Package 'xgboost'

July 12, 2016

Type Package

Title Extreme Gradient Boosting

Version 0.4-4

Date 2016-07-12

Author Tianqi Chen <tianqi.tchen@gmail.com>, Tong He <hetong007@gmail.com>, Michael Benesty <michael@benesty.fr>

Maintainer Tong He <hetong007@gmail.com>

Description Extreme Gradient Boosting, which is an
efficient implementation of gradient boosting framework.
This package is its R interface. The package includes efficient
linear model solver and tree learning algorithms. The package can automatically
do parallel computation on a single machine which could be more than 10 times faster
than existing gradient boosting packages. It supports various
objective functions, including regression, classification and ranking. The
package is made to be extensible, so that users are also allowed to define
their own objectives easily.

License Apache License (== 2.0) | file LICENSE

URL https://github.com/dmlc/xgboost

BugReports https://github.com/dmlc/xgboost/issues

VignetteBuilder knitr

Suggests knitr, ggplot2 (>= 1.0.0), DiagrammeR (>= 0.6), Ckmeans.1d.dp (>= 3.3.1), vcd (>= 1.3)

Depends R (>= 2.10)

Imports Matrix (>= 1.1-0), methods, data.table (>= 1.9.4), magrittr (>= 1.5), stringr (>= 0.6.2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-07-12 10:55:21

R topics documented:

`agaricus.test` *Test part from Mushroom Data Set*

Description

This data set is originally from the Mushroom data set, UCI Machine Learning Repository.

Usage

```
data(agaricus.test)
```

Format

A list containing a label vector, and a dgCMatrix object with 1611 rows and 126 variables

Details

This data set includes the following fields:

- `label` the label for each record
- `data` a sparse Matrix of `dgCMatrix` class, with 126 columns.

References

https://archive.ics.uci.edu/ml/datasets/Mushroom

Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

`agaricus.train` *Training part from Mushroom Data Set*

Description

This data set is originally from the Mushroom data set, UCI Machine Learning Repository.

Usage

```
data(agaricus.train)
```

Format

A list containing a label vector, and a dgCMatrix object with 6513 rows and 127 variables

Details

This data set includes the following fields:

- `label` the label for each record
- `data` a sparse Matrix of `dgCMatrix` class, with 126 columns.

References

https://archive.ics.uci.edu/ml/datasets/Mushroom

Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

getinfo *Get information of an xgb.DMatrix object*

Description

Get information of an xgb.DMatrix object

Usage

```
getinfo(object, ...)

## S4 method for signature 'xgb.DMatrix'
getinfo(object, name)
```

Arguments

object	Object of class xgb.DMatrix
...	other parameters
name	the name of the field to get

Details

The information can be one of the following:

- label: label Xgboost learn from ;
- weight: to do a weight rescale ;
- base_margin: base margin is the base prediction Xgboost will boost from ;
- nrow: number of rows of the xgb.DMatrix.

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
labels <- getinfo(dtrain, 'label')
setinfo(dtrain, 'label', 1-labels)
labels2 <- getinfo(dtrain, 'label')
stopifnot(all(labels2 == 1-labels))
```

nrow,xgb.DMatrix-method

Number of xgb.DMatrix rows

Description

nrow return the number of rows present in the xgb.DMatrix.

Usage

```
## S4 method for signature 'xgb.DMatrix'
nrow(x)
```

Arguments

x	Object of class xgb.DMatrix

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
stopifnot(nrow(dtrain) == nrow(train$data))
```

predict,xgb.Booster-method

Predict method for eXtreme Gradient Boosting model

Description

Predicted values based on xgboost model object.

Usage

```
## S4 method for signature 'xgb.Booster'
predict(object, newdata, missing = NULL,
  outputmargin = FALSE, ntreelimit = NULL, predleaf = FALSE)
```

Arguments

object	Object of class "xgb.Boost"
newdata	takes matrix, dgCMatrix, local data file or xgb.DMatrix.
missing	Missing is only used when input is dense matrix, pick a float value that represents missing value. Sometime a data use 0 or other extreme value to represents missing values.

outputmargin	whether the prediction should be shown in the original value of sum of functions, when outputmargin=TRUE, the prediction is untransformed margin value. In logistic regression, outputmargin=T will output value before logistic transformation.
ntreelimit	limit number of trees used in prediction, this parameter is only valid for gbtree, but not for gblinear. set it to be value bigger than 0. It will use all trees by default.
predleaf	whether predict leaf index instead. If set to TRUE, the output will be a matrix object.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
pred <- predict(bst, test$data)
```

---

predict,xgb.Booster.handle-method

Predict method for eXtreme Gradient Boosting model handle

Description

Predicted values based on xgb.Booster.handle object.

Usage

```
## S4 method for signature 'xgb.Booster.handle'
predict(object, ...)
```

Arguments

object	Object of class "xgb.Boost.handle"
...	Parameters pass to predict.xgb.Booster

setinfo *Set information of an xgb.DMatrix object*

Description

Set information of an xgb.DMatrix object

Usage

```
setinfo(object, ...)

## S4 method for signature 'xgb.DMatrix'
setinfo(object, name, info)
```

Arguments

object	Object of class "xgb.DMatrix"
...	other parameters
name	the name of the field to get
info	the specific field of information to set

Details

It can be one of the following:

- label: label Xgboost learn from ;
- weight: to do a weight rescale ;
- base_margin: base margin is the base prediction Xgboost will boost from ;
- group.

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
labels <- getinfo(dtrain, 'label')
setinfo(dtrain, 'label', 1-labels)
labels2 <- getinfo(dtrain, 'label')
stopifnot(all(labels2 == 1-labels))
```

slice *Get a new DMatrix containing the specified rows of orginal xgb.DMatrix object*

Description

Get a new DMatrix containing the specified rows of orginal xgb.DMatrix object

Usage

```
slice(object, ...)

## S4 method for signature 'xgb.DMatrix'
slice(object, idxset, ...)
```

Arguments

object	Object of class "xgb.DMatrix"
...	other parameters
idxset	a integer vector of indices of rows needed

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
dsub <- slice(dtrain, 1:3)
```

xgb.cv *Cross Validation*

Description

The cross valudation function of xgboost

Usage

```
xgb.cv(params = list(), data, nrounds, nfold, label = NULL,
  missing = NULL, prediction = FALSE, showsd = TRUE, metrics = list(),
  obj = NULL, feval = NULL, stratified = TRUE, folds = NULL,
  verbose = T, print.every.n = 1L, early.stop.round = NULL,
  maximize = NULL, ...)
```

Arguments

params	the list of parameters. Commonly used ones are: • objective objective function, common ones are – reg:linear linear regression – binary:logistic logistic regression for classification • eta step size of each boosting step • max.depth maximum depth of the tree • nthread number of thread used in training, if not set, all threads are used See xgb.train for further details. See also demo/ for walkthrough example in R.
data	takes an xgb.DMatrix or Matrix as the input.
nrounds	the max number of iterations
nfold	the original dataset is randomly partitioned into nfold equal size subsamples.
label	option field, when data is Matrix
missing	Missing is only used when input is dense matrix, pick a float value that represents missing value. Sometime a data use 0 or other extreme value to represents missing values.
prediction	A logical value indicating whether to return the prediction vector.
showsd	boolean, whether show standard deviation of cross validation
metrics,	list of evaluation metrics to be used in corss validation, when it is not specified, the evaluation metric is chosen according to objective function. Possible options are: • error binary classification error rate • rmse Rooted mean square error • logloss negative log-likelihood function • auc Area under curve • merror Exact matching error, used to evaluate multi-class classification
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.
stratified	boolean whether sampling of folds should be stratified by the values of labels in data
folds	list provides a possibility of using a list of pre-defined CV folds (each element must be a vector of fold's indices). If folds are supplied, the nfold and stratified parameters would be ignored.
verbose	boolean, print the statistics during the process
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

The original sample is randomly partitioned into nfold equal size subsamples.

Of the nfold subsamples, a single subsample is retained as the validation data for testing the model, and the remaining nfold - 1 subsamples are used as training data.

The cross-validation process is then repeated nrounds times, with each of the nfold subsamples used exactly once as the validation data.

All observations are used for both training and validation.

Adapted from http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29#k-fold_cross-validation

Value

If prediction = TRUE, a list with the following elements is returned:

- dt a data.table with each mean and standard deviation stat for training set and test set
- pred an array or matrix (for multiclass classification) with predictions for each CV-fold for the model having been trained on the data in all other folds.

If prediction = FALSE, just a data.table with each mean and standard deviation stat for training set and test set is returned.

Examples

```
data(agaricus.train, package='xgboost')
dtrain <- xgb.DMatrix(agaricus.train$data, label = agaricus.train$label)
history <- xgb.cv(data = dtrain, nround=3, nthread = 2, nfold = 5, metrics=list("rmse","auc"),
                  max.depth =3, eta = 1, objective = "binary:logistic")
print(history)
```

---

xgb.DMatrix *Contruct xgb.DMatrix object*

Description

Contruct xgb.DMatrix object from dense matrix, sparse matrix or local file.

Usage

```
xgb.DMatrix(data, info = list(), missing = 0, ...)
```

Arguments

data	a matrix object, a dgCMatrix object or a character indicating the data file.
info	a list of information of the xgb.DMatrix object
missing	Missing is only used when input is dense matrix, pick a float value that represents missing value. Sometime a data use 0 or other extreme value to represents missing values.
...	other information to pass to info.

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
xgb.DMatrix.save(dtrain, 'xgb.DMatrix.data')
dtrain <- xgb.DMatrix('xgb.DMatrix.data')
```

---

xgb.DMatrix.save *Save xgb.DMatrix object to binary file*

Description

Save xgb.DMatrix object to binary file

Usage

```
xgb.DMatrix.save(DMatrix, fname)
```

Arguments

DMatrix	the DMatrix object
fname	the name of the binary file.

Examples

```
data(agaricus.train, package='xgboost')
train <- agaricus.train
dtrain <- xgb.DMatrix(train$data, label=train$label)
xgb.DMatrix.save(dtrain, 'xgb.DMatrix.data')
dtrain <- xgb.DMatrix('xgb.DMatrix.data')
```

xgb.dump *Save xgboost model to text file*

Description

Save a xgboost model to text file. Could be parsed later.

Usage

```
xgb.dump(model = NULL, fname = NULL, fmap = "", with.stats = FALSE)
```

Arguments

model	the model object.
fname	the name of the text file where to save the model text dump. If not provided or set to NULL the function will return the model as a character vector.
fmap	feature map file representing the type of feature. Detailed description could be found at https://github.com/dmlc/xgboost/wiki/Binary-Classification#dump-model. See demo/ for walkthrough example in R, and https://github.com/dmlc/xgboost/blob/master/demo/data/featmap.txt for example Format.
with.stats	whether dump statistics of splits When this option is on, the model dump comes with two additional statistics: gain is the approximate loss function gain we get in each split; cover is the sum of second order gradient in each node.

Value

if fname is not provided or set to NULL the function will return the model as a character vector. Otherwise it will return TRUE.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
# save the model in file 'xgb.model.dump'
xgb.dump(bst, 'xgb.model.dump', with.stats = TRUE)

# print the model without saving it to a file
print(xgb.dump(bst))
```

xgb.importance *Show importance of features in a model*

Description

Read a xgboost model text dump. Can be tree or linear model (text dump of linear model are only supported in dev version of Xgboost for now).

Usage

```
xgb.importance(feature_names = NULL, filename_dump = NULL, model = NULL,
  data = NULL, label = NULL, target = function(x) ((x + label) == 2))
```

Arguments

feature_names	names of each feature as a character vector. Can be extracted from a sparse matrix (see example). If model dump already contains feature names, this argument should be NULL.
filename_dump	the path to the text file storing the model. Model dump must include the gain per feature and per tree (with.stats = T in function xgb.dump).
model	generated by the xgb.train function. Avoid the creation of a dump file.
data	the dataset used for the training step. Will be used with label parameter for co-occurence computation. More information in Detail part. This parameter is optional.
label	the label vetor used for the training step. Will be used with data parameter for co-occurence computation. More information in Detail part. This parameter is optional.
target	a function which returns TRUE or 1 when an observation should be count as a co-occurence and FALSE or 0 otherwise. Default function is provided for computing co-occurences in a binary classification. The target function should have only one parameter. This parameter will be used to provide each important feature vector after having applied the split condition, therefore these vector will be only made of 0 and 1 only, whatever was the information before. More information in Detail part. This parameter is optional.

Details

This is the function to understand the model trained (and through your model, your data).

Results are returned for both linear and tree models.

data.table is returned by the function. There are 3 columns :

- Features name of the features as provided in feature_names or already present in the model dump.

- Gain contribution of each feature to the model. For boosted tree model, each gain of each feature of each tree is taken into account, then average per feature to give a vision of the entire model. Highest percentage means important feature to predict the label used for the training ;
- Cover metric of the number of observation related to this feature (only available for tree models) ;
- Weight percentage representing the relative number of times a feature have been taken into trees. Gain should be prefered to search the most important feature. For boosted linear model, this column has no meaning.

Co-occurence count ————————

The gain gives you indication about the information of how a feature is important in making a branch of a decision tree more pure. However, with this information only, you can't know if this feature has to be present or not to get a specific classification. In the example code, you may wonder if odor=none should be TRUE to not eat a mushroom.

Co-occurence computation is here to help in understanding this relation between a predictor and a specific class. It will count how many observations are returned as TRUE by the target function (see parameters). When you execute the example below, there are 92 times only over the 3140 observations of the train dataset where a mushroom have no odor and can be eaten safely.

If you need to remember one thing only: until you want to leave us early, don't eat a mushroom which has no odor :-)

Value

A data.table of the features used in the model with their average gain (and their weight for boosted tree model) in the model.

Examples

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

# Both dataset are list with two items, a sparse matrix and labels
# (labels = outcome column which will be learned).
# Each column of the sparse Matrix is a feature in one hot encoding format.
train <- agaricus.train

bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")

# train$data@Dimnames[[2]] represents the column names of the sparse matrix.
xgb.importance(train$data@Dimnames[[2]], model = bst)

# Same thing with co-occurence computation this time
xgb.importance(train$data@Dimnames[[2]], model = bst, data = train$data, label = train$label)
```

xgb.load *Load xgboost model from binary file*

Description

Load xgboost model from the binary model file

Usage

```
xgb.load(modelfile)
```

Arguments

modelfile	the name of the binary file.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
xgb.save(bst, 'xgb.model')
bst <- xgb.load('xgb.model')
pred <- predict(bst, test$data)
```

xgb.model.dt.tree *Convert tree model dump to data.table*

Description

Read a tree model text dump and return a data.table.

Usage

```
xgb.model.dt.tree(feature_names = NULL, filename_dump = NULL,
  model = NULL, text = NULL, n_first_tree = NULL)
```

Arguments

feature_names	names of each feature as a character vector. Can be extracted from a sparse matrix (see example). If model dump already contains feature names, this argument should be NULL.
filename_dump	the path to the text file storing the model. Model dump must include the gain per feature and per tree (parameter with.stats = T in function xgb.dump).

model	dump generated by the xgb.train function. Avoid the creation of a dump file.
text	dump generated by the xgb.dump function. Avoid the creation of a dump file. Model dump must include the gain per feature and per tree (parameter with.stats = T in function xgb.dump).
n_first_tree	limit the plot to the n first trees. If NULL, all trees of the model are plotted. Performance can be low for huge models.

Details

General function to convert a text dump of tree model to a Matrix. The purpose is to help user to explore the model and get a better understanding of it.

The content of the data.table is organised that way:

- ID: unique identifier of a node ;
- Feature: feature used in the tree to operate a split. When Leaf is indicated, it is the end of a branch ;
- Split: value of the chosen feature where is operated the split ;
- Yes: ID of the feature for the next node in the branch when the split condition is met ;
- No: ID of the feature for the next node in the branch when the split condition is not met ;
- Missing: ID of the feature for the next node in the branch for observation where the feature used for the split are not provided ;
- Quality: it's the gain related to the split in this specific node ;
- Cover: metric to measure the number of observation affected by the split ;
- Tree: ID of the tree. It is included in the main ID ;
- Yes.X or No.X: data related to the pointer in Yes or No column ;

Value

A data.table of the features used in the model with their gain, cover and few other thing.

Examples

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

#Both dataset are list with two items, a sparse matrix and labels
#(labels = outcome column which will be learned).
#Each column of the sparse Matrix is a feature in one hot encoding format.
train <- agaricus.train

bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")

#agaricus.test$data@Dimnames[[2]] represents the column names of the sparse matrix.
xgb.model.dt.tree(agaricus.train$data@Dimnames[[2]], model = bst)
```

xgb.plot.importance *Plot feature importance bar graph*

Description

Read a data.table containing feature importance details and plot it.

Usage

```
xgb.plot.importance(importance_matrix = NULL, numberOfClusters = c(1:10))
```

Arguments

importance_matrix	a data.table returned by the xgb.importance function.
numberOfClusters	a numeric vector containing the min and the max range of the possible number of clusters of bars.

Details

The purpose of this function is to easily represent the importance of each feature of a model. The function return a ggplot graph, therefore each of its characteristic can be overriden (to customize it). In particular you may want to override the title of the graph. To do so, add `+ ggtitle("A GRAPH NAME")` next to the value returned by this function.

Value

A ggplot2 bar graph representing each feature by a horizontal bar. Longer is the bar, more important is the feature. Features are classified by importance and clustered by importance. The group is represented through the color of the bar.

Examples

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

#Both dataset are list with two items, a sparse matrix and labels
#(labels = outcome column which will be learned).
#Each column of the sparse Matrix is a feature in one hot encoding format.
train <- agaricus.train

bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")

#train$data@Dimnames[[2]] represents the column names of the sparse matrix.
importance_matrix <- xgb.importance(train$data@Dimnames[[2]], model = bst)
xgb.plot.importance(importance_matrix)
```

xgb.plot.tree *Plot a boosted tree model*

Description

Read a tree model text dump. Plotting only works for boosted tree model (not linear model).

Usage

```
xgb.plot.tree(feature_names = NULL, filename_dump = NULL, model = NULL,
  n_first_tree = NULL, CSSstyle = NULL, width = NULL, height = NULL)
```

Arguments

feature_names	names of each feature as a character vector. Can be extracted from a sparse matrix (see example). If model dump already contains feature names, this argument should be NULL.
filename_dump	the path to the text file storing the model. Model dump must include the gain per feature and per tree (parameter with.stats = T in function xgb.dump). Possible to provide a model directly (see model argument).
model	generated by the xgb.train function. Avoid the creation of a dump file.
n_first_tree	limit the plot to the n first trees. If NULL, all trees of the model are plotted. Performance can be low for huge models.
CSSstyle	a character vector storing a css style to customize the appearance of nodes. Look at the Mermaid wiki for more information.
width	the width of the diagram in pixels.
height	the height of the diagram in pixels.

Details

The content of each node is organised that way:

- feature value ;
- cover: the sum of second order gradient of training data classified to the leaf, if it is square loss, this simply corresponds to the number of instances in that branch. Deeper in the tree a node is, lower this metric will be ;
- gain: metric the importance of the node in the model.

Each branch finishes with a leaf. For each leaf, only the cover is indicated. It uses Mermaid library for that purpose.

Value

A DiagrammeR of the model.

Examples

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

#Both dataset are list with two items, a sparse matrix and labels
#(labels = outcome column which will be learned).
#Each column of the sparse Matrix is a feature in one hot encoding format.
train <- agaricus.train

bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")

#agaricus.test$data@Dimnames[[2]] represents the column names of the sparse matrix.
xgb.plot.tree(agaricus.train$data@Dimnames[[2]], model = bst)
```

---

xgb.save *Save xgboost model to binary file*

Description

Save xgboost model from xgboost or xgb.train

Usage

```
xgb.save(model, fname)
```

Arguments

model	the model object.
fname	the name of the binary file.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
xgb.save(bst, 'xgb.model')
bst <- xgb.load('xgb.model')
pred <- predict(bst, test$data)
```

xgb.save.raw *Save xgboost model to R's raw vector, user can call xgb.load to load the model back from raw vector*

Description

Save xgboost model from xgboost or xgb.train

Usage

```
xgb.save.raw(model)
```

Arguments

model	the model object.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
raw <- xgb.save.raw(bst)
bst <- xgb.load(raw)
pred <- predict(bst, test$data)
```

xgb.train *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)
```

xgboost *eXtreme Gradient Boosting (Tree) library*

Description

A simple interface for training xgboost model. Look at xgb.train function for a more advanced interface.

Usage

```
xgboost(data = NULL, label = NULL, missing = NULL, params = list(),
  nrounds, verbose = 1, print.every.n = 1L, early.stop.round = NULL,
  maximize = NULL, ...)
```

Arguments

data	takes matrix, dgCMatrix, local data file or xgb.DMatrix.
label	the response variable. User should not set this field, if data is local data file or xgb.DMatrix.
missing	Missing is only used when input is dense matrix, pick a float value that represents missing value. Sometimes a data use 0 or other extreme value to represents missing values.
params	the list of parameters. Commonly used ones are: • objective objective function, common ones are – reg:linear linear regression – binary:logistic logistic regression for classification • eta step size of each boosting step • max.depth maximum depth of the tree • nthread number of thread used in training, if not set, all threads are used Look at xgb.train for a more complete list of parameters or https://github.com/dmlc/xgboost/wiki/Parameters for the full list. See also demo/ for walkthrough example in R.
nrounds	the max number of iterations
verbose	If 0, xgboost will stay silent. If 1, xgboost will print information of performance. If 2, xgboost will print information of both performance and construction progress information
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 modeling function for Xgboost.

Parallelization is automatically enabled if OpenMP is present.

Number of threads can also be manually specified via nthread parameter.

Examples

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
               eta = 1, nthread = 2, nround = 2, objective = "binary:logistic")
pred <- predict(bst, test$data)
```

Index