R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2o-package</td>
<td>8</td>
</tr>
<tr>
<td>.addParm</td>
<td>9</td>
</tr>
<tr>
<td>.collapse</td>
<td>10</td>
</tr>
<tr>
<td>h2o.doGET</td>
<td>10</td>
</tr>
<tr>
<td>h2o.doPOST</td>
<td>11</td>
</tr>
<tr>
<td>h2o.doRawGET</td>
<td>11</td>
</tr>
<tr>
<td>h2o.doRawPOST</td>
<td>12</td>
</tr>
<tr>
<td>h2o.doSafeGET</td>
<td>13</td>
</tr>
<tr>
<td>h2o.doSafePOST</td>
<td>14</td>
</tr>
<tr>
<td>h2o.is_progress</td>
<td>14</td>
</tr>
<tr>
<td>h2o.locate</td>
<td>15</td>
</tr>
<tr>
<td>h2o.primitives</td>
<td>15</td>
</tr>
<tr>
<td>h2o.__ALL_CAPABILITIES</td>
<td>15</td>
</tr>
<tr>
<td>h2o.__checkConnectionHealth</td>
<td>16</td>
</tr>
<tr>
<td>h2o.__CREATE_FRAME</td>
<td>16</td>
</tr>
<tr>
<td>h2o.__DECRYPTION_SETUP</td>
<td>16</td>
</tr>
<tr>
<td>h2o.__DKV</td>
<td>17</td>
</tr>
<tr>
<td>h2o.__EXPORT_FILES</td>
<td>17</td>
</tr>
<tr>
<td>h2o.__FRAMES</td>
<td>17</td>
</tr>
<tr>
<td>h2o.__IMPORT</td>
<td>18</td>
</tr>
<tr>
<td>h2o.__JOBS</td>
<td>18</td>
</tr>
<tr>
<td>h2o.__LOGANDECHO</td>
<td>18</td>
</tr>
<tr>
<td>h2o.__MODELS</td>
<td>19</td>
</tr>
<tr>
<td>h2o.__MODEL_BUILDERS</td>
<td>19</td>
</tr>
<tr>
<td>h2o.__MODEL_METRICS</td>
<td>19</td>
</tr>
<tr>
<td>h2o.__PARSE_SETUP</td>
<td>20</td>
</tr>
<tr>
<td>h2o.__RAPIDS</td>
<td>20</td>
</tr>
<tr>
<td>h2o.__REST_API_VERSION</td>
<td>20</td>
</tr>
<tr>
<td>h2o.__SEGMENT_MODELS_BUILDERS</td>
<td>21</td>
</tr>
<tr>
<td>h2o.__W2V_SYNONYMS</td>
<td>21</td>
</tr>
<tr>
<td>.pkg.env</td>
<td>21</td>
</tr>
<tr>
<td>.skip_if_not_developer</td>
<td>22</td>
</tr>
<tr>
<td>.verify_dataxy</td>
<td>22</td>
</tr>
<tr>
<td>aaa</td>
<td>22</td>
</tr>
<tr>
<td>apply</td>
<td>23</td>
</tr>
<tr>
<td>as.character.H2OFrame</td>
<td>23</td>
</tr>
<tr>
<td>as.data.frame.H2OFrame</td>
<td>24</td>
</tr>
</tbody>
</table>
R topics documented:

as.data.frame.H2OSegmentModels ........................................ 25
as.factor ................................................................. 26
as.h2o ................................................................. 26
as.matrix.H2OFrame ...................................................... 28
as.numeric ............................................................... 28
as.vector.H2OFrame ........................................................ 29
australia ................................................................. 30
colnames ................................................................. 30
dim.H2OFrame ............................................................ 31
dimnames.H2OFrame ...................................................... 31
feature_frequencies.H2OModel ............................................ 32
generate_col_ind ........................................................ 33
get_seed.H2OModel ...................................................... 33
h2o.abs ................................................................. 34
h2o.acos ................................................................. 35
h2o.aggregated_frame .................................................. 35
h2o.aggregator ......................................................... 36
h2o.aic ................................................................. 37
h2o.all ................................................................. 38
h2o.anomaly ............................................................. 39
h2o.any ................................................................. 40
h2o.anyFactor .......................................................... 40
h2o.arrange ............................................................. 41
h2o.ascharacter ........................................................ 41
h2o.asfactor ............................................................ 42
h2o.asnumeric .......................................................... 43
h2o.assign .............................................................. 43
h2o.as_date .............................................................. 44
h2o.auc ................................................................. 44
h2o.aucpr ............................................................... 45
h2o.automl .............................................................. 46
h2o.betweenss .......................................................... 50
h2o.biases .............................................................. 51
h2o.bottomN ............................................................. 51
h2o.cbind ............................................................... 52
h2o.ceiling .............................................................. 53
h2o.centers .............................................................. 54
h2o.centersSTD ........................................................ 54
h2o.centroid_stats ...................................................... 55
h2o.clearLog ............................................................ 55
h2o.clusterInfo ........................................................ 56
h2o.clusterIsUp ........................................................ 56
h2o.clusterStatus ...................................................... 57
h2o.cluster_sizes ........................................................ 57
h2o.coef ............................................................... 58
h2o.coef_norm ........................................................... 58
h2o.colnames ........................................................... 59
h2o.columns_by_type .................................................... 59
h2o.computeGram ....................................................... 61
h2o.confusionMatrix .................................................... 61
h2o.confusionMatrix .................................................... 61
h2o.connect ............................................................. 63
h2o.cor ............................................................... 64
**R topics documented:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2o.cos</td>
<td>65</td>
</tr>
<tr>
<td>h2o.cosh</td>
<td>66</td>
</tr>
<tr>
<td>h2o.coxph</td>
<td>66</td>
</tr>
<tr>
<td>h2o.createFrame</td>
<td>68</td>
</tr>
<tr>
<td>h2o.cross_validation_fold_assignment</td>
<td>70</td>
</tr>
<tr>
<td>h2o.cross_validation_holdout_predictions</td>
<td>71</td>
</tr>
<tr>
<td>h2o.cross_validation_models</td>
<td>71</td>
</tr>
<tr>
<td>h2o.cross_validation_predictions</td>
<td>72</td>
</tr>
<tr>
<td>h2o.cummax</td>
<td>73</td>
</tr>
<tr>
<td>h2o.cummin</td>
<td>74</td>
</tr>
<tr>
<td>h2o.cumprod</td>
<td>74</td>
</tr>
<tr>
<td>h2o.cumsum</td>
<td>75</td>
</tr>
<tr>
<td>h2o.cut</td>
<td>76</td>
</tr>
<tr>
<td>h2o.day</td>
<td>77</td>
</tr>
<tr>
<td>h2o.dayOfWeek</td>
<td>78</td>
</tr>
<tr>
<td>h2o.dct</td>
<td>78</td>
</tr>
<tr>
<td>h2o.ddply</td>
<td>79</td>
</tr>
<tr>
<td>h2o.decryptionSetup</td>
<td>80</td>
</tr>
<tr>
<td>h2o.deepfeatures</td>
<td>81</td>
</tr>
<tr>
<td>h2o.deeplearning</td>
<td>82</td>
</tr>
<tr>
<td>h2o.describe</td>
<td>89</td>
</tr>
<tr>
<td>h2o.difflag1</td>
<td>90</td>
</tr>
<tr>
<td>h2o.dim</td>
<td>90</td>
</tr>
<tr>
<td>h2o.dimnames</td>
<td>91</td>
</tr>
<tr>
<td>h2o.distance</td>
<td>92</td>
</tr>
<tr>
<td>h2o.downloadAllLogs</td>
<td>92</td>
</tr>
<tr>
<td>h2o.downloadCSV</td>
<td>93</td>
</tr>
<tr>
<td>h2o.download_model</td>
<td>94</td>
</tr>
<tr>
<td>h2o.download_mojo</td>
<td>94</td>
</tr>
<tr>
<td>h2o.download_pojo</td>
<td>95</td>
</tr>
<tr>
<td>h2o.entropy</td>
<td>96</td>
</tr>
<tr>
<td>h2o.exp</td>
<td>97</td>
</tr>
<tr>
<td>h2o.exportFile</td>
<td>97</td>
</tr>
<tr>
<td>h2o.exportHDFS</td>
<td>98</td>
</tr>
<tr>
<td>h2o.fillna</td>
<td>99</td>
</tr>
<tr>
<td>h2o.filterNACols</td>
<td>100</td>
</tr>
<tr>
<td>h2o.findSynonyms</td>
<td>100</td>
</tr>
<tr>
<td>h2o.find_row_by_threshold</td>
<td>101</td>
</tr>
<tr>
<td>h2o.find_threshold_by_max_metric</td>
<td>102</td>
</tr>
<tr>
<td>h2o.floor</td>
<td>102</td>
</tr>
<tr>
<td>h2o.flow</td>
<td>103</td>
</tr>
<tr>
<td>h2o.gainsLift</td>
<td>103</td>
</tr>
<tr>
<td>h2o.gam</td>
<td>104</td>
</tr>
<tr>
<td>h2o.gbm</td>
<td>109</td>
</tr>
<tr>
<td>h2o.generic</td>
<td>114</td>
</tr>
<tr>
<td>h2o.genericModel</td>
<td>115</td>
</tr>
<tr>
<td>h2o.getConnection</td>
<td>116</td>
</tr>
<tr>
<td>h2o.getFrame</td>
<td>116</td>
</tr>
<tr>
<td>h2o.getGLMFullRegularizationPath</td>
<td>117</td>
</tr>
<tr>
<td>h2o.getGrid</td>
<td>118</td>
</tr>
<tr>
<td>h2o.getId</td>
<td>118</td>
</tr>
<tr>
<td>h2o.getModel</td>
<td>119</td>
</tr>
</tbody>
</table>
R topics documented:

h2o.getModelTree ......................................................... 120
h2o.getTimezone .......................................................... 120
h2o.getTypes .............................................................. 121
h2o.getVersion ............................................................. 121
h2o.get_automl ............................................................. 122
h2o.get_leaderboard .................................................... 122
h2o.get_ntrees_actual .................................................. 123
h2o.get_segment_models .............................................. 124
h2o.giniCoef .............................................................. 124
h2o.glm ................................................................. 125
h2o.glrm ................................................................. 131
h2o.grep ................................................................. 133
h2o.grid ................................................................. 134
h2o.group_by ............................................................. 136
h2o.gsub ................................................................. 137
h2o.head ................................................................. 138
h2o.HGLMMetrics .......................................................... 139
h2o.hist ................................................................. 139
h2o.hit_ratio_table .................................................... 140
h2o.hour ................................................................. 141
h2o.ifelse ............................................................... 141
h2o.importFile ............................................................ 142
h2o.import_hive_table .................................................. 145
h2o.import_mojo ........................................................... 145
h2o.import_sql_select .................................................. 146
h2o.import_sql_table .................................................... 147
h2o.impute ............................................................... 148
h2o.init ................................................................. 149
h2o.insertMissingValues ............................................... 152
h2o.interaction .......................................................... 153
h2o.isax ................................................................. 155
h2o.ischaracter .......................................................... 156
h2o.isfactor ............................................................. 156
h2o.isnumeric ............................................................ 157
h2o.isolationForest ..................................................... 158
h2o.is_client ............................................................ 160
h2o.keyof ............................................................... 160
h2o.kfold_column ........................................................ 161
h2o.killMinus3 ........................................................... 161
h2o.kmeans .............................................................. 162
h2o.kurtosis .............................................................. 164
h2o.levels ............................................................... 165
h2o.listTimezones ....................................................... 165
h2o.list_all_extensions ................................................ 166
h2o.list_api_extensions ................................................ 166
h2o.list_core_extensions ............................................. 166
h2o.list_jobs ............................................................ 166
h2o.loadGrid ............................................................. 167
h2o.loadModel ........................................................... 167
h2o.log ................................................................. 168
h2o.log10 ............................................................... 169
h2o.log1p ............................................................... 169
R topics documented:

h2o.log2 ........................................... 170
h2o.logAndEcho .................................... 171
h2o.logloss ........................................ 171
h2o.ls ............................................. 172
h2o.lstrip ......................................... 173
h2o.mae ............................................ 173
h2o.makeGLMModel .................................. 174
h2o.make_metrics ................................... 174
h2o.match ........................................... 175
h2o.max ............................................ 176
h2o.mean ............................................ 177
h2o.mean_per_class_error ............................. 178
h2o.mean_residual_deviance ......................... 179
h2o.median ........................................... 179
h2o.melt ............................................ 180
h2o.merge ........................................... 181
h2o.metric .......................................... 182
h2o.min ............................................. 184
h2o.mktime ......................................... 184
h2o.mojo_predict_csv ................................. 185
h2o.mojo_predict_df ................................. 186
h2o.month .......................................... 187
h2o.mse ............................................. 187
h2o.nacnt .......................................... 188
h2o.naiveBayes ..................................... 189
h2o.names .......................................... 191
h2o.na_omit ....................................... 192
h2o.nchar .......................................... 193
h2o.ncol ............................................ 193
h2o.networkTest .................................... 194
h2o.nlevels ........................................ 194
h2o.no_progress ................................... 195
h2o.nrow ............................................ 196
h2o.null_deviance .................................. 196
h2o.null_dof ...................................... 197
h2o.num_iterations ................................ 198
h2o.num_valid_substrings ............................ 198
h2o.openLog ....................................... 199
h2o.parseRaw ...................................... 199
h2o.parseSetup .................................... 201
h2o.partialPlot .................................... 202
h2o.performance .................................... 203
h2o.pivot .......................................... 205
h2o.prcomp ........................................ 205
h2o.predict ....................................... 207
h2o.predict_json ................................... 208
h2o.print .......................................... 209
h2o.prod ............................................ 209
h2o.prov_archetypes ................................ 210
h2o.psvm .......................................... 211
h2o.quantile ...................................... 212
h2o.r2 ............................................. 214
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2o.randomForest</td>
<td>214</td>
</tr>
<tr>
<td>h2o.range</td>
<td>219</td>
</tr>
<tr>
<td>h2o.rank_within_group_by</td>
<td>220</td>
</tr>
<tr>
<td>h2o.rbind</td>
<td>222</td>
</tr>
<tr>
<td>h2o.reconstruct</td>
<td>223</td>
</tr>
<tr>
<td>h2o.relevel</td>
<td>224</td>
</tr>
<tr>
<td>h2o.removeAll</td>
<td>224</td>
</tr>
<tr>
<td>h2o.removeVecs</td>
<td>225</td>
</tr>
<tr>
<td>h2o.rep_len</td>
<td>226</td>
</tr>
<tr>
<td>h2o.reset_threshold</td>
<td>226</td>
</tr>
<tr>
<td>h2o.residual_deviance</td>
<td>227</td>
</tr>
<tr>
<td>h2o.residual_dof</td>
<td>228</td>
</tr>
<tr>
<td>h2o.rm</td>
<td>228</td>
</tr>
<tr>
<td>h2o.rmse</td>
<td>229</td>
</tr>
<tr>
<td>h2o.rmsle</td>
<td>230</td>
</tr>
<tr>
<td>h2o.round</td>
<td>231</td>
</tr>
<tr>
<td>h2o.rstrip</td>
<td>231</td>
</tr>
<tr>
<td>h2o.runif</td>
<td>232</td>
</tr>
<tr>
<td>h2o.saveGrid</td>
<td>233</td>
</tr>
<tr>
<td>h2o.saveModel</td>
<td>234</td>
</tr>
<tr>
<td>h2o.saveModelDetails</td>
<td>234</td>
</tr>
<tr>
<td>h2o.saveMojo</td>
<td>235</td>
</tr>
<tr>
<td>h2o.save_mojo</td>
<td>236</td>
</tr>
<tr>
<td>h2o.scale</td>
<td>236</td>
</tr>
<tr>
<td>h2o.scoreHistory</td>
<td>237</td>
</tr>
<tr>
<td>h2o.sd</td>
<td>238</td>
</tr>
<tr>
<td>h2o.sdev</td>
<td>239</td>
</tr>
<tr>
<td>h2o.setLevels</td>
<td>239</td>
</tr>
<tr>
<td>h2o.setTimezone</td>
<td>240</td>
</tr>
<tr>
<td>h2o.set_s3_credentials</td>
<td>240</td>
</tr>
<tr>
<td>h2o.show_progress</td>
<td>241</td>
</tr>
<tr>
<td>h2o.shutdown</td>
<td>242</td>
</tr>
<tr>
<td>h2o.signif</td>
<td>243</td>
</tr>
<tr>
<td>h2o.sin</td>
<td>243</td>
</tr>
<tr>
<td>h2o.skewness</td>
<td>244</td>
</tr>
<tr>
<td>h2o.splitFrame</td>
<td>245</td>
</tr>
<tr>
<td>h2o.sqrt</td>
<td>245</td>
</tr>
<tr>
<td>h2o.stackedEnsemble</td>
<td>246</td>
</tr>
<tr>
<td>h2o.start Logging</td>
<td>248</td>
</tr>
<tr>
<td>h2o.std_coef_plot</td>
<td>249</td>
</tr>
<tr>
<td>h2o.stopLogging</td>
<td>250</td>
</tr>
<tr>
<td>h2o.str</td>
<td>251</td>
</tr>
<tr>
<td>h2o.stringdist</td>
<td>251</td>
</tr>
<tr>
<td>h2o.strsplit</td>
<td>252</td>
</tr>
<tr>
<td>h2o.sub</td>
<td>253</td>
</tr>
<tr>
<td>h2o.substring</td>
<td>253</td>
</tr>
<tr>
<td>h2o.sum</td>
<td>254</td>
</tr>
<tr>
<td>h2o.summary</td>
<td>255</td>
</tr>
<tr>
<td>h2o.svd</td>
<td>256</td>
</tr>
<tr>
<td>h2o.table</td>
<td>257</td>
</tr>
<tr>
<td>h2o.tabulate</td>
<td>258</td>
</tr>
<tr>
<td>h2o.tan</td>
<td>259</td>
</tr>
<tr>
<td>R topics documented:</td>
<td>7</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>h2o.tanh</td>
<td>260</td>
</tr>
<tr>
<td>h2o.targetencoder</td>
<td>260</td>
</tr>
<tr>
<td>h2o.target_encode_apply</td>
<td>262</td>
</tr>
<tr>
<td>h2o.target_encode_create</td>
<td>263</td>
</tr>
<tr>
<td>h2o.target_encode_fit</td>
<td>264</td>
</tr>
<tr>
<td>h2o.target_encode_transform</td>
<td>265</td>
</tr>
<tr>
<td>h2o.toFrame</td>
<td>267</td>
</tr>
<tr>
<td>h2o.tokenize</td>
<td>267</td>
</tr>
<tr>
<td>h2o_tolower</td>
<td>268</td>
</tr>
<tr>
<td>h2o.topN</td>
<td>268</td>
</tr>
<tr>
<td>h2o.totss</td>
<td>269</td>
</tr>
<tr>
<td>h2o.tot_withinss</td>
<td>270</td>
</tr>
<tr>
<td>h2o.toupper</td>
<td>271</td>
</tr>
<tr>
<td>h2o.train_segments</td>
<td>271</td>
</tr>
<tr>
<td>h2o.transform</td>
<td>272</td>
</tr>
<tr>
<td>h2o.transform.H2OTargetEncoderModel-method</td>
<td>273</td>
</tr>
<tr>
<td>h2o.transform.H2OWordEmbeddingModel-method</td>
<td>274</td>
</tr>
<tr>
<td>h2o.transform_word2vec</td>
<td>274</td>
</tr>
<tr>
<td>h2o.trim</td>
<td>275</td>
</tr>
<tr>
<td>h2o.trunc</td>
<td>276</td>
</tr>
<tr>
<td>h2o.unique</td>
<td>277</td>
</tr>
<tr>
<td>h2o.upload_model</td>
<td>277</td>
</tr>
<tr>
<td>h2o.upload_mojo</td>
<td>278</td>
</tr>
<tr>
<td>h2o.var</td>
<td>279</td>
</tr>
<tr>
<td>h2o.varimp</td>
<td>279</td>
</tr>
<tr>
<td>h2o.varimp_plot</td>
<td>280</td>
</tr>
<tr>
<td>h2o.varsplit</td>
<td>281</td>
</tr>
<tr>
<td>h2o.week</td>
<td>281</td>
</tr>
<tr>
<td>h2o.weights</td>
<td>282</td>
</tr>
<tr>
<td>h2o.which</td>
<td>283</td>
</tr>
<tr>
<td>h2o.which_max</td>
<td>284</td>
</tr>
<tr>
<td>h2o.which_min</td>
<td>285</td>
</tr>
<tr>
<td>h2o.withinss</td>
<td>286</td>
</tr>
<tr>
<td>h2o.word2vec</td>
<td>286</td>
</tr>
<tr>
<td>h2o.xgboost</td>
<td>287</td>
</tr>
<tr>
<td>h2o.xgboost.available</td>
<td>293</td>
</tr>
<tr>
<td>h2o.year</td>
<td>293</td>
</tr>
<tr>
<td>H2OAutoML-class</td>
<td>294</td>
</tr>
<tr>
<td>H2OClusteringModel-class</td>
<td>294</td>
</tr>
<tr>
<td>H2OConnection-class</td>
<td>295</td>
</tr>
<tr>
<td>H2OConnectionMutableState</td>
<td>296</td>
</tr>
<tr>
<td>H2OCoxPHModel-class</td>
<td>296</td>
</tr>
<tr>
<td>H2OCoxPHModelSummary-class</td>
<td>297</td>
</tr>
<tr>
<td>H2OFrame-class</td>
<td>297</td>
</tr>
<tr>
<td>H2OFrame-Extract</td>
<td>297</td>
</tr>
<tr>
<td>H2OGrid-class</td>
<td>298</td>
</tr>
<tr>
<td>H2OLeafNode-class</td>
<td>299</td>
</tr>
<tr>
<td>H2OModel-class</td>
<td>300</td>
</tr>
<tr>
<td>H2OModelFuture-class</td>
<td>300</td>
</tr>
<tr>
<td>H2OModelMetrics-class</td>
<td>301</td>
</tr>
<tr>
<td>H2ONode-class</td>
<td>301</td>
</tr>
<tr>
<td>H2OSegmentModels-class</td>
<td>302</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: Erin LeDell <erin@h2o.ai>

**References**

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

---

**.addParm**

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

**Description**

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

**Usage**

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

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

Details

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

---

### .collapse

**Helper Collapse Function**

**Description**

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

**Usage**

```
collapse(v)
```

**Arguments**

- **v**: Character vector.

---

### .h2o.doGET

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

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

A list object as described above
**.h2o.doPOST**  
*Just like doRawPOST but fills in the default h2oRestApiVersion if none is provided*

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

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

**Arguments**

- **h2oRestApiVersion**  
  (Optional) A version number to prefix to the urlSuffix. If no version is provided, a default version is chosen for you.

- **urlSuffix**  
  The partial URL suffix to add to the calculated base URL for the instance

- **parms**  
  (Optional) Parameters to include in the request

- **...**  
  (Optional) Additional parameters.

**Value**

A list object as described above

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

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

**Usage**

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

Perform a low-level HTTP POST operation on an H2O instance

Description

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

Usage

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

Arguments

- **conn**
  H2OConnection

- **h2oRestApiVersion**
  (Optional) A version number to prefix to the urlSuffix. If no version is provided, the version prefix is skipped.

- **urlSuffix**
  The partial URL suffix to add to the calculated base URL for the instance

- **parms**
  (Optional) Parameters to include in the request

- **fileUploadInfo**
  (Optional) Information to POST (NOTE: changes Content-type from XXX-www-url-encoded to multi-part). Use fileUpload(normalizePath("/path/to/file")).

- **...**
  (Optional) Additional parameters.

Details

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

Value

A list object as described above
## .h2o.doSafeGET

**Details**

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

**Value**

A list object as described above

---

### .h2o.doSafeGET

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

---

**Description**

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

**Usage**

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

**Arguments**

- `h2oRestApiVersion`  
  (Optional) A version number to prefix to the `urlSuffix`. If no version is provided, a default version is chosen for you.

- `urlSuffix`  
  The partial URL suffix to add to the calculated base URL for the instance

- `parms`  
  (Optional) Parameters to include in the request

- `...`  
  (Optional) Additional parameters.

**Value**

The raw response payload as a character vector
.h2o.doSafePOST  **Perform a safe (i.e. error-checked) HTTP POST request to an H2O cluster.**

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

The raw response payload as a character vector

---

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

**Description**

Check if Progress Bar is Enabled

**Usage**

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

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

**Description**

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

**Usage**

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

**Arguments**

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

---

.h2o.primitives

*Map of operations known to H2O*

**Description**

Map of operations known to H2O.

**Usage**

```
.h2o.primitives
```

**Format**

An object of class character of length 39.

---

.h2o.__ALL_CAPABILITIES

*Capabilities endpoints*

**Description**

Capabilities endpoints.

**Usage**

```
.h2o.__ALL_CAPABILITIES
```

**Format**

An object of class character of length 1.
.h2o.__checkConnectionHealth

*Check H2O Server Health*

**Description**
Warn if there are sick nodes.

**Usage**
.h2o.__checkConnectionHealth()

---

.h2o.__CREATE_FRAME

*H2OFrame Manipulation*

**Description**
H2OFrame Manipulation

**Usage**
.h2o.__CREATE_FRAME

**Format**
An object of class character of length 1.

---

.h2o.__DECRYPTION_SETUP

*Decryption Endpoints*

**Description**
Decryption Endpoints

**Usage**
.h2o.__DECRYPTION_SETUP

**Format**
An object of class character of length 1.
**Removal Endpoints**

**Description**
Removal Endpoints

**Usage**
.h2o.__DKV

**Format**
An object of class character of length 1.

---

**Export Files Endpoint Generator**

**Description**
Export Files Endpoint Generator

**Usage**
.h2o.__EXPORT_FILES(frame)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>H2OFrame</td>
</tr>
</tbody>
</table>

---

**Inspect/Summary Endpoints**

**Description**
Inspect/Summary Endpoints

**Usage**
.h2o.__FRAMES

**Format**
An object of class character of length 1.
**h2o.__IMPORT**  
*Import/Export Endpoints*

**Description**
Import/Export Endpoints

**Usage**
```
h2o.__IMPORT
```

**Format**
An object of class character of length 1.

---

**h2o.__JOBS**  
*Administrative Endpoints*

**Description**
Administrative Endpoints

**Usage**
```
h2o.__JOBS
```

**Format**
An object of class character of length 1.

---

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

**Description**
Log and Echo Endpoint

**Usage**
```
h2o.__LOGANDECHO
```

**Format**
An object of class character of length 1.
### Model Endpoint

**Description**
Model Endpoint

**Usage**
```
h2o.__MODELS
```

**Format**
An object of class character of length 1.

### Model Builder Endpoint Generator

**Description**
Model Builder Endpoint Generator

**Usage**
```
h2o.__MODEL_BUILDERS(algo)
```

**Arguments**
- `algo`  
  Cannonical identifier of H2O algorithm.

### Model Metrics Endpoint

**Description**
Model Metrics Endpoint

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

**Arguments**
- `model`  
  H2OModel.
- `data`  
  H2OFrame.
.h2o.__PARSE_SETUP  

**Parse Endpoints**

**Description**
Parse Endpoints

**Usage**
.h2o.__PARSE_SETUP

**Format**
An object of class character of length 1.

---

.h2o.__RAPIDS  

**Rapids Endpoint**

**Description**
Rapids Endpoint

**Usage**
.h2o.__RAPIDS

**Format**
An object of class character of length 1.

---

.h2o.__REST_API_VERSION  

**H2O Package Constants**

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

**Usage**
.h2o.__REST_API_VERSION

**Format**
An object of class integer of length 1.

**Details**
Additionally, environment variables for the H2O package are named here. Endpoint Version.
Segment Models Builder Endpoint Generator

Description
Segment Models Builder Endpoint Generator

Usage
```r
.h2o.__SEGMENT_MODELS_BUILDERS(algo)
```

Arguments
- **algo**: Canonical identifier of H2O algorithm.

Word2Vec Endpoints

Description
Word2Vec Endpoints

Usage
```r
.h2o.__W2V_SYNONYMS
```

Format
An object of class `character` of length 1.

The H2O Package Environment

Description
The H2O Package Environment

Usage
```r
.pkg.env
```

Format
An object of class `environment` of length 4.
Description
Collected here are the various methods used by the h2o-R package to communicate with the H2O backend. There are methods for checking cluster health, polling, and inspecting objects in the H2O store.

Usage
```
.skip_if_not_developer()
```

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

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

Arguments
- **data**: H2OFrame
- **x**: features
- **y**: response
- **autoencoder**: autoencoder flag

Examples
```r
## Not run:
if (Sys.info()
```
apply

Apply on H2O Datasets

Description
Method for apply on H2OFrame objects.

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

Arguments
X
an H2OFrame object on which apply will operate.

MARGIN
the vector on which the function will be applied over, either 1 for rows or 2 for columns.

FUN
the function to be applied.

...
optional arguments to FUN.

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

See Also
apply for the base generic

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

as.character.H2OFrame
Convert an H2OFrame to a String

Description
Convert an H2OFrame to a String

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

Converts parsed H2O data into an R data frame

Description

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

Usage

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

Arguments

x  An H2OFrame object.

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

Details

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

See Also

use.package
**as.data.frame.H2OSegmentModels**

Converts a collection of Segment Models to a data.frame

**Description**

Converts a collection of Segment Models to a data.frame

**Usage**

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

**Arguments**

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

**Value**

Returns data.frame with result of segment model training.

**Examples**

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

## End(Not run)
```

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

## End(Not run)
```
as.factor  

Convert H2O Data to Factors

Description

Convert a column into a factor column.

Usage

as.factor(x)

Arguments

x

a column from an H2OFrame data set.

See Also

as.factor.

Examples

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

as.h2o  

Create H2OFrame

Description

Import R object to the H2O cluster.

Usage

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

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

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

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

## 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.

Details

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

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

References

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

See Also

use.package

Examples

## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
euro_hf <- as.h2o(euro)
letters_hf <- as.h2o(letters)
state_hf <- as.h2o(state.x77)
iris_hf_2 <- as.h2o(iris_hf)
stopifnot(is.h2o(iris_hf), dim(iris_hf) == dim(iris),
is.h2o(euro_hf), dim(euro_hf) == c(length(euro), 1L),
is.h2o(letters_hf), dim(letters_hf) == c(length(letters), 1L),
is.h2o(state_hf), dim(state_hf) == dim(state.x77),
is.h2o(iris_hf_2), dim(iris_hf_2) == dim(iris_hf))

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

## End(Not run)
## as.matrix.H2OFrame

Convert an H2OFrame to a matrix

### Description

Convert an H2OFrame to a matrix

### Usage

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

### Arguments

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

### Examples

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

## as.numeric

Convert H2O Data to Numeric

### Description

Converts an H2O column into a numeric value column.

### Usage

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

### Arguments

- `x`: a column from an H2OFrame data set.
as.vector.H2OFrame

Convert an H2OFrame to a vector

Description

Convert an H2OFrame to a vector

Usage

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

Arguments

x      An H2OFrame object
mode   Mode to coerce vector to

Examples

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

## End(Not run)

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

## End(Not run)
australia  Australia Coastal Data

Description

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

Format

A data frame with 251 rows and 8 columns

colnames  Returns the column names of an H2OFrame

Description

Returns the column names of an H2OFrame

Usage

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

Arguments

x  An H2OFrame object.

do.NULL  logical. If FALSE and names are NULL, names are created.

prefix  for created names.

Examples

## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
colnames(iris_hf)  # Returns "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

## End(Not run)
dim.H2OFrame  

Returns the Dimensions of an H2OFrame

Description

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

Usage

## S3 method for class 'H2OFrame'

dim(x)

Arguments

x  
An H2OFrame object.

See Also

dim for the base R method.

Examples

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

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

## End(Not run)

---

dimnames.H2OFrame  

Column names of an H2OFrame

Description

Set column names of an H2O Frame

Usage

## S3 method for class 'H2OFrame'

dimnames(x)

Arguments

x  
An H2OFrame
```
## Not run:
library(h2o)
h2o.init()

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

## End(Not run)
```

---

**feature_frequencies.H2OModel**

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

**Description**

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

**Usage**

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

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

**Arguments**

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

**Value**

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

**See Also**

- [h2o.gbm](#) and [h2o.randomForest](#) for model generation in h2o.
**generate_col_ind**

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

**Usage**

generate_col_ind(data, by)

**Arguments**

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

---

**get_seed.H2OModel**

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

**Description**

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

**Usage**

get_seed.H2OModel(object)

h2o.get_seed(object)

**Arguments**

object a fitted H2OModel object.

**Value**

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

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

## End(Not run)
```

---

**h2o.abs**

**Compute the absolute value of x**

#### Description

Compute the absolute value of x

#### Usage

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

#### Arguments

- `x`  
  An H2OFrame object.

#### See Also

- `abs` for the base R implementation.

#### Examples

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

## End(Not run)
```
h2o.acos

Compute the arc cosine of x

Description

Compute the arc cosine of x

Usage

h2o.acos(x)

Arguments

x
An H2OFrame object.

See Also

acos for the base R implementation.

Examples

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

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

### h2o.aggregator

**Build an Aggregated Frame**

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,
  export_checkpoints_dir = NULL
)
```

### Arguments

- **training_frame**  
  Id of the training data frame.
- **x**  
  A vector containing the character names of the predictors in the model.
- **model_id**  
  Destination id for this model; auto-generated if not specified.
- **ignore_const_cols**  
  Logical. Ignore constant columns. Defaults to TRUE.
- **target_num_exemplars**  
  Targeted number of exemplars Defaults to 5000.
**rel_tol_num_exemplars**
Relative tolerance for number of exemplars (e.g., 0.5 is +/- 50 percent). 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.

**export_checkpoints_dir**
Automatically export generated models to this directory.

**Examples**

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

---

### h2o.aic

**Retrieve the Akaike information criterion (AIC) value**

**Description**

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

**Usage**

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

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

### Arguments

- **x**
  
  An H2OFrame object.

### See Also

- `all` for the base R implementation.

---

**Arguments**

- **object**
  
  An H2OModel or H2OModelMetrics.

- **train**
  
  Retrieve the training AIC

- **valid**
  
  Retrieve the validation AIC

- **xval**
  
  Retrieve the cross-validation AIC

### Examples

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
p.sid <- h2o.runif(prostate)
prostate_train <- prostate[p.sid > .2,]
prostate_glm <- h2o.glm(x=3:7, y=2, training_frame=prostate_train)
aic_basic <- h2o.aic(prostate_glm)
print(aic_basic)
## End(Not run)
```
Anomaly Detection via H2O Deep Learning Model

**Description**

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

**Usage**

```r
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**

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

```r
code
```

**Arguments**

- `x` An H2OFrame object.

**See Also**

- `all` for the base R implementation.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
iris <- h2o.importFile(f)
h2o.any(iris[,1] < 1000)
## End(Not run)
```

### h2o.anyFactor

**Check H2OFrame columns for factors**

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

**Usage**

```r
code
```

**Arguments**

- `x` An H2OFrame object.

**Value**

Returns a logical value indicating whether any of the columns in `x` are factors.
h2o.arrange

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

## End(Not run)

h2o.arrange

Sorts an H2O frame by columns

Description

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

Usage

h2o.arrange(x, ...)

Arguments

x
The H2OFrame input to be sorted.

...
The column names to sort by.

Examples

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

iris <- h2o.importFile(f)
h2o.arrange(iris, "species","petal_len","petal_wid")

## End(Not run)

h2o.ascharacter

Convert H2O Data to Characters

Description

Convert H2O Data to Characters

Usage

h2o.ascharacter(x)
Arguments

x  An H2OFrame object.

Examples

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

iris <- h2o.importFile(f)
h2o.ascharacter(iris["species"])

## End(Not run)
**h2o.asnumeric**  
*Convert H2O Data to Numerics*

**Description**  
Convert H2O Data to Numerics

**Usage**

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

**Arguments**

- `x`  
  An H2OFrame object.

**See Also**

- `as.factor` for the base R implementation.

**Examples**

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

cars <- h2o.importFile(f)
h2o.asnumeric(cars)

## End(Not run)
```

**h2o.assign**  
*Rename an H2O object.*

**Description**

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

**Usage**

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

**Arguments**

- `data`  
  An H2OFrame object

- `key`  
  The key to be associated with the H2O parsed data object
Examples

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

cars <- h2o.importFile(f)
dim(cars)
split <- h2o.splitFrame(data = cars, ratios = .8)
train <- h2o.assign(split[[1]], key = "train")
test <- h2o.assign(split[[2]], key = "test")
dim(train)
dim(test)

## End(Not run)
```

---

**h2o.as_date**

*Convert between character representations and objects of Date class*

**Description**

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

**Usage**

```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

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

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

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

## End(Not run)

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

Description

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

Usage

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

Arguments

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

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

Examples

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

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

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

## End(Not run)
seed = NULL,
project_name = NULL,
exclude_algos = NULL,
include_algos = NULL,
modeling_plan = NULL,
exploitation_ratio = 0,
monotone_constraints = NULL,
algo_parameters = NULL,
keep_cross_validation_predictions = FALSE,
keep_cross_validation_models = FALSE,
keep_cross_validation_fold_assignment = FALSE,
sort_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC",
"AUCPR", "mean_per_class_error"),
export_checkpoints_dir = NULL,
verbosity = "warn"
)

Arguments

\texttt{x} \hspace{1cm} A vector containing the names or indices of the predictor variables to use in building the model. If \texttt{x} is missing, then all columns except \texttt{y} are used.

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

\texttt{training_frame} \hspace{1cm} Training frame (H2OFrame or ID).

\texttt{validation_frame} \hspace{1cm} Validation frame (H2OFrame or ID); Optional. This argument is ignored unless the user sets \texttt{nfolds} = 0. If cross-validation is turned off, then a validation frame can be specified and used for early stopping of individual models and early stopping of the grid searches. By default and when \texttt{nfolds} > 1, cross-validation metrics will be used for early stopping and thus validation frame will be ignored.

\texttt{leaderboard_frame} \hspace{1cm} 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.

\texttt{blending_frame} \hspace{1cm} Blending frame (H2OFrame or ID) used to train the the metalearning algorithm in Stacked Ensembles (instead of relying on cross-validated predicted values); Optional. When provided, it also is recommended to disable cross validation by setting \texttt{nfolds}=0 and to provide a leaderboard frame for scoring purposes.

\texttt{nfolds} \hspace{1cm} 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).

\texttt{fold_column} \hspace{1cm} 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.

\texttt{weights_column} \hspace{1cm} 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
This argument specifies the maximum time that the AutoML process will run for, prior to training the final Stacked Ensemble models. If neither 'max_runtime_secs' nor 'max_models' are specified by the user, then 'max_runtime_secs' defaults to 3600 seconds (1 hour).

max_runtime_secs_per_model
Maximum runtime in seconds dedicated to each individual model training process. Use 0 to disable. Defaults to 0.

max_models
Maximum number of models to build in the AutoML process (does not include Stacked Ensembles). Defaults to NULL (no strict limit).

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

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

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

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

project_name
Character string to identify an AutoML project. Defaults to NULL, which means a project name will be auto-generated.

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

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

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

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

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

**algo_parameters**
List. A list of param_name=param_value to be passed to internal models. Defaults to none (Expert usage only). By default, params are set only to algorithms accepting them, and ignored by others. Only following parameters are currently allowed: "monotone_constraints".

**keep_cross_validation_predictions**
Logical. Whether to keep the predictions of the cross-validation predictions. This needs to be set to TRUE if running the same AutoML object for repeated runs because CV predictions are required to build additional Stacked Ensemble models in AutoML. This option defaults to FALSE.

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

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

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

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

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

**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

```r
## Not run:
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)

## End(Not run)
```

### h2o.betweenss

**Get the between cluster sum of squares**

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
 km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.betweenss(km, train = TRUE)

## End(Not run)
```
h2o.biases

Return the respective bias vector

Description

Return the respective bias vector

Usage

h2o.biases(object, vector_id = 1)

Arguments

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

Examples

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

census <- h2o.importFile(f)
census[,1] <- as.factor(census[,1])

dlmodel <- h2o.deeplearning(x = c(1:3), y = 4, training_frame = census,
    hidden = c(17,191),
    epochs = 1,
    balance_classes = FALSE,
    export_weights_and_biases = TRUE)
h2o.biases(dlmodel, vector_id = 1)
## End(Not run)

h2o.bottomN

H2O bottomN

Description

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

Usage

h2o.bottomN(x, column, nPercent)
Arguments

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

Value

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

Examples

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

f2 <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/jira/Bottom20Per.csv.zip"
dataFrame <- h2o.importFile(f1)
bottomAnswer <- h2o.importFile(f2)
nPercent <- c(1, 2, 3, 4)
frameNames <- names(dataFrame)
nP <- nPercent[sample(1:length(nPercent), 1, replace = FALSE)]
colIndex <- sample(1:length(frameNames), 1, replace = FALSE)
h2o.bottomN(dataFrame, frameNames[colIndex], nP)
## End(Not run)
```

---

**h2o.cbind**

*Combine H2O Datasets by Columns*

**Description**

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

**Usage**

`h2o.cbind(...)`

**Arguments**

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

**Value**

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

**See Also**

`cbind` for the base R method.
Examples

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

## End(Not run)
```

Description

`h2o.ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

Usage

`h2o.ceiling(x)`

Arguments

- `x` An H2OFrame object.

See Also

- `ceiling` for the base R implementation.

Examples

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

iris <- h2o.importFile(f)
h2o.ceiling(iris[,1])

## End(Not run)
```
### h2o.centers

**Retrieve the Model Centers**

**Description**
Retrieve the Model Centers

**Usage**

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

**Arguments**
- **object**
  An `H2OClusteringModel` object.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
h2o.ceiling(fr[,1])
## End(Not run)
```

### h2o.centersSTD

**Retrieve the Model Centers STD**

**Description**
Retrieve the Model Centers STD

**Usage**

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

**Arguments**
- **object**
  An `H2OClusteringModel` object.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.centersSTD(km)
## End(Not run)
```
h2o.centroid_stats  Retrieve centroid statistics

Description

Retrieve the centroid statistics. If "train", "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

Examples

## Not run:
library(h2o)
h2o.init()
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.centroid_stats(km, train = TRUE)
## End(Not run)

h2o.clearLog  Delete All H2O R Logs

Description

Clear all H2O R command and error response logs from the local disk. Used primarily for debug-
ging purposes.

Usage

h2o.clearLog()

See Also

h2o.startLogging, h2o.stopLogging, h2o.openLog
Examples

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

## End(Not run)
```

---

**h2o.clusterInfo**  
*Print H2O cluster info*

**Description**

Print H2O cluster info

**Usage**

```r
h2o.clusterInfo()
```

---

**h2o.clusterIsUp**  
*Determine if an H2O cluster is up or not*

**Description**

Determine if an H2O cluster is up or not

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>conn</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2OConnection object</td>
</tr>
</tbody>
</table>

**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**

```r
h2o.clusterStatus()
```

**See Also**

`H2OConnection`, `h2o.init`

**Examples**

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

### h2o.cluster_sizes

**Retrieve the cluster sizes**

**Description**

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

**Usage**

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

**Arguments**

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

```r
## Not run:
library(h2o)
h2o.init()
predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.cluster_sizes(km, train = TRUE)

## End(Not run)
```

---

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

Description

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

Usage

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

Arguments

- `object`: an `H2OModel` object.

Examples

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

cars <- h2o.importFile(f)
predictors <- c("displacement","power","weight","acceleration","year")
response <- "cylinders"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_glm <- h2o.glm(balance_classes = TRUE,
                     seed = 1234,
                     x = predictors,
                     y = response,
                     training_frame = train,
                     validation_frame = valid)
h2o.coef(cars_glm)

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

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

**Description**

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

**Usage**

h2o.coef_norm(object)

**Arguments**

- **object**: an H2OModel object.

**Examples**

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

cars <- h2o.importFile(f)
predictors <- c("displacement","power","weight","acceleration","year")
response <- "cylinders"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_glm <- h2o.glm(balance_classes = TRUE,
                   seed = 1234,
                   x = predictors,
                   y = response,
                   training_frame = train,
                   validation_frame = valid)
h2o.coef(cars_glm)
## End(Not run)
```

---

**h2o.colnames**

*Return column names of an H2OFrame*

**Description**

Return column names of an H2OFrame

**Usage**

h2o.colnames(x)
Arguments

- `x` An H2OFrame object.

See Also

- `colnames` for the base R implementation.

Examples

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

frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.colnames(frame)
## End(Not run)
```

### h2o.columns_by_type

**Obtain a list of columns that are specified by 'coltype'**

#### Description

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

#### Usage

```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)

#### Value

A list of column indices that correspond to "type"
Examples

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

## End(Not run)
```

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

Description

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

h2o.confusionMatrix(object, ...)  

## S4 method for signature 'H2OModel'

h2o.confusionMatrix(object, newdata, valid = FALSE, ...)  

## S4 method for signature 'H2OModelMetrics'

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

Arguments

object Either an H2OModel object or an H2OModelMetrics object.

... Extra arguments for extracting train or valid confusion matrices.

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

valid Retrieve the validation metric.

thresholds (Optional) A value or a list of valid values between 0.0 and 1.0. This value is only used in the case of H2OBinomialMetrics objects.

metrics (Optional) A metric or a list of valid metrics ("min_per_class_accuracy", "absolute_mcc", "tnr", "fwr", "fpr", "tpr", "precision", "accuracy", "f0point5", "f2", "f1"). This value is only used in the case of H2OBinomialMetrics objects.

Details

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

Value

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

See Also

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

Examples

## Not run:

library(h2o)
h2o.init()

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

prostate <- h2o.uploadFile(prostate_path)

prostate[,2] <- as.factor(prostate[,2])

model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")

h2o.confusionMatrix(model, prostate)

# Generating a ModelMetrics object

perf <- h2o.performance(model, prostate)

h2o.confusionMatrix(perf)

## End(Not run)
h2o.connect

Connect to a running H2O instance.

Description

Connect to a running H2O instance.

Usage

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

Arguments

  ip          Object of class character representing the IP address of the server where H2O is running.
  port        Object of class numeric representing the port number of the H2O server.
  strict_version_check
               (Optional) Setting this to FALSE is unsupported and should only be done when advised by technical support.
  proxy       (Optional) A character string specifying the proxy path.
  https       (Optional) Set this to TRUE to use https instead of http.
  cacert      Path to a CA bundle file with root and intermediate certificates of trusted CAs.
  insecure    (Optional) Set this to TRUE to disable SSL certificate checking.
  username    (Optional) Username to login with.
  password    (Optional) Password to login with.
  use_spnego  (Optional) Set this to TRUE to enable SPNEGO authentication.
  cookies     (Optional) Vector(or list) of cookies to add to request.
  context_path
               (Optional) The last part of connection URL: http://<ip>:<port>/<context_path>
  config      (Optional) A list describing connection parameters. Using config makes h2o.connect ignore other parameters and collect named list members instead (see examples).
Value

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

Examples

```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, method = "Pearson")
cor(x, ...)
```

Arguments

- `x` An H2OFrame object.
- `y` NULL (default) or an H2OFrame. The default is equivalent to `y = x`.
- `na.rm` logical. Should missing values be removed?
- `use` An optional character string indicating how to handle missing values. This must be one of the following: "everything" - outputs NaNs whenever one of its contributing observations is missing "all.obs" - presence of missing observations will throw an error "complete.obs" - discards missing values along with all observations in their rows so that only complete observations are used
- `method` str Method of correlation computation. Allowed values are: "Pearson" - Pearson’s correlation coefficient "Spearman" - Spearman’s correlation coefficient (Spearman’s Rho) Defaults to "Pearson"
- `...` Further arguments to be passed down from other methods.
h2o.cos

Examples

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

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

## End(Not run)
```

h2o.cos

Compute the cosine of x

Description

Compute the cosine of x

Usage

h2o.cos(x)

Arguments

x

An H2OFrame object.

See Also

cos for the base R implementation.

Examples

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

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

h2o.cos(frame["C1"])

## End(Not run)
```
h2o.cosh

Compute the hyperbolic cosine of x

Description
Compute the hyperbolic cosine of x

Usage
h2o.cosh(x)

Arguments
x An H2OFrame object.

See Also
cosh for the base R implementation.

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

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

h2o.cosh(frame["C1"])
## End(Not run)

h2o.coxph

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

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

Usage
h2o.coxph(
x, 
event_column, 
training_frame, 
model_id = NULL, 
start_column = NULL, 
stop_column = NULL, 
weights_column = NULL, 

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

test <- h2o.coxph(x = frame["C1"], 
event_column = "C1", 
training_frame = frame, 
model_id = "test")
## End(Not run)
offset_column = NULL,
stratify_by = NULL,
ties = c("efron", "breslow"),
init = 0,
lre_min = 9,
max_iterations = 20,
interactions = NULL,
interaction_pairs = NULL,
interactions_only = NULL,
use_all_factor_levels = FALSE,
export_checkpoints_dir = NULL,
single_node_mode = FALSE
)

Arguments

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

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

training_frame Id of the training data frame.

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

start_column Start Time Column.

stop_column Stop Time Column.

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

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

stratify_by List of columns to use for stratification.

ties Method for Handling Ties. Must be one of: "efron", "breslow". Defaults to efron.

init Coefficient starting value. Defaults to 0.

lre_min Minimum log-relative error. Defaults to 9.

max_iterations Maximum number of iterations. Defaults to 20.

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

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

interactions_only A list of columns that should only be used to create interactions but should not itself participate in model training.
h2o.createFrame

Data H2OFrame Creation in H2O

Description

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

Usage

h2o.createFrame(
  rows = 10000,
  cols = 10,
  randomize = TRUE,
  value = 0,
  real_range = 100,
  categorical_fraction = 0.2,
  factors = 100,
  integer_fraction = 0.2,
  integer_range = 100,
  binary_fraction = 0.1,
  binary_ones_fraction = 0.02,
)

Examples

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

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

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

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

## End(Not run)
h2o.createFrame

```r
h2o.createFrame

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 pre-
ceded 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

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

hf <- h2o.createFrame(rows = 100, cols = 10, randomize = FALSE, value = 5,
                      categorical_fraction = 0, integer_fraction = 0)
summary(hf)
## End(Not run)
```

h2o.cross_validation_fold_assignment

*Retrieve the cross-validation fold assignment*

Description

Retrieve the cross-validation fold assignment

Usage

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

Arguments

- **object**: An H2OModel object.

Value

Returns a H2OFrame

Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars.gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                    nfolds = 5, keep_cross_validation_fold_assignment= TRUE, seed = 1234)
h2o.cross_validation_fold_assignment(cars.gbm)
## End(Not run)
```
h2o.cross_validation_holdout_predictions

Retrieve the cross-validation holdout predictions

Description
Retrieve the cross-validation holdout predictions

Usage
h2o.cross_validation_holdout_predictions(object)

Arguments
object
An H2OModel object.

Value
Returns a H2OFrame

Examples
## Not run:
library(h2o)
h2o.init()
cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
nfolds = 5, keep_cross_validation_predictions = TRUE, seed = 1234)
h2o.cross_validation_holdout_predictions(cars_gbm)
## End(Not run)

h2o.cross_validation_models

Retrieve the cross-validation models

Description
Retrieve the cross-validation models

Usage
h2o.cross_validation_models(object)
Arguments

object  An H2OModel object.

Value

Returns a list of H2OModel objects

Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
nfolds = 5, keep_cross_validation_models = TRUE, seed = 1234)
h2o.cross_validation_models(cars_gbm)

## End(Not run)
```

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

Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
nfolds = 5, keep_cross_validation_predictions = TRUE, seed = 1234)
h2o.cross_validation_predictions(cars_gbm)
## End(Not run)
```

h2o.cummax

Return the cumulative max over a column or across a row

Description

Return the cumulative max over a column or across a row

Usage

```r
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.

Examples

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

frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.cummax(frame, 1)
## End(Not run)
```
h2o.cummin

Return the cumulative min over a column or across a row

Description

Return the cumulative min over a column or across a row

Usage

h2o.cummin(x, axis = 0)

Arguments

x
An H2OFrame object.

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

See Also

cummin for the base R implementation.

Examples

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

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

## End(Not run)

h2o.cumprod

Return the cumulative product over a column or across a row

Description

Return the cumulative product over a column or across a row

Usage

h2o.cumprod(x, axis = 0)

Arguments

x
An H2OFrame object.

axis
An int that indicates whether to do down a column (0) or across a row (1).
See Also

cumprod for the base R implementation.

Examples

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

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

## End(Not run)
**h2o.cut**  

*Cut H2O Numeric Data to Factor*

**Description**

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

**Usage**

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

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

**Arguments**

- `x`: An H2OFrame object with a single numeric column.
- `breaks`: A numeric vector of two or more unique cut points.
- `labels`: Labels for the levels of the resulting category. By default, labels are constructed using "(a,b]" interval notation.
- `include.lowest`: Logical, indicating if an `x[i]` equal to the lowest (or highest, for `right = FALSE`) `breaks` value should be included.
- `right`: Logical, indicating if the intervals should be closed on the right (opened on the left) or vice versa.
- `dig.lab`: Integer which is used when labels are not given, determines the number of digits used in formatting the break numbers.
- `...`: Further arguments passed to or from other methods.

**Value**

Returns an H2OFrame object containing the factored data with intervals as levels.
Examples

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

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

## End(Not run)
```

---

**h2o.day** Convert Milliseconds to Day of Month in H2O Datasets

### Description

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

### Usage

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

h2o.init()

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

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

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

## End(Not run)
```

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.
## Examples

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

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

## End(Not run)
```

---

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

### Description

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

### Usage

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

---

**h2o.deepfeatures**  
*Feature Generation via H2O Deep Learning*

**Description**

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

**Usage**

```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 (integer) of the hidden layer to extract

**Value**

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

**See Also**

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

**Examples**

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

### End(Not run)

---

**h2o.deeplearning**

**Build a Deep Neural Network model using CPUs**

**Description**

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

**Usage**

```r
h2o.deeplearning(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  weights_column = NULL,
  offset_column = NULL,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  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,
  rho = 0.99,
)```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsilon</td>
<td>1e-08</td>
</tr>
<tr>
<td>rate</td>
<td>0.005</td>
</tr>
<tr>
<td>rate_annealing</td>
<td>1e-06</td>
</tr>
<tr>
<td>rate_decay</td>
<td>1</td>
</tr>
<tr>
<td>momentum_start</td>
<td>0</td>
</tr>
<tr>
<td>momentum_ramp</td>
<td>1e+06</td>
</tr>
<tr>
<td>momentum_stable</td>
<td>0</td>
</tr>
<tr>
<td>nesterov_accelerated_gradient</td>
<td>TRUE</td>
</tr>
<tr>
<td>input_dropout_ratio</td>
<td>0</td>
</tr>
<tr>
<td>hidden_dropout_ratios</td>
<td>NULL</td>
</tr>
<tr>
<td>l1</td>
<td>0</td>
</tr>
<tr>
<td>l2</td>
<td>0</td>
</tr>
<tr>
<td>max_w2</td>
<td>3.4028235e+38</td>
</tr>
<tr>
<td>initial_weight_distribution</td>
<td>c(&quot;UniformAdaptive&quot;, &quot;Uniform&quot;, &quot;Normal&quot;)</td>
</tr>
<tr>
<td>initial_weight_scale</td>
<td>1</td>
</tr>
<tr>
<td>initial_weights</td>
<td>NULL</td>
</tr>
<tr>
<td>initial_biases</td>
<td>NULL</td>
</tr>
<tr>
<td>loss</td>
<td>c(&quot;Automatic&quot;, &quot;CrossEntropy&quot;, &quot;Quadratic&quot;, &quot;Huber&quot;, &quot;Absolute&quot;, &quot;Quantile&quot;)</td>
</tr>
<tr>
<td>distribution</td>
<td>c(&quot;AUTO&quot;, &quot;bernoulli&quot;, &quot;multinomial&quot;, &quot;gaussian&quot;, &quot;poisson&quot;, &quot;gamma&quot;, &quot;tweedie&quot;, &quot;laplace&quot;, &quot;quantile&quot;, &quot;huber&quot;)</td>
</tr>
<tr>
<td>quantile_alpha</td>
<td>0.5</td>
</tr>
<tr>
<td>tweedie_power</td>
<td>1.5</td>
</tr>
<tr>
<td>huber_alpha</td>
<td>0.9</td>
</tr>
<tr>
<td>score_interval</td>
<td>5</td>
</tr>
<tr>
<td>score_training_samples</td>
<td>10000</td>
</tr>
<tr>
<td>score_validation_samples</td>
<td>0</td>
</tr>
<tr>
<td>score_duty_cycle</td>
<td>0.1</td>
</tr>
<tr>
<td>classification_stop</td>
<td>0</td>
</tr>
<tr>
<td>regression_stop</td>
<td>1e-06</td>
</tr>
<tr>
<td>stopping_rounds</td>
<td>5</td>
</tr>
<tr>
<td>stopping_metric</td>
<td>c(&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;AUCPR&quot;, &quot;lift_top_group&quot;, &quot;misclassification&quot;, &quot;mean_per_class_error&quot;, &quot;custom&quot;, &quot;custom_increasing&quot;)</td>
</tr>
<tr>
<td>stopping_tolerance</td>
<td>0</td>
</tr>
<tr>
<td>max_runtime_secs</td>
<td>0</td>
</tr>
<tr>
<td>score_validation_sampling</td>
<td>c(&quot;Uniform&quot;, &quot;Stratified&quot;)</td>
</tr>
<tr>
<td>diagnostics</td>
<td>TRUE</td>
</tr>
<tr>
<td>fast_mode</td>
<td>TRUE</td>
</tr>
<tr>
<td>force_load_balance</td>
<td>TRUE</td>
</tr>
<tr>
<td>variable_importances</td>
<td>TRUE</td>
</tr>
<tr>
<td>replicate_training_data</td>
<td>TRUE</td>
</tr>
<tr>
<td>single_node_mode</td>
<td>FALSE</td>
</tr>
<tr>
<td>shuffle_training_data</td>
<td>FALSE</td>
</tr>
<tr>
<td>missing_values_handling</td>
<td>c(&quot;MeanImputation&quot;, &quot;Skip&quot;)</td>
</tr>
<tr>
<td>quiet_mode</td>
<td>FALSE</td>
</tr>
<tr>
<td>autoencoder</td>
<td>FALSE</td>
</tr>
<tr>
<td>sparse</td>
<td>FALSE</td>
</tr>
<tr>
<td>col_major</td>
<td>FALSE</td>
</tr>
<tr>
<td>average_activation</td>
<td>0</td>
</tr>
<tr>
<td>sparsity_beta</td>
<td>0</td>
</tr>
<tr>
<td>max_categorical_features</td>
<td>2147483647</td>
</tr>
<tr>
<td>reproducible</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
export_weights_and_biases = FALSE,
mini_batch_size = 1,
categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
"Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
elastic_averaging = FALSE,
elastic_averaging_moving_rate = 0.9,
elastic_averaging_regularization = 0.001,
export_checkpoints.dir = NULL,
verbose = FALSE
)

Arguments

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

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

training_frame
Id of the training data frame.

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

validation_frame
Id of the validation data frame.

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

keep_cross_validation_models
Logical. Whether to keep the cross-validation models. Defaults to TRUE.

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

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

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

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

ignore_const_cols
Logical. Ignore constant columns. Defaults to TRUE.

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

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

activation

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

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

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

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

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

adaptive_rate
 Logical. Adaptive learning rate. Defaults to TRUE.

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

epsilon
Adaptive learning rate smoothing factor (to avoid divisions by zero and allow progress). Defaults to 1e-08.
rate  Learning rate (higher => less stable, lower => slower convergence). Defaults to 0.005.
rate_annealing  Learning rate annealing: rate / (1 + rate_annealing * samples). Defaults to 1e-06.
rate_decay  Learning rate decay factor between layers (N-th layer: rate * rate_decay ^ (n - 1)). Defaults to 1.
momentum_start  Initial momentum at the beginning of training (try 0.5). Defaults to 0.
momentum_ramp  Number of training samples for which momentum increases. Defaults to 1000000.
momentum_stable  Final momentum after the ramp is over (try 0.99). Defaults to 0.
nesterov_accelerated_gradient  Logical. Use Nesterov accelerated gradient (recommended). Defaults to TRUE.
input_dropout_ratio  Input layer dropout ratio (can improve generalization, try 0.1 or 0.2). Defaults to 0.
hidden_dropout_ratios  Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5.
L1  L1 regularization (can add stability and improve generalization, causes many weights to become 0). Defaults to 0.
L2  L2 regularization (can add stability and improve generalization, causes many weights to be small. Defaults to 0.
max_w2  Constraint for squared sum of incoming weights per unit (e.g. for Rectifier). Defaults to 3.4028235e+38.
initial_weight_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.
shortest_time_interval  Shortest time interval (in seconds) between model scoring. Defaults to 5.
score_training_samples  Number of training set samples for scoring (0 for all). Defaults to 10000.
score_validation_samples
Number of validation set samples for scoring (0 for all). Defaults to 0.

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

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

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

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

stopping_metric
Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anonomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

stopping_tolerance
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.

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

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

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

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

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

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

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

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

shuffle_training_data
Logical. Enable shuffling of training data (recommended if training data is replicated and train_samples_per_iteration is close to #nodes x #rows, of if using balance_classes). Defaults to FALSE.
missing_values_handling
Handling of missing values. Either MeanImputation or Skip. Must be one of: "MeanImputation", "Skip". Defaults to MeanImputation.

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

autoencoder Logical. Auto-Encoder. Defaults to FALSE.

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

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

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

sparsity_beta Sparsity regularization. #Experimental Defaults to 0.

max_categorical_features Max. number of categorical features, enforced via hashing. #Experimental Defaults to 2147483647.

reproducible Logical. Force reproducibility on small data (will be slow - only uses 1 thread). Defaults to FALSE.

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

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

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

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

elastic_averaging Moving Rate Elastic averaging moving rate (only if elastic averaging is enabled). Defaults to 0.9.

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

export_checkpoints_dir Automatically export generated models to this directory.

verbose Logical. Print scoring history to the console (Metrics per epoch). Defaults to FALSE.

See Also
predict.H2OModel for prediction
Examples

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

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

## End(Not run)
```

description

**Description**

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

**Usage**

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

**Arguments**

- `frame`: An H2OFrame object.

**Value**

A table with the Frame stats.

**Examples**

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

## End(Not run)
```
h2o.difflag1  

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

**Description**

Conduct a lag 1 transform on a numeric H2OFrame column

**Usage**

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

**Arguments**

- `object`  
  H2OFrame object

**Value**

Returns an H2OFrame object.

**Examples**

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

cars <- h2o.importFile(f)
predictors <- c("displacement","power","weight","acceleration","year")
response <- "cylinders"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response, training_frame = train,
                     validation_frame = valid, nfolds = 5, seed = 1234)
h2o.difflag1(cars["cylinders"])
## End(Not run)
```

---

h2o.dim  

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

**Description**

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

**Usage**

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

**Arguments**

- `x`  
  An H2OFrame object.
See Also

dim for the base R implementation.

Examples

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

cars <- h2o.importFile(f)
h2o.dim(cars)

## End(Not run)
h2o.distance

*Compute a pairwise distance measure between all rows of two numeric H2O Frames.*

**Description**

Compute a pairwise distance measure between all rows of two numeric H2O Frames.

**Usage**

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

**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
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.distance(prostate[11:30,], prostate[1:10,], "cosine")
## End(Not run)
```

h2o.downloadAllLogs

*Download H2O Log Files to Disk*

**Description**

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

**Usage**

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

**Arguments**

- `dirname`: (Optional) A character string indicating the directory that the log file should be saved in.
- `filename`: (Optional) A character string indicating the name that the log file should be saved to. Note that the saved format is .zip, so the file name must include the .zip extension.
h2o.downloadCSV

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

## End(Not run)

### h2o.downloadCSV

**Download H2O Data to Disk**

#### Description

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

#### Usage

```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
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
file_path <- paste(getwd(), "my_iris_file.csv", sep = .Platform$file.sep)
h2o.downloadCSV(iris_hf, file_path)
file.info(file_path)
file.remove(file_path)
```

## End(Not run)
h2o.download_model

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

Description

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

Usage

h2o.download_model(model, path = NULL)

Arguments

model
An H2OModel

path
The path where binary file should be downloaded. Downloaded to current directory by default.

Examples

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

## End(Not run)

h2o.download_mojo

Download the model in MOJO format.

Description

Download the model in MOJO format.

Usage

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

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

## End(Not run)
get_jar  Whether to also download the h2o-genmodel.jar file needed to compile the POJO
jar_name  Custom name of genmodel jar.

Value

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

Examples

## Not run:
library(h2o)
h <- h2o.init()
fr <- as.h2o(iris)
my_model <- h2o.gbm(x = 1:4, y = 5, training_frame = fr)

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

## End(Not run)

h2o.entropy  Shannon entropy

Description

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

Usage

h2o.entropy(x)

Arguments

x  The column on which to calculate the entropy.

Examples

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

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

buys_entropy <- h2o.entropy(buys)

## End(Not run)
h2o.exp

Compute the exponential function of x

Description
Compute the exponential function of x

Usage
h2o.exp(x)

Arguments
x  An H2OFrame object.

See Also
exp for the base R implementation.

Examples
## Not run:
library(h2o)
h2o.init()
frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.exp(frame["C1"])
## End(Not run)

h2o.exportFile

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

Description
Exports an H2OFrame (which can be either VA or FV) to a file. This file may be on the H2O 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,
sep = ",",
compression = NULL,
parts = 1
)
**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

Examples

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

train <- h2o.importFile(f)
h2o.exportHDFS(train, path = " ", force = FALSE)

## End(Not run)

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

#### Usage

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

#### Arguments

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

#### Value
An H2OFrame after filling missing values

#### Examples

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

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

## End(Not run)
**h2o.filterNACols**  
*Filter NA Columns*

**Description**  
Filter NA Columns

**Usage**  
```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

**Examples**
```r  
## Not run:  
library(h2o)  
h2o.init()  
frame <- h2o.createFrame(rows = 6, cols = 2,  
  categorical_fraction = 0.0,  
  missing_fraction = 0.7,  
  seed = 123)  
h2o.filterNACols(frame, frac = 0.5)  
h2o.filterNACols(frame, frac = 0.6)  
## End(Not run)  
```

---

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

**Description**  
Find synonyms using a word2vec model.

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

**Arguments**
- **word2vec**  
  A word2vec model.
- **word**  
  A single word to find synonyms for.
- **count**  
  The top ‘count’ synonyms will be returned.
Examples

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

f <- "https://raw.githubusercontent.com/h2oai/sparkling-water/rel-1.6/examples/smalldata/
jobtitles <- h2o.importFile(paste0(f, "craigslistJobTitles.csv"),
    col.names = c("category", "jobtitle"),
    col.types = c("String", "String"), header = TRUE)
words <- h2o.tokenize(jobtitles, " ")
vec <- h2o.word2vec(training_frame = words)
h2o.findSynonyms(vec, "teacher", count = 20)
```

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

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

Arguments

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

Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
    training_frame = train, validation_frame = valid,
    build_tree_one_node = TRUE, seed = 1234)
perf <- h2o.performance(cars_gbm, cars)
h2o.find_row_by_threshold(perf, 0.5)
```

## End(Not run)
**h2o.find_threshold_by_max_metric**

*Find the threshold, give the max metric*

**Description**

Find the threshold, give the max metric

**Usage**

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

**Arguments**

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

**Examples**

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars,ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
                   training_frame = train, validation_frame = valid,
                   build_tree_one_node = TRUE , seed = 1234)
perf <- h2o.performance(cars_gbm, cars)
h2o.find_threshold_by_max_metric(perf, "fpr")

## End(Not run)
```

**h2o.floor**

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

**Description**

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

**Usage**

```r
h2o.floor(x)
```
Arguments

x An H2OFrame object.

See Also

floor for the base R implementation.

Examples

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

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

## End(Not run)

---

h2o.flow **Open H2O Flow**

Description

Open H2O Flow in your browser

Usage

h2o.flow()

---

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

Description

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

Usage

h2o.gainsLift(object, ...)

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

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

**Fit a General Additive Model**

**Description**

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

h2o.gam(
  x,
  y,
  training_frame,
  gam_columns,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  seed = -1,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  offset_column = NULL,
  weights_column = NULL,
  family = c("gaussian", "binomial", "quasibinomial", "ordinal", "multinomial",
             "poisson", "gamma", "tweedie", "negativebinomial", "fractionalbinomial"),
  tweedie_variance_power = 0,
  tweedie_link_power = 0,
  theta = 0,
  solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
             "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
  alpha = NULL,
  lambda = NULL,
  lambda_search = FALSE,
  early_stopping = TRUE,
  nlambdas = -1,
  standardize = FALSE,
  missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
  plug_values = NULL,
  compute_p_values = FALSE,
  remove_collinear_columns = FALSE,
  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"),
  prior = -1,
  lambda_min_ratio = -1,
  beta_constraints = NULL,
  max_active_predictors = -1,
  interactions = NULL,
  interaction_pairs = NULL,
  obj_reg = -1,
  export_checkpoints_dir = NULL,
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,
num_knots = NULL,
knot_ids = NULL,
bs = NULL,
scale = NULL,
keep_gam_cols = 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.

gam_columns Predictor column names for gam

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

validation_frame Id of the validation data frame.

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

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

keep_cross_validation_models Logical. Whether to keep the cross-validation models. Defaults to TRUE.

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

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

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.

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", "negativebinomial", "fractionalbinomial".

tweedie_variance_power  Tweedie variance power. Defaults to 0.

tweedie_link_power  Tweedie link power. Defaults to 0.

theta  Theta. Defaults to 0.

solver  AUTO will set the solver based on given data and the other parameters. IRLSM is fast on problems with small number of predictors and for lambda-search with L1 penalty. L_BFGS scales better for datasets with many columns. Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR". Defaults to AUTO.

alpha  Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.

lambda  Regularization strength.

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

ever_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 nlambdas is set to 30 (fewer lambdas are needed for ridge regression) otherwise it is set to 100. Defaults to -1.

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

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

plug_values  Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values Handling = 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
Link function. Must be one of: "family_default", "identity", "logit", "log", "inverse", "tweedie", "ologit".

prior
Prior probability for y=1. To be used only for logistic regression if 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 100000000. Defaults to -1.

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

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

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

export_checkpoints_dir
Automatically export generated models to this directory.

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’

num_knots
Number of knots for gam predictors

knot_ids
String arrays storing frame keys of knots. One for each gam column specified in gam_columns

bs
Basis function type for each gam predictors, 0 for cr

scale
Smoothing parameter for gam predictors

keep_gam_cols
Logical. Save keys of model matrix Defaults to FALSE.

Examples

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

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

## End(Not run)
```

h2o.gbm

Build gradient boosted classification or regression trees

Description
Builds gradient boosted classification trees and gradient boosted regression trees on a parsed data set. The default distribution function will guess the model type based on the response column type. In order to run properly, the response column must be an numeric for "gaussian" or an enum for "bernoulli" or "multinomial".
h2o.gbm(x, y, 
  training_frame, 
  model_id = NULL, 
  validation_frame = NULL, 
  nfolds = 0, 
  keep_cross_validation_models = TRUE, 
  keep_cross_validation_predictions = FALSE, 
  keep_cross_validation_fold_assignment = FALSE, 
  score_each_iteration = FALSE, 
  score_tree_interval = 0, 
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"), 
  fold_column = NULL, 
  ignore_const_cols = TRUE, 
  offset_column = NULL, 
  weights_column = NULL, 
  balance_classes = FALSE, 
  class_sampling_factors = NULL, 
  max_after_balance_size = 5, 
  max_hit_ratio_k = 0, 
  ntree = 50, 
  max_depth = 5, 
  min_rows = 10, 
  nbins = 20, 
  nbins_top_level = 1024, 
  nbins_cats = 1024, 
  r2_stopping = Inf, 
  stopping_rounds = 0, 
  stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", 
                    "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", 
                    "custom", "custom_increasing"), 
  stopping_tolerance = 0.001, 
  max_runtime_secs = 0, 
  seed = -1, 
  build_tree_one_node = FALSE, 
  learn_rate = 0.1, 
  learn_rate_annealing = 1, 
  distribution = c("AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian", 
                  "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom"), 
  quantile_alpha = 0.5, 
  tweedie_power = 1.5, 
  huber_alpha = 0.9, 
  checkpoint = NULL, 
  sample_rate = 1, 
  sample_rate_per_class = NULL, 
  col_sample_rate = 1, 
  col_sample_rate_change_per_level = 1, 
  col_sample_rate_per_tree = 1, 
  min_split_improvement = 1e-05, 
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", 
                    "UniformGlobal", "QuantilesGlobal", "UserDefined"), 
  distribution = c("AUTO", "bernoulli", "quasibinomial", "multinomial", "gaussian", 
                  "poisson", "gamma", "tweedie", "laplace", "quantile", "huber", "custom"), 
  quantile_alpha = 0.5, 
  tweedie_power = 1.5, 
  huber_alpha = 0.9, 
  checkpoint = NULL, 
  sample_rate = 1, 
  sample_rate_per_class = NULL, 
  col_sample_rate = 1, 
  col_sample_rate_change_per_level = 1, 
  col_sample_rate_per_tree = 1, 
  min_split_improvement = 1e-05, 
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", 
                    "UniformGlobal", "QuantilesGlobal", "UserDefined")
"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,
custom_distribution_func = NULL,
export_checkpoints_dir = NULL,
monotone_constraints = NULL,
check_constant_response = TRUE,
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 classi-
fication model.
training_frame Id of the training data frame.
model_id Destination id for this model; auto-generated if not specified.
validation_frame Id of the validation data frame.
nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
keep_cross_validation_models Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. De-
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
'Clasified' option will stratify the folds based on the response variable, for clas-
sification problems. Must be one of: "AUTO", "Random", "Modulo", "Strati-
fied". 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.

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 5.

min_rows Fewest allowed (weighted) observations in a leaf. Defaults to 10.

nbins For numerical columns (real/int), build a histogram of (at least) this many bins, then split at the best point Defaults to 20.

nbins_top_level For numerical columns (real/int), build a histogram of (at most) this many bins at the root level, then decrease by factor of two per level Defaults to 1024.

nbins_cats For categorical columns (factors), build a histogram of this many bins, then split at the best point. Higher values can lead to more overfitting. Defaults to 1024.

r2_stopping r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this Defaults to 1.797693135e+308.

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

stopping_metric Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

stopping_tolerance Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.
h2o.gbm

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

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

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

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

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

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

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

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

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

checkpoint
Model checkpoint to resume training with.

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

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

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

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

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

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

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

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

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

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

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

verbose

Logical. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

See Also

predict.H2OModel for prediction

Examples

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

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

## End(Not run)
**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

```r
## Not run:
# library(h2o)
# h2o.init()
# generic_model <- h2o.genericModel("/path/to/model.zip")
# predictions <- h2o.predict(generic_model, dataset)
## End(Not run)
```

---

**h2o.genericModel**

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

**Description**

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

**Usage**

```r
h2o.genericModel(mojo_file_path)
```

**Arguments**

- `mojo_file_path` Filesystem path to the model imported

**Value**

Returns H2O Generic Model based on given embedded model.
Examples

```r
## Not run:

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

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

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

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

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

## End(Not run)
```

h2o.getConnection

### Retrieve an H2O Connection

**Description**

Attempt to recover an h2o connection.

**Usage**

```
import 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**

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

**Arguments**

- `id` A string indicating the unique frame of the dataset to retrieve.
h2o.getGLMFullRegularizationPath

**Examples**

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

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

## End(Not run)
```

---

**h2o.getGLMFullRegularizationPath**

*Extract full regularization path from a GLM model*

**Description**

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

**Usage**

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

**Arguments**

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

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

**Description**

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

**Usage**

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

**Arguments**

- `grid_id`: ID of existing grid object to fetch
- `sort_by`: Sort the models in the grid space by a metric. Choices are "logloss", "residual_deviance", "mse", "auc", "accuracy", "precision", "recall", "f1", etc.
- `decreasing`: Specify whether sort order should be decreasing
- `verbose`: Controls verbosity of the output, if enabled prints out error messages for failed models (default: FALSE)

**Examples**

```r
## Not run:
library(h2o)
library(jsonlite)
h2o.init()
iris_hf <- as.h2o(iris)
h2o.grid("gbm", grid_id = "gbm_grid_id", x = c(1:4), y = 5,
          training_frame = iris_hf, hyper_params = list(ntrees = c(1, 2, 3)))
grid <- h2o.getGrid("gbm_grid_id")
# Get grid summary
summary(grid)
# Fetch grid models
model_ids <- grid@model_ids
models <- lapply(model_ids, function(id) { h2o.getModel(id)})
## End(Not run)
```

---

**h2o.getId**

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

**Description**

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

**Usage**

```r
h2o.getId(x)
```
**h2o.getModel**

Get an R reference to an H2O model

**Description**

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

**Usage**

```r
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**

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

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

## End(Not run)
```
h2o.getModelTree

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

Description

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

Usage

h2o.getModelTree(model, tree_number, tree_class = NA)

Arguments

model Model with trees
tree_number Number of the tree in the model to fetch, starting with 1
tree_class Name of the class of the tree (if applicable). This value is ignored for regression and binomial response column, as there is only one tree built. As there is exactly one class per categorical level, name of tree’s class equals to the corresponding categorical level of response column.

Value

Returns an H2OTree object with detailed information about a tree.

Examples

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

iris <- h2o.importFile(f)

gbm_model <- h2o.gbm(y = "species", training_frame = iris)
tree <- h2o.getModelTree(gbm_model, 1, "Iris-setosa")

## End(Not run)

h2o.getTimezone

Get the Time Zone on the H2O cluster Returns a string

Description

Get the Time Zone on the H2O cluster Returns a string

Usage

h2o.getTimezone()
**h2o.getTypes**

*Get the types-per-column*

---

**Description**

Get the types-per-column

**Usage**

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

**Arguments**

- `x` : An H2OFrame

**Value**

A list of types per column

**Examples**

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

iris <- h2o.importFile(f)
h2o.getTypes(iris)

## End(Not run)
```

---

**h2o.getVersion**

*Get h2o version*

---

**Description**

Get h2o version

**Usage**

```r
h2o.getVersion()
```
h2o.get_automl  Get an R object that is a subclass of H2OAutoML

**Description**

Get an R object that is a subclass of H2OAutoML.

**Usage**

```r
h2o.get_automl(project_name)
```

**Arguments**

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

**Value**

Returns an object that is a subclass of H2OAutoML.

**Examples**

```r
## Not run:
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", project_name="aml_housevotes",
                   training_frame = votes_hf, max_runtime_secs = 30)
automl_retrieved <- h2o.get_automl("aml_housevotes")
## End(Not run)
```

h2o.get_leaderboard  Retrieve the leaderboard from the AutoML instance.

**Description**

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

**Usage**

```r
h2o.get_leaderboard(object, extra_columns = NULL)
```
**Arguments**

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

**Value**

An H2OFrame representing the leaderboard.

**Examples**

```r
## Not run:
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", project_name="aml_housevotes", training_frame = votes_hf, max_runtime_secs = 30)
lb_all <- h2o.get_leaderboard(aml, "ALL")
lb_custom <- h2o.get_leaderboard(aml, c('predict_time_per_row_ms', 'training_time_ms'))
## End(Not run)
```

---

**Description**

Retrieve actual number of trees for tree algorithms

**Usage**

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

**Arguments**

- **object**: An H2OModel object.
h2o.get_segment_models

Retrieves an instance of H2OSegmentModels for a given id.

Description

Retrieves an instance of H2OSegmentModels for a given id.

Usage

h2o.get_segment_models(segment_models_id)

Arguments

segment_models_id

A string indicating the unique segment_models_id

Value

Returns an object that is a subclass of H2OSegmentModels.

Examples

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

## End(Not run)

h2o.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 GINIvalue is returned. If more than one parameter is set to TRUE, then a named vector of GINIs are returned, where the names are "train", "valid" or "xval".

Usage

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

Fit a generalized linear model

Description

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

Usage

h2o.glm(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  nfolds = 0,
  seed = -1,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  random_columns = NULL,
)
ignore_const_cols = TRUE,
score_each_iteration = FALSE,
offset_column = NULL,
weights_column = NULL,
family = c("gaussian", "binomial", "fractionalbinomial", "quasibinomial", "ordinal",
  "multinomial", "poisson", "gamma", "tweedie", "negativebinomial"),
rand_family = c("[gaussian]"),
tweedie.variance_power = 0,
tweedie_link_power = 1,
theta = 1e-10,
solver = c("AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE",
  "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"),
alpha = NULL,
lambda = NULL,
lambda_search = FALSE,
every_stopping = TRUE,
nlambdas = -1,
standardize = TRUE,
missing_values_handling = c("MeanImputation", "Skip", "PlugValues"),
plug_values = NULL,
compute_p_values = FALSE,
remove_collinear_columns = FALSE,
intercept = TRUE,
non_negative = FALSE,
max_iterations = -1,
objective_epsilon = -1,
beta_epsilon = 1e-04,
gradient_epsilon = -1,
link = c("family_default", "identity", "logit", "log", "inverse", "tweedie",
  "ologit"),
rand_link = c("[identity]", "[family_default]"),
startval = NULL,
calc_like = FALSE,
HGLM = FALSE,
prior = -1,
lambda_min_ratio = -1,
beta_constraints = NULL,
max_active_predictors = -1,
interactions = NULL,
interaction_pairs = NULL,
obj_reg = -1,
export_checkpoints_dir = NULL,
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.
nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
seed Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).
keep_cross_validation_models Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
fold_assignment Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
fold_column Column with cross-validation fold index assignment per observation.
random_columns random columns indices for HGLM.
ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.
score_each_iteration Logical. Whether to score during each iteration of model training. Defaults to FALSE.
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.
h2o.glm

rand_family
Random Component Family array. One for each random component. Only support gaussian for now. Must be one of: "[gaussian]".

tweedie_variance_power
Tweedie variance power Defaults to 0.

tweedie_link_power
Tweedie link power Defaults to 1.

theta
Theta Defaults to 1e-10.

solver
AUTO will set the solver based on given data and the other parameters. IRLSM is fast on problems with small number of predictors and for lambda-search with L1 penalty. L_BFGS scales better for datasets with many columns. Must be one of: "AUTO", "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR". Defaults to AUTO.

alpha
Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.

lambda
Regularization strength

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

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

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

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

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

plug_values
Plug Values (a single row frame containing values that will be used to impute missing values of the training/validation frame, use with conjunction missing_values_handling = PlugValues)

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

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 0.00001, otherwise the default value is 0.001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively. Defaults to -1.

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

rand_link  Link function array for random component in HGLM. Must be one of: "[identity]", "[family_default]".

startval  double array to initialize fixed and random coefficients for HGLM.

calc_like  Logical. if true, will return likelihood function value for HGLM. Defaults to FALSE.

HGLM  Logical. If set to true, will return HGLM model. Otherwise, normal GLM model will be returned. Defaults to FALSE.

prior  Prior probability for y==1. To be used only for logistic regression if 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 50000; otherwise it is set to 100000000. Defaults to -1.

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

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

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

export_checkpoints_dir  Automatically export generated models to this directory.

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

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

Examples

## Not run:
h2o.init()

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

# Run GLM of VOL ~ CAPSULE + AGE + RACE + PSA + GLEASON
predictors = setdiff(colnames(prostate), c("ID", "DPROS", "DCAPS", "VOL"))
h2o.glm(y = "VOL", x = predictors, training_frame = prostate, family = "gaussian",
nfolds = 0, alpha = 0.5, lambda_search = FALSE)

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

```r
lambda_search=TRUE)
h2o.std_coef_plot(glm, num_of_features = 20)
```

## End(Not run)

---

**h2o.glrm**  
**Generalized low rank decomposition of an H2O data frame**

**Description**

Builds a generalized low rank decomposition of an H2O data frame

**Usage**

```r
h2o.glrm(
  training_frame,
  cols = NULL,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  loading_name = NULL,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  k = 1,
  loss = c("Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic"),
  loss_by_col_idx = NULL,
  multi_loss = c("Categorical", "Ordinal"),
  period = 1,
  regularization_x = c("None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex"),
  regularization_y = c("None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex"),
  gamma_x = 0,
  gamma_y = 0,
  max_iterations = 1000,
  max_updates = 2000,
  init_step_size = 1,
  min_step_size = 1e-04,
  seed = -1,
  init = c("Random", "SVD", "PlusPlus", "User"),
  svd_method = c("GramSVD", "Power", "Randomized"),
  user_y = NULL,
  user_x = NULL,
  expand_user_y = TRUE,
  impute_original = FALSE,
  recover_svd = FALSE,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```
Arguments

training_frame  
Id of the training data frame.

cols  
(Optional) A vector containing the data columns on which k-means operates.

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

validation_frame  
Id of the validation data frame.

ignore_const_cols  
Logical. Ignore constant columns. Defaults to TRUE.

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

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  

loss_by_col  
Loss function by column (override) Must be one of: "Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic", "Categorical", "Ordinal".

loss_by_col_idx  
Loss function by column index (override)

multi_loss  
Categorical loss function Must be one of: "Categorical", "Ordinal". Defaults to Categorical.

period  
Length of period (only used with periodic loss function) Defaults to 1.

regularization_x  
Regularization function for X matrix Must be one of: "None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex". Defaults to None.

regularization_y  
Regularization function for Y matrix Must be one of: "None", "Quadratic", "L2", "L1", "NonNegative", "OneSparse", "UnitOneSparse", "Simplex". Defaults to None.

gamma_x  
Regularization weight on X matrix Defaults to 0.

gamma_y  
Regularization weight on Y matrix Defaults to 0.

max_iterations  
Maximum number of iterations Defaults to 1000.

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

init_step_size  
Initial step size Defaults to 1.

min_step_size  
Minimum step size Defaults to 0.0001.

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

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

svd_method  
Method for computing SVD during initialization (Caution: Randomized is currently experimental and unstable) Must be one of: "GramSVD", "Power", "Randomized". Defaults to Randomized.
user_y User-specified initial Y
user_x User-specified initial X
expand_user_y Logical. Expand categorical columns in user-specified initial Y Defaults to TRUE.
impute_original Logical. Reconstruct original training data by reversing transform Defaults to FALSE.
recover_svd Logical. Recover singular values and eigenvectors of XY Defaults to FALSE.
max_runtime_secs Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.

Value
an object of class H2ODimReductionModel.

References

See Also
h2o.kmeans, h2o.svd, h2o.prcomp

Examples
```r
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
h2o.glrm(training_frame = australia, k = 5, loss = "Quadratic", regularization_x = "L1", 
gamma_x = 0.5, gamma_y = 0, max_iterations = 1000)

## End(Not run)
```

h2o.grep

Search for matches to an argument pattern

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

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

Arguments

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

Details

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

Value

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

Examples

## Not run:
library(h2o)
h2o.init()
addresses <- as.h2o(c("2307", "Leghorn St", "Mountain View", "CA", "94043"))
zip.codes <- addresses[h2o.grep("[0-9]{5}\", addresses, output.logical = TRUE),]
## End(Not run)

---

h2o.grid  

H2O Grid Support

Description

Provides a set of functions to launch a grid search and get its results.
h2o.grid

Usage

h2o.grid(
    algorithm,  
    grid_id,   
    x,        
    y,        
    training_frame,   
    ...,        
    hyper_params = list(),  
    is_supervised = NULL,  
    do_hyper_params_check = FALSE,  
    search_criteria = NULL,  
    export_checkpoints_dir = NULL,  
    parallelism = 1
)

Arguments

algorithm        Name of algorithm to use in grid search (gbm, randomForest, kmeans, glm,  
                  deeplearning, naivebayes, pca).

grid_id          (Optional) ID for resulting grid search. If it is not specified then it is 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  
                  list can include values for: strategy, max_models, max_runtime_secs, stop-  
                  ping_metric, stopping_tolerance, stopping_rounds and seed. The default strat-  
                  egy 'Cartesian' covers the entire space of hyperparameter combinations. If you  
                  want to use cartesian grid search, you can leave the search_criteria argument  
                  unspecified. Specify the "RandomDiscrete" strategy to get random search of all  
                  the combinations of your hyperparameters with three ways of specifying when to  
                  stop the search: max number of models, max time, and metric-based early stop-  
                  ping (e.g., stop if MSE has not improved by 0.0001 over the 5 best models). Ex-  
                  amples below: list(strategy = "RandomDiscrete", max_runtime_secs = 600, max_models =
h2o.group_by

or list(strategy = "RandomDiscrete", max_models = 42, max_runtime_secs = 28800)
or list(strategy = "RandomDiscrete", stopping_metric = "AUTO", stopping_tolerance =
or list(strategy = "RandomDiscrete", stopping_metric = "misclassification", stopping_rounds =

export_checkpoints_dir
Directory to automatically export grid in binary form to.

parallelism Level of Parallelism during grid model building. 1 = sequential building (default). Use the value of 0 for adaptive parallelism - decided by H2O. Any number > 1 sets the exact number of models built in parallel.

Details
Launch grid search with given algorithm and parameters.

Examples

```r
## Not run:
library(h2o)
library(jsonlite)
h2o.init()
iris_hf <- as.h2o(iris)
grid <- h2o.grid("gbm", x = c(1:4), y = 5, training_frame = iris_hf,
  hyper_params = list(ntrees = c(1, 2, 3)))
# Get grid summary
summary(grid)
# Fetch grid models
model_ids <- grid@model_ids
models <- lapply(model_ids, function(id) { h2o.getModel(id)})
## End(Not run)
```

h2o.group_by

**Group and Apply by Column**

**Description**

Performs a group by and apply similar to `ddply`.

**Usage**

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

Value

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

Examples

```r
## Not run:
library(h2o)
h2o.init()
df <- h2o.importFile("http://s3.amazonaws.com/h2o-public-test-data/smalldata/prostate/prostate.csv")
h2o.group_by(data = df, by = "RACE", nrow("VOL"))
## End(Not run)
```

h2o.gsub

String Global Substitute

Description

Creates a copy of the target column in which each string has all 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

## Not run:
library(h2o)
h2o.init()
string_to_gsub <- as.h2o("r tutorial")
sub_string <- h2o.gsub("r ", "H2O ", string_to_gsub)
## End(Not run)

h2o.head

Return the Head or Tail of an H2O Dataset.

Description

Returns the first or last rows of an H2OFrame object.

Usage

h2o.head(x, n = 6L, m = 200L, ...)

## S3 method for class 'H2OFrame'
head(x, n = 6L, m = 200L, ...)

h2o.tail(x, n = 6L, m = 200L, ...)

## S3 method for class 'H2OFrame'
tail(x, n = 6L, m = 200L, ...)

Arguments

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

Value

An H2OFrame containing the first or last n rows and m columns of an H2OFrame object.
h2o.HGLMMetrics

Examples

```r
## Not run:
library(h2o)
h2o.init(ip <- "localhost", port = 54321, startH2O = TRUE)
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
# Return the first 10 rows and 6 columns
h2o.head(australia, n = 10L, m = 6L)
# Return the last 10 rows and 6 columns
h2o.tail(australia, n = 10L, m = 6L)

# For Jupyter notebook with an R kernel,
# view all rows of a data frame
options(repr.matrix.max.rows=600, repr.matrix.max.cols=200)

## End(Not run)
```

h2o.HGLMMetrics

Retrieve HGLM ModelMetrics

Description

Retrieve HGLM ModelMetrics

Usage

h2o.HGLMMetrics(object)

Arguments

- `object`: an H2OModel object or H2OModelMetrics.

h2o.hist

Compute A Histogram

Description

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

Usage

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

Arguments

- `x`: A single numeric column from an H2OFrame.
- `breaks`: Can be one of the following: A string: "Sturges", "Rice", "sqrt", "Doane", "FD", "Scott" A single number for the number of breaks splitting the range of the vec into number of breaks bins of equal width A vector of numbers giving the split points, e.g., c(-50,213.2123,9324834)
- `plot`: A logical value indicating whether or not a plot should be generated (default is TRUE).
## h2o.hit_ratio_table

### Retrieve the Hit Ratios

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

### 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

### Examples

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

iris <- h2o.importFile(f)
h2o.asnumeric(iris["petal_len"])
h2o.hist(iris["petal_len"], breaks = "Sturges", plot = TRUE)

## End(Not run)
```

```r
# h2o.hit_ratio_table

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

iris <- h2o.importFile(f)
iris.split <- h2o.splitFrame(data = iris, ratios = 0.8, seed = 1234)
train <- iris.split[[1]]
valid <- iris.split[[2]]

iris_xgb <- h2o.xgboost(x = 1:4, y = 5, training_frame = train, validation_frame = valid)
hrt_iris <- h2o.hit_ratio_table(iris_xgb, valid = TRUE)
hrt_iris

## End(Not run)
```
### h2o.hour

**Convert Milliseconds to Hour of Day in H2O Datasets**

**Description**

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

**Usage**

```r
h2o.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**

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

**Arguments**

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

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

```r
## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.importFile(path = australia_path)
australia[,9] <- ifelse(australia[,3] < 279.9, 1, 0)
summary(australia)
## End(Not run)
```
h2o.importFile

```r
  col.types = NULL,
  na.strings = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = 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,
  skipped_columns = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.</td>
</tr>
<tr>
<td>destination_frame</td>
<td>(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.</td>
</tr>
<tr>
<td>parse</td>
<td>(Optional) A logical value indicating whether the file should be parsed after import, for details see h2o.parseRaw.</td>
</tr>
<tr>
<td>header</td>
<td>(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.</td>
</tr>
<tr>
<td>sep</td>
<td>(Optional) The field separator character. Values on each line of the file are separated by this character. If <code>sep = &quot;&quot;</code>, the parser will automatically detect the separator.</td>
</tr>
<tr>
<td>col.names</td>
<td>(Optional) An H2OFrame object containing a single delimited line with the column names for the file.</td>
</tr>
<tr>
<td>col.types</td>
<td>(Optional) A vector to specify whether columns should be forced to a certain type upon import parsing.</td>
</tr>
</tbody>
</table>
na.strings  (Optional) H2O will interpret these strings as missing.
decrypt_tool  (Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.
skipped_columns
a list of column indices to be skipped during parsing.
custom_non_data_line_markers
((Optional) If a line in imported file starts with any character in given string it
will NOT be imported. Empty string means all lines are imported, NULL means
that default behaviour for given format will be used
pattern  (Optional) Character string containing a regular expression to match file(s) in
the folder.
progressBar  (Optional) When FALSE, tell H2O parse call to block synchronously instead of
polling. This can be faster for small datasets but loses the progress bar.
parse_type  (Optional) Specify which parser type H2O will use. Valid types are "ARFF",
"XLS", "CSV", "SVMLight"

Details

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

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

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

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

See Also

h2o.import_sql_select, h2o.import_sql_table, h2o.parseRaw

Examples

## Not run:
h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
prostate = h2o.importFile(path = prostate_path)
class(prostate)
summary(prostate)

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

## End(Not run)
**h2o.import_hive_table**  
*Import Hive Table into H2O*

**Description**

Import Hive table to H2OFrame in memory. Make sure to start H2O with Hive on classpath. Uses hive-site.xml on classpath to connect to Hive. When database is specified as jdbc URL uses Hive JDBC driver to obtain table metadata. Then uses direct HDFS access to import data.

**Usage**

```r
h2o.import_hive_table(
  database,
  table,
  partitions = NULL,
  allow_multi_format = FALSE
)
```

**Arguments**

- **database**
  
  Name of Hive database (default database will be used by default), can be also a JDBC URL.

- **table**
  
  Name of Hive table to import.

- **partitions**
  
  A list of lists of strings - partition key column values of partitions you want to import.

- **allow_multi_format**
  
  Enable import of partitioned tables with different storage formats used. WARNING: this may fail on out-of-memory for tables with a large number of small partitions.

**Details**

For example, 

```r
my_citibike_data = h2o.import_hive_table("default", "citibike20k", partitions = list(c("2017", "01"), c("2017", "02")))
```

```r
my_citibike_data = h2o.import_hive_table("jdbc:hive2://hive-server:10000/default", "citibike20k", allow_multi_format = TRUE)
```

---

**h2o.import_mojo**  
*Imports a MOJO under given path, creating a Generic model with it.*

**Description**

Usage example: 

```r
mojo_model <- h2o.import_mojo(model_file_path = "/path/to/mojo.zip")
```

```r
predictions <- h2o.predict(mojo_model, dataset)
```

**Usage**

```r
h2o.import_mojo(mojo_file_path)
```
Arguments

mojo_file_path  Filesystem path to the model imported

Value

Returns H2O Generic Model embedding given MOJO model

Examples

```r
## Not run:

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

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

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

# Import the MOJO and obtain a Generic model
mojo_model <- h2o.import_mojo(mojo_original_path)

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

## End(Not run)
```

---

### h2o.import_sql_select

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

**Description**

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

**Usage**

```r
h2o.import_sql_select(
  connection_url,  # Connection URL to SQL Data Source
  select_query,    # SQL Query
  username,        # Username
  password,        # Password
  use_temp_table = NULL,  # Use temporary SQL table
  temp_table_name = NULL,  # Temporary SQL table name
  optimize = NULL,  # Optimize query execution
  fetch_mode = NULL  # Query fetch mode
)
```
h2o.import_sql_table

Description

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

Usage

h2o.import_sql_table(
    connection_url,
    table,
    username,
    password,
    columns = NULL,
    optimize = NULL,
    fetch_mode = NULL
)
**h2o.impute**

**Arguments**

- `connection_url`: URL of the SQL database connection as specified by the Java Database Connectivity (JDBC) Driver. For example, "jdbc:mysql://localhost:3306/menagerie?&useSSL=false"
- `table`: Name of SQL table
- `username`: Username for SQL server
- `password`: Password for SQL server
- `columns`: (Optional) Character vector of column names to import from SQL table. Default is to import all columns.
- `optimize`: (Optional) Optimize import of SQL table for faster imports. Default is true. Ignored - use `fetch_mode` instead.
- `fetch_mode`: (Optional) Set to DISTRIBUTED to enable distributed import. Set to SINGLE to force a sequential read from the database. Can be used for databases that do not support OFFSET-like clauses in SQL statements.

**Details**

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

---

**h2o.impute**

**Basic Imputation of H2O Vectors**

**Description**

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

**Usage**

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

**Arguments**

- `data`: The dataset containing the column to impute.
- `column`: A specific column to impute, default of 0 means impute the whole frame.
- `method`: "mean" replaces NAs with the column mean; "median" replaces NAs with the column median; "mode" replaces with the most common factor (for factor columns only);
h2o.init

**combine_method** If method is "median", then choose how to combine quantiles on even sample sizes. This parameter is ignored in all other cases.

**by** group by columns

**groupByFrame** Impute the column col with this pre-computed grouped frame.

**values** A vector of impute values (one per column). NaN indicates to skip the column

**Details**

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

**Value**

an H2OFrame with imputed values

**Examples**

```
## Not run:
h2o.init()
iris_hf <- as.h2o(iris)
iris_hf[sample(nrow(iris_hf), 40), 5] <- NA  # randomly replace 50 values with NA
  # impute with a group by
iris_hf <- h2o.impute(iris_hf, "Species", "mode", by = c("Sepal.Length", "Sepal.Width"))
## End(Not run)
```

---

**h2o.init**  
*Initialize and Connect to H2O*

**Description**

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

**Usage**

```
h2o.init(
  ip = "localhost",
  port = 54321,
  name = NA_character_,
  startH2O = TRUE,
  forceDL = FALSE,
  enable_assertions = TRUE,
  license = NULL,
  nthreads = -1,
  max_mem_size = NULL,
  min_mem_size = NULL,
  ice_root = tempdir(),
  log_dir = NA_character_,
  log_level = NA_character_,
  strict_version_check = TRUE,
```

proxy = NA_character_,
https = FALSE,
cacert = NA_character_,
insecure = FALSE,
username = NA_character_,
password = NA_character_,
use_spnego = FALSE,
cookies = NA_character_,
context_path = NA_character_,
ignore_config = FALSE,
extra_classpath = NULL,
jvm_custom_args = NULL,
bind_to_localhost = TRUE
)

Arguments

ip
Object of class character representing the IP address of the server where H2O is running.

port
Object of class numeric representing the port number of the H2O server.

name
(Optional) A character string representing the H2O cluster name.

startH2O
(Optional) A logical value indicating whether to try to start H2O from R if no connection with H2O is detected. This is only possible if ip = "localhost" or ip = "127.0.0.1". If an existing connection is detected, R does not start H2O.

forceDL
(Optional) A logical value indicating whether to force download of the H2O executable. Defaults to FALSE, so the executable will only be downloaded if it does not already exist in the h2o R library resources directory h2o/java/h2o.jar. This value is only used when R starts H2O.

enable_assertions
(Optional) A logical value indicating whether H2O should be launched with assertions enabled. Used mainly for error checking and debugging purposes. This value is only used when R starts H2O.

license
(Optional) A character string value specifying the full path of the license file. This value is only used when R starts H2O.

nthreads
(Optional) Number of threads in the thread pool. This relates very closely to the number of CPUs used. -1 means use all CPUs on the host (Default). A positive integer specifies the number of CPUs directly. This value is only used when R starts H2O.

max_mem_size
(Optional) A character string specifying the maximum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O. If max_mem_size is not defined, then the amount of memory that H2O allocates will be determined by the default memory of Java Virtual Machine. This amount is dependent on the Java version, but it will generally be 25 percent of the machine’s physical memory.

min_mem_size
(Optional) A character string specifying the minimum size, in bytes, of the memory allocation pool to H2O. This value must a multiple of 1024 greater than 2MB. Append the letter m or M to indicate megabytes, or g or G to indicate gigabytes. This value is only used when R starts H2O.
ice_root (Optional) A directory to handle object spillage. The default varies by OS.

log_dir (Optional) A directory where H2O server logs are stored. The default varies by OS.

log_level (Optional) The level of logging of H2O server. The default is INFO.

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

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

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

cacert (Optional) Path to a CA bundle file with root and intermediate certificates of trusted CAs.

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

username (Optional) Username to login with.

password (Optional) Password to login with.

use_spnego (Optional) Set this to TRUE to enable SPNEGO authentication.

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

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

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

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

jvm_custom_args (Optional) A character list of custom arguments for the JVM where new H2O instance is going to run, if started. Ignored when connecting to an existing instance.

bind_to_localhost (Optional) A logical flag indicating whether access to the H2O instance should be restricted to the local machine (default) or if it can be reached from other computers on the network. Only applicable when H2O is started from R.

Details

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

When initializing H2O locally, this method searches for h2o.jar in the R library resources (system.file("java", "h2o.

and if the file does not exist, it will automatically attempt to download the correct version from Amazon S3. The user must have Internet access for this process to be successful.

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

Value

this method will load it and return a H2OConnection object containing the IP address and port number of the H2O server.
h2o.insertMissingValues

Insert Missing Values into an H2OFrame

Description

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

Usage

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

Arguments

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

Returns an H2OFrame object.

WARNING

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

Examples

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

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

irismiss <- h2o.insertMissingValues(iris_hf, fraction = 0.25)
head(irismiss)
summary(irismiss)

## End(Not run)
```

**h2o.interaction**  
*Categorical Interaction Feature Creation in H2O*

**Description**

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

**Usage**

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

**Arguments**

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

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

Value

Returns an H2OFrame object.

Examples

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

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

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

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

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

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

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

## End(Not run)
```
Description

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

Usage

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

Arguments

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

Value

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

References

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

Examples

## Not run:
library(h2o)
h2o.init()
df <- h2o.createFrame(rows = 1, cols = 256, randomize = TRUE, value = 0,
  real_range = 100, categorical_fraction = 0, factors = 0,
  integer_fraction = 0, integer_range = 100, binary_fraction = 0,
  binary_ones_fraction = 0, time_fraction = 0, string_fraction = 0,
  missing_fraction = 0, has_response = FALSE, seed = 123)
df2 <- h2o.cumsum(df, axis = 1)
h2o.isax(df2, num_words = 10, max_cardinality = 10)
## End(Not run)
### h2o.ischaracter

**Description**
Check if character

**Usage**

```r
h2o.ischaracter(x)
```

**Arguments**

- `x` An H2OFrame object.

**See Also**

`is.character` for the base R implementation.

**Examples**

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

iris <- h2o.importFile(f)
iris_char <- h2o.ascharacter(iris["class"])
h2o.ischaracter(iris_char)

## End(Not run)
```

### h2o.isfactor

**Description**
Check if factor

**Usage**

```r
h2o.isfactor(x)
```

**Arguments**

- `x` An H2OFrame object.

**See Also**

`is.factor` for the base R implementation.
Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
h2o.isfactor(cars["economy_20mpg"])

## End(Not run)
```

---

**h2o.isnumeric**

Check if numeric

Description

Check if numeric

Usage

```r
h2o.isnumeric(x)
```

Arguments

- `x` An H2OFrame object.

See Also

- `is.numeric` for the base R implementation.

Examples

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

iris <- h2o.importFile(f)
h2o.isnumeric(iris["sepal_len"])

## End(Not run)
```
h2o.isolationForest  Trains an Isolation Forest model

Description

Trains an Isolation Forest model

Usage

```r
h2o.isolationForest(
  training_frame,
  x,
  model_id = NULL,
  score_each_iteration = FALSE,
  score_tree_interval = 0,
  ignore_const_cols = TRUE,
  ntrees = 50,
  max_depth = 8,
  min_rows = 1,
  max_runtime_secs = 0,
  seed = -1,
  build_tree_one_node = FALSE,
  mtries = -1,
  sample_size = 256,
  sample_rate = -1,
  col_sample_rate_change_per_level = 1,
  col_sample_rate_per_tree = 1,
  categorical_encoding = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit",
                          "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"),
  stopping_rounds = 0,
  stopping_metric = c("AUTO", "anomaly_score"),
  stopping_tolerance = 0.01,
  export_checkpoints_dir = NULL
)
```

Arguments

- `training_frame`  Id of the training data frame.
- `x`  A vector containing the character names of the predictors in the model.
- `model_id`  Destination id for this model; auto-generated if not specified.
- `score_each_iteration`  Logical. Whether to score during each iteration of model training. Defaults to FALSE.
- `score_tree_interval`  Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
- `ignore_const_cols`  Logical. Ignore constant columns. Defaults to TRUE.
- `ntrees`  Number of trees. Defaults to 50.
- `max_depth`  Maximum tree depth. Defaults to 8.
**min_rows**  
Fewest allowed (weighted) observations in a leaf. Defaults to 1.

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

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

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

**mtries**  
Number of variables randomly sampled as candidates at each split. If set to -1, defaults (number of predictors)/3. Defaults to -1.

**sample_size**  
Number of randomly sampled observations used to train each Isolation Forest tree. Only one of parameters sample_size and sample_rate should be defined. If sample_rate is defined, sample_size will be ignored. Defaults to 256.

**sample_rate**  
Rate of randomly sampled observations used to train each Isolation Forest tree. Needs to be in range from 0.0 to 1.0. If set to -1, sample_rate is disabled and sample_size will be used instead. Defaults to -1.

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

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

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

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

**stopping_metric**  
Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anonomy_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "anomaly_score". Defaults to AUTO.

**stopping_tolerance**  
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.01.

**export_checkpoints_dir**  
Automatically export generated models to this directory.

### Examples

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

# Import the cars dataset
```
cars <- h2o.importFile(f)

# Set the predictors
predictors <- c("displacement","power","weight","acceleration","year")

# Train the IF model
cars_if <- h2o.isolationForest(x = predictors, training_frame = cars,
   seed = 1234, stopping_metric = "MSE",
   stopping_rounds = 3, stopping_tolerance = 0.1)

## End(Not run)

---

### h2o.is_client

**Check Client Mode Connection**

**Description**

Check Client Mode Connection

**Usage**

h2o.is_client()

---

### h2o.keyof

**Method on Keyed objects allowing to obtain their key.**

**Description**

Method on Keyed objects allowing to obtain their key.

**Usage**

h2o.keyof(object)

## S4 method for signature 'Keyed'
h2o.keyof(object)

## S4 method for signature 'H2OModel'
h2o.keyof(object)

## S4 method for signature 'H2OGrid'
h2o.keyof(object)

## S4 method for signature 'H2OFrame'
h2o.keyof(object)

## S4 method for signature 'H2OAutoML'
h2o.keyof(object)
**h2o.kfold_column**

*Produce a k-fold column vector.*

**Arguments**

- `object`: A Keyed object

**Value**

- the string key holding the persistent object.

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

- Returns an H2OFrame object with fold assignments.

**Examples**

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

iris <- h2o.importFile(f)
kfolds <- h2o.kfold_column(iris, nfolds = 5, seed = 1234)
## End(Not run)
```

**h2o.killMinus3**

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

**Description**

A poor man’s profiler, but effective.

**Usage**

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

Performs k-means clustering on an H2O dataset

Description

Performs k-means clustering on an H2O dataset

Usage

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

Arguments

- `training_frame`: Id of the training data frame.
- `x`: A vector containing the character names of the predictors in the model.
- `model_id`: Destination id for this model; auto-generated if not specified.
- `validation_frame`: Id of the validation data frame.
- `nfolds`: Number of folds for K-fold cross-validation (0 to disable or 0 to 2). Defaults to 0.
- `keep_cross_validation_models`: Logical. Whether to keep the cross-validation models. Defaults to TRUE.
- `keep_cross_validation_predictions`: Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment
Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.

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

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

ignore_const_cols
Logical. Ignore constant columns. Defaults to TRUE.

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

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

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

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

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

standardize
Logical. Standardize columns before computing distances Defaults to TRUE.

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

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

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

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

export_checkpoints_dir
Automatically export generated models to this directory.

cluster_size_constraints
An array specifying the minimum number of points that should be in each cluster. The length of the constraints array has to be the same as the number of clusters.

Value
an object of class H2OClusteringModel.
See Also
h2o.cluster_sizes, h2o.totss, h2o.num_iterations, h2o.betweenss, h2o.tot_withinss, h2o.withinss, h2o.centersSTD, h2o.centers

Examples
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.kmeans(training_frame = prostate, k = 10, x = c("AGE", "RACE", "VOL", "GLEASON"))

## End(Not run)

h2o.kurtosis

Kurtosis of a column

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

Usage
h2o.kurtosis(x, ..., na.rm = TRUE)

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

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

Examples
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.kurtosis(prostate$AGE)

## End(Not run)
h2o.levels

Return the levels from the column requested column.

Description
Return the levels from the column requested column.

Usage
h2o.levels(x, i)

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

See Also
levels for the base R method.

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

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

## End(Not run)

h2o.listTimezones

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

Description
List all of the Time Zones Acceptable by the H2O cluster.

Usage
h2o.listTimezones()
**h2o.list_all_extensions**

*List all H2O registered extensions*

**Description**

List all H2O registered extensions

**Usage**

```python
h2o.list_all_extensions()
```

**h2o.list_api_extensions**

*List registered API extensions*

**Description**

List registered API extensions

**Usage**

```python
h2o.list_api_extensions()
```

**h2o.list_core_extensions**

*List registered core extensions*

**Description**

List registered core extensions

**Usage**

```python
h2o.list_core_extensions()
```

**h2o.list_jobs**

*Return list of jobs performed by the H2O cluster*

**Description**

Return list of jobs performed by the H2O cluster

**Usage**

```python
h2o.list_jobs()
```
h2o.loadGrid

**Description**

Loads previously saved grid with all its models from the same folder.

**Usage**

```r
h2o.loadGrid(grid_path)
```

**Arguments**

- `grid_path`  
  A character string containing the path to the file with the grid saved.

**Examples**

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

iris.hex <- as.h2o(iris)

ntrees_opts = c(1, 5)
learn_rate_opts = c(0.1, 0.01)
size_of_hyper_space = length(ntrees_opts) * length(learn_rate_opts)

hyper_parameters = list(ntrees = ntrees_opts, learn_rate = learn_rate_opts)
# Tempdir is chosen arbitrarily. May be any valid folder on an H2O-supported filesystem.
baseline_grid <- h2o.grid("gbm", grid_id="gbm_grid_test", x=1:4, y=5, training_frame=iris.hex, hyper_params = hyper_parameters, export_checkpoints_dir = tempdir())
# Remove everything from the cluster or restart it
h2o.removeAll()

grid <- h2o.loadGrid(paste0(tempdir(),"\",baseline_grid@grid_id))
## End(Not run)
```

h2o.loadModel

**Description**

Load H2O Model from HDFS or Local Disk.

**Usage**

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

**Arguments**

- `path`  
  The path of the H2O Model to be imported.
Value

Returns a H2OModel object of the class corresponding to the type of model loaded.

See Also

h2o.saveModel, H2OModel

Examples

## Not run:
# library(h2o)
# h2o.init()
# prostate_path = system.file("extdata", "prostate.csv", package = "h2o")
# prostate = h2o.importFile(path = prostate_path)
# prostate_glm = h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
# training_frame = prostate, family = "binomial", alpha = 0.5)
# glmmodel_path = h2o.saveModel(prostate_glm, dir = "/Users/UserName/Desktop")
# glmmodel_load = h2o.loadModel(glmmodel_path)

## End(Not run)

h2o.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.

Examples

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

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

h2o.log(frame)

## End(Not run)
**h2o.log10**

*Compute the log10 of x*

**Description**

Compute the log10 of x

**Usage**

h2o.log10(x)

**Arguments**

x  
An H2OFrame object.

**See Also**

log10 for the base R implementation.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.log10(frame)
## End(Not run)
```

---

**h2o.log1p**

*Compute the log1p of x*

**Description**

Compute the log1p of x

**Usage**

h2o.log1p(x)

**Arguments**

x  
An H2OFrame object.

**See Also**

log1p for the base R implementation.
### Examples

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

h2o.init()

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

h2o.log1p(frame)

## End(Not run)
```

---

**h2o.log2**  
*Compute the log2 of x*

### Description

Compute the log2 of x

### Usage

```r
h2o.log2(x)
```

### Arguments

- **x**  
  An H2OFrame object.

### See Also

- `log2` for the base R implementation.

### Examples

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

h2o.init()

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

h2o.log2(frame)

## End(Not run)
```
h2o.logAndEcho

**Log a message on the server-side logs**

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

**Usage**
h2o.logAndEcho(message)

**Arguments**
- **message** A character string with the message to write to the log.

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

h2o.logloss

**Retrieve the Log Loss Value**

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

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

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

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.splits <- h2o.splitFrame(data = cars, ratios = .8, seed = 1234)
train <- cars.splits[[1]]
valid <- cars.splits[[2]]
car_drf <- h2o.randomForest(x = predictors, y = response,
                           training_frame = train,
                           validation_frame = valid)
h2o.logloss(car_drf, train = TRUE, valid = TRUE)
## End(Not run)
```

---

### h2o.ls

#### List Keys on an H2O Cluster

**Description**

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

**Usage**

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

**Value**

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

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.ls()
## End(Not run)
```
**h2o.lstrip**

*Strip set from left*

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
string_to_lstrip <- as.h2o("1234567890")
lstrip_string <- h2o.lstrip(string_to_lstrip, "123") # Remove "123"
## End(Not run)
```

**h2o.mae**

*Retrieve the Mean Absolute Error Value*

**Description**

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

**Usage**

```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
### 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,
  weights = NULL
)
```

### 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)

#### Examples

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

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

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

h2o.mae(m)

## End(Not run)
```
Arguments

- **predicted**: An H2OFrame containing predictions.
- **actuals**: An H2OFrame containing actual values.
- **domain**: Vector with response factors for classification.
- **distribution**: Distribution for regression.
- **weights** (optional): An H2OFrame containing observation weights.

Value

Returns an object of the `H2OModelMetrics` subclass.

Examples

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate$CAPSULE <- as.factor(prostate$CAPSULE)
prostate_gbm <- h2o.gbm(3:9, "CAPSULE", prostate)
pred <- h2o.predict(prostate_gbm, prostate)[,3] ## class-1 probability
h2o.make_metrics(pred, prostate$CAPSULE)
## End(Not run)
```

Description

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

Usage

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

Arguments

- **x**: A categorical vector from an H2OFrame object with values to be matched.
- **table**: An R object to match `x` against.
- **nomatch**: The value to be returned in the case when no match is found.
- **incomparables**: A vector of 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
h2o.max

See Also

match for base R implementation.

Examples

## Not run:
h2o.init()
iris_hf <- as.h2o(iris)
h2o.match(iris_hf[, 5], c("setosa", "versicolor"))
## End(Not run)

h2o.max

Returns the maxima of the input values.

Description

Returns the maxima of the input values.

Usage

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

Arguments

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

See Also

max for the base R implementation.

Examples

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

iris <- h2o.importFile(f)
h2o.max(iris["petal_len"], na.rm = TRUE)
## End(Not run)
**h2o.mean**

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

**Description**

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

**Usage**

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

## S3 method for class 'H2OFrame'

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
# Default behavior. Will return list of means per column.
h2o.mean(prostate$AGE)
# return_frame set to TRUE. This will return an H2O Frame
# with mean per row or column (depends on axis argument)
h2o.mean(prostate, na.rm=TRUE, axis=1, return_frame=TRUE)

## End(Not run)
```
h2o.mean_per_class_error

Retrieve the mean per class error

Description

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

Usage

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

Arguments

object
An H2OBinomialMetrics object.

train
Retrieve the training mean per class error

valid
Retrieve the validation mean per class error

xval
Retrieve the cross-validation mean per class error

See Also

h2o.mse for MSE, and h2o.metric for the various threshold metrics. See h2o.performance for creating H2OModelMetrics objects.

Examples

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

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

## End(Not run)
h2o.mean_residual_deviance

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

Usage

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

Arguments

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

Examples

## Not run:
library(h2o)

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

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

h2o.mean_residual_deviance(m)

## End(Not run)

h2o.median

H2O Median

Description

Compute the median of an H2OFrame.

Usage

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

## S3 method for class 'H2OFrame'
median(x, na.rm = TRUE)
h2o.melt

Converts a frame to key-value representation while optionally skipping NA values. Inverse operation to h2o.pivot.

Description

Pivot the frame designated by the three columns: index, column, and value. Index and column should be of type enum, int, or time. For cases of multiple indexes for a column label, the aggregation method is to pick the first occurrence in the data frame.

Usage

```r
h2o.melt(
  x,
  id_vars,
  value_vars = NULL,
  var_name = "variable",
  value_name = "value",
  skipna = FALSE
)
```

Arguments

- `x`: an H2OFrame
- `id_vars`: the columns used as identifiers
- `value_vars`: what columns will be converted to key-value pairs (optional, if not specified complement to id_vars will be used)
- `var_name`: name of the key-column (default: "variable")
- `value_name`: name of the value-column (default: "value")
- `skipna`: if enabled, do not include NAs in the result (default: FALSE)

Examples

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.median(prostate)
## End(Not run)
```
**h2o.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**

```r
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

```r
## Not run:
library(h2o)
h2o.init()
left <- data.frame(fruit = c("apple", "orange", "banana", "lemon", "strawberry", "blueberry"),
                   color = c("red", "orange", "yellow", "yellow", "red", "blue"))
right <- data.frame(fruit = c("apple", "orange", "banana", "lemon", "strawberry", "watermelon"),
                    citrus = c(FALSE, TRUE, FALSE, TRUE, FALSE, FALSE))
left_hf <- as.h2o(left)
right_hf <- as.h2o(right)
merged <- h2o.merge(left_hf, right_hf, all.x = TRUE)

## End(Not run)
```

**h2o.metric**  
*H2O Model Metric Accessor Functions*

**Description**
A series of functions that retrieve model metric details.

**Usage**

```r
h2o.metric(object, thresholds, metric, transform = NULL)
h2o.F0point5(object, thresholds)
h2o.F1(object, thresholds)
h2o.F2(object, thresholds)
h2o.accuracy(object, thresholds)
h2o.error(object, thresholds)
h2o.maxPerClassError(object, thresholds)
h2o.mean_per_class_accuracy(object, thresholds)
h2o.mcc(object, thresholds)
h2o.precision(object, thresholds)
h2o.tpr(object, thresholds)
h2o.fpr(object, thresholds)
h2o.fnr(object, thresholds)
h2o.tnr(object, thresholds)
h2o.recall(object, thresholds)
```
h2o.sensitivity(object, thresholds)

h2o.fallout(object, thresholds)

h2o.missrate(object, thresholds)

h2o.specificity(object, thresholds)

Arguments

object

An H2OModelMetrics object of the correct type.

thresholds

(Optional) A value or a list of values between 0.0 and 1.0. If not set, then all
thresholds will be returned. If "max", then the threshold maximizing the metric
will be used.

metric

(Optional) the metric to retrieve. If not set, then all metrics will be returned.

transform

(Optional) a list describing a transformer for the given metric, if any. e.g. trans-
form=list(op=foo_fn, name="foo") will rename the given metric to "foo" and
apply function foo_fn to the metric values.

Details

Many of these functions have an optional thresholds parameter. Currently only increments of 0.1
are allowed. If not specified, the functions will return all possible values. Otherwise, the function
will return the value for the indicated threshold.

Currently, 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
## Not run:
library(h2o)
h2o.init()

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

prostate$CAPSULE <- as.factor(prostate$CAPSULE)
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
perf <- h2o.performance(model, prostate)
h2o.F1(perf)
```

## End(Not run)
h2o.min

Returns the minima of the input values.

Description

Returns the minima of the input values.

Usage

h2o.min(x, na.rm = FALSE)

Arguments

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

See Also

- min for the base R implementation.

Examples

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

iris <- h2o.importFile(f)
h2o.min(iris["sepal_len"], na.rm = TRUE)
## End(Not run)
```

h2o.mktime

Compute msec since the Unix Epoch

Description

Compute msec since the Unix Epoch

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Defaults to 1970</td>
</tr>
<tr>
<td>month</td>
<td>zero based (months are 0 to 11)</td>
</tr>
<tr>
<td>day</td>
<td>zero based (days are 0 to 30)</td>
</tr>
<tr>
<td>hour</td>
<td>hour</td>
</tr>
<tr>
<td>minute</td>
<td>minute</td>
</tr>
<tr>
<td>second</td>
<td>second</td>
</tr>
<tr>
<td>msec</td>
<td>msec</td>
</tr>
</tbody>
</table>

Examples

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

x = as.h2o(c(2018, 3, 2, 6, 32, 0, 0))
h2o.mktime(x)

## End(Not run)
```

Description

Provides the method `h2o.mojo_predict_csv` with which you can predict a MOJO model from R.

Usage

```r
h2o.mojo_predict_csv(
  input_csv_path,
  mojo_zip_path,
  output_csv_path = NULL,
  genmodel_jar_path = NULL,
  classpath = NULL,
  java_options = NULL,
  verbose = F,
  setInvNumNA = F
)
```

Arguments

- `input_csv_path`: Path to input CSV file.
- `mojo_zip_path`: Path to MOJO zip downloaded from H2O.
- `output_csv_path`: Optional, path to the output CSV file with computed predictions. If NULL (default), then predictions will be saved as prediction.csv in the same folder as the MOJO zip.
**h2o.mojo_predict_df**

Returns a data.frame containing computed predictions

**Value**

Provides the method `h2o.mojo_predict_df` with which you can predict a MOJO model from R.

**Usage**

```r
h2o.mojo_predict_df(
  frame,         
  mojo_zip_path, 
  genmodel_jar_path = NULL,  
  classpath = NULL, 
  java_options = NULL, 
  verbose = F, 
  setInvNumNA = F 
)
```

**Arguments**

- `frame`: data.frame to score.
- `mojo_zip_path`: Path to MOJO zip downloaded from H2O.
- `genmodel_jar_path`: Optional, path to genmodel jar file. If NULL (default) then the h2o-genmodel.jar in the same folder as the MOJO zip will be used.
- `classpath`: Optional, specifies custom user defined classpath which will be used when scoring. If NULL (default) then the default classpath for this MOJO model will be used.
- `java_options`: Optional, custom user defined options for Java. By default `-Xmx4g -XX:ReservedCodeCacheSize=256m` is used.
- `verbose`: Optional, if TRUE, then additional debug information will be printed. FALSE by default.
- `setInvNumNA`: Optional, if TRUE, then for an string that cannot be parsed into a number an N/A value will be produced, if false the command will fail. FALSE by default.
**h2o.month**

Optional, if TRUE, then additional debug information will be printed. FALSE by default.

Optional, if TRUE, then for an string that cannot be parsed into a number an N/A value will be produced, if false the command will fail. FALSE by default.

**Value**

Returns a data.frame containing computed predictions

---

### h2o.month

**Convert Milliseconds to Months in H2O Datasets**

**Description**

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

**Usage**

```r
h2o.month(x)
```

```r
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**

```r
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**

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

---

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

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

## End(Not run)
```

---

**h2o.naiveBayes**

*Compute naive Bayes probabilities on an H2O dataset.*

**Description**

The naive Bayes classifier assumes independence between predictor variables conditional on the response, and a Gaussian distribution of numeric predictors with mean and standard deviation computed from the training dataset. When building a naive Bayes classifier, every row in the training dataset that contains at least one NA will be skipped completely. If the test dataset has missing values, then those predictors are omitted in the probability calculation during prediction.

**Usage**

```r
h2o.naiveBayes(
  x,
  y,
  training_frame,
  model_id = NULL,
  nfolds = 0,
  seed = -1,
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  fold_column = NULL,
  keep_cross_validation_models = TRUE,
  keep_cross_validation_predictions = FALSE,
  keep_cross_validation_fold_assignment = FALSE,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  balance_classes = FALSE,
  class_sampling_factors = NULL,
  max_after_balance_size = 5,
  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,
export_checkpoints_dir = 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.
nfolds  Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
seed  Seed for random numbers (affects certain parts of the algo that are stochastic and those might or might not be enabled by default). Defaults to -1 (time-based random number).

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

fold_column  Column with cross-validation fold index assignment per observation.
keep_cross_validation_models  Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions  Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment  Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
validation_frame  Id of the validation data frame.
ignore_const_cols  Logical. Ignore constant columns. Defaults to TRUE.

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

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

class_sampling_factors  Desired over/under-sampling ratios per class (in lexicographic order). If not specified, sampling factors will be automatically computed to obtain class balance during training. Requires balance_classes.
max_after_balance_size
  Maximum relative size of the training data after balancing class counts (can be less than 1.0). Requires balance_classes. Defaults to 5.0.

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

export_checkpoints_dir
  Automatically export generated models to this directory.

Value

an object of class H2OBinomialModel if the response has two categorical levels, and H2OMultinomialModel otherwise.

Examples

```r
## Not run:
h2o.init()
votes_path <- system.file("extdata", "housevotes.csv", package = "h2o")
votes <- h2o.uploadFile(path = votes_path, header = TRUE)
h2o.naiveBayes(x = 2:17, y = 1, training_frame = votes, laplace = 3)
## End(Not run)
```

h2o.names

Column names of an H2OFrame

Description

Column names of an H2OFrame

Usage

h2o.names(x)
h2o.na_omit

Remove Rows With NAs

Description
Remove Rows With NAs

Usage
h2o.na_omit(object, ...)

Arguments

object H2OFrame object

... Ignored

Value
Returns an H2OFrame object containing non-NA rows.

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

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

## End(Not run)
### h2o.nchar

**String length**

**Description**
String length

**Usage**

`h2o.nchar(x)`

**Arguments**

- `x` The column whose string lengths will be returned.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
string_to_nchar <- as.h2o("r tutorial")
nchar_string <- h2o.nchar(string_to_nchar)
## End(Not run)
```

### h2o.ncol

**Return the number of columns present in x.**

**Description**
Return the number of columns present in x.

**Usage**

`h2o.ncol(x)`

**Arguments**

- `x` An H2OFrame object.

**See Also**

`ncol` for the base R implementation.
Examples

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

iris <- h2o.importFile(f)
h2o.ncol(iris)

## End(Not run)

---

h2o.networkTest  View Network Traffic Speed

Description

View speed with various file sizes.

Usage

h2o.networkTest()

Value

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

---

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

Description

Get the number of factor levels for this frame.

Usage

h2o.nlevels(x)

Arguments

x  An H2OFrame object.

See Also

nlevels for the base R method.
### Examples

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

cars <- h2o.importFile(f)
h2o.nlevels(cars)

## End(Not run)
```

-----

### Disable Progress Bar

**Description**

Disable Progress Bar

**Usage**

```r
h2o.no_progress()
```

**Examples**

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

iris <- h2o.importFile(f)
iris["class"] <- as.factor(iris["class"])
predictors <- c("sepal_len","sepal_wid","petal_len","petal_wid")
splits <- h2o.splitFrame(iris, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]

iris_km <- h2o.kmeans(x = predictors,
                      training_frame = train,
                      validation_frame = valid,
                      k = 10, estimate_k = TRUE,
                      standardize = FALSE, seed = 1234)

## End(Not run)
```
h2o.nrow

Return the number of rows present in x.

Description

Return the number of rows present in x.

Usage

h2o.nrow(x)

Arguments

x

An H2OFrame object.

See Also

nrow for the base R implementation.

Examples

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

cars <- h2o.importFile(f)
h2o.nrow(cars)

## End(Not run)

h2o.null_deviance

Retrieve the null deviance

Description

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

Usage

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

Arguments

object

An H2OModel or H2OModelMetrics

train

Retrieve the training null deviance

valid

Retrieve the validation null deviance

xval

Retrieve the cross-validation null deviance
Examples

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
                          training_frame = prostate, family = "binomial", nfolds = 0,
                          alpha = 0.5, lambda_search = FALSE)
h2o.null_deviance(prostate_glm, train = TRUE)
## End(Not run)
```

### h2o.null_dof

**Retrieve the null degrees of freedom**

#### Description

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

#### Usage

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

#### Arguments

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

#### Examples

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
                        training_frame = prostate, family = "binomial", nfolds = 0,
                        alpha = 0.5, lambda_search = FALSE)
h2o.null_dof(prostate_glm, train = TRUE)
## End(Not run)
```
**h2o.num_iterations**

Retrieve the number of iterations.

**Description**

Retrieve the number of iterations.

**Usage**

`h2o.num_iterations(object)`

**Arguments**

- `object` An `H2OClusteringModel` object.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.num_iterations(prostate_glm)
## End(Not run)
```

---

**h2o.num_valid_substrings**

`Count of substrings >= 2 chars that are contained in file`

**Description**

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

**Usage**

`h2o.num_valid_substrings(x, path)`

**Arguments**

- `x` The column on which to calculate the number of valid substrings.
- `path` Path to text file containing line-separated strings to be referenced.
**h2o.openLog**

**View H2O R Logs**

**Description**

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

**Usage**

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

**Arguments**

- `type` Currently unimplemented.

**See Also**

- `h2o.startLogging`, `h2o.stopLogging`, `h2o.clearLog`

**Examples**

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

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

# Not run to avoid windows being opened during R CMD check
# h2o.openLog("Command")
# h2o.openLog("Error")

## End(Not run)
```

---

**h2o.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,
skipped_columns = NULL,
custom_non_data_line_markers = 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. If skipped_columns are specified, only list column names of columns that are not skipped.
col.types (Optional) A vector specifying the types to attempt to force over columns. If skipped_columns are specified, only list column types of columns that are not skipped.
na.strings (Optional) H2O will interpret these strings as missing.
blocking (Optional) Tell H2O parse call to block synchronously instead of polling. This can be faster for small datasets but loses the progress bar.
parse_type (Optional) Specify which parser type H2O will use. Valid types are "ARFF", "XLS", "CSV", "SVMLight"
chunk_size size of chunk of (input) data in bytes
decrypt_tool (Optional) Specify a Decryption Tool (key-reference acquired by calling h2o.decryptionSetup.
skipped_columns a list of column indices to be excluded from parsing
custom_non_data_line_markers (Optional) If a line in imported file starts with any character in given string it will NOT be imported. Empty string means all lines are imported, NULL means that default behaviour for given format will be used

Details
Parse the Raw Data produced by the import phase.

See Also
h2o.importFile, h2o.parseSetup
h2o.parseSetup

Get a parse setup back for the staged data.

Description

Get a parse setup back for the staged data.

Usage

h2o.parseSetup(
  data,
  pattern = "",
  destination_frame = "",
  header = NA,
  sep = "",
  col.names = NULL,
  col.types = NULL,
  na.strings = NULL,
  parse_type = NULL,
  chunk_size = NULL,
  decrypt_tool = NULL,
  skipped_columns = NULL,
  custom_non_data_line_markers = NULL
)

Arguments

data An H2OFrame object to be parsed.

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

destination_frame (Optional) The hex key assigned to the parsed file.

header (Optional) A logical value indicating whether the first row is the column header. If missing, H2O will automatically try to detect the presence of a header.

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

col.names (Optional) An H2OFrame object containing a single delimited line with the column names for the file. If skipped_columns are specified, only list column names of columns that are not skipped.

col.types (Optional) A vector specifying the types to attempt to force over columns. If skipped_columns are specified, only list column types of columns that are not skipped.

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

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

chunk_size size of chunk of (input) data in bytes

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

a list of column indices to be excluded from parsing

custom_non_data_line_markers

(Optional) If a line in imported file starts with any character in given string it will NOT be imported. Empty string means all lines are imported, NULL means that default behaviour for given format will be used

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 random-Forest’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,
  weight_column = -1,
  include_na = FALSE,
  user_splits = NULL,
  col_pairs_2dpdp = NULL,
  save_to = NULL,
  row_index = -1
)
```

**Arguments**

- **object**: An H2OModel object.
- **data**: An H2OFrame object used for scoring and constructing the plot.
- **cols**: Feature(s) for which partial dependence will be calculated.
- **destination_key**: An key reference to the created partial dependence tables in H2O.
- **nbins**: Number of bins used. For categorical columns make sure the number of bins exceeds the level count. If you enable add_missing_NA, the returned length will be nbins+1.
- **plot**: A logical specifying whether to plot partial dependence table.
plot_stddev A logical specifying whether to add std err to partial dependence plot.

weight_column A string denoting which column of data should be used as the weight column.

include_na A logical specifying whether missing value should be included in the Feature values.

user_splits A two-level nested list containing user defined split points for pdp plots for each column. If there are two columns using user defined split points, there should be two lists in the nested list. Inside each list, the first element is the column name followed by values defined by the user.

col_pairs_2dpdp A two-level nested list like this: col_pairs_2dpdp = list(c("col1_name", "col2_name"), c("col1_name", "col3_name"), ....) where a 2D partial plots will be generated for col1_name, col2_name pair, for col1_name, col3_name pair and whatever other pairs that are specified in the nested list.

save_to Fully qualified prefix of the image files the resulting plots should be saved to, e.g. '/home/user/pdp'. Plots for each feature are saved separately in PNG format, each file receives a suffix equal to the corresponding feature name, e.g. '/home/user/pdp_AGE.png'. If the files already exists, they will be overridden. Files are only saves if plot = TRUE (default).

row_index Row for which partial dependence will be calculated instead of the whole input frame.

Value
Plot and list of calculated mean response tables for each feature requested.

Examples
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate[, "CAPSULE"] <- as.factor(prostate[, "CAPSULE"])
prostate[, "RACE"] <- as.factor(prostate[, "RACE"])
prostate_gbm <- h2o.gbm(x = c("AGE", "RACE"),
y = "CAPSULE",
training_frame = prostate,
ntrees = 10,
max_depth = 5,
learn_rate = 0.1)
h2o.partialPlot(object = prostate_gbm, data = prostate, cols = c("AGE", "RACE"))
## End(Not run)

h2o.performance Model Performance Metrics in H2O

Description
Given a trained h2o model, compute its performance on the given dataset. However, if the dataset does not contain the response/target column, no performance will be returned. Instead, a warning message will be printed.
Usage

```r
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).
  
  Note: when the trained h2o model uses balance_classes, the training metrics constructed during training will be from the balanced training dataset. For more information visit: [https://0xdata.atlassian.net/browse/TN-9](https://0xdata.atlassian.net/browse/TN-9)
- `valid`: A logical value indicating whether to return the validation metrics (constructed during training).
- `xval`: A logical value indicating whether to return the cross-validation metrics (constructed during training).
- `data`: (DEPRECATED) An H2OFrame. This argument is now called ‘newdata’.

Value

Returns an object of the H2OModelMetrics subclass.

Examples

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

## If model uses balance_classes
## the results from train = TRUE will not match the results from newdata = prostate.hex
prostate_gbm_balanced <- h2o.gbm(3:9, "CAPSULE", prostate, balance_classes = TRUE)
h2o.performance(model = prostate_gbm_balanced, newdata = prostate)
h2o.performance(model = prostate_gbm_balanced, train = TRUE)

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

**Examples**

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

df = h2o.createFrame(rows = 1000, cols=3, factors=10, integer_fraction=1.0/3,
categorical_fraction=1.0/3, missing_fraction=0.0, seed=123)
df$C3 = h2o.abs(df$C3)
h2o.pivot(df,index="C3",column="C2",value="C1")
## End(Not run)
```

**h2o.prcomp**  

**Principal component analysis of an H2O data frame**

**Description**

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

\[
\text{h2o.prcomp}( \\
\text{training_frame}, \\
x, \\
\text{model_id} = \text{NULL}, \\
\text{validation_frame} = \text{NULL}, \\
\text{ignore_const_cols} = \text{TRUE}, \\
\text{score_each_iteration} = \text{FALSE}, \\
\text{transform} = \text{c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE")}, \\
\text{pca_method} = \text{c("GramSVD", "Power", "Randomized", "GLRM")}, \\
\text{pca_impl} = \text{c("MTJ_EVD_DENSEMATRIX", "MTJ_EVD_SYMMATRIX", "MTJ_SVD_DENSEMATRIX", "JAMA")}, \\
k = 1, \\
\text{max_iterations} = 1000, \\
\text{use_all_factor_levels} = \text{FALSE}, \\
\text{compute_metrics} = \text{TRUE}, \\
\text{impute_missing} = \text{FALSE}, \\
\text{seed} = -1, \\
\text{max_runtime_secs} = 0, \\
\text{export_checkpoints_dir} = \text{NULL} \\
) 
\]

Arguments

- **training_frame**: Id of the training data frame.
- **x**: A vector containing the character names of the predictors in the model.
- **model_id**: Destination id for this model; auto-generated if not specified.
- **validation_frame**: Id of the validation data frame.
- **ignore_const_cols**: Logical. Ignore constant columns. Defaults to TRUE.
- **score_each_iteration**: Logical. Whether to score during each iteration of model training. Defaults to FALSE.
- **transform**: Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
- **pca_method**: Specify the algorithm to use for computing the principal components: GramSVD - uses a distributed computation of the Gram matrix, followed by a local SVD; Power - computes the SVD using the power iteration method (experimental); Randomized - uses randomized subspace iteration method; GLRM - fits a generalized low-rank model with L2 loss function and no regularization and solves for the SVD using local matrix algebra (experimental) Must be one of: "GramSVD", "Power", "Randomized", "GLRM". Defaults to GramSVD.
k  Rank of matrix approximation Defaults to 1.

max_iterations Maximum training iterations Defaults to 1000.

use_all_factor_levels Logical. Whether first factor level is included in each categorical expansion Defaults to FALSE.

compute_metrics Logical. Whether to compute metrics on the training data Defaults to TRUE.

impute_missing Logical. Whether to impute missing entries with the column mean Defaults to FALSE.

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

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

export_checkpoints_dir Automatically export generated models to this directory.

Value

an object of class H2ODimReductionModel.

References


See Also

h2o.svd, h2o.glrm

Examples

## Not run:
library(h2o)
h2o.init()
australia_path <- system.file("extdata", "australia.csv", package = "h2o")
australia <- h2o.uploadFile(path = australia_path)
h2o.pcomp(training_frame = australia, k = 8, transform = "STANDARDIZE")

## End(Not run)
Usage

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

Arguments

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

Value

Returns an H2OFrame object with probabilities and default predictions.

h2o.predict_json  H2O Prediction from R without having H2O running

Description

Provides the method h2o.predict with which you can predict a MOJO or POJO Jar model from R.

Usage

h2o.predict_json(model, json, genmodelpath, labels, classpath, javaoptions)

Arguments

model  String with file name of MOJO or POJO Jar
json  JSON String with inputs to model
genmodelpath  (Optional) path name to h2o-genmodel.jar, if not set defaults to same dir as MOJO
labels  (Optional) if TRUE then show output labels in result
classpath  (Optional) Extra items for the class path of where to look for Java classes, e.g., h2o-genmodel.jar
javaoptions  (Optional) Java options string, default if "-Xmx4g"

Value

Returns an object with the prediction result

Examples

## Not run:
library(h2o)
h2o.predict_json("~/GBM_model_python_1473313897851_6.zip", '{"C7":1}')
h2o.predict_json("~/GBM_model_python_1473313897851_6.zip", '{"C7":1}', c('.', "lib"))

## End(Not run)
h2o.print

Print An H2OFrame

Description

Print An H2OFrame

Usage

h2o.print(x, n = 6L)

Arguments

x

An H2OFrame object

n

An (Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).

Examples

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

iris <- h2o.importFile(f)
h2o.print(iris["species"], n = 15)

## End(Not run)

h2o.prod

Return the product of all the values present in its arguments.

Description

Return the product of all the values present in its arguments.

Usage

h2o.prod(x)

Arguments

x

An H2OFrame object.

See Also

prod for the base R implementation.
h2o.proj_archetypes

Convert Archetypes to Features from H2O GLRM Model

Description

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

Usage

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

Arguments

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

Value

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

See Also

h2o.glrm for making an H2ODimReductionModel.

Examples

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

iris <- h2o.importFile(f)
h2o.prod(iris["petal_len"])

## End(Not run)

# Convert Archetypes to Features from H2O GLRM Model

h2o.proj_archetypes(iris_glrm, iris_hf)
**h2o.psvm**

*Trains a Support Vector Machine model on an H2O dataset*

**Description**

Alpha version. Supports only binomial classification problems.

**Usage**

```r
h2o.psvm(
  x, y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  hyper_param = 1,
  kernel_type = c("gaussian"),
  gamma = -1,
  rank_ratio = -1,
  positive_weight = 1,
  negative_weight = 1,
  disable_training_metrics = TRUE,
  sv_threshold = 1e-04,
  fact_threshold = 1e-05,
  feasible_threshold = 0.001,
  surrogate_gap_threshold = 0.001,
  mu_factor = 10,
  max_iterations = 200,
  seed = -1
)
```

**Arguments**

- **x** *(Optional)* A vector containing the names or indices of the predictor variables to use in building the model. If x is missing, then all columns except y are used.
- **y** The name or column index of the response variable in the data. The response must be either a binary categorical/factor variable or a numeric variable with values -1/1 (for compatibility with SVMlight format).
- **training_frame** Id of the training data frame.
- **model_id** Destination id for this model; auto-generated if not specified.
- **validation_frame** Id of the validation data frame.
- **ignore_const_cols** Logical. Ignore constant columns. Defaults to TRUE.
- **hyper_param** Penalty parameter C of the error term Defaults to 1.
- **kernel_type** Type of used kernel Must be one of: "gaussian". Defaults to gaussian.
- **gamma** Coefficient of the kernel (currently RBF gamma for gaussian kernel, -1 means 1/#features) Defaults to -1.
rank_ratio  Desired rank of the ICF matrix expressed as an ration of number of input rows (-1 means use sqrt(#rows)). Defaults to -1.

positive_weight  Weight of positive (+1) class of observations Defaults to 1.

negative_weight  Weight of positive (-1) class of observations Defaults to 1.

disable_training_metrics  Logical. Disable calculating training metrics (expensive on large datasets) Defaults to TRUE.

tovthreshold  Threshold for accepting a candidate observation into the set of support vectors Defaults to 0.0001.

fact_threshold  Convergence threshold of the Incomplete Cholesky Factorization (ICF) Defaults to 1e-05.

feasible_threshold  Convergence threshold for primal-dual residuals in the IPM iteration Defaults to 0.001.

surrogate_gap_threshold  Feasibility criterion of the surrogate duality gap (eta) Defaults to 0.001.

mu_factor  Increasing factor mu Defaults to 10.

max_iterations  Maximum number of iteration of the algorithm Defaults to 200.

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

Examples

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

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

# Train the Support Vector Machine model
svm_model <- h2o.psvm(gamma = 0.01, rank_ratio = 0.1, 
y = "C1", training_frame = splice, 
disable_training_metrics = FALSE)

## End(Not run)
```

---

**h2o.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

```r
## Not run:
# Request quantiles for an H2O parsed data set:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
# Request quantiles for a subset of columns in an H2O parsed data set
quantile(prostate[,3])
for(i in 1:ncol(prostate))
  quantile(prostate[, i])

## End(Not run)
```
h2o.r2

Retrieve the R2 value

Description

Retrieves the R2 value from an H2O model. Will return R^2 for GLM Models 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

## Not run:
library(h2o)

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

m <- h2o.glm(x = 2:5, y = 1, training_frame = fr)
h2o.r2(m)

## End(Not run)

h2o.randomForest

Build a Random Forest model

Description

Builds a Random Forest model on an H2OFrame.
**Usage**

```r
h2o.randomForest(
  x,
  y,
  training_frame, 
  model_id = NULL, 
  validation_frame = NULL, 
  nfolds = 0, 
  keep_cross_validation_models = TRUE, 
  keep_cross_validation_predictions = FALSE, 
  keep_cross_validation_fold_assignment = FALSE, 
  score_each_iteration = FALSE, 
  score_tree_interval = 0, 
  fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"), 
  fold_column = NULL, 
  ignore_const_cols = TRUE, 
  offset_column = NULL, 
  weights_column = NULL, 
  balance_classes = FALSE, 
  class_sampling_factors = NULL, 
  max_after_balance_size = 5, 
  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", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", 
                  "custom", "custom_increasing"), 
  stopping_tolerance = 0.001, 
  max_runtime_secs = 0, 
  seed = -1, 
  build_tree_one_node = FALSE, 
  mtries = -1, 
  sample_rate = 0.632, 
  sample_rate_per_class = NULL, 
  binomial_double_trees = FALSE, 
  checkpoint = NULL, 
  col_sample_rate_change_per_level = 1, 
  col_sample_rate_per_tree = 1, 
  min_split_improvement = 1e-05, 
  histogram_type = c("AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", 
                   "RoundRobin"), 
  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,
export_checkpoints_dir = NULL,
check_constant_response = TRUE,
verbose = FALSE)
)

Arguments

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

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

training_frame Id of the training data frame.
model_id Destination id for this model; auto-generated if not specified.
validation_frame Id of the validation data frame.
nfolds Number of folds for K-fold cross-validation (0 to disable or >= 2). Defaults to 0.
keep_cross_validation_models Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
score_each_iteration Logical. Whether to score during each iteration of model training. Defaults to FALSE.
score_tree_interval Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
fold_assignment Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems. Must be one of: "AUTO", "Random", "Modulo", "Stratified". Defaults to AUTO.
fold_column Column with cross-validation fold index assignment per observation.
ignore_const_cols Logical. Ignore constant columns. Defaults to TRUE.
offset_column Offset column. This argument is deprecated and has no use for Random Forest.
weights_column Column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed. Note: Weights are per-row observation weights and do not increase the size of the data frame. This is typically the number of times a row is repeated, but non-integer values are supported as well. During training, rows with higher weights matter more, due to the larger loss function pre-factor.
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 and anomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

stopping_tolerance
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

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

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

build_tree_one_node
Logical. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
mtries Number of variables randomly sampled as candidates at each split. If set to -1, defaults to \( \sqrt{p} \) 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 0.632.

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

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

checkpoint Model checkpoint to resume training with.

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

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

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

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

export_checkpoints_dir Automatically export generated models to this directory.

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

verbose Logical. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

Value

Creates a H2OModel object of the right type.
**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**

```r
h2o.range(x, na.rm = FALSE, finite = FALSE)
```

**Arguments**

- `x`: An H2OFrame object.
- `na.rm`: logical, indicating whether missing values should be removed.
- `finite`: logical, indicating if all non-finite elements should be omitted.

**See Also**

- `range` for the base R implementation.
Examples

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

iris <- h2o.importFile(f)
h2o.range(iris["petal_len"], na.rm = TRUE, finite = TRUE)

## End(Not run)
```

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 Column_to_arrange_by num_1 fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column 12 1 2941.552 1 3 -3177.9077 1 12 1 2941.552 1 5 -13311.8247 1 12 2 -22722.174 1 3 -3177.9077 1 12 2 -22722.174 1 5 -13311.8247 1 13 3 -12776.884 1 15 -18421.6171 0 13 3 -12776.884 1 4 28080.1607 0 15 3 -16995.346 1 11 -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 1 17 2 -13465.496 1 2 12094.4851 1 17 2 -13465.496 1 3 -11772.1338 1 17 2 -13465.496 1 3 -415.1114 0 17 2 -3329.6191 1 2 12094.4851 1 17 2 -3329.6191 1 3 -415.1114 0
```

If the following commands are issued:

```r
rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE))
```

The returned frame `rankedF1` will look like this:

```r
ID Group_by_column num fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column 15 3 -16995.346 1 1 -9781.6373 0 1 17 2 -13465.496 0 2 12094.4851 1 17 2 -3329.619 0 2 12094.4851 1 17 2 -3329.619 0 2 -13465.496 0 3 -415.1114 0 3 -11772.1338 1 6 17 2 -3329.619 0 3 -415.1114 0 7 12 2 -22722.174 1 5 -13311.8247 1 8 15 3 -16995.346 1 1 -9781.6373 0 1 16 3 26052.495 0 3 -61284.6900 0 2 16 3 -22905.288 1 3 -61284.6900 0 3 13 3 -12776.884 1 4 28080.1607 0 4 13 3 -12776.884 1 5 -18421.6171 0 5
```

If the following commands are issued:

```r
rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE), sort_cols_sorted=TRUE)
```

The returned frame will be sorted according to sortCols and hence look like this instead:

```r
ID Group_by_column num fdata Column_to_arrange_by num_1 fdata.1 New_Rank_column 12 1 2941.552 1 3 -3177.9077 1 11 16 1 -10003.593 0 3 -61284.6900 0 2 13 1 -6049.830 0 4 28080.1607 0 3 12 1 2941.552 1 5 -13311.8247 1 4 13 1 -6049.830 0 5 -18421.6171 0 5 17 2 -13465.496 0 2 12094.4851 1 17 2 -3329.619 0 2 -12094.4851 1 2 12 -22722.174 1 3 -3177.9077 1 3 17 2 -13465.496 0 3 -415.1114 0 5 17 2 -3329.619 0 3 -11772.1338 1 6 17 2 -3329.619 0 3 -415.1114 0 7 12 2 -22722.174 1 5 -13311.8247 1 8 15 3 -16995.346 1 1 -9781.6373 0 1 16 3 26052.495 0 3 -61284.6900 0 2 16 3 -22905.288 1 3 -61284.6900 0 3 13 3 -12776.884 1 4 28080.1607 0 4 13 3 -12776.884 1 5 -18421.6171 0 5
```

If the following commands are issued:

```r
rankedF1 <- h2o.rank_within_group_by(train, c("Group_by_column"), c("Column_to_arrange_by"), c(TRUE), sort_cols_sorted=TRUE)
```
Examples

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

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/airlines/allyears2k_headers.zip"
air <- h2o.importFile(f)
group_cols <- c("Distance")
sort_cols <- c("IsArrDelayed", "IsDepDelayed")
sort_directions <- c(TRUE, FALSE)
h2o.rank_within_group_by(x = air, group_by_cols = group_cols,
  sort_cols = sort_cols,
  ascending = sort_directions,
  new_col_name = "New_Rank",
  sort_cols_sorted = TRUE)

## End(Not run)
```

h2o.rbind

### Combine H2O Datasets by Rows

**Description**

Takes a sequence of H2O data sets and combines them by rows.

**Usage**

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_rbind <- h2o.rbind(prostate, prostate)
head(prostate_rbind)
dim(prostate)
dim(prostate_rbind)

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

Examples

```r
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_glrm <- h2o.glrm(training_frame = iris_hf, k = 4, transform = "STANDARDIZE",
                      loss = "Quadratic", multi_loss = "Categorical", max_iterations = 1000)
iris_rec <- h2o.reconstruct(iris_glrm, iris_hf, reverse_transform = TRUE)
head(iris_rec)
## End(Not run)
```
**h2o.relevel**  
*Reorders levels of an H2O factor, similarly to standard R's relevel.*

**Description**

The levels of a factor are reordered 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
## Not run:
library(h2o)
h2o.init()

# Convert iris dataset to an H2OFrame
iris_hf <- as.h2o(iris)
# Look at current ordering of the Species column levels
h2o.levels(iris_hf["Species"])
# "setosa" "versicolor" "virginica"
# Change the reference level to "virginica"
iris_hf["Species"] <- h2o.relevel(x = iris_hf["Species"], y = "virginica")
# Observe new ordering
h2o.levels(iris_hf["Species"])
# "virginica" "setosa" "versicolor"

## End(Not run)
```

---

**h2o.removeAll**  
*Remove All Objects on the H2O Cluster*

**Description**

Removes the data from the h2o cluster, but does not remove the local references. Retains frames and vectors specified in retained_elements argument. Retained keys must be keys of models and frames only. For models retained, training and validation frames are retained as well. Cross validation models of a retained model are NOT retained automatically, those must be specified explicitly.
Usage

```
> h2o.removeAll(timeout_secs = 0, retained_elements = c())
```

Arguments

- `timeout_secs`: Timeout in seconds. Default is no timeout.
- `retained_elements`: Frames and vectors to be retained. Other keys provided are ignored.

See Also

- `h2o.rm`

Examples

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
h2o.ls()
h2o.removeAll()
h2o.ls()
## End(Not run)
```

---

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

**Description**

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

**Usage**

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

**Arguments**

- `data`: The H2OFrame.
- `cols`: The columns to remove.
**h2o.rep_len**

*Replicate Elements of Vectors or Lists into H2O*

**Description**

h2o.rep_len performs just as `rep` does. It replicates the values in `x` in the H2O backend.

**Usage**

```
h2o.rep_len(x, length.out)
```

**Arguments**

- `x`: an H2O frame
- `length.out`: non negative integer. The desired length of the output vector.

**Value**

Creates an H2OFrame of the same type as `x`

**Examples**

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

iris <- h2o.importFile(f)
h2o.rep_len(iris, length.out = 3)

## End(Not run)
```

---

**h2o.reset_threshold**

*Reset model threshold and return old threshold value.*

**Description**

Reset model threshold and return old threshold value.

**Usage**

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

**Arguments**

- `object`: An `H2OModel` object.
- `threshold`: A threshold value from 0 to 1 included.
examples

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
training_frame = prostate, family = "binomial",
nfolds = 0, alpha = 0.5, lambda_search = FALSE)
old_threshold <- h2o.reset_threshold(prostate_glm, 0.9)

## End(Not run)

### h2o.residual_deviance

Retrieve the residual deviance

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
training_frame = prostate, family = "binomial",
nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.residual_deviance(prostate_glm, train = TRUE)

## End(Not run)
h2o.residual_dof

Retrieve the residual degrees of freedom

Description

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

Usage

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

Arguments

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

Examples

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
training_frame = prostate, family = "binomial",
nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.residual_dof(prostate_glm, train = TRUE)

## End(Not run)

h2o.rm

Delete Objects In H2O

Description

Remove the h2o Big Data object(s) having the key name(s) from ids.

Usage

h2o.rm(ids, cascade = TRUE)
Arguments

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

cascade Boolean, if set to TRUE (default), the object dependencies (e.g. submodels) are also removed.

See Also

h2o.assign, h2o.ls

Examples

```r
## Not run:
library(h2o)
h2o.init()
iris_hex <- as.h2o(iris)
model <- h2o.glm(1:4,5,training = iris_hex, family = "multinomial")
h2o.rm(iris_hex)
## End(Not run)
```

---

h2o.rmse (Retrieves Root Mean Squared Error Value)

Description

Retrieves the root mean squared error value from an H2OModelMetrics object. If "train", "valid", and "xval" parameters are FALSE (default), then the training RMSE value is returned. If more than one parameter is set to TRUE, then a named vector of RMSE values are returned, where the names are "train", "valid" or "xval".

Usage

```r
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

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

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

prostate[,2] <- as.factor(prostate[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
perf <- h2o.performance(model, prostate)
h2o.rmse(perf)
## End(Not run)
```

---

h2o.rmsle

Retrieve the Root Mean Squared Log Error

Description

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

Usage

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

Arguments

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

Examples

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

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

m <- h2o.deeplearning(x = 2:5, y = 1, training_frame = fr)
h2o.rmsle(m)
## End(Not run)
```
**h2o.round**

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

**Description**

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

**Usage**

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

**Arguments**

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

**See Also**

`round` for the base R implementation.

**Examples**

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

f <- "http://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

h2o.round(heart["age"], digits = 3)
## End(Not run)
```

**h2o.rstrip**

Strip set from right

**Description**

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

**Usage**

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

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

Examples

```r
## Not run:
library(h2o)
h2o.init()
string_to_rstrip <- as.h2o("1234567890")
rstrip_string <- h2o.rstrip(string_to_rstrip, "890") #Remove "890"
## End(Not run)
```

h2o.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
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path)
s <- h2o.runif(prostate)
summary(s)
prostate_train <- prostate[s <= 0.8,]
prostate_test <- prostate[s > 0.8,]
nrow(prostate_train) + nrow(prostate_test)
## End(Not run)
```
h2o.saveGrid

Saves an existing Grid of models into a given folder.

Description

Returns a reference to the saved Grid.

Usage

h2o.saveGrid(grid_directory, grid_id)

Arguments

grid_directory  A character string containing the path to the folder for the grid to be saved to.
grid_id        A character string with identification of the grid to be saved.

Value

Returns an object that is a subclass of H2OGrid.

Examples

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

iris.hex <- as.h2o(iris)

ntrees_opts = c(1, 5)
learn_rate_opts = c(0.1, 0.01)
size_of_hyper_space = length(ntrees_opts) * length(learn_rate_opts)

hyper_parameters = list(ntrees = ntree, learn_rate = learn_rate)
# Tempdir is chosen arbitrarily. May be any valid folder on an H2O-supported filesystem.
baseline_grid <- h2o.grid("gbm", grid_id="gbm_grid_test", x=1:4, y=5, training_frame=iris.hex,
hyper_params = hyper_parameters)

grid_path <- h2o.saveGrid(grid_directory = tempdir(), grid_id = baseline_grid@grid_id)
# Remove everything from the cluster or restart it
h2o.removeAll()
grid <- h2o.loadGrid(grid_path)

## End(Not run)
234

h2o.saveModelDetails

h2o.saveModel

Save an H2O Model Object to Disk

Description
Save an H2OModel to disk. (Note that ensemble binary models can be saved.)
Usage
h2o.saveModel(object, path = "", force = FALSE)
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.
The owner of the file saved is the user by which H2O cluster was executed.
See Also
h2o.loadModel for loading a model to H2O from disk
Examples
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.importFile(path = paste("https://raw.github.com",
#
"h2oai/h2o-2/master/smalldata/logreg/prostate.csv", sep = "/"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
#
training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveModel(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)
## End(Not run)

h2o.saveModelDetails

Save an H2O Model Details

Description
Save Model Details of an H2O Model in JSON Format
Usage
h2o.saveModelDetails(object, path = "", force = FALSE)


**Arguments**

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

**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**

```r
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package = "h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
# training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveModelDetails(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)
## End(Not run)
```

---

**Description**

Save an MOJO (Model Object, Optimized) to disk.

**Usage**

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

**Arguments**

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

**Details**

MOJO will download as a zip file. In the case of existing files `force = TRUE` will overwrite the file. Otherwise, the operation will fail.

**See Also**

`h2o.saveModel` for saving a model to disk as a binary object.
Examples
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
# training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.saveMojo(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)

h2o.save.mojo

Save an H2O Model Object as Mojo to Disk

Description

Save an MOJO (Model Object, Optimized) to disk.

Usage

h2o.save_mojo(object, path = "", force = FALSE)

Arguments

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

Details

MOJO will download as a zip file. In the case of existing files force = TRUE will overwrite the file. Otherwise, the operation will fail.

See Also

h2o.saveModel for saving a model to disk as a binary object.

Examples
## Not run:
# library(h2o)
# h2o.init()
# prostate <- h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"))
# prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE","RACE","PSA","DCAPS"),
# training_frame = prostate, family = "binomial", alpha = 0.5)
# h2o.save_mojo(object = prostate_glm, path = "/Users/UserName/Desktop", force = TRUE)

## End(Not run)
**h2o.scale**  

*Scaling and Centering of an H2OFrame*

**Description**

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

**Usage**

\[
h2o.scale(x, center = TRUE, scale = TRUE, inplace = FALSE)
\]

**Arguments**

- `x`: An H2OFrame object.
- `center`: either a logical value or numeric vector of length equal to the number of columns of `x`.
- `scale`: either a logical value or numeric vector of length equal to the number of columns of `x`.
- `inplace`: a logical value indicating whether directly overwrite original data (disabled by default). Exposed for backwards compatibility (prior versions of this functions were always doing an inplace update).

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
summary(iris_hf)
# Scale and center all the numeric columns in iris data set
iris_scaled <- h2o.scale(iris_hf[, 1:4])
## End(Not run)
```

---

**h2o.scoreHistory**  

*Retrieve Model Score History*

**Description**

Retrieve Model Score History

**Usage**

\[
h2o.scoreHistory(object)
\]

**Arguments**

- `object`: An H2OModel object.
Examples

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

cars <- h2o.importFile(f)
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
cars.split <- h2o.splitFrame(data = cars, ratios = 0.8, seed = 1234)
train <- cars.split[[1]]
valid <- cars.split[[2]]
cars_gbm <- h2o.gbm(x = predictors, y = response,
                   training_frame = train,
                   validation_frame = valid,
                   seed = 1234)
h2o.scoreHistory(cars_gbm)
## End(Not run)
```

h2o.sd

### Standard Deviation of a column of data.

**Description**

Obtain the standard deviation of a column of data.

**Usage**

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

```r
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**

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
sd(prostate$AGE)
## End(Not run)
```
h2o.sdev

Retrieve the standard deviations of principal components

Description

Retrieve the standard deviations of principal components

Usage

h2o.sdev(object)

Arguments

object An H2ODimReductionModel object.

Examples

## Not run:
library(h2o)
h2o.init()
cars <- h2o.importFile(f)
predictors <- c("displacement","power","weight","acceleration","year")
cars_pca <- h2o.prcomp(cars, transform = "STANDARDIZE",
    k = 3, x = predictors, seed = 12345)
h2o.sdev(cars_pca)
## End(Not run)

h2o.setLevels

Set Levels of H2O Factor Column

Description

Works on a single categorical vector. New domains must be aligned with the old domains. This call has SIDE EFFECTS and mutates the column in place (change of the levels will also affect all the frames that are referencing this column). If you want to make a copy of the column instead, use parameter in.place = FALSE.

Usage

h2o.setLevels(x, levels, in.place = TRUE)

Arguments

x A single categorical column.
levels A character vector specifying the new levels. The number of new levels must match the number of old levels.
in.place Indicates whether new domain will be directly applied to the column (in place change) or if a copy of the column will be created with the given domain levels.
## h2o.set_s3_credentials

Creates a new Amazon S3 client internally with specified credentials.

### Description

There are no validations done to the credentials. Incorrect credentials are thus revealed with first S3 import call.

### Usage

```r
h2o.set_s3_credentials(secretKeyId, secretAccessKey, sessionToken = NULL)
```

## h2o.setTimezone

Set the Time Zone on the H2O cluster

### Description

Set the Time Zone on the H2O cluster

### Usage

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

### Arguments

- `tz`: The desired timezone.

### Examples

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

h2o.setTimezone("America/Juneau")
h2o.getTimezone()
## End(Not run)
```
h2o.show_progress

Arguments

secretKeyId     Amazon S3 Secret Key ID (provided by Amazon)
secretAccessKey Amazon S3 Secret Access Key (provided by Amazon)
sessionToken   Amazon Session Token (optional, only when using AWS Temporary Credentials)

Description

Enable Progress Bar

Usage

h2o.show_progress()

Examples

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

iris <- h2o.importFile(f)
iris["class"] <- as.factor(iris["class"])
predictors <- c("sepal_len","sepal_wid","petal_len","petal_wid")
splits <- h2o.splitFrame(iris, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]
h2o.show_progress()

iris_km <- h2o.kmeans(x = predictors,
                      training_frame = train,
                      validation_frame = valid,
                      k = 10, estimate_k = TRUE,
                      standardize = FALSE, seed = 1234)

## End(Not run)
### h2o.shutdown

**Shut Down H2O Instance**

**Description**

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

**Usage**

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

**Arguments**

- `prompt`: A logical value indicating whether to prompt the user before shutting down the H2O server.

**Details**

This method checks if H2O is running at the specified IP address and port, and if it is, shuts down that H2O instance.

**WARNING**

All data, models, and other values stored on the server will be lost! Only call this function if you and all other clients connected to the H2O server are finished and have saved your work.

**Note**

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

**See Also**

- `h2o.init`

**Examples**

```r
# Don't run automatically to prevent accidentally shutting down a cluster
## Not run:
library(h2o)
h2o.init()
h2o.shutdown()

## End(Not run)
```
**h2o.signif**  
Round doubles/floats to the given number of significant digits.

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

**Usage**  
```r  
h2o.signif(x, digits = 6)  
signif(x, digits = 6)  
```

**Arguments**  
- `x` An H2OFrame object.  
- `digits` Number of significant digits to round doubles/floats.

**See Also**  
`signif` for the base R implementation.

**Examples**  
```r  
## Not run:  
library(h2o)  
h2o.init()  
f <- "http://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"  
heart <- h2o.importFile(f)  
h2o.signif(heart["age"], digits = 3)  
## End(Not run)  
```

---

**h2o.sin**  
Compute the sine of x

**Description**  
Compute the sine of x

**Usage**  
```r  
h2o.sin(x)  
```

**Arguments**  
- `x` An H2OFrame object.
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.
...
na.rm A logical value indicating whether NA or missing values should be stripped before the computation.

Value

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

Examples

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

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

## End(Not run)
**h2o.splitFrame**

*Split an H2O Data Set*

**Description**

Split an existing H2O data set according to user-specified ratios. The number of subsets is always 1 more than the number of given ratios. Note that this does not give an exact split. H2O is designed to be efficient on big data using a probabilistic splitting method rather than an exact split. For example, when specifying a split of 0.75/0.25, H2O will produce a test/train split with an expected value of 0.75/0.25 rather than exactly 0.75/0.25. On small datasets, the sizes of the resulting splits will deviate from the expected value more than on big data, where they will be very close to exact.

**Usage**

```r
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**

```r
## Not run:
library(h2o)
h2o.init()
iris_hf <- as.h2o(iris)
iris_split <- h2o.splitFrame(iris_hf, ratios = c(0.2, 0.5))
head(iris_split[[1]])
summary(iris_split[[1]])
## End(Not run)
```

---

**h2o.sqrt**

*Compute the square root of x*

**Description**

Compute the square root of x
Usage

h2o.sqrt(x)

Arguments

x

An H2OFrame object.

See Also

sqrt for the base R implementation.

Examples

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

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

## End(Not run)

h2o.stackedEnsemble

Builds a Stacked Ensemble

Description

Build a stacked ensemble (aka. Super Learner) using the H2O base learning algorithms specified by the user.

Usage

h2o.stackedEnsemble(
  x,
  y,
  training_frame,
  model_id = NULL,
  validation_frame = NULL,
  blending_frame = NULL,
  base_models = list(),
  metalearner_algorithm = c("AUTO", "deelearning", "drf", "gbm", "glm", "naivebayes", "xgboost"),
  metalearner_nfolds = 0,
  metalearner_fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
  metalearner_fold_column = NULL,
  metalearner_params = NULL,
  seed = -1,
  keep_levelone_frame = FALSE,
  export_checkpoints_dir = 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.
blending_frame Frame used to compute the predictions that serve as the training frame for the metalearnet (triggers blending mode if provided)
base_models List of models or grids (or their ids) to ensemble/stack together. Grids are expanded to individual models. If not using blending frame, then models must have been cross-validated using nfolds > 1, and folds must be identical across models.
metalearnet_algorithm Type of algorithm to use as the metalearnet. Options include 'AUTO' (GLM with non negative weights; if validation_frame is present, a lambda search is performed), 'deeplearning' (Deep Learning with default parameters), 'drf' (Random Forest with default parameters), 'gbm' (GBM with default parameters), 'glm' (GLM with default parameters), 'naivebayes' (NaiveBayes with default parameters), or 'xgboost' (if available, XGBoost with default parameters). Must be one of: "AUTO", "deeplearning", "drf", "gbm", "glm", "naivebayes", "xgboost". Defaults to AUTO.
metalearnet_nfolds Number of folds for K-fold cross-validation of the metalearnet algorithm (0 to disable or >= 2). Defaults to 0.
metalearnet_fold_assignment Cross-validation fold assignment scheme for metalearnet 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".
metalearnet_fold_column Column with cross-validation fold index assignment per observation for cross-validation of the metalearnet.
metalearnet_params Parameters for metalearnet algorithm
seed Seed for random numbers; passed through to the metalearnet algorithm. Defaults to -1 (time-based random number).
keep_levelone_frame Logical. Keep level one frame used for metalearnet training. Defaults to FALSE.
export_checkpoints_dir Automatically export generated models to this directory.
### Examples

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

# Import a sample binary outcome train/test set
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.csv")
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")

# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[,y] <- as.factor(train[,y])
test[,y] <- as.factor(test[,y])

# Number of CV folds
nfolds <- 5

# Train & Cross-validate a GBM
my_gbm <- h2o.gbm(x = x,
y = y,
training_frame = train,
distribution = "bernoulli",
ntrees = 10,
max_depth = 3,
min_rows = 2,
learn_rate = 0.2,
nfolds = nfolds,
fold_assignment = "Modulo",
keep_cross_validation_predictions = TRUE,
seed = 1)

# Train & Cross-validate a RF
my_rf <- h2o.randomForest(x = x,
y = y,
training_frame = train,
ntrees = 50,
nfolds = nfolds,
fold_assignment = "Modulo",
keep_cross_validation_predictions = TRUE,
seed = 1)

# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
y = y,
training_frame = train,
model_id = "my_ensemble_binomial",
base_models = list(my_gbm, my_rf))

## End(Not run)
```

Start Writing H2O R Logs
**h2o.startLogging**

**Description**

Begin logging H2o R POST commands and error responses to local disk. Used primarily for debugging purposes.

**Usage**

```r
h2o.startLogging(file)
```

**Arguments**

- `file`: a character string name for the file, automatically generated

**See Also**

- `h2o.stopLogging`, `h2o.clearLog`, `h2o.openLog`

**Examples**

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

---

**h2o.std_coef_plot**

**Plot Standardized Coefficient Magnitudes**

**Description**

Plot a GLM model’s standardized coefficient magnitudes.

**Usage**

```r
h2o.std_coef_plot(model, num_of_features = NULL)
```

**Arguments**

- `model`: A trained generalized linear model
- `num_of_features`: The number of features to be shown in the plot

**See Also**

- `h2o.varimp_plot` for variable importances plot of random forest, GBM, deep learning.
Examples

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

prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
prostate_glm <- h2o.glm(y = "CAPSULE", x = c("AGE", "RACE", "PSA", "DCAPS"),
                        training_frame = prostate, family = "binomial",
                        nfolds = 0, alpha = 0.5, lambda_search = FALSE)
h2o.std_coef_plot(prostate_glm)

## End(Not run)
```

### Description

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

### Usage

```r
h2o.stopLogging()
```

### See Also

`h2o.startLogging`, `h2o.clearLog`, `h2o.openLog`

### Examples

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

## End(Not run)
```
h2o.str  
Display the structure of an H2OFrame object

Description
Display the structure of an H2OFrame object

Usage
h2o.str(object, ..., cols = FALSE)

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

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

frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.str(frame, cols = FALSE)
## End(Not run)

h2o.stringdist  
Compute element-wise string distances between two H2OFrames

Description
Compute element-wise string distances between two H2OFrames. Both frames need to have the same shape (N x M) and only contain string/factor columns. Return a matrix (H2OFrame) of shape N x M.

Usage
h2o.stringdist(
  x,
  y,
  method = c("lv", "lcs", "qgram", "jaccard", "jw", "soundex"),
  compare_empty = TRUE
)
**h2o.strsplit**

**String Split**

**Description**

String Split

**Usage**

h2o.strsplit(x, split)

**Arguments**

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

**Value**

An H2OFrame where each column is the outcome of the string split.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
string_to_split <- as.h2o("Split at every character.")
split_string <- h2o.strsplit(string_to_split,"")
## End(Not run)
```
h2o.sub

String Substitute

Description

Creates a copy of the target column in which each string has the first 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
## Not run:
library(h2o)
h2o.init()
string_to_sub <- as.h2o("r tutorial")
sub_string <- h2o.sub("r ", "H2O ", string_to_sub)
## End(Not run)
```

h2o.substring

Substr

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

```r
## Not run:
library(h2o)
h2o.init()
string_to_substring <- as.h2o("1234567890")
substr <- h2o.substring(string_to_substring, 2) # Get substring from second index onwards

## End(Not run)
```

---

**h2o.sum**

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

Description

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

Usage

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

Arguments

- `x` An H2OFrame object.
- `na.rm` logical. indicating whether missing values should be removed.
- `axis` An int that indicates whether to do down a column (0) or across a row (1). For row or column sums, the `return_frame` parameter must be TRUE.
- `return_frame` A boolean that indicates whether to return an H2O frame or one single aggregated value. Default is FALSE.

See Also

- `sum` for the base R implementation.

Examples

```r
## Not run:
library(h2o)
h2o.init()
frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.sum(frame["C1"], na.rm = TRUE, axis = 0, return_frame = TRUE)

## End(Not run)
```
h2o.summary

Summarizes the columns of an H2OFrame.

Description

A method for the summary generic. Summarizes the columns of an H2O data frame or subset of columns and rows using vector notation (e.g. dataset[row, col]).

Usage

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

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

Arguments

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

Details

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

Value

A table displaying the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column, and the levels and category counts of the levels in each categorical column.

Examples

```r
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(path = prostate_path)
summary(prostate)
summary(prostate$GLEASON)
summary(prostate[,4:6])
summary(prostate, exact_quantiles=TRUE)
## End(Not run)
```
h2o.svd  Singular value decomposition of an H2O data frame using the power method

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

Usage

h2o.svd(
  training_frame,
  x,
  destination_key,
  model_id = NULL,
  validation_frame = NULL,
  ignore_const_cols = TRUE,
  score_each_iteration = FALSE,
  transform = c("NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE"),
  svd_method = c("GramSVD", "Power", "Randomized"),
  nv = 1,
  max_iterations = 1000,
  seed = -1,
  keep_u = TRUE,
  u_name = NULL,
  use_all_factor_levels = TRUE,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)

Arguments

training_frame  Id of the training data frame.
x  A vector containing the character names of the predictors in the model.
destination_key  (Optional) The unique key assigned to the resulting model. Automatically generated if none is provided.
model_id  Destination id for this model; auto-generated if not specified.
validation_frame  Id of the validation data frame.
ignore_const_cols  Logical. Ignore constant columns. Defaults to TRUE.
score_each_iteration  Logical. Whether to score during each iteration of model training. Defaults to FALSE.
transform  Transformation of training data Must be one of: "NONE", "STANDARDIZE", "NORMALIZE", "DEMEAN", "DESCALE". Defaults to NONE.
svd_method  Method for computing SVD (Caution: Randomized is currently experimental and unstable) Must be one of: "GramSVD", "Power", "Randomized". Defaults to GramSVD.
h2o.table

**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)

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

---

**Value**

an object of class `H2ODimReductionModel`.

**References**

h2o.tabulate

Tabulation between Two Columns of an H2OFrame

Description

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

Usage

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

Arguments

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

Arguments

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

Value

Returns a tabulated H2OFrame object.

Examples

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

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

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

## End(Not run)
Value

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

Examples

## Not run:
library(h2o)
h2o.init()
df <- as.h2o(iris)
tab <- h2o.tabulate(data = df, x = "Sepal.Length", y = "Petal.Width",
weights_column = NULL, nbins_x = 10, nbins_y = 10)
plot(tab)
## End(Not run)

h2o.tan

Compute the tangent of x

Description

Compute the tangent of x

Usage

h2o.tan(x)

Arguments

x

An H2OFrame object.

See Also

tan for the base R implementation.

Examples

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

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

## End(Not run)
**h2o.tanh**

*Compute the hyperbolic tangent of x*

**Description**

Compute the hyperbolic tangent of x

**Usage**

```r
h2o.tanh(x)
```

**Arguments**

- `x` An H2OFrame object.

**See Also**

`tanh` for the base R implementation.

**Examples**

```r
## Not run:
library(h2o)
h2o.init()
frame <- h2o.createFrame(rows = 6, cols = 2,
categorical_fraction = 0.0,
missing_fraction = 0.7,
seed = 123)
h2o.tanh(frame)
## End(Not run)
```

---

**h2o.targetencoder**

*Transformation of a categorical variable with a mean value of the target variable*

**Description**

Transformation of a categorical variable with a mean value of the target variable

**Usage**

```r
h2o.targetencoder(
  x, 
  y, 
  training_frame, 
  model_id = NULL, 
  fold_column = NULL, 
  blending = FALSE, 
  k = 10, 
```
f = 20,
        data_leakage_handling = c("None", "KFold", "LeaveOneOut"),
noise_level = 0.01,
seed = -1
)

Arguments

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

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

training_frame Id of the training data frame.

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

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

blending Logical. Blending enabled/disabled Defaults to FALSE.

k Inflection point. Used for blending (if enabled). Blending is to be enabled sepa-
    rately using the 'blending' parameter. Defaults to 10.

f Smoothing. Used for blending (if enabled). Blending is to be enabled separately
    using the 'blending' parameter. Defaults to 20.

data_leakage_handling Data leakage handling strategy. Must be one of: "None", "KFold", "LeaveOne-
Out". Defaults to None.

noise_level Noise level Defaults to 0.01.

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

Examples

## Not run:
library(h2o)
h2o.init()
#Import the titanic dataset
f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/titanic.csv"
titanic <- h2o.importFile(f)

# Set response as a factor
response <- "survived"
titanic[[response]] <- as.factor(titanic[[response]])

# Split the dataset into train and test
splits <- h2o.splitFrame(data = titanic, ratios = .8, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]

# Choose which columns to encode
encode_columns <- c("home.dest", "cabin", "embarked")

# Train a TE model
```r
te_model <- h2o.targetencoder(x = encode_columns,
                               y = response,
                               training_frame = train,
                               fold_column = "pclass",
                               data_leakage_handling = "KFold")

# New target encoded train and test sets
train_te <- h2o.transform(te_model, train)
test_te <- h2o.transform(te_model, test)
```

### h2o.target_encode_apply

**Apply Target Encoding Map to Frame**

**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".
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.

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

data <- h2o.importFile(path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv")
splits <- h2o.splitFrame(data, seed = 1234)
train <- splits[[1]]
test <- splits[[2]]
mapping <- h2o.target_encode_create(data = train, x = list(c("job"), c("job", "marital")),
                                    y = "age")

# Apply mapping to the training dataset
train_encode <- h2o.target_encode_apply(data = train, x = list(c("job"), c("job", "marital")),
                                         y = "age", mapping, holdout_type = "LeaveOneOut")

# Apply mapping to a test dataset
test_encode <- h2o.target_encode_apply(data = test, x = list(c("job"), c("job", "marital")),
                                        y = "age", target_encode_map = mapping,
                                        holdout_type = "None")
```

## End(Not run)

Usage

h2o.target_encode_create(data, x, y, fold_column = NULL)

Arguments

data
An H2OFrame object with which to create the target encoding map.

x
A list containing the names or indices of the variables to encode. A target encoding map will be created for each element in the list. Items in the list can be multiple columns. For example, if `x = list(c("A"), c("B", "C"))`, then there will be one mapping frame for A and one mapping frame for B & C (in this case, we group by two columns).

y
The name or column index of the response variable in the data. The response variable can be either numeric or binary.

fold_column
(Optional) The name or column index of the fold column in the data. Defaults to NULL (no `fold_column`).

Value

Returns a list of H2OFrame objects containing the target encoding mapping for each column in `x`.

See Also

h2o.target_encode_apply for applying the target encoding mapping to a frame.

Examples

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

# Get Target Encoding Map on bank-additional-full data with numeric response
data <- h2o.importFile(path = "https://s3.amazonaws.com/h2o-public-test-data/smalldata/demos/bank-additional-full.csv")
mapping_age <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")), y = "age")
head(mapping_age)

# Get Target Encoding Map on bank-additional-full data with binary response
mapping_y <- h2o.target_encode_create(data = data, x = list(c("job"), c("job", "marital")), y = "y")
head(mapping_y)

## End(Not run)
```

h2o.target_encode_fit

Deprecated API. Please use h2o.targetencoder model instead.

Description

Create Target Encoding Map
Usage

```r
h2o.target_encode_fit(frame, x, y, fold_column = NULL)
```

Arguments

- `frame`: An H2OFrame object with which to create the target encoding map.
- `x`: List of categorical column names or indices that we want apply target encoding to. Case when item in the list is a list of multiple columns itself is not supported for now.
- `y`: The name or column index of the response variable in the frame.
- `fold_column`: (Optional) The name or column index of the fold column in the frame.

Details

This is an API for a new target encoding implemented in JAVA. Creates a target encoding map based on group-by columns (`x`) and binary target column (`y`). Computing target encoding for high cardinality categorical columns can improve performance of supervised learning models.

Value

Returns an object containing the target encoding mapping for each column in `x`.

See Also

`h2o.target_encode_transform` for applying the target encoding mapping to a frame.

---

**h2o.target_encode_transform**

*Deprecated API. Please use h2o.targetencoder model instead. Transform Frame by Target Encoding Map*

Description

This is an API for a new target encoding implemented in JAVA. Applies a target encoding map to an H2OFrame object. Computing target encoding for high cardinality categorical columns can improve performance of supervised learning models.

Usage

```r
h2o.target_encode_transform(
  frame,
  x,
  y,
  target_encode_map,
  holdout_type,
  fold_column = NULL,
  blended_avg = TRUE,
  inflection_point = 10,
  smoothing = 20,
)```
noise = -1,
seed = -1
)

Arguments

frame  An H2OFrame object with which to apply the target encoding map.
x List of categorical column names or indices that we want apply target encoding to. Case when item in the list is a list of multiple columns itself is not supported for now.
y The name or column index of the response variable in the frame.
target_encode_map An object that is a result of the calling h2o.target_encode_fit function.
holdout_type Supported options:
1) "kfold" - encodings for a fold are generated based on out-of-fold data.
2) "loo" - leave one out. Current row’s response value is subtracted from the pre-calculated per-level frequencies.
3) "none" - we do not holdout anything. Using whole frame for training
fold_column (Optional) The name or column index of the fold column in the frame.
blended_avg Logical. (Optional) Whether to perform blended average. Defaults to TRUE
inflection_point (Optional) Parameter for blending. Used to calculate ‘lambda’. Determines half of the minimal sample size for which we completely trust the estimate based on the sample in the particular level of categorical variable. Default value is 10.
smoothing (Optional) Parameter for blending. Used to calculate ‘lambda’. Controls the rate of transition between the particular level’s posterior probability and the prior probability. For smoothing values approaching infinity it becomes a hard threshold between the posterior and the prior probability. Default value is 20.
noise (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_fit for creating the target encoding map
**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**

```r
h2o.toFrame(word2vec)
```

**Arguments**

- `word2vec` A word2vec model.

**Examples**

```r
## Not run:
h2o.init()
# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "c")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)
# Transform words to vectors and return average vector for each sentence
h2o.toFrame(w2v_model) # -> Frame made of 2 rows and 2 columns
## End(Not run)
```

**h2o.tokenize**

*Tokenize String*

**Description**

h2o.tokenize is similar to h2o.strsplit, the difference between them is that h2o.tokenize will store the tokenized text into a single column making it easier for additional processing (filtering stop words, word2vec algo, ...).

**Usage**

```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
## Not run:
library(h2o)
h2o.init()
string_to_tokenize <- as.h2o("Split at every character and tokenize.")
tokenize_string <- h2o.tokenize(as.character(string_to_tokenize), "}")
## End(Not run)
```

### h2o.tolower

Convert strings to lowercase

#### Description

Convert strings to lowercase

#### Usage

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

#### Arguments

- `x`: An H2OFrame object whose strings should be lower cased

#### Value

An H2OFrame with all entries in lowercase format

### 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)
```
**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

### Examples

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

dataset <- h2o.importFile(f)
frameNames <- names(dataset)
nPercent <- c(1, 2, 3, 4)
nP <- nPercent[sample(1:length(nPercent), 1, replace = FALSE)]
colIndex <- sample(1:length(frameNames), 1, replace = FALSE)
h2o.topN(dataset, frameNames[colIndex], nP)

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

predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.totss(km, train = TRUE)

## End(Not run)

h2o.tot_withinss

Get the total within cluster sum of squares.

### Description

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

### Usage

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

### Arguments

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

### Examples

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

predictors <- c("sepal_len", "sepal_wid", "petal_len", "petal_wid")
km <- h2o.kmeans(x = predictors, training_frame = fr, k = 3, nfolds = 3)
h2o.tot_withinss(km, train = TRUE)

## End(Not run)
h2o.toupper

Convert strings to uppercase

Description
Convert strings to uppercase

Usage
h2o.toupper(x)

Arguments
x
An H2OFrame object whose strings should be upper cased

Value
An H2OFrame with all entries in uppercase format

Examples
## Not run:
library(h2o)
h2o.init()
string_to_upper <- as.h2o("abcde")
upper_string <- h2o.toupper(string_to_upper)
## End(Not run)

h2o.train_segments
H2O Segmented-Data Bulk Model Training

Description
Provides a set of functions to train a group of models on different segments (subpopulations) of the training set.

Usage
h2o.train_segments(
  algorithm,
  segment_columns,
  segment_models_id,
  parallelism = 1,
  ...)

Arguments

algorithm        Name of algorithm to use in training segment models (gbm, randomForest, kmeans, glm, deeplearning, naivebayes, psvm, xgboost, pca, svd, targetencoder, aggregator, word2vec, coxph, isolationforest, kmeans, stackedensemble, glm, gam).

segment_columns  A list of columns to segment-by. H2O will group the training (and validation) dataset by the segment-by columns and train a separate model for each segment (group of rows).

segment_models_id Identifier for the returned collection of Segment Models. If not specified it will be automatically generated.

parallelism      Level of parallelism of bulk model building, it is the maximum number of models each H2O node will be building in parallel, defaults to 1.

...               Use to pass along training_frame parameter, x, y, and all non-default parameter values to the algorithm Look at the specific algorithm - h2o.gbm, h2o.glm, h2o.kmeans, h2o.deepLearning - for available parameters.

Details

Start Segmented-Data bulk Model Training for a given algorithm and parameters.

Examples

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

as.data.frame(models)

## End(Not run)
```

h2o.transform

Use H2O Transformation model and apply the underlying transformation

Description

Use H2O Transformation model and apply the underlying transformation

Usage

```r
h2o.transform(model, ...)
```
Arguments

- **model**: A trained model representing the transformation strategy
- **...**: Transformation model-specific parameters

Value

Returns an H2OFrame object with data transformed.

Description

Applies target encoding to a given dataset

Usage

```r
## S4 method for signature 'H2OTargetEncoderModel'
h2o.transform(
  model,
  data,
  data_leakage_handling = NULL,
  use_blending = NULL,
  inflection_point = -1,
  smoothing = -1,
  noise = -1,
  seed = -1
)
```

Arguments

- **model**: A trained model representing the transformation strategy
- **data**: An H2OFrame with data to be transformed
- **data_leakage_handling**: Handling of data leakage. Available options are: ["None", "LeaveOneOut", "KFold"]. Defaults to "None".
- **use_blending**: Use blending during the transformation. Respects model settings when not set.
- **inflection_point**: Blending parameter. Only effective when blending is enabled. By default, model settings are respected, if not overridden by this setting.
- **smoothing**: Blending parameter. Only effective when blending is enabled. By default, model settings are respected, if not overridden by this setting.
- **noise**: An amount of random noise added to the encoding. This helps prevent overfitting. Defaults to 0.01 * range of response.
- **seed**: A random seed used to generate draws from the uniform distribution for random noise. Defaults to -1.

Value

Returns an H2OFrame object with data transformed.
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
```r
## S4 method for signature 'H2OWordEmbeddingModel'

h2o.transform(model, words, aggregate_method = c("NONE", "AVERAGE"))
```

Arguments
- `model`: A word2vec model.
- `words`: An H2OFrame made of a single column containing source words.
- `aggregate_method`: Specifies how to aggregate sequences of words. If method is 'NONE' then no aggregation is performed and each input word is mapped to a single word-vector. If method is 'AVERAGE' then input is treated as sequences of words delimited by NA. Each word of a sequences is internally mapped to a vector and vectors belonging to the same sentence are averaged and returned in the result.

Examples
```r
## Not run:

h2o.init()

# Build a simple word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors without aggregation
sentences <- as.character(as.h2o(c("b", "c", "a", NA, "b")))
h2o.transform(w2v_model, sentences) # -> 5 rows total, 2 rows NA ("c" is not in the vocabulary)

# Transform words to vectors and return average vector for each sentence
h2o.transform(w2v_model, sentences, aggregate_method = "AVERAGE") # -> 2 rows

## End(Not run)
```
Usage

```r
h2o.transform_word2vec(
  word2vec, 
  words, 
  aggregate_method = c("NONE", "AVERAGE")
)
```

Arguments

- **word2vec**: A word2vec model.
- **words**: An H2OFrame made of a single column containing source words.
- **aggregate_method**: Specifies how to aggregate sequences of words. If method is `"NONE"` then no aggregation is performed and each input word is mapped to a single word-vector. If method is `"AVERAGE"` then input is treated as sequences of words delimited by NA. Each word of a sequences is internally mapped to a vector and vectors belonging to the same sentence are averaged and returned in the result.

Examples

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

# Build a dummy word2vec model
data <- as.character(as.h2o(c("a", "b", "a")))
w2v_model <- h2o.word2vec(data, sent_sample_rate = 0, min_word_freq = 0, epochs = 1, vec_size = 2)

# Transform words to vectors without aggregation
sentences <- as.character(as.h2o(c("b", "c", "a", NA, "b")))
h2o.transform(w2v_model, sentences) # -> 5 rows total, 2 rows NA ("c" is not in the vocabulary)

# Transform words to vectors and return average vector for each sentence
h2o.transform(w2v_model, sentences, aggregate_method = "AVERAGE") # -> 2 rows

## End(Not run)
```

---

**h2o.trim**

Trim Space

**Description**

Trim Space

**Usage**

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

**Arguments**

- **x**: The column whose strings should be trimmed.
Examples

```r
## Not run:
library(h2o)
h2o.init()
string_to_trim <- as.h2o("r tutorial")
trim_string <- h2o.trim(string_to_trim)

## End(Not run)
```

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.

Examples

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

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

h2o.trunc(frame["C1"])

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

**H2O Unique**

**Description**

Extract unique values in the column.

**Usage**

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

**Arguments**

- `x`: An H2OFrame object.

**Value**

Returns an H2OFrame object.

**Examples**

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

iris <- h2o.importFile(f)
h2o.unique(iris["class"])
## End(Not run)
```

---

**h2o.upload_model**

Upload a binary model from the provided local path to the H2O cluster. (H2O model can be saved in a binary form either by `saveModel()` or by `download_model()` function.)

**Description**

Upload a binary model from the provided local path to the H2O cluster. (H2O model can be saved in a binary form either by `saveModel()` or by `download_model()` function.)

**Usage**

```r
h2o.upload_model(path)
```

**Arguments**

- `path`: A path on the machine this python session is currently connected to, specifying the location of the model to upload.
Value

Returns a new H2OModel object.

See Also

h2o.saveModel, h2o.download_model

h2o.upload_mojo

Imports a MOJO from a local filesystem, creating a Generic model with it.

Description

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

Usage

h2o.upload_mojo(mojo_local_file_path)

Arguments

mojo_local_file_path

Filesystem path to the model imported

Value

Returns H2O Generic Model embedding given MOJO model

Examples

## Not run:

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

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

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

# Upload the MOJO from local filesystem and obtain a Generic model
mojo_model <- h2o.upload_mojo(mojo_original_path)

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

## End(Not run)
h2o.var

Variance of a column or covariance of columns.

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

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

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

Arguments

x
An H2OFrame object.

y
NULL (default) or an H2OFrame. The default is equivalent to y = x.

na.rm
logical. Should missing values be removed?

use
An optional character string indicating how to handle missing values. This must
be one of the following: "everything" - outputs NaNs whenever one of its con-
thributing 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
## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
var(prostate$AGE)

## End(Not run)

h2o.varimp

Retrieve the variable importance.

Description
Retrieve the variable importance.

Usage
h2o.varimp(object)
Arguments

object An H2OModel object.

Examples

## Not run:
library(h2o)
h2o.init()
pros <- h2o.importFile(f)
response <- "GLEASON"
predictors <- c("ID","AGE","CAPSULE","DCAPS","PSA","VOL","DPROS")
model <- h2o.glm(x = predictors, y = response, training_frame = pros)
h2o.varimp(model)
## End(Not run)

h2o.varimp_plot

Plot Variable Importances

Description

Plot Variable Importances

Usage

h2o.varimp_plot(model, num_of_features = NULL)

Arguments

model A trained model (accepts a trained random forest, GBM, or deep learning model, will use h2o.std_coef_plot for a trained GLM
num_of_features The number of features shown in the plot (default is 10 or all if less than 10).

See Also

h2o.std_coef_plot for GLM.

Examples

## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.importFile(prostate_path)
prostate[,2] <- as.factor(prostate[,2])
model <- h2o.gbm(x = 3:9, y = 2, training_frame = prostate, distribution = "bernoulli")
h2o.varimp_plot(model)

# for deep learning set the variable_importance parameter to TRUE
iris_hf <- as.h2o(iris)
iris_dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris_hf, variable_importances = TRUE)
h2o.varimp_plot(iris_dl)

### End(Not run)

---

**h2o.varsplits**

Retrieve per-variable split information for a given Isolation Forest model. Output will include:

- **count** - The number of times a variable was used to make a split.
- **aggregated_split_ratios** - The split ratio is defined as \( \frac{\text{abs}(\#\text{left\_observations} - \#\text{right\_observations})}{\#\text{before\_split}} \). Even splits (\#left\_observations approx the same as \#right\_observations) contribute less to the total aggregated split ratio value for the given feature; highly imbalanced splits (eg. \#left\_observations \( \gg \) \#right\_observations) contribute more.
- **aggregated_split_depths** - The sum of all depths of a variable used to make a split. (If a variable is used on level \( N \) of a tree, then it contributes with \( N \) to the total aggregate.)

**Description**

Retrieve per-variable split information for a given Isolation Forest model. Output will include:

- **count** - The number of times a variable was used to make a split.
- **aggregated_split_ratios** - The split ratio is defined as \( \frac{\text{abs}(\#\text{left\_observations} - \#\text{right\_observations})}{\#\text{before\_split}} \). Even splits (\#left\_observations approx the same as \#right\_observations) contribute less to the total aggregated split ratio value for the given feature; highly imbalanced splits (eg. \#left\_observations \( \gg \) \#right\_observations) contribute more.
- **aggregated_split_depths** - The sum of all depths of a variable used to make a split. (If a variable is used on level \( N \) of a tree, then it contributes with \( N \) to the total aggregate.)

**Usage**

h2o.varsplits(object)

**Arguments**

- **object** An Isolation Forest model represented by **H2OModel** object.

---

**h2o.week**

Convert Milliseconds to Week of Week Year in H2O Datasets

**Description**

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

h2o.week(x)

week(x)

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

Arguments

x       An H2OFrame object.

Value

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

See Also

h2o.month

Examples

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

hdf <- h2o.importFile(f)
h2o.week(hdf["ds9"])

## End(Not run)

h2o.weights  Retrieve the respective weight matrix

Description

Retrieve the respective weight matrix

Usage

h2o.weights(object, matrix_id = 1)

Arguments

object  An H2OModel or H2OModelMetrics
matrix_id An integer, ranging from 1 to number of layers + 1, that specifies the weight matrix to return.
### h2o.which

#### Description

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

#### Usage

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

#### Arguments

- `x`  
  An H2OFrame object.

#### Value

Returns an H2OFrame object.

#### See Also

- `which` for the base R method.

#### Examples

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

census <- h2o.importFile(f)  
census[,1] <- as.factor(census[,1])  
dlmodel <- h2o.deeplearning(x = c(1:3), y = 4, training_frame = census,  
                           hidden = c(17,191),  
                           epochs = 1,  
                           balance_classes = FALSE,  
                           export_weights_and_biases = TRUE)  

h2o.weights(dlmodel, matrix_id = 1)

## End(Not run)
```

---

```r
h2o.which  
Which indices are TRUE?
```

---

#### Description

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

#### Usage

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

#### Arguments

- `x`  
  An H2OFrame object.

#### Value

Returns an H2OFrame object.

#### See Also

- `which` for the base R method.

#### Examples

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

iris_hf <- as.h2o(iris)  
h2o.which(iris_hf[, 1] == 4.4)

## End(Not run)
```
### h2o.which_max

**Which indice contains the max value?**

**Description**

Get the index of the max value in a column or row.

**Usage**

```r
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.

**Examples**

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

census <- h2o.importFile(f)
census[,1] <- as.factor(census[,1])
dlmodel <- h2o.deeplearning(x = c(1:3), y = 4, hidden = c(17,191),
                           epochs = 1, training_frame = census,
                           balance_classes = FALSE,
                           export_weights_and_biases = TRUE)
h2o.which_max(census["PER CAPITA INCOME "], na.rm = FALSE, axis = 0)
## End(Not run)
```
h2o.which_min

Which index contains the min value?

Description
Get the index of the min value in a column or row

Usage
h2o.which_min(x, na.rm = TRUE, axis = 0)

Arguments
- **x**: An H2OFrame object.
- **na.rm**: logical. Indicate whether missing values should be removed.
- **axis**: integer. Indicate whether to calculate the mean down a column (0) or across a row (1).

Value
Returns an H2OFrame object.

See Also
- `which.min` for the base R method.

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

census <- h2o.importFile(f)
dlmodel <- h2o.deeplearning(x = c(1:3), y = 4, hidden = c(17,191),
epochs = 1, training_frame = census,
balance_classes = FALSE,
export_weights_and_biases = TRUE)
h2o.which_min(census["PER CAPITA INCOME "], na.rm = FALSE, axis = 0)
## End(Not run)
```
**h2o.withinss**  
*Get the Within SS*

**Description**

Get the Within SS

**Usage**

```r
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**

```r
h2o.word2vec(
  training_frame = NULL,
  model_id = NULL,
  min_word_freq = 5,
  word_model = c("SkipGram", "CBOW"),
  norm_model = c("HSM"),
  vec_size = 100,
  window_size = 5,
  sent_sample_rate = 0.001,
  init_learning_rate = 0.025,
  epochs = 5,
  pre_trained = NULL,
  max_runtime_secs = 0,
  export_checkpoints_dir = NULL
)
```

**Arguments**

- **training_frame**: Id of the training data frame.
- **model_id**: Destination id for this model; auto-generated if not specified.
- **min_word_freq**: This will discard words that appear less than <int> times. Defaults to 5.
- **word_model**: The word model to use (SkipGram or CBOW) Must be one of: "SkipGram", "CBOW". Defaults to SkipGram.
- **norm_model**: Use Hierarchical Softmax Must be one of: "HSM". Defaults to HSM.
vec_size  Set size of word vectors Defaults to 100.
window_size  Set max skip length between words Defaults to 5.
sent_sample_rate  Set threshold for occurrence of words. Those that appear with higher frequency in the training data will be randomly down-sampled; useful range is (0, 1e-5) Defaults to 0.001.
init_learning_rate  Set the starting learning rate Defaults to 0.025.
epochs  Number of training iterations to run Defaults to 5.
pre_trained  Id of a data frame that contains a pre-trained (external) word2vec model
max_runtime_secs  Maximum allowed runtime in seconds for model training. Use 0 to disable. Defaults to 0.
export_checkpoints_dir  Automatically export generated models to this directory.

Examples

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

# Import the CraigslistJobTitles dataset
f <- "https://raw.githubusercontent.com/h2oai/sparkling-water/rel-1.6/examples/smalldata/"
jobtitles <- h2o.importFile(paste0(f, "craigslistJobTitles.csv"),
    col.names = c("category", "jobtitle"),
    col.types = c("String", "String"), header = TRUE)

# Build and train the Word2Vec model
words <- h2o.tokenize(jobtitles, " ")
vec <- h2o.word2vec(training_frame = words)
h2o.findSynonyms(vec, "teacher", count = 20)

## End(Not run)
```

---

**h2o.xgboost**  
*Build an eXtreme Gradient Boosting model*

**Description**

Builds a eXtreme Gradient Boosting model using the native XGBoost backend.

**Usage**

```r
h2o.xgboost(
  x,  
  y,  
  training_frame,  
  model_id = NULL,  
  validation_frame = NULL,  
  nfolds = 0,
```
keep_cross_validation_models = TRUE,
keep_cross_validation_predictions = FALSE,
keep_cross_validation_fold_assignment = FALSE,
score_each_iteration = FALSE,
fold_assignment = c("AUTO", "Random", "Modulo", "Stratified"),
fold_column = NULL,
ignore_const_cols = TRUE,
offset_column = NULL,
weights_column = NULL,
stopping_rounds = 0,
stopping_metric = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing"),
stopping_tolerance = 0.001,
max_runtime_secs = 0,
seed = -1,
distribution = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"),
tweedie_power = 1.5,
quiet_mode = TRUE,
checkpoint = NULL,
export_checkpoints_dir = NULL,
ntrees = 50,
max_depth = 6,
min_rows = 1,
min_child_weight = 1,
learn_rate = 0.3,
eta = 0.3,
sample_rate = 1,
subsample = 1,
col_sample_rate = 1,
colsample_bylevel = 1,
col_sample_rate_per_tree = 1,
colsample_bytree = 1,
max_abs_leafnode_pred = 0,
max_delta_step = 0,
monotone_constraints = NULL,
score_tree_interval = 0,
min_split_improvement = 0,
gamma = 0,
nthread = -1,
save_matrix_directory = NULL,
built_tree_one_node = FALSE,
calibrate_model = FALSE,
calibration_frame = NULL,
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 = 1,
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_models Logical. Whether to keep the cross-validation models. Defaults to TRUE.
keep_cross_validation_predictions Logical. Whether to keep the predictions of the cross-validation models. Defaults to FALSE.
keep_cross_validation_fold_assignment Logical. Whether to keep the cross-validation fold assignment. Defaults to FALSE.
score_each_iteration Logical. Whether to score during each iteration of model training. Defaults to FALSE.
fold_assignment Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for 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 and anomomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client. Must be one of: "AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "AUCPR", "lift_top_group", "misclassification", "mean_per_class_error", "custom", "custom_increasing". Defaults to AUTO.

stopping_tolerance
Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much) Defaults to 0.001.

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

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

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

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

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

quiet_mode
Logical. Enable quiet mode Defaults to TRUE.

checkpoint
Model checkpoint to resume training with.

export_checkpoints_dir
Automatically export generated models to this directory.

ntrees
(same as n_estimators) Number of trees. Defaults to 50.

max_depth
Maximum tree depth. 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.
monotone_constraints A mapping representing monotonic constraints. Use +1 to enforce an increasing constraint and -1 to specify a decreasing constraint.
score_tree_interval Score the model after every so many trees. Disabled if set to 0. Defaults to 0.
min_split_improvement (same as gamma) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
gamma (same as min_split_improvement) Minimum relative improvement in squared error reduction for a split to happen Defaults to 0.0.
nthread Number of parallel threads that can be used to run XGBoost. Cannot exceed H2O cluster limits (-nthreads parameter). Defaults to maximum available Defaults to -1.
save_matrix_directory Directory where to save matrices passed to XGBoost library. Useful for debugging.
build_tree_one_node Logical. Run on one node only; no network overhead but fewer cpus used. Suitable for small datasets. Defaults to FALSE.
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
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 1.0.

**reg_alpha**  
L1 regularization Defaults to 0.0.

**dmatrix_type**  
Type of DMatrix. For sparse, NAs and 0 are treated equally. Must be one of: "auto", "dense", "sparse". Defaults to auto.

**backend**  
Backend. By default (auto), a GPU is used if available. Must be one of: "auto", "gpu", "cpu". Defaults to auto.

**gpu_id**  
Which GPU to use. Defaults to 0.

**verbose**  
Logical. Print scoring history to the console (Metrics per tree). Defaults to FALSE.

**Examples**

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

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

# Set predictors and response; set response as a factor
titanic['survived'] <- as.factor(titanic['survived'])
predictors <- setdiff(colnames(titanic), colnames(titanic)[2:3])
response <- "survived"

# Split the dataset into train and valid
splits <- h2o.splitFrame(data = titanic, ratios = .8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]

# Train the XGB model
titanic_xgb <- h2o.xgboost(x = predictors, y = response,
 training_frame = train, validation_frame = valid,
 booster = "dart", normalize_type = "tree",
 seed = 1234)

## End(Not run)
```
**h2o.xgboost.available**  
*Determines whether an XGBoost model can be built*

**Description**

Ask the H2O server whether an XGBoost model can be built. (Depends on availability of native backend.) Returns True if an XGBoost model can be built, or False otherwise.

**Usage**

```r
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**

```r
h2o.year(x)

year(x)
```

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

**Arguments**

- `x`  
  An H2OFrame object.

**Details**

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

**Value**

An H2OFrame object containing the entries of `x` converted to years

**See Also**

`h2o.month`
Examples

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

hdf <- h2o.importFile(f)
h2o.year(hdf["ds9"])
## End(Not run)
```

H2OAutoML-class

The H2OAutoML class

Description

This class represents an H2OAutoML object

H2OClusteringModel-class

The H2OClusteringModel object.

Description

This virtual class represents a clustering model built by H2O.

Details

This object has slots for the key, which is a character string that points to the model key existing in the H2O cluster, the data used to build the model (an object of class H2OFrame).

Slots

- `model_id` A character string specifying the key for the model fit in the H2O cluster’s key-value store.
- `algorithm` A character string specifying the algorithm that was used to fit the model.
- `parameters` A list containing the parameter settings that were used to fit the model that differ from the defaults.
- `allparameters` A list containing all parameters used to fit the model.
- `model` A list containing the characteristics of the model returned by the algorithm.
  - `size` The number of points in each cluster.
  - `totss` Total sum of squared error to grand mean.
  - `withinss` A vector of within-cluster sum of squared error.
  - `tot_withinss` Total within-cluster sum of squared error.
  - `betweenss` Between-cluster sum of squared error.
H2OConnection-class

Description

This class represents a connection to an H2O cluster.

Usage

## S4 method for signature 'H2OConnection'
show(object)

Arguments

object an H2OConnection object.

Details

Because H2O is not a master-slave architecture, there is no restriction on which H2O node is used to establish the connection between R (the client) and H2O (the server).

A new H2O connection is established via the h2o.init() function, which takes as parameters the 'ip' and 'port' of the machine running an instance to connect with. The default behavior is to connect with a local instance of H2O at port 54321, or to boot a new local instance if one is not found at port 54321.

Slots

- ip A character string specifying the IP address of the H2O cluster.
- port A numeric value specifying the port number of the H2O cluster.
- name A character value specifying the name of the H2O cluster.
- proxy A character specifying the proxy path of the H2O cluster.
- https Set this to TRUE to use https instead of http.
- cacert Path to a CA bundle file with root and intermediate certificates of trusted CAs.
- insecure Set this to TRUE to disable SSL certificate checking.
- username Username to login with.
- password Password to login with.
- use_spnego Set this to TRUE to use SPNEGO authentication.
- cookies Cookies to add to request
- context_path Context path which is appended to H2O server location.
- mutable An H2OConnectionMutableState object to hold the mutable state for the H2O connection.
The H2OConnectionMutableState class

Description
This class represents the mutable aspects of a connection to an H2O cluster.

Slots
- session_id A character string specifying the H2O session identifier.
- key_count A integer value specifying count for the number of keys generated for the session_id.

The H2OCoxPHModel object.

Description
Virtual object representing H2O’s CoxPH Model.

Usage

### S4 method for signature 'H2OCoxPHModel'
show(object)

### S3 method for class 'H2OCoxPHModel'
coef(object, ...)

extractAIC(fit, scale, k = 2, ...)

logLik(object, ...)

## S3 method for class 'H2OCoxPHModel'

survfit.H2OCoxPHModel(formula, newdata, ...)

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**

```r
## 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 summary compatible with CoxPH summary used in the survival package.

### H2OFrame-class

The H2OFrame class

**Description**

This class represents an H2OFrame object.

### H2OFrame-Extract

Extract or Replace Parts of an H2OFrame Object

**Description**

Operators to extract or replace parts of H2OFrame objects.
Usage

```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
# 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.

---

**H2OLeafNode-class**  
*The H2OLeafNode class.*

## Description

This class represents a single leaf node in an `H2OTree`.

## Details

```r
#' @aliases H2OLeafNode
```
H2OModel-class

The H2OModel object.

Description

This virtual class represents a model built by H2O.

Usage

## S4 method for signature 'H2OModel'
show(object)

Arguments

object an H2OModel object.

Details

This object has slots for the key, which is a character string that points to the model key existing in
the H2O cluster, the data used to build the model (an object of class H2OFrame).

Slots

model_id A character string specifying the key for the model fit in the H2O cluster's key-value
store.
algorithm A character string specifying the algorithm that were used to fit the model.
parameters A list containing the parameter settings that were used to fit the model that differ
from the defaults.
allparameters A list 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

See Also

H2OModel for the final model types.
**H2OModelMetrics-class**

*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

---

**H2ONode-class**

*The H2ONode class.*

**Description**

The H2ONode class.

**Usage**

```r
## S4 method for signature 'H2ONode'
show(object)
```
Arguments

object an H2OModel object.

Slots

id An integer representing node’s unique identifier. Generated by H2O.
levels A character representing categorical levels on split from parent’s node belonging into this node. NULL for root node or non-categorical splits.

#’ @aliases H2ONode

Description

A class to contain the information for segment models.

Usage

## S4 method for signature 'H2OSegmentModels'
show(object)

Arguments

object an H2OModel object.

Slots

segment_models_id the identifier for the segment models collections

Description

A class to contain the information for background segment models jobs.

Slots

job_key a character key representing the identification of the job process.
segment_models_id the final identifier for the segment models collections

See Also

H2OSegmentModels for the final segment models types.
H2OSplitNode-class

The H2OSplitNode class.

Description

This class represents a single non-terminal node in an H2OTree.

Slots

threshold A numeric split threshold, typically when the split column is numerical.
left_child A H2ONodeOrNULL representing the left child node, if a node has one.
right_child A H2ONodeOrNULL representing the right child node, if a node has one.
split_feature A character representing the name of the column this node splits on.
left_levels A character representing the levels of a categorical feature heading to the left child of this node. NA for non-categorical split.
right_levels A character representing the levels of a categorical feature heading to the right child of this node. NA for non-categorical split.
na_direction A character representing the direction of NA values. LEFT means NA values go to the left child node, RIGH means NA values go to the right child node.

H2OTree-class

The H2OTree class.

Description

This class represents a model of a Tree built by one of H2O’s algorithms (GBM, Random Forest).

Usage

## S4 method for signature 'H2OTree'
show(object)

Arguments

object an H2OTree object.

Slots

root_node A H2ONode representing the beginning of the tree behind the model. Allows further tree traversal.
left_children An integer vector with left child nodes of tree’s nodes
right_children An integer vector with right child nodes of tree’s nodes
node_ids An integer representing identification number of a node. Node IDs are generated by H2O.
descriptions A character vector with descriptions for each node to be found in the tree. Contains split threshold if the split is based on numerical column. For categorical splits, it contains list of categorical levels for transition from the parent node.
model_id  A character with the name of the model this tree is related to.

tree_number  An integer representing the order in which the tree has been built in the model.

tree_class  A character representing name of tree’s class. Number of tree classes equals to the number of levels in categorical response column. As there is exactly one class per categorical level, name of tree’s class equals to the corresponding categorical level of response column. In case of regression and binomial, the name of the categorical level is ignored can be omitted, as there is exactly one tree built in both cases.

thresholds  A numeric split thresholds. Split thresholds are not only related to numerical splits, but might be present in case of categorical split as well.

features  A character with names of the feature/column used for the split.

levels  A character representing categorical levels on split from parent’s node belonging into this node. NULL for root node or non-categorical splits.

nas  A character representing if NA values go to the left node or right node. May be NA if node is a leaf.

predictions  A numeric representing predictions for each node in the graph.

<table>
<thead>
<tr>
<th>housevotes</th>
<th>United States Congressional Voting Records 1984</th>
</tr>
</thead>
</table>

**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**

Edgar Anderson’s Iris Data

Description

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

Format

A data frame with 150 rows and 5 columns

Source


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

is.character

Check if character

Description

Check if character

Usage

is.character(x)

Arguments

x An H2OFrame object

Examples

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

f <- "http://s3.amazonaws.com/h2o-public-test-data/smalldata/coxph_test/heart.csv"
heart <- h2o.importFile(f)

heart["transplant"] <- as.character(heart["transplant"])
is.character(heart["transplant"])

## End(Not run)
is.factor  
\textit{Check if factor}

\textbf{Description}

Check if factor

\textbf{Usage}

\texttt{is.factor(x)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{An H2OFrame object}
\end{itemize}

\is.h2o  
\textit{Is H2O Frame object}

\textbf{Description}

Test if object is H2O Frame.

\textbf{Usage}

\texttt{is.h2o(x)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{An R object.}
\end{itemize}

\textbf{Examples}

\begin{verbatim}
## Not run:
library(h2o)
h2o.init()

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

is.h2o(frame)

## End(Not run)
\end{verbatim}
is.numeric  

Check if numeric

**Description**

Check if numeric

**Usage**

`is.numeric(x)`

**Arguments**

- `x`  
  An H2OFrame object

---

**Keyed-class**  

Virtual Keyed class

**Description**

Base class for all objects having a persistent representation on backend.

**length,H2OTree-method**  

*Overrides the behavior of length() function on H2OTree class. Returns number of nodes in an H2OTree*

**Description**

Overrides the behavior of length() function on H2OTree class. Returns number of nodes in an H2OTree

**Usage**

```r
## S4 method for signature 'H2OTree'
length(x)
```

**Arguments**

- `x`  
  An H2OTree to count nodes for.
Logical-or  

**Logical or for H2OFrames**

**Description**
Logical or for H2OFrames

**Usage**
```
'\|\|(x, y)
```

**Arguments**
- `x`: An H2OFrame object
- `y`: An H2OFrame object

---

**ModelAccessors**  

**Accessor Methods for H2OModel Object**

**Description**
Function accessor methods for various H2O output fields.

**Usage**
```r
getParms(object)
```

```
## S4 method for signature 'H2OModel'
getParms(object)
```

```
getCenters(object)
```

```
getCentersStd(object)
```

```
getWithinSS(object)
```

```
getTotWithinSS(object)
```

```
getBetweenSS(object)
```

```
getTotSS(object)
```

```
getIterations(object)
```

```
getClusterSizes(object)
```

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

## S4 method for signature 'H2OClusteringModel'
getWithinSS(object)

## S4 method for signature 'H2OClusteringModel'
getTotWithinSS(object)

## S4 method for signature 'H2OClusteringModel'
getBetweenSS(object)

## S4 method for signature 'H2OClusteringModel'
getTotSS(object)

## S4 method for signature 'H2OClusteringModel'
getIterations(object)

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

Arguments

object an H2OModel class object.

---

names.H2OFrame

Column names of an H2OFrame

Description

Column names of an H2OFrame

Usage

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

Arguments

x An H2OFrame

Examples

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

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

names(frame)

## End(Not run)
Ops.H2OFrame

S3 Group Generic Functions for H2O

Description

Methods for group generic functions and H2O objects.

Usage

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

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

## S3 method for class 'H2OFrame'
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)
\texttt{h2o.length(x)}

\texttt{## S3 replacement method for class 'H2OFrame'
names(x) <- value}

\texttt{colnames(x) <- value}

\textbf{Arguments}

- \texttt{e1} \hspace{1cm} \texttt{object}
- \texttt{e2} \hspace{1cm} \texttt{object}
- \texttt{x} \hspace{1cm} \texttt{object}
- \texttt{...} \hspace{1cm} Further arguments passed to or from other methods.
- \texttt{na.rm} \hspace{1cm} \texttt{logical. whether or not missing values should be removed}
- \texttt{y} \hspace{1cm} \texttt{object}
- \texttt{value} \hspace{1cm} \texttt{To be assigned}

\textbf{plot.H2OModel} \hspace{6cm} \textit{Plot an H2O Model}

\textbf{Description}

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

\textbf{Usage}

\texttt{## S3 method for class 'H2OModel'
plot(x, timestep = "AUTO", metric = "AUTO", ...)}

\textbf{Arguments}

- \texttt{x} \hspace{1cm} A fitted \texttt{H2OModel} object for which the scoring history plot is desired.
- \texttt{timestep} \hspace{1cm} A unit of measurement for the x-axis.
- \texttt{metric} \hspace{1cm} A unit of measurement for the y-axis.
- \texttt{...} \hspace{1cm} additional arguments to pass on.

\textbf{Details}

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

\textbf{Value}

Returns a scoring history plot.

\textbf{See Also}

\texttt{h2o.deeplearning, h2o.gbm, h2o.glm, h2o.randomForest} for model generation in h2o.
Examples

```r
## Not run:
if (requireNamespace("mlbench", quietly=TRUE)) {
  library(h2o)
  h2o.init()

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

  gbm <- h2o.gbm(x = 1:10, y = "y", training_frame = train, validation_frame = valid,
                  ntrees=500, learn_rate=0.01, score_each_iteration = TRUE)
  plot(gbm)
  plot(gbm, timestep = "duration", metric = "deviance")
  plot(gbm, timestep = "number_of_trees", metric = "deviance")
  plot(gbm, timestep = "number_of_trees", metric = "rmse")
  plot(gbm, timestep = "number_of_trees", metric = "mae")
}
## End(Not run)
```

plot.H2OTabulate

Plot an H2O Tabulate Heatmap

Description

Plots the simple co-occurrence based tabulation of X vs Y as a heatmap, where X and Y are two Vecs in a given dataset. This function requires suggested ggplot2 package.

Usage

```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`
Examples

```r
## Not run:
library(h2o)
h2o.init()
df <- as.h2o(iris)
tab <- h2o.tabulate(data = df, x = "Sepal.Length", y = "Petal.Width",
                   weights_column = NULL, nbins_x = 10, nbins_y = 10)
plot(tab)
## End(Not run)
```

predict.H2OAutoML

Predict on an AutoML object

Description

Obtains predictions from an AutoML object.

Usage

```r
## 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

## S3 method for class 'H2OModel'
predict(object, newdata, ...)

## S3 method for class 'H2OModel'
h2o.predict(object, newdata, ...)

Arguments

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

Examples

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

f <- "https://s3.amazonaws.com/h2o-public-test-data/smalldata/glm_test/insurance.csv"
insurance <- h2o.importFile(f)
predictors <- colnames(insurance)[1:4]
response <- "Claims"
insurance["Group"] <- as.factor(insurance["Group"])
insurance["Age"] <- as.factor(insurance["Age"])
splits <- h2o.splitFrame(data = insurance, ratios = 0.8, seed = 1234)
train <- splits[[1]]
valid <- splits[[2]]
insurance_gbm <- h2o.gbm(x = predictors, y = response,
training_frame = train,
predict_contributions.H2OModel

Predict feature contributions - SHAP values on an H2O Model (only DRF, GBM and XGBoost models).

Description

Returned H2OFrame has shape (#rows, #features + 1) - there is a feature contribution column for each input feature, the last column is the model bias (same value for each row). The sum of the feature contributions and the bias term is equal to the raw prediction of the model. Raw prediction of tree-based model is the sum of the predictions of the individual trees before the inverse link function is applied to get the actual prediction. For Gaussian distribution the sum of the contributions is equal to the model prediction.

Usage

predict_contributions.H2OModel(object, newdata, ...)

h2o.predict_contributions(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

Note: Multinomial classification models are currently not supported.

Value

Returns an H2OFrame contain feature contributions for each input row.

See Also

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

## Not run:
library(h2o)
h2o.init()
prostate_path <- system.file("extdata", "prostate.csv", package = "h2o")
prostate <- h2o.uploadFile(path = prostate_path)
prostate_gbm <- h2o.gbm(3:9, "AGE", prostate)
h2o.predict(prostate_gbm, prostate)
h2o.predict_contributions(prostate_gbm, prostate)

## End(Not run)

predict_leaf_node_assignment.H2OModel

Predict the Leaf Node Assignment on an H2O Model

Description

Obtains leaf node assignment from fitted H2O model objects.

Usage

predict_leaf_node_assignment.H2OModel(
  object,
  newdata,
  type = c("Path", "Node_ID"),
  ...
)

h2o.predict_leaf_node_assignment(
  object,
  newdata,
  type = c("Path", "Node_ID"),
  ...
)

Arguments

object a fitted H2OModel object for which prediction is desired
newdata An H2OFrame object in which to look for variables with which to predict.
type choice of either "Path" when tree paths are to be returned (default); or "Node_ID" when the output
... additional arguments to pass on.

Details

For every row in the test set, return the leaf placements of the row in all the trees in the model. Placements can be represented either by paths to the leaf nodes from the tree root or by H2O’s internal identifiers. The order of the rows in the results is the same as the order in which the data was loaded.
Value

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

See Also

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

Examples

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

print.H2OFrame

Print An H2OFrame

Description

Print An H2OFrame

Usage

```r
## S3 method for class 'H2OFrame'
print(x, n = 6L, m = 200L, ...)
```

Arguments

- `x` An H2OFrame object
- `n` An (Optional) A single integer. If positive, number of rows in x to return. If negative, all but the n first/last number of rows in x. Anything bigger than 20 rows will require asking the server (first 20 rows are cached on the client).
- `m` An (Optional) A single integer. If positive, number of columns in x to return. If negative, all but the m first/last number of columns in x.
- `...` Further arguments to be passed from or to other methods.

Examples

```r
## Not run:
library(h2o)
h2o.init()
cars <- h2o.importFile(f)
print(cars, n = 8)
## End(Not run)
```
print.H2OTable  

*Print method for H2OTable objects*

**Description**

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

**Usage**

```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

**Examples**

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

cars <- h2o.importFile(f)
print(cars, header = TRUE)

## End(Not run)
```

---

**prostate**  

*Prostate Cancer Study*

**Description**

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

**Format**

A data frame with 380 rows and 9 columns

**Source**

range.H2OFrame

Range of an H2O Column

Description

Range of an H2O Column

Usage

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

Arguments

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

Examples

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

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

## End(Not run)

scale

Scaling and Centering of an H2OFrame

Description

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

Usage

## S3 method for class 'H2OFrame'
scale(x , center = TRUE, scale = TRUE)

Arguments

x An H2OFrame object.
center either a logical value or numeric vector of length equal to the number of
columns of x.
scale either a logical value or numeric vector of length equal to the number of
columns of x.
staged_predict_proba.H2OModel

Predict class probabilities at each stage of an H2O Model

Description

The output structure is analogous to the output of `h2o.predict_leaf_node_assignment`. For each tree \( t \) and class \( c \) there will be a column \( T_t.C_c \) (eg. \( T_3.C_1 \) for tree 3 and class 1). The value will be the corresponding predicted probability of this class by combining the raw contributions of trees \( T_1.C_c,\ldots,T_t.C_c \). Binomial models build the trees just for the first class and values in columns \( T_x.C_1 \) thus correspond to the the probability \( p_0 \).

Usage

```r
staged_predict_proba.H2OModel(object, newdata, ...)

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

Arguments

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

Value

Returns an `H2OFrame` object with predicted probability for each tree in the model.

See Also

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

Examples

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

# Scale and center all the numeric columns in iris data set
iris_scaled <- scale(iris_hf[, 1:4])

## End(Not run)

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

## End(Not run)

---

### str.H2OFrame

Display the structure of an H2OFrame object

#### Description

Display the structure of an H2OFrame object

#### Usage

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

#### Arguments

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

---

### summary.H2OCoxPHModel-method

Summary method for H2OCoxPHModel objects

#### Description

Summary method for H2OCoxPHModel objects

#### Usage

```
## S4 method for signature 'H2OCoxPHModel'
summary(object, conf.int = 0.95, scale = 1)
```

#### Arguments

- `object`: an H2OCoxPHModel object.
- `conf.int`: a specification of the confidence interval.
- `scale`: a scale.
### summary,H2OGrid-method

**Format grid object in user-friendly way**

**Description**

Format grid object in user-friendly way

**Usage**

```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)
**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", TRUE)[package == "data.table"]
)
```

**Arguments**

- `package`: character scalar name of a package that we Suggests or Enhances on.
- `version`: character scalar required version of a package.
- `use`: logical scalar, extra escape option, to be used as global option.

**Details**

We use this function to control csv read/write with optional `data.table` package. Currently data.table is enabled by default for some operations, to disable it set options("h2o.use.data.table"=FALSE). It is possible to control just `fread` or `fwrite` with options("h2o.fread"=FALSE, "h2o fwrite"=FALSE). `h2o fread` and `h2o fwrite` options are not handled in this function but next to `fread` and `fwrite` calls.

**See Also**

`as.h2o.data.frame`, `as.data.frame.H2OFrame`

**Examples**

```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)
```
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

Logical and for H2OFrames

Description
Logical and for H2OFrames

Usage
`&&`(x, y)

Arguments
- `x`: An H2OFrame object
- `y`: An H2OFrame object

Examples
```r
## Not run:
library(h2o)
h2o.init()
h2o.shutdown(prompt = FALSE)
Sys.sleep(3)
## End(Not run)
```
Index

!.H2OFrame (Ops.H2OFrame), 310
*Topic datasets
  .h2o.__ALL_CAPABILITIES, 15
  .h2o.__CREATE_FRAME, 16
  .h2o.__DECRYPTION_SETUP, 16
  .h2o.__DKV, 17
  .h2o.__FRAMES, 17
  .h2o.__IMPORT, 18
  .h2o.__JOBS, 18
  .h2o.__LOGANDECHO, 18
  .h2o.__MODELS, 19
  .h2o.__PARSE_SETUP, 20
  .h2o.__RAPIDS, 20
  .h2o.__REST_API_VERSION, 20
  .h2o.__W2V_SYNONYMS, 21
  .h2o.primitives, 15
  .pkg.env, 21
  australia, 30
  housevotes, 304
  iris, 305
  prostate, 318
  walking, 324
*Topic package
  h2o-package, 8
  .addParm, 9
  .collapse, 10
  .h2o.__ALL_CAPABILITIES, 15
  .h2o.__CREATE_FRAME, 16
  .h2o.__DECRYPTION_SETUP, 16
  .h2o.__DKV, 17
  .h2o.__EXPORT_FILES, 17
  .h2o.__FRAMES, 17
  .h2o.__IMPORT, 18
  .h2o.__JOBS, 18
  .h2o.__LOGANDECHO, 18
  .h2o.__MODELS, 19
  .h2o.__MODEL_BUILDERS, 19
  .h2o.__MODEL_METRICS, 19
  .h2o.__PARSE_SETUP, 20
  .h2o.__RAPIDS, 20
  .h2o.__REST_API_VERSION, 20
  .h2o.__SEGMENT_MODELS_BUILDERS, 21
  .h2o.__W2V_SYNONYMS, 21
  .h2o.__checkConnectionHealth, 16
  .h2o.doGET, 10
  .h2o.doHEAD, 11
  .h2o.doPUT, 11
  .h2o.doPOST, 12
  .h2o.doSafeGET, 13
  .h2o.doSafePOST, 14
  .h2o.is_progress, 14
  .h2o.locate, 15
  .h2o.primitives, 15
  .pkg.env, 21
  .skip_if_not_developer, 22
  .verify_dataxy, 22
  [.H2OFrame-method (H2OFrame-Extract), 297
  [.H2OFrame (H2OFrame-Extract), 297
  [[.H2OFrame (H2OFrame-Extract), 297
  [[<-.H2OFrame (H2OFrame-Extract), 297
  $.H2OFrame (H2OFrame-Extract), 297
  $<-.H2OFrame (H2OFrame-Extract), 297
  %*% (Ops.H2OFrame), 310
  %in% (h2o.match), 175
  &amp;, 324
  aaa, 22
  abs, 34
  acos, 35
  all, 38, 40
  apply, 23, 23
  as.character.H2OFrame, 23
  as.data.frame.H2OFrame, 24, 323
  as.data.frame.H2OSegmentModels, 25
  as.factor, 26, 26, 43
  as.h2o, 26
  as.h2o.data.frame, 323
  as.matrix.H2OFrame, 28
  as.numeric, 28, 42
  as.vector.H2OFrame, 29
  australia, 30
  cbind, 52
  ceiling, 53
coef.H2OCoxPHModel (H2OCoxPHModel-class), 296
coef.H2OCoxPHModelSummary (H2OCoxPHModelSummary-class), 297
colMeans, 177
colnames, 30, 60
colnames<- (Ops.H2OFrame), 310
cor (h2o.cor), 64
cos, 65
cosh, 66
cummax, 73
cummin, 74
cumprod, 75
cumsum, 75
cut.H2OFrame (h2o.cut), 76
data.table, 323
day (h2o.day), 77
dayOfWeek (h2o.dayOfWeek), 78
ddply, 79
dim, 31, 91
dim.H2OFrame, 31
dimnames, 91
dimnames.H2OFrame, 31

expc, 97
extractAIC.H2OCoxPHModel (H2OCoxPHModel-class), 296
feature_frequencies.H2OModel, 32
floor, 103
fread, 24, 323
fwrite, 27, 323
generate_col_ind, 33
get_seed.H2OModel, 33
getBetweenSS (ModelAccessors), 308
getBetweenSS, H2OClusteringModel-method (ModelAccessors), 308
getCenters (ModelAccessors), 308
getCenters, H2OClusteringModel-method (ModelAccessors), 308
getCentersStd (ModelAccessors), 308
getCentersStd, H2OClusteringModel-method (ModelAccessors), 308
getClusterSizes (ModelAccessors), 308
getClusterSizes, H2OClusteringModel-method (ModelAccessors), 308
getIterations (ModelAccessors), 308
getIterations, H2OClusteringModel-method (ModelAccessors), 308
getParms (ModelAccessors), 308

getParms, H2OModel-method (ModelAccessors), 308
getTotSS (ModelAccessors), 308
getTotSS, H2OClusteringModel-method (ModelAccessors), 308
getTotWithinSS (ModelAccessors), 308
getTotWithinSS, H2OClusteringModel-method (ModelAccessors), 308
getWithinSS (ModelAccessors), 308
getWithinSS, H2OClusteringModel-method (ModelAccessors), 308

h2o (h2o-package), 8
h2o-package, 8
h2o.abs, 34
h2o.accuracy (h2o.metric), 182
h2o.acos, 35
h2o.aggregated_frame, 35
h2o.aggregator, 36
h2o.aic, 37
h2o.all, 38
h2o.anomaly, 39
h2o.any, 40
h2o.anyFactor, 40
h2o.arrange, 41
h2o.as_date, 44
h2o.ascharacter, 41
h2o.asfactor, 42
h2o.asnumeric, 43
h2o.assign, 43, 229
h2o.auc, 44, 125, 130, 183, 188, 229
h2o.aucpr, 45
h2o.automl, 46
h2o.betweenss, 50, 164
h2o.biases, 51
h2o.bottomN, 51
h2o.chind, 52
h2o.ceiling, 53
h2o.centers, 54, 164
h2o.centersSTD, 54, 164
h2o.centroid_stats, 55
h2o.clearLog, 55, 199, 249, 250
h2o.cluster_sizes, 57, 164
h2o.clusterInfo, 56
h2o.clusterIsUp, 56
h2o.clusterStatus, 57
h2o.coef, 58
h2o.coef_norm, 59
h2o.colnames, 59
h2o.columns_by_type, 60
h2o.computeGram, 61
h2o.confusionMatrix, 61, 130
INDEX

h2o.confusionMatrix, H2OModel-method (h2o.confusionMatrix), 61
h2o.confusionMatrix, H2OModelMetrics-method (h2o.confusionMatrix), 61
h2o.connect, 63
h2o.cor, 64
h2o.cos, 65
h2o.cosh, 66
h2o.coxph, 66
h2o.createFrame, 68
h2o.cross_validation_fold_assignment, 70
h2o.cross_validation_holdout_predictions, 71
h2o.cross_validation_models, 71
h2o.cross_validation_predictions, 72
h2o.cummax, 73
h2o.cummin, 74
h2o.cumprod, 74
h2o.cumsum, 75
h2o.cut, 76
h2o.day, 77, 78, 141
h2o.dayOfWeek, 78
h2o.dct, 78
h2o.ddply, 79
h2o.decryptionSetup, 80, 144, 200, 201
h2o.deepfeatures, 81
h2o.deeplearning, 39, 81, 82, 311, 314
h2o.describe, 89
h2o.difflag1, 90
h2o.dim, 90
h2o.dimnames, 91
h2o.distance, 92
h2o.download_model, 94, 278
h2o.download_mojo, 94
h2o.download_pojo, 95
h2o.downloadAllLogs, 92
h2o.downloadCSV, 93
h2o.entropy, 96
h2o.error (h2o.metric), 182
h2o.exp, 97
h2o.exportFile, 97
h2o.exportHDFS, 98
h2o.F0point5 (h2o.metric), 182
h2o.F1 (h2o.metric), 182
h2o.F2 (h2o.metric), 182
h2o.fallout (h2o.metric), 182
h2o.feature_frequencies (feature_frequencies.H2OModel), 32
h2o.fillna, 99
h2o.filterNACols, 100
h2o.find_row_by_threshold, 101
h2o.find_threshold_by_max_metric, 102
h2o.findSynonyms, 100
h2o.floor, 102
h2o.flow, 103
h2o.fnr (h2o.metric), 182
h2o.fpr (h2o.metric), 182
h2o.gainsLift, 103
h2o.gainsLift, H2OModel-method (h2o.gainsLift), 103
h2o.gainsLift, H2OModelMetrics-method (h2o.gainsLift), 103
h2o.gam, 104
h2o.gbm, 32, 109, 311, 314, 315, 317, 320
h2o.generic, 114
h2o.genericModel, 115
h2o.get_automl, 122
h2o.get_leaderboard, 122
h2o.get_nframes_actual, 123
h2o.get_seed (get_seed.H2OModel), 33
h2o.get_segment_models, 124
h2o.getAutoML (h2o.get_automl), 122
h2o.getConnection, 116
h2o.getFrame, 116
h2o.getGLMFullRegularizationPath, 117
h2o.getGrid, 118
h2o.getId, 118
h2o.getModel, 119
h2o.getModelTree, 120
h2o.getTimezone, 120
h2o.getTypes, 121
h2o.getVersion, 121
h2o.giniCoef, 45, 46, 124, 125, 130, 183
h2o.glm, 9, 125, 311, 314
h2o.glrm, 131, 207, 210, 223
h2o.grep, 133
h2o.grid, 134
h2o.group_by, 136
h2o.gsub, 137
h2o.head, 138
h2o.HGLMMetrics, 139
h2o.hist, 139
h2o.hit_ratio_table, 140
h2o.hour, 141
h2o.ifelse, 141
h2o.import_hive_table, 145
h2o.import_mojo, 145
h2o.import_sql_select, 144, 146
h2o.import_sql_table, 144, 147
h2o.importFile, 80, 142, 200
h2o.importFolder (h2o.importFile), 142
h2o.importHDFS (h2o.importFile), 142
h2o.impute, 148
h2o.init, 57, 149, 242
h2o.insertMissingValues, 152
h2o.interaction, 153
h2o.is_client, 160
h2o.isax, 155
h2o.ischaracter, 156
h2o.isfactor, 156
h2o.isnumeric, 157
h2o.isolationForest, 158
h2o.keyof, 160
h2o.keyof, H2OAutoML-method (h2o.keyof), 160
h2o.keyof, H2OFrame-method (h2o.keyof), 160
h2o.keyof, H2OGrid-method (h2o.keyof), 160
h2o.keyof, H2OModel-method (h2o.keyof), 160
h2o.keyof, Keyed-method (h2o.keyof), 160
h2o.kfold_column, 161
h2o.killMinus3, 161
h2o.kmeans, 133, 162
h2o.kurtosis, 164
h2o.length (Ops.H2OFrame), 310
h2o.levels, 165
h2o.list_all_extensions, 166
h2o.list_api_extensions, 166
h2o.list_core_extensions, 166
h2o.list_jobs, 166
h2o.listTimezones, 165
h2o.loadGrid, 167
h2o.loadModel, 167, 234
h2o.log, 168
h2o.log10, 169
h2o.log1p, 169
h2o.log2, 170
h2o.logAndEcho, 171
h2o.logLoss, 130, 171
h2o.ls, 172, 229
h2o.lstrip, 173
h2o.mae, 173
h2o.make_metrics, 174
h2o.makeGLMModel, 174
h2o.match, 175
h2o.max, 176
h2o.maxPerClassError (h2o.metric), 182
h2o.mcc (h2o.metric), 182
h2o.mean, 177
h2o.mean_per_class_accuracy (h2o.metric), 182
h2o.mean_per_class_error, 178
h2o.mean_residual_deviance, 179
h2o.median, 179
h2o.melt, 180
h2o.merge, 181
h2o.metric, 45, 46, 125, 178, 182, 188, 229
h2o.min, 184
h2o.missrate (h2o.metric), 182
h2o.mktime, 184
h2o.mojo_predict_csv, 185
h2o.mojo_predict_df, 186
h2o.month, 77, 78, 187, 282, 293
h2o.mse, 45, 46, 130, 178, 183, 187, 188, 229
h2o.na_omit, 192
h2o.nacnt, 188
h2o.naiveBayes, 189
h2o.names, 191
h2o.nchar, 193
h2o.ncol, 193
h2o.networkTest, 194
h2o.nlevels, 194
h2o.no_progress, 195
h2o.nrow, 196
h2o.null_deviance, 196
h2o.null_dof, 197
h2o.num_iterations, 164, 198
h2o.num_valid_substrings, 198
h2o.openLog, 55, 199, 249, 250
h2o.parseRaw, 143, 144, 199, 202
h2o.parseSetup, 80, 200, 201
h2o.partialPlot, 202
h2o.performance, 45, 46, 62, 104, 125, 130, 178, 183, 188, 203, 229
h2o.pivot, 205
h2o.pr_auc (h2o.aucpr), 45
h2o.pcomp, 133, 205
h2o.precision (h2o.metric), 182
h2o.predict, 207
h2o.predict.H2OAutoML (predict.H2OAutoML), 313
h2o.predict.H2OModel (predict.H2OModel), 314
h2o.predict_contributions (predict_contributions.H2OModel), 315
h2o.predict_json, 208
h2o.predict_leaf_node_assignment, 320
h2o.predict_leaf_node_assignment (predict_leaf_node_assignment.H2OModel), 316
h2o.print, 209
h2o.prod, 209
h2o.proj_archetypes, 210
H2OAnomalyDetectionMetrics-class
(H2OModelMetrics-class), 301
H2OAnomalyDetectionModel-class
(H2OModel-class), 300
H2OAutoEncoderMetrics-class
(H2OModelMetrics-class), 301
H2OAutoEncoderModel-class
(H2OModel-class), 39
H2OAutoML, 49, 122, 313
H2OAutoML-class, 294
H2OBinomialMetrics, 44, 45, 62, 104, 124, 125, 171, 178, 183, 188, 229
H2OBinomialMetrics-class
(H2OModelMetrics-class), 301
H2OBinomialModel, 130, 191
H2OBinomialModel-class
(H2OModel-class), 300
H2OClusteringMetrics-class
(H2OModelMetrics-class), 301
H2OClusteringModel, 35, 50, 54, 55, 57, 163, 198, 269, 270, 286
H2OClusteringModel-class, 294
H2OConnection, 57, 116
H2OConnection (H2OConnection-class), 295
H2OConnection-class, 295
H2OConnectionMutableState, 296
H2OCoxPHMetrics-class
(H2OModelMetrics-class), 301
H2OCoxPHModel (H2OCoxPHModel-class), 296
H2OCoxPHModel-class, 296
H2OCoxPHModelSummary
(H2OCoxPHModelSummary-class), 297
H2OCoxPHModelSummary-class, 297
H2ODimReductionMetrics-class
(H2OModelMetrics-class), 301
H2ODimReductionModel, 133, 207, 210, 223, 239, 257
H2ODimReductionModel-class
(H2OModel-class), 300
H2OFrame-class, 297
H2OFrame-Extract, 297
H2OGrid, 233
H2OGrid (H2OGrid-class), 298
H2OGrid-class, 298
H2OLeafNode-class, 299
H2OModel, 32, 33, 38, 51, 58, 61, 62, 70–72, 81, 98, 104, 117, 119, 123, 130, 140, 168, 173, 174, 179, 196,
H2OModel (H2OModel-class), 300
H2OModel-class, 300
H2OModelFuture-class, 300
H2OModelMetrics, 38, 51, 62, 104, 171, 175, 183, 187, 188, 196, 197, 204, 227–229, 282
H2OModelMetrics
(H2OModelMetrics-class), 301
H2OModelMetrics-class, 301
H2OMultinomialMetrics, 62, 171, 188, 229
H2OMultinomialMetrics-class
(H2OModelMetrics-class), 301
H2OMultinomialModel, 191
H2OMultinomialModel-class
(H2OModel-class), 300
H2ONode-class, 301
H2OOrdinalModel-class
(H2OModelMetrics-class), 301
H2OOrdinalModel-class (H2OModel-class), 300
H2ORegressionMetrics, 188, 229
H2ORegressionMetrics-class
(H2OModelMetrics-class), 301
H2ORegressionModel, 130
H2ORegressionModel-class
(H2OModel-class), 300
H2OSegmentModels, 25, 124, 302
H2OSegmentModels-class, 302
H2OSegmentModelsFuture-class, 302
H2OSplitNode (H2OSplitNode-class), 303
H2OSplitNode-class, 303
H2OTargetEncoderMetrics-class
(H2OModelMetrics-class), 301
H2OTargetEncoderModel-class
(H2OModel-class), 300
H2OTree (H2OTree-class), 303
H2OTree-class, 303
H2UUnknownMetrics-class
(H2OModelMetrics-class), 301
H2UUnknownModel-class (H2OModel-class), 300
H2UWordEmbeddingMetrics-class
(H2OModelMetrics-class), 301
H2UWordEmbeddingModel-class
(H2OModel-class), 300
head.H2OFrame (h2o.head), 138
hour (h2o.hour), 141
housevotes, 304
ifelse (h2o.ifelse), 141
iris, 305
is.character, 156, 305
is.factor, 156, 306
is.h2o, 306
is.na.H2OFrame (Ops.H2OFrame), 310
is.numeric, 157, 307
Keyed-class, 307
kurtosis.H2OFrame (h2o.kurtosis), 164
length.H2OFrame (Ops.H2OFrame), 310
levels, 165
log, 168
log (Ops.H2OFrame), 310
log10, 169
log10 (Ops.H2OFrame), 310
log1pl (Ops.H2OFrame), 170
log2 (Ops.H2OFrame), 310
Logical-or, 308
logLik.H2OCoxPHModel (H2OCoxPHModel-class), 296
match, 176
match.H2OFrame (h2o.match), 175
Math.H2OFrame (Ops.H2OFrame), 310
max, 176
mean, 177
mean.H2OFrame (h2o.mean), 177
median.H2OFrame (h2o.median), 179
min, 184
ModelAccessors, 308
month (h2o.month), 187
names, 192
names.H2OFrame, 309
names<-.H2OFrame (Ops.H2OFrame), 310
ncol, 193
ncol.H2OFrame (Ops.H2OFrame), 310
nlevels, 194
nrow, 196
nrow.H2OFrame (Ops.H2OFrame), 310
Ops.H2OFrame, 310
plot.H2OModel, 311
plot.H2OTabulate, 312
predict, 62, 104
predict.H2OAutoML, 313
predict.H2OModel, 88, 114, 130, 219, 314
predict_contributions.H2OModel, 315
predict_leaf_node_assignment.H2OModel, 316
print.H2OFrame, 317
print.H2OTable, 318
prod, 209
prostate, 318
quantile, 213
quantile.H2OFrame (h2o.quantile), 212
range, 219
range.H2OFrame, 319
round, 222
round (h2o.round), 231
rowMeans, 177
scale, 319
sd, 238
sd (h2o.sd), 238
show.H2AutoEncoderMetrics-method (H2OModelMetrics-class), 301
show.H2BinomialMetrics-method (H2OModelMetrics-class), 301
show.H2ClusteringMetrics-method (H2OModelMetrics-class), 301
show.H2Connection-method (H2OConnection-class), 295
show.H2CoxPHModel-method (H2OCoxPHModel-class), 296
show.H2CoxPHModelSummary-method (H2OCoxPHModelSummary-class), 297
show.H2DimReductionMetrics-method (H2OModelMetrics-class), 301
show.H2Grid-method (H2OGrid-class), 298
show.H2OModel-method (H2OModel-class), 300
show.H2OModelMetrics-method (H2OModelMetrics-class), 301
show.H2MultinomialMetrics-method (H2OModelMetrics-class), 301
show.H2Node-method (H2ONode-class), 301
show.H2OrdinalMetrics-method (H2OModelMetrics-class), 301
show.H2RegressionMetrics-method (H2OModelMetrics-class), 301
show.H2SegmentModels-method (H2OSegmentModels-class), 302
show.H2Tree-method (H2OTree-class), 303
signif, 243
signif (h2o.signif), 243
sin, 244
skewness.H2OFrame (h2o.skewness), 244
sqrt, 246
staged_predict_proba.H2OModel, 320
str.H2OFrame, 321
sum, 254
summary, 255
summary.H2OCoxPHModel-method, 321
summary.H2OGrid-method, 322
summary.H2OModel-method, 322
Summary.H2OFrame (Ops.H2OFrame), 310
summary.H2OFrame (h2o.summary), 255
survfit.H2OCoxPHModel
(H2OCoxPHModel-class), 296
t.H2OFrame (Ops.H2OFrame), 310
table.H2OFrame (h2o.table), 257
tail.H2OFrame (h2o.head), 138
tan, 259
tanh, 260
trunc, 276
trunc (Ops.H2OFrame), 310
use.package, 24, 27, 323
var, 279
var (h2o.var), 279
vcov.H2OCoxPHModel
(H2OCoxPHModel-class), 296
walking, 324
week (h2o.week), 281
which, 283
which.max, 284
which.max.H2OFrame (h2o.which_max), 284
which.min, 285
which.min.H2OFrame (h2o.which_max), 284
year (h2o.year), 293
zzz, 324