eXtreme Gradient Boosting Training

Description

An advanced interface for training xgboost model. Look at `xgboost` function for a simpler interface.

Usage

```
xgb.train(params = list(), data, nrounds, watchlist = list(), obj = NULL,
  feval = NULL, verbose = 1, print.every.n = 1L,
  early.stop.round = NULL, maximize = NULL, ...)
```

Arguments

`params` the list of parameters.

1. General Parameters

- `booster` which booster to use, can be `gbtree` 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 for `nrounds`: 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

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 to tonum_class.
    - multi:softprob same as softmax, but output a vector of ndata * nclass, 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	takes an xgb.DMatrix as the input.
nrounds	the max number of iterations
watchlist	what information should be printed when verbose=1 or 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. Returns list(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 information of both
print.every.n	Print every N progress messages when verbose>0. Default is 1 which means all messages are printed.
early.stop.round	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 keeps getting worse consecutively for k rounds.
maximize	If feval and early.stop.round are set, then maximize must be set as well. maximize=TRUE means the larger the evaluation score the better.
...	other parameters to pass to params.

Details

This is the training function for xgboost.

It supports advanced features such as watchlist, customized objective function (feval), therefore it is more flexible than xgboost function.

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

eval_metric parameter (not listed above) is set automatically by Xgboost but can be overriden by parameter. Below is provided the list of different metric optimized by Xgboost to help you to understand how it works inside or to use them with the watchlist parameter.

- 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
- error Binary classification error rate. It is calculated as (wrong cases) / (all cases). For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances.
- 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

Full list of parameters is available in the Wiki https://github.com/dmlc/xgboost/wiki/Parameters.

This function only accepts an xgb.DMatrix object as the input.

Examples

```
data(agaricus.train, package='xgboost')
dtrain <- xgb.DMatrix(agaricus.train$data, label = agaricus.train$label)
dtest <- dtrain
watchlist <- list(eval = dtest, train = dtrain)
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))
}
param <- list(max.depth = 2, eta = 1, silent = 1, objective=logregobj,eval_metric=evalerror)
bst <- xgb.train(param, dtrain, nthread = 2, nround = 2, watchlist)
```
