Package 'h2o'

May 29, 2014

R topics documented:

h2o-package . . . 1
apply . . . 2
as.data.frame.H2OParsedData . . . 3
as.factor . . . 4
as.h2o . . . 4
as.matrix.H2OParsedData . . . 5
cbind.H2OParsedData . . . 6
colnames . . . 7
diff.H2OParsedData . . . 8
Extremes . . . 8
h2o.addFunction . . . 9
h2o.anyFactor . . . 10
h2o.assign . . . 11
h2o.clearLogs . . . 11
h2o.clusterInfo . . . 12
h2o.clusterStatus . . . 13
h2o.confusionMatrix . . . 14
h2o.createFrame . . . 15
h2o.cut . . . 16
h2o.ddply . . . 16
h2o.deeplearning . . . 18
h2o.downloadAllLogs . . . 21
h2o.downloadCSV . . . 22
h2o.exportFile . . . 23
h2o.exportHDFS . . . 24
h2o.gapStatistic . . . 24
h2o.gbm . . . 26
h2o.getLogPath . . . 28
h2o.glm . . . 28
h2o.hitRatio . . . 31
h2o.importFile . . . 32
h2o.importFolder . . . 33
h2o.importHDFS . . . 35

h2o.importURL . . . 37
h2o.init . . . 38
h2o.kmeans . . . 40
h2o.logAndEcho . . . 41
h2o.ls . . . 42
h2o.month . . . 43
h2o.naiveBayes . . . 43
h2o.openLog . . . 45
h2o.parseRaw . . . 46
h2o.pcr . . . 47
h2o.performance . . . 49
h2o.prcomp . . . 50
h2o.predict . . . 51
h2o.randomForest . . . 52
h2o.rm . . . 54
h2o.runif . . . 55
h2o.setLogPath . . . 56
h2o.shutdown . . . 57
h2o.SpeeDRF . . . 58
h2o.startLogging . . . 60
h2o.stopLogging . . . 60
h2o.table . . . 61
h2o.uploadFile . . . 61
h2o.year . . . 63
H2OClient-class . . . 64
H2ODeepLearningGrid-class . . . 64
H2ODeepLearningModel-class . . . 65
H2ODRFGrid-class . . . 66
H2ODRFModel-class . . . 67
H2OGBMGrid-class . . . 68
H2OGBMModel-class . . . 69
H2OGLMGrid-class . . . 70
H2OGLMGridVA-class . . . 71
H2OGLMModel-class . . . 72
H2OGLMModelVA-class . . . 73
H2OGrid-class . . . 74
H2OGridVA-class . . . 74
H2OKMeansGrid-class . . . 75
H2OKMeansModel-class . . . 76
H2OKMeansModelVA-class . . . 77
H2OModel-class . . . 78
H2OModelVA-class . . . 78
H2ONBModel-class . . . 79
H2OParsedData-class . . . 80
H2OParsedDataVA-class . . . 83
H2OPCAModel-class . . . 84
H2OPerfModel-class . . . 85
H2ORawData-class . . . 86

H2ORawDataVA-class . . . 86
H2ORFModelVA-class . . . 87
H2OSpeeDRFModel-class . . . 88
head . . . 89
ifelse . . . 90
is.factor . . . 91
levels . . . 91
mean.H2OParsedData . . . 92
nrow . . . 93
plot.H2OPerfModel . . . 93
quantile.H2OParsedData . . . 94
screeplot.H2OPCAModel . . . 95
sd . . . 96
str . . . 97
sum . . . 97
summary . . . 98
summary.H2OPCAModel . . . 99
unique.H2OParsedData . . . 100

`h2o-package` *H2O R Interface*

Description

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

Details

Package:	h2o
Type:	Package
Version:	2.5.0.1358
Date:	2014-05-15
License:	Apache License (== 2.0)
Depends:	R (>= 2.13.0), RCurl, rjson, statmod, tools, methods, utils

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

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

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

Author(s)

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

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

References

- 0xdata Homepage
- H2O Documentation
- H2O on Github

Examples

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

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

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

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

Description

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

Usage

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

Arguments

X	An H2OParsedData object.
MARGIN	The margin along wich the function should be applied
FUN	The function to be applied by H2O.
...	Optional arguments to FUN. (Currently unimplemented).

Value

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

Examples

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

---

as.data.frame.H2OParsedData

Converts a parsed H2O object to a data frame.

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

`as.factor` *Converts a column from numeric to factor*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

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

Description

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

Usage

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

Arguments

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

Details

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

Value

Converts an R object to an H2O Parsed data object.

Examples

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

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

as.matrix.H2OParsedData *Converts a parsed H2O object to a matrix.*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

cbind.H2OParsedData *Combine H2O Datasets by Columns*

Description

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

Usage

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

Arguments

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

Value

An H2OParsedData object containing the combined ... arguments column-wise.

Examples

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

---

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

Description

Returns column names for an H2OParsedData object.

Usage

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

Arguments

x	AnH2OParsedData object.
do.NULL	Logical value. If FALSE and names are NULL, names are created.
prefix	Character string denoting prefix for created column names.

Value

Returns a vector of column names.

Examples

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

diff.H2OParsedData *Lagged Differences of H2O Dataset*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

Extremes *Maxima and Minima*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

---

h2o.addFunction *Adds an R function to H2O*

Description

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

Usage

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

Arguments

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

Details

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

See Also

h2o.ddply

Examples

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

---

h2o.anyFactor *Determine if an H2O parsed data object contains categorical data.*

Description

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

Usage

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

Arguments

x	An H2OParsedData object.

Value

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

See Also

H2OParsedData

Examples

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

h2o.assign *Assigns an H2O hex.key to an H2O object so that it can be utilized in subsequent calls*

Description

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

Usage

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

Arguments

data	An H2OParsedData object
key	The hex key to be associated with the H2O parsed data object

Value

The function returns an object of class H2OParsedData

Examples

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

h2o.clearLogs *Delete All H2O R Logs*

Description

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

Usage

```
h2o.clearLogs()
```

See Also

h2o.startLogging, h2o.stopLogging, h2o.openLog, h2o.getLogPath, h2o.setLogPath

Examples

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

---

h2o.clusterInfo *Get Information on H2O Cluster*

Description

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

Usage

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

Arguments

client	An H2OClient object containing the IP address and port of the server running H2O.

See Also

H2OClient

Examples

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

h2o.clusterStatus *Retrieve Status of H2O Cluster*

Description

Retrieve information on the status of the cluster running H2O.

Usage

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

Arguments

client	An H2OClient object containing the IP address and port of the server running H2O.

Details

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

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

Furthermore, for each node, this function displays:

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

See Also

H2OClient, h2o.init

Examples

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

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

Description

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

Usage

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

Arguments

data	An H2OParsedData object that represents the predicted response values. (Must be a single column).
reference	An H2OParsedData object that represents the actual response values. Must have the same dimensions as data.

Value

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

See Also

H2OParsedData

Examples

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

h2o.createFrame *Create an H2O Frame*

Description

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

Usage

```
h2o.createFrame(object, key, rows, cols, seed, randomize, value, categorical_fraction, factors, intege
```

Arguments

key	Name (Key) of frame to be created
rows	Number of rows
cols	Number of columns
seed	Random number seed
randomize	Whether frame should be randomized
value	Constant value (for randomize=false)
real_range	Range for real variables (-range ... range)
categorical_fraction	Fraction of categorical columns (for randomize=true)
factors	Factor levels for categorical variables
integer_fraction	Fraction of integer columns (for randomize=true)
integer_range	Range for integer variables (-range ... range)
missing_fraction	Fraction of missing values

Value

Returns an H2O data frame.

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, beta = TRUE)
myframe = h2o.createFrame(localH2O, 'myframekey', rows = 1000, cols = 10,
                          seed = -12301283, randomize = T, value = 0, real_range = 2.0,
                          categorical_fraction = 0.2, factors = 100,
                      integer_fraction = 0.2, integer_range = 100, missing_fraction = 0.1)
head(myframe)
summary(myframe)
h2o.shutdown(localH2O)
```

h2o.cut *Convert H2O Numeric Data to Factor*

Description

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

Usage

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

Arguments

x	An H2OParsedData object with numeric columns.
breaks	A numeric vector of two or more unique cut points.

Value

A H2OParsedData object containing the factored data with intervals as levels.

Examples

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

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

h2o.ddply *Split H2O dataset, apply function, and return results*

Description

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

Usage

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

Arguments

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

Details

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

Value

An H2OParsedData object containing the results from the split/apply operation, arranged row-by-row.

References

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

See Also

h2o.addFunction

Examples

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

# Import iris dataset to H2O
irisPath = system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")

# Add function taking mean of sepal_len column
fun = function(df) { sum(df[,1], na.rm = T)/nrow(df) }
h2o.addFunction(localH2O, fun)

# Apply function to groups by class of flower
res = h2o.ddply(iris.hex, "class", fun)
head(res)
```

h2o.deeplearning *H2O: Deep Learning Neural Networks*

Description

Performs Deep Learning neural networks on an H2OParsedData object.

Usage

```
h2o.deeplearning(x, y, data, classification = TRUE, validation, activation,
    hidden, epochs, train_samples_per_iteration, seed, adaptive_rate,
    rho, epsilon, rate, rate_annealing, rate_decay, momentum_start,
    momentum_ramp, momentum_stable, nesterov_accelerated_gradient,
    input_dropout_ratio, hidden_dropout_ratios, l1, l2, max_w2,
    initial_weight_distribution, initial_weight_scale, loss,
    score_interval, score_training_samples, score_validation_samples,
    score_duty_cycle, classification_stop, regression_stop, quiet_mode,
    max_confusion_matrix_size, max_hit_ratio_k, balance_classes,
    max_after_balance_size, score_validation_sampling, diagnostics,
    variable_importances, fast_mode, ignore_const_cols, force_load_balance,
    replicate_training_data, single_node_mode, shuffle_training_data,
    sparse, col_major)
```

Arguments

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An H2OParsedData object containing the variables in the model.
classification	(Optional) A logical value indicating whether the algorithm should conduct classification.
validation	(Optional) An H2OParsedData object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data.
activation	A string indicating the activation function to use. Must be either "Tanh", "TanhDropout", "Rectifier", "RectifierDropout", "Maxout" or "MaxoutDropout".
hidden	Hidden layer sizes (e.g. c(100,100)
epochs	How many times the dataset should be iterated (streamed), can be fractional
train_samples_per_iteration	Number of training samples (globally) per MapReduce iteration. Special values are 0: one epoch, -1: all available data (e.g., replicated training data)
seed	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded
adaptive_rate	Adaptive learning rate (ADADELTA)
rho	Adaptive learning rate time decay factor (similarity to prior updates)

`epsilon` Adaptive learning rate smoothing factor (to avoid divisions by zero and allow progress)

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

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

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

`momentum_start` Initial momentum at the beginning of training (try 0.5)

`momentum_ramp` Number of training samples for which momentum increases

`momentum_stable`
Final momentum after the ramp is over (try 0.99)

`nesterov_accelerated_gradient`
Use Nesterov accelerated gradient (recommended)

`input_dropout_ratio`
Input layer dropout ratio (can improve generalization, try 0.1 or 0.2)

`hidden_dropout_ratios`
Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5

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

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

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

`initial_weight_distribution`
Initial Weight Distribution

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

`loss` Loss function

`score_interval` Shortest time interval (in secs) between model scoring

`score_training_samples`
Number of training set samples for scoring (0 for all)

`score_validation_samples`
Number of validation set samples for scoring (0 for all)

`score_duty_cycle`
Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring).

`classification_stop`
Stopping criterion for classification error fraction on training data (-1 to disable)

`regression_stop`
Stopping criterion for regression error (MSE) on training data (-1 to disable)

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

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

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

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

`max_after_balance_size`
: Maximum relative size of the training data after balancing class counts (can be less than 1.0)

`score_validation_sampling`
: Method used to sample validation dataset for scoring

`diagnostics`
: Enable diagnostics for hidden layers

`variable_importances`
: Compute variable importances for input features (Gedeon method) - can be slow for large networks

`fast_mode`
: Enable fast mode (minor approximation in back-propagation)

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

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

`replicate_training_data`
: Replicate the entire training dataset onto every node for faster training on small datasets

`single_node_mode`
: Run on a single node for fine-tuning of model parameters

`shuffle_training_data`
: Enable shuffling of training data (recommended if training data is replicated and train_samples_per_iteration is close to #nodes x #rows)

`sparse`
: Sparse data handling (Experimental).

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

Value

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

`confusion`
: The confusion matrix of the response, with actual observations as rows and predicted values as columns.

`train_class_err`
: Classification error on the training dataset.

`train_sqr_err`
: Mean-squared error on the training dataset.

`valid_class_err`
: Classification error on the validation dataset.

`valid_sqr_err`
: Mean-squared error on the validation dataset.

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
h2o.deeplearning(x = 1:4, y = 5, data = iris.hex, activation = "Tanh",
hidden = c(50, 50, 50), epochs = 500)
```

---

h2o.downloadAllLogs *Download H2O Log Files to Disk*

Description

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

Usage

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

Arguments

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

See Also

H2OClient

Examples

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

h2o.downloadCSV *Download H2O Data to Disk*

Description

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

Usage

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

Arguments

data	An H2OParsedData object to be downloaded.
filename	A character string indicating the name that the CSV file should be saved to.

Details

This method requires wget or curl to be installed on your local system. WARNING: Files located on the H2O server may be very large! Make sure you have enough hard drive space to accommodate the entire file.

See Also

H2OParsedData, H2OParsedDataVA

Examples

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

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

h2o.exportFile *Export H2O Data Frame to a File.*

Description

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

Usage

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

Arguments

data	An H2OParsedData or H2OParsedDataVA data frame.
path	The path to write the file to. Must include the directory and filename. May be prefaced with hdfs:// or s3n://. Each row of data appears as one line of the file.
force	(Optional) If `force = TRUE` any existing file will be overwritten. Otherwise if the file already exists the operation will fail.

Value

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

Examples

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

## End(Not run)
```

h2o.exportHDFS *Export a H2O Model to HDFS*

Description

Saves a model built from a H2O algorithm to HDFS.

Usage

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

Arguments

object	An H2OModel object representing the model to be exported.
path	The HDFS file path where the model should be saved.

See Also

H2OModel

Examples

```
## Not run:
# This is an example of how to export H2O models to HDFS.
# The user must modify the path to his or her specific HDFS path for this example to run.
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath)
iris.gbm = h2o.gbm(x = 1:4, y = 5, data = iris.hex)
h2o.exportHDFS(iris.gbm, path = "hdfs://192.168.1.161/datasets/models")

## End(Not run)
```

h2o.gapStatistic *Compute Gap Statistic from H2O Dataset*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list containing the following components:

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

References

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

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

See Also

H2OParsedData, h2o.kmeans

Examples

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

## End(Not run)
```

h2o.gbm *H2O: Gradient Boosted Machines*

Description

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

Usage

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

Arguments

x	A vector containing the names or indices of the predictor variables to use in building the GBM model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index number starting at 0, and increasing from left to right. (The response must be either an integer or a categorical variable).
distribution	The type of GBM model to be produced: classification is "multinomial" (default), "gaussian" is used for regression.
data	An H2OParsedData object containing the variables in the model.
n.trees	(Optional) Number of trees to grow. Must be a nonnegative integer.
interaction.depth	(Optional) Maximum depth to grow the tree.
n.minobsinnode	(Optional) Minimum number of rows to assign to teminal nodes.
shrinkage	(Optional) A learning-rate parameter defining step size reduction.
n.bins	(Optional) Number of bins to use in building histogram.
importance	(Optional) A logical value indicating whether variable importance should be calculated. This will increase the amount of time for the algorithm to complete.
validation	(Optional) An H2OParsedData object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data.
balance.classes	(Optional) Balance training data class counts via over/under-sampling (for imbalanced data)
max.after.balance.size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)

Value

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

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

References

1. Elith, Jane, John R Leathwick, and Trevor Hastie. "A Working Guide to Boosted Regression Trees." Journal of Animal Ecology 77.4 (2008): 802-813

2. Friedman, Jerome, Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. "Discussion of Boosting Papers." Ann. Statist 32 (2004): 102-107

3. Hastie, Trevor, Robert Tibshirani, and J Jerome H Friedman. The Elements of Statistical Learning. Vol.1. N.p.: Springer New York, 2001. http://www.stanford.edu/~hastie/local.ftp/Springer/OLD//ESLII_print4.pdf

See Also

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

Examples

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

# Run regression GBM on australia.hex data
ausPath = system.file("extdata", "australia.csv", package="h2o")
australia.hex = h2o.importFile(localH2O, path = ausPath)
independent <- c("premax", "salmax","minairtemp", "maxairtemp", "maxsst",
  "maxsoilmoist", "Max_czcs")
dependent <- "runoffnew"
h2o.gbm(y = dependent, x = independent, data = australia.hex, n.trees = 10, interaction.depth = 3,
  n.minobsinnode = 2, shrinkage = 0.2, distribution= "gaussian")

# Run multinomial classification GBM on australia data
h2o.gbm(y = dependent, x = independent, data = australia.hex, n.trees = 15, interaction.depth = 5,
  n.minobsinnode = 2, shrinkage = 0.01, distribution= "multinomial")
```

h2o.getLogPath *Get Path Where H2O R Logs are Saved*

Description

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

Usage

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

Arguments

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

See Also

h2o.startLogging, h2o.stopLogging, h2o.clearLogs, h2o.openLog, h2o.setLogPath

Examples

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

h2o.glm *H2O: Generalized Linear Models*

Description

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

Usage

```
## Default method:
h2o.glm(x, y, data, family, nfolds = 10, alpha = 0.5, lambda = 1e-5, epsilon = 1e-4,
  standardize = TRUE, prior, tweedie.p = ifelse(family == 'tweedie', 1.5,
 as.numeric(NA)), thresholds, iter.max, higher_accuracy, lambda_search, version = 2)

## Import to a ValueArray object:
h2o.glm.VA(x, y, data, family, nfolds = 10, alpha = 0.5, lambda = 1e-5, epsilon = 1e-4,
  standardize = TRUE, prior, tweedie.p = ifelse(family == 'tweedie', 1.5,
  as.numeric(NA)), thresholds = ifelse(family == 'binomial', seq(0, 1, 0.01),
  as.numeric(NA)))
```

```
## Import to a FluidVecs object:
h2o.glm.FV(x, y, data, family, nfolds = 10, alpha = 0.5, lambda = 1e-5, epsilon = 1e-4,
  standardize = TRUE, prior, tweedie.p = ifelse(family == 'tweedie', 1.5,
 as.numeric(NA)), iter.max = 100, higher_accuracy = FALSE, lambda_search = FALSE)
```

Arguments

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing the variables in the model.
family	A description of the error distribution and corresponding link function to be used in the model. Currently, Gaussian, binomial, Poisson, gamma, and Tweedie are supported. When a model is specified as Tweedie, users must also specify the appropriate Tweedie power.
nfolds	(Optional) Number of folds for cross-validation. The default is 10.
alpha	(Optional) The elastic-net mixing parameter, which must be in [0,1]. The penalty is defined to be $$P(\alpha,\beta) = (1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1 = \sum_j [(1-\alpha)/2\beta_j^2 + \alpha\|\beta_j\|]$$ so alpha=1 is the lasso penalty, while alpha=0 is the ridge penalty.
lambda	The shrinkage parameter, which multiples $P(\alpha,\beta)$ in the objective. The larger lambda is, the more the coefficients are shrunk toward zero (and each other).
epsilon	(Optional) Number indicating the cutoff for determining if a coefficient is zero.
standardize	(Optional) Logical value indicating whether the data should be standardized (set to mean = 0, variance = 1) before running GLM.
prior	(Optional) Prior probability of class 1. Only used if family = "binomial". When omitted, prior will default to the frequency of class 1 in the response column.
tweedie.p	(Optional) The index of the power variance function for the tweedie distribution. Only used if family = "tweedie".
thresholds	(Optional) Degree to which to weight the sensitivity (the proportion of correctly classified 1's) and specificity (the proportion of correctly classified 0s). The default option is joint optimization for the overall classification rate. Changing this will alter the confusion matrix and the AUC. Only used if family = "binomial".
iter.max	(Optional) Maximum number of iterations allowed.
higher_accuracy	(Optional) A logical value indicating whether to use line search. This will cause the algorithm to run slower, so generally, it should only be set to TRUE if GLM does not converge otherwise.
lambda_search	(Optional) A logical value indicating whether to onduct a search over the space of lambda values, starting from lambda_max. When this is set to TRUE, lambda will be interpreted as lambda_min.

version	(Optional) The version of GLM to run. If version = 1, this will run the more stable ValueArray implementation, while version = 2 runs the faster, but still beta stage FluidVecs implementation.

Details

IMPORTANT: Currently, to run GLM with version = 1, you must import data to a ValueArray object using h2o.importFile.VA, h2o.importFolder.VA or one of its variants. To run with version = 2, you must import data to a FluidVecs object using h2o.importFile.FV, h2o.importFolder.FV or one of its variants.

Value

An object of class H2OGLMModelVA (version = 1) or H2OGLMModel (version = 2) with slots key, data, model and xval. The slot model is a list of the following components:

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

The slot xval is a list of H2OGLMModel or H2OGLMModelVA objects representing the cross-validation models. (Each of these objects themselves has xval equal to an empty list).

See Also

h2o.importFile, h2o.importFolder, h2o.importHDFS, h2o.importURL, h2o.uploadFile

Examples

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

# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.github.com",
```

```
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data = prostate.hex, family = "binomial",
  nfolds = 10, alpha = 0.5)
# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
myX = setdiff(colnames(prostate.hex), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.glm(y = "VOL", x = myX, data = prostate.hex, family = "gaussian", nfolds = 5, alpha = 0.1)
```

`h2o.hitRatio` *Compute Hit Ratio from H2O Classification Predictions*

Description

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

Usage

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

Arguments

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

Value

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

See Also

`H2OParsedData`

Examples

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

h2o.importFile *Import Local Data File*

Description

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

Usage

```
## Default method:
h2o.importFile(object, path, key = "", parse = TRUE, header, sep = "", col.names,
  version = 2)

## Import to a ValueArray object:
h2o.importFile.VA(object, path, key = "", parse = TRUE, header, sep = "", col.names)

## Import to a FluidVecs object:
h2o.importFile.FV(object, path, key = "", parse = TRUE, header, sep = "", col.names)
```

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
path	The path of the file to be imported. Each row of data appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
version	(Optional) If version = 1, the file will be imported to a ValueArray object. Otherwise, if version = 2, the file will be imported as a FluidVecs object.

Details

Calling the method with `version = 1` is equivalent to `h2o.importFile.VA`, and `version = 2` is equivalent to `h2o.importFile.FV`.

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

Value

If `parse = TRUE`, the function returns an object of class `H2OParsedDataVA` when `version = 1` and an object of class `H2OParsedData` when `version = 2`. Otherwise, when `parse = FALSE`, it returns an object of class `H2ORawDataVA` when `version = 1` and an object of class `H2ORawData` when `version = 2`.

See Also

`h2o.importFolder`, `h2o.importHDFS`, `h2o.importURL`, `h2o.uploadFile`

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
irisPath = system.file("extdata", "iris.csv", package = "h2o")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
class(iris.hex)
summary(iris.hex)
iris.fv = h2o.importFile(localH2O, path = irisPath, key = "iris.fv", version = 2)
class(iris.fv)
```

---

h2o.importFolder *Import Local Directory of Data Files*

Description

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

Usage

```
## Default method:
h2o.importFolder(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names, version = 2)

## Import to a ValueArray object:
h2o.importFolder.VA(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)

## Import to a FluidVecs object:
h2o.importFolder.FV(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)
```

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
path	The path of the folder directory to be imported. Each row of data appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
version	(Optional) If version = 1, the folder of files will be imported to a ValueArray object. Otherwise, if version = 2, the files will be imported as a FluidVecs object.

Details

Calling the method with version = 1 is equivalent to h2o.importFolder.VA, and version = 2 is equivalent to h2o.importFolder.FV.

This method imports all the data files in a given folder and concatenates them together row-wise into a single matrix represented by a H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object. The data files must all have the same number of columns, and the columns must be lined up in the same order, otherwise an error will be returned.

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

Value

If parse = TRUE, the function returns an object of class H2OParsedDataVA when version = 1 and an object of class H2OParsedData when version = 2. Otherwise, when parse = FALSE, it returns an object of class H2ORawDataVA when version = 1 and an object of class H2ORawData when version = 2.

See Also

h2o.importFile, h2o.importHDFS, h2o.importURL, h2o.uploadFile

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
myPath = system.file("extdata", "prostate_folder", package = "h2o")
prostate_all.hex = h2o.importFolder(localH2O, path = myPath)
class(prostate_all.hex)
summary(prostate_all.hex)
prostate_all.fv = h2o.importFolder(localH2O, path = myPath, version = 2)
class(prostate_all.fv)
```

---

h2o.importHDFS *Import from HDFS*

Description

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

Usage

```
## Default method:
h2o.importHDFS(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names, version = 2)

## Import to a ValueArray object:
h2o.importHDFS.VA(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)

## Import to a FluidVecs object:
h2o.importHDFS.FV(object, path, pattern = "", key = "", parse = TRUE, header,
  sep = "", col.names)
```

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
path	The path of the file or folder directory to be imported. If it does not contain an absolute path, the file name is relative to the current working directory.
pattern	(Optional) Character string containing a regular expression to match file(s) in the folder.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the file path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.

sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
version	(Optional) If version = 1, the file will be imported to a ValueArray object. Otherwise, if version = 2, the file will be imported as a FluidVecs object.

Details

Calling the method with version = 1 is equivalent to h2o.importHDFS.VA, and version = 2 is equivalent to h2o.importHDFS.FV.

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

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

Value

If parse = TRUE, the function returns an object of class H2OParsedDataVA when version = 1 and an object of class H2OParsedData when version = 2. Otherwise, when parse = FALSE, it returns an object of class H2ORawDataVA when version = 1 and an object of class H2ORawData when version = 2.

See Also

h2o.importFile, h2o.importFolder, h2o.importURL, h2o.uploadFile

Examples

```
## Not run:
# This is an example of how to import files from HDFS.
# The user must modify the path to his or her specific HDFS path for this example to run.
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
iris.hex = h2o.importHDFS(localH2O, path = paste("hdfs://192.168.1.161",
  "datasets/runit/iris_wheader.csv", sep = "/"), parse = TRUE)
class(iris.hex)
summary(iris.hex)
iris.fv = h2o.importHDFS(localH2O, path = paste("hdfs://192.168.1.161",
  "datasets/runit/iris_wheader.csv", sep = "/"), parse = TRUE, version = 2)
class(iris.fv)

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

## End(Not run)
```

h2o.importURL *Import Data from URL*

Description

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

Usage

```
## Default method:
h2o.importURL(object, path, key = "", parse = TRUE, header,
  sep = "", col.names, version = 2)

## Import to a ValueArray object:
h2o.importURL.VA(object, path, key = "", parse = TRUE, header,
  sep = "", col.names)

## Import to a FluidVecs object:
h2o.importURL.FV(object, path, key = "", parse = TRUE, header,
  sep = "", col.names)
```

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
path	The complete URL of the file to be imported. Each row of data appears as one line of the file.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
version	(Optional) If version = 1, the file will be imported to a ValueArray object. Otherwise, if version = 2, the file will be imported as a FluidVecs object.

Details

Calling the method with `version = 1` is equivalent to `h2o.importURL.VA`, and `version = 2` is equivalent to `h2o.importURL.FV`.

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

Value

If `parse = TRUE`, the function returns an object of class `H2OParsedDataVA` when `version = 1` and an object of class `H2OParsedData` when `version = 2`. Otherwise, when `parse = FALSE`, it returns an object of class `H2ORawDataVA` when `version = 1` and an object of class `H2ORawData` when `version = 2`.

See Also

`h2o.importFile`, `h2o.importFolder`, `h2o.importHDFS`, `h2o.uploadFile`

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.github.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
class(prostate.hex)
summary(prostate.hex)

prostate.fv = h2o.importURL(localH2O, path = paste("https://raw.github.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex",
  version = 2)
class(prostate.fv)
```

---

h2o.init *Connect to H2O and Install R Package*

Description

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

Usage

```
h2o.init(ip = "127.0.0.1", port = 54321, startH2O = TRUE, forceDL = FALSE, Xmx = "1g",
        beta = FALSE, assertion = TRUE, license = 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.
startH2O	(Optional) A logical value indicating whether to start the H2O launcher GUI if no connection with H2O is detected. This is only possible if ip = "localhost" or ip = "127.0.0.1".
forceDL	(Optional) A logical value indicating whether to force download of the H2O executable. Defaults to FALSE, so the executable will only be downloaded if it does not already exist in the h2o R library resources directory h2o/java/h2o.jar.
Xmx	(Optional) A string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes.
beta	(Optional) A logical value indicating whether H2O should be launch in beta mode.
assertion	(Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes.
license	(Optional) A string value specifying the full path of the license file.

Details

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

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

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

Value

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

Note

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

See Also

h2o.shutdown

Examples

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

---

h2o.kmeans *H2O: K-Means Clustering*

Description

Performs k-means clustering on a data set.

Usage

```
## Default method:
h2o.kmeans(data, centers, cols = "", iter.max = 10, normalize = FALSE,
  init = "none", seed = 0, dropNACols, version = 2)

## Import to a ValueArray object:
h2o.kmeans.VA(data, centers, cols = "", iter.max = 10, normalize = FALSE,
  init = "none", seed = 0)

## Import to a FluidVecs object:
h2o.kmeans.FV(data, centers, cols = "", iter.max = 10, normalize = FALSE,
  init = "none", seed = 0, dropNACols = FALSE)
```

Arguments

data	An H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing the variables in the model.
centers	The number of clusters k.
cols	(Optional) A vector containing the names of the data columns on which k-means runs. If blank, k-means clustering will be run on the entire data set.
iter.max	(Optional) The maximum number of iterations allowed.
normalize	(Optional) A logical value indicating whether the data should be normalized before running k-means.
init	(Optional) Method by which to select the k initial cluster centroids. Possible values are "none" for random initialization, "plusplus" for k-means++ initialization, and "furthest" for initialization at the furthest point from each successive centroid. See the H2O K-means documentation for more details.
seed	(Optional) Random seed used to initialize the cluster centroids.
dropNACols	(Optional) A logical value indicating whether to drop columns with more than 10% entries that are NA.
version	(Optional) The version of k-means clustering to run. If version = 1, this will run the more stable ValueArray implementation, while version = 2 selects the faster, but still beta stage FluidVecs implementation.

Details

IMPORTANT: Currently, to run k-means with `version = 1`, you must import data to a ValueArray object using `h2o.importFile.VA`, `h2o.importFolder.VA` or one of its variants. To run with `version = 2`, you must import data to a FluidVecs object using `h2o.importFile.FV`, `h2o.importFolder.FV` or one of its variants.

Value

An object of class `H2OKMeansModelVA` (`version = 1`) or `H2OKMeansModel` (`version = 2`) with slots key, data, and model, where the last is a list of the following components:

`centers`	A matrix of cluster centers.
`cluster`	A `H2OParsedDataVA` (`version = 1`) or `H2OParsedData` (`version = 2`) object containing the vector of integers (from 1 to k), which indicate the cluster to which each point is allocated.
`size`	The number of points in each cluster.
`withinss`	Vector of within-cluster sum of squares, with one component per cluster.
`tot.withinss`	Total within-cluster sum of squares, i.e., sum(withinss).

See Also

`h2o.importFile`, `h2o.importFolder`, `h2o.importHDFS`, `h2o.importURL`, `h2o.uploadFile`

Examples

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

---

h2o.logAndEcho *Write and Echo Message to H2O Log*

Description

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

Usage

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

Arguments

`conn`	An `H2OClient` object containing the IP address and port of the server running H2O.
`message`	A character string to write to the H2O Java log file.

See Also

H2OClient, h2o.downloadAllLogs

Examples

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

---

h2o.ls *Obtain a list of H2O keys from the running instance of H2O*

Description

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

Usage

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

Arguments

object	An H2OClient object containing the IP address and port number of the H2O server.
pattern	A string indicating the type of key to be returned. When pattern is left is unspecified all keys are returned.

Value

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

Examples

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

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

Description

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

Usage

```
h2o.month(x)

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

Arguments

x	An H2OParsedData object.

Details

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

Value

A H2OParsedData object containing the entries of x converted to months of the year.

See Also

h2o.year

h2o.naiveBayes *H2O: Naive Bayes Classifier*

Description

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

Usage

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

Arguments

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An H2OParsedData (version = 2) object containing the variables in the model.
laplace	(Optional) A positive number controlling Laplace smoothing. The default (0) disables Laplace smoothing.
dropNACols	(Optional) A logical value indicating whether to drop predictor columns with >= 20% NAs.

Details

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

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

Value

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

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

See Also

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

Examples

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

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

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

---

h2o.openLog *View H2O R Logs*

Description

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

Usage

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

Arguments

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

See Also

h2o.startLogging, h2o.stopLogging, h2o.clearLogs, h2o.getLogPath, h2o.setLogPath

Examples

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

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

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

## End(Not run)
```

h2o.parseRaw *Parse Raw Data File*

Description

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

Usage

```
## Default method:
h2o.parseRaw(data, key = "", header, sep = "", col.names, version = 2)

## Import to a ValueArray object:
h2o.parseRaw.VA(data, key = "", header, sep = "", col.names)

## Import to a FluidVecs object:
h2o.parseRaw.FV(data, key = "", header, sep = "", col.names)
```

Arguments

data	An H2ORawDataVA (version = 1) or H2ORawData (version = 2) object to be parsed.
key	(Optional) The hex key assigned to the parsed file.
header	(Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
version	(Optional) If version = 1, the file will be parsed to a ValueArray object. Otherwise, if version = 2, the file will be parsed to a FluidVecs object.

Details

Calling the method with version = 1 is equivalent to h2o.parseRaw.VA, and version = 2 is equivalent to h2o.parseRaw.FV. h2o.parseRaw.VA should only be used to parse raw data imported using h2o.importFile.VA, h2o.importFolder.VA, or one of its variants. Similarly, h2o.parseRaw.FV should only be used to parse raw data imported using h2o.importFile.FV, h2o.importFolder.FV, or one of its variants.

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

Value

An object of class H2OParsedDataVA (version = 1) or H2OParsedData (version = 2), representing the dataset that was parsed.

See Also

h2o.importFile, h2o.importFolder, h2o.importHDFS, h2o.importURL, h2o.uploadFile

Examples

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

---

h2o.pcr *H2O: Principal Components Regression*

Description

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

Usage

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

Arguments

x	A vector containing the names of the predictors in the model.
y	The name of the response variable in the model.
data	An H2OParsedData object containing the variables in the model.
ncomp	A number indicating the number of principal components to use in the regression model.
family	A description of the error distribution and corresponding link function to be used in the model. Currently, Gaussian, binomial, Poisson, gamma, and Tweedie are supported.
nfolds	(Optional) Number of folds for cross-validation. The default is 10.
alpha	(Optional) The elastic-net mixing parameter, which must be in [0,1]. The penalty is defined to be $$P(\alpha,\beta) = (1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1 = \sum_j [(1-\alpha)/2\beta_j^2 + \alpha\|\beta_j\|]$$ so alpha=1 is the lasso penalty, while alpha=0 is the ridge penalty.
lambda	(Optional) The shrinkage parameter, which multiples $P(\alpha,\beta)$ in the objective. The larger lambda is, the more the coefficients are shrunk toward zero (and each other).

epsilon	(Optional) Number indicating the cutoff for determining if a coefficient is zero.
tweedie.p	The index of the power variance function for the tweedie distribution. Only used if family = "tweedie"

Details

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

Value

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

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

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

See Also

h2o.prcomp, h2o.glm

Examples

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

# Run PCR on Prostate Data
prostate.hex = h2o.importURL(localH2O, path = paste("https://raw.github.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
h2o.pcr(x = c("AGE","RACE","PSA","DCAPS"), y = "CAPSULE", data = prostate.hex, family = "binomial",
  nfolds = 10, alpha = 0.5, ncomp = 3)
```

h2o.performance *Performance Measures*

Description

Evaluate the predictive performance of a model via various measures.

Usage

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

Arguments

data	An H2OParsedData object containing the predicted outcome scores. Must be a single column with the same number of rows as reference.
reference	An H2OParsedData object containing the actual outcomes for comparison. Must be a single binary column with all entries in {0,1}.
measure	A character string indicating the performance measure to optimize. Must be one of the following: • F1: F1 score, equal to $2 * (Precision * Recall)/Precision + Recall$ • accuracy: Accuracy of model, estimated as $(TP + TN)/(P + N)$. • precision: Precision of model, estimated as $TP/(TP + FP)$. • recall: Recall of model, i.e. the true positive rate TP/P. • specificity: Specificity of model, i.e. the true negative rate TN/N. • max_per_class_error: Maximum per class error in model.
thresholds	(Optional) A numeric vector from 0 to 1 indicating the threshold values at which to compute the performance measure. If missing, the range will be automatically generated. TODO: Still not sure I understand what exactly these thresholds are, is it the FPR or something else?

Value

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

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

specificity	Specificity value at best cutoff.
max_per_class_err	Maximum per class error at best cutoff.
confusion	Confusion matrix at best cutoff.

Examples

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

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

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

---

h2o.prcomp *Principal Components Analysis*

Description

Performs principal components analysis on the given data set.

Usage

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

Arguments

data	An H2OParsedData object on which to run principal components analysis.
tol	(Optional) A value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to tol times the standard deviation of the first component.) With the default setting `tol = 0`, no components are omitted.
cols	(Optional) A vector of column names or indices indicating the features to perform PCA on. By default, all columns in the dataset are analyzed.
standardize	(Optional) A logical value indicating whether the variables should be shifted to be zero centered and scaled to have unit variance before the analysis takes place.
retx	(Optional) A logical value indicating whether the rotated variables should be returned.

Details

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

Value

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

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

Note

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

See Also

h2o.pcr

Examples

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

---

h2o.predict *H2O Model Predictions*

Description

Obtains predictions from various fitted H2O model objects.

Usage

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

Arguments

object	A fitted H2OModel or H2OModelVA object for which prediction is desired.
newdata	(Optional) A H2OParsedData or H2OParsedDataVA object in which to look for variables with which to predict. If omitted, the data used to fit the model object@data are used.

Details

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

Value

A H2OParsedData or H2OParsedDataVA object containing the predictions.

See Also

h2o.glm, h2o.kmeans, h2o.randomForest, h2o.prcomp, h2o.gbm, h2o.deeplearning

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate.hex = h2o.importURL.VA(localH2O, path = paste("https://raw.github.com",
  "0xdata/h2o/master/smalldata/logreg/prostate.csv", sep = "/"), key = "prostate.hex")
prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data = prostate.hex,
   family = "binomial", nfolds = 10, alpha = 0.5)
# Get fitted values of prostate dataset
prostate.fit = h2o.predict(object = prostate.glm, newdata = prostate.hex)
summary(prostate.fit)
```

---

h2o.randomForest *H2O: Random Forest*

Description

Performs random forest classification on a data set.

Usage

```
## Default method:
h2o.randomForest(x, y, data, classification = TRUE, ntree = 50, depth = 20,
  sample.rate = 2/3, classwt = NULL, nbins = 100, seed = -1, importance = FALSE,
  validation, nodesize = 1, balance.classes = FALSE, max.after.balance.size = 5,
  use_non_local = TRUE, version = 2)

## Import to a ValueArray object:
h2o.randomForest.VA(x, y, data, ntree = 50, depth = 20, sample.rate = 2/3,
  classwt = NULL, nbins = 100, seed = -1, use_non_local = TRUE)

## Import to a FluidVecs object:
h2o.randomForest.FV(x, y, data, classification = TRUE, ntree = 50, depth = 20,
  sample.rate = 2/3, nbins = 100, seed = -1, importance = FALSE, validation,
  nodesize = 1, balance.classes = FALSE, max.after.balance.size = 5)
```

Arguments

x	A vector containing the names or indices of the predictor variables to use in building the random forest model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index, designated by increasing numbers from left to right. (The response must be either an integer or a categorical variable).
data	An H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing the variables in the model.
classification	(Optional) A logical value indicating whether a classification model should be built (as opposed to regression).
ntree	(Optional) Number of trees to grow. (Must be a nonnegative integer).
depth	(Optional) Maximum depth to grow the tree.
sample.rate	(Optional) Sampling rate for constructing data from which individual trees are grown.
classwt	(Optional) Numeric vector of class weights for a categorical response.
nbins	(Optional) Build a histogram of this many bins, then split at best point.
seed	(Optional) Seed for building the random forest. If seed = -1, one will automatically be generated by H2O.
importance	(Optional) A logical value indicating whether to calculate variable importance. Set to FALSE to speed up computations.
validation	(Optional) An H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data.
nodesize	(Optional) Number of nodes to use for computation.
balance.classes	(Optional) Balance training data class counts via over/under-sampling (for imbalanced data)
max.after.balance.size	Maximum relative size of the training data after balancing class counts (can be less than 1.0)
use_non_local	(Optional) Logical value indicating whether to use non-local data in building random forest model.
version	(Optional) The version of random forest to run. If version = 1, this will run the single-node ValueArray implementation, while version = 2 selects the distributed, but still beta stage FluidVecs implementation.

Details

IMPORTANT: Currently, to run k-means with version = 1, you must import data to a ValueArray object using h2o.importFile.VA, h2o.importFolder.VA or one of its variants. To run with version = 2, you must import data to a FluidVecs object using h2o.importFile.FV, h2o.importFolder.FV or one of its variants.

Value

An object of class H2ORFModelVA (version = 1) or H2ODRFModel (version = 2) with slots key, data, and model, where the last is a list of the following components:

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

Examples

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

---

h2o.rm *Removes H2O objects from the server where H2O is running.*

Description

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

Usage

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

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
keys	the hex key associated with the object to be removed.

Note

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

See Also

h2o.assign, h2o.ls

Examples

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

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

---

h2o.runif *Produces a vector of specified length contain random uniform numbers*

Description

Produces a vector of random uniform numbers.

Usage

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

Arguments

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

Details

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

Value

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

Examples

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

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

---

h2o.setLogPath *Set Path Where H2O R Logs are Saved*

Description

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

Usage

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

Arguments

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

See Also

h2o.startLogging, h2o.stopLogging, h2o.clearLogs, h2o.openLog, h2o.getLogPath

Examples

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

h2o.shutdown *Shutdown H2O server*

Description

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

Usage

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

Arguments

client	An H2OClient client containing the IP address and port of the server running H2O.
prompt	(Optional) A logical value indicating whether to prompt the user before shutting down the H2O server.

Details

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

Note

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

See Also

h2o.init

Examples

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

## End(Not run)
```

h2o.SpeeDRF *H2O: Single-Node Random Forest*

Description

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

Usage

```
h2o.SpeeDRF(x, y, data, classification = TRUE, validation, mtry = -1, ntree = 50,
  depth = 50, sample.rate = 2/3, oobee = TRUE, importance = FALSE, nbins = 1024,
  seed = -1, stat.type = "ENTROPY", classwt = NULL, sampling_strategy = "RANDOM",
  strata_samples = NULL)
```

Arguments

x	A vector containing the names or indices of the predictor variables to use in building the random forest model.
y	The name or index of the response variable. If the data does not contain a header, this is the column index, designated by increasing numbers from left to right. (The response must be either an integer or a categorical variable).
data	An H2OParsedData object containing the variables in the model.
classification	(Optional) A logical value indicating whether a classification model should be built (as opposed to regression).
validation	(Optional) An H2OParsedData object indicating the validation dataset used to construct confusion matrix. If left blank, this defaults to the training data.
mtry	(Optional) Number of features to randomly select at each split in the tree. If set to the default of -1, this will be set to sqrt(ncol(data)), rounded down to the nearest integer.
ntree	(Optional) Number of trees to grow. (Must be a nonnegative integer).
depth	(Optional) Maximum depth to grow the tree.
sample.rate	(Optional) Sampling rate for constructing data from which individual trees are grown.
oobee	(Optional) A logical value indicating whether to calculate the out of bag error estimate.
importance	(Optional) A logical value indicating whether to compute variable importance measures. (If set to TRUE, the algorithm will take longer to finish.)
nbins	(Optional) Build a histogram of this many bins, then split at best point.
seed	(Optional) Seed for building the random forest. If seed = -1, one will automatically be generated by H2O.
stat.type	(Optional) Type of statistic to use, equal to either "ENTROPY" or "GINI".
classwt	(Optional) Numeric vector of class weights for a categorical response.

`sampling_strategy`
: (Optional) Sampling strategy to use, equal to either "RANDOM" or "STRATIFIED_LOCAL".

`strata_samples`
: (Optional) A numeric sequence indicating the strata for sampling. Only used if `sampling_strategy = "STRATIFIED_LOCAL"`.

Details

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

This method runs random forest model building on a single node, as opposed to the multi-node implementation in `h2o.randomForest.FV`.

Value

An object of class `H2OSpeeDRFModel` with slots key, data, valid (the validation dataset), and model, where the last is a list of the following components:

`params`
: Input parameters for building the model.

`ntree`
: Number of trees grown.

`depth`
: Depth of the trees grown.

`nbins`
: Number of bins used in building the histogram.

`classification`
: Logical value indicating if the model is classification.

`mse`
: Mean-squared error for each tree.

`confusion`
: Confusion matrix of the prediction.

See Also

`H2OSpeeDRFModel`, `h2o.randomForest`

Examples

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

## End(Not run)
```

h2o.startLogging *Start Writing H2O R Logs*

Description

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

Usage

```
h2o.startLogging()
```

See Also

h2o.stopLogging, h2o.clearLogs, h2o.openLog, h2o.getLogPath, h2o.setLogPath

Examples

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

h2o.stopLogging *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.clearLogs, h2o.openLog, h2o.getLogPath, h2o.setLogPath

Examples

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

h2o.table *Cross Tabulation of H2O Data*

Description

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

Usage

```
h2o.table(x)
```

Arguments

x	An H2OParsedData object with at most two integer or factor columns.

Value

A H2OParsedData object containing the contingency table. If x has a single column, this will just be the counts of each factor level.

Examples

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

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

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

h2o.uploadFile *Upload Local Data File*

Description

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

Usage

```
## Default method:
h2o.uploadFile(object, path, key = "", parse = TRUE, header,
  sep = "", col.names, silent = TRUE, version = 2)

## Import to a ValueArray object:
h2o.uploadFile.VA(object, path, key = "", parse = TRUE, header,
  sep = "", col.names, silent = TRUE)

## Import to a FluidVecs object:
h2o.uploadFile.FV(object, path, key = "", parse = TRUE, header,
  sep = "", col.names, silent = TRUE)
```

Arguments

object	An H2OClient object containing the IP address and port of the server running H2O.
path	The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.
key	(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.
parse	(Optional) A logical value indicating whether the file should be parsed after import.
header	(Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.
sep	(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.
col.names	(Optional) A H2OParsedDataVA (version = 1) or H2OParsedData (version = 2) object containing a single delimited line with the column names for the file.
silent	(Optional) A logical value indicating whether or not to display an upload progress bar.
version	(Optional) If version = 1, the file will be imported to a ValueArray object. Otherwise, if version = 2, the file will be imported as a FluidVecs object.

Details

Calling the method with version = 1 is equivalent to h2o.uploadFile.VA, and version = 2 is equivalent to h2o.uploadFile.FV.

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

Value

If parse = TRUE, the function returns an object of class H2OParsedDataVA when version = 1 and an object of class H2OParsedData when version = 2. Otherwise, when parse = FALSE, it returns

an object of class H2OParsedDataVA when version = 1 and an object of class H2ORawData when version = 2.

See Also

h2o.importFile, h2o.importFolder, h2o.importHDFS, h2o.importURL

Examples

```
library(h2o)
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.uploadFile(localH2O, path = prosPath, key = "prostate.hex")
class(prostate.hex)
summary(prostate.hex)
prostate.fv = h2o.uploadFile(localH2O, path = prosPath, key = "prostate.fv", version = 2)
class(prostate.fv)
```

---

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

Description

Converts the entries of a H2OParsedData object from milliseconds to years, indexed starting from 1900.

Usage

```
h2o.year(x)

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

Arguments

x	An H2OParsedData object

Details

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

Value

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

See Also

h2o.month

H2OClient-class *Class* "H2OClient"

Description

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

Objects from the Class

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

Slots

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

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

Methods

h2o.importFile `signature(object = "H2OClient", path = "character", + key = "character", parse = "logic`
...

h2o.importFolder `signature(object = "H2OClient", path = "character", + parse = "logical")`:
...

h2o.importURL `signature(object = "H2OClient", path = "character", + key = "character", parse = "logi`
...

show `signature(object = "H2OClient")`: ...

Examples

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

H2ODeepLearningGrid-class *Class* "H2ODeepLearningGrid"

Description

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

Objects from the Class

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

Slots

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

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

`model`: Object of class `"list"` containing `"H2ODeepLearningModel"` objects representing the models returned by the Deep Learning neural networks grid search.

`sumtable`: Object of class `"list"` containing summary statistics of all the models returned by the Deep Learning neural networks grid search.

Extends

Class `"H2OGrid"`, directly.

Methods

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

See Also

`H2ODeepLearningModel`, `h2o.deeplearning`

Examples

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

H2ODeepLearningModel-class

Class `"H2ODeepLearningModel"`

Description

A class for representing Deep Learning neural network models.

Objects from the Class

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

Slots

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

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

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

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

- `confusion`: The confusion matrix of the response, with actual observations as rows and predicted values as columns.
- `train_class_err`: Classification error on the training dataset.

- train_sqr_err: Mean-squared error on the training dataset.
- train_cross_entropy: Cross-entropy on the training dataset.
- valid_class_err: Classification error on the validation dataset.
- valid_sqr_err: Mean-squared error on the validation dataset.
- valid_cross_entropy: Cross-entropy on the validation dataset.

Extends

Class "H2OModel", directly.

Methods

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

See Also

h2o.deeplearning

Examples

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

H2ODRFGrid-class *Class* "H2ODRFGrid"

Description

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

Objects from the Class

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

Slots

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

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

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

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

Extends

Class "H2OGrid", directly.

Methods

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

See Also

H2ODRFModel, h2o.randomForest

Examples

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

H2ODRFModel-class *Class* "H2ODRFModel"

Description

A class for representing random forest ensembles built on FluidVecs data.

Objects from the Class

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

Slots

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

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

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

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

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

Extends

Class "H2OModel", directly.

Methods

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

See Also

h2o.randomForest

Examples

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

H2OGBMGrid-class *Class* "H2OGBMGrid"

Description

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

Objects from the Class

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

Slots

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

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

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

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

Extends

Class "H2OGrid", directly.

Methods

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

See Also

H2OGBMModel, h2o.gbm

Examples

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

H2OGBMModel-class *Class* "H2OGBMModel"

Description

A class for representing generalized boosted classification/regression models.

Objects from the Class

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

Slots

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

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

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

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

valid: Object of class H2OParsedData, which is the dataset used to validate the model.

Extends

Class "H2OModel", directly.

Methods

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

See Also

h2o.gbm

Examples

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

H2OGLMGrid-class *Class* "H2OGLMGrid"

Description

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

Objects from the Class

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

Slots

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

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

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

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

Extends

Class "H2OGrid", directly.

Methods

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

See Also

H2OGLMModel, h2o.glm

Examples

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

H2OGLMGridVA-class *Class* "H2OGLMGridVA"

Description

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

Objects from the Class

Objects can be created by calls of the form `new("H2OGLMGridVA", ...)`.

Slots

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

data: Object of class "H2OParsedDataVA", which is the input data used to build the model.

model: Object of class "list" containing "H2OGLMModelVA" objects representing the models returned by the GLM (ValueArray) grid search.

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

Extends

Class "H2OGridVA", directly.

Methods

show `signature(object = "H2OGLMGridVA")`: ...

See Also

H2OGLMModelVA, h2o.glm

Examples

```
showClass("H2OGLMGridVA")
```

H2OGLMModel-class *Class* "H2OGLMModel"

Description

A class for representing generalized linear models.

Objects from the Class

Objects can be created by calls of the form `new("H2OGLMModel", ...)`.

Slots

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

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

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

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

xval: List of objects of class `"H2OGLMModel"`, representing the n-fold cross-validation models.

Extends

Class `"H2OModel"`, directly.

Methods

show `signature(object = "H2OGLMModel")`: ...

See Also

`h2o.glm`

Examples

```
showClass("H2OGLMModel")
```

H2OGLMModelVA-class *Class* "H2OGLMModelVA"

Description

A class for representing generalized linear models built on ValueArray data.

Objects from the Class

Objects can be created by calls of the form `new("H2OGLMModelVA", ...)`.

Slots

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

data: Object of class H2OParsedDataVA, which is the input data used to build the model.

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

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

xval: List of objects of class "H2OGLMModelVA", representing the n-fold cross-validation models.

Extends

Class "H2OModelVA", directly.

Methods

show `signature(object = "H2OGLMModelVA")`: ...

See Also

h2o.glm

Examples

```
showClass("H2OGLMModelVA")
```

H2OGrid-class *Class* "H2OGrid"

Description

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

Objects from the Class

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

Slots

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

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

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

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

Methods

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

See Also

`H2OGLMGrid`, `H2OKMeansGrid`, `H2ODRFGrid`, `H2OGBMGrid`, `H2ODeepLearningGrid`

Examples

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

H2OGridVA-class *Class* "H2OGridVA"

Description

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

Objects from the Class

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

Slots

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

`data`: Object of class `"H2OParsedDataVA"`, which is the input data used to build the model.

`model`: Object of class `"list"` containing `"H2OModelVA"` objects representing the models returned by the grid search algorithm.

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

Methods

show `signature(object = "H2OGridVA")`: ...

See Also

H2OGLMGridVA

Examples

```
showClass("H2OGridVA")
```

H2OKMeansGrid-class *Class* "H2OKMeansGrid"

Description

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

Objects from the Class

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

Slots

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

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

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

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

Extends

Class `"H2OGrid"`, directly.

Methods

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

See Also

H2OKMeansModel, h2o.kmeans

Examples

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

H2OKMeansModel-class *Class* "H2OKMeansModel"

Description

A class for representing k-means models.

Objects from the Class

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

Slots

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

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

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

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

Methods

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

See Also

h2o.kmeans

Examples

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

H2OKMeansModelVA-class

Class "H2OKMeansModelVA"

Description

A class for representing k-means clustering models built on ValueArray data.

Objects from the Class

Objects can be created by calls of the form `new("H2OKMeansModelVA", ...)`.

Slots

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

`data`: Object of class `H2OParsedDataVA`, which is the input data used to build the model.

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

- `cluster`: A `H2OParsedDataVA` object, which contains the cluster assignment for each observation in the input data.
- `centers`: A matrix of cluster centers.
- `withinss`: Within-cluster sum of squared errors for each cluster.
- `tot.withinss`: Sum total within-cluster sum of squared errors.
- `size`: Number of observations in each cluster.

Extends

Class `"H2OModelVA"`, directly.

Methods

show `signature(object = "H2OKMeansModelVA")`: ...

See Also

`h2o.kmeans`

Examples

```
showClass("H2OKMeansModelVA")
```

H2OModel-class *Class "H2OModel"*

Description

Object representing the model built by an H2O algorithm on a FluidVecs dataset.

Objects from the Class

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

Slots

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

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

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

Methods

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

See Also

H2OGLMModel, H2OKMeansModel, H2ODRFModel, H2OGBMModel, H2OPCAModel, H2ODeepLearningModel

Examples

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

H2OModelVA-class *Class "H2OModelVA"*

Description

Object representing the model built by an H2O algorithm on a ValueArray dataset.

Objects from the Class

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

Slots

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

data: Object of class "H2OParsedDataVA", which is the input data used to build the model.

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

Methods

No methods defined with class "H2OModelVA" in the signature.

See Also

H2OGLMModelVA, H2OKMeansModelVA, H2ORFModelVA

Examples

```
showClass("H2OModelVA")
```

H2ONBModel-class *Class "H2ONBModel"*

Description

A class for representing naive Bayes models.

Objects from the Class

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

Slots

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

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

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

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

Extends

Class "H2OModel", directly.

Methods

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

See Also

h2o.naiveBayes

Examples

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

H2OParsedData-class *Class* "H2OParsedData"

Description

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

Objects from the Class

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

Slots

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

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

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

Methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

apply signature(X = "H2OParsedData"): ...

as.data.frame signature(x = "H2OParsedData"): ...

as.factor signature(x = "H2OParsedData"): ...

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

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

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

colnames<- signature(x = "H2OParsedData", value = "character"): ...

colnames<- signature(x = "H2OParsedData", value = "H2OParsedData"): ...

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

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

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

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

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

h2o.cut signature(x = "H2OParsedData", breaks = "numeric"): ...

h2o<- signature(x = "H2OParsedData", value = "H2OParsedData"): ...

h2o<- signature(x = "H2OParsedData", value = "numeric"): ...

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

histograms signature(object = "H2OParsedData"): ...

ifelse signature(test = "H2OParsedData"): ...

is.factor signature(x = "H2OParsedData"): ...

is.na signature(x = "H2OParsedData"): ...

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

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

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

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

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

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

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

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

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

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

show signature(object = "H2OParsedData"): ...

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

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

summary signature(object = "H2OParsedData"): ...

t signature(object = "H2OParsedData"): ...

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

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

See Also

H2ORawData, h2o.parseRaw

Examples

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

H2OParsedDataVA-class *Class* "H2OParsedDataVA"

Description

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

Objects from the Class

Objects can be created by calls of the form `new("H2OParsedDataVA", ...)`.

Slots

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

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

logic: Object of class "logical", indicating whether the "H2OParsedDataVA" object represents logical data

Extends

Class "H2OParsedData", directly.

Methods

colnames signature(x = "H2OParsedDataVA"): ...

colnames<- signature(x = "H2OParsedDataVA", value = "character"): ...

colnames<- signature(x = "H2OParsedDataVA", value = "H2OParsedDataVA"): ...

dim signature(x = "H2OParsedDataVA"): ...

head signature(x = "H2OParsedDataVA"): ...

names signature(x = "H2OParsedDataVA"): ...

names<- signature(x = "H2OParsedDataVA"): ...

ncol signature(x = "H2OParsedDataVA"): ...

nrow signature(x = "H2OParsedDataVA"): ...

show signature(object = "H2OParsedDataVA"): ...

summary signature(object = "H2OParsedDataVA"): ...

tail signature(x = "H2OParsedDataVA"): ...

See Also

H2ORawDataVA, h2o.parseRaw.VA

Examples

```
showClass("H2OParsedDataVA")
```

H2OPCAModel-class *Class* "H2OPCAModel"

Description

A class for representing principal components analysis results.

Objects from the Class

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

Slots

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

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

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

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

Extends

Class "H2OModel", directly.

Methods

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

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

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

See Also

h2o.prcomp

Examples

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

H2OPerfModel-class *Class* "H2OPerfModel"

Description

A class for constructing performance measures of H2O models.

Objects from the Class

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

Slots

cutoffs: A numeric vector of threshold values.

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

perf: A character string indicating the performance measure used to evaluate the model. One of either "F1", "accuracy", "precision", "recall", "specificity", or "max_per_class_error". See `h2o.performance` for a detailed description of each.

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

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

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

Methods

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

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

See Also

`h2o.performance`, `plot.H2OPerfModel`

Examples

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

H2ORawData-class *Class* "H2ORawData"

Description

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

Objects from the Class

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

Slots

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

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

Methods

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

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

See Also

H2OParsedData

Examples

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

H2ORawDataVA-class *Class* "H2ORawDataVA"

Description

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

Objects from the Class

Objects can be created by calls of the form `new("H2ORawDataVA", ...)`.

Slots

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

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

Methods

h2o.parseRaw.VA `signature(data = "H2OParsedDataVA", key = "character", header = "logical" sep = "charac`
...

show `signature(object = "H2ORawDataVA")`: ...

See Also

H2OParsedDataVA

Examples

```
showClass("H2ORawDataVA")
```

H2ORFModelVA-class *Class* "H2ORFModelVA"

Description

A class for representing random forest ensembles built on ValueArray data.

Objects from the Class

Objects can be created by calls of the form `new("H2ORFModelVA", ...)`.

Slots

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

`data`: Object of class H2OParsedDataVA, which is the input data used to build the model.

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

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

Extends

Class "H2OModelVA", directly.

Methods

show `signature(object = "H2ORFModelVA")`: ...

See Also

h2o.randomForest

Examples

```
showClass("H2ORFModelVA")
```

H2OSpeeDRFModel-class *Class* "H2OSpeeDRFModel"

Description

A class for representing single-node random forest ensembles built on FluidVecs data.

Objects from the Class

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

Slots

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

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

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

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

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

Extends

Class `"H2OModel"`, directly.

Methods

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

See Also

`h2o.SpeeDRF`

Examples

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

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

Description

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

Usage

```
## S3 method for class 'H2OParsedData'
head(x, n = 6L, ...)
## S3 method for class 'H2OParsedData'
tail(x, n = 6L, ...)
## S3 method for class 'H2OParsedDataVA'
head(x, n = 6L, ...)
## S3 method for class 'H2OParsedDataVA'
tail(x, n = 6L, ...)
```

Arguments

x	An H2O parsed data object.
n	(Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x.
...	Arguments to be passed to or from other methods. (Currently unimplemented).

Value

A data frame containing the first or last n rows of an H2OParsedData object.

Examples

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

`ifelse` *Applies conditional statements to an* `H2OParsedData` `object.`

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Examples

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

is.factor — *Tells user if given column is categorical data or not.*

Description

Tells user if given column is categorical data or not.

Usage

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

Arguments

x	Columns of an H2O parsed data object.

Value

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

Examples

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

levels — *Levels of Categorical Data*

Description

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

Usage

```
levels(x)
```

Arguments

x	Column of categorical data in an H2OParsedData object.

Value

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

Examples

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

`mean.H2OParsedData` *Arithmetic Mean of H2O Dataset*

Description

`mean.H2OParsedData`, a method for the `mean` generic. Calculate the mean of each numeric column in a H2O dataset.

Usage

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

Arguments

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

Value

An `H2OParsedData` object of scalar numeric value representing the arithmetic mean of each numeric column of `x`. If `x` is not logical or numeric, then `NA_real_` is returned, with a warning.

Examples

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

nrow *The Number of Rows/Columns of a H2O Dataset*

Description

Returns a count of the number of rows in an H2OParsedData object.

Usage

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

Arguments

x	An H2OParsedData object.

Value

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

See Also

dim which returns all dimensions

Examples

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

plot.H2OPerfModel *Scatterplot of H2O Performance Measures*

Description

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

Usage

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

Arguments

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

See Also

H2OPerfModel, h2o.performance

Examples

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

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

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

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

---

quantile.H2OParsedData

Obtain and display quantiles for H2O parsed data.

Description

quantile.H2OParsedData, a method for the quantile generic. Obtain and return quantiles for an H2OParsedData object.

Usage

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

Arguments

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

Details

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

Value

A vector describing the percentiles at the given cutoffs for the H2OParsedData object.

Examples

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

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

---

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

Description

screeplot.H2OPCAModel, a method for the screeplot generic. Plots the variances against the number of the principal component generated by h2o.prcomp.

Usage

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

Arguments

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

Examples

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

---

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

Description

Calculates the standard deviation of a H2OParsedData column of continuous real valued data.

Usage

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

Arguments

x	An H2OParsedData object containing numeric data.
na.rm	Logical value where FALSE does not remove NA's in the calculation and TRUE removes NA's in the calculation.

Value

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

Examples

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

str *Display the Structure of a H2O Dataset*

Description

A method for the str generic. Obtain information about H2O parsed data objects and their structure.

Usage

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

Arguments

object	An H2OParsedData object.
...	Potential further arguments. (Currently unimplemented).

Value

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

Examples

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

sum *Sum of Numeric Values*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

---

summary *Summarizes the columns of a H2O Dataset*

Description

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

Usage

```
## S3 method for class 'H2OParsedData'
summary(object, ...)
## S3 method for class 'H2OParsedDataVA'
summary(object, ...)
```

Arguments

object	An H2OParsedData object.
...	Additional arguments affecting the summary produced. (Currently unimplemented).

Value

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

Examples

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

`summary.H2OPCAModel` *Summarizes the H2O PCA Model*

Description

`summary.H2OPCAModel`, a method for the `summary` generic. Summarizes the importance of each principal component returned by `h2o.prcomp`.

Usage

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

Arguments

`object`	An `H2OPCAModel` object.
`...`	Additional arguments affecting the summary produced. (Currently unimplemented).

Value

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

Examples

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

unique.H2OParsedData *Extract Unique Elements from H2O Dataset*

Description

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

Usage

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

h2o.unique(x, incomparables = FALSE, ...)
```

Arguments

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

Details

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

Value

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

Examples

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