eXtreme Gradient Boosting Training

**Description**

xgb.train is an advanced interface for training an xgboost model. The xgboostfunction provides a simpler interface.

**Usage**

xgb.train(params = list(), data, nrounds, watchlist = list(), obj = NULL,

feval = NULL, verbose = 1, print\_every\_n = 1L,

early\_stopping\_rounds = NULL, maximize = NULL, save\_period = NULL,

save\_name = "xgboost.model", xgb\_model = NULL, callbacks = list(), ...)

xgboost(data = NULL, label = NULL, missing = NA, weight = NULL,

params = list(), nrounds, verbose = 1, print\_every\_n = 1L,

early\_stopping\_rounds = NULL, maximize = NULL, save\_period = 0,

save\_name = "xgboost.model", xgb\_model = NULL, callbacks = list(), ...)

**Arguments**

|  |  |
| --- | --- |
| params | the list of parameters. The complete list of parameters is available at<http://xgboost.readthedocs.io/en/latest/parameter.html>. Below is a shorter summary:  1. General Parameters   * booster which booster to use, can begbtree or gblinear. Default: gbtree * silent 0 means printing running messages, 1 means silent mode. Default: 0   2. Booster Parameters  2.1. Parameter for Tree Booster   * eta control the learning rate: scale the contribution of each tree by a factor of 0 < eta < 1 when it is added to the current approximation. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value fornrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3 * gamma minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be. * max\_depth maximum depth of a tree. Default: 6 * min\_child\_weight minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min\_child\_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. Default: 1 * subsample subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this will prevent overfitting. It makes computation shorter (because less data to analyse). It is advised to use this parameter with eta and increase nround. Default: 1 * colsample\_bytree subsample ratio of columns when constructing each tree. Default: 1 * num\_parallel\_tree Experimental parameter. number of trees to grow per round. Useful to test Random Forest through Xgboost (set colsample\_bytree < 1, subsample < 1 and round = 1) accordingly. Default: 1 * monotone\_constraints A numerical vector consists of 1, 0 and -1 with its length equals to the number of features in the training data. 1 is increasing, -1 is decreasing and 0 is no constraint.   2.2. Parameter for Linear Booster   * lambda L2 regularization term on weights. Default: 0 * lambda\_bias L2 regularization term on bias. Default: 0 * alpha L1 regularization term on weights. (there is no L1 reg on bias because it is not important). Default: 0   3. Task Parameters   * objective specify the learning task and the corresponding learning objective, users can pass a self-defined function to it. The default objective options are below:   + reg:linear linear regression (Default).   + reg:logistic logistic regression.   + binary:logistic logistic regression for binary classification. Output probability.   + binary:logitraw logistic regression for binary classification, output score before logistic transformation.   + num\_class set the number of classes. To use only with multiclass objectives.   + multi:softmax set xgboost to do multiclass classification using the softmax objective. Class is represented by a number and should be from 0 tonum\_class - 1.   + multi:softprob same as softmax, but prediction outputs a vector of ndata \* nclass elements, which can be further reshaped to ndata, nclass matrix. The result contains predicted probabilities of each data point belonging to each class.   + rank:pairwise set xgboost to do ranking task by minimizing the pairwise loss. * base\_score the initial prediction score of all instances, global bias. Default: 0.5 * eval\_metric evaluation metrics for validation data. Users can pass a self-defined function to it. Default: metric will be assigned according to objective(rmse for regression, and error for classification, mean average precision for ranking). List is provided in detail section. |
| data | input dataset. xgb.train takes only anxgb.DMatrix as the input. xgboost, in addition, also accepts matrix, dgCMatrix, or local data file. |
| nrounds | the max number of iterations |
| watchlist | what information should be printed when verbose=1or verbose=2. Watchlist is used to specify validation set monitoring during training. For example user can specify watchlist=list(validation1=mat1, validation2=mat2) to watch the performance of each round's model on mat1 and mat2 |
| obj | customized objective function. Returns gradient and second order gradient with given prediction and dtrain. |
| feval | custimized evaluation function. Returnslist(metric='metric-name', value='metric-value') with given prediction and dtrain. |
| verbose | If 0, xgboost will stay silent. If 1, xgboost will print information of performance. If 2, xgboost will print some additional information. Setting verbose > 0automatically engages the [cb.evaluation.log](http://127.0.0.1:21878/help/library/xgboost/help/cb.evaluation.log) and[cb.print.evaluation](http://127.0.0.1:21878/help/library/xgboost/help/cb.print.evaluation) callback functions. |
| print\_every\_n | Print each n-th iteration evaluation messages whenverbose>0. Default is 1 which means all messages are printed. This parameter is passed to the[cb.print.evaluation](http://127.0.0.1:21878/help/library/xgboost/help/cb.print.evaluation) callback. |
| early\_stopping\_rounds | If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set will stop if the performance doesn't improve for k rounds. Setting this parameter engages the [cb.early.stop](http://127.0.0.1:21878/help/library/xgboost/help/cb.early.stop)callback. |
| maximize | If feval and early\_stopping\_rounds are set, then this parameter must be set as well. When it isTRUE, it means the larger the evaluation score the better. This parameter is passed to the[cb.early.stop](http://127.0.0.1:21878/help/library/xgboost/help/cb.early.stop) callback. |
| save\_period | when it is non-NULL, model is saved to disk after everysave\_period rounds, 0 means save at the end. The saving is handled by the [cb.save.model](http://127.0.0.1:21878/help/library/xgboost/help/cb.save.model) callback. |
| save\_name | the name or path for periodically saved model file. |
| xgb\_model | a previously built model to continue the training from. Could be either an object of class xgb.Booster, or its raw data, or the name of a file with a previously saved model. |
| callbacks | a list of callback functions to perform various task during boosting. See [callbacks](http://127.0.0.1:21878/help/library/xgboost/help/callbacks). Some of the callbacks are automatically created depending on the parameters' values. User can provide either existing or their own callback methods in order to customize the training process. |
| ... | other parameters to pass to params. |
| label | vector of response values. Should not be provided when data is a local data file name or an xgb.DMatrix. |
| missing | by default is set to NA, which means that NA values should be considered as 'missing' by the algorithm. Sometimes, 0 or other extreme value might be used to represent missing values. This parameter is only used when input is a dense matrix. |
| weight | a vector indicating the weight for each row of the input. |

**Details**

These are the training functions for xgboost.

The xgb.train interface supports advanced features such as watchlist, customized objective and evaluation metric functions, therefore it is more flexible than the [xgboost](http://127.0.0.1:21878/help/library/xgboost/help/xgboost) interface.

Parallelization is automatically enabled if OpenMP is present. Number of threads can also be manually specified via nthread parameter.

The evaluation metric is chosen automatically by Xgboost (according to the objective) when the eval\_metric parameter is not provided. User may set one or severaleval\_metric parameters. Note that when using a customized metric, only this single metric can be used. The folloiwing is the list of built-in metrics for which Xgboost provides optimized implementation:

* rmse root mean square error.<http://en.wikipedia.org/wiki/Root_mean_square_error>
* logloss negative log-likelihood. <http://en.wikipedia.org/wiki/Log-likelihood>
* mlogloss multiclass logloss. <http://wiki.fast.ai/index.php/Log_Loss>
* error Binary classification error rate. It is calculated as (# wrong cases) / (# all cases). By default, it uses the 0.5 threshold for predicted values to define negative and positive instances. Different threshold (e.g., 0.) could be specified as "error@0."
* merror Multiclass classification error rate. It is calculated as (# wrong cases) / (# all cases).
* auc Area under the curve.<http://en.wikipedia.org/wiki/Receiver_operating_characteristic#'Area_under_curve>for ranking evaluation.
* ndcg Normalized Discounted Cumulative Gain (for ranking task).<http://en.wikipedia.org/wiki/NDCG>

The following callbacks are automatically created when certain parameters are set:

* cb.print.evaluation is turned on when verbose > 0; and theprint\_every\_n parameter is passed to it.
* cb.evaluation.log is on when verbose > 0 and watchlist is present.
* cb.early.stop: when early\_stopping\_rounds is set.
* cb.save.model: when save\_period > 0 is set.

**Value**

An object of class xgb.Booster with the following elements:

* handle a handle (pointer) to the xgboost model in memory.
* raw a cached memory dump of the xgboost model saved as R's raw type.
* niter number of boosting iterations.
* evaluation\_log evaluation history storead as a data.table with the first column corresponding to iteration number and the rest corresponding to evaluation metrics' values. It is created by the [cb.evaluation.log](http://127.0.0.1:21878/help/library/xgboost/help/cb.evaluation.log)callback.
* call a function call.
* params parameters that were passed to the xgboost library. Note that it does not capture parameters changed by the [cb.reset.parameters](http://127.0.0.1:21878/help/library/xgboost/help/cb.reset.parameters) callback.
* callbacks callback functions that were either automatically assigned or explicitely passed.
* best\_iteration iteration number with the best evaluation metric value (only available with early stopping).
* best\_ntreelimit the ntreelimit value corresponding to the best iteration, which could further be used in predict method (only available with early stopping).
* best\_score the best evaluation metric value during early stopping. (only available with early stopping).

**See Also**

[callbacks](http://127.0.0.1:21878/help/library/xgboost/help/callbacks), [predict.xgb.Booster](http://127.0.0.1:21878/help/library/xgboost/help/predict.xgb.Booster), [xgb.cv](http://127.0.0.1:21878/help/library/xgboost/help/xgb.cv)

**Examples**

data(agaricus.train, package='xgboost')

data(agaricus.test, package='xgboost')

dtrain <- xgb.DMatrix(agaricus.train$data, label = agaricus.train$label)

dtest <- xgb.DMatrix(agaricus.test$data, label = agaricus.test$label)

watchlist <- list(eval = dtest, train = dtrain)

## A simple xgb.train example:

param <- list(max\_depth = 2, eta = 1, silent = 1, nthread = 2,

objective = "binary:logistic", eval\_metric = "auc")

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist)

## An xgb.train example where custom objective and evaluation metric are used:

logregobj <- function(preds, dtrain) {

labels <- getinfo(dtrain, "label")

preds <- 1/(1 + exp(-preds))

grad <- preds - labels

hess <- preds \* (1 - preds)

return(list(grad = grad, hess = hess))

}

evalerror <- function(preds, dtrain) {

labels <- getinfo(dtrain, "label")

err <- as.numeric(sum(labels != (preds > 0)))/length(labels)

return(list(metric = "error", value = err))

}

# These functions could be used by passing them either:

# as 'objective' and 'eval\_metric' parameters in the params list:

param <- list(max\_depth = 2, eta = 1, silent = 1, nthread = 2,

objective = logregobj, eval\_metric = evalerror)

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist)

# or through the ... arguments:

param <- list(max\_depth = 2, eta = 1, silent = 1, nthread = 2)

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist,

objective = logregobj, eval\_metric = evalerror)

# or as dedicated 'obj' and 'feval' parameters of xgb.train:

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist,

obj = logregobj, feval = evalerror)

## An xgb.train example of using variable learning rates at each iteration:

param <- list(max\_depth = 2, eta = 1, silent = 1, nthread = 2)

my\_etas <- list(eta = c(0.5, 0.1))

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist,

callbacks = list(cb.reset.parameters(my\_etas)))

## Explicit use of the cb.evaluation.log callback allows to run

## xgb.train silently but still store the evaluation results:

bst <- xgb.train(param, dtrain, nrounds = 2, watchlist,

verbose = 0, callbacks = list(cb.evaluation.log()))

print(bst$evaluation\_log)

## An 'xgboost' interface example:

bst <- xgboost(data = agaricus.train$data, label = agaricus.train$label,

max\_depth = 2, eta = 1, nthread = 2, nrounds = 2,

objective = "binary:logistic")

pred <- predict(bst, agaricus.test$data)