Package ‘lightgbm’

January 16, 2023

Type  Package
Title  Light Gradient Boosting Machine
Version  3.3.5
Date  2023-01-11

Description  Tree based algorithms can be improved by introducing boosting frameworks. 'LightGBM' is one such framework, based on Ke, Guolin et al. (2017) <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision>. This package offers an R interface to work with it. It is designed to be distributed and efficient with the following advantages:
1. Faster training speed and higher efficiency.
2. Lower memory usage.
4. Parallel learning supported.
5. Capable of handling large-scale data.
In recognition of these advantages, 'LightGBM' has been widely-used in many winning solutions of machine learning competitions. Comparison experiments on public datasets suggest that 'LightGBM' can outperform existing boosting frameworks on both efficiency and accuracy, with significantly lower memory consumption. In addition, parallel experiments suggest that in certain circumstances, 'LightGBM' can achieve a linear speed-up in training time by using multiple machines.

Encoding  UTF-8
License  MIT + file LICENSE
URL  https://github.com/Microsoft/LightGBM
BugReports  https://github.com/Microsoft/LightGBM/issues
NeedsCompilation  yes
Biarch  true
Suggests  testthat
Depends  R (>= 3.5), R6 (>= 2.0)
Imports  data.table (>= 1.9.6), graphics, jsonlite (>= 1.0), Matrix (>= 1.1-0), methods, utils
SystemRequirements  C++11
RoxygenNote 7.1.2

Author Yu Shi [aut, cre],
    Guolin Ke [aut],
    Damien Soukhavong [aut],
    James Lamb [aut],
    Qi Meng [aut],
    Thomas Finley [aut],
    Taifeng Wang [aut],
    Wei Chen [aut],
    Weidong Ma [aut],
    Qiwei Ye [aut],
    Tie-Yan Liu [aut],
    Nikita Titov [aut],
    Yachen Yan [ctb],
Microsoft Corporation [cph],
Dropbox, Inc. [cph],
Jay Loden [cph],
Dave Daeschler [cph],
Giampaolo Rodola [cph],
Alberto Ferreira [ctb],
Daniel Lemire [ctb],
Victor Zverovich [cph],
IBM Corporation [ctb],
David Cortes [ctb]

Maintainer Yu Shi <yushi2@microsoft.com>

Repository CRAN

Date/Publication 2023-01-16 19:00:07 UTC

R topics documented:

agaricus.test .................................................. 3
agaricus.train .................................................. 4
bank ............................................................... 4
dim.lgb.Dataset ................................................ 5
dimnames.lgb.Dataset ......................................... 6
getinfo ............................................................ 7
get_field .......................................................... 8
lgb.convert_with_rules ....................................... 9
lgb.cv ............................................................ 10
lgb.Dataset ......................................................... 13
lgb.Dataset.construct ....................................... 15
lgb.Dataset.create.valid .................................. 16
lgb.Dataset.save ............................................... 18
lgb.Dataset.set.categorical ................................. 18
lgb.Dataset.set.reference ................................ 19
lgb.dump ........................................................ 20
Description

This data set is originally from the Mushroom data set, UCI Machine Learning Repository. This data set includes the following fields:

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

Usage

data(agaricus.test)

Format

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

References

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

Description
This data set is originally from the Mushroom data set, UCI Machine Learning Repository. This data set includes the following fields:
  • label: the label for each record
  • data: a sparse Matrix of dgCMatrix class, with 126 columns.

Usage
data(agaricus.train)

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

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

Description
This data set is originally from the Bank Marketing data set, UCI Machine Learning Repository. It contains only the following: bank.csv with 10 randomly selected from 3 (older version of this dataset with less inputs).

Usage
data(bank)

Format
A data.table with 4521 rows and 17 variables

References
http://archive.ics.uci.edu/ml/datasets/Bank+Marketing
S. Moro, P. Cortez and P. Rita. (2014) A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems
## dim.lgb.Dataset

### Dimensions of an lgb.Dataset

**Description**

Returns a vector of numbers of rows and of columns in an lgb.Dataset.

**Usage**

```r
## S3 method for class 'lgb.Dataset'
dim(x, ...)
```

**Arguments**

- `x` Object of class lgb.Dataset
- `...` other parameters (ignored)

**Details**

Note: since `nrow` and `ncol` internally use `dim`, they can also be directly used with an lgb.Dataset object.

**Value**

a vector of numbers of rows and of columns

**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

stopifnot(nrow(dtrain) == nrow(train$data))
stopifnot(ncol(dtrain) == ncol(train$data))
stopifnot(all(dim(dtrain) == dim(train$data)))
```
dimnames.lgb.Dataset  Handling of column names of lgb.Dataset

Description

Only column names are supported for lgb.Dataset, thus setting of row names would have no effect and returned row names would be NULL.

Usage

```r
## S3 method for class 'lgb.Dataset'
dimnames(x)
## S3 replacement method for class 'lgb.Dataset'
dimnames(x) <- value
```

Arguments

- **x**: object of class lgb.Dataset
- **value**: a list of two elements: the first one is ignored and the second one is column names

Details

Generic dimnames methods are used by colnames. Since row names are irrelevant, it is recommended to use colnames directly.

Value

A list with the dimension names of the dataset

Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)
dimnames(dtrain)
colnames(dtrain)
colnames(dtrain) <- make.names(seq_len(ncol(train$data)))
print(dtrain, verbose = TRUE)
```
Description

Get one attribute of a `lgb.Dataset` object

Usage

```r
getinfo(dataset, ...)  # S3 method for class 'lgb.Dataset'
getinfo(dataset, name, ...)
```

Arguments

- `dataset`: Object of class `lgb.Dataset`
- `...`: other parameters (ignored)
- `name`: the name of the information field to get (see details)

Details

The `name` field can be one of the following:

- `label`: label lightgbm learn from;
- `weight`: to do a weight rescale;
- `group`: used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with `group = c(10, 20, 40, 10, 10, 10)`, that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.
- `init_score`: initial score is the base prediction lightgbm will boost from.

Value

- `info data`

Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)

labels <- lightgbm::getinfo(dtrain, "label")
lightgbm::setinfo(dtrain, "label", 1 - labels)
```
labels2 <- lightgbm::getinfo(dtrain, "label")
stopifnot(all(labels2 == 1 - labels))

---

**get_field**

*Get one attribute of a lgb.Dataset*

**Usage**

```r
get_field(dataset, field_name)
```

## S3 method for class 'lgb.Dataset'
get_field(dataset, field_name)

**Arguments**

- **dataset**:
  Object of class `lgb.Dataset`

- **field_name**:
  String with the name of the attribute to get. One of the following.
  - `label`: label lightgbm learns from;
  - `weight`: to do a weight rescale;
  - `group`: used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with `group = c(10, 20, 40, 10, 10, 10)`, that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.
  - `init_score`: initial score is the base prediction lightgbm will boost from.

**Value**

requested attribute

**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)

labels <- lightgbm::get_field(dtrain, "label")
lightgbm::set_field(dtrain, "label", 1 - labels)
```
labels2 <- lightgbm::get_field(dtrain, "label")
stopifnot(all(labels2 == 1 - labels))

---

**Description**

Attempts to prepare a clean dataset to prepare to put in a `lgb.Dataset`. Factor, character, and logical columns are converted to integer. Missing values in factors and characters will be filled with 0L. Missing values in logicals will be filled with -1L.

This function returns and optionally takes in "rules" the describe exactly how to convert values in columns.

Columns that contain only NA values will be converted by this function but will not show up in the returned rules.

NOTE: In previous releases of LightGBM, this function was called `lgb.prepare_rules2`.

**Usage**

```r
lgb.convert_with_rules(data, rules = NULL)
```

**Arguments**

- `data` A data.frame or data.table to prepare.
- `rules` A set of rules from the data preparator, if already used. This should be an R list, where names are column names in `data` and values are named character vectors whose names are column values and whose values are new values to replace them with.

**Value**

A list with the cleaned dataset (`data`) and the rules (`rules`). Note that the data must be converted to a matrix format (as.matrix) for input in `lgb.Dataset`.

**Examples**

```r
data(iris)
str(iris)

new_iris <- lgb.convert_with_rules(data = iris)
str(new_iris$data)
```

data(iris) # Erase iris dataset
iris$Species[1L] <- "NEW FACTOR" # Introduce junk factor (NA)

# Use conversion using known rules
# Unknown factors become 0, excellent for sparse datasets
newer_iris <- lgb.convert_with_rules(data = iris, rules = new_iris.rules)

# Unknown factor is now zero, perfect for sparse datasets
newer_iris$data[1L, ] # Species became 0 as it is an unknown factor

newer_iris$data[1L, 5L] <- 1.0 # Put back real initial value

# Is the newly created dataset equal? YES!
all.equal(new_iris$data, newer_iris$data)

# Can we test our own rules?
data(iris) # Erase iris dataset

# We remapped values differently
personal_rules <- list(
  Species = c(
    "setosa" = 3L,
    "versicolor" = 2L,
    "virginica" = 1L
  )
)
newest_iris <- lgb.convert_with_rules(data = iris, rules = personal_rules)
str(newest_iris$data) # SUCCESS!

---

**lgb.cv**

*Main CV logic for LightGBM*

**Description**

Cross validation logic used by LightGBM

**Usage**

```

lgb.cv(
  params = list(),
  data,
  nrounds = 100L,
  nfold = 3L,
  label = NULL,
  weight = NULL,
  obj = NULL,
  eval = NULL,
  verbose = 1L,
)

```
record = TRUE,
eval_freq = 1L,
showsd = TRUE,
stratified = TRUE,
folds = NULL,
init_model = NULL,
colnames = NULL,
categorical_feature = NULL,
early_stopping_rounds = NULL,
callbacks = list(),
reset_data = FALSE,
)

Arguments

params a list of parameters. See the "Parameters" section of the documentation for a list of parameters and valid values.
data a lgb.Dataset object, used for training. Some functions, such as lgb.cv, may allow you to pass other types of data like matrix and then separately supply label as a keyword argument.
nrounds number of training rounds
nfold the original dataset is randomly partitioned into nfold equal size subsamples.
label Vector of labels, used if data is not an lgb.Dataset
weight vector of response values. If not NULL, will set to dataset
obj objective function, can be character or custom objective function. Examples include regression, regression_l1, huber, binary, lambdarank, multiclass, multiclass
eval evaluation function(s). This can be a character vector, function, or list with a mixture of strings and functions.

- a. character vector: If you provide a character vector to this argument, it should contain strings with valid evaluation metrics. See The "metric" section of the documentation for a list of valid metrics.

- b. function: You can provide a custom evaluation function. This should accept the keyword arguments preds and dtrain and should return a named list with three elements:
  - name: A string with the name of the metric, used for printing and storing results.
  - value: A single number indicating the value of the metric for the given predictions and true values
  - higher_better: A boolean indicating whether higher values indicate a better fit. For example, this would be FALSE for metrics like MAE or RMSE.

- c. list: If a list is given, it should only contain character vectors and functions. These should follow the requirements from the descriptions above.
verbose

verbosity for output, if <= 0, also will disable the print of evaluation during training.

record

Boolean, TRUE will record iteration message to booster$record_evals.

eval_freq

evaluation output frequency, only effect when verbose > 0

showsd

boolean, whether to show standard deviation of cross validation. This parameter defaults to TRUE. Setting it to FALSE can lead to a slight speedup by avoiding unnecessary computation.

stratified

a boolean indicating whether sampling of folds should be stratified by the values of outcome labels.

folds

list provides a possibility to use a list of pre-defined CV folds (each element must be a vector of test fold’s indices). When folds are supplied, the nfold and stratified parameters are ignored.

init_model

path of model file of lgb.Booster object, will continue training from this model.

colnames

feature names, if not null, will use this to overwrite the names in dataset.

categorical_feature

categorical features. This can either be a character vector of feature names or an integer vector with the indices of the features (e.g. c(1L, 10L) to say "the first and tenth columns").

early_stopping_rounds

int. Activates early stopping. When this parameter is non-null, training will stop if the evaluation of any metric on any validation set fails to improve for early_stopping_rounds consecutive boosting rounds. If training stops early, the returned model will have attribute best_iter set to the iteration number of the best iteration.

callbacks

List of callback functions that are applied at each iteration.

reset_data

Boolean, setting it to TRUE (not the default value) will transform the booster model into a predictor model which frees up memory and the original datasets.

... other parameters, see Parameters.rst for more information. A few key parameters:

- boosting: Boosting type. "gbdt", "rf", "dart" or "goss".
- num_leaves: Maximum number of leaves in one tree.
- max_depth: Limit the max depth for tree model. This is used to deal with overfit when #data is small. Tree still grow by leaf-wise.
- num_threads: Number of threads for LightGBM. For the best speed, set this to the number of real CPU cores(parallel::detectCores(logical = FALSE)), not the number of threads (most CPU using hyper-threading to generate 2 threads per CPU core).

NOTE: As of v3.3.0, use of ... is deprecated. Add parameters to params directly.

Value

a trained model lgb.CVBooster.
Early Stopping

"early stopping" refers to stopping the training process if the model’s performance on a given validation set does not improve for several consecutive iterations.

If multiple arguments are given to eval, their order will be preserved. If you enable early stopping by setting `early_stopping_rounds` in `params`, by default all metrics will be considered for early stopping.

If you want to only consider the first metric for early stopping, pass `first_metric_only = TRUE` in `params`. Note that if you also specify `metric` in `params`, that metric will be considered the "first" one. If you omit `metric`, a default metric will be used based on your choice for the parameter `obj` (keyword argument) or `objective` (passed into `params`).

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
params <- list(
    objective = "regression",
    metric = "l2",
    min_data = 1L,
    learning_rate = 1.0
)
model <- lgb.cv(
    params = params,
    data = dtrain,
    nrounds = 5L,
    nfold = 3L
)

lgb.Dataset

Construct lgb.Dataset object

Description

Construct lgb.Dataset object from dense matrix, sparse matrix or local file (that was created previously by saving an lgb.Dataset).

Usage

lgb.Dataset(
    data,
    params = list(),
    reference = NULL,
    colnames = NULL,
    categorical_feature = NULL,
free_raw_data = TRUE,
info = list(),
label = NULL,
weight = NULL,
group = NULL,
init_score = NULL,
...)
}

Arguments

**data**
- a matrix object, a dgCMatrix object, a character representing a path to a text file (CSV, TSV, or LibSVM), or a character representing a path to a binary lgb.Dataset file

**params**
- a list of parameters. See The "Dataset Parameters" section of the documentation for a list of parameters and valid values.

**reference**
- reference dataset. When LightGBM creates a Dataset, it does some preprocessing like binning continuous features into histograms. If you want to apply the same bin boundaries from an existing dataset to new data, pass that existing Dataset to this argument.

**colnames**
- names of columns

**categorical_feature**
- categorical features. This can either be a character vector of feature names or an integer vector with the indices of the features (e.g. c(1L, 10L) to say "the first and tenth columns").

**free_raw_data**
- LightGBM constructs its data format, called a "Dataset", from tabular data. By default, that Dataset object on the R side does not keep a copy of the raw data. This reduces LightGBM's memory consumption, but it means that the Dataset object cannot be changed after it has been constructed. If you'd prefer to be able to change the Dataset object after construction, set free_raw_data = FALSE.

**info**
- a list of information of the lgb.Dataset object. NOTE: use of info is deprecated as of v3.3.0. Use keyword arguments (e.g. init_score = init_score) directly.

**label**
- vector of labels to use as the target variable

**weight**
- numeric vector of sample weights

**group**
- used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with group = c(10, 20, 40, 10, 10, 10), that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.

**init_score**
- initial score is the base prediction lightgbm will boost from

... other parameters passed to params

Value

constructed dataset
Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data_file <- tempfile(fileext = ".data")
lgb.Dataset.save(dtrain, data_file)
dtrain <- lgb.Dataset(data_file)
lgb.Dataset.construct(dtrain)
```

**Description**

Construct Dataset explicitly

**Usage**

```r
lgb.Dataset.construct(dataset)
```

**Arguments**

- **dataset**
  - Object of class `lgb.Dataset`

**Value**

constructed dataset

**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)
```
**lgb.Dataset.create.valid**

*Construct validation data*

**Description**

Construct validation data according to training data

**Usage**

```r
lgb.Dataset.create.valid(
    dataset,
    data,
    info = list(),
    label = NULL,
    weight = NULL,
    group = NULL,
    init_score = NULL,
    params = list(),
    ...
)
```

**Arguments**

- **dataset**: lgb.Dataset object, training data
- **data**: a matrix object, a dgCMatrix object, a character representing a path to a text file (CSV, TSV, or LibSVM), or a character representing a path to a binary Dataset file
- **info**: a list of information of the lgb.Dataset object. NOTE: use of info is deprecated as of v3.3.0. Use keyword arguments (e.g. `init_score = init_score`) directly.
- **label**: vector of labels to use as the target variable
- **weight**: numeric vector of sample weights
- **group**: used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with `group = c(10, 20, 40, 10, 10, 10)`, that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.
- **init_score**: initial score is the base prediction lightgbm will boost from
- **params**: a list of parameters. See The "Dataset Parameters" section of the documentation for a list of parameters and valid values. If this is an empty list (the default), the validation Dataset will have the same parameters as the Dataset passed to argument `dataset`.
- **...**: additional lgb.Dataset parameters. NOTE: As of v3.3.0, use of ... is deprecated. Add parameters to `params` directly.
**Value**

constructed dataset

**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)

# parameters can be changed between the training data and validation set,
# for example to account for training data in a text file with a header row
# and validation data in a text file without it
train_file <- tempfile(pattern = "train_", fileext = ".csv")
write.table(
  data.frame(y = rnorm(100L), x1 = rnorm(100L), x2 = rnorm(100L)),
  file = train_file,
  sep = ",",
  col.names = TRUE,
  row.names = FALSE,
  quote = FALSE
)

valid_file <- tempfile(pattern = "valid_", fileext = ".csv")
write.table(
  data.frame(y = rnorm(100L), x1 = rnorm(100L), x2 = rnorm(100L)),
  file = valid_file,
  sep = ",",
  col.names = FALSE,
  row.names = FALSE,
  quote = FALSE
)

dtrain <- lgb.Dataset(
  data = train_file,
  params = list(has_header = TRUE)
)
dtrain$construct()

dvalid <- lgb.Dataset(
  data = valid_file,
  params = list(has_header = FALSE)
)
dvalid$construct()
```
### lgb.Dataset.save

**Description**

Please note that `init_score` is not saved in binary file. If you need it, please set it again after loading Dataset.

**Usage**

```
1gb.Dataset.save(dataset, fname)
```

**Arguments**

- `dataset` object of class `lgb.Dataset`
- `fname` object filename of output file

**Value**

the dataset you passed in

**Examples**

```
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.save(dtrain, tempfile(fileext = ".bin"))
```

### lgb.Dataset.set_categorical

**Description**

Set the categorical features of an `lgb.Dataset` object. Use this function to tell LightGBM which features should be treated as categorical.

**Usage**

```
lgb.Dataset.set.categorical(dataset, categorical_feature)
```
### Arguments

- **dataset**: object of class `lgb.Dataset`
- **categorical_feature**: categorical features. This can either be a character vector of feature names or an integer vector with the indices of the features (e.g. `c(1L, 10L)` to say "the first and tenth columns").

### Value

the dataset you passed in

### Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data_file <- tempfile(fileext = ".data")
lgb.Dataset.save(dtrain, data_file)
dtrain <- lgb.Dataset(data_file)
lgb.Dataset.set.categorical(dtrain, 1L:2L)
```

---

### Description

If you want to use validation data, you should set reference to training data

### Usage

```r
lgb.Dataset.set.reference(dataset, reference)
```

### Arguments

- **dataset**: object of class `lgb.Dataset`
- **reference**: object of class `lgb.Dataset`

### Value

the dataset you passed in
Examples

```r
# create training Dataset
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

# create a validation Dataset, using dtrain as a reference
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset(test$data, label = test$label)
lgb.Dataset.set.reference(dtest, dtrain)
```

---

**lgb.dump**

*Dump LightGBM model to json*

**Description**

Dump LightGBM model to json

**Usage**

```r
lgb.dump(booster, num_iteration = NULL)
```

**Arguments**

- `booster` Object of class `lgb.Booster`
- `num_iteration` number of iteration want to predict with, NULL or <= 0 means use best iteration

**Value**

json format of model

**Examples**

```r
library(lightgbm)
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
```r
lgb.get.eval.result

valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 10L,
  valids = valids,
  early_stopping_rounds = 5L
)
json_model <- lgb.dump(model)
```

---

### lgb.get.eval.result

#### Get record evaluation result from booster

**Description**

Given a `lgb.Booster`, return evaluation results for a particular metric on a particular dataset.

**Usage**

```r
lgb.get.eval.result(
  booster,
  data_name,
  eval_name,
  iters = NULL,
  is_err = FALSE
)
```

**Arguments**

- `booster`: Object of class `lgb.Booster`
- `data_name`: Name of the dataset to return evaluation results for.
- `eval_name`: Name of the evaluation metric to return results for.
- `iters`: An integer vector of iterations you want to get evaluation results for. If `NULL` (the default), evaluation results for all iterations will be returned.
- `is_err`: TRUE will return evaluation error instead

**Value**

numeric vector of evaluation result
Examples

# train a regression model
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression"
  , metric = "l2"
  , min_data = 1L
  , learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params
  , data = dtrain
  , nrounds = 5L
  , valids = valids
)

# Examine valid data_name values
print(setdiff(names(model$record_evals), "start_iter"))

# Examine valid eval_name values for dataset "test"
print(names(model$record_evals[["test"]]))

# Get L2 values for "test" dataset
lgb.get.eval.result(model, "test", "l2")

---

**lgb.importance**

*Compute feature importance in a model*

**Description**

Creates a data.table of feature importances in a model.

**Usage**

```r
lgb.importance(model, percentage = TRUE)
```

**Arguments**

- `model` : object of class `lgb.Booster`
- `percentage` : whether to show importance in relative percentage.
**Value**

For a tree model, a data table with the following columns:

- **Feature**: Feature names in the model.
- **Gain**: The total gain of this feature’s splits.
- **Cover**: The number of observation related to this feature.
- **Frequency**: The number of times a feature splited in trees.

**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

params <- list(
  objective = "binary",
  learning_rate = 0.1,
  max_depth = -1L,
  min_data_in_leaf = 1L,
  min_sum_hessian_in_leaf = 1.0
)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L
)

tree_imp1 <- lgb.importance(model, percentage = TRUE)
tree_imp2 <- lgb.importance(model, percentage = FALSE)
```

---

**lgb.interprete**

*Compute feature contribution of prediction*

**Description**

Computes feature contribution components of rawscore prediction.

**Usage**

`lgb.interprete(model, data, idxset, num_iteration = NULL)`

**Arguments**

- **model**: object of class `lgb.Booster`.
- **data**: a matrix object or a dgCMatrix object.
- **idxset**: an integer vector of indices of rows needed.
- **num_iteration**: number of iteration want to predict with, NULL or <= 0 means use best iteration.
Value

For regression, binary classification and lambdaRank model, a list of data.table with the following columns:

- Feature: Feature names in the model.
- Contribution: The total contribution of this feature’s splits.

For multiclass classification, a list of data.table with the Feature column and Contribution columns to each class.

Examples

```r
Logit <- function(x) log(x / (1.0 - x))
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
set_field(
  dataset = dtrain
  , field_name = "init_score"
  , data = rep(Logit(mean(train$label)), length(train$label))
)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test

params <- list(
  objective = "binary"
  , learning_rate = 0.1
  , max_depth = -1L
  , min_data_in_leaf = 1L
  , min_sum_hessian_in_leaf = 1.0
)
model <- lgb.train(
  params = params
  , data = dtrain
  , nrounds = 3L
)

tree_interpretation <- lgb.interpret(model, test$data, 1L:5L)
```

---

**lgb.load**

*Load LightGBM model*

Description

Load LightGBM takes in either a file path or model string. If both are provided, Load will default to loading from file.
Usage

```r
lgb.load(filename = NULL, model_str = NULL)
```

Arguments

- `filename` path of model file
- `model_str` a str containing the model

Value

`lgb.Booster`

Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L,
  valids = valids,
  early_stopping_rounds = 3L
)
model_file <- tempfile(fileext = ".txt")
lgb.save(model, model_file)
load_booster <- lgb.load(filename = model_file)
model_string <- model$save_model_to_string(NULL) # saves best iteration
load_booster_from_str <- lgb.load(model_str = model_string)
```

---

**Description**

Parse a LightGBM model json dump into a `data.table` structure.
Usage

\texttt{lgb.model.dt.tree(model, num\_iteration = NULL)}

Arguments

- \texttt{model}: object of class \texttt{lgb.Booster}
- \texttt{num\_iteration}: number of iterations you want to predict with. NULL or <= 0 means use best iteration

Value

A \texttt{data.table} with detailed information about model trees’ nodes and leaves.

The columns of the \texttt{data.table} are:

- \texttt{tree\_index}: ID of a tree in a model (integer)
- \texttt{split\_index}: ID of a node in a tree (integer)
- \texttt{split\_feature}: for a node, it’s a feature name (character); for a leaf, it simply labels it as "NA"
- \texttt{node\_parent}: ID of the parent node for current node (integer)
- \texttt{leaf\_index}: ID of a leaf in a tree (integer)
- \texttt{leaf\_parent}: ID of the parent node for current leaf (integer)
- \texttt{split\_gain}: Split gain of a node
- \texttt{threshold}: Splitting threshold value of a node
- \texttt{decision\_type}: Decision type of a node
- \texttt{default\_left}: Determine how to handle NA value, TRUE -> Left, FALSE -> Right
- \texttt{internal\_value}: Node value
- \texttt{internal\_count}: The number of observation collected by a node
- \texttt{leaf\_value}: Leaf value
- \texttt{leaf\_count}: The number of observation collected by a leaf

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

params <- list(
  objective = "binary",
  learning_rate = 0.01,
  num_leaves = 63L,
  max_depth = -1L,
  min_data_in_leaf = 1L,
  min_sum_hessian_in_leaf = 1.0
)

lgb.plot.importance

```r
model <- lgb.train(params, dtrain, 10L)

tree_dt <- lgb.model.dt.tree(model)
```

---

**lgb.plot.importance**  
*Plot feature importance as a bar graph*

**Description**

Plot previously calculated feature importance: Gain, Cover and Frequency, as a bar graph.

**Usage**

```r
lgb.plot.importance(
  tree_imp,
  top_n = 10L,
  measure = "Gain",
  left_margin = 10L,
  cex = NULL
)
```

**Arguments**

- `tree_imp`: a data.table returned by `lgb.importance`.
- `top_n`: maximal number of top features to include into the plot.
- `measure`: the name of importance measure to plot, can be "Gain", "Cover" or "Frequency".
- `left_margin`: (base R `barplot`) allows to adjust the left margin size to fit feature names.
- `cex`: (base R `barplot`) passed as `cex.names` parameter to `barplot`. Set a number smaller than 1.0 to make the bar labels smaller than R’s default and values greater than 1.0 to make them larger.

**Details**

The graph represents each feature as a horizontal bar of length proportional to the defined importance of a feature. Features are shown ranked in a decreasing importance order.

**Value**

The `lgb.plot.importance` function creates a `barplot` and silently returns a processed data.table with `top_n` features sorted by defined importance.
**Examples**

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

params <- list(
    objective = "binary",
    learning_rate = 0.1,
    min_data_in_leaf = 1L,
    min_sum_hessian_in_leaf = 1.0
)

model <- lgb.train(
    params = params,
    data = dtrain,
    nrounds = 5L
)

tree_imp <- lgb.importance(model, percentage = TRUE)
lgb.plot.importance(tree_imp, top_n = 5L, measure = "Gain")
```

---

**lgb.plot.interpretation**

*Plot feature contribution as a bar graph*

**Description**

Plot previously calculated feature contribution as a bar graph.

**Usage**

```r
lgb.plot.interpretation(
    tree_interpretation_dt,
    top_n = 10L,
    cols = 1L,
    left_margin = 10L,
    cex = NULL
)
```

**Arguments**

- `tree_interpretation_dt`  
  a data.table returned by `lgb.interprete`.
- `top_n`  
  maximal number of top features to include into the plot.
- `cols`  
  the column numbers of layout, will be used only for multiclass classification feature contribution.
left_margin  (base R barplot) allows to adjust the left margin size to fit feature names.
cex         (base R barplot) passed as cex.names parameter to barplot.

Details

The graph represents each feature as a horizontal bar of length proportional to the defined contribution of a feature. Features are shown ranked in a decreasing contribution order.

Value

The `lgb.plot.interpretation` function creates a barplot.

Examples

```r
Logit <- function(x) {
  log(x / (1.0 - x))
}
data(agaricus.train, package = "lightgbm")
labels <- agaricus.train$label
dtrain <- lgb.Dataset(
  agaricus.train$data,
  label = labels
)
set_field(
  dataset = dtrain,
  field_name = "init_score",
  data = rep(Logit(mean(labels)), length(labels))
)

data(agaricus.test, package = "lightgbm")

params <- list(
  objective = "binary",
  learning_rate = 0.1,
  max_depth = -1L,
  min_data_in_leaf = 1L,
  min_sum_hessian_in_leaf = 1.0
)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L
)

tree_interpretation <- lgb.interpret(
  model = model,
  data = agaricus.test$data,
  idxset = 1L:5L
)

lgb.plot.interpretation(
  tree_interpretation_dt = tree_interpretation[[1L]]
)```
lgb.save

Save LightGBM model

Description

Save LightGBM model

Usage

lgb.save(booster, filename, num_iteration = NULL)

Arguments

booster Object of class lgb.Booster
filename saved filename
num_iteration number of iteration want to predict with, NULL or <= 0 means use best iteration

Value

lgb.Booster

Examples

library(lightgbm)
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 10L,
  valids = valids,
  early_stopping_rounds = 5L
)
lgb.train

lgb.save(model, tempfile(fileext = ".txt"))

---

**lgb.train**  

Main training logic for LightGBM

---

**Description**

Logic to train with LightGBM

**Usage**

```r
lgb.train(
  params = list(),
  data,
  nrounds = 100L,
  valids = list(),
  obj = NULL,
  eval = NULL,
  verbose = 1L,
  record = TRUE,
  eval_freq = 1L,
  init_model = NULL,
  colnames = NULL,
  categorical_feature = NULL,
  early_stopping_rounds = NULL,
  callbacks = list(),
  reset_data = FALSE,
  ...
)
```

**Arguments**

- **params**: a list of parameters. See the "Parameters" section of the documentation for a list of parameters and valid values.
- **data**: a `lgb.Dataset` object, used for training. Some functions, such as `lgb.cv`, may allow you to pass other types of data like `matrix` and then separately supply `label` as a keyword argument.
- **nrounds**: number of training rounds
- **valids**: a list of `lgb.Dataset` objects, used for validation
- **obj**: objective function, can be character or custom objective function. Examples include `regression`, `regression_l1`, `huber`, `binary`, `lambdarank`, `multiclass`, `multiclass`
- **eval**: evaluation function(s). This can be a character vector, function, or list with a mixture of strings and functions.
- **a. character vector**: If you provide a character vector to this argument, it should contain strings with valid evaluation metrics. See The "metric" section of the documentation for a list of valid metrics.

- **b. function**: You can provide a custom evaluation function. This should accept the keyword arguments `preds` and `dtrain` and should return a named list with three elements:
  - `name`: A string with the name of the metric, used for printing and storing results.
  - `value`: A single number indicating the value of the metric for the given predictions and true values
  - `higher_better`: A boolean indicating whether higher values indicate a better fit. For example, this would be `FALSE` for metrics like MAE or RMSE.

- **c. list**: If a list is given, it should only contain character vectors and functions. These should follow the requirements from the descriptions above.

- **verbose**
  verbosity for output, if <= 0, also will disable the print of evaluation during training

- **record**
  Boolean, TRUE will record iteration message to booster$record_evals

- **eval_freq**
  evaluation output frequency, only effect when verbose > 0

- **init_model**
  path of model file of lgb.Booster object, will continue training from this model

- **colnames**
  feature names, if not null, will use this to overwrite the names in dataset

- **categorical_feature**
  categorical features. This can either be a character vector of feature names or an integer vector with the indices of the features (e.g. `c(1L, 10L)` to say "the first and tenth columns").

- **early_stopping_rounds**
  int. Activates early stopping. When this parameter is non-null, training will stop if the evaluation of any metric on any validation set fails to improve for `early_stopping_rounds` consecutive boosting rounds. If training stops early, the returned model will have attribute `best_iter` set to the iteration number of the best iteration.

- **callbacks**
  List of callback functions that are applied at each iteration.

- **reset_data**
  Boolean, setting it to TRUE (not the default value) will transform the booster model into a predictor model which frees up memory and the original datasets

... other parameters, see the "Parameters" section of the documentation for more information. A few key parameters:

- **boosting**: Boosting type. "gbdt", "rf", "dart" or "goss".
- **num_leaves**: Maximum number of leaves in one tree.
- **max_depth**: Limit the max depth for tree model. This is used to deal with overfitting. Tree still grow by leaf-wise.
- **num_threads**: Number of threads for LightGBM. For the best speed, set this to the number of real CPU cores(parallel::detectCores(logical = FALSE)), not the number of threads (most CPU using hyper-threading to generate 2 threads per CPU core).

**NOTE**: As of v3.3.0, use of ... is deprecated. Add parameters to `params` directly.
Value

a trained booster model `lgb.Booster`.

Early Stopping

"early stopping" refers to stopping the training process if the model’s performance on a given validation set does not improve for several consecutive iterations.

If multiple arguments are given to `eval`, their order will be preserved. If you enable early stopping by setting `early_stopping_rounds` in `params`, by default all metrics will be considered for early stopping.

If you want to only consider the first metric for early stopping, pass `first_metric_only = TRUE` in `params`. Note that if you also specify `metric` in `params`, that metric will be considered the "first" one. If you omit `metric`, a default metric will be used based on your choice for the parameter `obj` (keyword argument) or `objective` (passed into `params`).

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L,
  valids = valids,
  early_stopping_rounds = 3L
)

lgb.unloader

Remove lightgbm and its objects from an environment

Description

Attempts to unload LightGBM packages so you can remove objects cleanly without having to restart R. This is useful for instance if an object becomes stuck for no apparent reason and you do not want to restart R to fix the lost object.
Usage

```r
lgb.unloader(restore = TRUE, wipe = FALSE, envir = .GlobalEnv)
```

Arguments

- `restore`: Whether to reload LightGBM immediately after detaching from R. Defaults to TRUE which means automatically reload LightGBM once unloading is performed.
- `wipe`: Whether to wipe all `lgb.Dataset` and `lgb.Booster` from the global environment. Defaults to FALSE which means to not remove them.
- `envir`: The environment to perform wiping on if `wipe` == TRUE. Defaults to `.GlobalEnv` which is the global environment.

Value

NULL invisibly.

Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L,
  valids = valids
)

lgb.unloader(restore = FALSE, wipe = FALSE, envir = .GlobalEnv)
rm(model, dtrain, dtest) # Not needed if wipe == TRUE
gc() # Not needed if wipe == TRUE

library(lightgbm)
# Do whatever you want again with LightGBM without object clashing
```
Train a LightGBM model

Description
Simple interface for training a LightGBM model.

Usage
```
lightgbm(
    data,
    label = NULL,
    weight = NULL,
    params = list(),
    nrounds = 100L,
    verbose = 1L,
    eval_freq = 1L,
    early_stopping_rounds = NULL,
    save_name = "lightgbm.model",
    init_model = NULL,
    callbacks = list(),
    ...
)
```

Arguments
- **data**: a `lgb.Dataset` object, used for training. Some functions, such as `lgb.cv`, may allow you to pass other types of data like `matrix` and then separately supply `label` as a keyword argument.
- **label**: Vector of labels, used if data is not an `lgb.Dataset`.
- **weight**: vector of response values. If not NULL, will set to dataset.
- **params**: a list of parameters. See the "Parameters" section of the documentation for a list of parameters and valid values.
- **nrounds**: number of training rounds
- **verbose**: verbosity for output, if <= 0, also will disable the print of evaluation during training
- **eval_freq**: evaluation output frequency, only effect when verbose > 0
- **early_stopping_rounds**: int. Activates early stopping. When this parameter is non-null, training will stop if the evaluation of any metric on any validation set fails to improve for early_stopping_rounds consecutive boosting rounds. If training stops early, the returned model will have attribute `best_iter` set to the iteration number of the best iteration.
- **save_name**: File name to use when writing the trained model to disk. Should end in ".model".
init_model  path of model file of `lgb.Booster` object, will continue training from this model
callbacks  List of callback functions that are applied at each iteration.
...  Additional arguments passed to `lgb.train`. For example

- **valids**: a list of `lgb.Dataset` objects, used for validation
- **obj**: objective function, can be character or custom objective function. Examples include `regression`, `regression_l1`, `huber`, `binary`, `lambdarank`, `multiclass`, `multiclass`
- **eval**: evaluation function, can be (a list of) character or custom eval function
- **record**: Boolean, TRUE will record iteration message to `booster$record_evals`
- **colnames**: feature names, if not null, will use this to overwrite the names in dataset
- **categorical_feature**: categorical features. This can either be a character vector of feature names or an integer vector with the indices of the features (e.g. `c(1L, 10L)` to say "the first and tenth columns").
- **reset_data**: Boolean, setting it to TRUE (not the default value) will transform the booster model into a predictor model which frees up memory and the original datasets
- **boosting**: Boosting type. "gbdt", "rf", "dart" or "goss".
- **num_leaves**: Maximum number of leaves in one tree.
- **max_depth**: Limit the max depth for tree model. This is used to deal with overfit when #data is small. Tree still grow by leaf-wise.
- **num_threads**: Number of threads for LightGBM. For the best speed, set this to the number of real CPU cores(`parallel::detectCores(logical = FALSE)`), not the number of threads (most CPU using hyper-threading to generate 2 threads per CPU core).

**Value**

a trained `lgb.Booster`

**Early Stopping**

"early stopping" refers to stopping the training process if the model’s performance on a given validation set does not improve for several consecutive iterations.

If multiple arguments are given to `eval`, their order will be preserved. If you enable early stopping by setting `early_stopping_rounds` in `params`, by default all metrics will be considered for early stopping.

If you want to only consider the first metric for early stopping, pass `first_metric_only = TRUE` in `params`. Note that if you also specify `metric` in `params`, that metric will be considered the "first" one. If you omit `metric`, a default metric will be used based on your choice for the parameter `obj` (keyword argument) or `objective` (passed into `params`).
predict.lgb.Booster  

Predict method for LightGBM model

Description

Predicted values based on class lgb.Booster

Usage

```r
## S3 method for class 'lgb.Booster'
predict(
  object,
  data,
  start_iteration = NULL,
  num_iteration = NULL,
  rawscore = FALSE,
  predleaf = FALSE,
  predcontrib = FALSE,
  header = FALSE,
  reshape = FALSE,
  params = list(),
  ...
)
```

Arguments

- **object**: Object of class lgb.Booster
- **data**: a matrix object, a dgCMatrix object or a character representing a path to a text file (CSV, TSV, or LibSVM)
- **start_iteration**: int or None, optional (default=None) Start index of the iteration to predict. If None or <= 0, starts from the first iteration.
- **num_iteration**: int or None, optional (default=None) Limit number of iterations in the prediction. If None, if the best iteration exists and start_iteration is None or <= 0, the best iteration is used; otherwise, all iterations from start_iteration are used. If <= 0, all iterations from start_iteration are used (no limits).
- **rawscore**: whether the prediction should be returned in the for of original untransformed sum of predictions from boosting iterations’ results. E.g., setting rawscore=TRUE for logistic regression would result in predictions for log-odds instead of probabilities.
- **predleaf**: whether predict leaf index instead.
- **predcontrib**: return per-feature contributions for each record.
- **header**: only used for prediction for text file. True if text file has header
- **reshape**: whether to reshape the vector of predictions to a matrix form when there are several prediction outputs per case.
predict.lgb.Booster

params

- a list of additional named parameters. See the "Predict Parameters" section of the documentation for a list of parameters and valid values.

... Additional prediction parameters. NOTE: deprecated as of v3.3.0. Use `params` instead.

Value

For regression or binary classification, it returns a vector of length `nrows(data)`. For multiclass classification, either a `num_class * nrows(data)` vector or a `(nrows(data), num_class)` dimension matrix is returned, depending on the `reshape` value.

When `predleaf = TRUE`, the output is a matrix object with the number of columns corresponding to the number of trees.

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 5L,
  valids = valids
)
preds <- predict(model, test$data)

# pass other prediction parameters
preds <- predict(
  model,
  test$data,
  params = list(
    predict_disable_shape_check = TRUE
  )
)
**Description**

Attempts to load a model stored in a .rds file, using `readRDS`.

**Usage**

```r
readRDS.lgb.Booster(file, refhook = NULL)
```

**Arguments**

- `file` a connection or the name of the file where the R object is saved to or read from.
- `refhook` a hook function for handling reference objects.

**Value**

`lgb.Booster`

**Examples**

```r
library(lightgbm)
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 10L,
  valids = valids,
  early_stopping_rounds = 5L
)
model_file <- tempfile(fileext = ".rds")
saveRDS.lgb.Booster(model, model_file)
new_model <- readRDS.lgb.Booster(model_file)
```
Description

Attempts to save a model using RDS. Has an additional parameter (raw) which decides whether to save the raw model or not.

Usage

```r
saveRDS.lgb.Booster(
  object,  
  file,  
  ascii = FALSE,  
  version = NULL,  
  compress = TRUE,  
  refhook = NULL,  
  raw = TRUE
)
```

Arguments

- **object**: lgb.Booster object to serialize.
- **file**: a connection or the name of the file where the R object is saved to or read from.
- **ascii**: a logical. If TRUE or NA, an ASCII representation is written; otherwise (default), a binary one is used. See the comments in the help for save.
- **version**: the workspace format version to use. NULL specifies the current default version (2). Versions prior to 2 are not supported, so this will only be relevant when there are later versions.
- **compress**: a logical specifying whether saving to a named file is to use "gzip" compression, or one of "gzip", "bzip2" or "xz" to indicate the type of compression to be used. Ignored if file is a connection.
- **refhook**: a hook function for handling reference objects.
- **raw**: whether to save the model in a raw variable or not, recommended to leave it to TRUE.

Value

NULL invisibly.

Examples

```r
library(lightgbm)
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
```
dtrain <- lgb.Dataset(train$data, label = train$label)
data(agaricus.test, package = "lightgbm")
test <- agaricus.test
dtest <- lgb.Dataset.create.valid(dtrain, test$data, label = test$label)
params <- list(
  objective = "regression",
  metric = "l2",
  min_data = 1L,
  learning_rate = 1.0
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params,
  data = dtrain,
  nrounds = 10L,
  valids = valids,
  early_stopping_rounds = 5L
)
model_file <- tempfile(fileext = ".rds")
saveRDS.lgb.Booster(model, model_file)

---

**setinfo**

*Set information of an lgb.Dataset object*

**Description**

Set one attribute of a lgb.Dataset

**Usage**

```r
setinfo(dataset, ...)```

## S3 method for class 'lgb.Dataset'

```r
setinfo(dataset, name, info, ...)```

**Arguments**

- **dataset**: Object of class lgb.Dataset
- **...**: other parameters (ignored)
- **name**: the name of the field to get
- **info**: the specific field of information to set

**Details**

The name field can be one of the following:

- **label**: vector of labels to use as the target variable
• weight: to do a weight rescale
• init_score: initial score is the base prediction lightgbm will boost from
• group: used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with group = c(10, 20, 40, 10, 10, 10), that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.

Value
the dataset you passed in

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)
labels <- lightgbm::getinfo(dtrain, "label")
lightgbm::setinfo(dtrain, "label", 1 - labels)
labels2 <- lightgbm::getinfo(dtrain, "label")
stopifnot(all.equal(labels2, 1 - labels))

set_field

Description
Set one attribute of a lgb.Dataset object

Usage
set_field(dataset, field_name, data)

## S3 method for class 'lgb.Dataset'
set_field(dataset, field_name, data)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Object of class lgb.Dataset</td>
</tr>
<tr>
<td>field_name</td>
<td>String with the name of the attribute to set. One of the following.</td>
</tr>
<tr>
<td></td>
<td>• label: label lightgbm learns from ;</td>
</tr>
<tr>
<td></td>
<td>• weight: to do a weight rescale ;</td>
</tr>
</tbody>
</table>
- **group**: used for learning-to-rank tasks. An integer vector describing how to group rows together as ordered results from the same set of candidate results to be ranked. For example, if you have a 100-document dataset with `group = c(10, 20, 40, 10, 10, 10)`, that means that you have 6 groups, where the first 10 records are in the first group, records 11-30 are in the second group, etc.

- **init_score**: initial score is the base prediction LightGBM will boost from.

### Value

The `lgb.Dataset` you passed in.

### Examples

```r
data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)
lgb.Dataset.construct(dtrain)

labels <- lightgbm::get_field(dtrain, "label")
lightgbm::set_field(dtrain, "label", 1 - labels)

labels2 <- lightgbm::get_field(dtrain, "label")
stopifnot(all.equal(labels2, 1 - labels))
```

---

**slice**

*Slice a dataset*

**Description**

Get a new `lgb.Dataset` containing the specified rows of original `lgb.Dataset` object.

**Usage**

```r
slice(dataset, ...)
```

---

**Arguments**

- `dataset`: Object of class `lgb.Dataset`
- `...`: other parameters (currently not used)
- `idxset`: an integer vector of indices of rows needed
Value

constructed sub dataset

Examples

data(agaricus.train, package = "lightgbm")
train <- agaricus.train
dtrain <- lgb.Dataset(train$data, label = train$label)

dsub <- lightgbm::slice(dtrain, seq_len(42L))
lgb.Dataset.construct(dsub)
labels <- lightgbm::get_field(dsub, "label")
Index

* datasets
  agaricus.test, 3
  agaricus.train, 4
  bank, 4

  agaricus.test, 3
  agaricus.train, 4

  bank, 4
  barplot, 27

  dim.lgb.Dataset, 5
  dimnames.lgb.Dataset, 6
  dimnames<-.lgb.Dataset
    (dimnames.lgb.Dataset), 6

  get_field, 8
  getinfo, 7

  lgb.convert_with_rules, 9
  lgb.cv, 10, 11, 31, 35
  lgb.Dataset, 11, 13, 35
  lgb.Dataset.construct, 15
  lgb.Dataset.create.valid, 16
  lgb.Dataset.save, 18
  lgb.Dataset.set.categorical, 18
  lgb.Dataset.set.reference, 19
  lgb.dump, 20
  lgb.get.eval.result, 21
  lgb.importance, 22, 27
  lgb.interprete, 23, 28
  lgb.load, 24
  lgb.model.dt.tree, 25
  lgb.plot.importance, 27
  lgb.plot.interpretation, 28
  lgb.save, 30
  lgb.train, 31, 36
  lgb.unloader, 33
  lightgbm, 35

  predict.lgb.Booster, 37

readRDS, 39
readRDS.lgb.Booster, 39
saveRDS.lgb.Booster, 40
set_field, 42
setinfo, 41
slice, 43