Package ‘h2o’

June 17, 2018

Version 3.20.0.2

Type Package

Title R Interface for ‘H2O’

Date 2018-06-15

Description R interface for ‘H2O’, the scalable open source machine learning platform that offers parallelized implementations of many supervised and unsupervised machine learning algorithms such as generalized linear models, gradient boosting machines (including xgboost), random forests, deep neural networks (deep learning), stacked ensembles, naive bayes, cox proportional hazards, k-means, PCA, word2vec, as well as a fully automatic machine learning algorithm (AutoML).

License Apache License (== 2.0)

URL https://github.com/h2oai/h2o-3

BugReports https://0xdata.atlassian.net/projects/PUBDEV

NeedsCompilation no

SystemRequirements Java (>= 7)

Depends R (>= 2.13.0), methods, stats

Imports graphics, tools, utils, RCurl, jsonlite

Suggests ggplot2, mlbench, Matrix, slam, bit64 (>= 0.9.7), data.table (>= 1.9.8), survival

Collate 'aggregator.R' 'astfun.R' 'autml.R' 'classes.R' 'config.R'
 'connection.R' 'constants.R' 'datasets.R' 'logging.R'
 'communication.R' 'kvstore.R' 'frame.R' 'import.R' 'parse.R'
 'export.R' 'edicts.R' 'models.R' 'coxph.R' 'coxphutils.R'
 'kmeans.R' 'gbm.R' 'glm.R' 'glrm.R' 'pca.R' 'svd.R'
 'deeplearning.R' 'stackedensemble.R' 'deepwater.R' 'xgboost.R'
 'randomforest.R' 'naivebayes.R' 'word2vec.R' 'w2vutils.R'
 'locate.R' 'grid.R' 'predict.R' 'zzz.R'

RoxygenNote 6.0.1
Author Erin LeDell [aut, cre],
Navdeep Gill [aut],
Spencer Aiello [aut],
Anqi Fu [aut],
Arno Candel [aut],
Cliff Click [aut],
Tomas Nykodym [aut],
Patrick Aboyoun [aut],
Michal Kurka [aut],
Michal Malohlava [aut],
Ludi Rehak [ctb],
Eric Eckstrand [ctb],
Brandon Hill [ctb],
Sebastian Vidrio [ctb],
Surekha Jadhawani [ctb],
Amy Wang [ctb],
Raymond Peck [ctb],
Wendy Wong [ctb],
Jan Gorecki [ctb],
Matt Dowle [ctb],
Yuan Tang [ctb],
Lauren DiPerna [ctb],
H2O.ai [cph, fnd]

Maintainer Erin LeDell <erin@h2o.ai>

Repository CRAN

Date/Publication 2018-06-17 15:50:32 UTC

R topics documented:

h2o-package ......................................................... 9
aaa ................................................................. 10
apply ............................................................... 10
as.character.H2OFrame ........................................... 11
as.data.frame.H2OFrame .......................................... 12
as.factor ......................................................... 12
as.h2o ............................................................. 13
as.matrix.H2OFrame ................................................ 14
as.numeric ....................................................... 15
as.vector.H2OFrame ................................................ 16
australia .......................................................... 16
colnames .......................................................... 17
dim.H2OFrame ..................................................... 17
dimnames.H2OFrame ............................................... 18
generate_col_ind .................................................. 18
h2o.abs .......................................................... 19
h2o.acos .......................................................... 20
h2o.aggregated_frame ......................................................... 20
h2o.aggregator .................................................................. 21
h2o.aic .............................................................................. 22
h2o.all .............................................................................. 23
h2o.anomaly ..................................................................... 23
h2o.any .............................................................................. 24
h2o.anyFactor ................................................................... 25
h2o.arrange ....................................................................... 25
h2o.ascharacter .................................................................. 26
h2o.asfactor ...................................................................... 26
h2o.asnumeric .................................................................... 27
h2o.assign ........................................................................ 27
h2o.as_date ....................................................................... 28
h2o.auc .............................................................................. 28
h2o.automl ........................................................................ 29
h2o.betweenss ................................................................... 32
h2o.biases ........................................................................ 32
h2o.bottomN ..................................................................... 33
h2o.cbind .......................................................................... 33
h2o.ceiling ....................................................................... 34
h2o.centers ....................................................................... 34
h2o.centersSTD .................................................................. 35
h2o.centroid_stats ............................................................... 35
h2o.clearLog ..................................................................... 36
h2o.clusterInfo ................................................................. 36
h2o.clusterIsUp .................................................................. 37
h2o.clusterStatus ............................................................... 37
h2o.cluster_sizes ............................................................... 38
h2o.coef ............................................................................ 38
h2o.coef_norm ................................................................... 39
h2o.colnames .................................................................... 39
h2o.columns_by_type .......................................................... 40
h2o.computeGram ............................................................... 40
h2o.confusionMatrix ............................................................ 41
h2o.connect ...................................................................... 42
h2o.cor .............................................................................. 43
h2o.cos .............................................................................. 44
h2o.cosh .......................................................................... 45
h2o.coxph ........................................................................ 45
h2o.createFrame ................................................................. 46
h2o.cross_validation_fold_assignment ................................ 48
h2o.cross_validation_holdout_predictions ............................. 48
h2o.cross_validation_models .............................................. 49
h2o.cross_validation_predictions ....................................... 49
h2o.cummax ..................................................................... 50
h2o.cummin ..................................................................... 50
h2o.cumprod .................................................................... 51
h2o.cumsum ..................................................................... 51
R topics documented:

- `h2o.cut` .......................................................... 52
- `h2o.day` ..................................................... 53
- `h2o.dayOfWeek` ............................................ 53
- `h2o.dct` ..................................................... 54
- `h2o.ddply` .................................................. 55
- `h2o.decryptionSetup` ...................................... 56
- `h2o.deepfeatures` ........................................ 57
- `h2o.deeplearning` ......................................... 58
- `h2o.deepwater` ............................................. 64
- `h2o.deepwater.available` ............................... 69
- `h2o.describe` ............................................. 69
- `h2o.diffflag1` ........................................... 70
- `h2o.dim` ................................................... 70
- `h2o.dimnames` ............................................ 71
- `h2o.distance` .............................................. 71
- `h2o.downloadAllLogs` ...................................... 72
- `h2o.downloadCSV` .......................................... 72
- `h2o.download_mojo` ....................................... 73
- `h2o.download_pojo` ....................................... 74
- `h2o.entropy` .............................................. 75
- `h2o.exp` ................................................... 76
- `h2o.exportFile` ........................................... 76
- `h2o.exportHDFS` .......................................... 77
- `h2o.fillna` ................................................ 77
- `h2o.filterNACols` ........................................ 78
- `h2o.findSynonyms` ....................................... 79
- `h2o.find_row_by_threshold` ............................ 79
- `h2o.find_threshold_by_max_metric` ................... 80
- `h2o.floor` ................................................ 80
- `h2o.flow` .................................................. 81
- `h2o.gainsLift` ............................................ 81
- `h2o.gbm` .................................................. 82
- `h2o.getAutoML` ........................................... 86
- `h2o.getConnection` ........................................ 87
- `h2o.getFrame` ............................................. 87
- `h2o.getFutureModel` ...................................... 88
- `h2o.getGLMFullRegularizationPath` .................... 88
- `h2o.getGrid` .............................................. 89
- `h2o.getId` ................................................. 89
- `h2o.getModel` ............................................ 90
- `h2o.getTimezone` ......................................... 91
- `h2o.getTypes` ............................................ 91
- `h2o.getVersion` ........................................... 91
- `h2o.giniCoef` ............................................ 92
- `h2o.glm` .................................................. 93
- `h2o.glrm` ................................................ 97
- `h2o.grep` .................................................. 100
- `h2o.grid` .................................................. 101
R topics documented:

- h2o.group_by .................................................. 102
- h2o.gsub .......................................................... 103
- h2o.head .......................................................... 104
- h2o.hist .......................................................... 105
- h2o.hit_ratio_table ............................................. 105
- h2o.hour .......................................................... 106
- h2o.ifelse ......................................................... 106
- h2o.importFile .................................................. 107
- h2o.import_sql_select ........................................... 109
- h2o.import_sql_table ............................................ 110
- h2o.impute ....................................................... 111
- h2o.init .......................................................... 112
- h2o.insertMissingValues ....................................... 114
- h2o.interaction ................................................ 115
- h2o.isax .......................................................... 116
- h2o.ischaracter ................................................ 117
- h2o.isfactor ..................................................... 118
- h2o.isnumeric ................................................... 118
- h2o.is_client .................................................... 119
- h2o.kfold_column ............................................... 119
- h2o.killMinus3 .................................................. 119
- h2o.kmeans ...................................................... 120
- h2o.kurtosis ..................................................... 122
- h2o.levels ....................................................... 122
- h2o.listTimezones .............................................. 123
- h2o.list_all_extensions ....................................... 123
- h2o.list_api_extensions ....................................... 124
- h2o.list_core_extensions ..................................... 124
- h2o.loadModel .................................................. 124
- h2o.log .......................................................... 125
- h2o.log10 ........................................................ 125
- h2o.log1p ........................................................ 126
- h2o.log2 ........................................................ 126
- h2o.logAndEcho ................................................ 127
- h2o.logloss ..................................................... 127
- h2o.ls ............................................................. 128
- h2o.lstrip ....................................................... 128
- h2o.mae .......................................................... 129
- h2o.makeGLMModel ............................................. 130
- h2o.make_metrics ............................................. 130
- h2o.match ....................................................... 131
- h2o.max .......................................................... 132
- h2o.mean ........................................................ 132
- h2o.mean_per_class_error .................................... 133
- h2o.mean_residual_deviance .................................. 134
- h2o.median ...................................................... 135
- h2o.merge ........................................................ 136
- h2o.metric ....................................................... 137
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2o.min</td>
<td>138</td>
</tr>
<tr>
<td>h2o.mkt ime</td>
<td>139</td>
</tr>
<tr>
<td>h2o.mojo_predict_csv</td>
<td>139</td>
</tr>
<tr>
<td>h2o.mojo_predict_df</td>
<td>140</td>
</tr>
<tr>
<td>h2o.month</td>
<td>141</td>
</tr>
<tr>
<td>h2o.mse</td>
<td>142</td>
</tr>
<tr>
<td>h2o.nacnt</td>
<td>143</td>
</tr>
<tr>
<td>h2o.naiveBayes</td>
<td>143</td>
</tr>
<tr>
<td>h2o.names</td>
<td>146</td>
</tr>
<tr>
<td>h2o.na_omit</td>
<td>146</td>
</tr>
<tr>
<td>h2o.nchar</td>
<td>147</td>
</tr>
<tr>
<td>h2o.ncol</td>
<td>147</td>
</tr>
<tr>
<td>h2o.networkTest</td>
<td>148</td>
</tr>
<tr>
<td>h2o.nlevels</td>
<td>148</td>
</tr>
<tr>
<td>h2o.no_progress</td>
<td>148</td>
</tr>
<tr>
<td>h2o.nrow</td>
<td>149</td>
</tr>
<tr>
<td>h2o.null_deviance</td>
<td>149</td>
</tr>
<tr>
<td>h2o.null_dof</td>
<td>150</td>
</tr>
<tr>
<td>h2o.num_iterations</td>
<td>150</td>
</tr>
<tr>
<td>h2o.num_valid_substrings</td>
<td>151</td>
</tr>
<tr>
<td>h2o.openLog</td>
<td>151</td>
</tr>
<tr>
<td>h2o.parseLog</td>
<td>152</td>
</tr>
<tr>
<td>h2o.parseSetup</td>
<td>153</td>
</tr>
<tr>
<td>h2o.partialPlot</td>
<td>154</td>
</tr>
<tr>
<td>h2o.performance</td>
<td>155</td>
</tr>
<tr>
<td>h2o.pivot</td>
<td>156</td>
</tr>
<tr>
<td>h2o.prcomp</td>
<td>157</td>
</tr>
<tr>
<td>h2o.predict_json</td>
<td>159</td>
</tr>
<tr>
<td>h2o.print</td>
<td>159</td>
</tr>
<tr>
<td>h2o.prod</td>
<td>160</td>
</tr>
<tr>
<td>h2o.proj_archetypes</td>
<td>160</td>
</tr>
<tr>
<td>h2o.quantile</td>
<td>161</td>
</tr>
<tr>
<td>h2o.r2</td>
<td>162</td>
</tr>
<tr>
<td>h2o.randomForest</td>
<td>163</td>
</tr>
<tr>
<td>h2o.range</td>
<td>167</td>
</tr>
<tr>
<td>h2o.rank_within_group_by</td>
<td>167</td>
</tr>
<tr>
<td>h2o.rbind</td>
<td>169</td>
</tr>
<tr>
<td>h2o.reconstruct</td>
<td>170</td>
</tr>
<tr>
<td>h2o.relevel</td>
<td>171</td>
</tr>
<tr>
<td>h2o.removeAll</td>
<td>172</td>
</tr>
<tr>
<td>h2o.removeVecs</td>
<td>172</td>
</tr>
<tr>
<td>h2o.rep_len</td>
<td>173</td>
</tr>
<tr>
<td>h2o.residual_deviance</td>
<td>173</td>
</tr>
<tr>
<td>h2o.residual_dof</td>
<td>174</td>
</tr>
<tr>
<td>h2o.rm</td>
<td>174</td>
</tr>
<tr>
<td>h2o.rmse</td>
<td>175</td>
</tr>
<tr>
<td>h2o.rmsle</td>
<td>176</td>
</tr>
<tr>
<td>h2o.round</td>
<td>176</td>
</tr>
</tbody>
</table>
R topics documented:

h2o.rstrip .......................................................... 177
h2o.runif ......................................................... 178
h2o.saveModel .................................................. 178
h2o.saveModelDetails ......................................... 179
h2o.saveMojo .................................................... 180
h2o.scale ......................................................... 181
h2o.scoreHistory ............................................... 182
h2o.sd ............................................................ 182
h2o.sdev ........................................................ 183
h2o.setLevels .................................................. 183
h2o.setTimezone ................................................ 184
h2o.show_progress ............................................. 184
h2o.shutdown ................................................... 184
h2o.signif ........................................................ 185
h2o.sin .......................................................... 186
h2o.skewness .................................................... 186
h2o.splitFrame .................................................. 187
h2o.sqrt ........................................................ 188
h2o.stackedEnsemble ......................................... 188
h2o.startLogging ............................................... 190
h2o.std_coef_plot .............................................. 190
h2o.stopLogging ............................................... 191
h2o.str .......................................................... 192
h2o.stringdist .................................................. 192
h2o.strsplit ..................................................... 193
h2o.sub ........................................................ 194
h2o.substring ................................................... 194
h2o.sum .......................................................... 195
h2o.summary ..................................................... 196
h2o.svd .......................................................... 197
h2o.table ........................................................ 198
h2o.tabulate ..................................................... 199
h2o.tan .......................................................... 200
h2o.tanh ........................................................ 200
h2o.target_encode_apply ..................................... 201
h2o.target_encode_create .................................... 202
h2o.toFrame ..................................................... 203
h2o.tokenize ..................................................... 204
h2o.tolower ...................................................... 205
h2o.topN .......................................................... 205
h2o.totss ........................................................ 206
h2o.tot_withinss ............................................... 206
h2o.toupper ....................................................... 207
h2o.transform .................................................... 207
h2o.trim .......................................................... 208
h2o.trunc ........................................................ 209
h2o.unique ....................................................... 209
h2o.var ........................................................... 210
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2o.varimp</td>
<td>211</td>
</tr>
<tr>
<td>h2o.varimp_plot</td>
<td>211</td>
</tr>
<tr>
<td>h2o.week</td>
<td>212</td>
</tr>
<tr>
<td>h2o.weights</td>
<td>213</td>
</tr>
<tr>
<td>h2o.which</td>
<td>213</td>
</tr>
<tr>
<td>h2o.which_max</td>
<td>214</td>
</tr>
<tr>
<td>h2o.which_min</td>
<td>214</td>
</tr>
<tr>
<td>h2o.withinss</td>
<td>215</td>
</tr>
<tr>
<td>h2o.word2vec</td>
<td>215</td>
</tr>
<tr>
<td>h2o.xgboost</td>
<td>216</td>
</tr>
<tr>
<td>h2o.xgboost.available</td>
<td>220</td>
</tr>
<tr>
<td>h2o.year</td>
<td>220</td>
</tr>
<tr>
<td>H2OAutoML-class</td>
<td>221</td>
</tr>
<tr>
<td>H2OClusteringModel-class</td>
<td>221</td>
</tr>
<tr>
<td>H2OConnection-class</td>
<td>222</td>
</tr>
<tr>
<td>H2OConnectionMutableState</td>
<td>223</td>
</tr>
<tr>
<td>H2OCoxPHModel-class</td>
<td>223</td>
</tr>
<tr>
<td>H2OCoxPHModelSummary-class</td>
<td>224</td>
</tr>
<tr>
<td>H2OFrame-class</td>
<td>224</td>
</tr>
<tr>
<td>H2OFrame-Extract</td>
<td>225</td>
</tr>
<tr>
<td>H2OGrid-class</td>
<td>226</td>
</tr>
<tr>
<td>H2OModel-class</td>
<td>227</td>
</tr>
<tr>
<td>H2OModelFuture-class</td>
<td>227</td>
</tr>
<tr>
<td>H2OModelMetrics-class</td>
<td>228</td>
</tr>
<tr>
<td>housevotes</td>
<td>229</td>
</tr>
<tr>
<td>iris</td>
<td>229</td>
</tr>
<tr>
<td>is.character</td>
<td>230</td>
</tr>
<tr>
<td>is.factor</td>
<td>230</td>
</tr>
<tr>
<td>is.h2o</td>
<td>230</td>
</tr>
<tr>
<td>is.numeric</td>
<td>231</td>
</tr>
<tr>
<td>Logical-or</td>
<td>231</td>
</tr>
<tr>
<td>ModelAccessors</td>
<td>232</td>
</tr>
<tr>
<td>names.H2OFrame</td>
<td>233</td>
</tr>
<tr>
<td>Ops.H2OFrame</td>
<td>233</td>
</tr>
<tr>
<td>plot.H2OModel</td>
<td>235</td>
</tr>
<tr>
<td>plot.H2OTabulate</td>
<td>236</td>
</tr>
<tr>
<td>predict.H2OAutoML</td>
<td>237</td>
</tr>
<tr>
<td>predict.H2OModel</td>
<td>238</td>
</tr>
<tr>
<td>predict_leaf_node_assignment.H2OModel</td>
<td>238</td>
</tr>
<tr>
<td>print.H2OFrame</td>
<td>239</td>
</tr>
<tr>
<td>print.H2OTable</td>
<td>240</td>
</tr>
<tr>
<td>prostate</td>
<td>240</td>
</tr>
<tr>
<td>range.H2OFrame</td>
<td>241</td>
</tr>
<tr>
<td>str.H2OFrame</td>
<td>241</td>
</tr>
<tr>
<td>summary.H2OCoxPHModel-method</td>
<td>242</td>
</tr>
<tr>
<td>summary.H2OGrid-method</td>
<td>242</td>
</tr>
<tr>
<td>summary.H2OModel-method</td>
<td>243</td>
</tr>
<tr>
<td>use.package</td>
<td>243</td>
</tr>
</tbody>
</table>
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.

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

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

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

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

**Porting Scripts**

**Author(s)**

Maintainer: The H2O.ai team <tomk@0xdata.com>
References

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

Starting H2O For examples

Examples

```r
if(Sys.info()['sysname'] == "Darwin" && Sys.info()['release'] == '13.4.0')(  
  quit(save="no")
) else{
  h2o.init(nthreads = 2)
}
```

Apply on H2O Datasets

Description

Method for apply on H2OFrame objects.

Usage

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

Arguments

- `X` an H2OFrame object on which `apply` will operate.
- `MARGIN` the vector on which the function will be applied over, either 1 for rows or 2 for columns.
- `FUN` the function to be applied.
- `...` optional arguments to `FUN`.

Value

Produces a new H2OFrame of the output of the applied function. The output is stored in H2O so that it can be used in subsequent H2O processes.
as.character.H2OFrame

See Also

apply for the base generic

Examples

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

as.character.H2OFrame  Convert an H2OFrame to a String

Description

Convert an H2OFrame to a String

Usage

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

Arguments

x  An H2OFrame object

...  Further arguments to be passed from or to other methods.

Examples

h2o.init()
pretrained.frame <- as.h2o(data.frame(
   C1 = c("a", "b"), C2 = c(0, 1), C3 = c(1, 0), C4 = c(0.2, 0.8),
   stringsAsFactors = FALSE))
pretrained.w2v <- h2o.word2vec(pre_trained = pretrained.frame, vec_size = 3)
words <- as.character(as.h2o(c("b", "a", "c", NA, "a")))
vecs <- h2o.transform(pretrained.w2v, words = words)
as.data.frame.H2OFrame

Converts parsed H2O data into an R data frame

Description

Downloads the H2O data and then scans it in to an R data frame.

Usage

```r
## S3 method for class 'H2OFrame'
as.data.frame(x, ...)
```

Arguments

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

Details

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

See Also

- `use.package`

Examples

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

as.factor

Convert H2O Data to Factors

Description

Convert a column into a factor column.

Usage

```r
as.factor(x)
```
as.h2o

Arguments

x a column from an H2OFrame data set.

See Also

as.factor.

Examples

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

as.h2o Create H2OFrame

Description

Import R object to the H2O cloud.

Usage

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

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

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

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

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

Arguments

x An R object.
destination_frame A string with the desired name for the H2OFrame.
... arguments passed to method arguments.
as.matrix.H2OFrame

Details

Method \texttt{as.h2o.data.frame} will use \texttt{fwrite} if data.table package is installed in required version.
To speedup execution time for large sparse matrices, use \texttt{h2o datatable}. Make sure you have installed and imported data.table and slam packages. Turn on \texttt{h2o datatable} by options("h2o.use.data.table"=TRUE)

References

http://blog.h2o.ai/2016/04/fast-csv-writing-for-r/

See Also

use-package

Examples

\begin{verbatim}

h2o.init()
hi <- as.h2o(iris)
he <- as.h2o(euro)
hl <- as.h2o(letters)
hm <- as.h2o(state.x77)
hh <- as.h2o(hi)
stopifnot(is.h2o(hi), dim(hi)==dim(iris),
is.h2o(he), dim(he)==c(length(euro),1L),
is.h2o(hl), dim(hl)==c(length(letters),1L),
is.h2o(hm), dim(hm)==dim(state.x77),
is.h2o(hh), dim(hh)==dim(hi))

if (requireNamespace("Matrix", quietly=TRUE)) {
  data <- rep(0, 100)
data[(1:10)*2] <- 1:10 * pi
m <- matrix(data, ncol = 20, byrow = TRUE)
m <- Matrix::Matrix(m, sparse = TRUE)
hs <- as.h2o(m)
stopifnot(is.h2o(hs), dim(hs)==dim(m))
}
\end{verbatim}

as.matrix.H2OFrame

\textit{Convert an H2OFrame to a matrix}

Description

Convert an H2OFrame to a matrix

Usage

\begin{verbatim}

## S3 method for class 'H2OFrame'
as.matrix(x, ...)
\end{verbatim}
as.numeric

Arguments

x An H2OFrame object

... Further arguments to be passed down from other methods.

Examples

```r
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris <- h2o.uploadFile(path = irisPath)
iris.hex <- as.h2o(iris)
describe <- h2o.describe(iris.hex)
mins = as.matrix(apply(iris.hex, 2, min))
print(mins)
```

as.numeric Convert H2O Data to Numeric

Description

Converts an H2O column into a numeric value column.

Usage

as.numeric(x)

Arguments

x a column from an H2OFrame data set.

... Further arguments to be passed from or to other methods.

Examples

```r
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
prostate.hex[,2] <- as.factor(prostate.hex[,2])
prostate.hex[,2] <- as.numeric(prostate.hex[,2])
```
as.vector.H2OFrame  

Convert an H2OFrame to a vector

Description

Convert an H2OFrame to a vector

Usage

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

Arguments

- `x`: An H2OFrame object
- `mode`: Mode to coerce vector to

Examples

```r
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package="h2o")
iris <- h2o.uploadFile(path = irisPath)
hex <- as.h2o(iris)
cor.R <- cor(as.matrix(iris[,1]))
cor.h2o <- cor(hex[,1])
iris.Rcor <- cor(iris[,1:4])
iris.H2Ocor <- as.data.frame(cor(hex[,1:4]))
h2o_vec <- as.vector(unlist(iris.H2Ocor))
r_vec <- as.vector(unlist(iris.Rcor))
```

australia  

Australia Coastal Data

Description

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

Format

A data frame with 251 rows and 8 columns
### colnames

Returns the column names of an H2OFrame

**Description**

Returns the column names of an H2OFrame

**Usage**

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

**Arguments**

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

**Examples**

```r
h2o.init()
iris.hex <- as.h2o(iris)
colnames(iris) # Returns "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

### dim.H2OFrame

Returns the Dimensions of an H2OFrame

**Description**

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

**Usage**

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

**Arguments**

- `x`: An H2OFrame object.

**See Also**

`dim` for the base R method.
generate_col_ind

**Examples**

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

---

dimnames.H2OFrame  *Column names of an H2OFrame*

---

**Description**

Set column names of an H2O Frame

**Usage**

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

**Arguments**

- `x`  
  An H2OFrame

**Examples**

```r
h2o.init()
n <- 2000
# Generate variables V1, ... V10
X <- matrix(rnorm(10*n), n, 10)
# y = +1 if sum_i x_{ij}^2 > chisq median on 10 df
y <- rep(-1, n)
y[apply(X*X, 1, sum) > qchisq(.95, 10)] <- 1
# Assign names to the columns of X:
dimnames(X)[[2]] <- c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10")
```

---

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

---

**Description**

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

generate_col_ind(data, by)

Arguments

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

---

h2o.abs

*Compute the absolute value of x*

Description

Compute the absolute value of x

Usage

h2o.abs(x)

Arguments

x An H2OFrame object.

See Also

abs for the base R implementation.

Examples

```r
h2o.init()
smtreesH20 <- h2o.importFile(url)
fith2o <- h2o.gbm(x=c("girth", "height"), y="vol", ntrees=3, max_depth=1, distribution="gaussian", min_rows=2, learn_rate=.1, training_frame=smtreesH20)
pred <- as.data.frame(predict(fith2o, newdata=smtreesH20))
diff <- pred-smtreesR[,4]
diff_abs <- abs(diff)
print(diff_abs)
```
**h2o.acos**

*Compute the arc cosine of x*

**Description**

Compute the arc cosine of x

**Usage**

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

**Arguments**

- **x**
  
  An H2OFrame object.

**See Also**

- [acos](#) for the base R implementation.

**Examples**

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

---

**h2o.aggregated_frame**

*Retrieve an aggregated frame from an Aggregator model*

**Description**

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

**Usage**

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

**Arguments**

- **model**
  
  an H2OClusteringModel corresponding from a h2o.aggregator call.
Examples

```r
library(h2o)
h2o.init()
df <- h2o.createFrame(rows=100, cols=5, categorical_fraction=0.6, integer_fraction=0,
                      binary_fraction=0, real_range=100, integer_range=100, missing_fraction=0)
target_num_exemplars=1000
rel_tol_num_exemplars=0.5
encoding="Eigen"
agg <- h2o.aggregator(training_frame=df,
                      target_num_exemplars=target_num_exemplars,
                      rel_tol_num_exemplars=rel_tol_num_exemplars,
                      categorical_encoding=encoding)
# Use the aggregated frame to create a new dataframe
new_df <- h2o.aggregated_frame(agg)
```

---

**h2o.aggregator**

**Build an Aggregated Frame**

**Description**

Builds an Aggregated Frame of an H2OFrame.

**Usage**

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

**Arguments**

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

save_mapping_frame
   Logical. Whether to export the mapping of the aggregated frame Defaults to FALSE.

num_iteration_without_new_exemplar
   The number of iterations to run before aggregator exits if the number of exemplars collected didn’t change Defaults to 500.

Examples

library(h2o)
h2o.init()
df <- h2o.createFrame(rows=100, cols=5, categorical_fraction=0.6, integer_fraction=0,
binary_fraction=0, real_range=100, integer_range=100, missing_fraction=0)
target_num_exemplars=1000
rel_tol_num_exemplars=0.5
encoding="Eigen"
agg <- h2o.aggregator(training_frame=df,
target_num_exemplars=target_num_exemplars,
rel_tol_num_exemplars=rel_tol_num_exemplars,
categorical_encoding=encoding)

---

h2o.aic

Retrieve the Akaike information criterion (AIC) value

Description

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

Usage

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

Arguments

object An H2OModel or H2OModelMetrics.

train Retrieve the training AIC

valid Retrieve the validation AIC

xval Retrieve the cross-validation AIC
Examples

h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
p.sid <- h2o.runif(prostate.hex)
prostate.train <- h2o.assign(prostate.hex[p.sid > .2,], "prostate.train")
prostate.glm <- h2o.glm(x=3:7, y=2, training_frame=prostate.train)
aic.basic <- h2o.aic(prostate.glm)
print(aic.basic)

h2o.all

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

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

Usage

h2o.all(x)

Arguments

x An H2OFrame object.

See Also

all for the base R implementation.

h2o.anomaly

Anomaly Detection via H2O Deep Learning Model

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

Usage

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

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

Value

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

See Also

h2o.deeplearning for making an H2OAutoEncoderModel.

Examples

library(h2o)
h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(path = prosPath)
prostate.dl = h2o.deeplearning(x = 3:9, training_frame = prostate.hex, autoencoder = TRUE,
                               hidden = c(10, 10), epochs = 5)
prostate.anon = h2o.anomaly(prostate.dl, prostate.hex)
head(prostate.anon)
prostate.anon.per.feature = h2o.anomaly(prostate.dl, prostate.hex, per_feature=TRUE)
head(prostate.anon.per.feature)

h2o.any

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

Description

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

Usage

h2o.any(x)

Arguments

x An H2OFrame object.

See Also

all for the base R implementation.
**h2o.anyFactor**

*Check H2OFrame columns for factors*

**Description**

Determines if any column of an H2OFrame object contains categorical data.

**Usage**

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

**Arguments**

- `x` An H2OFrame object.

**Value**

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

**Examples**

```r
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.importFile(path = irisPath)
h2o.anyFactor(iris.hex)
```

---

**h2o.arrange**

*Sorts an H2O frame by columns*

**Description**

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

**Usage**

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

**Arguments**

- `x` The H2OFrame input to be sorted.
- `...` The column names to sort by.
h2o.asfactor  

Convert H2O Data to Characters

Description

Convert H2O Data to Characters

Usage

h2o.asfactor(x)

Arguments

  x     An H2OFrame object.

See Also

  as.character for the base R implementation.

---

h2o.asfactor  

Convert H2O Data to Factors

Description

Convert H2O Data to Factors

Usage

h2o.asfactor(x)

Arguments

  x     An H2OFrame object.

See Also

  as.factor for the base R implementation.
h2o.asnumeric

Convert H2O Data to Numerics

Description

Convert H2O Data to Numerics

Usage

h2o.asnumeric(x)

Arguments

x  An H2OFrame object.

See Also

as.numeric for the base R implementation.

h2o.assign

Rename an H2O object.

Description

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

Usage

h2o.assign(data, key)

Arguments

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


### h2o.as_date

**Convert between character representations and objects of Date class**

**Description**

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

**Usage**

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

**Arguments**

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

### h2o.auc

**Retrieve the AUC**

**Description**

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

**Usage**

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

**Arguments**

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

**See Also**

- `h2o.giniCoef` for the Gini coefficient, `h2o.mse` for MSE, and `h2o.metric` for the various threshold metrics. See `h2o.performance` for creating H2OModelMetrics objects.
Examples

library(h2o)
h2o.init()

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

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

Description

The Automatic Machine Learning (AutoML) function automates the supervised machine learning model training process. The current version of AutoML trains and cross-validates a Random Forest, an Extremely-Randomized Forest, a random grid of Gradient Boosting Machines (GBMs), a random grid of Deep Neural Nets, and then trains a Stacked Ensemble using all of the models.

Usage

h2o.automl(x, y, training_frame, validation_frame = NULL, leaderboard_frame = NULL, nfolds = 5, fold_column = NULL, weights_column = NULL, balance_classes = FALSE, class_sampling_factors = NULL, max_after_balance_size = 5, max_runtime_secs = 3600, max_models = NULL, stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error"), stopping_tolerance = NULL, stopping_rounds = 3, seed = NULL, project_name = NULL, exclude_algos = NULL, keep_cross_validation_predictions = TRUE, keep_cross_validation_models = TRUE, sort_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "mean_per_class_error"))

Arguments

x A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

y The name or index of the response variable in the model. For classification, the y column must be a factor, otherwise regression will be performed. Indexes are 1-based in R.

training_frame Training frame (H2OFrame or ID).
**validation_frame**
Validation frame (H2OFrame or ID); Optional. This frame is used for early stopping of individual models and early stopping of the grid searches (unless max_models or max_runtimes_secs overrides metric-based early stopping).

**leaderboard_frame**
Leaderboard frame (H2OFrame or ID); Optional. If provided, the Leaderboard will be scored using this data frame instead of using cross-validation metrics, which is the default.

**nfolds**
Number of folds for k-fold cross-validation. Defaults to 5. Use 0 to disable cross-validation; this will also disable Stacked Ensemble (thus decreasing the overall model performance).

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

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

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

**class_sampling_factors**
Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

**max_after_balance_size**
Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

**max_runtime_secs**
Maximum allowed runtime in seconds for the entire model training process. Use 0 to disable. Defaults to 3600 secs (1 hour).

**max_models**
Maximum number of models to build in the AutoML process (does not include Stacked Ensembles). Defaults to NULL.

**stopping_metric**
Metric to use for early stopping (AUTO is logloss for classification, deviance for regression). Must be one of "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.

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

**stopping_rounds**
Integer. Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k
h2o.automl

(stopping_rounds) scoring events. Defaults to 3 and must be an non-zero integer. Use 0 to disable early stopping.

seed

Integer. Set a seed for reproducibility. AutoML can only guarantee reproducibility if max_models or early stopping is used because max_runtime_secs is resource limited, meaning that if the resources are not the same between runs, AutoML may be able to train more models on one run vs another.

project_name

Character string to identify an AutoML project. Defaults to NULL, which means a project name will be auto-generated based on the training frame ID.

exclude_algos

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

keep_cross_validation_predictions

Logical. Whether to keep the predictions of the cross-validation predictions. If set to FALSE then running the same AutoML object for repeated runs will cause an exception as CV predictions are are required to build additional Stacked Ensemble models in AutoML. Defaults to TRUE.

keep_cross_validation_models

Logical. Whether to keep the cross-validated models. Deleting cross-validation models will save memory in the H2O cluster. Defaults to TRUE.

sort_metric

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

Details

AutoML finds the best model, given a training frame and response, and returns an H2OAutoML object, which contains a leaderboard of all the models that were trained in the process, ranked by a default model performance metric.

Value

An H2OAutoML object.

Examples

library(h2o)
h2o.init()
votes_path <- system.file("extdata", "housevotes.csv", package = "h2o")
votes_hf <- h2o.uploadFile(path = votes_path, header = TRUE)
aml <- h2o.automl(y = "Class", training_frame = votes_hf, max_runtime_secs = 30)
**h2o.betweenss**  
*Get the between cluster sum of squares*

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

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

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

**h2o.biases**  
*Return the respective bias vector*

**Description**
Return the respective bias vector

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

**Arguments**
- **object**  
  An H2OModel or H2OModelMetrics
- **vector_id**  
  An integer, ranging from 1 to number of layers + 1, that specifies the bias vector to return.
Description
bottomN function will will grab the bottom N percent of values of a column and return it in a H2OFrame. Extract the top N percent of values of a column and return it in a H2OFrame.

Usage
h2o.bottomN(x, column, nPercent)

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

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

h2o.cbind Combine H2O Datasets by Columns

Description
Takes a sequence of H2O data sets and combines them by column

Usage
h2o.cbind(...)

Arguments
... A sequence of H2OFrame arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number of rows.

Value
An H2OFrame object containing the combined ...arguments column-wise.

See Also
cbind for the base R method.
Examples

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

---

**h2o.ceiling**

*Take a single numeric argument and return a numeric vector with the smallest integers*

**Description**

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

**Usage**

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

**Arguments**

- `x`: An H2OFrame object.

**See Also**

- `ceiling` for the base R implementation.

---

**h2o.centers**

*Retrieve the Model Centers*

**Description**

Retrieve the Model Centers

**Usage**

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

**Arguments**

- `object`: An H2OClusteringModel object.
h2o.centersSTD

Retrieve the Model Centers STD

Description

Retrieve the Model Centers STD

Usage

h2o.centersSTD(object)

Arguments

object An H2OClusteringModel object.

h2o.centroid_stats

Retrieve centroid statistics

Description

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

Usage

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

Arguments

object An H2OClusteringModel object.
train Retrieve the training centroid statistics
valid Retrieve the validation centroid statistics
xval Retrieve the cross-validation centroid statistics
h2o.clearLog

Delete All H2O R Logs

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

Usage
h2o.clearLog()

See Also
h2o.startLogging, h2o.stopLogging, h2o.openLog

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

h2o.clusterInfo
Print H2O cluster info

Description
Print H2O cluster info

Usage
h2o.clusterInfo()
**h2o.clusterIsUp**  

*Determine if an H2O cluster is up or not*

**Description**

Determine if an H2O cluster is up or not

**Usage**

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

**Arguments**

- **conn**  
  H2OConnection object

**Value**

TRUE if the cluster is up; FALSE otherwise

---

**h2o.clusterStatus**  

*Return the status of the cluster*

**Description**

Retrieve information on the status of the cluster running H2O.

**Usage**

```
h2o.clusterStatus()
```

**See Also**

- `H2OConnection, h2o.init`

**Examples**

```
h2o.init()
h2o.clusterStatus()
```
### h2o.clustersizes

**Retrieve the cluster sizes**

Description

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

Usage

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

Arguments

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

### h2o.coef

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

Description

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

Usage

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

Arguments

- **object**: an H2OModel object.
**h2o.coef_norm**

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

**Description**

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

**Usage**

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

**Arguments**

- `object` an H2OModel object.

**h2o.colnames**

Return column names of an H2OFrame

**Description**

Return column names of an H2OFrame

**Usage**

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

**Arguments**

- `x` An H2OFrame object.

**See Also**

`colnames` for the base R implementation.
h2o.columns_by_type  
*Obtain a list of columns that are specified by `coltype`*

**Description**

Obtain a list of columns that are specified by `coltype`.

**Usage**

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

**Arguments**

- `object`  
  H2OFrame object

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

... Ignored

**Value**

A list of column indices that correspond to "type"

**Examples**

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

h2o.computeGram  
*Compute weighted gram matrix.*

**Description**

Compute weighted gram matrix.

**Usage**

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

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

**h2o.confusionMatrix**

Access H2O Confusion Matrices

Description

Retrieve either a single or many confusion matrices from H2O objects.

Usage

```
h2o.confusionMatrix(object, ...) # S4 method for signature 'H2OModel'
h2o.confusionMatrix(object, newdata, valid = FALSE, ...) # S4 method for signature 'H2OModelMetrics'

h2o.confusionMatrix(object, thresholds = NULL, metrics = NULL)
```

Arguments

- **object**: Either an `H2OModel` object or an `H2OModelMetrics` object.
- **newdata**: An H2OFrame object that can be scored on. Requires a valid response column.
- **valid**: Retrieve the validation metric.
- **thresholds**: (Optional) A value or a list of valid values between 0.0 and 1.0. This value is only used in the case of `H2OBinomialMetrics` objects.
- **metrics**: (Optional) A metric or a list of valid metrics ("min_per_class_accuracy", "absolute_mcc", "tnr", "fmr", "fpr", "tpr", "precision", "accuracy", "f0point5", "f2", "f1"). This value is only used in the case of `H2OBinomialMetrics` objects.

Details

The `H2OModelMetrics` version of this function will only take `H2OBinomialMetrics` or `H2OMultinomialMetrics` objects. If no threshold is specified, all possible thresholds are selected.
Value

Calling this function on H2OModel objects returns a confusion matrix corresponding to the predict function. If used on an H2OBinomialMetrics object, returns a list of matrices corresponding to the number of thresholds specified.

See Also

predict for generating prediction frames, h2o.performance for creating H2OModelMetrics.

Examples

library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)
hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
h2o.confusionMatrix(model, hex)
# Generating a ModelMetrics object
perf <- h2o.performance(model, hex)
h2o.confusionMatrix(perf)

h2o.connect

Connect to a running H2O instance.

Description

Connect to a running H2O instance.

Usage

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

Arguments

ip Object of class character representing the IP address of the server where H2O is running.
port Object of class numeric representing the port number of the H2O server.
strict_version_check (Optional) Setting this to FALSE is unsupported and should only be done when advised by technical support.
proxy (Optional) A character string specifying the proxy path.
h2o.cor

- `https` (Optional) Set this to TRUE to use https instead of http.
- `insecure` (Optional) Set this to TRUE to disable SSL certificate checking.
- `username` (Optional) Username to login with.
- `password` (Optional) Password to login with.
- `cookies` (Optional) Vector(or list) of cookies to add to request.
- `context_path` (Optional) The last part of connection URL: http://<ip>:<port>/<context_path>
- `config` (Optional) A list describing connection parameters.

Value

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

Examples

```r
## Not run:
library(h2o)

# Try to connect to a H2O instance running at http://localhost:54321/cluster_X
# If not found, start a local H2O instance from R with the default settings.
# h2o.connect(ip = "localhost", port = 54321, context_path = "cluster_X")
# Or
# config = list(ip = "localhost", port = 54321, context_path = "cluster_X")
# h2o.connect(config = config)

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

## End(Not run)
```

h2o.cor  Correlation of columns.

Description

Compute the correlation matrix of one or two H2OFrames.

Usage

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

cor(x, ...)
```
Arguments

x  An H2OFrame object.
y  NULL (default) or an H2OFrame. The default is equivalent to y = x.
na.rm logical. Should missing values be removed?
use  An optional character string indicating how to handle missing values. This must
be one of the following: "everything" - outputs NaNs whenever one of its contrib-
uting observations is missing "all.obs" - presence of missing observations
will throw an error "complete.obs" - discards missing values along with all ob-
servations in their rows so that only complete observations are used

...  Further arguments to be passed down from other methods.

Examples

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

h2o.cos  Compute the cosine of x

Description

Compute the cosine of x

Usage

h2o.cos(x)

Arguments

x  An H2OFrame object.

See Also

cos for the base R implementation.
h2o.cosh

Compute the hyperbolic cosine of x

Description
Compute the hyperbolic cosine of x

Usage
h2o.cosh(x)

Arguments
x An H2OFrame object.

See Also
cosh for the base R implementation.

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

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

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

Arguments
x (Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except event_column, start_column and stop_column are used.

event_column The name of binary data column in the training frame indicating the occurrence of an event.

training_frame Id of the training data frame.

model_id Destination id for this model; auto-generated if not specified.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_column</td>
<td>Start Time Column.</td>
</tr>
<tr>
<td>stop_column</td>
<td>Stop Time Column.</td>
</tr>
<tr>
<td>weights_column</td>
<td>Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.</td>
</tr>
<tr>
<td>offset_column</td>
<td>Offset column. This will be added to the combination of columns before applying the link function.</td>
</tr>
<tr>
<td>stratify_by</td>
<td>List of columns to use for stratification.</td>
</tr>
<tr>
<td>ties</td>
<td>Method for Handling Ties. Must be one of: &quot;efron&quot;, &quot;breslow&quot;. Defaults to efron.</td>
</tr>
<tr>
<td>init</td>
<td>Coefficient starting value. Defaults to 0.</td>
</tr>
<tr>
<td>lre_min</td>
<td>Minimum log-relative error. Defaults to 9.</td>
</tr>
<tr>
<td>max_iterations</td>
<td>Maximum number of iterations. Defaults to 20.</td>
</tr>
<tr>
<td>interactions</td>
<td>A list of predictor column indices to interact. All pairwise combinations will be computed for the list.</td>
</tr>
<tr>
<td>interaction_pairs</td>
<td>A list of pairwise (first order) column interactions.</td>
</tr>
<tr>
<td>interactions_only</td>
<td>A list of columns that should only be used to create interactions but should not itself participate in model training.</td>
</tr>
<tr>
<td>use_all_factor_levels</td>
<td>Logical. (Internal. For development only!) Indicates whether to use all factor levels. Defaults to FALSE.</td>
</tr>
</tbody>
</table>

**h2o.createFrame**

*Data H2OFrame Creation in H2O*

**Description**

Creates a data frame in H2O with real-valued, categorical, integer, and binary columns specified by the user.

**Usage**

```r
h2o.createFrame(rows = 10000, cols = 10, randomize = TRUE, value = 0, real_range = 100, categorical_fraction = 0.2, factors = 100, integer_fraction = 0.2, integer_range = 100, binary_fraction = 0.1, binary_ones_fraction = 0.02, time_fraction = 0, string_fraction = 0, missing_fraction = 0.01, response_factors = 2, has_response = FALSE, seed, seed_for_column_types)
```
Arguments

rows: The number of rows of data to generate.
cols: The number of columns of data to generate. Excludes the response column if has_response = TRUE.
randomize: A logical value indicating whether data values should be randomly generated. This must be TRUE if either categorical_fraction or integer_fraction is non-zero.
value: If randomize = FALSE, then all real-valued entries will be set to this value.
real_range: The range of randomly generated real values.
categorical_fraction: The fraction of total columns that are categorical.
factors: The number of (unique) factor levels in each categorical column.
integer_fraction: The fraction of total columns that are integer-valued.
integer_range: The range of randomly generated integer values.
binary_fraction: The fraction of total columns that are binary-valued.
binary_ones_fraction: The fraction of values in a binary column that are set to 1.
time_fraction: The fraction of randomly created date/time columns.
string_fraction: The fraction of randomly created string columns.
missing_fraction: The fraction of total entries in the data frame that are set to NA.
response_factors: If has_response = TRUE, then this is the number of factor levels in the response column.
has_response: A logical value indicating whether an additional response column should be prepended to the final H2O data frame. If set to TRUE, the total number of columns will be cols+1.
seed: A seed used to generate random values when randomize = TRUE.
seed_for_column_types: A seed used to generate random column types when randomize = TRUE.

Value

Returns an H2OFrame object.

Examples

library(h2o)
h2o.init()
hex <- h2o.createFrame(rows = 1000, cols = 100, categorical_fraction = 0.1,
h2o.cross_validation_holdout_predictions

Description
Retrieve the cross-validation holdout predictions

Usage
h2o.cross_validation_holdout_predictions(object)

Arguments
object An H2OModel object.

Value
Returns a H2OFrame

h2o.cross_validation_fold_assignment
Retrieve the cross-validation fold assignment

Description
Retrieve the cross-validation fold assignment

Usage
h2o.cross_validation_fold_assignment(object)

Arguments
object An H2OModel object.

Value
Returns a H2OFrame
h2o.cross_validation_models

Retrieve the cross-validation models

Description
Retrieve the cross-validation models

Usage
h2o.cross_validation_models(object)

Arguments
object An H2OModel object.

Value
Returns a list of H2OModel objects

h2o.cross_validation_predictions

Retrieve the cross-validation predictions

Description
Retrieve the cross-validation predictions

Usage
h2o.cross_validation_predictions(object)

Arguments
object An H2OModel object.

Value
Returns a list of H2OFrame objects
h2o.cummax  

Return the cumulative max over a column or across a row

Description

Return the cumulative max over a column or across a row

Usage

h2o.cummax(x, axis = 0)

Arguments

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

See Also

cummax for the base R implementation.

h2o.cummin  

Return the cumulative min over a column or across a row

Description

Return the cumulative min over a column or across a row

Usage

h2o.cummin(x, axis = 0)

Arguments

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

See Also

cummin for the base R implementation.
h2o.cumprod

Return the cumulative product over a column or across a row

Description

Return the cumulative product over a column or across a row

Usage

h2o.cumprod(x, axis = 0)

Arguments

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

See Also

cumprod for the base R implementation.

h2o.cumsum

Return the cumulative sum over a column or across a row

Description

Return the cumulative sum over a column or across a row

Usage

h2o.cumsum(x, axis = 0)

Arguments

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

See Also

cumsum for the base R implementation.
**h2o.cut**  
*Cut H2O Numeric Data to Factor*

**Description**

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

**Usage**

```
# S3 method for class 'H2OFrame'

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

```
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

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

*Convert Milliseconds to Day of Month in H2O Datasets*

**Description**

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

**Usage**

```r
h2o.day(x)
day(x)
```

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

**Arguments**

- `x` An H2OFrame object.

**Value**

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

**See Also**

- `h2o.month`

---

**h2o.dayOfWeek**  

*Convert Milliseconds to Day of Week in H2O Datasets*

**Description**

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

h2o.dayOfWeek(x)

dayOfWeek(x)

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

Arguments

x An H2OFrame object.

Value

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

See Also

h2o.day, h2o.month

h2o.dct

Compute DCT of an H2OFrame

Description

Compute the Discrete Cosine Transform of every row in the H2OFrame

Usage

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

Arguments

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

inverse Whether to perform the inverse transform

Value

Returns an H2OFrame object.
**Examples**

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

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

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

---

**h2o.ddply**

*Split H2O Dataset, Apply Function, and Return Results*

**Description**

For each subset of an H2O data set, apply a user-specified function, then combine the results. This is an experimental feature.

**Usage**

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

**Arguments**

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

**Value**

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

**See Also**

`ddply` for the plyr library implementation.
h2o.decryptionSetup

**Examples**

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

# Import iris dataset to H2O
irisPath <- system.file("extdata", "iris_wheader.csv", package = "h2o")
iris.hex <- h2o.uploadFile(path = irisPath, destination_frame = "iris.hex")
# Add function taking mean of sepal_len column
fun <- function(df) { sum(df[,1], na.rm = TRUE)/nrow(df) }
# Apply function to groups by class of flower
# uses h2o's ddply, since iris.hex is an H2OFrame object
res <- h2o.ddply(iris.hex, "class", fun)
head(res)
```

---

**h2o.decryptionSetup**  
*Setup a Decryption Tool*

**Description**

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

**Usage**

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

**Arguments**

- **keystore**  
  An H2OFrame object referencing a loaded Java Keystore (see example).

- **keystore_type**  
  (Optional) Specification of Keystore type, defaults to JCEKS.

- **key_alias**  
  Which key from the keystore to use for decryption.

- **password**  
  Password to the keystore and the key.

- **decrypt_tool**  
  (Optional) Name of the decryption tool.

- **decrypt_impl**  
  (Optional) Java class name implementing the Decryption Tool.

- **cipher_spec**  
  Specification of a cipher (eg.: AES/ECB/PKCS5Padding).

**See Also**

- `h2o.importFile`, `h2o.parseSetup`
Examples

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

## End(Not run)

---

### h2o.deepfeatures

**Feature Generation via H2O Deep Learning or DeepWater Model**

**Description**

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

**Usage**

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

**Arguments**

- **object**: An `H2OModel` object that represents the deep learning model to be used for feature extraction.
- **data**: An `H2OFrame` object.
- **layer**: Index (for DeepLearning, integer) or Name (for DeepWater, String) of the hidden layer to extract

**Value**

Returns an `H2OFrame` object with as many features as the number of units in the hidden layer of the specified index.

**See Also**

- `h2o.deeplearning` for making H2O Deep Learning models.
- `h2o.deepwater` for making H2O DeepWater models.
Examples

library(h2o)

h2o.init()
prosPath = system.file("extdata", "prostate.csv", package = "h2o")
prostate.hex = h2o.importFile(path = prosPath)

prostate.dl = h2o.deeplearning(x = 3:9, y = 2, training_frame = prostate.hex,
hidden = c(100, 200), epochs = 5)

prostate.deepfeatures_layer1 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 1)
prostate.deepfeatures_layer2 = h2o.deepfeatures(prostate.dl, prostate.hex, layer = 2)

head(prostate.deepfeatures_layer1)
head(prostate.deepfeatures_layer2)

# if (h2o.deepwater.available()) {
  # prostate.dl = h2o.deepwater(x = 3:9, y = 2, backend="mxnet", training_frame = prostate.hex,
  #   hidden = c(100, 200), epochs = 5)
  # prostate.deepfeatures_layer1 =
  #   h2o.deepfeatures(prostate.dl, prostate.hex, layer = "fc1_w")
  # prostate.deepfeatures_layer2 =
  #   h2o.deepfeatures(prostate.dl, prostate.hex, layer = "fc2_w")
  # head(prostate.deepfeatures_layer1)
  # head(prostate.deepfeatures_layer2)
  #}

---

h2o.deeplearning  Build a Deep Neural Network model using CPUs

Description

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

Usage

h2o.deeplearning(x, y, training_frame, model_id = NULL,
   validation_frame = NULL, nfolds = 0,
   keep_cross_validation_predictions = FALSE,
   keep_cross_validation_fold_assignment = FALSE, fold_assignment = c("AUTO",
   "Random", "Modulo", "Stratified"), fold_column = NULL,
   ignore_const_cols = TRUE, score_each_iteration = FALSE,
   weights_column = NULL, offset_column = NULL, balance_classes = FALSE,
   class_sampling_factors = NULL, max_after_balance_size = 5,
   max_hit_ratio_k = 0, checkpoint = NULL, pretrained_autoencoder = NULL,
   overwrite_with_best_model = TRUE, use_all_factor_levels = TRUE,
   standardize = TRUE, activation = c("Tanh", "TanhWithDropout", "Rectifier",
   "RectifierWithDropout", "Maxout", "MaxoutWithDropout"), hidden = c(200,
   200), epochs = 10, train_samples_per_iteration = -2,
   target_ratio_comm_to_comp = 0.05, seed = -1, adaptive_rate = TRUE,
Arguments

**x**
(Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

**y**
The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

**training_frame**
Id of the training data frame.

**model_id**
Destination id for this model; auto-generated if not specified.

**validation_frame**
Id of the validation data frame.

**nfolds**
Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

**keep_cross_validation_predictions**
Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment
    Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

ignore_const_cols
    Logical. Ignore constant columns. Defaults to TRUE.

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

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

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

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

class_sampling_factors
    Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

max_after_balance_size
    Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

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

checkpoint
    Model checkpoint to resume training with.

pretrained_autoencoder
    Pretrained autoencoder model to initialize this model with.

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

use_all_factor_levels
    Logical. Use all factor levels of categorical variables. Otherwise, the first factor level is omitted (without loss of accuracy). Useful for variable importances and auto-enabled for autoencoder. Defaults to TRUE.
standardize Logical. If enabled, automatically standardize the data. If disabled, the user must provide properly scaled input data. Defaults to TRUE.


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

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

train_samples_per_iteration Number of training samples (globally) per MapReduce iteration. Special values are 0: one epoch, -1: all available data (e.g., replicated training data), -2: automatic. Defaults to -2.

target_ratio_comm_to_comp Target ratio of communication overhead to computation. Only for multi-node operation and train_samples_per_iteration = -2 (auto-tuning). Defaults to 0.05.

seed Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Note: only reproducible when running single threaded. Defaults to -1 (time-based random number).

adaptive_rate Logical. Adaptive learning rate. Defaults to TRUE.

rho Adaptive learning rate time decay factor (similarity to prior updates). Defaults to 0.99.

epsilon Adaptive learning rate smoothing factor (to avoid divisions by zero and allow progress). Defaults to 1e-08.

rate Learning rate (higher => less stable, lower => slower convergence). Defaults to 0.005.

rate_annealing Learning rate annealing: rate / (1 + rate_annealing * samples). Defaults to 1e-06.

rate_decay Learning rate decay factor between layers (N-th layer: rate * rate_decay ^ (n - 1)). Defaults to 1.

momentum_start Initial momentum at the beginning of training (try 0.5). Defaults to 0.

momentum_ramp Number of training samples for which momentum increases. Defaults to 1000000.

momentum_stable Final momentum after the ramp is over (try 0.99). Defaults to 0.

nesterov_accelerated_gradient Logical. Use Nesterov accelerated gradient (recommended). Defaults to TRUE.

input_dropout_ratio Input layer dropout ratio (can improve generalization, try 0.1 or 0.2). Defaults to 0.

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

L1 regularization (can add stability and improve generalization, causes many weights to become 0). Defaults to 0.
L2 regularization (can add stability and improve generalization, causes many weights to be small. Defaults to 0.

max_w2 Constraint for squared sum of incoming weights per unit (e.g. for Rectifier). Defaults to $3.4028235e+38$.


initial_weight_scale Uniform: -value...value, Normal: stddev. Defaults to 1.

initial_weights A list of H2OFrame ids to initialize the weight matrices of this model with.

initial_biases A list of H2OFrame ids to initialize the bias vectors of this model with.


distribution Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.

quantile_alpha Desired quantile for Quantile regression, must be between 0 and 1. Defaults to 0.5.

tweedie_power Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to 1.5.

huber_alpha Desired quantile for Huber/M-regression (threshold between quadratic and linear loss, must be between 0 and 1). Defaults to 0.9.

score_interval Shortest time interval (in seconds) between model scoring. Defaults to 5.

score_training_samples Number of training set samples for scoring (0 for all). Defaults to 10000.

score_validation_samples Number of validation set samples for scoring (0 for all). Defaults to 0.

score_duty_cycle Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring). Defaults to 0.1.

classification_stop Stopping criterion for classification error fraction on training data (-1 to disable). Defaults to 0.

regression_stop Stopping criterion for regression error (MSE) on training data (-1 to disable). Defaults to 1e-06.

stopping_rounds Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable) Defaults to 5.

stopping_metric Metric to use for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.
stopping_tolerance
    Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.

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

score_validation_sampling
    Method used to sample validation dataset for scoring. Must be one of: "Uniform", "Stratified". Defaults to Uniform.

diagnostics
    Logical. Enable diagnostics for hidden layers. Defaults to TRUE.

fast_mode
    Logical. Enable fast mode (minor approximation in back-propagation). Defaults to TRUE.

force_load_balance
    Logical. Force extra load balancing to increase training speed for small datasets (to keep all cores busy). Defaults to TRUE.

variable_importances
    Logical. Compute variable importances for input features (Gedeon method) - can be slow for large networks. Defaults to TRUE.

replicate_training_data
    Logical. Replicate the entire training dataset onto every node for faster training on small datasets. Defaults to TRUE.

single_node_mode
    Logical. Run on a single node for fine-tuning of model parameters. Defaults to FALSE.

shuffle_training_data
    Logical. Enable shuffling of training data (recommended if training data is replicated and train_samples_per_iteration is close to #nodes x #rows, or if using balance_classes). Defaults to FALSE.

missing_values_handling
    Handling of missing values. Either MeanImputation or Skip. Must be one of: "MeanImputation", "Skip". Defaults to MeanImputation.

quiet_mode
    Logical. Enable quiet mode for less output to standard output. Defaults to FALSE.

autoencoder
    Logical. Auto-Encoder. Defaults to FALSE.

sparse
    Logical. Sparse data handling (more efficient for data with lots of 0 values). Defaults to FALSE.

col_major
    Logical. #DEPRECATED Use a column major weight matrix for input layer. Can speed up forward propagation, but might slow down backpropagation. Defaults to TRUE.

average_activation
    Average activation for sparse auto-encoder. #Experimental Defaults to 0.

sparsity_beta
    Sparsity regularization. #Experimental Defaults to 0.

max_categorical_features
    Max. number of categorical features, enforced via hashing. #Experimental Defaults to 2147483647.
reproducible Logical. Force reproducibility on small data (will be slow - only uses 1 thread).
Defaults to FALSE.

export_weights_and_biases Logical. Whether to export Neural Network weights and biases to H2O Frames.
Defaults to FALSE.

mini_batch_size Mini-batch size (smaller leads to better fit, larger can speed up and generalize better). Defaults to 1.

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

elastic_averaging Logical. Elastic averaging between compute nodes can improve distributed model convergence. #Experimental Defaults to FALSE.

elastic_averaging_moving_rate Elastic averaging moving rate (only if elastic averaging is enabled). Defaults to 0.9.

elastic_averaging_regularization Elastic averaging regularization strength (only if elastic averaging is enabled). Defaults to 0.001.

verbose Logical. Print scoring history to the console (Metrics per tree for GBM, DRF, & XGBoost. Metrics per epoch for Deep Learning). Defaults to FALSE.

See Also

predict.H2OModel for prediction

Examples

```r
library(h2o)
h2o.init()
iris.hex <- as.h2o(iris)
iris.d1 <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex, seed=123456)

# now make a prediction
predictions <- h2o.predict(iris.d1, iris.hex)
```

h2o.deepwater Build a Deep Learning model using multiple native GPU backends

Description

Builds a deep neural network on an H2OFrame containing various data sources.
Usage

```r
h2o.deepwater(x, y, training_frame, model_id = NULL, checkpoint = NULL,
  autoencoder = FALSE, validation_frame = NULL, nfolds = 0,
  balance_classes = FALSE, max_after_balance_size = 5,
  class_sampling_factors = NULL, keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE, fold_assignment = c("AUTO",
    "Random", "Modulo", "Stratified"), fold_column = NULL,
  offset_column = NULL, weights_column = NULL,
  score_each_iteration = FALSE, categorical_encoding = c("AUTO", "Enum",
    "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder",
    "SortByResponse", "EnumLimited"), overwrite_with_best_model = TRUE,
  epochs = 10, train_samples_per_iteration = -2,
  target_ratio_comm_to_comp = 0.05, seed = -1, standardize = TRUE,
  learning_rate = 0.001, learning_rate_annealing = 1e-06,
  momentum_start = 0.9, momentum_ramp = 10000, momentum_stable = 0.9,
  distribution = c("AUTO", "bernoulli", "multinomial", "gaussian",
    "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"), score_interval = 5,
  score_training_samples = 10000, score_validation_samples = 0,
  score_duty_cycle = 0.1, classification_stop = 0, regression_stop = 0,
  stopping_rounds = 5, stopping_metric = c("AUTO", "deviance", "logloss",
    "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification",
    "mean_per_class_error"), stopping_tolerance = 0, max_runtime_secs = 0,
  ignore_const_cols = TRUE, shuffle_training_data = TRUE,
  mini_batch_size = 32, clip_gradient = 10, network = c("auto", "user",
    "lenet", "alexnet", "vgg", "googlenet", "inception_bn", "resnet"),
  backend = c("mxnet", "caffe", "tensorflow"), image_shape = c(0, 0),
  channels = 3, sparse = FALSE, gpu = TRUE, device_id = c(0),
  cache_data = TRUE, network_definition_file = NULL,
  network_parameters_file = NULL, mean_image_file = NULL,
  export_native_parameters_prefix = NULL, activation = c("Rectifier",
    "Tanh"), hidden = NULL, input_dropout_ratio = 0,
  hidden_dropout_ratios = NULL, problem_type = c("auto", "image",
    "dataset")
```

Arguments

- **x** *(Optional)* A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

- **y** The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

- **training_frame** Id of the training data frame.

- **model_id** Destination id for this model; auto-generated if not specified.

- **checkpoint** Model checkpoint to resume training with.

- **autoencoder** Logical. Auto-Encoder. Defaults to FALSE.
validation_frame
   Id of the validation data frame.

nfolds
   Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

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

max_after_balance_size
   Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

class_sampling_factors
   Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

keep_cross_validation_predictions
   Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment
   Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

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

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

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

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

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

epochs
   How many times the dataset should be iterated (streamed), can be fractional. Defaults to 10.
train_samples_per_iteration
Number of training samples (globally) per MapReduce iteration. Special values are 0: one epoch, -1: all available data (e.g., replicated training data), -2: automatic. Defaults to -2.

target_ratio_comm_to_comp
Target ratio of communication overhead to computation. Only for multi-node operation and train_samples_per_iteration = -2 (auto-tuning). Defaults to 0.05.

seed
Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Note: only reproducible when running single threaded. Defaults to -1 (time-based random number).

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

learning_rate
Learning rate (higher => less stable, lower => slower convergence). Defaults to 0.001.

learning_rate_annealing
Learning rate annealing: rate / (1 + rate_annealing * samples). Defaults to 1e-06.

momentum_start
Initial momentum at the beginning of training (try 0.5). Defaults to 0.9.

momentum_ramp
Number of training samples for which momentum increases. Defaults to 10000.

momentum_stable
Final momentum after the ramp is over (try 0.99). Defaults to 0.9.

distribution
Distribution function Must be one of: "AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.

score_interval
Shortest time interval (in seconds) between model scoring. Defaults to 5.

score_training_samples
Number of training set samples for scoring (0 for all). Defaults to 10000.

score_validation_samples
Number of validation set samples for scoring (0 for all). Defaults to 0.

score_duty_cycle
Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring). Defaults to 0.1.

classification_stop
Stopping criterion for classification error fraction on training data (-1 to disable). Defaults to 0.

regression_stop
Stopping criterion for regression error (MSE) on training data (-1 to disable). Defaults to 0.

stopping_rounds
Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable) Defaults to 5.

stopping_metric
Metric to use for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.
stopping_tolerance
   Relative tolerance for metric-based stopping criterion (stop if relative improve-
   ment is not at least this much) Defaults to 0.
max_runtime_secs
   Maximum allowed runtime in seconds for model training. Use 0 to disable.
   Defaults to 0.
ignore_const_cols
   Logical. Ignore constant columns. Defaults to TRUE.
shuffle_training_data
   Logical. Enable global shuffling of training data. Defaults to TRUE.
mini_batch_size
   Mini-batch size (smaller leads to better fit, larger can speed up and generalize
   better). Defaults to 32.
clip_gradient
   Clip gradients once their absolute value is larger than this value. Defaults to 10.
network
   Network architecture. Must be one of: "auto", "user", "lenet", "alexnet", "vgg",
   "googlelenet", "inception_bn", "resnet". Defaults to auto.
backend
   Deep Learning Backend. Must be one of: "mxnet", "caffe", "tensorflow". De-
  faults to mxnet.
image_shape
   Width and height of image. Defaults to [0, 0].
channels
   Number of (color) channels. Defaults to 3.
sparse
   Logical. Sparse data handling (more efficient for data with lots of 0 values).
   Defaults to FALSE.
gpu
   Logical. Whether to use a GPU (if available). Defaults to TRUE.
device_id
   Device IDs (which GPUs to use). Defaults to [0].
cache_data
   Logical. Whether to cache the data in memory (automatically disabled if data
   size is too large). Defaults to TRUE.
network_definition_file
   Path of file containing network definition (graph, architecture).
network_parameters_file
   Path of file containing network (initial) parameters (weights, biases).
mean_image_file
   Path of file containing the mean image data for data normalization.
export_native_parameters_prefix
   Path (prefix) where to export the native model parameters after every iteration.
activation
   Activation function. Only used if no user-defined network architecture file is
   provided, and only for problem_type=dataset. Must be one of: "Rectifier",
   "Tanh".
hidden
   Hidden layer sizes (e.g. [200, 200]). Only used if no user-defined network
   architecture file is provided, and only for problem_type=dataset.
input_dropout_ratio
   Input layer dropout ratio (can improve generalization, try 0.1 or 0.2). Defaults
   to 0.
**hidden_dropout_ratios**

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

**problem_type**

Problem type, auto-detected by default. If set to image, the H2OFrame must contain a string column containing the path (URI or URL) to the images in the first column. If set to text, the H2OFrame must contain a string column containing the text in the first column. If set to dataset, Deep Water behaves just like any other H2O Model and builds a model on the provided H2OFrame (non-String columns). Must be one of: "auto", "image", "dataset". Defaults to auto.

---

**h2o.deepwater.available**

*Determines whether Deep Water is available*

**Description**

Ask the H2O server whether a Deep Water model can be built. (Depends on availability of native backends.) Returns TRUE if a Deep Water model can be built, or FALSE otherwise.

**Usage**

```
  h2o.deepwater.available(h2oRestApiVersion = .h2o.__REST_API_VERSION)
```

**Arguments**

- **h2oRestApiVersion**
  
  (Optional) Specific version of the REST API to use.

---

**h2o.describe**

*H2O Description of A Dataset*

**Description**

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

**Usage**

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

**Arguments**

- **frame**
  
  An H2OFrame object.
Value

A table with the Frame stats.

Examples

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

---

**h2o.difflag1**

*Conduct a lag 1 transform on a numeric H2OFrame column*

Description

Conduct a lag 1 transform on a numeric H2OFrame column

Usage

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

Arguments

- **object**: H2OFrame object

Value

Returns an H2OFrame object.

---

**h2o.dim**

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

Description

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

Usage

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

Arguments

- **x**: An H2OFrame object.
h2o.dimnames

See Also

dim for the base R implementation.

h2o.dimnames

Column names of an H2OFrame

Description

Column names of an H2OFrame

Usage

h2o.dimnames(x)

Arguments

x An H2OFrame object.

See Also

dimnames for the base R implementation.

h2o.distance

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

Description

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

Usage

h2o.distance(x, y, measure)

Arguments

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

```r
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
h2o.distance(prostate.hex[1:30,], prostate.hex[1:10,], "cosine")
```

h2o.downloadAllLogs  Download H2O Log Files to Disk

Description

h2o.downloadAllLogs downloads all H2O log files to local disk in .zip format. Generally used for debugging purposes.

Usage

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

Arguments

dirname  (Optional) A character string indicating the directory that the log file should be saved in.

filename  (Optional) A character string indicating the name that the log file should be saved to. Note that the saved format is .zip, so the file name must include the .zip extension.

Examples

```r
h2o.downloadAllLogs(dirname='.\your_directory_name\', filename = 'autoh2o_log.zip')
```

h2o.downloadCSV  Download H2O Data to Disk

Description

Download an H2O data set to a CSV file on the local disk

Usage

```r
h2o.downloadCSV(data, filename)
```
### Arguments

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

### Warning

Files located on the H2O server may be very large! Make sure you have enough hard drive space to accommodate the entire file.

### Examples

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

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

---

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

### Description

Download the model in MOJO format.

### Usage

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

### Arguments

- **model**: An H2OModel
- **path**: The path where MOJO file should be saved. Saved to current directory by default.
- **get_genmodel_jar**: If TRUE, then also download h2o-genmodel.jar and store it in either in the same folder
- **genmodel_name**: Custom name of genmodel jar.
- **genmodel_path**: Path to store h2o-genmodel.jar. If left blank and "get_genmodel_jar" is TRUE, then the h2o-genmodel.jar
Value

Name of the MOJO file written to the path.

Examples

```r
library(h2o)
h <- h2o.init()
fr <- as.h2o(iris)
my_model <- h2o.gbm(x=1:4, y=5, training_frame=fr)
h2o.download_mojo(my_model)  # save to the current working directory
```

**Description**

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

**Usage**

```r
h2o.download_pojo(model, path = NULL, getjar = NULL, get_jar = TRUE, jar_name = "")
```

**Arguments**

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

**Value**

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

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

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

---

**h2o.entropy**  
**Shannon entropy**

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

buys <- as.h2o(c("no", "no", "yes", "yes", "no", "yes", "no", "yes", "yes","no"))

buys_entropy <- h2o.entropy(buys)
```
h2o.exp

*Compute the exponential function of x*

**Description**

Compute the exponential function of x

**Usage**

h2o.exp(x)

**Arguments**

- **x**  
  An H2OFrame object.

**See Also**

exp for the base R implementation.

h2o.exportFile

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

**Description**

Exports an H2OFrame (which can be either VA or FV) to a file. This file may be on the H2O instance’s local filesystem, or to HDFS (preface the path with hdfs://) or to S3N (preface the path with s3n://).

**Usage**

h2o.exportFile(data, path, force = FALSE, parts = 1)

**Arguments**

- **data**  
  An H2OFrame object.
- **path**  
  The path to write the file to. Must include the directory and also filename if exporting to a single file. May be prefaced with hdfs:// or s3n://. Each row of data appears as line of the file.
- **force**  
  logical, indicates how to deal with files that already exist.
- **parts**  
  integer, number of part files to export to. Default is to write to a single file. Large data can be exported to multiple 'part' files, where each part file contains subset of the data. User can specify the maximum number of part files or use value -1 to indicate that H2O should itself determine the optimal number of files. Parameter path will be considered to be a path to a directory if export to multiple part files is desired. Part files conform to naming scheme 'part-m-?????'.


Details

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

Examples

```r
## Not run:
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.uploadFile(path = irisPath)

# These aren't real paths
# h2o.exportFile(iris.hex, path = "/path/on/h2o/server/filesystem/iris.csv")
# h2o.exportFile(iris.hex, path = "hdfs://path/in/hdfs/iris.csv")
# h2o.exportFile(iris.hex, path = "s3n://path/in/s3/iris.csv")

## End(Not run)
```

---

**h2o.exportHDFS**

*Export a Model to HDFS*

**Description**

Exports an H2OModel to HDFS.

**Usage**

```r
h2o.exportHDFS(object, path, force = FALSE)
```

**Arguments**

- **object**: an H2OModel class object.
- **path**: The path to write the model to. Must include the directory and filename.
- **force**: logical, indicates how to deal with files that already exist.

---

**h2o.fillna**

*fillNA*

**Description**

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

**Usage**

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

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

**Value**

An H2OFrame after filling missing values

**Examples**

```r
library(h2o)
h2o.init()
fr.with.nas = h2o.createFrame(categorical_fraction=0.0,missing_fraction=0.7,rows=6,cols=2,seed=123)
fr <- h2o.fillna(fr.with.nas, "forward", axis=1, maxlen=2L)
```

---

**h2o.filterNACols**

Filter NA Columns

**Description**

Filter NA Columns

**Usage**

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

**Arguments**

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

**Value**

Returns a numeric vector of indexes that pertain to non-NA columns
**h2o.findSynonyms**

*Find synonyms using a word2vec model.*

**Description**

Find synonyms using a word2vec model.

**Usage**

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

**Arguments**

- **word2vec**: A word2vec model.
- **word**: A single word to find synonyms for.
- **count**: The top 'count' synonyms will be returned.

---

**h2o.find_row_by_threshold**

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

**Description**

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

**Usage**

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

**Arguments**

- **object**: H2OBinomialMetrics
- **threshold**: number between 0 and 1
h2o.find_threshold_by_max_metric

*Find the threshold, give the max metric*

**Description**

Find the threshold, give the max metric

**Usage**

h2o.find_threshold_by_max_metric(object, metric)

**Arguments**

- **object** H2OBinomialMetrics
- **metric** "F1," for example

h2o.floor

*Take a single numeric argument and return a numeric vector with the largest integers*

**Description**

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

**Usage**

h2o.floor(x)

**Arguments**

- **x** An H2OFrame object.

**See Also**

floor for the base R implementation.
**h2o.flow**  
*Open H2O Flow*

**Description**

Open H2O Flow in your browser

**Usage**

h2o.flow()

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

**Description**

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

**Usage**

h2o.gainsLift(object, ...)

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

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

**Arguments**

- **object**  
  Either an `H2OModel` object or an `H2OModelMetrics` object.

- **...**  
  Further arguments to be passed to/from this method.

- **newdata**  
  An H2OFrame object that can be scored on. Requires a valid response column.

- **valid**  
  Retrieve the validation metric.

- **xval**  
  Retrieve the cross-validation metric.

**Details**

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

**Value**

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

predict for generating prediction frames, h2o.performance for creating H2OModelMetrics.

Examples

```r
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.uploadFile(prosPath)
hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, distribution = "bernoulli",
                 training_frame = hex, validation_frame = hex, nfolds=3)

h2o.gainslift(model)  ## extract training metrics
h2o.gainslift(model, valid=TRUE) ## extract validation metrics (here: the same)

h2o.gainslift(model, xval =TRUE) ## extract cross-validation metrics
h2o.gainslift(model, newdata=hex) ## score on new data (here: the same)

# Generating a ModelMetrics object
perf <- h2o.performance(model, hex)

h2o.gainslift(perf)  ## extract from existing metrics object
```

---

**h2o.gbm**

Build gradient boosted classification or regression trees

Description

Builds gradient boosted classification trees and gradient boosted regression trees on a parsed data set. The default distribution function will guess the model type based on the response column type. In order to run properly, the response column must be an numeric for "gaussian" or an enum for "bernoulli" or "multinomial".

Usage

```r
h2o.gbm(x, y, training_frame, model_id = NULL, validation_frame = NULL,
nfolds = 0, keep_cross_validation_predictions = FALSE,
keep_cross_validation_fold_assignment = FALSE,
score_each_iteration = FALSE, score_tree_interval = 0,
fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
fold_column = NULL, ignore_const_cols = TRUE, offset_column = NULL,
weights_column = NULL, balance_classes = FALSE,
class_sampling_factors = NULL, max_after_balance_size = 5,
max_hit_ratio_k = 0, ntrees = 50, max_depth = 5, min_rows = 10,
nbins = 20, nbins_top_level = 1024, nbins_cats = 1024,
r2_stopping = Inf, stopping_rounds = 0, stopping_metric = c("AUTO",
"deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC",
"lift_top_group", "misclassification", "mean_per_class_error"),
stopping_tolerance = 0.001,
max_runtime_secs = 0, seed = -1, build_tree_one_node = FALSE,
```

learn_rate = 0.1, learn_rate_annealing = 1, distribution = c("AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"), quantile_alpha = 0.5, tweedie_power = 1.5, huber_alpha = 0.9, checkpoint = NULL, sample_rate = 1, sample_rate_per_class = NULL, col_sample_rate = 1, col_sample_rate_change_per_level = 1, col_sample_rate_per_tree = 1, min_split_improvement = 1e-05, histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin"), max_abs_leafnode_pred = Inf, pred_noise_bandwidth = 0, categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"), calibrate_model = FALSE, calibration_frame = NULL, custom_metric_func = NULL, verbose = FALSE)

Arguments

x (Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

y The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

training_frame Id of the training data frame.

model_id Destination id for this model; auto-generated if not specified.

validation_frame Id of the validation data frame.

nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

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

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

ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset_column</td>
<td>Offset column. This will be added to the combination of columns before applying the link function.</td>
</tr>
<tr>
<td>weights_column</td>
<td>Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.</td>
</tr>
<tr>
<td>balance_classes</td>
<td>Logical. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.</td>
</tr>
<tr>
<td>class_sampling_factors</td>
<td>Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.</td>
</tr>
<tr>
<td>max_after_balance_size</td>
<td>Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.</td>
</tr>
<tr>
<td>max_hit_ratio_k</td>
<td>Max. number (top K) of predictions to use for hit ratio computation (for multi-class only, 0 to disable) Defaults to 0.</td>
</tr>
<tr>
<td>ntrees</td>
<td>Number of trees. Defaults to 50.</td>
</tr>
<tr>
<td>max_depth</td>
<td>Maximum tree depth. Defaults to 5.</td>
</tr>
<tr>
<td>min_rows</td>
<td>Fewest allowed (weighted) observations in a leaf. Defaults to 10.</td>
</tr>
<tr>
<td>nbins</td>
<td>For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.</td>
</tr>
<tr>
<td>nbins_top_level</td>
<td>For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level Defaults to 1024.</td>
</tr>
<tr>
<td>nbins_cats</td>
<td>For categorical columns (factors), build a histogram of this many bins, then split at the best point. Higher values can lead to more overfitting. Defaults to 1024.</td>
</tr>
<tr>
<td>r2_stopping</td>
<td>r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this Defaults to 1.797693135e+308.</td>
</tr>
<tr>
<td>stopping_rounds</td>
<td>Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.</td>
</tr>
<tr>
<td>stopping_metric</td>
<td>Metric to use for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: &quot;AUTO&quot;, &quot;deviance&quot;, &quot;logloss&quot;, &quot;MSE&quot;, &quot;RMSE&quot;, &quot;MAE&quot;, &quot;RMSLE&quot;, &quot;AUC&quot;, &quot;lift_top_group&quot;, &quot;misclassification&quot;, &quot;mean_per_class_error&quot;. Defaults to AUTO.</td>
</tr>
</tbody>
</table>
**stopping_tolerance**
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

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

**seed**
Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Defaults to -1 (time-based random number).

**build_tree_one_node**
Logical. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.

**learn_rate**
Learning rate (from 0.0 to 1.0) Defaults to 0.1.

**learn_rate_annealing**
Scale the learning rate by this factor after each tree (e.g., 0.99 or 0.999) Defaults to 1.

**distribution**
Distribution function Must be one of: "AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". Defaults to AUTO.

**quantile_alpha**
Desired quantile for Quantile regression, must be between 0 and 1. Defaults to 0.5.

**tweedie_power**
Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to 1.5.

**huber_alpha**
Desired quantile for Huber/M-regression (threshold between quadratic and linear loss, must be between 0 and 1). Defaults to 0.9.

**checkpoint**
Model checkpoint to resume training with.

**sample_rate**
Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.

**sample_rate_per_class**
A list of row sample rates per class (relative fraction for each class, from 0.0 to 1.0), for each tree

**col_sample_rate**
Column sample rate (from 0.0 to 1.0) Defaults to 1.

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

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

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

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

**max_abs_leafnode_pred**
Maximum absolute value of a leaf node prediction Defaults to 1.797693135e+308.
pred_noise_bandwidth
- Bandwidth (sigma) of Gaussian multiplicative noise ~N(1, sigma) for tree node predictions. Defaults to 0.

categorical_encoding

calibrate_model
- Logical. Use Platt Scaling to calculate calibrated class probabilities. Calibration can provide more accurate estimates of class probabilities. Defaults to FALSE.

calibration_frame
- Calibration frame for Platt Scaling.

custom_metric_func
- Reference to custom evaluation function, format: 'language:keyName=funcName'.

verbose
- Logical. Print scoring history to the console (Metrics per tree for GBM, DRF, & XGBoost. Metrics per epoch for Deep Learning). Defaults to FALSE.

See Also
- `predict.H2OModel` for prediction.

Examples

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

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

---

**Description**

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

**Usage**

```r
h2o.getAutoML(project_name)
```
**h2o.getConnection**

Retrieve an H2O Connection

**Description**

Attempt to recover an h2o connection.

**Usage**

```
h2o.getConnection()
```

**Value**

Returns an **H2OConnection** object.

---

**h2o.getFrame**

Get an R Reference to an H2O Dataset, that will NOT be GC’d by default

**Description**

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

**Usage**

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

**Arguments**

id A string indicating the unique frame of the dataset to retrieve.
h2o.getFutureModel  
*Get future model*

**Description**
Get future model

**Usage**
```r
h2o.getFutureModel(object, verbose = FALSE)
```

**Arguments**
- **object**: H2OModel
- **verbose**: Print model progress to console. Default is FALSE

h2o.getGLMFullRegularizationPath

*Extract full regularization path from a GLM model*

**Description**
Extract the full regularization path from a GLM model (assuming it was run with the lambda search option).

**Usage**
```r
h2o.getGLMFullRegularizationPath(model)
```

**Arguments**
- **model**: an H2OModel corresponding from a h2o.glm call.
h2o.getGrid

Get a grid object from H2O distributed KV store.

Description

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

Usage

h2o.getGrid(grid_id, sort_by, decreasing)

Arguments

grid_id: ID of existing grid object to fetch

sort_by: Sort the models in the grid space by a metric. Choices are "logloss", "residual_deviance", "mse", "auc", "accuracy", "precision", "recall", "f1", etc.

decreasing: Specify whether sort order should be decreasing

Examples

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

h2o.getId

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

Description

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

h2o.getModel(x)

Arguments

x  An H2OFrame

Value

The id of the H2OFrame

h2o.getModel  Get an R reference to an H2O model

Description

Returns a reference to an existing model in the H2O instance.

Usage

h2o.getModel(model_id)

Arguments

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

Value

Returns an object that is a subclass of H2OModel.

Examples

library(h2o)
h2o.init()

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

*Get the Time Zone on the H2O Cloud Returns a string*

**Description**

Get the Time Zone on the H2O Cloud Returns a string

**Usage**

```r
h2o.getTimezone()
```

**h2o.getTypes**

*Get the types-per-column*

**Description**

Get the types-per-column

**Usage**

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

**Arguments**

- **x**: An H2OFrame

**Value**

A list of types per column

**h2o.getVersion**

*Get h2o version*

**Description**

Get h2o version

**Usage**

```r
h2o.getVersion()
```
h2o.giniCoef

Retrieve the GINI Coefficient

Description

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

Usage

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

Arguments

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

See Also

h2o.auc for AUC, h2o.giniCoef for the GINI coefficient, and h2o.metric for the various. See h2o.performance for creating H2OModelMetrics objects. threshold metrics.

Examples

library(h2o)
h2o.init()

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

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

Description

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

Usage

```r
h2o.glm(x, y, training_frame, model_id = NULL, validation_frame = NULL, n_folds = 0, seed = -1, keep_cross_validation_predictions = FALSE, keep_cross_validation_fold_assignment = FALSE, fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"), fold_column = NULL, ignore_const_cols = TRUE, score_each_iteration = FALSE, offset_column = NULL, weights_column = NULL, family = c("gaussian", "binomial", "quasibinomial", "ordinal", "multinomial", "poisson", "gamma", "tweedie"), tweedie_variance_power = 0, tweedie_link_power = 1, solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"), alpha = NULL, lambda = NULL, lambda_search = FALSE, early_stopping = TRUE, n_lambdas = -1, standardize = TRUE, missing_values_handling = c("MeanImputation", "Skip"), compute_p_values = TRUE, remove_collinear_columns = FALSE, intercept = TRUE, non_negative = FALSE, max_iterations = -1, objective_epsilon = -1, beta_epsilon = 1e-04, gradient_epsilon = -1, link = c("family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit", "oprobit", "ologlog"), prior = -1, lambda_min_ratio = -1, beta_constraints = NULL, max_active_predictors = -1, interactions = NULL, interaction_pairs = NULL, obj_reg = -1, balance_classes = FALSE, class_sampling_factors = NULL, max_after_balance_size = 5, max_hit_ratio_k = 0, max_runtime_secs = 0, custom_metric_func = NULL)
```

Arguments

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

seed Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Defaults to -1 (time-based random number).

keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.

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

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

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

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

tweedie_variance_power Tweedie variance power Defaults to 0.

tweedie_link_power Tweedie link power Defaults to 1.

solver AUTO will set the solver based on given data and the other parameters. IRLSM is fast on problems with small number of predictors and for lambda-search with L1 penalty. L_BFGS scales better for datasets with many columns. Coordinate descent is experimental (beta). Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR". Defaults to AUTO.
alpha  Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.

lambda  Regularization strength

lambda_search  Logical. Use lambda search starting at lambda max, given lambda is then interpreted as lambda min Defaults to FALSE.

early_stopping  Logical. Stop early when there is no more relative improvement on train or validation (if provided) Defaults to TRUE.

nlambdas  Number of lambdas to be used in a search. Default indicates: If alpha is zero, with lambda search set to True, the value of nlamdas is set to 30 (fewer lambdas are needed for ridge regression) otherwise it is set to 100. Defaults to -1.

standardize  Logical. Standardize numeric columns to have zero mean and unit variance Defaults to TRUE.

missing_values_handling  Handling of missing values. Either MeanImputation or Skip. Must be one of: "MeanImputation", "Skip". Defaults to MeanImputation.

compute_p_values  Logical. Request p-values computation, p-values work only with IRLSM solver and no regularization Defaults to FALSE.

remove_collinear_columns  Logical. In case of linearly dependent columns, remove some of the dependent columns Defaults to FALSE.

intercept  Logical. Include constant term in the model Defaults to TRUE.

non_negative  Logical. Restrict coefficients (not intercept) to be non-negative Defaults to FALSE.

max_iterations  Maximum number of iterations Defaults to -1.

objective_epsilon  Converge if objective value changes less than this. Default indicates: If lambda_search is set to True the value of objective_epsilon is set to .0001. If the lambda_search is set to False and lambda is equal to zero, the value of objective_epsilon is set to .000001, for any other value of lambda the default value of objective_epsilon is set to .0001. Defaults to -1.

beta_epsilon  Converge if beta changes less (using L-infinity norm) than beta epsilon, ONLY applies to IRLSM solver Defaults to 0.0001.

gradient_epsilon  Converge if objective changes less (using L-infinity norm) than this, ONLY applies to L-BFGS solver. Default indicates: If lambda_search is set to False and lambda is equal to zero, the default value of gradient_epsilon is equal to .000001, otherwise the default value is .0001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively. Defaults to -1.

link  Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit", "oprobit", "ologlog". Defaults to family_default.

prior  Prior probability for y==1. To be used only for logistic regression iff the data has been sampled and the mean of response does not reflect reality. Defaults to -1.
lambda_min_ratio
Minimum lambda used in lambda search, specified as a ratio of lambda_max (the smallest lambda that drives all coefficients to zero). Default indicates: if the number of observations is greater than the number of variables, then lambda_min_ratio is set to 0.0001; if the number of observations is less than the number of variables, then lambda_min_ratio is set to 0.01. Defaults to -1.

beta_constraints
Beta constraints

max_active_predictors
Maximum number of active predictors during computation. Use as a stopping criterion to prevent expensive model building with many predictors. Default indicates: If the IRLSM solver is used, the value of max_active_predictors is set to 5000 otherwise it is set to 10000000. Defaults to -1.

interactions
A list of predictor column indices to interact. All pairwise combinations will be computed for the list.

interaction_pairs
A list of pairwise (first order) column interactions.

obj_reg
Likelihood divider in objective value computation, default is 1/nobs Defaults to -1.

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

class_sampling_factors
Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

max_after_balance_size
Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

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

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

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

Value
A subclass of H2OModel is returned. The specific subclass depends on the machine learning task at hand (if it’s binomial classification, then an H2OBinomialModel is returned, if it’s regression then a H2ORegressionModel is returned). The default print- out of the models is shown, but further GLM-specific information can be queried out of the object. To access these various items, please refer to the seealso section below. Upon completion of the GLM, the resulting object has coefficients, normalized coefficients, residual/null deviance, aic, and a host of model metrics including MSE, AUC (for logistic regression), degrees of freedom, and confusion matrices. Please refer to the
more in-depth GLM documentation available here: https://h2o-release.s3.amazonaws.com/h2o-dev/rel-shannon/2/docs-website/h2o-docs/index.html#Data+Science+Algorithms-GLM

See Also

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

Examples

h2o.init()

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

# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
myX = setdiff(colnames(prostate.hex), c("ID", "DPROS", "DCAPS", "VOL"))
myY=

h2o.glm(y = "VOL", x = myX, training_frame = prostate.hex, family = "gaussian", nfolds = 0, alpha = 0.1, lambda_search = FALSE)

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

my.glm = h2o.glm(x=myX, y=myY, training_frame=data.hex, family="binomial", standardize=TRUE, lambda_search=TRUE)
"STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"), k = 1,
loss = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic",
"Periodic"), loss_by_col = c("Quadratic", "Absolute", "Huber", "Poisson",
"Hinge", "Logistic", "Periodic", "Categorical", "Ordinal"),
loss_by_col_idx = NULL, multi_loss = c("Categorical", "Ordinal"),
period = 1, regularization_x = c("None", "Quadratic", "L2", "L1",
"NonNegative", "OneSparse", "UnitOneSparse", "Simplex"),
regularization_y = c("None", "Quadratic", "L2", "L1", "NonNegative",
"OneSparse", "UnitOneSparse", "Simplex"), gamma_x = 0, gamma_y = 0,
max_iterations = 1000, max_updates = 2000, init_step_size = 1,
min_step_size = 1e-04, seed = -1, init = c("Random", "SVD", "PlusPlus",
"User"), svd_method = c("GramSVD", "Power", "Randomized"), user_y = NULL,
user_x = NULL, expand_user_y = TRUE, impute_original = FALSE,
recover_svd = FALSE, max_runtime_secs = PI

Arguments

training_frame  Id of the training data frame.
cols  (Optional) A vector containing the data columns on which k-means operates.
model_id  Destination id for this model; auto-generated if not specified.
validation_frame  Id of the validation data frame.
ignore_const_cols  Logical. Ignore constant columns. Defaults to TRUE.
score_each_iteration  Logical. Whether to score during each iteration of model training. Defaults to FALSE.
loading_name  Frame key to save resulting X
transform  Transformation of training data Must be one of: "NONE", "STANDARDIZE",
"NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
k  Rank of matrix approximation Defaults to 1.
loss  Numeric loss function Must be one of: "Quadratic", "Absolute", "Huber", "Poisson",
"Hinge", "Logistic", "Periodic". Defaults to Quadratic.
loss_by_col  Loss function by column (override) Must be one of: "Quadratic", "Absolute",
"Huber", "Poisson", "Hinge", "Logistic", "Periodic", "Categorical", "Ordinal".
loss_by_col_idx  Loss function by column index (override)
multi_loss  Categorical loss function Must be one of: "Categorical", "Ordinal". Defaults to Categorical.
period  Length of period (only used with periodic loss function) Defaults to 1.
regularization_x  Regularization function for X matrix Must be one of: "None", "Quadratic",
"L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex". Defaults to None.
regularization_y  
Regularization function for Y matrix Must be one of: "None", "Quadratic",  
"L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex".  
Defaults to None.

gamma_x  
Regularization weight on X matrix Defaults to 0.

gamma_y  
Regularization weight on Y matrix Defaults to 0.

max_iterations  
Maximum number of iterations Defaults to 1000.

max_updates  
Maximum number of updates, defaults to 2*max_iterations Defaults to 2000.

init_step_size  
Initial step size Defaults to 1.

min_step_size  
Minimum step size Defaults to 0.0001.

seed  
Seed for random numbers (affects certain parts of the algo that are stochastic  
and those might or might not be enabled by default) Defaults to -1 (time-based  
random number).

init  
Initialization mode Must be one of: "Random", "SVD", "PlusPlus", "User".  
Defaults to PlusPlus.

svd_method  
Method for computing SVD during initialization (Caution: Randomized is cur-  
rently experimental and unstable) Must be one of: "GramSVD", "Power", "Random-  
dized". Defaults to Randomized.

user_y  
User-specified initial Y

user_x  
User-specified initial X

expand_user_y  
Logical. Expand categorical columns in user-specified initial Y Defaults to  
TRUE.

impute_original  
Logical. Reconstruct original training data by reversing transform Defaults to  
FALSE.

recover_svd  
Logical. Recover singular values and eigenvectors of XY Defaults to FALSE.

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

Value

Returns an object of class H2ODimReductionModel.

References

Unpublished manuscript, Stanford Electrical Engineering Department N. Halko, P.G. Martinsson,  
J.A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approxi-  

See Also

h2o.kmeans, h2o.svd, h2o.prcomp
Examples

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

**h2o.grep**

*Search for matches to an argument pattern*

Description

Searches for matches to argument ‘pattern’ within each element of a string column.

Usage

```r
h2o.grep(pattern, x, ignore.case = FALSE, invert = FALSE,
output.logical = FALSE)
```

Arguments

- **pattern**: A character string containing a regular expression.
- **x**: An H2O frame that wraps a single string column.
- **ignore.case**: If TRUE case is ignored during matching.
- **invert**: Identify elements that do not match the pattern.
- **output.logical**: If TRUE returns logical vector of indicators instead of list of matching positions.

Details

This function has similar semantics as R’s native grep function and it supports a subset of its parameters. Default behavior is to return indices of the elements matching the pattern. Parameter ‘output.logical’ can be used to return a logical vector indicating if the element matches the pattern (1) or not (0).

Value

H2OFrame holding the matching positions or a logical vector if ‘output.logical’ is enabled.

Examples

```r
library(h2o)
h2o.init()
addresses <- as.h2o(c("2307", "Leghorn St", "Mountain View", "CA", "94043"))
zip.codes <- addresses[h2o.grep("[0-9]{5}", addresses, output.logical = TRUE),]
```
Description

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

Usage

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

Arguments

algorithm Name of algorithm to use in grid search (gbm, randomForest, kmeans, glm,
          deepLearning, naivebayes, pca).
grid_id (Optional) ID for resulting grid search. If it is not specified then it is autogener-
          ated.
x (Optional) A vector containing the names or indices of the predictor variables to
          use in building the model. If x is missing, then all columns except y are used.
y The name or column index of the response variable in the data. The response
          must be either a numeric or a categorical/factor variable. If the response is
          numeric, then a regression model will be trained, otherwise it will train a classi-
          fication model.
training_frame Id of the training data frame.
... arguments describing parameters to use with algorithm (i.e., x, y, training_frame).
          Look at the specific algorithm - h2o.gbm, h2o.glm, h2o.kmeans, h2o.deepLearning
          - for available parameters.
hyper_params List of lists of hyper parameters (i.e., list(ntrees=c(1,2), max_depth=c(5,7))).
is_supervised (Optional) If specified then override the default heuristic which decides if the
given algorithm name and parameters specify a supervised or unsupervised al-
          gorithm.
do_hyper_params_check Perform client check for specified hyper parameters. It can be time expensive
          for large hyper space.
search_criteria (Optional) List of control parameters for smarter hyperparameter search. The
default strategy 'Cartesian' covers the entire space of hyperparameter combi-
          nations. Specify the 'RandomDiscrete' strategy to get random search of all
          the combinations of your hyperparameters. RandomDiscrete should be usually
          combined with at least one early stopping criterion, max_models and/or
          max_runtime_secs, e.g. list(strategy = "RandomDiscrete", max_models = 42, max_runtime_secs
          or list(strategy = "RandomDiscrete", stopping_metric = "AUTO", stopping_tolerance = 0.01,
          or list(strategy = "RandomDiscrete", stopping_metric = "misclassification", stopping_time...
h2o.group_by

Group and Apply by Column

Description

Perform a group by and apply similar to ddply.

Usage

h2o.group_by(data, by, ..., gb.control = list(na.methods = NULL, col.names = NULL))

Arguments

data an H2OFrame object.
by a list of column names
... any supported aggregate function. See Details: for more help.
gb.control a list of how to handle NA values in the dataset as well as how to name output columns. The method is specified using the rm.method argument. See Details: for more help.

Details

In the case of na.methods within gb.control, there are three possible settings. "all" will include NAs in computation of functions. "rm" will completely remove all NA fields. "ignore" will remove NAs from the numerator but keep the rows for computational purposes. If a list smaller than the number of columns groups is supplied, the list will be padded by "ignore".
Note that to specify a list of column names in the gb.control list, you must add the col.names argument. Similar to na.methods, col.names will pad the list with the default column names if the length is less than the number of columns supplied.

Supported functions include \texttt{nrow}. This function is required and accepts a string for the name of the generated column. Other supported aggregate functions accept \texttt{col} and \texttt{na} arguments for specifying columns and the handling of NAs ("\texttt{all}", "\texttt{ignore}", and GroupBy object; \texttt{max} calculates the maximum of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{mean} calculates the mean of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{min} calculates the minimum of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{mode} calculates the mode of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{sd} calculates the standard deviation of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{ss} calculates the sum of squares of each column specified in \texttt{col} for each group of a GroupBy object; \texttt{sum} calculates the sum of each column specified in \texttt{col} for each group of a GroupBy object; and \texttt{var} calculates the variance of each column specified in \texttt{col} for each group of a GroupBy object. If an aggregate is provided without a value (for example, as \texttt{max} in \texttt{sum(col="X1", na="all").mean(col="X5", na="all").max()}) then it is assumed that the aggregation should apply to all columns except the GroupBy columns. Note again that \texttt{nrow} is required and cannot be empty.

### Value

Returns a new H2OFrame object with columns equivalent to the number of groups created.

### Description

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

### Usage

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

### Arguments

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

```r
library(h2o)
h2o.init()
string_to_gsub <- as.h2o("r tutorial")
sub_string <- h2o.gsub("r","H2O ",string_to_gsub)
```

---

**h2o.head**

*Return the Head or Tail of an H2O Dataset.*

Description

Returns the first or last rows of an H2OFrame object.

Usage

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

### S3 method for class 'H2OFrame'

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

### S3 method for class 'H2OFrame'

```r
tail(x, n = 6L, ...)
```

Arguments

- **x**  
  An H2OFrame object.
- **n**  
  (Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x.
- **...**  
  Ignored.

Value

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

Examples

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

**Compute A Histogram**

**Description**

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

**Usage**

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

**Arguments**

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

h2o.hit_ratio_table

**Retrieve the Hit Ratios**

**Description**

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

**Usage**

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

**Arguments**

- **object**: An H2OModel object.
- **train**: Retrieve the training Hit Ratio
- **valid**: Retrieve the validation Hit Ratio
- **xval**: Retrieve the cross-validation Hit Ratio
h2o.hour  

Convert Milliseconds to Hour of Day in H2O Datasets

Description

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

Usage

h2o.hour(x)

hour(x)

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

Arguments

x  An H2OFrame object.

Value

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

See Also

h2o.day

h2o.ifelse  

H2O Apply Conditional Statement

Description

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

Usage

h2o.ifelse(test, yes, no)

ifelse(test, yes, no)
Arguments

test

test A logical description of the condition to be met (>, <, =, etc...)

yes

The value to return if the condition is TRUE.

no

The value to return if the condition is FALSE.

Details

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

Value

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

Examples

```r
h2o.init()
ausPath <- system.file("extdata", "australia.csv", package="h2o")
australia.hex <- h2o.importFile(path = ausPath)
australia.hex[,9] <- ifelse(australia.hex[,3] < 279.9, 1, 0)
summary(australia.hex)
```

Import Files into H2O

Description

Imports files into an H2O cloud. The default behavior is to pass-through to the parse phase automatically.

Usage

```r
h2o.importFile(path, destination_frame = "", parse = TRUE, header = NA,
sep = "", col.names = NULL, col.types = NULL, na.strings = NULL,
decrypt_tool = NULL)

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

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

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

Arguments

path
The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.

destination_frame
(Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.

parse
(Optional) A logical value indicating whether the file should be parsed after import, for details see h2o.parseRaw.

header
(Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.

sep
(Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.

col.names
(Optional) An H2OFrame object containing a single delimited line with the column names for the file.

col.types
(Optional) A vector to specify whether columns should be forced to a certain type upon import parsing.

na.strings
(Optional) H2O will interpret these strings as missing.

decrypt_tool
(Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.

pattern
(Optional) Character string containing a regular expression to match file(s) in the folder.

progressBar
(Optional) When FALSE, tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.

parse_type
(Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"

Details

h2o.importFile is a parallelized reader and pulls information from the server from a location specified by the client. The path is a server-side path. This is a fast, scalable, highly optimized way to read data. H2O pulls the data from a data store and initiates the data transfer as a read operation.

Unlike the import function, which is a parallelized reader, h2o.uploadFile is a push from the client to the server. The specified path must be a client-side path. This is not scalable and is only intended for smaller data sizes. The client pushes the data from a local filesystem (for example, on your machine where R is running) to H2O. For big-data operations, you don’t want the data stored on or flowing through the client.

h2o.importFolder imports an entire directory of files. If the given path is relative, then it will be relative to the start location of the H2O instance. The default behavior is to pass-through to the parse phase automatically.

h2o.importHDFS is deprecated. Instead, use h2o.importFile.

See Also

h2o.import_sql_select, h2o.import_sql_table, h2o.parseRaw
**Examples**

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

#Import files with a certain regex pattern by utilizing h2o.importFolder()
#In this example we import all .csv files in the directory prostate_folder
prosPath = system.file("extdata", "prostate_folder", package = "h2o")
prostate_pattern.hex = h2o.importFolder(path = prosPath, pattern = ".*\.csv",
                                       destination_frame = "prostate.hex")
class(prostate_pattern.hex)
summary(prostate_pattern.hex)
```

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

**Description**

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

**Usage**

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

**Arguments**

- `connection_url`  
  URL of the SQL database connection as specified by the Java Database Connectivity (JDBC) Driver. For example, "jdbc:mysql://localhost:3306/menagerie?&useSSL=false"

- `select_query`  
  SQL query starting with 'SELECT' that returns rows from one or more database tables.

- `username`  
  Username for SQL server

- `password`  
  Password for SQL server

- `optimize`  
  (Optional) Optimize import of SQL table for faster imports. Experimental. Default is true.
**Details**

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

---

**h2o.import_sql_table**  
**Import SQL Table into H2O**

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

Usage

\[
\text{h2o.impute(data, column = 0, method = c(\"mean\", \"median\", \"mode\"),}
\text{combine_method = c(\"interpolate\", \"average\", \"lo\", \"hi\"), by = NULL,}
\text{groupByFrame = NULL, values = NULL)}
\]

Arguments

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

Details

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

Value

an H2OFrame with imputed values

Examples

\[
\begin{align*}
\text{h2o.init()}
\text{fr <- as.h2o(iris, destination_frame=\"iris\")}
\text{fr[sample(nrow(fr), 50)] <- NA} \quad \# \text{randomly replace 50 values with NA}
\text{# impute with a group by}
\text{fr <- h2o.impute(fr, \"Species\", \"mode\", by=c(\"Sepal.Length\", \"Sepal.Width\")}
\end{align*}
\]
**Description**

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

**Usage**

```r
h2o.init(ip = "localhost", port = 54321, startH2O = TRUE,
          forceDL = FALSE, enable_assertions = TRUE, license = NULL,
          nthreads = -1, max_mem_size = NULL, min_mem_size = NULL,
          ice_root = tempdir(), strict_version_check = TRUE,
          proxy = NA_character_, https = FALSE, insecure = FALSE,
          username = NA_character_, password = NA_character_,
          cookies = NA_character_, context_path = NA_character_,
          ignore_config = FALSE, extra_classpath = 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 try to start H2O from R if no connection with H2O is detected. This is only possible if `ip = "localhost"` or `ip = "127.0.0.1"`. If an existing connection is detected, R does not start H2O.
- **forceDL** (Optional) A logical value indicating whether to force download of the H2O executable. Defaults to FALSE, so the executable will only be downloaded if it does not already exist in the h2o R library resources directory `h2o/java/h2o.jar`. This value is only used when R starts H2O.
- **enable_assertions** (Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes. This value is only used when R starts H2O.
- **license** (Optional) A character string value specifying the full path of the license file. This value is only used when R starts H2O.
- **nthreads** (Optional) Number of threads in the thread pool. This relates very closely to the number of CPUs used. -1 means use all CPUs on the host (Default). A positive integer specifies the number of CPUs directly. This value is only used when R starts H2O.
- **max_mem_size** (Optional) A character string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.
min_mem_size (Optional) A character string specifying the minimum size, in bytes, of the memory allocation pool to H2O. This value must be a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.

ice_root (Optional) A directory to handle object spillage. The default varies by OS.

strict_version_check (Optional) Setting this to FALSE is unsupported and should only be done when advised by technical support.

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

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

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

username (Optional) Username to login with.

password (Optional) Password to login with.

cookies (Optional) Vector (or list) of cookies to add to request.

context_path (Optional) The last part of connection URL: http://<ip>:<port>/<context_path>

ignore_config (Optional) A logical value indicating whether a search for a .h2oconfig file should be conducted or not. Default value is FALSE.

extra_classpath (Optional) A vector of paths to libraries to be added to the Java classpath when H2O is started from R.

Details

By default, this method first checks if an H2O instance is connectible. If it cannot connect and start = TRUE with ip = "localhost", it will attempt to start and instance of H2O at localhost:54321. If an open ip and port of your choice are passed in, then this method will attempt to start an H2O instance at that specified ip port.

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

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

Value

this method will load it and return a H2OConnection object containing the IP address and port number of the H2O server.

Note

Users may wish to manually upgrade their package (rather than waiting until being prompted), which requires that they fully uninstall and reinstall the H2O package, and the H2O client package. You must unload packages running in the environment before upgrading. It’s recommended that users restart R or R studio after upgrading.
h2o.insertMissingValues

Insert Missing Values into an H2OFrame

Description
Randomly replaces a user-specified fraction of entries in an H2O dataset with missing values.

Usage
h2o.insertMissingValues(data, fraction = 0.1, seed = -1)

Arguments
- data: An H2OFrame object representing the dataset.
- fraction: A number between 0 and 1 indicating the fraction of entries to replace with missing.
- seed: A random number used to select which entries to replace with missing values. Default of seed = -1 will automatically generate a seed in H2O.

Value
Returns an H2OFrame object.

See Also
H2O R package documentation for more details. h2o.shutdown for shutting down from R.

Examples
```r
## Not run:
# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R with the default settings.
h2o.init()

# Try to connect to a local H2O instance.
# If not found, raise an error.
h2o.init(startH2O = FALSE)

# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R with 5 gigabytes of memory.
h2o.init(max_mem_size = "5g")

# Try to connect to a local H2O instance that is already running.
# If not found, start a local H2O instance from R that uses 5 gigabytes of memory.
h2o.init(max_mem_size = "5g")

## End(Not run)
```
**h2o.interaction**

**Description**

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

**Usage**

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

**Arguments**

- **data**
  - An H2OFrame object containing the categorical columns.

- **destination_frame**
  - A string indicating the destination key. If empty, this will be auto-generated by H2O.

- **factors**
  - Factor columns (either indices or column names).

- **pairwise**
  - Whether to create pairwise interactions between factors (otherwise create one higher-order interaction). Only applicable if there are 3 or more factors.

- **max_factors**
  - Max. number of factor levels in pair-wise interaction terms (if enforced, one extra catch-all factor will be made)

- **min_occurrence**
  - Min. occurrence threshold for factor levels in pair-wise interaction terms

**Value**

Returns an H2OFrame object.

**WARNINGS**

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

**Examples**

```r
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris.csv", package = "h2o")
iris.hex <- h2o.importFile(path = irisPath)
summary(iris.hex)
irismiss.hex <- h2o.insertMissingValues(iris.hex, fraction = 0.25)
head(irismiss.hex)
summary(irismiss.hex)
```
Examples

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

# Create some random data
myframe <- h2o.frame(rows = 20, cols = 5,
seed = -12301283, randomize = TRUE, value = 0,
categorical_fraction = 0.8, factors = 10, real_range = 1,
integer_fraction = 0.2, integer_range = 10,
binary_fraction = 0, binary_ones_fraction = 0.5,
missing_fraction = 0.2,
response_factors = 1)

# Turn integer column into a categorical
myframe[,5] <- as.factor(myframe[,5])
head(myframe, 20)

# Create pairwise interactions
pairwise <- h2o.frame(rows = 20)
head(pairwise, 20)

# Create 5-th order interaction
higherorder <- h2o.frame(rows = 20)
head(higherorder, 20)

# Limit the number of factors of the "categoricalized" integer column
# to at most 3 factors, and only if they occur at least twice
head(myframe[,5], 20)

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

---

**h2o.isax**

**iSAX**

**Description**

Compute the iSAX index for a DataFrame which is assumed to be numeric time series data.
h2o.ischaracter

Usage

h2o.isax(x, num_words, max_cardinality, optimize_card = FALSE)

Arguments

x an H2OFrame
num_words Number of iSAX words for the timeseries. i.e granularity along the time series
max_cardinality Maximum cardinality of the iSAX word. Each word can have less than the max
optimize_card An optimization flag that will find the max cardinality regardless of what is passed in for max_cardinality.

Value

An H2OFrame with the name of time series, string representation of iSAX word, followed by binary representation

References

http://www.cs.ucr.edu/~eamonn/iSAX_2.0.pdf
http://www.cs.ucr.edu/~eamonn/SAX.pdf

h2o.ischaracter Check if character

Description

Check if character

Usage

h2o.ischaracter(x)

Arguments

x An H2OFrame object.

See Also

is.character for the base R implementation.
h2o.isfactor  

**Description**
Check if factor

**Usage**
h2o.isfactor(x)

**Arguments**
x  
An H2OFrame object.

**See Also**
is.factor for the base R implementation.

h2o.isnumeric  

**Description**
Check if numeric

**Usage**
h2o.isnumeric(x)

**Arguments**
x  
An H2OFrame object.

**See Also**
is.numeric for the base R implementation.
**h2o.is_client**  
*Check Client Mode Connection*

**Description**

Check Client Mode Connection

**Usage**

```r
h2o.is_client()
```

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

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

Returns an H2OFrame object with fold assignments.

**h2o.killMinus3**  
*Dump the stack into the JVM’s stdout.*

**Description**

A poor man’s profiler, but effective.

**Usage**

```r
h2o.killMinus3()
```
h2o.kmeans

Performs k-means clustering on an H2O dataset

Description

Performs k-means clustering on an H2O dataset

Usage

```r
h2o.kmeans(training_frame, x, model_id = NULL, validation_frame = NULL,
           nfolds = 0, keep_cross_validation_predictions = FALSE,
           keep_cross_validation_fold_assignment = FALSE, fold_assignment = c("AUTO",
           "Random", "Modulo", "Stratified"), fold_column = NULL,
           ignore_const_cols = TRUE, score_each_iteration = FALSE, k = 1,
           estimate_k = FALSE, user_points = NULL, max_iterations = 10,
           standardize = TRUE, seed = -1, init = c("Random", "PlusPlus",
           "Furthest", "User"), max_runtime_secs = 0,
           categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
           "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"))
```

Arguments

- `training_frame` Id of the training data frame.
- `x` A vector containing the character names of the predictors in the model.
- `model_id` Destination id for this model; auto-generated if not specified.
- `validation_frame` Id of the validation data frame.
- `nfolds` Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
- `keep_cross_validation_predictions` Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
- `keep_cross_validation_fold_assignment` Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
- `fold_assignment` Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
- `fold_column` Column with cross-validation fold index assignment per observation.
- `ignore_const_cols` Logical. Ignore constant columns. Defaults to TRUE.
- `score_each_iteration` Logical. Whether to score during each iteration of model training. Defaults to FALSE.
k

The max. number of clusters. If estimate_k is disabled, the model will find k centroids, otherwise it will find up to k centroids. Defaults to 1.

estimate_k

Logical. Whether to estimate the number of clusters (<=k) iteratively and deterministically. Defaults to FALSE.

user_points

This option allows you to specify a dataframe, where each row represents an initial cluster center. The user-specified points must have the same number of columns as the training observations. The number of rows must equal the number of clusters.

max_iterations

Maximum training iterations (if estimate_k is enabled, then this is for each inner Lloyds iteration) Defaults to 10.

standardize

Logical. Standardize columns before computing distances Defaults to TRUE.

seed

Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Defaults to -1 (time-based random number).

init

Initialization mode Must be one of: "Random", "PlusPlus", "Furthest", "User". Defaults to Furthest.

max_runtime_secs

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

categorical_encoding

Encoding scheme for categorical features Must be one of: "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "Sort-ByResponse", "EnumLimited". Defaults to AUTO.

Value

Returns an object of class H2OClusteringModel.

See Also

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

Examples

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

**Description**

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

**Usage**

```r
h2o.kurtosis(x, ..., na.rm = TRUE)
kurtosis.H2OFrame(x, ..., na.rm = TRUE)
```

**Arguments**

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

**Value**

Returns a list containing the kurtosis for each column (`NaN` for non-numeric columns).

**Examples**

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

h2o.kurtosis(prostate.hex$AGE)
```

---

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

**Description**

Return the levels from the column requested column.

**Usage**

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

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

See Also

levels for the base R method.

Examples

iris.hex <- as.h2o(iris)
h2o.levels(iris.hex, 5)  # returns "setosa"  "versicolor" "virginica"

h2o.listTimezones List all of the Time Zones Acceptable by the H2O Cloud.

Description

List all of the Time Zones Acceptable by the H2O Cloud.

Usage

h2o.listTimezones()

h2o.list_all_extensions List all H2O registered extensions

Description

List all H2O registered extensions

Usage

h2o.list_all_extensions()
h2o.list_api_extensions

Description
List registered API extensions

Usage
h2o.list_api_extensions()

h2o.list_core_extensions

Description
List registered core extensions

Usage
h2o.list_core_extensions()

h2o.loadModel

Description
Load H2O Model from HDFS or Local Disk

Usage
h2o.loadModel(path)

Arguments
path
The path of the H2O Model to be imported. and port of the server running H2O.

Value
Returns a H2OModel object of the class corresponding to the type of model built.
h2o.log

**h2o.log**

Compute the logarithm of x

**Description**

Compute the logarithm of x

**Usage**

h2o.log(x)

**Arguments**

x An H2OFrame object.

**See Also**

log for the base R implementation.

---

h2o.log10

**h2o.log10**

Compute the log10 of x

**Description**

Compute the log10 of x

**Usage**

h2o.log10(x)
Arguments
  x  An H2OFrame object.

See Also
  \texttt{log1p} for the base R implementation.

\begin{tabular}{ll}
\hline
h2o.log1p & \textit{Compute the log1p of x} \\
\hline
\end{tabular}

Description
  Compute the log1p of x

Usage
  h2o.log1p(x)

Arguments
  x  An H2OFrame object.

See Also
  \texttt{log1p} for the base R implementation.

\begin{tabular}{ll}
\hline
h2o.log2 & \textit{Compute the log2 of x} \\
\hline
\end{tabular}

Description
  Compute the log2 of x

Usage
  h2o.log2(x)

Arguments
  x  An H2OFrame object.

See Also
  \texttt{log2} for the base R implementation.
**h2o.logAndEcho**

Log a message on the server-side logs

**Description**

This is helpful when running several pieces of work one after the other on a single H2O cluster and you want to make a notation in the H2O server side log where one piece of work ends and the next piece of work begins.

**Usage**

```r
h2o.logAndEcho(message)
```

**Arguments**

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

**Details**

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

---

**h2o.logloss**

Retrieve the Log Loss Value

**Description**

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

**Usage**

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

**Arguments**

- `object` a H2OModelMetrics object of the correct type.
- `train` Retrieve the training Log Loss
- `valid` Retrieve the validation Log Loss
- `xval` Retrieve the cross-validation Log Loss
**h2o.ls**

*List Keys on an H2O Cluster*

**Description**

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

**Usage**

```r
h2o.ls()
```

**Value**

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

**Examples**

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

---

**h2o.lstrip**

*Strip set from left*

**Description**

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

**Usage**

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

**Arguments**

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

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

string_to_lstrip <- as.h2o("1234567890")
lstrip_string <- h2o.lstrip(string_to_lstrip,"123") #Remove "123"
```

---

```r
h2o.mae

Retrieve the Mean Absolute Error Value
```

Description

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

Usage

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

Arguments

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

Examples

```r
library(h2o)

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

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

h2o.mae(m)
```
**h2o.makeGLMModel**  
*Set betas of an existing H2O GLM Model*

**Description**

This function allows setting betas of an existing glm model.

**Usage**

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

**Arguments**

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

**h2o.make_metrics**  
*Create Model Metrics from predicted and actual values in H2O*

**Description**

Given predicted values (target for regression, class-1 probabilities or binomial or per-class probabilities for multinomial), compute a model metrics object

**Usage**

```r
h2o.make_metrics(predicted, actuals, domain = NULL, distribution = NULL)
```

**Arguments**

- `predicted`: An H2OFrame containing predictions
- `actuals`: An H2OFrame containing actual values
- `domain`: Vector with response factors for classification.
- `distribution`: Distribution for regression.

**Value**

Returns an object of the `H2OModelMetrics` subclass.
Examples

```r
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
prostate.hex$CAPSULE <- as фактор(prostate.hex$CAPSULE)
prostate.gbm <- h2o.gbm(3:9, "CAPSULE", prostate.hex)
pred <- h2o.predict(prostate.gbm, prostate.hex)[,3] ## class-1 probability
h2o.make_metrics(pred, prostate.hex$CAPSULE)
```

---

**h2o.match**  
*Value Matching in H2O*

### Description

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

### Usage

```r
h2o.match(x, table, nomatch = 0, incomparables = NULL)
macth.H2OFrame(x, table, nomatch = 0, incomparables = NULL)
```

`x %in% table`

### Arguments

- **x**: A categorical vector from an H2OFrame object with values to be matched.
- **table**: An R object to match `x` against.
- **nomatch**: The value to be returned in the case when no match is found.
- **incomparables**: A vector of values that cannot be matched. Any value in `x` matching a value in this vector is assigned the `nomatch` value.

### Value

Returns a vector of the positions of (first) matches of its first argument in its second

### See Also

`match` for base R implementation.
### Examples

```r
h2o.init()
hex <- as.h2o(iris)
h2o.match(hex[,5], c("setosa", "versicolor"))
```

---

#### `h2o.max`

*Returns the maxima of the input values.*

**Description**

Returns the maxima of the input values.

**Usage**

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

**Arguments**

- `x`: An H2OFrame object.
- `na.rm`: logical. indicating whether missing values should be removed.

**See Also**

- `max` for the base R implementation.

---

#### `h2o.mean`

*Compute the frame's mean by-column (or by-row).*

**Description**

Compute the frame’s mean by-column (or by-row).

**Usage**

```
h2o.mean(x, na.rm = FALSE, axis = 0, return_frame = FALSE, ...)
```

```r
## S3 method for class 'H2OFrame'
mean(x, na.rm = FALSE, axis = 0, return_frame = FALSE, ...
```
Arguments

- **x**: An H2OFrame object.
- **na.rm**: logical. Indicate whether missing values should be removed.
- **axis**: integer. Indicate whether to calculate the mean down a column (0) or across a row (1). NOTE: This is only applied when return_frame is set to TRUE. Otherwise, this parameter is ignored.
- **return_frame**: logical. Indicate whether to return an H2O frame or a list. Default is FALSE (returns a list).
- **...**: Further arguments to be passed from or to other methods.

Value

Returns a list containing the mean for each column (NaN for non-numeric columns) if return_frame is set to FALSE. If return_frame is set to TRUE, then it will return an H2O frame with means per column or row (depends on axis argument).

See Also

- `mean`, `rowMeans`, or `colMeans` for the base R implementation

Examples

```r
h2o.init()
prospath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
# Default behavior. Will return list of means per column.
h2o.mean(prostate.hex$AGE)
# return_frame set to TRUE. This will return an H2O Frame
# with mean per row or column (depends on axis argument)
h2o.mean(prostate.hex, na.rm=TRUE, axis=1, return_frame=TRUE)
```

---

**h2o.mean_per_class_error**

*Retrieve the mean per class error*

Description

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

Usage

```r
h2o.mean_per_class_error(object, train = FALSE, valid = FALSE, xval = FALSE)
```
Arguments

object: An H2OBinomialMetrics object.

train: Retrieve the training mean per class error
valid: Retrieve the validation mean per class error
xval: Retrieve the cross-validation mean per class error

See Also

h2o.mse for MSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples

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

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

hex[,]2 <- as.factor(hex[,]2)
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.mean_per_class_error(perf)
h2o.mean_per_class_error(model, train=TRUE)
```

### h2o.mean_residual_deviance

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

**Usage**

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

**Arguments**

object: An H2OModel object.

train: Retrieve the training Mean Residual Deviance
valid: Retrieve the validation Mean Residual Deviance
xval: Retrieve the cross-validation Mean Residual Deviance
**h2o.median**

**Description**

Compute the median of an H2OFrame.

**Usage**

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

### S3 method for class 'H2OFrame'

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

**Arguments**

- **x**
  - An H2OFrame object.

- **na.rm**
  - a logical, indicating whether na’s are omitted.

**Value**

Returns a list containing the median for each column (NaN for non-numeric columns)

**Examples**

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

Merge Two H2O Data Frames

Description

Merges two H2OFrame objects with the same arguments and meanings as merge() in base R. However, we do not support all=TRUE, all.x=TRUE and all.y=TRUE. The default method is auto and it will default to the radix method. The radix method will return the correct merge result regardless of duplicated rows in the right frame. In addition, the radix method can perform merge even if you have string columns in your frames. If there are duplicated rows in your right frame, they will not be included if you use the hash method. The hash method cannot perform merge if you have string columns in your left frame. Hence, we consider the radix method superior to the hash method and is the default method to use.

Usage

h2o.merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by,
           all = FALSE, all.x = all, all.y = all, method = "auto")

Arguments

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

Examples

h2o.init()
left <- data.frame(fruit = c('apple', 'orange', 'banana', 'lemon', 'strawberry', 'blueberry'),
                   color = c('red', 'orange', 'yellow', 'yellow', 'red', 'blue'))
right <- data.frame(fruit = c('apple', 'orange', 'banana', 'lemon', 'strawberry', 'watermelon'),
                    citrus = c(FALSE, TRUE, FALSE, TRUE, FALSE, FALSE))
l.hex <- as.h2o(left)
r.hex <- as.h2o(right)
left.hex <- h2o.merge(l.hex, r.hex, all.x = TRUE)
**H2O Model Metric Accessor Functions**

**Description**
A series of functions that retrieve model metric details.

**Usage**

```r
h2o.metric(object, thresholds, metric)
```

```r
h2o.F0point5(object, thresholds)
```

```r
h2o.F1(object, thresholds)
```

```r
h2o.F2(object, thresholds)
```

```r
h2o.accuracy(object, thresholds)
```

```r
h2o.error(object, thresholds)
```

```r
h2o.maxPerClassError(object, thresholds)
```

```r
h2o.mean_per_class_accuracy(object, thresholds)
```

```r
h2o.mcc(object, thresholds)
```

```r
h2o.precision(object, thresholds)
```

```r
h2o.tpr(object, thresholds)
```

```r
h2o.fpr(object, thresholds)
```

```r
h2o.fnr(object, thresholds)
```

```r
h2o.tnr(object, thresholds)
```

```r
h2o.recall(object, thresholds)
```

```r
h2o.sensitivity(object, thresholds)
```

```r
h2o.fallout(object, thresholds)
```

```r
h2o.missrate(object, thresholds)
```

```r
h2o.specificity(object, thresholds)
```
Arguments

object
  An H2OModelMetrics object of the correct type.

thresholds
  (Optional) A value or a list of values between 0.0 and 1.0.

metric
  (Optional) A specified parameter to retrieve.

Details

Many of these functions have an optional thresholds parameter. Currently only increments of 0.1 are allowed. If not specified, the functions will return all possible values. Otherwise, the function will return the value for the indicated threshold.

Currently, these functions are only supported by H2OBinomialMetrics objects.

Value

Returns either a single value, or a list of values.

See Also

h2o.auc for AUC, h2o.giniCoef for the GINI coefficient, and h2o.mse for MSE. See h2o.performance for creating H2OModelMetrics objects.

Examples

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

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

hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
perf <- h2o.performance(model, hex)
h2o.F1(perf)
```

h2o.min

Returns the minima of the input values.

Description

Returns the minima of the input values.

Usage

h2o.min(x, na.rm = FALSE)
**h2o.mktie**

**Arguments**
- *x*  
  An H2OFrame object.
- *na.rm*  
  logical, indicating whether missing values should be removed.

**See Also**
- *min* for the base R implementation.

---

**h2o.mktie**  
*Compute msec since the Unix Epoch*

**Description**
Compute msec since the Unix Epoch

**Usage**

\[
\text{h2o.mktie(year = 1970, month = 0, day = 0, hour = 0, minute = 0, second = 0, msec = 0)}
\]

**Arguments**

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

---

**h2o.mojo_predict_csv**  
*H2O Prediction from R without having H2O running*

**Description**
Provides the method *h2o.mojo_predict_csv* with which you can predict a MOJO model from R.

**Usage**

\[
\text{h2o.mojo_predict_csv(input_csv_path, mojo_zip_path, output_csv_path = NULL, genmodel_jar_path = NULL, classpath = NULL, java_options = NULL, verbose = F)}
\]
h2o.mojo_predict_df

Arguments

input_csv_path Path to input CSV file.
mojo_zip_path Path to MOJO zip downloaded from H2O.
output_csv_path Optional, path to the output CSV file with computed predictions. If NULL (default), then predictions will be saved as prediction.csv in the same folder as the MOJO zip.
genmodel_jar_path Optional, path to genmodel jar file. If NULL (default) then the h2o-genmodel.jar in the same folder as the MOJO zip will be used.
classpath Optional, specifies custom user defined classpath which will be used when scoring. If NULL (default) then the default classpath for this MOJO model will be used.
java_options Optional, custom user defined options for Java. By default `-Xmx4g -XX:ReservedCodeCacheSize=256m` is used.
verbose Optional, if TRUE, then additional debug information will be printed. FALSE by default.

Value

Returns a data.frame containing computed predictions

h2o.mojo_predict_df  H2O Prediction from R without having H2O running

Description

Provides the method h2o.mojo_predict_df with which you can predict a MOJO model from R.

Usage

h2o.mojo_predict_df(frame, mojo_zip_path, genmodel_jar_path = NULL,
classpath = NULL, java_options = NULL, verbose = F)

Arguments

frame data.frame to score.
mojo_zip_path Path to MOJO zip downloaded from H2O.
genmodel_jar_path Optional, path to genmodel jar file. If NULL (default) then the h2o-genmodel.jar in the same folder as the MOJO zip will be used.
classpath Optional, specifies custom user defined classpath which will be used when scoring. If NULL (default) then the default classpath for this MOJO model will be used.
**h2o.month**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java_options</td>
<td>Optional, custom user defined options for Java. By default <code>-Xmx4g -XX:ReservedCodeCacheSize=256m</code> is used.</td>
</tr>
<tr>
<td>verbose</td>
<td>Optional, if TRUE, then additional debug information will be printed. FALSE by default.</td>
</tr>
</tbody>
</table>

**Value**

Returns a data.frame containing computed predictions

---

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

**Description**

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

**Usage**

```r
h2o.month(x)
month(x)
```

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

**Arguments**

- **x** An H2OFrame object.

**Value**

An H2OFrame object containing the entries of `x` converted to months of the year.

**See Also**

- h2o.year
h2o.mse

Retrieves Mean Squared Error Value

Description
Retrieves the mean squared error value from an H2OModelMetrics object. If "train", "valid", and "xval" parameters are FALSE (default), then the training MSE value is returned. If more than one parameter is set to TRUE, then a named vector of MSEs are returned, where the names are "train", "valid" or "xval".

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

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

Details
This function only supports H2OBinomialMetrics, H2OMultinomialMetrics, and H2ORegressionMetrics objects.

See Also
h2o.auc for AUC, h2o.mse for MSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples
library(h2o)
h2o.init()

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

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

**Count of NAs per column**

**Description**

Gives the count of NAs per column.

**Usage**

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

**Arguments**

- `x` An H2OFrame object.

**Value**

Returns a list containing the count of NAs per column

**Examples**

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

---

h2o.naiveBayes

**Compute naïve Bayes probabilities on an H2O dataset.**

**Description**

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

h2o.naiveBayes(x, y, training_frame, model_id = NULL, nfolds = 0, seed = -1, fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"), fold_column = NULL, keep_cross_validation_predictions = FALSE, keep_cross_validation_fold_assignment = FALSE, validation_frame = NULL, ignore_const_cols = TRUE, score_each_iteration = FALSE, balance_classes = FALSE, class_sampling_factors = NULL, max_after_balance_size = 5, max_hit_ratio_k = 0, laplace = 0, threshold = 0.001, min_sdev = 0.001, eps = 0, eps_sdev = 0, min_prob = 0.001, eps_prob = 0, compute_metrics = TRUE, max_runtime_secs = 0)

Arguments

x (Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

y The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

training_frame Id of the training data frame.

model_id Destination id for this model; auto-generated if not specified.

nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

seed Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Defaults to -1 (time-based random number).

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

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

keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

validation_frame Id of the validation data frame.

ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.

score_each_iteration Logical. Whether to score during each iteration of model training. Defaults to FALSE.
balance_classes
Logical. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.

class_sampling_factors
Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

max_after_balance_size
Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

max_hit_ratio_k
Max. number (top K) of predictions to use for hit ratio computation (for multi-class only, 0 to disable) Defaults to 0.
laplace
Laplace smoothing parameter Defaults to 0.
threshold
This argument is deprecated, use 'min_sdev' instead. The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.

min_sdev
The minimum standard deviation to use for observations without enough data. Must be at least 1e-10.

eps
This argument is deprecated, use 'eps_sdev' instead. A threshold cutoff to deal with numeric instability, must be positive.

eps_sdev
A threshold cutoff to deal with numeric instability, must be positive.

min_prob
Min. probability to use for observations with not enough data.

eps_prob
Cutoff below which probability is replaced with min_prob.

compute_metrics
Logical. Compute metrics on training data Defaults to TRUE.

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

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
Returns an object of class H2OBinomialModel if the response has two categorical levels, and H2OMultinomialModel otherwise.

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

### h2o.names

**Column names of an H2OFrame**

**Description**

Column names of an H2OFrame

**Usage**

```
h2o.names(x)
```

**Arguments**

- `x` An H2OFrame object.

**See Also**

- `names` for the base R implementation.

### h2o.na_omit

**Remove Rows With NAs**

**Description**

Remove Rows With NAs

**Usage**

```
h2o.na_omit(object, ...)
```

**Arguments**

- `object` H2OFrame object
- `...` Ignored

**Value**

Returns an H2OFrame object containing non-NA rows.
h2o.nchar  

**String length**

**Description**

String length

**Usage**

h2o.nchar(x)

**Arguments**

x  
The column whose string lengths will be returned.

**Examples**

```r
library(h2o)
h2o.init()
string_to_nchar <- as.h2o("r tutorial")
nchar_string <- h2o.nchar(string_to_nchar)
```

---

h2o.ncol  

**Return the number of columns present in x.**

**Description**

Return the number of columns present in x.

**Usage**

h2o.ncol(x)

**Arguments**

x  
An H2OFrame object.

**See Also**

ncol for the base R implementation.
### h2o.networkTest
**View Network Traffic Speed**

**Description**
View speed with various file sizes.

**Usage**
```
h2o.networkTest()
```

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

---

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

**Description**
Get the number of factor levels for this frame.

**Usage**
```
h2o.nlevels(x)
```

**Arguments**
- `x` An H2OFrame object.

**See Also**
- `nlevels` for the base R method.

---

### h2o.no_progress
**Disable Progress Bar**

**Description**
Disable Progress Bar

**Usage**
```
h2o.no_progress()
```
**h2o.nrow**

Return the number of rows present in x.

Description

Return the number of rows present in x.

Usage

h2o.nrow(x)

Arguments

x An H2OFrame object.

See Also

nrow for the base R implementation.

---

**h2o.null_deviance**

Retrieve the null deviance

Description

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

Usage

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

Arguments

object An H2OModel or H2OModelMetrics

train Retrieve the training null deviance

valid Retrieve the validation null deviance

xval Retrieve the cross-validation null deviance
h2o.null_dof

Retrieve the null degrees of freedom

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

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

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

h2o.num_iterations

Retrieve the number of iterations.

Description
Retrieve the number of iterations.

Usage
h2o.num_iterations(object)

Arguments
- object: An H2OClusteringModel object.
- ... further arguments to be passed on (currently unimplemented)
h2o.num_valid_substrings

Count of substrings \( \geq 2 \) chars that are contained in file

Description

Find the count of all possible substrings \( \geq 2 \) chars that are contained in the specified line-separated text file.

Usage

\[
h2o.num_valid_substrings(x, \text{path})
\]

Arguments

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

h2o.openLog    View H2O R Logs

Description

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

Usage

\[
h2o.openLog(type)
\]

Arguments

- \( \text{type} \)    Currently unimplemented.

See Also

h2o.startLogging, h2o.stopLogging, h2o.clearLog
Examples

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

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

# Not run to avoid windows being opened during R CMD check
# h2o.openLog("Command")
# h2o.openLog("Error")

## End(Not run)
```

---

**h2o.parseRaw**  

**H2O Data Parsing**

**Description**

The second phase in the data ingestion step.

**Usage**

```r
h2o.parseRaw(data, pattern = "", destination_frame = "", header = NA,  
sep = "", col.names = NULL, col.types = NULL, na.strings = NULL,  
blocking = FALSE, parse_type = NULL, chunk_size = NULL,  
decrypt_tool = NULL)
```

**Arguments**

- **data**  
  An H2OFrame object to be parsed.

- **pattern**  
  (Optional) Character string containing a regular expression to match file(s) in the folder.

- **destination_frame**  
  (Optional) The hex key assigned to the parsed file.

- **header**  
  (Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.

- **sep**  
  (Optional) The field separator character. Values on each line of the file are separated by this character. If `sep = ""`, the parser will automatically detect the separator.

- **col.names**  
  (Optional) An H2OFrame object containing a single delimited line with the column names for the file.

- **col.types**  
  (Optional) A vector specifying the types to attempt to force over columns.

- **na.strings**  
  (Optional) H2O will interpret these strings as missing.
blocking  (Optional) Tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.

parse_type  (Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"

chunk_size  size of chunk of (input) data in bytes

decrypt_tool  (Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup).

Details

Parse the Raw Data produced by the import phase.

See Also

h2o.importFile, h2o.parseSetup

h2o.parseSetup  Get a parse setup back for the staged data.

Description

Get a parse setup back for the staged data.

Usage

h2o.parseSetup(data, pattern = "", destination_frame = "", header = NA,
sep = "", col.names = NULL, col.types = NULL, na.strings = NULL,
parse_type = NULL, chunk_size = NULL, decrypt_tool = NULL)

Arguments

data  An H2OFrame object to be parsed.

pattern  (Optional) Character string containing a regular expression to match file(s) in the folder.

destination_frame  (Optional) The hex key assigned to the parsed file.

header  (Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.

sep  (Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.

col.names  (Optional) An H2OFrame object containing a single delimited line with the column names for the file.

col.types  (Optional) A vector specifying the types to attempt to force over columns.

na.strings  (Optional) H2O will interpret these strings as missing.
parse_type   (Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"
chunk_size   size of chunk of (input) data in bytes
decrypt_tool (Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup).

See Also
  h2o.parseRaw

---

**h2o.partialPlot**  
*Partial Dependence Plots*

**Description**

Partial dependence plot gives a graphical depiction of the marginal effect of a variable on the response. The effect of a variable is measured in change in the mean response. Note: Unlike randomForest's partialPlot when plotting partial dependence the mean response (probabilities) is returned rather than the mean of the log class probability.

**Usage**

```r
h2o.partialPlot(object, data, cols, destination_key, nbins = 20, 
plot = TRUE, plot_stddev = TRUE)
```

**Arguments**

- **object**  
  An H2OModel object.
- **data**  
  An H2OFrame object used for scoring and constructing the plot.
- **cols**  
  Feature(s) for which partial dependence will be calculated.
- **destination_key**  
  An key reference to the created partial dependence tables in H2O.
- **nbins**  
  Number of bins used. For categorical columns make sure the number of bins exceed the level count.
- **plot**  
  A logical specifying whether to plot partial dependence table.
- **plot_stddev**  
  A logical specifying whether to add std err to partial dependence plot.

**Value**

Plot and list of calculated mean response tables for each feature requested.
Examples

library(h2o)
h2o.init()
prostate.path <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prostate.path, destination_frame = "prostate.hex")
prostate.hex[, "CAPSULE"] <- as.factor(prostate.hex[, "CAPSULE"])
prostate.hex[, "RACE"] <- as.factor(prostate.hex[, "RACE"])
prostate.gbm <- h2o.gbm(x = c("AGE", "RACE"),
                         y = "CAPSULE",
                         training_frame = prostate.hex,
                         ntrees = 10,
                         max_depth = 5,
                         learn_rate = 0.1)
h2o.partialPlot(object = prostate.gbm, data = prostate.hex, cols = c("AGE", "RACE"))

---

h2o.performance  Model Performance Metrics in H2O

Description

Given a trained h2o model, compute its performance on the given dataset. However, if the dataset does not contain the response/target column, no performance will be returned. Instead, a warning message will be printed.

Usage

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

Arguments

- **model**: An H2OModel object
- **newdata**: An H2OFrame. The model will make predictions on this dataset, and subsequently score them. The dataset should match the dataset that was used to train the model, in terms of column names, types, and dimensions. If newdata is passed in, then train, valid, and xval are ignored.
- **train**: A logical value indicating whether to return the training metrics (constructed during training).
- **valid**: A logical value indicating whether to return the validation metrics (constructed during training).
- **xval**: A logical value indicating whether to return the cross-validation metrics (constructed during training).
- **data** (DEPRECATED): An H2OFrame. This argument is now called ‘newdata’.
Value

Returns an object of the H2OModelMetrics subclass.

Examples

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

# If model uses balance_classes
# the results from train = TRUE will not match the results from newdata = prostate.hex
prostate.gbm.balanced <- h2o.gbm(3:9, "CAPSULE", prostate.hex, balance_classes = TRUE)
h2o.performance(model = prostate.gbm.balanced, newdata = prostate.hex)
h2o.performance(model = prostate.gbm.balanced, train = TRUE)
```

h2o.pivot  

**Pivot a frame**

Description

Pivot the frame designated by the three columns: index, column, and value. Index and column should be of type enum, int, or time. For cases of multiple indexes for a column label, the aggregation method is to pick the first occurrence in the data frame.

Usage

```r
h2o.pivot(x, index, column, value)
```

Arguments

- **x**: an H2OFrame
- **index**: the column where pivoted rows should be aligned on
- **column**: the column to pivot
- **value**: values of the pivoted table

Value

An H2OFrame with columns from the columns arg, aligned on the index arg, with values from values arg
h2o.prcomp

Principal component analysis of an H2O data frame

Description

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

Usage

h2o.prcomp(training_frame, x, model_id = NULL, validation_frame = NULL, ignore_const_cols = TRUE, score_each_iteration = FALSE, transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"), pca_method = c("GramSVD", "Power", "Randomized", "GLRM"), pca_impl = c("MTJ_EVD_DENSEMATRIX", "MTJ_EVD_SYMMATRIX", "MTJ_SVD_DENSEMATRIX", "JAMA"), k = 1, max_iterations = 1000, use_all_factor_levels = FALSE, compute_metrics = TRUE, impute_missing = FALSE, seed = -1, max_runtime_secs = 0)

Arguments

training_frame  Id of the training data frame.
x  A vector containing the character names of the predictors in the model.
model_id  Destination id for this model; auto-generated if not specified.
validation_frame  Id of the validation data frame.
ignore_const_cols  Logical. Ignore constant columns. Defaults to TRUE.
score_each_iteration  Logical. Whether to score during each iteration of model training. Defaults to FALSE.
transform  Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
pca_method  Specify the algorithm to use for computing the principal components: GramSVD - uses a distributed computation of the Gram matrix, followed by a local SVD; Power - computes the SVD using the power iteration method (experimental); Randomized - uses randomized subspace iteration method; GLRM - fits a generalized low-rank model with L2 loss function and no regularization and solves for the SVD using local matrix algebra (experimental) Must be one of: "GramSVD", "Power", "Randomized", "GLRM". Defaults to GramSVD.
pca_impl  Specify the implementation to use for computing PCA (via SVD or EVD): MTJ_EVD_DENSEMATRIX - eigenvalue decompositions for dense matrix using MTJ; MTJ_EVD_SYMMATRIX - eigenvalue decompositions for symmetric matrix using MTJ; MTJ_SVD_DENSEMATRIX - singular-value decompositions for dense matrix using MTJ; JAMA - eigenvalue decompositions for
MTJ - https://github.com/fommil/matrix-toolkits-java/ Must be one of: "MTJ_EVD_DENSEMATRIX", "MTJ_EVD_SYMMATRIX", "MTJ_SVD_DENSEMATRIX", "JAMA".

k  Rank of matrix approximation Defaults to 1.
max_iterations Maximum training iterations Defaults to 1000.
use_all_factor_levels Logical. Whether first factor level is included in each categorical expansion
Defaults to FALSE.
compute_metrics Logical. Whether to compute metrics on the training data Defaults to TRUE.
impute_missing Logical. Whether to impute missing entries with the column mean Defaults to
FALSE.
seed Seed for random numbers (affects certain parts of the algo that are stochastic
and those might or might not be enabled by default) Defaults to -1 (time-based
random number).
max_runtime_secs Maximum allowed runtime in seconds for model training. Use 0 to disable.
Defaults to 0.

Value

Returns an object of class H2ODimReductionModel.

References

N. Halko, P.G. Martinsson, J.A. Tropp. Finding structure with randomness: Probabilistic algorithms

See Also

h2o.svd, h2o.glm

Examples

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

**H2O Prediction from R without having H2O running**

**Description**

Provides the method h2o.predict with which you can predict a MOJO or POJO Jar model from R.

**Usage**

```r
h2o.predict_json(model, json, genmodelpath, labels, classpath, javaoptions)
```

**Arguments**

- **model**: String with file name of MOJO or POJO Jar
- **json**: JSON String with inputs to model
- **genmodelpath**: (Optional) path name to h2o-genmodel.jar, if not set defaults to same dir as MOJO
- **labels**: (Optional) if TRUE then show output labels in result
- **classpath**: (Optional) Extra items for the class path of where to look for Java classes, e.g., h2o-genmodel.jar
- **javaoptions**: (Optional) Java options string, default if "-Xmx4g"

**Value**

Returns an object with the prediction result

**Examples**

```r
library(h2o)
h2o.predict_json('~/GBM_model_python_1473313897851_6.zip', '{"C7":1}')
h2o.predict_json('~/GBM_model_python_1473313897851_6.zip', '{"C7":1}', c('.', "lib"))
```

---

h2o.print

**Print An H2OFrame**

**Description**

Print An H2OFrame

**Usage**

```r
h2o.print(x, n = 6L)
```
Arguments
- `x`: An H2OFrame object.
- `n`: An (Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).
- `...`: Further arguments to be passed from or to other methods.

**h2o.prod**

*Return the product of all the values present in its arguments.*

**Description**

Return the product of all the values present in its arguments.

**Usage**

`h2o.prod(x)`

**Arguments**

- `x`: An H2OFrame object.

**See Also**

- `prod` for the base R implementation.

**h2o.proj_archetypes**

*Convert Archetypes to Features from H2O GLRM Model*

**Description**

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

**Usage**

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

**Arguments**

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

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

See Also

`h2o.glrm` for making an H2ODimReductionModel.

Examples

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

---

### h2o.quantile

#### Quantiles of H2O Frames.

**Description**

Obtain and display quantiles for H2O parsed data.

**Usage**

```r
h2o.quantile(x, probs = c(0.001, 0.01, 0.1, 0.25, 0.333, 0.5, 0.667, 0.75,
                      0.9, 0.99, 0.999), combine_method = c("interpolate", "average", "avg",
                      "low", "high"), weights_column = NULL, ...)
```

```r
# S3 method for class 'H2OFrame'
quantile(x, probs = c(0.001, 0.01, 0.1, 0.25, 0.333, 0.5,
               0.667, 0.75, 0.9, 0.99, 0.999), combine_method = c("interpolate", "average",
               "avg", "low", "high"), weights_column = NULL, ...)
```

**Arguments**

- **x**: An H2OFrame object with a single numeric column.
- **probs**: Numeric vector of probabilities with values in [0,1].
- **combine_method**: How to combine quantiles for even sample sizes. Default is to do linear interpolation. E.g., If method is "lo", then it will take the lo value of the quantile. Abbreviations for average, low, and high are acceptable (avg, lo, hi).
weights_column (Optional) String name of the observation weights column in x or an H2OFrame object with a single numeric column of observation weights.

Further arguments passed to or from other methods.

Details


Value

A vector describing the percentiles at the given cutoffs for the H2OFrame object.

Examples

# Request quantiles for an H2O parsed data set:
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.uploadFile(path = prosPath)
# Request quantiles for a subset of columns in an H2O parsed data set
quantile(prostate.hex[,3])
for(i in 1:ncol(prostate.hex))
  quantile(prostate.hex[,i])

h2o.r2 Retrieve the R2 value

Description

Retrieves the R2 value from an H2O model. Will return R^2 for GLM Models and will return NaN otherwise. If "train", "valid", and "xval" parameters are FALSE (default), then the training R2 value is returned. If more than one parameter is set to TRUE, then a named vector of R2s are returned, where the names are "train", "valid" or "xval".

Usage

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

Arguments

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

library(h2o)

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

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

h2o.r2(m)

h2o.randomForest

Build a Random Forest model

Description

Builds a Random Forest model on an H2OFrame.

Usage

h2o.randomForest(x, y, training_frame, model_id = NULL,
    validation_frame = NULL, nfolds = 0,
    keep_cross_validation_predictions = FALSE,
    keep_cross_validation_fold_assignment = FALSE,
    score_each_iteration = FALSE, score_tree_interval = 0,
    fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
    fold_column = NULL, ignore_const_cols = TRUE, offset_column = NULL,
    weights_column = NULL, balance_classes = FALSE,
    class_sampling_factors = NULL, max_after_balance_size = 5,
    max_hit_ratio_k = 0, ntrees = 50, max_depth = 20, min_rows = 1,
    nbins = 20, nbins_top_level = 1024, nbins_cats = 1024,
    r2_stopping = Inf, stopping_rounds = 0, stopping_metric = c("AUTO",
    "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group",
    "misclassification", "mean_per_class_error"), stopping_tolerance = 0.001,
    max_runtime_secs = 0, seed = -1, build_tree_one_node = FALSE,
    mtries = -1, sample_rate = 0.6320000291, sample_rate_per_class = NULL,
    binomial_double_trees = FALSE, checkpoint = NULL,
    col_sample_rate_change_per_level = 1, col_sample_rate_per_tree = 1,
    min_split_improvement = 1e-05, histogram_type = c("AUTO",
    "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin"),
    categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
    "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
    calibrate_model = FALSE, calibration_frame = NULL,
    distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson",
    "gamma", "tweedie", "laplace", "quantile", "huber"),
    custom_metric_func = NULL, verbose = FALSE)
Arguments

x  (Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

y  The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

training_frame  Id of the training data frame.

model_id  Destination id for this model; auto-generated if not specified.

validation_frame  Id of the validation data frame.

nfolds  Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

keep_cross_validation_predictions  Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment  Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

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

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

ignore_const_cols  Logical. Ignore constant columns. Defaults to TRUE.

offset_column  Offset column. This argument is deprecated and has no use for Random Forest.

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

balance_classes  Logical. Balance training data class counts via over/under-sampling (for imbalanced data). Defaults to FALSE.
class_sampling_factors
Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.

max_after_balance_size
Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

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

ntrees
Number of trees. Defaults to 50.

max_depth
Maximum tree depth. Defaults to 20.

min_rows
Fewest allowed (weighted) observations in a leaf. Defaults to 1.

nbins
For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.

nbins_top_level
For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level Defaults to 1024.

nbins_cats
For categorical columns (factors), build a histogram of this many bins, then split at the best point. Higher values can lead to more overfitting. Defaults to 1024.

r2_stopping
r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this. Defaults to 1.797693135e+308.

stopping_rounds
Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable) Defaults to 0.

stopping_metric
Metric to use for early stopping (AUTO: logloss for classification, deviance for regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error". Defaults to AUTO.

stopping_tolerance
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

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

seed
Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default) Defaults to -1 (time-based random number).

build_tree_one_node
Logical. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
mtries
Number of variables randomly sampled as candidates at each split. If set to -1, defaults to \( \sqrt{p} \) for classification and \( p/3 \) for regression (where \( p \) is the # of predictors Defaults to -1.

sample_rate
Row sample rate per tree (from 0.0 to 1.0) Defaults to 0.6320000291.

sample_rate_per_class
A list of row sample rates per class (relative fraction for each class, from 0.0 to 1.0), for each tree

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

checkpoint
Model checkpoint to resume training with.

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

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

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

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

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

calibrate_model
Logical. Use Platt Scaling to calculate calibrated class probabilities. Calibration can provide more accurate estimates of class probabilities. Defaults to FALSE.

calibration_frame
Calibration frame for Platt Scaling

distribution
Distribution. This argument is deprecated and has no use for Random Forest.

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

verbose
Logical. Print scoring history to the console (Metrics per tree for GBM, DRF, & XGBoost. Metrics per epoch for Deep Learning). Defaults to FALSE.

Value
Creates a H2OModel object of the right type.

See Also
predict.H2OModel for prediction
h2o.range

Returns a vector containing the minimum and maximum of all the
given arguments.

Description

Returns a vector containing the minimum and maximum of all the given arguments.

Usage

h2o.range(x, na.rm = FALSE, finite = FALSE)

Arguments

x     An H2OFrame object.
na.rm  logical. indicating whether missing values should be removed.
finite logical. indicating if all non-finite elements should be omitted.

See Also

range for the base R implementation.

h2o.rank_within_group_by

This function will add a new column rank where the ranking is pro-
duced as follows: 1. sorts the H2OFrame by columns sorted in by
columns specified in group_by_cols and sort_cols in the directions
specified by the ascending for the sort_cols. The sort directions for
the group_by_cols are ascending only. 2. A new rank column is added
to the frame which will contain a rank assignment performed next.
The user can choose to assign a name to this new column. The default
name is New_Rank_column. 3. For each groupby groups, a rank is
assigned to the row starting from 1, 2, ... to the end of that group. 4. If
sort_cols_sorted is TRUE, a final sort on the frame will be performed
frame according to the sort_cols and the sort directions in ascending.
If sort_cols_sorted is FALSE (by default), the frame from step 3 will
be returned as is with no extra sort. This may provide a small speedup
if desired.
**Description**

This function will add a new column rank where the ranking is produced as follows: 1. sorts the H2OFrame by columns sorted in by columns specified in `group_by_cols` and `sort_cols` in the directions specified by the ascending for the `sort_cols`. The sort directions for the `group_by_cols` are ascending only. 2. A new rank column is added to the frame which will contain a rank assignment performed next. The user can choose to assign a name to this new column. The default name is `New_Rank_column`. 3. For each groupby groups, a rank is assigned to the row starting from 1, 2, ... to the end of that group. 4. If `sort_cols_sorted` is TRUE, a final sort on the frame will be performed according to the `sort_cols` and the sort directions in ascending. If `sort_cols_sorted` is FALSE (by default), the frame from step 3 will be returned as is with no extra sort. This may provide a small speedup if desired.

**Usage**

```r
h2o.rank_within_group_by(x, group_by_cols, sort_cols, ascending = NULL,
  new_col_name = "New_Rank_column", sort_cols_sorted = FALSE)
```

**Arguments**

- `x` The H2OFrame input to be sorted.
- `group_by_cols` a list of column names or indices to form the groupby groups
- `sort_cols` a list of column names or indices for sorting
- `ascending` a list of Boolean to determine if ascending sort (set to TRUE) is needed for each column in `sort_cols` (optional). Default is ascending sort for all. To perform descending sort, set value to FALSE
- `new_col_name` new column name for the newly added rank column if specified (optional). Default name is `New_Rank_column`.
- `sort_cols_sorted` Boolean to determine if the final returned frame is to be sorted according to the `sort_cols` and sort directions in ascending. Default is FALSE.

The following example is generated by Nidhi Mehta.

If the input frame is `train`:

```r
ID Group_by_column num data Column_to_arrange_by num_1 fdata 12 1 2941.552 1 3 -3177.9077 1 12 1 2941.552 1 5 -13311.8247 1 12 2 -22722.174 1 3 -3177.9077 1 12 2 -22722.174 1 5 -13311.8247 1 12 3 -12776.884 1 4 28080.1607 0 13 1 -6049.830 1 5 -18421.6171 0 13 1 -6049.830 1 4 28080.1607 0 15 3 -16995.346 1 1 -9781.6373 0 16 1 -10003.593 0 3 -61284.6900 0 16 3 26052.495 1 3 -61284.6900 0 16 3 -22905.288 0 3 -61284.6900 0 17 2 -13465.496 1 2 12094.4851 1 17 2 -13465.496 1 2 12094.4851 1 17 2 -13465.496 1 3 -415.1114 0 17 2 -3329.619 1 12 1 -6049.830 1 3 -3177.9077 1 1 16 1 -10003.593 0 3 -61284.6900 0 2 13 1 -6049.830
```

If the following commands are issued: `rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE))` `h2o.summary(rankedF1)`

The returned frame `rankedF1` will look like this:

```r
ID Group_by_column num data Column_to_arrange_by num_1 fdata.1 New_Rank_column 12 1 2941.552 1 3 -3177.9077 1 1 16 1 -10003.593 0 3 -61284.6900 0 2 13 1 -6049.830
```
If the following commands are issued: rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE), sort_cols_sorted=TRUE) h2o.summary(rankedF1)

The returned frame will be sorted according to sortCols and hence look like this instead: ID Group_by_column num fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column

h2o.rbind

**Combine H2O Datasets by Rows**

**Description**

Takes a sequence of H2O data sets and combines them by rows

**Usage**

h2o.rbind(...)  

**Arguments**

...  

A sequence of H2OFrame arguments. All datasets must exist on the same H2O instance (IP and port) and contain the same number and types of columns.

**Value**

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

**See Also**

rbind for the base R method.
**Examples**

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

---

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

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**See Also**

`h2o.glrm` for making an `H2ODimReductionModel`. 
**h2o.relevel**

Reorders levels of an H2O factor, similarly to standard R’s `relevel`.

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

new reordered factor column

**Examples**

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

# Convert iris dataset to an H2OFrame
hf <- as.h2o(iris)
# Look at current ordering of the Species column levels
h2o.levels(hf["Species"])
# "setosa" "versicolor" "virginica"
# Change the reference level to "virginica"
hf["Species"] <- h2o.relevel(x = hf["Species"], y = "virginica")
# Observe new ordering
h2o.levels(hf["Species"])
# "virginica" "setosa" "versicolor"
```
h2o.removeAll

Remove All Objects on the H2O Cluster

**Description**

Removes the data from the h2o cluster, but does not remove the local references.

**Usage**

```
h2o.removeAll(timeout_secs = 0)
```

**Arguments**

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

**See Also**

`h2o.rm`

**Examples**

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

h2o.removeVecs

Delete Columns from an H2OFrame

**Description**

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

**Usage**

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

**Arguments**

- `data` The H2OFrame.
- `cols` The columns to remove.
**h2o.rep_len**  
*Replicate Elements of Vectors or Lists into H2O*

**Description**

*h2o.rep_len* performs just as *rep* does. It replicates the values in *x* in the H2O backend.

**Usage**

```r
h2o.rep_len(x, length.out)
```

**Arguments**

- **x**: an H2O frame
- **length.out**: non negative integer. The desired length of the output vector.

**Value**

Creates an H2OFrame of the same type as *x*

---

**h2o.residual_deviance**  
*Retrieve the residual deviance*

**Description**

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

**Usage**

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

**Arguments**

- **object**: An H2OModel or H2OModelMetrics
- **train**: Retrieve the training residual deviance
- **valid**: Retrieve the validation residual deviance
- **xval**: Retrieve the cross-validation residual deviance
h2o.residual_dof  

Retrieves the residual degrees of freedom

Description

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

Usage

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

Arguments

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

h2o.rm  

Delete Objects In H2O

Description

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

Usage

h2o.rm(ids)

Arguments

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

See Also

h2o.assign, h2o.ls
Description

Retrieves the root mean squared error value from an H2OModelMetrics object. If "train", "valid", and "xval" parameters are FALSE (default), then the training RMSE value is returned. If more than one parameter is set to TRUE, then a named vector of RMSEs are returned, where the names are "train", "valid" or "xval".

Usage

h2o.rmse(object, train = FALSE, valid = FALSE, xval = FALSE)

Arguments

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

Details

This function only supports H2OBinomialMetrics, H2OMultinomialMetrics, and H2ORegressionMetrics objects.

See Also

h2o.auc for AUC, h2o.mse for RMSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples

library(h2o)
h2o.init()

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

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

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

Description

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

Usage

h2o.round(x, digits = 0)

round(x, digits = 0)
h2o.rstrip

Arguments

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

See Also

round for the base R implementation.

---

h2o.rstrip Strip set from right

Description

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

Usage

h2o.rstrip(x, set = " ")

Arguments

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

Examples

library(h2o)
h2o.init()
string_to_rstrip <- as.h2o("1234567890")
rstrip_string <- h2o.rstrip(string_to_rstrip,"890") #Remove "890"
**h2o.runif**  
*Produce a Vector of Random Uniform Numbers*

**Description**

Creates a vector of random uniform numbers equal in length to the length of the specified H2O dataset.

**Usage**

```r
h2o.runif(x, seed = -1)
```

**Arguments**

- `x`  
  An H2OFrame object.
- `seed`  
  A random seed used to generate draws from the uniform distribution.

**Value**

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

**Examples**

```r
library(h2o)
h2o.init()
prospath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.importFile(path = prosPath, destination_frame = "prostate.hex")
s <- h2o.runif(prostate.hex)
summary(s)

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

**h2o.saveModel**  
*Save an H2O Model Object to Disk*

**Description**

Save an `H2OModel` to disk. (Note that ensemble binary models can be saved.)
Usage

h2o.saveModelDetails(object, path = "", force = FALSE)

Arguments

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

Details

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

See Also

h2o.loadModel for loading a model to H2O from disk

Examples

```r
# Not run:
# library(h2o)
# h2o.init()
# prostate.hex <- h2o.importFile(path = paste("https://raw.github.com",
# "h20ai/h2o-2/master/smalldata/logreg/prostate.csv", sep = "/"),
# "prostate.hex")
# prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
# training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# h2o.saveModelDetails(object = prostate.glm, path = "/Users/UserName/Desktop", force=TRUE)
```

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

Model Details will download as a JSON file. In the case of existing files force = TRUE will overwrite the file. Otherwise, the operation will fail.

Examples

```
## Not run:
# library(h2o)
# h2o.init()
# prostate.hex <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#   training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# h2o.saveModelDetails(object = prostate.glm, path = "/Users/UserName/Desktop", force=TRUE)

## End(Not run)
```

---

table of function 'h2o.saveMojo'

h2o.saveMojo  
Save an H2O Model Object as Mojo to Disk

Description

Save an MOJO (Model Object, Optimized) to disk.

Usage

```
h2o.saveMojo(object, path = "", force = FALSE)
```

Arguments

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

Details

MOJO will download as a zip file. In the case of existing files force = TRUE will overwrite the file. Otherwise, the operation will fail.

See Also

- `h2o.saveModel` for saving a model to disk as a binary object.
Examples

```r
## Not run:
# library(h2o)
# h2o.init()
# prostate.hex <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#     training_frame = prostate.hex, family = "binomial", alpha = 0.5)
# h2o.saveMojo(object = prostate.glm, path = "/Users/UserName/Desktop", force=TRUE)

## End(Not run)
```

---

**h2o.scale**

### Scaling and Centering of an H2OFrame

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

#### Usage

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

```
## S3 method for class 'H2OFrame'
scale(x, center = TRUE, scale = TRUE)
```

#### Arguments

- **x**: An H2OFrame object.
- **center**: either a logical value or numeric vector of length equal to the number of columns of x.
- **scale**: either a logical value or numeric vector of length equal to the number of columns of x.

#### Examples

```r
library(h2o)
h2o.init()
irisPath <- system.file("extdata", "iris_wheader.csv", package="h2o")
iris.hex <- h2o.uploadFile(path = irisPath, destination_frame = "iris.hex")
summary(iris.hex)

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

Description

Retrieve Model Score History

Usage

h2o.scoreHistory(object)

Arguments

object  
An H2OModel object.

h2o.sd  
Standard Deviation of a column of data.

Description

Obtain the standard deviation of a column of data.

Usage

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

sd(x, na.rm = FALSE)

Arguments

x  
An H2OFrame object.

na.rm  
logical. Should missing values be removed?

See Also

h2o.var for variance, and sd for the base R implementation.

Examples

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

*Retrieve the standard deviations of principal components*

**Description**
Retrieve the standard deviations of principal components

**Usage**

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

**Arguments**
- `object`: An `H2ODimReductionModel` object.

---

**h2o.setLevels**

*Set Levels of H2O Factor Column*

**Description**
Works on a single categorical vector. New domains must be aligned with the old domains. This call has SIDE EFFECTS and mutates the column in place (change of the levels will also affect all the frames that are referencing this column). If you want to make a copy of the column instead, use parameter `in.place = FALSE`.

**Usage**

```r
h2o.setLevels(x, levels, in.place = TRUE)
```

**Arguments**
- `x`: A single categorical column.
- `levels`: A character vector specifying the new levels. The number of new levels must match the number of old levels.
- `in.place`: Indicates whether new domain will be directly applied to the column (in place change) or if a copy of the column will be created with the given domain levels.

**Examples**

```r
h2o.init()
iris.hex <- as.h2o(iris)
new.levels <- c("setosa", "versicolor", "caroliniana")
iris.hex$Species <- h2o.setLevels(iris.hex$Species, new.levels, in.place = FALSE)
h2o.levels(iris.hex$Species)
```
**h2o.set_timezone**

*Set the Time Zone on the H2O Cloud*

**Description**

Set the Time Zone on the H2O Cloud

**Usage**

```
h2o.set_timezone(tz)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tz</td>
<td>The desired timezone.</td>
</tr>
</tbody>
</table>

**h2o.show_progress**

*Enable Progress Bar*

**Description**

Enable Progress Bar

**Usage**

```
h2o.show_progress()
```

**h2o.shutdown**

*Shut Down H2O Instance*

**Description**

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

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompt</td>
<td>A logical value indicating whether to prompt the user before shutting down the H2O server.</td>
</tr>
</tbody>
</table>
h2o.signif

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)
h2o.init()
h2o.shutdown()

## End(Not run)

---

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

Description

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

Usage

h2o.signif(x, digits = 6)

signif(x, digits = 6)

Arguments

x  An H2OFrame object.

digits  Number of significant digits to round doubles/floats.
h2o.sin  

*Compute the sine of x*

**Description**

Compute the sine of x

**Usage**

h2o.sin(x)

**Arguments**

- **x**: An H2OFrame object.

**See Also**

sin for the base R implementation.

h2o.skewness  

*Skewness of a column*

**Description**

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

**Usage**

h2o.skewness(x, ..., na.rm = TRUE)

skewness.H2OFrame(x, ..., na.rm = TRUE)

**Arguments**

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

**Value**

Returns a list containing the skewness for each column (NaN for non-numeric columns).
h2o.splitFrame

Examples

h2o.init()
prosPath <- system.file(“extdata”, “prostate.csv”, package = “h2o”)
prostate.hex <- h2o.uploadFile(path = prosPath)
h2o.skewness(prostate.hex$AGE)

h2o.splitFrame

Split an H2O Data Set

Description

Split an existing H2O data set according to user-specified ratios. The number of subsets is always 1 more than the number of given ratios. Note that this does not give an exact split. H2O is designed to be efficient on big data using a probabilistic splitting method rather than an exact split. For example, when specifying a split of 0.75/0.25, H2O will produce a test/train split with an expected value of 0.75/0.25 rather than exactly 0.75/0.25. On small datasets, the sizes of the resulting splits will deviate from the expected value more than on big data, where they will be very close to exact.

Usage

h2o.splitFrame(data, ratios = 0.75, destination_frames, seed = -1)

Arguments

data An H2OFrame object representing the data set to split.
ratios A numeric value or array indicating the ratio of total rows contained in each split. Must total up to less than 1.
destination_frames An array of frame IDs equal to the number of ratios specified plus one.
seed Random seed.

Value

Returns a list of split H2OFrame’s

Examples

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

*Compute the square root of* \( x \)

**Description**

Compute the square root of \( x \)

**Usage**

\[ h2o.sqrt(x) \]

**Arguments**

\( x \)  
An H2OFrame object.

**See Also**

\( \text{sqrt} \) for the base R implementation.

---

h2o.stackedEnsemble  

*Builds a Stacked Ensemble*

**Description**

Build a stacked ensemble (aka. Super Learner) using the H2O base learning algorithms specified by the user.

**Usage**

\[ h2o.stackedEnsemble(x, y, training_frame, model_id = NULL, validation_frame = NULL, base_models = list(), metalearn_algorithm = c("AUTO", "glm", "gbm", "drf", "deeplearning"), metalearn_nfolds = 0, metalearn_fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"), metalearn_fold_column = NULL, keep_levelone_frame = FALSE, seed = -1, metalearn_params = NULL) \]

**Arguments**

\( x \)  
(Optional). A vector containing the names or indices of the predictor variables to use in building the model. If \( x \) is missing, then all columns except \( y \) are used. Training frame is used only to compute ensemble training metrics.

\( y \)  
The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.
training_frame  Id of the training data frame.
model_id        Destination id for this model; auto-generated if not specified.
validation_frame Id of the validation data frame.
base_models     List of models (or model ids) to ensemble/stack together. Models must have been cross-validated using nfolds > 1, and folds must be identical across models. Defaults to [].
metalearner_algorithm Type of algorithm to use as the metalearner. Options include 'AUTO' (GLM with non negative weights; if validation_frame is present, a lambda search is performed), 'glm' (GLM with default parameters), 'gbm' (GBM with default parameters), 'drf' (Random Forest with default parameters), or 'deeplearning' (Deep Learning with default parameters). Must be one of: "AUTO", "glm", "gbm", "drf", "deeplearning". Defaults to AUTO.
metalearner_nfolds Number of folds for K-fold cross-validation of the metalearner algorithm (0 to disable or >= 2). Defaults to 0.
metalearner_fold_assignment Cross-validation fold assignment scheme for metalearner cross-validation. Defaults to AUTO (which is currently set to Random). The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified".
metalearner_fold_column Column with cross-validation fold index assignment per observation for cross-validation of the metalearner.
keep_levelone_frame Logical. Keep level one frame used for metalearner training. Defaults to FALSE.
seed             Seed for random numbers; passed through to the metalearner algorithm. Defaults to -1 (time-based random number). Defaults to -1 (time-based random number).
metalearner_params Parameters for metalearner algorithm Defaults to NULL.

Examples

# See example R code here:
# http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html
h2o.startLogging  Start Writing H2O R Logs

Description

Begin logging H2o R POST commands and error responses to local disk. Used primarily for de-buggin purposes.

Usage

h2o.startLogging(file)

Arguments

file  a character string name for the file, automatically generated

See Also

h2o.stopLogging, h2o.clearLog, h2o.openLog

Examples

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

h2o.std_coef_plot  Plot Standardized Coefficient Magnitudes

Description

Plot a GLM model’s standardized coefficient magnitudes.

Usage

h2o.std_coef_plot(model, num_of_features = NULL)

Arguments

model  A trained generalized linear model
num_of_features  The number of features to be shown in the plot
h2o.stopLogging

See Also

h2o.varimp.plot for variable importances plot of random forest, GBM, deep learning.

Examples

library(h2o)
h2o.init()

prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.importFile(prosPath)
prostate.hex[,2] <- as.factor(prostate.hex[,2])
prostate.glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
training_frame = prostate.hex, family = "binomial",
    nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.std_coef_plot(prostate.glm)

h2o.stopLogging

Stop Writing H2O R Logs

Description

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

Usage

h2o.stopLogging()

See Also

h2o.startLogging, h2o.clearLog, h2o.openLog

Examples

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

Display the structure of an H2OFrame object

**Description**

Display the structure of an H2OFrame object

**Usage**

h2o.str(object, ..., cols = FALSE)

**Arguments**

- **object**
  
  An H2OFrame.

- **...**
  
  Further arguments to be passed from or to other methods.

- **cols**
  
  Print the per-column str for the H2OFrame

---

**h2o.stringdist**

Compute element-wise string distances between two H2OFrames

**Description**

Compute element-wise string distances between two H2OFrames. Both frames need to have the same shape (N x M) and only contain string/factor columns. Return a matrix (H2OFrame) of shape N x M.

**Usage**

h2o.stringdist(x, y, method = c("lv", "lcs", "qgram", "jaccard", "jw", "soundex"), compare_empty = TRUE)

**Arguments**

- **x**
  
  An H2OFrame

- **y**
  
  A comparison H2OFrame

- **method**
  
  A string identifier indicating what string distance measure to use. Must be one of: "lv" - Levenshtein distance "lcs" - Longest common substring distance "qgram" - q-gram distance "jaccard" - Jaccard distance between q-gram profiles "jw" - Jaro, or Jaro-Winker distance "soundex" - Distance based on soundex encoding

- **compare_empty**
  
  if set to FALSE, empty strings will be handled as NaNs
h2o.strsplit

**Examples**

```r
h2o.init()
x <- as.h2o(c("Martha", "Dwayne", "Dixon"))
y <- as.character(as.h2o(c("Marhta", "Duane", "Dicksonx")))
h2o.stringdist(x, y, method = "jw")
```

---

**h2o.strsplit**  
*String Split*

**Description**

String Split

**Usage**

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

**Arguments**

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

**Value**

An H2OFrame where each column is the outcome of the string split.

**Examples**

```r
library(h2o)
h2o.init()
string_to_split <- as.h2o("Split at every character.")
split_string <- h2o.strsplit(string_to_split,"")
```
h2o.sub  

**String Substitute**

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

```r
library(h2o)
h2o.init()
string_to_sub <- as.h2o("r tutorial")
sub_string <- h2o.sub("r \,"H2O \,"",string_to_sub)
```

---

h2o.substring  

**Substring**

**Description**

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

**Usage**

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

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

Examples

library(h2o)
h2o.init()
string_to_substring <- as.h2o("1234567890")
substr <- h2o.substring(string_to_substring,2)  #Get substring from second index onwards

h2o.sum  Compute the frame's sum by-column (or by-row).

Description

Compute the frame's sum by-column (or by-row).

Usage

h2o.sum(x, na.rm = FALSE, axis = 0, return_frame = FALSE)

Arguments

x  An H2OFrame object.
na.rm  logical, indicating whether missing values should be removed.
axis  An int that indicates whether to do down a column (0) or across a row (1).
return_frame  A boolean that indicates whether to return an H2O frame or a list. Default is FALSE.

See Also

sum for the base R implementation.
h2o.summary

*Summarizes the columns of an H2OFrame.*

**Description**

A method for the `summary` generic. Summarizes the columns of an H2O data frame or subset of columns and rows using vector notation (e.g. `dataset[row, col]`).

**Usage**

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

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

**Arguments**

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

**Details**

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

**Value**

A table displaying the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column, and the levels and category counts of the levels in each categorical column.

**Examples**

```r
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
prostate.hex <- h2o.importFile(path = prosPath)
summary(prostate.hex)
summary(prostate.hex$GLEASON)
summary(prostate.hex[,4:6])
summary(prostate.hex, exact_quantiles=TRUE)
```
Singular value decomposition of an H2O data frame using the power method

Description

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

Usage

h2o.svd(training_frame, x, destination_key, model_id = NULL,
validation_frame = NULL, ignore_const_cols = TRUE,
score_each_iteration = FALSE, transform = c("NONE", "STANDARDIZE",
"NORMALIZE", "DEMEAN", "DESCALE"), svd_method = c("GramSVD", "Power",
"Randomized"), nv = 1, max_iterations = 1000, seed = -1,
keep_u = TRUE, u_name = NULL, use_all_factor_levels = TRUE,
max_runtime_secs = 0)

Arguments

training_frame  
Id of the training data frame.

x  
A vector containing the character names of the predictors in the model.

destination_key  
(Optional) The unique hex key assigned to the resulting model. Automatically
generated if none is provided.

model_id  
Destination id for this model; auto-generated if not specified.

validation_frame  
Id of the validation data frame.

ignore_const_cols  
Logical. Ignore constant columns. Defaults to TRUE.

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

transform  
Transformation of training data Must be one of: "NONE", "STANDARDIZE",
"NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.

svd_method  
Method for computing SVD (Caution: Randomized is currently experimental
and unstable) Must be one of: "GramSVD", "Power", "Randomized". Defaults to GramSVD.

nv  
Number of right singular vectors Defaults to 1.

max_iterations  
Maximum iterations Defaults to 1000.

seed  
Seed for random numbers (affects certain parts of the algo that are stochastic
and those might or might not be enabled by default) Defaults to -1 (time-based
random number).

keep_u  
Logical. Save left singular vectors? Defaults to TRUE.


**u_name**
Frame key to save left singular vectors

**use_all_factor_levels**
Logical. Whether first factor level is included in each categorical expansion
Defaults to TRUE.

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

**Value**
Returns an object of class **H2ODimReductionModel**.

**References**

**Examples**

```r
library(h2o)
h2o.init()
australia.hex <- h2o.uploadFile(path = ausPath)
h2o.svd(training_frame = australia.hex, nv = 8)
```

---

**h2o.table**

*Cross Tabulation and Table Creation in H2O*

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

**Usage**

```r
h2o.table(x, y = NULL, dense = TRUE)
```

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

**Arguments**

**x**
An H2OFrame object with at most two columns.

**y**
An H2OFrame similar to x, or NULL.

**dense**
A logical for dense representation, which lists only non-zero counts, 1 combination per row. Set to FALSE to expand counts across all combinations.
h2o.tabulate

Value

Returns a tabulated H2OFrame object.

Examples

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

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

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

h2o.tabulate  

Tabulation between Two Columns of an H2OFrame

Description

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

Usage

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

Arguments

data  
An H2OFrame object.

x  
predictor column

y  
response column

weights_column  
(optional) observation weights column

nbins_x  
number of bins for predictor column

nbins_y  
number of bins for response column

Value

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

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

---

h2o.tan

Compute the tangent of x

Description

Compute the tangent of x

Usage

h2o.tan(x)

Arguments

x  
An H2OFrame object.

See Also

tan for the base R implementation.

---

h2o.tanh

Compute the hyperbolic tangent of x

Description

Compute the hyperbolic tangent of x

Usage

h2o.tanh(x)

Arguments

x  
An H2OFrame object.

See Also

tanh for the base R implementation.
**Description**


**Usage**

```r
h2o.target_encode_apply(data, x, y, target_encode_map, holdout_type, fold_column = NULL, blended_avg = TRUE, noise_level = NULL, seed = -1)
```

**Arguments**

- `data` An H2OFrame object with which to apply the target encoding map.
- `x` A list containing the names or indices of the variables to encode. A target encoding column will be created for each element in the list. Items in the list can be multiple columns. For example, if `x = list(c("A"), c("B", "C"))`, then the resulting frame will have a target encoding column for A and a target encoding column for B & C (in this case, we group by two columns).
- `y` The name or column index of the response variable in the data. The response variable can be either numeric or binary.
- `target_encode_map` A list of H2OFrame objects that is the results of the `h2o.target_encode_create` function.
- `holdout_type` The holdout type used. Must be one of: "LeaveOneOut", "KFold", "None".
- `fold_column` (Optional) The name or column index of the fold column in the data. Defaults to NULL (no `fold_column`). Only required if `holdout_type` = "KFold".
- `blended_avg` Logical. (Optional) Whether to perform blended average.
- `noise_level` (Optional) The amount of random noise added to the target encoding. This helps prevent overfitting. Defaults to 0.01 * range of y.
- `seed` (Optional) A random seed used to generate draws from the uniform distribution for random noise. Defaults to -1.

**Value**

Returns an H2OFrame object containing the target encoding per record.

**See Also**

- `h2o.target_encode_create` for creating the target encoding map
Examples

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

# Get Target Encoding Frame on bank-additional-full data with numeric 'y'
data <- h2o.importFile(
  destination_frame = "data")
splits <- h2o.splitFrame(data, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]
mapping <- h2o.target_encode_create(data = train, x = list(c("job"), c("job", "marital")),
y = "age")

# Apply mapping to the training dataset
train_encode <- h2o.target_encode_apply(data = train, x = list(c("job"), c("job", "marital")),
y = "age", mapping, holdout_type = "LeaveOneOut")

# Apply mapping to a test dataset
test_encode <- h2o.target_encode_apply(data = test, x = list(c("job"), c("job", "marital")),
y = "age", target_encode_map = mapping, holdout_type = "None")
```

---

### h2o.target_encode_create

**Create Target Encoding Map**

**Description**

Creates a target encoding map based on group-by columns (’x’) and a numeric or binary target column (’y’). Computing target encoding for high cardinality categorical columns can improve performance of supervised learning models. A Target Encoding tutorial is available here: https://github.com/h2oai/h2o-tutorials/blob/master/best-practices/categorical-predictors/target_encoding.md.

**Usage**

```r
h2o.target_encode_create(data, x, y, fold_column = NULL)
```

**Arguments**

- **data**: An H2OFrame object with which to create the target encoding map.
- **x**: A list containing the names or indices of the variables to encode. A target encoding map will be created for each element in the list. Items in the list can be multiple columns. For example, if `x = list(c("A"), c("B", "C"))`, then there will be one mapping frame for A and one mapping frame for B & C (in this case, we group by two columns).
The name or column index of the response variable in the data. The response variable can be either numeric or binary.

fold_column
(Optional) The name or column index of the fold column in the data. Defaults to NULL (no fold_column).

Value
Returns a list of H2OFrame objects containing the target encoding mapping for each column in 'x'.

See Also
h2o.target_encode_apply for applying the target encoding mapping to a frame.

Examples

library(h2o)
h2o.init()

# Get Target Encoding Map on bank-additional-full data with numeric response
data <- h2o.importFile(
  destination_frame = "data"
)
mapping_age <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")),
y = "age")
head(mapping_age)

# Get Target Encoding Map on bank-additional-full data with binary response
mapping_y <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")),
y = "y")
head(mapping_y)

---

**h2o.toFrame**

*Convert a word2vec model into an H2OFrame*

**Description**

Converts a given word2vec model into an H2OFrame. The frame represents learned word embeddings.

**Usage**

h2o.toFrame(word2vec)

**Arguments**

word2vec A word2vec model.
Examples

```r
h2o.init()

# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v.model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors and return average vector for each sentence
h2o.toFrame(w2v.model) # -> Frame made of 2 rows and 2 columns
```

### h2o.tokenize

**Tokenize String**

**Description**

h2o.tokenize is similar to h2o.strsplit, the difference between them is that h2o.tokenize will store the tokenized text into a single column making it easier for additional processing (filtering stop words, word2vec algo, ...).

**Usage**

```r
h2o.tokenize(x, split)
```

**Arguments**

- `x` The column or columns whose strings to tokenize.
- `split` The regular expression to split on.

**Value**

An H2OFrame with a single column representing the tokenized Strings. Original rows of the input DF are separated by NA.

**Examples**

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

string_to_tokenize <- as.h2o("Split at every character and tokenize."")
tokenize_string <- h2o.tokenize(as.character(string_to_tokenize),"")
```
**h2o.tolower**  
*Convert strings to lowercase*

**Description**  
Convert strings to lowercase

**Usage**  
```r  
h2o.tolower(x)  
```

**Arguments**  
- **x**  
  An H2OFrame object whose strings should be lower cased

**Value**  
An H2OFrame with all entries in lowercase format

**Examples**
```
library(h2o)
h2o.init()
string_to_lower <- as.h2o("ABCDE")
lowered_string <- h2o.tolower(string_to_lower)
```

---

**h2o.topN**  
*H2O topN*

**Description**  
Extract the top N percent of values of a column and return it in a H2OFrame.

**Usage**  
```r  
h2o.topN(x, column, nPercent)  
```

**Arguments**
- **x**  
  an H2OFrame  
- **column**  
  is a column name or column index to grab the top N percent value from  
- **nPercent**  
  is a top percentage value to grab
**h2o.tot_withinss**

*Get the total within cluster sum of squares.*

**Description**

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

**Usage**

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

**Arguments**

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

---

**Value**

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

```r
h2o.totss
```

*Get the total sum of squares.*

**Description**

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

**Usage**

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

**Arguments**

- **object**
  - An H2OClusteringModel object.
- **train**
  - Retrieve the training total sum of squares
- **valid**
  - Retrieve the validation total sum of squares
- **xval**
  - Retrieve the cross-validation total sum of squares
**h2o.toupper**

Convert strings to uppercase

**Description**

Convert strings to uppercase

**Usage**

h2o.toupper(x)

**Arguments**

x An H2OFrame object whose strings should be upper cased

**Value**

An H2OFrame with all entries in uppercase format

**Examples**

```r
library(h2o)
h2o.init()
string_to_upper <- as.h2o("abcde")
upper_string <- h2o.toupper(string_to_upper)
```

---

**h2o.transform**

Transform words (or sequences of words) to vectors using a word2vec model.

**Description**

Transform words (or sequences of words) to vectors using a word2vec model.

**Usage**

h2o.transform(word2vec, words, aggregate_method = c("NONE", "AVERAGE"))
Arguments

word2vec  A word2vec model.
words  An H2OFrame made of a single column containing source words.
aggregate_method  Specifies how to aggregate sequences of words. If method is ‘NONE’ then no aggregation is performed and each input word is mapped to a single word-vector. If method is ‘AVERAGE’ then input is treated as sequences of words delimited by NA. Each word of a sequences is internally mapped to a vector and vectors belonging to the same sentence are averaged and returned in the result.

Examples

h2o.init()

# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v.model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors without aggregation
sentences <- as.character(as.h2o(c("b", "c", "a", NA, "b")))
h2o.transform(w2v.model, sentences) # -> 5 rows total, 2 rows NA ("c" is not in the vocabulary)

# Transform words to vectors and return average vector for each sentence
h2o.transform(w2v.model, sentences, aggregate_method = "AVERAGE") # -> 2 rows

h2o.trim  Trim Space

Description

Trim Space

Usage

h2o.trim(x)

Arguments

x  The column whose strings should be trimmed.

Examples

library(h2o)
h2o.init()
string_to_trim <- as.h2o("r tutorial")
trim_string <- h2o.trim(string_to_trim)
**h2o.trunc**

*Truncate values in x toward 0*

**Description**

trunc takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0.

**Usage**

h2o.trunc(x)

**Arguments**

x  
An H2OFrame object.

**See Also**

trunc for the base R implementation.

---

**h2o.unique**

*H2O Unique*

**Description**

Extract unique values in the column.

**Usage**

h2o.unique(x)

**Arguments**

x  
An H2OFrame object.

**Value**

Returns an H2OFrame object.
h2o.var

Variance of a column or covariance of columns.

Description

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

Usage

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

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

Arguments

x An H2OFrame object.
y NULL (default) or an H2OFrame. The default is equivalent to y = x.
na.rm logical. Should missing values be removed?
use An optional character string indicating how to handle missing values. This must
be one of the following: "everything" - outputs NaNs whenever one of its con-
tributing observations is missing "all.obs" - presence of missing observations
will throw an error "complete.obs" - discards missing values along with all ob-
servations in their rows so that only complete observations are used

See Also

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

Examples

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

Retrieve the variable importance.

Description

Retrieve the variable importance.

Usage

h2o.varimp(object)

Arguments

object An H2OModel object.

h2o.varimp_plot

Plot Variable Importances

Description

Plot Variable Importances

Usage

h2o.varimp_plot(model, num_of_features = NULL)

Arguments

model A trained model (accepts a trained random forest, GBM, or deep learning model, will use h2o.std_coef_plot for a trained GLM

num_of_features The number of features shown in the plot (default is 10 or all if less than 10).

See Also

h2o.std_coef_plot for GLM.
Examples

```r
library(h2o)
h2o.init()
prosPath <- system.file("extdata", "prostate.csv", package="h2o")
hex <- h2o.importFile(prosPath)
hex[,2] <- as.factor(hex[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = hex, distribution = "bernoulli")
h2o.varimp_plot(model)

# for deep learning set the variable_importance parameter to TRUE
iris.hex <- as.h2o(iris)
iris.dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex,
 variable_importances = TRUE)
h2o.varimp_plot(iris.dl)
```

**h2o.week**

*Convert Milliseconds to Week of Week Year in H2O Datasets*

**Description**

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

**Usage**

```r
h2o.week(x)
```

```r
week(x)
```

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

**Arguments**

- `x` An H2OFrame object.

**Value**

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

**See Also**

- `h2o.month`
h2o.weights

Retrieve the respective weight matrix

Description
Retrieve the respective weight matrix

Usage
h2o.weights(object, matrix_id = 1)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An H2OModel or H2OModelMetrics</td>
</tr>
<tr>
<td>matrix_id</td>
<td>An integer, ranging from 1 to number of layers + 1, that specifies the weight matrix to return.</td>
</tr>
</tbody>
</table>

h2o.which

Which indices are TRUE?

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

Usage
h2o.which(x)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>An H2OFrame object.</td>
</tr>
</tbody>
</table>

Value
Returns an H2OFrame object.

See Also
which for the base R method.

Examples

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

Which indice contains the max value?

Description
Get the index of the max value in a column or row

Usage
h2o.which_max(x, na.rm = TRUE, axis = 0)

which.max.H2OFrame(x, na.rm = TRUE, axis = 0)

which.min.H2OFrame(x, na.rm = TRUE, axis = 0)

Arguments
x An H2OFrame object.
na.rm logical. Indicate whether missing values should be removed.
axis integer. Indicate whether to calculate the mean down a column (0) or across a row (1).

Value
Returns an H2OFrame object.

See Also
which.max for the base R method.

h2o.which_min

Which index contains the min value?

Description
Get the index of the min value in a column or row

Usage
h2o.which_min(x, na.rm = TRUE, axis = 0)

Arguments
x An H2OFrame object.
na.rm logical. Indicate whether missing values should be removed.
axis integer. Indicate whether to calculate the mean down a column (0) or across a row (1).
Value
Returns an H2OFrame object.

See Also
which.min for the base R method.

---

**h2o.withinss**

*Get the Within SS*

Description
Get the Within SS

Usage
h2o.withinss(object)

Arguments
- **object**
  - An H2OClusteringModel object.

---

**h2o.word2vec**

*Trains a word2vec model on a String column of an H2O data frame*

Description
Trains a word2vec model on a String column of an H2O data frame

Usage
h2o.word2vec(training_frame = NULL, model_id = NULL, min_word_freq = 5,
word_model = c("SkipGram"), norm_model = c("HSM"), vec_size = 100,
window_size = 5, sent_sample_rate = 0.001, init_learning_rate = 0.025,
ePOCHS = 5, pre_trained = NULL, max_runtime_secs = 0)

Arguments
- **training_frame**
  - Id of the training data frame.
- **model_id**
  - Destination id for this model; auto-generated if not specified.
- **min_word_freq**
  - This will discard words that appear less than <int> times Defaults to 5.
- **word_model**
  - Use the Skip-Gram model Must be one of: "SkipGram". Defaults to SkipGram.
- **norm_model**
  - Use Hierarchical Softmax Must be one of: "HSM". Defaults to HSM.
- **vec_size**
  - Set size of word vectors Defaults to 100.
window_size Set max skip length between words Defaults to 5.
sent_sample_rate Set threshold for occurrence of words. Those that appear with higher frequency in the training data will be randomly down-sampled; useful range is (0, 1e-5) Defaults to 0.001.
init_learning_rate Set the starting learning rate Defaults to 0.025.
epochs Number of training iterations to run Defaults to 5.
pre_trained Id of a data frame that contains a pre-trained (external) word2vec model
max_runtime_secs Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.

h2o.xgboost Build an eXtreme Gradient Boosting model

Description
Builds a eXtreme Gradient Boosting model using the native XGBoost backend.

Usage
h2o.xgboost(x, y, training_frame, model_id = NULL, validation_frame = NULL,
nfolds = 0, keep_cross_validation_predictions = FALSE,
keep_cross_validation_fold_assignment = FALSE,
score_each_iteration = FALSE, fold_assignment = c("AUTO", "Random",
"Modulo", "Stratified"), fold_column = NULL, ignore_const_cols = TRUE,
ofset_column = NULL, weights_column = NULL, stopping_rounds = 0,
stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE",
"RMSLE", "AUC", "lift_top_group", "misclassification",
"mean_per_class_error"), stopping_tolerance = 0.001, max_runtime_secs = 0,
seed = -1, distribution = c("AUTO", "bernoulli", "multinomial",
"gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"),
tweedie_power = 1.5, categorical_encoding = c("AUTO", "Enum",
"OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder",
"SortByResponse", "EnumLimited"), quiet_mode = TRUE, ntree = 50,
max_depth = 6, min_rows = 1, min_child_weight = 1, learn_rate = 0.3,
eta = 0.3, sample_rate = 1, subsample = 1, col_sample_rate = 1,
colsample_bylevel = 1, col_sample_rate_per_tree = 1,
colsample_bytree = 1, max_abs_leafnode_pred = 0, max_delta_step = 0,
score_tree_interval = 0, min_split_improvement = 0, gamma = 0,
ndread = -1, max_bins = 256, max_leaves = 0,
min_sum_hessian_in_leaf = 100, min_data_in_leaf = 0,
sample_type = c("uniform", "weighted"), normalize_type = c("tree",
"forest"), rate_drop = 0, one_drop = FALSE, skip_drop = 0,
tree_method = c("auto", "exact", "approx", "hist"),
grow_policy = c("depthwise", "lossguide"), booster = c("gbtree", "gblinear", "dart"), reg_lambda = 0, reg_alpha = 0,
dmatrix_type = c("auto", "dense", "sparse"), backend = c("auto", "gpu", "cpu"), gpu_id = 0, verbose = FALSE)

Arguments

x (Optional) A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.

y The name or column index of the response variable in the data. The response must be either a numeric or a categorical/factor variable. If the response is numeric, then a regression model will be trained, otherwise it will train a classification model.

training_frame Id of the training data frame.

model_id Destination id for this model; auto-generated if not specified.

validation_frame Id of the validation data frame.

nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.

keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.

keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

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

ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.

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

weights_column Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.
stopping_rounds
Early stopping based on convergence of stopping_metric. Stop if simple moving
average of length k of the stopping_metric does not improve for k:=stopping_rounds
scoring events (0 to disable) Defaults to 0.

stopping_metric
Metric to use for early stopping (AUTO: logloss for classification, deviance for
regression) Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE",
"MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error".
Defaults to AUTO.

stopping_tolerance
Relative tolerance for metric-based stopping criterion (stop if relative improve-
ment is not at least this much) Defaults to 0.001.

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

seed
Seed for random numbers (affects certain parts of the algo that are stochastic
and those might or might not be enabled by default) Defaults to -1 (time-based
random number).

distribution
Distribution function Must be one of: "AUTO", "bernoulli", "multinomial",
"gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber". De-
faults to AUTO.

tweedie_power
Tweedie power for Tweedie regression, must be between 1 and 2. Defaults to
1.5.

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

quiet_mode
Logical. Enable quiet mode Defaults to TRUE.

ntrees
(same as n_estimators) Number of trees. Defaults to 50.

max_depth
Maximum tree depth. Defaults to 6.

min_rows
(same as min_child_weight) Fewest allowed (weighted) observations in a leaf.
Defaults to 1.

min_child_weight
(same as min_rows) Fewest allowed (weighted) observations in a leaf. Defaults
to 1.

learn_rate
(same as eta) Learning rate (from 0.0 to 1.0) Defaults to 0.3.

eta
(same as learn_rate) Learning rate (from 0.0 to 1.0) Defaults to 0.3.

sample_rate
(same as subsample) Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.

subsample
(same as sample_rate) Row sample rate per tree (from 0.0 to 1.0) Defaults to 1.

col_sample_rate
(same as colsample_bylevel) Column sample rate (from 0.0 to 1.0) Defaults to
1.

colsample_bylevel
(same as col_sample_rate) Column sample rate (from 0.0 to 1.0) Defaults to 1.
col_sample_rate_per_tree
(same as colsample_bytree) Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
colsample_bytree
(same as col_sample_rate_per_tree) Column sample rate per tree (from 0.0 to 1.0) Defaults to 1.
max_abs_leafnode_pred
(same as max_delta_step) Maximum absolute value of a leaf node prediction Defaults to 0.0.
max_delta_step
(same as max_abs_leafnode_pred) Maximum absolute value of a leaf node prediction Defaults to 0.0.
score_tree_interval
Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
min_split_improvement
(same as gamma) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
gamma
(same as min_split_improvement) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
nthread
Number of parallel threads that can be used to run XGBoost. Cannot exceed H2O cluster limits (-nthreads parameter). Defaults to maximum available Defaults to -1.
max_bins
For tree_method=hist only: maximum number of bins Defaults to 256.
max_leaves
For tree_method=hist only: maximum number of leaves Defaults to 0.
min_sum_hessian_in_leaf
For tree_method=hist only: the minimum sum of hessian in a leaf to keep splitting Defaults to 100.0.
min_data_in_leaf
For tree_method=hist only: the minimum data in a leaf to keep splitting Defaults to 0.0.
sample_type
For booster=dart only: sample_type Must be one of: "uniform", "weighted". Defaults to uniform.
normalize_type
For booster=dart only: normalize_type Must be one of: "tree", "forest". Defaults to tree.
rate_drop
For booster=dart only: rate_drop (0..1) Defaults to 0.0.
one_drop
Logical. For booster=dart only: one_drop Defaults to FALSE.
skip_drop
For booster=dart only: skip_drop (0..1) Defaults to 0.0.
tree_method
Tree method Must be one of: "auto", "exact", "approx", "hist". Defaults to auto.
grow_policy
Grow policy - depthwise is standard GBM, lossguide is LightGBM Must be one of: "depthwise", "lossguide". Defaults to depthwise.
booster
Booster type Must be one of: "gbtree", "gblinear", "dart". Defaults to gbtree.
reg_lambda
L2 regularization Defaults to 0.0.
reg_alpha
L1 regularization Defaults to 0.0.
dmatrix_type  Type of DMatrix. For sparse, NAs and 0 are treated equally. Must be one of: "auto", "dense", "sparse". Defaults to auto.

backend  Backend. By default (auto), a GPU is used if available. Must be one of: "auto", "gpu", "cpu". Defaults to auto.

gpu_id  Which GPU to use. Defaults to 0.

verbose  Logical. Print scoring history to the console (Metrics per tree for GBM, DRF, & XGBoost. Metrics per epoch for Deep Learning). Defaults to FALSE.

h2o.xgboost.available  Determines whether an XGBoost model can be built

Description
Ask the H2O server whether a XGBoost model can be built. (Depends on availability of native backend.) Returns True if a XGBoost model can be built, or False otherwise.

Usage
h2o.xgboost.available()

h2o.year  Convert Milliseconds to Years in H2O Datasets

Description
Convert the entries of an H2OFrame object from milliseconds to years, indexed starting from 1900.

Usage
h2o.year(x)

year(x)

## S3 method for class 'H2OFrame'
year(x)

Arguments
x  An H2OFrame object.

Details
This method calls the function of the MutableDateTime class in Java.
Value
An H2OFrame object containing the entries of x converted to years

See Also
h2o.month

---

### H2OAutoML-class

#### Description
This class represents an H2OAutoML object

---

### H2OClusteringModel-class

#### Description
This virtual class represents a clustering model built by H2O.

#### Details
This object has slots for the key, which is a character string that points to the model key existing in the H2O cloud, the data used to build the model (an object of class H2OFrame).

#### Slots
- **model_id**: A character string specifying the key for the model fit in the H2O cloud's key-value store.
- **algorithm**: A character string specifying the algorithm that was used to fit the model.
- **parameters**: A list containing the parameter settings that were used to fit the model that differ from the defaults.
- **allparameters**: A list containing all parameters used to fit the model.
- **model**: A list containing the characteristics of the model returned by the algorithm.
  - **size**: The number of points in each cluster.
  - **totss**: Total sum of squared error to grand mean.
  - **withinss**: A vector of within-cluster sum of squared error.
  - **tot_withinss**: Total within-cluster sum of squared error.
  - **betweenss**: Between-cluster sum of squared error.
The H2OConnection class.

Description
This class represents a connection to an H2O cloud.

Usage
```
## S4 method for signature 'H2OConnection'
show(object)
```

Arguments
object an H2OConnection object.

Details
Because H2O is not a master-slave architecture, there is no restriction on which H2O node is used to establish the connection between R (the client) and H2O (the server).

A new H2O connection is established via the h2o.init() function, which takes as parameters the 'ip' and 'port' of the machine running an instance to connect with. The default behavior is to connect with a local instance of H2O at port 54321, or to boot a new local instance if one is not found at port 54321.

Slots
- ip A character string specifying the IP address of the H2O cloud.
- port A numeric value specifying the port number of the H2O cloud.
- proxy A character specifying the proxy path of the H2O cloud.
- https Set this to TRUE to use https instead of http.
- insecure Set this to TRUE to disable SSL certificate checking.
- username Username to login with.
- password Password to login with.
- cookies Cookies to add to request
- context_path Context path which is appended to H2O server location.
- mutable An H2OConnectionMutableState object to hold the mutable state for the H2O connection.
**H2OConnectionMutableState**

*The H2OConnectionMutableState class*

**Description**

This class represents the mutable aspects of a connection to an H2O cloud.

**Slots**

- `session_id` A character string specifying the H2O session identifier.
- `key_count` A integer value specifying count for the number of keys generated for the `session_id`.

**H2OCoxPHModel-class**  
*The H2OCoxPHModel object.*

**Description**

Virtual object representing H2O’s CoxPH Model.

**Usage**

```r
## S4 method for signature 'H2OCoxPHModel'
show(object)

## S3 method for class 'H2OCoxPHModel'
coef(object, ...)

## S3 method for class 'H2OCoxPHModel'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'H2OCoxPHModel'
logLik(object, ...)

## S3 method for class 'H2OCoxPHModel'
survfit.H2OCoxPHModel(formula, newdata, ...)

## S3 method for class 'H2OCoxPHModel'
vcov(object, ...)
```

**Arguments**

- `object` an H2OCoxPHModel object.
- `...` additional arguments to pass on.
- `fit` an H2OCoxPHModel object.
scale optional numeric specifying the scale parameter of the model.
k numeric specifying the weight of the equivalent degrees of freedom.
formula an H2OcoxPHModel object.
newdata an optional H2OFrame or data.frame with the same variable names as those that appear in the H2OcoxPHModel object.

---

H2OCoxPHModelSummary-class

The H2OCoxPHModelSummary object.

Description

Wrapper object for summary information compatible with survival package.

Usage

## S4 method for signature 'H2OCoxPHModelSummary'
show(object)

## S3 method for class 'H2OCoxPHModelSummary'
coef(object, ...)

Arguments

object An H2OCoxPHModelSummary object.
...
additional arguments to pass on.

Slots

summary A list containing the a summary compatible with CoxPH summary used in the survival package.

---

H2OFrame-class

The H2OFrame class

Description

This class represents an H2OFrame object
Description

Operators to extract or replace parts of H2OFrame objects.

Usage

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

## S3 method for class 'H2OFrame'
x$name

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

## S3 method for class 'H2OFrame'
x$name

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

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

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

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

Arguments

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

H2OGrid-class H2O Grid

Description

A class to contain the information about grid results
Format grid object in user-friendly way

Usage

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

Arguments

object an H2OGrid object.

Slots

- `grid_id` the final identifier of grid
- `model_ids` list of model IDs which are included in the grid object
- `hyper_names` list of parameter names used for grid search
- `failed_params` list of model parameters which caused a failure during model building, it can contain a null value
- `failure_details` list of detailed messages which correspond to failed parameters field
- `failure_stack_traces` list of stack traces corresponding to model failures reported by failed_params and failure_details fields
- `failed_raw_params` list of failed raw parameters
- `summary_table` table of models built with parameters and metric information.

See Also

H2OModel for the final model types.
**H2OModel-class**

The H2OModel object.

**Description**

This virtual class represents a model built by H2O.

**Usage**

```r
## S4 method for signature 'H2OModel'
show(object)
```

**Arguments**

- `object` an H2OModel object.

**Details**

This object has slots for the key, which is a character string that points to the model key existing in the H2O cloud, the data used to build the model (an object of class H2OFrame).

**Slots**

- `model_id` A character string specifying the key for the model fit in the H2O cloud’s key-value store.
- `algorithm` A character string specifying the algorithm that were used to fit the model.
- `parameters` A list containing the parameter settings that were used to fit the model that differ from the defaults.
- `allparameters` A list containing all parameters used to fit the model.
- `have_pojo` A logical indicating whether export to POJO is supported.
- `have_mojo` A logical indicating whether export to MOJO is supported.
- `model` A list containing the characteristics of the model returned by the algorithm.

---

**H2OModelFuture-class**

**H2O Future Model**

**Description**

A class to contain the information for background model jobs.

**Slots**

- `job_key` a character key representing the identification of the job process.
- `model_id` the final identifier for the model.
The `H2OModelMetrics` Object.

**Description**

A class for constructing performance measures of H2O models.

**Usage**

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

## S4 method for signature 'H2OBinomialMetrics'
show(object)

## S4 method for signature 'H2OMultinomialMetrics'
show(object)

## S4 method for signature 'H2OOrdinalMetrics'
show(object)

## S4 method for signature 'H2ORegressionMetrics'
show(object)

## S4 method for signature 'H2OClusteringMetrics'
show(object)

## S4 method for signature 'H2OAutoEncoderMetrics'
show(object)

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

**Arguments**

- `object` An `H2OModelMetrics` object

**See Also**

`H2OModel` for the final model types.
Description

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

Format

A data frame with 435 rows and 17 columns

Source


References


Description

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

Format

A data frame with 150 rows and 5 columns

Source


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

Description  
Check if character

Usage  
is.character(x)

Arguments  
  x  An H2OFrame object

is.factor  

Description  
Check if factor

Usage  
is.factor(x)

Arguments  
  x  An H2OFrame object

is.h2o  

Description  
Test if object is H2O Frame.

Usage  
is.h2o(x)

Arguments  
  x  An R object.
is.numeric

**Check if numeric**

**Description**

Check if numeric

**Usage**

`is.numeric(x)`

**Arguments**

- `x` An H2OFrame object

---

Logical-or

**Logical or for H2OFrames**

**Description**

Logical or for H2OFrames

**Usage**

`||`(x, y)

**Arguments**

- `x` An H2OFrame object
- `y` An H2OFrame object
Accessor Methods for H2OModel Object

Description

Function accessor methods for various H2O output fields.

Usage

getParms(object)

## S4 method for signature 'H2OModel'
getParms(object)

getCenters(object)

getCentersStd(object)

getWithinSS(object)

getTotWithinSS(object)

getBetweenSS(object)

getTotSS(object)

getIterations(object)

getClusterSizes(object)

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

## S4 method for signature 'H2OClusteringModel'
getCentersStd(object)

## S4 method for signature 'H2OClusteringModel'
getWithinSS(object)

## S4 method for signature 'H2OClusteringModel'
getTotWithinSS(object)

## S4 method for signature 'H2OClusteringModel'
getBetweenSS(object)

## S4 method for signature 'H2OClusteringModel'
getTotSS(object)
names.H2OFrame

## S4 method for signature 'H2OClusteringModel'
getIterations(object)

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

**Arguments**

object an **H2OModel** class object.

---

**Description**

Column names of an H2OFrame

**Usage**

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

**Arguments**

x An H2OFrame

---

**Description**

Methods for group generic functions and H2O objects.

**Usage**

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

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

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

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

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

## S3 method for class 'H2OFrame'
!x

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

## S3 method for class 'H2OFrame'
t(x)

log(x, ...)

log10(x)

log2(x)

log1p(x)

trunc(x, ...)

x %*% y

nrow.H2OFrame(x)

ncol.H2OFrame(x)

## S3 method for class 'H2OFrame'
length(x)

h2o.length(x)

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

colnames(x) <- value

### Arguments

e1 object
e2 object
x object
...
Further arguments passed to or from other methods.
na.rm logical. whether or not missing values should be removed
plot.H2OModel

y object
value To be assigned

plot.H2OModel  Plot an H2O Model

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

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

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

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

Value
Returns a scoring history plot.

See Also
- `h2o.deeplearning`, `h2o.gbm`, `h2o.glm`, `h2o.randomForest` for model generation in h2o.

Examples
```r
if (requireNamespace("mlbench", quietly=TRUE)) {
  library(h2o)
  h2o.init()

  df <- as.h2o(mlbench::mlbench.friedman1(10000,1))
  rng <- h2o.runif(df, seed=1234)
  train <- df[rng<0.8,]
  valid <- df[rng>=0.8,]
```
```r
gbm <- h2o.gbm(x = 1:nrow(train), y = "y", training_frame = train, validation_frame = valid, ntree = 500, learn_rate = 0.01, score_each_iteration = TRUE)
plot(gbm)
plot(gbm, timestep = "duration", metric = "deviance")
plot(gbm, timestep = "number_of_trees", metric = "deviance")
plot(gbm, timestep = "number_of_trees", metric = "rmse")
plot(gbm, timestep = "number_of_trees", metric = "mae")
```

---

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

**Description**

Plots the simple co-occurrence based tabulation of X vs Y as a heatmap, where X and Y are two Vecs in a given dataset. This function requires suggested ggplot2 package.

**Usage**

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

**Arguments**

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

**Value**

Returns a ggplot2-based heatmap of co-occurrence.

**See Also**

`h2o.tabulate`
predict.H2OAutoML

Examples

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

Predict on an AutoML object

Description

Obtains predictions from an AutoML object.

Usage

## S3 method for class 'H2OAutoML'
predict(object, newdata, ...)

Arguments

object       a fitted H2OAutoML object for which prediction is desired
newdata     An H2OFrame object in which to look for variables with which to predict.
...          additional arguments to pass on.

Details

This method generated predictions on the leader model from an AutoML run. The order of the rows
in the results is the same as the order in which the data was loaded, even if some rows fail (for
example, due to missing values or unseen factor levels).

Value

Returns an H2OFrame object with probabilities and default predictions.
predict.H2OModel  

**Predict on an H2O Model**

Description

Obtains predictions from various fitted H2O model objects.

Usage

```r
## S3 method for class 'H2OModel'
predict(object, newdata, ...)

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

Arguments

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

Details

This method dispatches on the type of H2O model to select the correct prediction/scoring algorithm. The order of the rows in the results is the same as the order in which the data was loaded, even if some rows fail (for example, due to missing values or unseen factor levels).

Value

Returns an H2OFrame object with probabilities and default predictions.

See Also

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

predict_leaf_node_assignment.H2OModel  

**Predict the Leaf Node Assignment on an H2O Model**

Description

Obtains leaf node assignment from fitted H2O model objects.
Usage

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

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

Arguments

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

Details

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

Value

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

See Also

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

Examples

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

- **x**: An H2OFrame object
- **n**: An (Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).
- **...**: Further arguments to be passed from or to other methods.

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

The original x object

---

**prostate**

*Prostate Cancer Study*

**Description**

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

**Format**

A data frame with 380 rows and 9 columns

**Source**

range.H2OFrame

Range of an H2O Column

**Description**

Range of an H2O Column

**Usage**

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

**Arguments**

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

---

str.H2OFrame

Display the structure of an H2OFrame object

**Description**

Display the structure of an H2OFrame object

**Usage**

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

**Arguments**

- `object`
  - An H2OFrame.
- `...`
  - Further arguments to be passed from or to other methods.
- `cols`
  - Print the per-column str for the H2OFrame
**summary,H2OGrid-method**

*Summary method for H2OCoxPHModel objects*

**Description**

Summary method for H2OCoxPHModel objects

**Usage**

```r
## S4 method for signature 'H2OCoxPHModel'
summary(object, conf.int = 0.95, scale = 1)
```

**Arguments**

- `object`: an `H2OCoxPHModel` object.
- `conf.int`: a specification of the confidence interval.
- `scale`: a scale.

**summary,H2OGrid-method**

*Format grid object in user-friendly way*

**Description**

Format grid object in user-friendly way

**Usage**

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

**Arguments**

- `object`: an `H2OGrid` object.
- `show_stack_traces`: a flag to show stack traces for model failures
**summary.H2OModel-method**

*Print the Model Summary*

**Description**

Print the Model Summary

**Usage**

```r
## S4 method for signature 'H2OModel'
summary(object, ...)
```

**Arguments**

- `object` : An `H2OModel` object.
- `...` : further arguments to be passed on (currently unimplemented)

**use.package**

*Use optional package*

**Description**

Testing availability of optional package, its version, and extra global default. This function is used internally. It is exported and documented because user can control behavior of the function by global option.

**Usage**

```r
use.package(package, version = "1.9.8"[package == "data.table"],
            use =getOption("h2o.use.data.table", FALSE)[package == "data.table"])
```

**Arguments**

- `package` : character scalar name of a package that we Suggests or Enhances on.
- `version` : character scalar required version of a package.
- `use` : logical scalar, extra escape option, to be used as global option.

**Details**

We use this function to control csv read/write with optional `data.table` package. Currently `data.table` is disabled by default, to enable it set options("h2o.use.data.table"=TRUE). It is possible to control just `fread` or `fwrite` with options("h2o.fread"=FALSE, "h2o.fwrite"=FALSE). `h2o.fread` and `h2o.fwrite` options are not handled in this function but next to `fread` and `fwrite` calls.
See Also

`as.h2o.data.frame, as.data.frame.H2OFrame`

Examples

```r
op <- options("h2o.use.data.table" = TRUE)
if (use.package("data.table")) {
  cat("optional package data.table 1.9.8+ is available\n")
} else {
  cat("optional package data.table 1.9.8+ is not available\n")
}
options(op)
```

---

**walking**

*Muscular Actuations for Walking Subject*

Description

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

Format

A data frame with 151 rows and 124 columns

References


---

**Shutdown H2O cloud after examples run**

Description

Shutdown H2O cloud after examples run

Examples

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

Logical and for H2OFrames

Description
Logical and for H2OFrames

Usage
"&&"(x, y)

Arguments
x An H2OFrame object
y An H2OFrame object
Index

datasets

australia, 16
housevotes, 229
iris, 229
prostate, 240
walking, 244

package

h2o-package, 9

[.H2OFrame-method (H2OFrame-Extract), 225
[<-.H2OFrame (H2OFrame-Extract), 225
[[-.H2OFrame (H2OFrame-Extract), 225
[[<-.H2OFrame (H2OFrame-Extract), 225
$ .H2OFrame (H2OFrame-Extract), 225
$<-.H2OFrame (H2OFrame-Extract), 225
% .(Ops.H2OFrame), 233
& &, 245

aaa, 10
abs, 19
acos, 20
all, 23, 24
apply, 10, 11
as.character, 26
as.character.H2OFrame, 11
as.data.frame.H2OFrame, 12, 244
as.factor, 12, 13, 26
as.h2o, 13
as.h2o.data.frame, 244
as.matrix.H2OFrame, 14
as.numeric, 15, 27
as.vector.H2OFrame, 16
australia, 16
cbind, 33
ceiling, 34
colmeans, 133
colnames, 17, 39
colnames<-(Ops.H2OFrame), 233
cor (h2o.cor), 43
cos, 44
cosh, 45
cummax, 50
cummin, 50
cumprod, 51
cumsum, 51
cut.H2OFrame (h2o.cut), 52
data.table, 243
day (h2o.day), 53
dayOfWeek (h2o.dayOfWeek), 53
ddply, 55
dim, 17, 71
dim.H2OFrame, 17
dimnames, 71
dimnames.H2OFrame, 18
exp, 76
extractAIC.H2OcoxPHModel
(H2OcoxPHModel-class), 223
floore, 80
fread, 12, 243
fwrite, 14, 243
generate_col_ind, 18
getBetweenSS (ModelAccessors), 232
getBetweenSS, H2OClusteringModel-method
(ModelAccessors), 232
getCenters (ModelAccessors), 232
getCenters, H2OClusteringModel-method
(ModelAccessors), 232

coef.H2OcoxPHModel
(H2OcoxPHModel-class), 223
coef.H2OcoxPHModelSummary
(H2OcoxPHModelSummary-class), 224
colMeans, 133
colnames, 17, 39
colnames<-(Ops.H2OFrame), 233
cor (h2o.cor), 43
cos, 44
cosh, 45
cummax, 50
cummin, 50
cumprod, 51
cumsum, 51
cut.H2OFrame (h2o.cut), 52
data.table, 243
day (h2o.day), 53
dayOfWeek (h2o.dayOfWeek), 53
ddply, 55
dim, 17, 71
dim.H2OFrame, 17
dimnames, 71
dimnames.H2OFrame, 18
exp, 76
extractAIC.H2OcoxPHModel
(H2OcoxPHModel-class), 223
floore, 80
fread, 12, 243
fwrite, 14, 243
generate_col_ind, 18
getBetweenSS (ModelAccessors), 232
getBetweenSS, H2OClusteringModel-method
(ModelAccessors), 232
getCenters (ModelAccessors), 232
getCenters, H2OClusteringModel-method
(ModelAccessors), 232
getCentersStd (ModelAccessors), 232
getCentersStd, H2OClusteringModel-method  
(ModelAccessors), 232
getClusterSizes (ModelAccessors), 232
getClusterSizes, H2OClusteringModel-method  
(ModelAccessors), 232
getIterations (ModelAccessors), 232
getIterations, H2OClusteringModel-method  
(ModelAccessors), 232
getParms (ModelAccessors), 232
getParms, H2OModel-method  
(ModelAccessors), 232
getTotSS (ModelAccessors), 232
getTotSS, H2OClusteringModel-method  
(ModelAccessors), 232
getTotWithinSS (ModelAccessors), 232
getTotWithinSS, H2OClusteringModel-method  
(ModelAccessors), 232
getWithinSS (ModelAccessors), 232
getWithinSS, H2OClusteringModel-method  
(ModelAccessors), 232

h2o (h2o-package), 9
h2o-package, 9
h2o.abs, 19
h2o.accuracy (h2o.metric), 137
h2o.acos, 20
h2o.aggregated_frame, 20
h2o.aggregator, 21
h2o.aic, 22
h2o.all, 23
h2o.anomaly, 23
h2o.any, 24
h2o.anyFactor, 25
h2o.arrange, 25
h2o.as_date, 28
h2o.ascharacter, 26
h2o.asfactor, 26
h2o.asnumeric, 27
h2o.assign, 27, 174
h2o.auc, 28, 92, 97, 138, 142, 175
h2o.autolr, 29
h2o.betweenss, 32, 121
h2o.biases, 32
h2o.bottomN, 33
h2o.cbind, 33
h2o.ceiling, 34
h2o.centers, 34, 121
h2o.centersSTD, 35, 121
h2o.centroid_stats, 35
h2o.clearLog, 36, 151, 190, 191
h2o.cluster_sizes, 38, 121
h2o.clusterInfo, 36
h2o.clusterIsUp, 37
h2o.clusterStatus, 37
h2o.coef, 38
h2o.coef_norm, 39
h2o.colnames, 39
h2o.columns_by_type, 40
h2o.computeGram, 40
h2o.confusionMatrix, 41, 97
h2o.confusionMatrix, H2OModel-method  
(h2o.confusionMatrix), 41
h2o.confusionMatrix, H2OMetrics-method  
(h2o.confusionMatrix), 41
h2o.connect, 42
h2o.cor, 43
h2o.cos, 44
h2o.cosh, 45
h2o.coxph, 45
h2o.createFrame, 46
h2o.cross_validation_fold_assignment, 48
h2o.cross_validation_holdout_predictions, 48
h2o.cross_validation_models, 49
h2o.cross_validation_predictions, 49
h2o.cummax, 50
h2o.cummin, 50
h2o.cumprod, 51
h2o.cumsum, 51
h2o.cut, 52
h2o.day, 53, 54, 106
h2o.dayOfWeek, 53
h2o.dct, 54
h2o.ddply, 55
h2o.decryptionSetup, 56, 108, 153, 154
h2o.deepfeatures, 57
h2o.deeplearning, 24, 57, 58, 235, 238
h2o.deepwater, 57, 64
h2o.deepwater.available, 69
h2o.describe, 69
h2o.diffflag1, 70
h2o.dim, 70
h2o.dimmnames, 71
h2o.distance, 71
h2o.download_mojo, 73
248

h2o.download_pojo, 74
h2o.downloadAllLogs, 72
h2o.downloadCSV, 72
h2o.entropy, 75
h2o.error (h2o.metric), 137
h2o.exp, 76
h2o.exportFile, 76
h2o.exportHDFS, 77
h2o.F0point5 (h2o.metric), 137
h2o.F1 (h2o.metric), 137
h2o.F2 (h2o.metric), 137
h2o.fallout (h2o.metric), 137
h2o.fillna, 77
h2o.findSynonyms, 79
h2o.find_row_by_threshold, 79
h2o.find_threshold_by_max_metric, 80
h2o.floor, 80
h2o.fnr (h2o.metric), 137
h2o.fpr (h2o.metric), 137
h2o.gainsLift, 81
h2o.gainsLift, H2OModel-method (h2o.gainsLift), 81
h2o.gainsLift, H2OModelMetrics-method (h2o.gainsLift), 81
h2o.gbm, 82, 235, 238, 239
h2o.getAutoML, 86
h2o.getConnection, 87
h2o.getFrame, 87
h2o.getFutureModel, 88
h2o.getGLMFullRegularizationPath, 88
h2o.getGrid, 89
h2o.getId, 89
h2o.getModel, 90
h2o.getTimezone, 91
h2o.getTypes, 91
h2o.getVersion, 91
h2o.giniCoef, 28, 92, 97, 138
h2o.glm, 9, 93, 235, 238
h2o.glrm, 97, 158, 161, 170
h2o.grep, 100
h2o.grid, 101
h2o.group_by, 102
h2o.gsub, 103
h2o.head, 104
h2o.hist, 105
h2o.hit_ratio_table, 105
h2o.hour, 106
h2o.ifelse, 106
h2o.import_sql_select, 108, 109
h2o.import_sql_table, 108, 110
h2o.importFile, 56, 107, 153
h2o.importFolder (h2o.importFile), 107
h2o.importHDFS (h2o.importFile), 107
h2o.impute, 111
h2o.init, 37, 112, 185
h2o.insertMissingValues, 114
h2o.interaction, 115
h2o.is_client, 119
h2o.isax, 116
h2o.ischaracter, 117
h2o.isfactor, 118
h2o.isnumeric, 118
h2o.kfold_column, 119
h2o.killMinus3, 119
h2o.kmeans, 99, 120
h2o.kurtosis, 122
h2o.length (Ops.H2OFrame), 233
h2o.levels, 122
h2o.list_all_extensions, 123
h2o.list_api_extensions, 124
h2o.list_core_extensions, 124
h2o.list_timezones, 123
h2o.loadModel, 124, 179
h2o.log, 125
h2o.log10, 125
h2o.log1p, 126
h2o.log2, 126
h2o.logAndEcho, 127
h2o.logloss, 97, 127
h2o.ls, 128, 174
h2o.lstrip, 128
h2o.mae, 129
h2o.make_metrics, 130
h2o.makeGLMModel, 130
h2o.match, 131
h2o.max, 132
h2o.maxPerClassError (h2o.metric), 137
h2o.mcc (h2o.metric), 137
h2o.mean, 132
h2o.mean_per_class_accuracy (h2o.metric), 137
h2o.mean_per_class_error, 133
h2o.mean_residual_deviance, 134
h2o.median, 135
h2o.merge, 136
h2o.metric, 28, 92, 134, 137, 142, 175
h2o.min, 138
h2o.missrate (h2o.metric), 137
h2o.mktme, 139
h2o mojo_predict_csv, 139
h2o mojo_predict_df, 140
h2o.month, 53, 54, 141, 212, 221
h2o.mse, 28, 97, 134, 138, 142, 175
h2o.na_omit, 146
h2o.nacnt, 143
h2o.naiveBayes, 143
h2o.names, 146
h2o.nchar, 147
h2o.ncol, 147
h2o.networkTest, 148
h2o.nlevels, 148
h2o.no_progress, 148
h2o.nrow, 149
h2o.null_deviance, 149
h2o.null_dof, 150
h2o.num_iterations, 121, 150
h2o.num_valid_substrings, 151
h2o.openLog, 36, 151, 190, 191
h2o.parseRaw, 108, 152, 154
h2o.parseSetup, 36, 153, 153
h2o.partialPlot, 154
h2o.performance, 28, 42, 82, 92, 97, 134, 138, 142, 155, 175
h2o.pivot, 156
h2o.prcomp, 99, 157
h2o.precision (h2o.metric), 137
h2o.predict (predict.H2OModel), 238
h2o.predict_json, 159
h2o.predict_leaf_node_assignment
  (predict_leaf_node_assignment.H2OModel), 238
h2o.print, 159
h2o.prod, 160
h2o.proj_archetypes, 160
h2o.quantile, 161
h2o.r2, 162
h2o.randomForest, 163, 235, 238, 239
h2o.range, 167
h2o.rank_within_group_by, 167
h2o.rbind, 169
h2o.recall (h2o.metric), 137
h2o.reconstruct, 170
h2o.relevel, 171
h2o.removeAll, 172
h2o.removeVecs, 172
h2o.rep_len, 173
h2o.residual_deviance, 173
h2o.residual_dof, 174
h2o.rm, 172, 174
h2o.rmse, 175
h2o.rmsle, 176
h2o.round, 176
h2o.rstrip, 177
h2o.runif, 178
h2o.saveModel, 125, 178, 180
h2o.saveModelDetails, 179
h2o.saveMojo, 180
h2o.scale, 181
h2o.scoreHistory, 97, 182
h2o.sd, 182, 210
h2o.sdev, 183
h2o.sensitivity (h2o.metric), 137
h2o.setLevels, 183
h2o.setTimezone, 184
h2o.show_progress, 184
h2o.shutdown, 114, 184
h2o.signif, 185
h2o.sin, 186
h2o.skewness, 186
h2o.specifcity (h2o.metric), 137
h2o.splitFrame, 187
h2o.sqrt, 188
h2o.stackedEnsemble, 188
h2o.startLogging, 36, 151, 190, 191
h2o.std_coef_plot, 190, 211
h2o.stopLogging, 36, 151, 190, 191
h2o.str, 192
h2o.stringdist, 192
h2o.table, 198
h2o.tabulate, 199, 236
h2o.tail (h2o.head), 104
h2o.tan, 200
h2o.tanh, 200
h2o.target_encode_apply, 201, 203
h2o.target_encode_create, 201, 202
h2o.trn (h2o.metric), 137
h2o.toFrame, 203
h2o.tokenize, 204
h2o.toLower, 205
h2o.topN, 205
h2o.tot_withinss, 121, 206
h2o.totss, 121, 206
h2o.toupper, 207
h2o.tpr (h2o.metric), 137
h2o.transform, 207
h2o.trunc, 209
h2o.unique, 209
h2o.uploadFile (h2o.importFile), 107
h2o.var, 182, 210
h2o.varimp, 97, 211
h2o.varimp_plot, 191, 211
h2o.week, 212
h2o.weights, 213
h2o.which, 213
h2o.which_max, 214
h2o.which_min, 214
h2o.withinss, 121, 215
h2o.word2vec, 215
h2o.xgboost, 216
h2o.xgboost.available, 220
h2o.year, 141, 220
H20AutoEncoderMetrics-class
(H20ModelMetrics-class), 228
H20AutoEncoderModel, 24
H20AutoEncoderModel-class
(H20Model-class), 227
H20AutoML, 31, 86, 87, 237
H20AutoML-class, 221
H20BinomialMetrics, 28, 41, 42, 81, 92, 127, 133, 134, 138, 142, 175
H20BinomialMetrics-class
(H20ModelMetrics-class), 228
H20BinomialModel, 96, 145
H20BinomialModel-class
(H20Model-class), 227
H20ClusteringMetrics-class
(H20ModelMetrics-class), 228
H20ClusteringModel, 20, 32, 34, 35, 38, 121, 150, 206, 215
H20ClusteringModel-class, 221
H20Connection, 37, 87
H20Connection (H20Connection-class), 222
H20Connection-class, 222
H20ConnectionMutableState, 223
H20CoxPHMetrics-class
(H20ModelMetrics-class), 228
H20CoxPHModel (H20CoxPHModel-class), 223
H20CoxPHModel-class, 223
H20CoxPHModelSummary
(H20CoxPHModelSummary-class), 224
H20CoxPHModelSummary-class, 224
H20DimReductionMetrics-class
(H20ModelMetrics-class), 228
H20DimReductionModel, 99, 158, 160, 170, 183, 198
H20DimReductionModel-class
(H20Model-class), 227
H20Frame-class, 224
H20Frame-Extract, 225
H20Grid (H20Grid-class), 226
H20Grid-class, 226
H20Model, 22, 32, 38, 39, 41, 42, 48, 49, 57, 77, 81, 88, 90, 96, 105, 124, 125,
129, 130, 134, 149, 150, 154, 155, 162, 166, 173, 174, 176, 178–180, 182, 211, 213, 226, 228, 233, 235, 238, 239, 243
H20Model (H20Model-class), 227
H20Model-class, 227
H20ModelFuture-class, 227
H20ModelMetrics, 22, 32, 41, 42, 81, 82, 127,
130, 138, 142, 149, 150, 156, 173–175, 213
H20ModelMetrics
(H20ModelMetrics-class), 228
H20ModelMetrics-class, 228
H20MultinomialMetrics, 41, 127, 142, 175
H20MultinomialMetrics-class
(H20ModelMetrics-class), 228
H20MultinomialModel, 145
H20MultinomialModel-class
(H20Model-class), 227
H20OrdinalMetrics-class
(H20ModelMetrics-class), 228
H20OrdinalModel-class (H20Model-class), 227
H20RegressionMetrics, 142, 175
H2ORegressionMetrics-class (H2OModelMetrics-class), 228
H2ORegressionModel, 96
H2ORegressionModel-class (H2OModel-class), 227
H2OUNknownMetrics-class (H2OModelMetrics-class), 228
H2OUNknownModel-class (H2OModel-class), 227
H2OWordEmbeddingMetrics-class (H2OModelMetrics-class), 228
H2OWordEmbeddingModel-class (H2OModel-class), 227
head.H2OFrame (h2o.head), 104
hour (h2o.hour), 106
housevotes, 229
ifelse (h2o.ifelse), 106
iris, 229
is.character, 117, 230
is.factor, 118, 230
is.h2o, 230
is.na.H2OFrame (Ops.H2OFrame), 233
is.numeric, 118, 231
kurtosis.H2OFrame (h2o.kurtosis), 122
length.H2OFrame (Ops.H2OFrame), 233
levels, 123
log, 125
log (Ops.H2OFrame), 233
log10, 126
log10 (Ops.H2OFrame), 233
log1p, 126
log1p (Ops.H2OFrame), 233
log2, 126
log2 (Ops.H2OFrame), 233
Logical-or, 231
logLik.H2OCoxPHModel
  (H2OCoxPHModel-class), 223
match, 131
match.H2OFrame (h2o.match), 131
Math.H2OFrame (Ops.H2OFrame), 233
max, 132
mean, 133
mean.H2OFrame (h2o.mean), 132
median.H2OFrame (h2o.median), 135
min, 139
ModelAccessors, 232
month (h2o.month), 141
names, 146
names.H2OFrame, 233
names<-.H2OFrame (Ops.H2OFrame), 233
ncol, 147
ncol.H2OFrame (Ops.H2OFrame), 233
nlevels, 148
nrow, 149
nrow.H2OFrame (Ops.H2OFrame), 233
Ops.H2OFrame, 233
plot.H2OModel, 235
plot.H2OTabulate, 236
predict, 42, 81, 82
predict.H2OAutoML, 237
predict.H2OModel, 64, 86, 97, 166, 238
predict_leaf_node_assignment.H2OModel, 238
print.H2OFrame, 239
print.H2OTable, 240
prod, 160
prostate, 240
quantile, 162
quantile.H2OFrame (h2o.quantile), 161
range, 167
range.H2OFrame, 241
rbind, 169
round, 177
round (h2o.round), 176
rowMeans, 133
scale.H2OFrame (h2o.scale), 181
sd, 182
sd (h2o.sd), 182
show, H2OAutoEncoderMetrics-method (H2OModelMetrics-class), 228
show, H2OBinomialMetrics-method (H2OModelMetrics-class), 228
show, H2OClusteringMetrics-method (H2OModelMetrics-class), 228
show, H2OConnection-method (H2OConnection-class), 222
show, H2OCoxPHModel-method (H2OCoxPHModel-class), 223
show,H2Ocoxphmodelsummary-method
(H2OcoxphmodelSummary-class), 224
show,H2Odimreductionmetrics-method
(H2OModelMetrics-class), 228
show,H2Ogrid-method (H2Ogrid-class), 226
show,H2Omodel-method (H2OModel-class), 227
show,H2OModelMetrics-method
(H2OModelMetrics-class), 228
show,H2Omultinomialmetrics-method
(H2OModelMetrics-class), 228
show,H2Oordinalmetrics-method
(H2OModelMetrics-class), 228
show,H2Oregressionmetrics-method
(H2OModelMetrics-class), 228
signif, 186
signif (h2o.signif), 185
sin, 186
skewness.H2Oframe (h2o.skewness), 186
sqrt, 188
str.H2Oframe, 241
sum, 195
summary, 196
summary,H2Ocoxphmodel-method, 242
summary,H2Ogrid-method, 242
summary,H2Omodel-method, 243
Summary.H2Oframe (Ops.H2Oframe), 233
summary.H2Oframe (h2o.summary), 196
survfit.H2Ocoxphmodel
(H2Ocoxphmodel-class), 223
t.H2Oframe (Ops.H2Oframe), 233
table.H2Oframe (h2o.table), 198
tail.H2Oframe (h2o.head), 104
tan, 200
tanh, 200
trunc, 209
trunc (Ops.H2Oframe), 233
use.package, 12, 14, 243
var, 210
var (h2o.var), 210
vcov.H2Ocoxphmodel
(H2Ocoxphmodel-class), 223
walking, 244
week (h2o.week), 212

which, 213
which.max, 214
which.max.H2Oframe (h2o.which_max), 214
which.min, 215
which.min.H2Oframe (h2o.which_max), 214
year (h2o.year), 220

zzz, 244