`reg_alpha`	L1 regularization Defaults to 0.0.
`dmatrix_type`	Type of DMatrix. For sparse, NAs and 0 are treated equally. Must be one of: "auto", "dense", "sparse". Defaults to auto.
`backend`	Backend. By default (auto), a GPU is used if available. Must be one of: "auto", "gpu", "cpu". Defaults to auto.
`gpu_id`	Which GPU(s) to use.
`gainslift_bins`	Gains/Lift table number of bins. 0 means disabled.. Default value -1 means automatic binning. Defaults to -1.
`auc_type`	Set default multinomial AUC type. Must be one of: "AUTO", "NONE", "MACRO_OVR", "WEIGHTED_OVR", "MACRO_OVO", "WEIGHTED_OVO". Defaults to AUTO.
`scale_pos_weight`	Controls the effect of observations with positive labels in relation to the observations with negative labels on gradient calculation. Useful for imbalanced problems. Defaults to 1.0.
`eval_metric`	Specification of evaluation metric that will be passed to the native XGBoost backend.
`score_eval_metric_only`	`Logical`. If enabled, score only the evaluation metric. This can make model training faster if scoring is frequent (eg. each iteration). Defaults to FALSE.
`verbose`	`Logical`. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

Examples

```
## Not run:
library(h2o)
h2o.init()

# Import the titanic dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/titanic.csv"
titanic <- h2o.importFile(f)

# Set predictors and response; set response as a factor
titanic['survived'] <- as.factor(titanic['survived'])
predictors <- setdiff(colnames(titanic), colnames(titanic)[2:3])
response <- "survived"

# Split the dataset into train and valid
splits <- h2o.splitFrame(data =  titanic, ratios = .8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]

# Train the XGB model
titanic_xgb <- h2o.xgboost(x = predictors, y = response,
                           training_frame = train, validation_frame = valid,
                           booster = "dart", normalize_type = "tree",
                           seed = 1234)

## End(Not run)
```

h2o.xgboost.available *Determines whether an XGBoost model can be built*

Description

Ask the H2O server whether a XGBoost model can be built. (Depends on availability of native backend.) Returns True if a XGBoost model can be built, or False otherwise.

Usage

```
h2o.xgboost.available()
```

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 containing the entries of x converted to years

See Also

h2o.month

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/v-11-eurodate.csv"
hdf <- h2o.importFile(f)
h2o.year(hdf["ds9"])

## End(Not run)
```

H2OAutoML-class *The H2OAutoML class*

Description

This class represents an H2OAutoML object

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 cluster, 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 cluster'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 cluster.

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 cluster.

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

`name` A `character` value specifying the name of the H2O cluster.

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

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

`cacert` Path to a CA bundle file with root and intermediate certificates of trusted CAs.

- `insecure` Set this to TRUE to disable SSL certificate checking.
- `username` Username to login with.
- `password` Password to login with.
- `use_spnego` Set this to TRUE to use SPNEGO authentication.
- `cookies` Cookies to add to request
- `context_path` Context path which is appended to H2O server location.
- `mutable` An `H2OConnectionMutableState` object to hold the mutable state for the H2O connection.

`H2OConnectionMutableState` *The H2OConnectionMutableState class*

Description

This class represents the mutable aspects of a connection to an H2O cluster.

Slots

- `session_id` A `character` string specifying the H2O session identifier.
- `key_count` A `integer` value specifying count for the number of keys generated for the `session_id`.

`H2OCoxPHModel-class` *The H2OCoxPHModel object.*

Description

Virtual object representing H2O's CoxPH Model.

Usage

```
## S4 method for signature 'H2OCoxPHModel'
show(object)

## S3 method for class 'H2OCoxPHModel'
coef(object, ...)

## S3 method for class 'H2OCoxPHModel'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'H2OCoxPHModel'
logLik(object, ...)

survfit.H2OCoxPHModel(formula, newdata, ...)

## S3 method for class 'H2OCoxPHModel'
vcov(object, ...)
```

Arguments

`object`	an `H2OCoxPHModel` object.
`...`	additional arguments to pass on.
`fit`	an `H2OCoxPHModel` object.
`scale`	optional numeric specifying the scale parameter of the model.
`k`	numeric specifying the weight of the equivalent degrees of freedom.
`formula`	an `H2OCoxPHModel` object.
`newdata`	an optional `H2OFrame` or `data.frame` with the same variable names as those that appear in the `H2OCoxPHModel` object.

`H2OCoxPHModelSummary-class`
The H2OCoxPHModelSummary object.

Description

Wrapper object for summary information compatible with survival package.

Usage

```
## S4 method for signature 'H2OCoxPHModelSummary'
show(object)

## S3 method for class 'H2OCoxPHModelSummary'
coef(object, ...)
```

Arguments

`object`	An `H2OCoxPHModelSummary` object.
`...`	additional arguments to pass on.

Slots

`summary` A `list` containing the a summary compatible with CoxPH summary used in the survival package.

`H2OFrame-class` *The H2OFrame class*

Description

This class represents an H2OFrame object

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

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.

`H2OInfogram` *wrapper function for instantiating H2OInfogram*

Description

wrapper function for instantiating H2OInfogram

Usage

```
H2OInfogram(model_id, ...)
```

Arguments

`model_id`	is string of H2OModel object
`...`	parameters to algorithm, admissible_features, ...

Value

A `H2OInfogram` object

`H2OInfogram-class` *H2OInfogram class*

Description

H2OInfogram class contains a subset of what a normal H2OModel will return

Slots

`model_id` string returned as part of every `H2OModel`

`algorithm` string denoting the algorithm used to build infogram

`admissible_features` string array denoting all predictor names which pass the cmi and relelvance threshold

`admissible_features_valid` string array denoting all predictor names which pass the cmi and relelvance threshold from validation frame

`admissible_features_xval` string array denoting all predictor names which pass the cmi and relelvance threshold from cv holdout set

`net_information_threshold` numeric value denoting threshold used for predictor selection

`total_information_threshold` numeric value denoting threshold used for predictor selection

`safety_index_threshold` numeric value denoting threshold used for predictor selection

`relevance_index_threshold` numeric value denoting threshold used for predictor selection

`admissible_score` `H2OFrame` that contains columns, admissible, admissible_index, relevance, cmi, cmi_raw

`admissible_score_valid` `H2OFrame` that contains columns, admissible, admissible_index, relevance, cmi, cmi_raw from validation frame

`admissible_score_xval` `H2OFrame` that contains averages of columns, admissible, admissible_index, relevance, cmi, cmi_raw from cv hold-out

---

`H2OLeafNode-class` *The H2OLeafNode class.*

Description

This class represents a single leaf node in an `H2OTree`.

Details

#' @aliases H2OLeafNode

`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 cluster, 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 cluster'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.

`params` A `list` containing default, set, and actual parameters.

`have_pojo` A `logical` indicating whether export to POJO is supported

`have_mojo` A `logical` indicating whether export to MOJO is supported

`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 'H2OBinomialUpliftMetrics'
show(object)

## S4 method for signature 'H2OMultinomialMetrics'
show(object)

## S4 method for signature 'H2OOrdinalMetrics'
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)

## S4 method for signature 'H2OAnomalyDetectionMetrics'
show(object)
```

Arguments

`object`	An `H2OModelMetrics` object

H2ONode-class *The H2ONode class.*

Description

The H2ONode class.

Usage

```
## S4 method for signature 'H2ONode'
show(object)
```

Arguments

`object`	an `H2ONode` object.

Slots

`id` An `integer` representing node's unique identifier. Generated by H2O.

`levels` A `character` representing categorical levels on split from parent's node belonging into this node. NULL for root node or non-categorical splits.
#' @aliases H2ONode

`H2OSegmentModels-class`

H2O Segment Models

Description

A class to contain the information for segment models.

Usage

```
## S4 method for signature 'H2OSegmentModels'
show(object)
```

Arguments

`object`	an `H2OModel` object.

Slots

`segment_models_id` the identifier for the segment models collections

`H2OSegmentModelsFuture-class`

H2O Future Segment Models

Description

A class to contain the information for background segment models jobs.

Slots

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

`segment_models_id` the final identifier for the segment models collections

See Also

H2OSegmentModels for the final segment models types.

H2OSplitNode-class *The H2OSplitNode class.*

Description

This class represents a single non-terminal node in an H2OTree.

Slots

threshold A numeric split threshold, typically when the split column is numerical.

left_child A H2ONodeOrNULL representing the left child node, if a node has one.

right_child A H2ONodeOrNULL representing the right child node, if a node has one.

split_feature A character representing the name of the column this node splits on.

left_levels A character representing the levels of a categorical feature heading to the left child of this node. NA for non-categorical split.

right_levels A character representing the levels of a categorical feature heading to the right child of this node. NA for non-categorical split.

na_direction A character representing the direction of NA values. LEFT means NA values go to the left child node, RIGH means NA values go to the right child node.

H2OTree-class *The H2OTree class.*

Description

This class represents a model of a Tree built by one of H2O's algorithms (GBM, Random Forest).

Usage

```
## S4 method for signature 'H2OTree'
show(object)
```

Arguments

object	an H2OTree object.

Slots

root_node A H2ONode representing the beginning of the tree behind the model. Allows further tree traversal.

left_children An integer vector with left child nodes of tree's nodes

right_children An integer vector with right child nodes of tree's nodes

node_ids An integer representing identification number of a node. Node IDs are generated by H2O.

descriptions A character vector with descriptions for each node to be found in the tree. Contains split threshold if the split is based on numerical column. For cactegorical splits, it contains list of categorical levels for transition from the parent node.

`model_id` A `character` with the name of the model this tree is related to.

`tree_number` An `integer` representing the order in which the tree has been built in the model.

`tree_class` A `character` representing name of tree's class. Number of tree classes equals to the number of levels in categorical response column. As there is exactly one class per categorical level, name of tree's class equals to the corresponding categorical level of response column. In case of regression and binomial, the name of the categorical level is ignored can be omitted, as there is exactly one tree built in both cases.

`thresholds` A `numeric` split thresholds. Split thresholds are not only related to numerical splits, but might be present in case of categorical split as well.

`features` A `character` with names of the feature/column used for the split.

`levels` A `character` representing categorical levels on split from parent's node belonging into this node. NULL for root node or non-categorical splits.

`nas` A `character` representing if NA values go to the left node or right node. May be NA if node is a leaf.

`predictions` A `numeric` representing predictions for each node in the graph.

`tree_decision_path` A `character`, plain language rules representation of a trained decision tree

`decision_paths` A `character` representing plain language rules that were used in a particular prediction.

`left_cat_split` A `character` list of categorical levels leading to the left child node. Only present when split is categorical, otherwise none.

`right_cat_split` A `character` list of categorical levels leading to the right child node. Only present when split is categorical, otherwise none.

`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 [https://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

initialize,H2OInfogram-method — *Method on* H2OInfogram *object which in this case is to instantiate and initialize it*

Description

Method on H2OInfogram object which in this case is to instantiate and initialize it

Usage

```
## S4 method for signature 'H2OInfogram'
initialize(.Object, model_id, ...)
```

Arguments

.Object	An H2OInfogram object
model_id	string returned as part of every H2OModel
...	additional arguments to pass on

Value

A H2OInfogram object

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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

heart["transplant"] <- as.character(heart["transplant"])
is.character(heart["transplant"])

## End(Not run)
```

is.factor *Check if factor*

Description

Check if factor

Usage

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

Arguments

x	An H2OFrame object

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
is.factor(cars["economy_20mpg"])

## End(Not run)
```

---

is.h2o *Is H2O Frame object*

Description

Test if object is H2O Frame.

Usage

```
is.h2o(x)
```

Arguments

x	An R object.

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
is.h2o(frame)

## End(Not run)
```

---

is.numeric *Check if numeric*

Description

Check if numeric

Usage

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

Arguments

`x`	An H2OFrame object

`Keyed-class` *Virtual Keyed class*

Description

Base class for all objects having a persistent representation on backend.

`length,H2OTree-method` *Overrides the behavior of length() function on H2OTree class. Returns number of nodes in an* `H2OTree`

Description

Overrides the behavior of length() function on H2OTree class. Returns number of nodes in an `H2OTree`

Usage

```
## S4 method for signature 'H2OTree'
length(x)
```

Arguments

`x`	An `H2OTree` to count nodes for.

`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.

`model_cache-class` *Needed to be able to memoise the models*

Description

Needed to be able to memoise the models

`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

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
names(frame)

## End(Not run)
```

`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,H2OParetoFront-method` *Plot Pareto front*

Description

Plot Pareto front

Usage

```
## S4 method for signature 'H2OParetoFront'
plot(x, y, ...)
```

Arguments

`x`	`H2OParetoFront` object
`y`	missing
`...`	unused

plot.H2OInfogram *Plot an H2O Infogram*

Description

Plots the Infogram for an H2OInfogram object.

Usage

```
## S3 method for class 'H2OInfogram'
plot(x, ...)
```

Arguments

`x`	A fitted H2OInfogram object.
`...`	additional arguments to pass on.

Value

A ggplot2 object.

See Also

`h2o.infogram`

Examples

```
## Not run:
h2o.init()

# Convert iris dataset to an H2OFrame
train <- as.h2o(iris)

# Create and plot infogram
ig <- h2o.infogram(y = "Species", training_frame = train)
plot(ig)


## End(Not run)
```

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

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

Examples

```
## Not run:
if (requireNamespace("mlbench", quietly=TRUE)) {
    library(h2o)
    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 = "rmse")
    plot(gbm, timestep = "number_of_trees", metric = "mae")
}

## End(Not run)
```

`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. This function requires suggested ggplot2 package.

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

`h2o.tabulate`

Examples

```
## Not run:
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)

## End(Not run)
```

`predict.H2OAutoML` *Predict on an AutoML object*

Description

Obtains predictions from an AutoML object.

Usage

```
## S3 method for class 'H2OAutoML'
predict(object, newdata, ...)

## S3 method for class 'H2OAutoML'
h2o.predict(object, newdata, ...)
```

Arguments

`object`	a fitted H2OAutoML 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 generated predictions on the leader model from an AutoML run. 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.

`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, ...)

## S3 method for class 'H2OModel'
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.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/glm_test/insurance.csv"
insurance <- h2o.importFile(f)
predictors <- colnames(insurance)[1:4]
response <- "Claims"
insurance['Group'] <- as.factor(insurance['Group'])
insurance['Age'] <- as.factor(insurance['Age'])
splits <- h2o.splitFrame(data =  insurance, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]
insurance_gbm <- h2o.gbm(x = predictors, y = response,
                         training_frame = train,
                         validation_frame = valid,
                         distribution = "huber",
                         huber_alpha = 0.9, seed = 1234)
h2o.predict(insurance_gbm, newdata = insurance)

## End(Not run)
```

---

predict_contributions.H2OModel

Predict feature contributions - SHAP values on an H2O Model (only DRF, GBM, XGBoost models and equivalent imported MOJOs).

Description

Default implemntation return H2OFrame shape (#rows, #features + 1) - there is a feature contribution column for each input feature, the last column is the model bias (same value for each row). The sum of the feature contributions and the bias term is equal to the raw prediction of the model. Raw prediction of tree-based model is the sum of the predictions of the individual trees before the inverse link function is applied to get the actual prediction. For Gaussian distribution the sum of the contributions is equal to the model prediction.

Usage

```
predict_contributions.H2OModel(
  object,
  newdata,
  output_format = c("original", "compact"),
  top_n = 0,
  bottom_n = 0,
  compare_abs = FALSE,
  ...
)

h2o.predict_contributions(
  object,
  newdata,
  output_format = c("original", "compact"),
  top_n = 0,
  bottom_n = 0,
  compare_abs = FALSE,
  ...
)
```

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.
`output_format`	Specify how to output feature contributions in XGBoost - XGBoost by default outputs contributions for 1-hot encoded features, specifying a compact output format will produce a per-feature contribution. Defaults to original.
`top_n`	Return only #top_n highest contributions + bias If top_n<0 then sort all SHAP values in descending order If top_n<0 && bottom_n<0 then sort all SHAP values in descending order
`bottom_n`	Return only #bottom_n lowest contributions + bias If top_n and bottom_n are defined together then return array of #top_n + #bottom_n + bias If bottom_n<0 then sort all SHAP values in ascending order If top_n<0 && bottom_n<0 then sort all SHAP values in descending order
`compare_abs`	True to compare absolute values of contributions
`...`	additional arguments to pass on.

Details

Note: Multinomial classification models are currently not supported.

Value

Returns an H2OFrame contain feature contributions for each input row.

See Also

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

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_gbm <- h2o.gbm(3:9, "AGE", prostate)
h2o.predict(prostate_gbm, prostate)
# Compute SHAP
h2o.predict_contributions(prostate_gbm, prostate)
# Compute SHAP and pick the top two highest
h2o.predict_contributions(prostate_gbm, prostate, top_n=2)
# Compute SHAP and pick the top two lowest
h2o.predict_contributions(prostate_gbm, prostate, bottom_n=2)
# Compute SHAP and pick the top two highest regardless of the sign
h2o.predict_contributions(prostate_gbm, prostate, top_n=2, compare_abs=TRUE)
```

```
# Compute SHAP and pick the top two lowest regardless of the sign
h2o.predict_contributions(prostate_gbm, prostate, bottom_n=2, compare_abs=TRUE)
# Compute SHAP values and show them all in descending order
h2o.predict_contributions(prostate_gbm, prostate, top_n=-1)
# Compute SHAP and pick the top two highest and top two lowest
h2o.predict_contributions(prostate_gbm, prostate, top_n=2, bottom_n=2)

## End(Not run)
```

`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,
  type = c("Path", "Node_ID"),
  ...
)

h2o.predict_leaf_node_assignment(
  object,
  newdata,
  type = c("Path", "Node_ID"),
  ...
)
```

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.
`type`	choice of either "Path" when tree paths are to be returned (default); or "Node_ID" when the output
`...`	additional arguments to pass on.

Details

For every row in the test set, return the leaf placements of the row in all the trees in the model. Placements can be represented either by paths to the leaf nodes from the tree root or by H2O's internal identifiers. 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

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
h2o.predict(prostate_gbm, prostate)
h2o.predict_leaf_node_assignment(prostate_gbm, prostate)

## End(Not run)
```

---

print.H2OFrame *Print An H2OFrame*

Description

Print An H2OFrame

Usage

```
## S3 method for class 'H2OFrame'
print(x, n = 6L, m = 200L, ...)
```

Arguments

x	An H2OFrame object
n	An (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. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).
m	An (Optional) A single integer. If positive, number of columns in x to return. If negative, all but the m first/last number of columns in x.
...	Further arguments to be passed from or to other methods.

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
print(cars, n = 8)

## End(Not run)
```

`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

Examples

```
## Not run:
library(h2o)
h2o.init()

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv"
cars <- h2o.importFile(f)
print(cars, header = TRUE)

## End(Not run)
```

`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

Examples

```
## Not run:
library(h2o)
h2o.init()

frame <- h2o.createFrame(rows = 6, cols = 2,
                         categorical_fraction = 0.0,
                         missing_fraction = 0.7,
                         seed = 123)
range(frame, na.rm = TRUE)

## End(Not run)
```

row_to_tree_assignment.H2OModel *Output row to tree assignment for the model and provided training data.*

Description

Output is frame of size nrow = nrow(original_training_data) and ncol = number_of_trees_in_model+1 in format: row_id tree_1 tree_2 tree_3 0 0 1 1 1 1 1 1 2 1 0 0 3 1 1 0 4 0 1 1 5 1 1 1 6 1 0 0 7 0 1 0 8 0 1 1 9 1 0 0

Usage

```
row_to_tree_assignment.H2OModel(object, original_training_data, ...)

h2o.row_to_tree_assignment(object, original_training_data, ...)
```

Arguments

`object`	a fitted H2OModel object
`original_training_data`	An H2OFrame object that was used for model training. Currently there is no validation of the input.
`...`	additional arguments to pass on.

Details

Where 1 in the tree_number cols means row is used in the tree and 0 means that row is not used. The structure of the output depends on sample_rate or sample_size parameter setup.

Note: Multinomial classification generate tree for each category, each tree use the same sample of the data.

Value

Returns an H2OFrame contain row to tree assignment for each tree and row.

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_gbm <- h2o.gbm(4:9, "AGE", prostate, sample_rate = 0.6)
# Get row to tree assignment
h2o.row_to_tree_assignment(prostate_gbm, prostate)

## End(Not run)
```

`scale` *Scaling and Centering of an H2OFrame*

Description

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

Usage

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

Arguments

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

Examples

```
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
summary(iris_hf)

# Scale and center all the numeric columns in iris data set
iris_scaled <- scale(iris_hf[, 1:4])

## End(Not run)
```

`show,H2OAutoML-method` *Format AutoML object in user-friendly way*

Description

Format AutoML object in user-friendly way

Usage

```
## S4 method for signature 'H2OAutoML'
show(object)
```

Arguments

`object`	an `H2OAutoML` object.

`show,H2OParetoFront-method` *Show H2OParetoFront*

Description

Show H2OParetoFront

Usage

```
## S4 method for signature 'H2OParetoFront'
show(object)
```

Arguments

`object`	`H2OParetoFront` object

staged_predict_proba.H2OModel

Predict class probabilities at each stage of an H2O Model

Description

The output structure is analogous to the output of h2o.predict_leaf_node_assignment. For each tree t and class c there will be a column Tt.Cc (eg. T3.C1 for tree 3 and class 1). The value will be the corresponding predicted probability of this class by combining the raw contributions of trees T1.Cc,..,TtCc. Binomial models build the trees just for the first class and values in columns Tx.C1 thus correspond to the the probability p0.

Usage

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

h2o.staged_predict_proba(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.

Value

Returns an H2OFrame object with predicted probability for each tree in the model.

See Also

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

Examples

```
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
h2o.predict(prostate_gbm, prostate)
h2o.staged_predict_proba(prostate_gbm, prostate)

## End(Not run)
```

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,H2OAutoML-method *Format AutoML object in user-friendly way*

Description

Format AutoML object in user-friendly way

Usage

```
## S4 method for signature 'H2OAutoML'
summary(object)
```

Arguments

object	an H2OAutoML object.

`summary,H2OCoxPHModel-method`
Summary method for H2OCoxPHModel objects

Description

Summary method for H2OCoxPHModel objects

Usage

```
## S4 method for signature 'H2OCoxPHModel'
summary(object, conf.int = 0.95, scale = 1)
```

Arguments

`object`	an `H2OCoxPHModel` object.
`conf.int`	a specification of the confidence interval.
`scale`	a scale.

`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)

use.package *Use optional package*

Description

Testing availability of optional package, its version, and extra global default. This function is used internally. It is exported and documented because user can control behavior of the function by global option.

Usage

```
use.package(
  package,
  version = "1.9.8"[package == "data.table"],
  use = getOption("h2o.use.data.table", TRUE)[package == "data.table"]
)
```

Arguments

package	character scalar name of a package that we Suggests or Enhances on.
version	character scalar required version of a package.
use	logical scalar, extra escape option, to be used as global option.

Details

We use this function to control csv read/write with optional data.table package. Currently data.table is enabled by default for some operations, to disable it set `options("h2o.use.data.table"=FALSE)`. It is possible to control just `fread` or `fwrite` with `options("h2o.fread"=FALSE, "h2o.fwrite"=FALSE)`. `h2o.fread` and `h2o.fwrite` options are not handled in this function but next to *fread* and *fwrite* calls.

See Also

as.h2o.data.frame, as.data.frame.H2OFrame

Examples

```
op <- options("h2o.use.data.table" = TRUE)
if (use.package("data.table")) {
  cat("optional package data.table 1.9.8+ is available\n")
} else {
  cat("optional package data.table 1.9.8+ is not available\n")
}
options(op)
```

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/frs/index.php?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 cluster after examples run*

Description

Shutdown H2O cluster after examples run

Examples

```
## Not run:
library(h2o)
h2o.init()
h2o.shutdown(prompt = FALSE)
Sys.sleep(3)

## End(Not run)
```

`&&` *Logical and for H2OFrames*

Description

Logical and for H2OFrames

Usage

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

Arguments

`x`	An H2OFrame object
`y`	An H2OFrame object

Index