Package 'h2o'

November 29, 2013

R topics documented:

`h2o-package` *Installer for H2O R Package*

Description

This wrapper connects to an H2O instance and installs the corresponding R package "h2o", that allows users to run H2O via its REST API from within R. To communicate, the version of the R package must match the version of H2O, so when connecting to a new H2O cluster, it is necessary to re-run the initializer.

Details

Package:	h2o
Type:	Package
Version:	1.0.3
Date:	2013-09-13
License:	Apache-2
Depends:	R (>= 2.13.0), RCurl, rjson, tools

You can install the package h2o by going here: http://0xdata.com/h2O/. Click the "Download H2O" button. Once the H2O file has downloaded call install.packages("~/h2o_1.0.3.tar.gz", repos= NULL, type = "source"). Note that you will need to provide a path to the .tar.gz file within the quotation marks in order for R to correctly find the package. Alternatively users can also change their working directory to the same file where the downloaded .tar.gz file is found, and specify just the file name.

First, you must have H2O installed and running on a computer/cluster. See the references for more

details. If H2O is running on your local machine, call `h2o.init` with the IP and port that H2O is communicating on. By default, this is `http://localhost:54321`, where the IP is "localhost" and the port is "54321". If H2O is running on a cluster, you must provide the IP and port of the remote machine.

Author(s)

Anqi Fu

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

References

- 0xdata Homepage
- H2O Documentation
- H2O on Github

Examples

```
# Install H2O R package dependencies
h2o.installDepPkgs()

# 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 localhost and you must
# have installed with the Windows or Mac installer package so H2O is in
# a known place.  startH2O requires the port to be 54321.
myIP = "localhost"
myPort = 54321
localH2O = h2o.init(ip = myIP, port = myPort, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = TRUE)

# Import iris dataset into H2O and print summary
irisPath = system.file("extdata", "iris.csv", package="h2oRClient")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)

# Attach H2O R package and run GLM demo
library(h2oRClient)
??h2o
demo(package = "h2oRClient")
demo(h2o.glm)
```

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

Description

Connects to a H2O instance and checks the local H2O R package is the correct version.

Usage

```
h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = FALSE, promptUpgrade = TRUE)
```

Arguments

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.
startH2O	A logical value indicating whether to start the H2O launcher GUI if no connection with H2O is detected. This is only possible if H2O was installed from the InstallBuilder executable and ip = "localhost" or ip = "127.0.0.1".
silentUpgrade	A logical value indicating whether to automatically install the H2O R package from the H2O server if a version mismatch is detected.
promptUpgrade	A logical value indicating whether to prompt the user to install the H2O R package from the H2O server if a version mismatch is detected. Ignored when silentUpgrade = TRUE.

Details

This method first checks if H2O is connectable. If it cannot connect and startH2O = TRUE with IP of localhost, it will attempt to start the H2O launcher GUI, which is installed with the H2O InstallBuilder executable. Otherwise, it stops immediately with an error. 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.

Examples

```
# Try to create a localhost connection to H2O. Launch H2O GUI if needed.
localH2O = h2o.init()
localH2O = h2o.init(ip = "localhost", port = 54321)

# Prompt to install H2O R package from server if version mismatch
localH2O = h2o.init(ip = "localhost", port = 54321, silentUpgrade = FALSE, promptUpgrade = TRUE)

# Automatically install H2O R package from server if version mismatch
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = FALSE, silentUpgrade = TRUE, promptUpgrade = FALSE)
```

`h2o.installDepPkgs` *Install H2O R Package Dependencies*

Description

Installs all R packages that the H2O R package requires for correct functionality. This method must be executed first on every new machine before attempting to communicate with H2O from R.

Usage

```
h2o.installDepPkgs(optional = FALSE)
```

Arguments

`optional`	A logical value indicating whether to install optional packages for visualizing data.

Details

The R packages required to run H2O are RCurl, rjson, uuid, and tools. All packages are installed from CRAN. The package RCurl needs the latest version of libcurl. See the RCurl FAQ for details.

If `optional = TRUE`, the packages fpc and cluster, along with all associated dependencies, will be installed. Note that this requires R version 3.0 or higher.

Examples

```
h2o.installDepPkgs()
```

Package 'h2oRClient'

November 29, 2013

R topics documented:

h2oRClient-package *H2O R Interface*

Description

This is a package for running H2O via its REST API from within R.

Details

Package:	h2oRClient
Type:	Package
Version:	2.0.1.2
Date:	2013-07-16
License:	Apache-2
Depends:	R (>= 2.13.0), RCurl, rjson

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). Load the library and create a `H2OClient` object with the server IP and port. The command `h2o.importFile` will import and parse a delimited data file, returning an `H2OParsedData` object.

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 parsed data and parameters (response variable, error distribution) as arguments. (The operation will be done on the server associated with the data object).

Note that no actual data is stored in the workspace - R only saves the hex keys, 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 on Github

Examples

```
library(h2oRClient)
localH2O = new("H2OClient", ip = "localhost", port = 54321)
h2o.checkClient(localH2O)
```

```
prostate.hex = h2o.importURL(localH2O, path = "https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostat
summary(prostate.hex)
prostate.glm = h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data = prostate.hex, family = "binomial",
print(prostate.glm)
h2o.kmeans(data = prostate.hex, centers = 5)
```

`h2o.checkClient` *Check if H2O is Running*

Description

Checks if H2O is running on the specified IP and port, and stops the program if it is not. Th method also compares the version number of the H2O program and the installed h2o R package. If there is a mismatch, it will warn the user.

Usage

```
h2o.checkClient(object)
```

Arguments

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

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,
+ silentUpgrade = TRUE, promptUpgrade = FALSE)
h2o.checkClient(localH2O)
```

`h2o.gbm` *H2O: GBM*

Description

Builds gradient boosed classification trees on a parsed data set.

Usage

```
h2o.gbm(x, y, distribution = "multinomial", data, n.trees = 10, interaction.depth = 8,
+ n.minobsinnode = 10, shrinkage = 0.2)
```

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, categorization is "multinomial" (default), "gaussian" used for regression.
data	An H2OParsedData object containing the variables in the model.
n.trees	Number of trees to grow. Must be a nonnegative integer.
interaction.depth	Maximum depth to grow the tree.
n.minobsinnode	Minimum number of rows to assign to teminal nodes.
shrinkage	A learning-rate parameter defining step size reduction.

Value

An object of class H2OGBM with slots key, data, and model, where the last is a list of the following components:

type	The type of the tree, which currently must be classification.
n.trees	Numbe r 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.

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)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321,
+ startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
# Run classification GBM on CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate.hex = h2o.importURL(localH2O, path = "https://raw.github.com/0xdata/h2o/
+ master/smalldata/logreg/prostate.csv", key = "prostate.hex")
h2o.gbm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"), data = prostate.hex,
+ n.trees = 100, interaction.depth = 8, n.minobsinnode = 10, shrinkage = 0.2)
# Run regression GBM on CAPSULE ~ AGE + RACE + PSA + DCAPS
myX = setdiff(colnames(prostate.hex), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.gbm(y = "VOL", x = myX, distribution = "gaussian", data = prostate.hex,
+ n.trees = 10, interaction.depth = 5, shrinkage = 0.1)
```

---

h2o.getTree *Get Tree from Random Forest Model*

Description

Returns the depth and number of leaves of a particular tree in the random forest ensemble.

Usage

```
h2o.getTree(forest, k)
```

Arguments

forest	An H2ORForestModel object indicating the random forest model to examine.
k	The particular tree to retrieve. (Must be an integer between 1 and ntree).

See Also

h2o.randomForest, H2ORForestModel

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2oWrapper.init(ip = "localhost", port = 54321, startH2O = TRUE,
+ silentUpgrade = TRUE, promptUpgrade = FALSE)
irisPath = system.file("extdata", "iris.csv", package="h2oRClient")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
iris.rf = h2o.randomForest(y = 4, data = iris.hex, ntree = 50, depth = 100)
h2o.getTree(forest = iris.rf, k = 5)
```

h2o.glm *H2O: Generalized Linear Model*

Description

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

Usage

```
h2o.glm(x, y, data, family, nfolds = 10, alpha = 0.5, lambda = 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.
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	Number of folds for cross-validation. The default is 10.
alpha	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).
tweedie.p	The index of the power variance function for the tweedie distribution. Only used if family = "tweedie".

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).

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE,
+ promptUpgrade = FALSE)
# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate.hex = h2o.importURL(localH2O,
+ path = "https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostate.csv",
+ 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.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

```
h2o.importFile(object, path, key = "", parse = TRUE)
```

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.

Details

WARNING: In H2O, import is lazy! Do not modify the data on hard disk until after parsing is complete. By default, h2o.importFile will automatically parse the file.

Value

If parse = TRUE, the function returns an object of class H2OParsedData, otherwise it returns an object of class H2ORawData.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,
+ silentUpgrade = TRUE, promptUpgrade = FALSE)
irisPath = system.file("extdata", "iris.csv", package="h2oRClient")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)
```

---

h2o.importFolder *Import Local Directory*

Description

Imports all the files in the local directory and parses them, returning a list of objects containing the identifying hex keys.

Usage

```
h2o.importFolder(object, path, parse = TRUE)
```

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.

Details

WARNING: In H2O, import is lazy! Do not modify the data files on hard disk until after parsing is complete. By default, h2o.importFolder will automatically parse all files in the directory.

Value

If parse = TRUE, the function returns a list of objects of class H2OParsedData, otherwise it returns a list of objects of class H2ORawData.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
myPath = paste(path.package("h2oRClient"), "extdata", sep="/")
all_files.hex = h2o.importFolder(localH2O, path = myPath)
for(i in 1:length(all_files.hex))
  print(summary(all_files.hex[[i]]))
```

---

h2o.importHDFS *Import from HDFS*

Description

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

Usage

```
h2o.importHDFS(object, path, parse = TRUE)
```

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.
parse	(Optional) A logical value indicating whether the file should be parsed after import.

Details

WARNING: In H2O, import is lazy! Do not modify the data files on hard disk until after parsing is complete. By default, h2o.importHDFS will automatically parse the file.

Value

When the path is a directory, if `parse = TRUE`, the function returns a list of objects of class `H2OParsedData`, otherwise it returns a list of objects of class `H2ORawData`. When the path is a single file, the same holds, except the function returns a single object rather than a list.

See Also

`h2o.importFolder`, `h2o.importFile`, `h2o.importURL`

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE,
+ promptUpgrade = FALSE)
covtype.hex = h2o.importHDFS(localH2O, path = "hdfs://192.168.1.173:54321/0xdiag/datasets/standard/covtype.data"
+ parse = TRUE)
summary(covtype.hex)

stdfolder.data = h2o.importHDFS(localH2O, path = "hdfs://192.168.1.173:54321/0xdiag/datasets/standard",
+ parse = FALSE)
stdfolder.parsed = vector("list", length(stdfolder.data))
for(i in 1:length(stdfolder.data))
  stdfolder.parsed[[i]] = h2o.parseRaw(stdfolder.data[[i]])
```

---

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

```
h2o.importURL(object, path, key = "", parse = TRUE)
```

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.

Details

WARNING: In H2O, import is lazy! Do not modify the data on hard disk until after parsing is complete. By default, h2o.importURL will automatically parse the file.

Value

If `parse = TRUE`, the function returns an object of class `H2OParsedData`, otherwise it returns an object of class `H2ORawData`.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
prostate.hex = h2o.importURL(localH2O, path = "https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostat
summary(prostate.hex)
```

---

h2o.kmeans *H2O: K-Means Clustering*

Description

Performs k-means clustering on a parsed data file.

Usage

```
h2o.kmeans(data, centers, cols = "", iter.max = 10)
```

Arguments

data	An `H2OParsedData` 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	The maximum number of iterations allowed.

Value

An object of class H2OKMeansModel with slots key, data, and model, where the last is a list of the following components:

centers	A matrix of cluster centers.
cluster	A H2OParsedData 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).

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
prosPath = system.file("extdata", "prostate.csv", package="h2oRClient")
prostate.hex = h2o.importFile(localH2O, path = prosPath)
h2o.kmeans(data = prostate.hex, centers = 10, cols = c("AGE", "RACE", "VOL", "GLEASON"))
covPath = system.file("extdata", "covtype.csv", package="h2oRClient")
covtype.hex = h2o.importFile(localH2O, path = covPath)
covtype.km = h2o.kmeans(data = covtype.hex, centers = 5, cols = c(1, 2, 3))
print(covtype.km)
```

---

h2o.parseRaw *Parse Raw Data File*

Description

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

Usage

```
h2o.parseRaw(data, key = "")
```

Arguments

data	An H2ORawData object to be parsed.
key	(Optional) The hex key assigned to the parsed file.

Details

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

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
prosPath = system.file("extdata", "prostate.csv", package="h2oRClient")
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, 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`	Number of folds for cross-validation. The default is 10.
`alpha`	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).
`tweedie.p`	The index of the power variance function for the tweedie distribution. Only used if `family = "tweedie"`

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).

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
# Run PCR on Prostate Data
prostate.hex = h2o.importURL(localH2O, path = "https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostat
+ key = "prostate.hex")
h2o.pcr(x = c("AGE","RACE","PSA","DCAPS"), y = "CAPSULE", data = prostate.hex, family = "binomial",
+ nfolds = 10, alpha = 0.5)
```

---

h2o.prcomp *Principal Components Analysis*

Description

Performs principal components analysis on the given data set.

Usage

```
h2o.prcomp(data, tol = 0, standardize = TRUE)
```

Arguments

data	An H2OParsedData object on which to run principal components analysis.
tol	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.
standardize	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.

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.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
ausPath = system.file("extdata", "australia.csv", package="h2oRClient")
australia.hex = h2o.importFile(localH2O, path = ausPath)
australia.pca = h2o.prcomp(data = australia.hex, standardize = TRUE)
print(australia.pca)
summary(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 object for which prediction is desired.
newdata	(Optional) A H2OParsedData object in which to look for variables with which to predict. If omitted, the data used to fit the model object@data are used.

Value

A H2OParsedData object containing the predictions.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
# Run GLM of CAPSULE ~ AGE + RACE + PSA + DCAPS
prostate.hex = h2o.importURL(localH2O, path = "https://raw.github.com/0xdata/h2o/master/smalldata/logreg/prostat
+ 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 parsed data set.

Usage

```
h2o.randomForest(y, x, data, ntree = 50, depth = 2147483647, classwt = as.numeric(NA))
```

Arguments

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).
x	A vector containing the names or indices of the predictor variables to use in building the random forest model.
data	An H2OParsedData object containing the variables in the model.
ntree	Number of trees to grow. (Must be a nonnegative integer).
depth	Maximum depth to grow the tree.
classwt	(Optional) Priors of the classes. Need not add up to one. If missing, defaults to all weights set at 1.0.

Details

Currently, only classification trees are supported. Note that indexing begins at zero, so for example, to specify the first column as the response variable, set `y = 0`.

Value

An object of class H2ORForestModel with slots key, data, and model, where the last is a list of the following components:

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.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
irisPath = system.file("extdata", "iris.csv", package="h2oRClient")
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,
classwt = c("Iris-versicolor" = 20.0, "Iris-virginica" = 30.0))
covPath = system.file("extdata", "covtype.csv", package="h2oRClient")
covtype.hex = h2o.importFile(localH2O, path = covPath, key = "covtype.hex")
h2o.randomForest(y = "Cover_Type", x = setdiff(colnames(covtype.hex), c("Cover_Type", "Aspect", "Hillshade_9am"))
```

h2o.setColNames *Set Column Names of Data Set*

Description

Sets the column names of a parsed data set to the values provided in a separate file.

Usage

```
h2o.setColNames(data, col.names)
```

Arguments

data	An H2OParsedData object containing the data set whose column names will be changed.
col.names	An H2OParsedData object containing the new column names.

Details

This function modifies the data file directly. It does not create a new copy, and thus does not take up much additional memory. If the number of columns in data and col.names do not match, then it will simply set the names of the provided columns (from left to right) and skip the rest.

Examples

```
library(h2o)
h2o.installDepPkgs()
localH2O = h2o.init(ip = "localhost", port = 54321, startH2O = TRUE, silentUpgrade = TRUE, promptUpgrade = FALSE)
irisPath = system.file("extdata", "iris.csv", package="h2oRClient")
iris.hex = h2o.importFile(localH2O, path = irisPath, key = "iris.hex")
summary(iris.hex)
namesPath = system.file("extdata", "iris_header.csv", package="h2oRClient")
names.hex = h2o.importFile(localH2O, path = namesPath, key = "iris_names.hex")
h2o.setColNames(data = iris.hex, col.names = names.hex)
summary(iris.hex)
```

h2o.startLauncher *Launch H2O GUI*

Description

Launches the H2O graphical user interface from inside R.

Usage

```
h2o.startLauncher()
```

Details

Upon launch, use the graphical user interface to configure the IP, port, and amount of memory to allot H2O. Hit the "Start H2O" button to run the program with the settings. If no errors occur, a browser window will pop up with the H2O start page.

Examples

```
h2o.startLauncher()
```

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 = "logical")`
...

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

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

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

Examples

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

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.

Methods

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

Examples

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

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.

Methods

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

Examples

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

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")`: ...

Examples

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

H2OModel-class *Class* "H2OModel"

Objects from the Class

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

Slots

`key`: Object of class `"character"` ~~

`data`: Object of class `"H2OParsedData"` ~~

`model`: Object of class `"list"` ~~

Methods

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

See Also

`H2OGLMModel`, `H2OKMeansModel`, `H2ORForestModel`, `H2OGBMModel`, `H2OPCAModel`

Examples

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

H2OParsedData-class *Class* "H2OParsedData"

Description

A class for representing imported 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.

Methods

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

h2o.gbm signature(x = "character", y = "character", data = "H2OParsedData", n.trees = "numeric", inte
...

h2o.glm signature(x = "character", y = "character", data = "H2OParsedData", family = "character", nfo
...

h2o.kmeans signature(data = "H2OParsedData", centers = "numeric", cols = "character", iter.max = "num
...

h2o.prcomp signature(data = "H2OParsedData", tol = "numeric", standardize = "logical", retx = "logical'
...

h2o.randomForest signature(y = "character", data = "H2OParsedData", ntree = "numeric"):
...

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

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

Examples

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

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).

Methods

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

Examples

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

H2ORawData-class *Class* "H2ORawData"

Description

A class for representing imported 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" ~~

key: Object of class "character" ~~

Methods

parseRaw `signature(data = "H2ORawData", key = "character")`: ...

parseRaw `signature(data = "H2ORawData", key = "missing")`: ...

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

Examples

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

H2ORForestModel-class *Class* "H2ORForestModel"

Description

A class for representing random forest ensembles.

Objects from the Class

Objects can be created by calls of the form `new("H2ORForestModel", ...)`.

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.

Methods

h2o.getTree `signature(forest = "H2ORForestModel", k = "numeric")`: ...

show `signature(object = "H2ORForestModel")`: ...

Examples

```
showClass("H2ORForestModel")
```