Package ‘h2o4gpu’

March 23, 2018

Type Package
Title Interface to 'H2O4GPU'
Version 0.2.0
Description Interface to 'H2O4GPU' <https://github.com/h2oai/h2o4gpu>, a collection of 'GPU' solvers for machine learning algorithms.
License Apache License 2.0
URL https://github.com/h2oai/h2o4gpu
BugReports https://github.com/h2oai/h2o4gpu/issues
SystemRequirements Python (>= 3.6) with header files and shared library; H2O4GPU (https://github.com/h2oai/h2o4gpu)
Encoding UTF-8
LazyData true
Depends R (>= 3.1)
Imports magrittr, reticulate (>= 1.4)
RoxygenNote 6.0.1
Suggests testthat, knitr
VignetteBuilder knitr
NeedsCompilation no
Author Yuan Tang [aut, cre] (<https://orcid.org/0000-0001-5243-233X>), Navdeep Gill [aut], Erin LeDell [aut], H2O.ai [cph, fnd]
Maintainer Yuan Tang <terrytangyuan@gmail.com>
Repository CRAN
Date/Publication 2018-03-23 17:04:51 UTC
R topics documented:

- fit ........................................... 2
- fit.h2o4gpu_model ............................ 3
- h2o4gpu ....................................... 3
- h2o4gpu.elastic_net_classifier ............... 4
- h2o4gpu.elastic_net_regressor ............... 6
- h2o4gpu.gradient_boosting_classifier ......... 8
- h2o4gpu.gradient_boosting_regressor ......... 10
- h2o4gpu.kmeans ............................... 12
- h2o4gpu.pca .................................. 14
- h2o4gpu.random_forest_classifier .......... 15
- h2o4gpu.random_forest_regressor .......... 16
- h2o4gpu.truncated_svd ...................... 18
- predict.h2o4gpu_model ..................... 19
- transform.h2o4gpu_model ................... 20

Index 22

---

**fit**  
*Generic Method to Train an H2O4GPU Estimator*

**Description**

Generic Method to Train an H2O4GPU Estimator

Generic Method to Transform a Dataset using Trained H2O4GPU Estimator

**Usage**

```r
fit(object, ...)
```

```r
transform(object, ...)
```

**Arguments**

- **object**  
  The h2o4gpu model object

- **...**  
  Additional arguments (unused for now).
fit.h2o4gpu_model

Train an H2O4GPU Estimator

Description
This function builds the model using the training data specified.

Usage
## S3 method for class 'h2o4gpu_model'
fit(object, x, y = NULL, ...)

Arguments
object
The h2o4gpu model object

x
The training data where each column represents a different predictor variable to
be used in building the model.

y
A vector of numeric values to be used as response variable in building the model.
Note that if the vector is character or factor, it will be converted to numeric
column (e.g. 0, 1, 2, ...) implicitly. For unsupervised models, this argument can
be ignored or specified as NULL.

...
Additional arguments (unused for now).

Examples
## Not run:
library(h2o4gpu)

# Setup dataset
x <- iris[1:4]
y <- as.integer(iris$Species) - 1

# Train the classifier
h2o4gpu.random_forest_classifier() %>% fit(x, y)

## End(Not run)

h2o4gpu

h2o4gpu in R

Description
h2o4gpu in R
Examples

```r
## Not run:
library(h2o4gpu)

# Setup dataset
x <- iris[1:4]
y <- as.integer(iris$Species) - 1

# Initialize and train the classifier
model <- h2o4gpu.random_forest_classifier() %>% fit(x, y)

# Make predictions
predictions <- model %>% predict(x)

## End(Not run)
```

---

**h2o4gpu.elastic_net_classifier**

*Elastic Net Classifier*

**Description**

Elastic Net Classifier

**Usage**

```r
h2o4gpu.elastic_net_classifier(alpha = 1, l1_ratio = 0.5,
    fit_intercept = TRUE, normalize = FALSE, precompute = FALSE,
    max_iter = 5000L, copy_X = TRUE, tol = 0.01, warm_start = FALSE,
    positive = FALSE, random_state = NULL, selection = "cyclic",
    n_gpus = -1L, lambda_stop_early = TRUE, glm_stop_early = TRUE,
    glm_stop_early_error_fraction = 1, verbose = FALSE, n_threads = NULL,
    gpu_id = 0L, lambda_min_ratio = 1e-07, n_lambdas = 100L, n_folds = 5L,
    tol_seek_factor = 0.1, store_full_path = 0L, lambda_max = NULL,
    lambdas = NULL, double_precision = NULL, order = NULL,
    backend = "h2o4gpu")
```

**Arguments**

**alpha**

Constant that multiplies the penalty terms. Defaults to 1.0. See the notes for the exact mathematical meaning of this parameter.\( \alpha = 0 \) is equivalent to an ordinary least square, solved by the :class:`LinearRegressionSklearn` object. For numerical reasons, using \( \alpha = 0 \) with the LassoSklearn object is not advised. Given this, you should use the :class:`LinearRegressionSklearn` object.
l1_ratio The ElasticNetSklearn mixing parameter, with 0 <= l1_ratio <= 1. For l1_ratio = 0 the penalty is an L2 penalty. For l1_ratio = 1 it is an L1 penalty. For 0 < l1_ratio < 1, the penalty is a combination of L1 and L2.

fit_intercept Whether the intercept should be estimated or not. If FALSE, the data is assumed to be already centered.

normalize This parameter is ignored when fit_intercept is set to FALSE. If TRUE, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use class: h2o4gpu.preprocessing.StandardScaler before calling fit on an estimator with normalize=FALSE.

precompute Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always TRUE to preserve sparsity.

max_iter The maximum number of iterations

copy_X If TRUE, X will be copied; else, it may be overwritten.

tol The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.

warm_start When set to TRUE, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

positive When set to TRUE, forces the coefficients to be positive.

random_state The seed of the pseudo random number generator that selects a random feature to update. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random. Used when selection == 'random'.

selection If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default. This (setting to 'random') often leads to significantly faster convergence especially when tol is higher than 1e-4.

n_gpus Number of gpu’s to use in GLM solver.

lambda_stop_early Stop early when there is no more relative improvement on train or validation.

glm_stop_early Stop early when there is no more relative improvement in the primary and dual residuals for ADMM.

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

verbose Print verbose information to the console if set to > 0.

n_threads Number of threads to use in the gpu. Each thread is an independent model builder.

gpu_id ID of the GPU on which the algorithm should run.

lambda_min_ratio Minimum lambda ratio to maximum lambda, used in lambda search.

n_lambdas Number of lambdas to be used in a search.
n_folds

tol.seek.factor

store.full.path

lambda.max

lambdas

double.precision

order

backend

h2o4gpu.elastic_net_regressor

Elastic Net Regressor

Description

Elastic Net Regressor

Usage

h2o4gpu.elastic_net_regressor(alpha = 1, l1_ratio = 0.5,
fit_intercept = TRUE, normalize = FALSE, precompute = FALSE,
max_iter = 5000L, copy_X = TRUE, tol = 0.01, warm_start = FALSE,
positive = FALSE, random_state = NULL, selection = "cyclic",
n_gpus = -1L, lambda_stop_early = TRUE, glm_stop_early = TRUE,

gsm_stop_early_error_fraction = 1, verbose = FALSE, n_threads = NULL,
gpu_id = 0L, lambda_min_ratio = 1e-07, n_lambdas = 100L, n_folds = 5L,
tol.seek_factor = 0.1, store_full_path = FALSE, lambda_max = NULL,
lambdas = NULL, double_precision = NULL, order = NULL,
backend = "h2o4gpu")

Arguments

alpha

Constant that multiplies the penalty terms. Defaults to 1.0. See the notes for
the exact mathematical meaning of this parameter.alpha = 0 is equivalent to
an ordinary least square, solved by the :class:LinearRegressionSklearn object.
For numerical reasons, using alpha = 0 with the LassoSklearn object is
not advised. Given this, you should use the :class:LinearRegressionSklearn object.
l1_ratio The ElasticNetSklearn mixing parameter, with $0 \leq l1_{\text{ratio}} \leq 1$. For $l1_{\text{ratio}} = 0$ the penalty is an L2 penalty. For $l1_{\text{ratio}} = 1$ it is an L1 penalty. For $0 < l1_{\text{ratio}} < 1$, the penalty is a combination of L1 and L2.

fit_intercept Whether the intercept should be estimated or not. If FALSE, the data is assumed to be already centered.

normalize This parameter is ignored when fit_intercept is set to FALSE. If TRUE, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use :class:h2o4gpu.preprocessing.StandardScaler before calling fit on an estimator with normalize=FALSE.

precompute Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always TRUE to preserve sparsity.

max_iter The maximum number of iterations

copy_X If TRUE, X will be copied; else, it may be overwritten.

tol The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.

warm_start When set to TRUE, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

positive When set to TRUE, forces the coefficients to be positive.

random_state The seed of the pseudo random number generator that selects a random feature to update. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random. Used when selection == 'random'.

selection If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default. This (setting to 'random') often leads to significantly faster convergence especially when tol is higher than 1e-4.

n_gpus Number of gpu's to use in GLM solver.

lambda_stop_early Stop early when there is no more relative improvement on train or validation.

glm_stop_early Stop early when there is no more relative improvement in the primary and dual residuals for ADMM.

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

verbose Print verbose information to the console if set to > 0.

n_threads Number of threads to use in the gpu. Each thread is an independent model builder.

gpu_id ID of the GPU on which the algorithm should run.

lambda_min_ratio Minimum lambda ratio to maximum lambda, used in lambda search.

n_lambdas Number of lambdas to be used in a search.
h2o4gpu.gradient_boosting_classifier

Gradient Boosting Classifier

Description

Gradient Boosting Classifier

Usage

h2o4gpu.gradient_boosting_classifier(loss = "deviance", learning_rate = 0.1, n_estimators = 100L, subsample = 1, criterion = "friedman_mse", min_samples_split = 2L, min_samples_leaf = 1L, min_weight_fraction_leaf = 0, max_depth = 3L, min_impurity_decrease = 0, min_impurity_split = NULL, init = NULL, random_state = NULL, max_features = "auto", verbose = 0L, max_leaf_nodes = NULL, warm_start = FALSE, presort = "auto", colsample_bytree = 1, num_parallel_tree = 1L, tree_method = "gpu_hist", n_gpus = -1L, predictor = "gpu_predictor", backend = "h2o4gpu")

Arguments

loss loss function to be optimized. ‘deviance’ refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss ‘exponential’ gradient boosting recovers the AdaBoost algorithm.

learning_rate learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
n_estimators  The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

subsample  The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias.

criterion  The function to measure the quality of a split. Supported criteria are "friedman_mse" for the mean squared error with improvement score by Friedman, "mse" for mean squared error, and "mae" for the mean absolute error. The default value of "friedman_mse" is generally the best as it can provide a better approximation in some cases.

min_samples_split  The minimum number of samples required to split an internal node:

min_samples_leaf  The minimum number of samples required to be at a leaf node:

min_weight_fraction_leaf  The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_depth  maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

min_impurity_decrease  A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

min_impurity_split  Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

init  An estimator object that is used to compute the initial predictions. init has to provide fit and predict. If NULL it uses loss.init_estimator.

random_state  If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random.

max_features  The number of features to consider when looking for the best split:

verbose  Enable verbose output. If 1 then it prints progress and performance once in a while (the more trees the lower the frequency). If greater than 1 then it prints progress and performance for every tree.

max_leaf_nodes  Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If NULL then unlimited number of leaf nodes.

warm_start  When set to TRUE, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just erase the previous solution.

presort  Whether to presort the data to speed up the finding of best splits in fitting. Auto mode by default will use presorting on dense data and default to normal sorting on sparse data. Setting presort to true on sparse data will raise an error.
Gradient Boosting Regressor

Description

Gradient Boosting Regressor

Usage

h2o4gpu.gradient_boosting_regressor(loss = "ls", learning_rate = 0.1, n_estimators = 100L, subsample = 1L, criterion = "friedman_mse", min_samples_split = 2L, min_samples_leaf = 1L, min_weight_fraction_leaf = 0, max_depth = 3L, min_impurity_decrease = 0, min_impurity_split = NULLL, init = NULLL, random_state = NULLL, max_features = "auto", alpha = 0.9, verbose = 0L, max_leaf_nodes = NULLL, warm_start = FALSE, presort = "auto", colsample_bytree = 1L, num_parallel_tree = 1L, tree_method = "gpu_hist", n_gpus = -1L, predictor = "gpu_predictor", backend = "h2o4gpu")
Arguments

loss  loss function to be optimized. ’ls’ refers to least squares regression. ’lad’ (least absolute deviation) is a highly robust loss function solely based on order information of the input variables. ’huber’ is a combination of the two. ’quantile’ allows quantile regression (use alpha to specify the quantile).

learning_rate  learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.

n_estimators  The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

subsample  The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample < 1NP leads to a reduction of variance and an increase in bias.

criterion  The function to measure the quality of a split. Supported criteria are "friedman_mse" for the mean squared error with improvement score by Friedman, "mse" for mean squared error, and "mae" for the mean absolute error. The default value of "friedman_mse" is generally the best as it can provide a better approximation in some cases.

min_samples_split  The minimum number of samples required to split an internal node:

min_samples_leaf  The minimum number of samples required to be at a leaf node:

min_weight_fraction_leaf  The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_depth  maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

min_impurity_decrease  A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

min_impurity_split  Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

init  An estimator object that is used to compute the initial predictions. init has to provide fit and predict. If NULL it uses loss.init_estimator.

random_state  If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random.

max_features  The number of features to consider when looking for the best split:

alpha  The alpha-quantile of the huber loss function and the quantile loss function. Only if loss='huber' or loss='quantile'.

11
### h2o4gpu.kmeans

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>Enable verbose output. If 1 then it prints progress and performance once in a while (the more trees the lower the frequency). If greater than 1 then it prints progress and performance for every tree.</td>
</tr>
<tr>
<td>max_leaf_nodes</td>
<td>Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If NULL then unlimited number of leaf nodes.</td>
</tr>
<tr>
<td>warm_start</td>
<td>When set to TRUE, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just erase the previous solution.</td>
</tr>
<tr>
<td>presort</td>
<td>Whether to presort the data to speed up the finding of best splits in fitting. Auto mode by default will use presorting on dense data and default to normal sorting on sparse data. Setting presort to true on sparse data will raise an error.</td>
</tr>
<tr>
<td>(colsample_bytree)</td>
<td>Subsample ratio of columns when constructing each tree.</td>
</tr>
<tr>
<td>num_parallel_tree</td>
<td>Number of trees to grow per round</td>
</tr>
<tr>
<td>tree_method</td>
<td>The tree construction algorithm used in XGBoost Distributed and external memory version only support approximate algorithm. Choices: ‘auto’, ‘exact’, ‘approx’, ‘hist’, ‘gpu_exact’, ‘gpu_hist’ ‘auto’: Use heuristic to choose faster one. - For small to medium dataset, exact greedy will be used. - For very large dataset, approximate algorithm will be chosen. - Because old behavior is always use exact greedy in single machine, - user will get a message when approximate algorithm is chosen to notify this choice. ‘exact’: Exact greedy algorithm. ‘approx’: Approximate greedy algorithm using sketching and histogram. ‘hist’: Fast histogram optimized approximate greedy algorithm. It uses some performance improvements such as bins caching. ‘gpu_exact’: GPU implementation of exact algorithm. ‘gpu_hist’: GPU implementation of hist algorithm.</td>
</tr>
<tr>
<td>n_gpus</td>
<td>Number of gpu’s to use in GradientBoostingRegressor solver. Default is -1.</td>
</tr>
<tr>
<td>predictor</td>
<td>The type of predictor algorithm to use. Provides the same results but allows the use of GPU or CPU. - ’cpu_predictor’: Multicore CPU prediction algorithm. - ’gpu_predictor’: Prediction using GPU. Default for ’gpu_exact’ and ’gpu_hist’ tree method.</td>
</tr>
<tr>
<td>backend</td>
<td>Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as attribute for actual backend used.</td>
</tr>
</tbody>
</table>

---

### Description

K-means Clustering

### Usage

```python
h2o4gpu.kmeans(n_clusters = 8L, init = "k-means++", n_init = 1L, max_iter = 300L, tol = 1e-04, precompute_distances = "auto", verbose = 0L, random_state = NULL, copy_x = TRUE, n_jobs = 1L, algorithm = "auto", gpu_id = 0L, n_gpus = -1L, do_checks = 1L, backend = "h2o4gpu")
```
Arguments

n_clusters
The number of clusters to form as well as the number of centroids to generate.

init
Method for initialization, defaults to 'random': 'k-means++': selects initial
center for k-mean clustering in a smart way to speed up convergence.
Not supported yet - if chosen we will use SKLearn's methods. 'random': choose k
observations (rows) at random from data for the initial centers. If an ndarray
is passed, it should be of shape (n_clusters, n_features) and gives the initial
centers. Not supported yet - if chosen we will use SKLearn's methods.

n_init
Number of time the k-means algorithm will be run with different centroid seeds.
The final results will be the best output of n_init consecutive runs in terms of
inertia. Not supported yet - always runs 1.

max_iter
Maximum number of iterations of the algorithm.

tol
Relative tolerance to declare convergence.

precompute_distances
Precompute distances (faster but takes more memory). 'auto': do not precom-
pute distances if n_samples * n_clusters > 12 million. This corresponds to about
100MB overhead per job using double precision. TRUE: always precompute
distances FALSE: never precompute distances Not supported yet - always uses
auto if running h2o4gpu version.

verbose
Logger verbosity level.

random_state
random_state for RandomState. Must be convertible to 32 bit unsigned integers.

copy_x
When pre-computing distances it is more numerically accurate to center the data
first. If copy_x is TRUE, then the original data is not modified. If FALSE, the
original data is modified, and put back before the function returns, but small
numerical differences may be introduced by subtracting and then adding the
data mean. Not supported yet - always uses TRUE if running h2o4gpu version.

n_jobs
The number of jobs to use for the computation. This works by computing each
of the n_init runs in parallel. If -1 all CPUs are used. If 1 is given, no parallel
computing code is used at all, which is useful for debugging. For n_jobs below
-1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are
used. Not supported yet - CPU backend not yet implemented.

algorithm
K-means algorithm to use. The classical EM-style algorithm is "full". The
"elkan" variation is more efficient by using the triangle inequality, but currently
doesn't support sparse data. "auto" chooses "elkan" for dense data and "full" for
sparse data. Not supported yet - always uses full if running h2o4gpu version.

gpu_id
ID of the GPU on which the algorithm should run.

n_gpus
Number of GPUs on which the algorithm should run. < 0 means all possible
GPUs on the machine. 0 means no GPUs, run on CPU.

do_checks
If set to 0 GPU error check will not be performed.

backend
Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as at-
tribute for actual backend used.
Principal Component Analysis (PCA)

Description

Principal Component Analysis (PCA)

Usage

```
h2o4gpu.pca(n_components = 2L, copy = TRUE, whiten = FALSE,
svd_solver = "arpack", tol = 0, iterated_power = "auto",
random_state = NULL, verbose = FALSE, backend = "h2o4gpu",
gpu_id = NULL)
```

Arguments

- **n_components**: Desired dimensionality of output data
- **copy**: If FALSE, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.
- **whiten**: When TRUE (false by default) the components vectors are multiplied by the square root of (n_samples) and divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.
- **svd_solver**: 'auto' is selected by a default policy based on X.shape and n_components: if the input data is larger than 500x500 and the number of components to extract is lower than 80 percent of the smallest dimension of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards. 'full' runs exact full SVD calling the standard LAPACK solver via scipy.linalg.svd and select the components by postprocessing 'arpack' runs SVD truncated to n_components calling ARPACK solver via scipy.sparse.linalg.svds. It requires strictly 0 < n_components < columns. 'randomized' runs randomized SVD by the method of Halko et al.
- **tol**: Tolerance for singular values computed by svd_solver == 'arpack'.
- **iterated_power**: Number of iterations for the power method computed by svd_solver == 'randomized'.
- **random_state**: If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random. Used when svd_solver == 'arpack' or 'randomized'.
- **verbose**: Verbose or not
- **backend**: Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as attribute for actual backend used.
- **gpu_id**: ID of the GPU on which the algorithm should run. Only used by h2o4gpu backend.
h2o4gpu.random_forest_classifier

Random Forest Classifier

Description

Random Forest Classifier

Usage

h2o4gpu.random_forest_classifier(n_estimators = 10L, criterion = "gini",
max_depth = 3L, min_samples_split = 2L, min_samples_leaf = 1L,
min_weight_fraction_leaf = 0, max_features = "auto",
max_leaf_nodes = NULL, min_impurity_decrease = 0,
min_impurity_split = NULL, bootstrap = TRUE, oob_score = FALSE,
n_jobs = 1L, random_state = NULL, verbose = 0L, warm_start = FALSE,
class_weight = NULL, subsample = 1, colsample_bytree = 1,
num_parallel_tree = 1L, tree_method = "gpu_hist", n_gpus = -1L,
predictor = "gpu_predictor", backend = "h2o4gpu")

Arguments

n_estimators The number of trees in the forest.
criterion The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.
max_depth The maximum depth of the tree. If NULL, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
min_samples_split The minimum number of samples required to split an internal node:
min_samples_leaf The minimum number of samples required to be at a leaf node:
min_weight_fraction_leaf The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
max_features The number of features to consider when looking for the best split:
max_leaf_nodes Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If NULL then unlimited number of leaf nodes.
min_impurity_decrease A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
min_impurity_split Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.
activation
Whether bootstrap samples are used when building trees.
oob_score
Whether to use out-of-bag samples to estimate the R^2 on unseen data.
n_jobs
The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.
random_state
If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random.
verbose
Controls the verbosity of the tree building process.
warm_start
When set to TRUE, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.
class_weight
"balanced_subsample" or NULL, optional (default=NULL) Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.
subsample
Subsample ratio of the training instance.
colsample_bytree
Subsample ratio of columns when constructing each tree.
num_parallel_tree
Number of trees to grow per round.
tree_method
The tree construction algorithm used in XGBoost Distributed and external memory version only support approximate algorithm. Choices: ‘auto’, ‘exact’, ‘approx’, ‘hist’, ‘gpu_exact’, ‘gpu_hist’ ‘auto’: Use heuristic to choose faster one. - For small to medium dataset, exact greedy will be used. - For very large dataset, approximate algorithm will be chosen. - Because old behavior is always use exact greedy in single machine, - user will get a message when approximate algorithm is chosen to notify this choice. ‘exact’: Exact greedy algorithm. ‘approx’: Approximate greedy algorithm using sketching and histogram. ‘hist’: Fast histogram optimized approximate greedy algorithm. It uses some performance improvements such as bins caching. ‘gpu_exact’: GPU implementation of exact algorithm. ‘gpu_hist’: GPU implementation of hist algorithm.
n_gpus
Number of gpus to use in RandomForestClassifier solver. Default is -1.
predictor
The type of predictor algorithm to use. Provides the same results but allows the use of GPU or CPU. - 'cpu_predictor': Multicore CPU prediction algorithm. - 'gpu_predictor': Prediction using GPU. Default for 'gpu_exact' and 'gpu_hist' tree method.
backend
Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as attribute for actual backend used.
h2o4gpu.random_forest_regressor

Description

Random Forest Regressor

Usage

```r
h2o4gpu.random_forest_regressor(n_estimators = 10L, criterion = "mse",
max_depth = 3L, min_samples_split = 2L, min_samples_leaf = 1L,
min_weight_fraction_leaf = 0, max_features = "auto",
max_leaf_nodes = NULL, min_impurity_decrease = 0,
min_impurity_split = NULL, bootstrap = TRUE, oob_score = FALSE,
n_jobs = 1L, random_state = NULL, verbose = 0L, warm_start = FALSE,
subsample = 1, colsample_bytree = 1, num_parallel_tree = 1L,
tree_method = "gpu_hist", n_gpus = -1L, predictor = "gpu_predictor",
backend = "h2o4gpu")
```

Arguments

- **n_estimators**: The number of trees in the forest.
- **criterion**: The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.
- **max_depth**: The maximum depth of the tree. If NULL, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split**: The minimum number of samples required to split an internal node:
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node:
- **min_weight_fraction_leaf**: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features**: The number of features to consider when looking for the best split:
- **max_leaf_nodes**: Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If NULL then unlimited number of leaf nodes.
- **min_impurity_decrease**: A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **min_impurity_split**: Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.
- **bootstrap**: Whether bootstrap samples are used when building trees.
- **oob_score**: Whether to use out-of-bag samples to estimate the $R^2$ on unseen data.
- **n_jobs**: The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.
random_state If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If NULL, the random number generator is the RandomState instance used by np.random.

verbose Controls the verbosity of the tree building process.

warm_start When set to TRUE, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

subsample Subsample ratio of the training instance.

colsample_bytree Subsample ratio of columns when constructing each tree.

num_parallel_tree Number of trees to grow per round

tree_method The tree construction algorithm used in XGBoost Distributed and external memory version only support approximate algorithm. Choices: 'auto', 'exact', 'approx', 'hist', 'gpu_exact', 'gpu_hist' 'auto': Use heuristic to choose faster one. - For small to medium dataset, exact greedy will be used. - For very large dataset, approximate algorithm will be chosen. - Because old behavior is always use exact greedy in single machine, user will get a message when approximate algorithm is chosen to notify this choice. 'exact': Exact greedy algorithm. 'approx': Approximate greedy algorithm using sketching and histogram. 'hist': Fast histogram optimized approximate greedy algorithm. It uses some performance improvements such as bins caching. 'gpu_exact': GPU implementation of exact algorithm. 'gpu_hist': GPU implementation of hist algorithm.

n_gpus Number of gpu’s to use in RandomForestRegressor solver. Default is -1.

predictor The type of predictor algorithm to use. Provides the same results but allows the use of GPU or CPU. - 'cpu_predictor': Multicore CPU prediction algorithm. - 'gpu_predictor': Prediction using GPU. Default for 'gpu_exact' and 'gpu_hist' tree method.

backend Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as attribute for actual backend used.

Description

Truncated Singular Value Decomposition (TruncatedSVD)

Usage

h2o4gpu.truncated_svd(n_components = 2L, algorithm = "power", n_iter = 100L, random_state = NULL, tol = 1e-05, verbose = FALSE, backend = "h2o4gpu", n_gpus = 1L, gpu_id = 0L)
**predict.h2o4gpu_model**

**Arguments**

- **n_components**: Desired dimensionality of output data
- **algorithm**: SVD solver to use. H2O4GPU options: Either "cusolver" (similar to ARPACK) or "power" for the power method. SKlearn options: Either "arpack" for the ARPACK wrapper in SciPy (scipy.sparse.linalg.svds), or "randomized" for the randomized algorithm due to Halko (2009).
- **n_iter**: number of iterations (only relevant for power method) Should be at most 2147483647 due to INT_MAX in C++ backend.
- **random_state**: seed (NULL for auto-generated)
- **tol**: Tolerance for "power" method. Ignored by "cusolver". Should be > 0.0 to ensure convergence. Should be 0.0 to effectively ignore and only base convergence upon n_iter
- **verbose**: Verbose or not
- **backend**: Which backend to use. Options are 'auto', 'sklearn', 'h2o4gpu'. Saves as attribute for actual backend used.
- **n_gpus**: How many gpus to use. If 0, use CPU backup method. Currently SVD only uses 1 GPU, so >1 has no effect compared to 1.
- **gpu_id**: ID of the GPU on which the algorithm should run.

**predict.h2o4gpu_model**  
*Make Predictions using Trained H2O4GPU Estimator*

**Description**

This function makes predictions from new data using a trained H2O4GPU model and returns class predictions for classification and predicted values for regression.

**Usage**

```r
## S3 method for class 'h2o4gpu_model'
predict(object, x, type = "raw", ...)
```

**Arguments**

- **object**: The h2o4gpu model object
- **x**: The new data where each column represents a different predictor variable to be used in generating predictions.
- **type**: One of "raw" or "prob", indicating the type of output: predicted values or probabilities
- **...**: Additional arguments (unused for now).
Examples

```r
## Not run:
library(h2o4gpu)

# Setup dataset
x <- iris[1:4]
y <- as.integer(iris$species) - 1

# Initialize and train the classifier
model <- h2o4gpu.random_forest_classifier() %>% fit(x, y)

# Make predictions
predictions <- model %>% predict(x)

## End(Not run)
```

transform.h2o4gpu_model

Transform a Dataset using Trained H2O4GPU Estimator

Description

This function transforms the given new data using a trained H2O4GPU model.

Usage

```r
## S3 method for class 'h2o4gpu_model'
transform(object, x, ...)
```

Arguments

- `object`: The h2o4gpu model object
- `x`: The new data where each column represents a different predictor variable to be used in generating predictions.
- `...`: Additional arguments (unused for now).

Examples

```r
## Not run:
library(h2o4gpu)

# Prepare data
iris$Species <- as.integer(iris$Species) # convert to numeric data
```
# Randomly sample 80% of the rows for the training set
set.seed(1)
train_idx <- sample(1:nrow(iris), 0.8*nrow(iris))
train <- iris[train_idx,]
test <- iris[-train_idx,]

# Train a K-Means model
model_km <- h2o4gpu.kmeans(n_clusters = 3L) %>% fit(train)

# Transform test data
test_dist <- model_km %>% transform(test)

## end (not run)
Index

fit, 2
fit.h2o4gpu_model, 3

h2o4gpu, 3
h2o4gpu-package (h2o4gpu), 3
h2o4gpu.elastic_net_classifier, 4
h2o4gpu.elastic_net_regressor, 6
h2o4gpu.gradient_boosting_classifier, 8
h2o4gpu.gradient_boosting_regressor, 10
h2o4gpu.kmeans, 12
h2o4gpu.pca, 14
h2o4gpu.random_forest_classifier, 15
h2o4gpu.random_forest_regressor, 16
h2o4gpu.truncated_svd, 18

predict.h2o4gpu_model, 19
transform(fit), 2
transform.h2o4gpu_model, 20