Package 'lightgbm'

January 18, 2024

Type Package

Title Light Gradient Boosting Machine

Version 4.3.0

Date 2024-01-17

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.
3. Better accuracy.
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

VignetteBuilder knitr

Suggests knitr, markdown, RhpcBLASctl, testthat

Depends R (>= 3.5)

Imports R6 (>= 2.0), data.table (>= 1.9.6), graphics, jsonlite (>= 1.0), Matrix (>= 1.1-0), methods, parallel, utils

SystemRequirements C++17

RoxygenNote 7.2.3

Author Yu Shi [aut],
Guolin Ke [aut],
Damien Soukhavong [aut],
James Lamb [aut, cre],
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],
Alberto Ferreira [ctb],
Daniel Lemire [ctb],
Victor Zverovich [cph],
IBM Corporation [ctb],
David Cortes [aut],
Michael Mayer [ctb]

Maintainer James Lamb <jaylamb20@gmail.com>

Repository CRAN

Date/Publication 2024-01-18 11:50:09 UTC

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. 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

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. 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

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.

bank *Bank Marketing Data Set*

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

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

Arguments

`x`	Object of class `lgb.Dataset`

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

```
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

```
## 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

```
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)
```

getLGBMThreads *Get default number of threads used by LightGBM*

Description

LightGBM attempts to speed up many operations by using multi-threading. The number of threads used in those operations can be controlled via the `num_threads` parameter passed through `params` to functions like lgb.train and lgb.Dataset. However, some operations (like materializing a model from a text file) are done via code paths that don't explicitly accept thread-control configuration.

Use this function to see the default number of threads LightGBM will use for such operations.

Usage

```
getLGBMthreads()
```

Value

number of threads as an integer. `-1` means that in situations where parameter `num_threads` is not explicitly supplied, LightGBM will choose a number of threads to use automatically.

See Also

setLGBMthreads

get_field *Get one attribute of a* `lgb.Dataset`

Description

Get one attribute of a `lgb.Dataset`

Usage

```
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

```
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))
```

---

lgb.configure_fast_predict

Configure Fast Single-Row Predictions

Description

Pre-configures a LightGBM model object to produce fast single-row predictions for a given input data type, prediction type, and parameters.

Usage

```
lgb.configure_fast_predict(
  model,
  csr = FALSE,
  start_iteration = NULL,
  num_iteration = NULL,
  type = "response",
  params = list()
)
```

Arguments

`model`	LighGBM model object (class `lgb.Booster`). **The object will be modified in-place**.
`csr`	Whether the prediction function is going to be called on sparse CSR inputs. If `FALSE`, will be assumed that predictions are going to be called on single-row regular R matrices.
`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).
`type`	Type of prediction to output. Allowed types are: • `"response"`: will output the predicted score according to the objective function being optimized (depending on the link function that the objective uses), after applying any necessary transformations - for example, for `objective="binary"`, it will output class probabilities. • `"class"`: for classification objectives, will output the class with the highest predicted probability. For other objectives, will output the same as "response". Note that `"class"` is not a supported type for lgb.configure_fast_predict (see the documentation of that function for more details). • `"raw"`: will output the non-transformed numbers (sum of predictions from boosting iterations' results) from which the "response" number is produced for a given objective function - for example, for `objective="binary"`, this corresponds to log-odds. For many objectives such as "regression", since no transformation is applied, the output will be the same as for "response". • `"leaf"`: will output the index of the terminal node / leaf at which each observations falls in each tree in the model, outputted as integers, with one column per tree. • `"contrib"`: will return the per-feature contributions for each prediction, including an intercept (each feature will produce one column). Note that, if using custom objectives, types "class" and "response" will not be available and will default towards using "raw" instead. If the model was fit through function lightgbm and it was passed a factor as labels, passing the prediction type through `params` instead of through this argument might result in factor levels for classification objectives not being applied correctly to the resulting output. *New in version 4.0.0*
`params`	a list of additional named parameters. See the "Predict Parameters" section of the documentation for a list of parameters and valid values. Where these conflict with the values of keyword arguments to this function, the values in `params` take precedence.

Details

Calling this function multiple times with different parameters might not override the previous configuration and might trigger undefined behavior.

Any saved configuration for fast predictions might be lost after making a single-row prediction of a different type than what was configured (except for types "response" and "class", which can be switched between each other at any time without losing the configuration).

In some situations, setting a fast prediction configuration for one type of prediction might cause the prediction function to keep using that configuration for single-row predictions even if the requested type of prediction is different from what was configured.

Note that this function will not accept argument `type="class"` - for such cases, one can pass `type="response"` to this function and then `type="class"` to the `predict` function - the fast configuration will not be lost or altered if the switch is between "response" and "class".

The configuration does not survive de-serializations, so it has to be generated anew in every R process that is going to use it (e.g. if loading a model object through `readRDS`, whatever configuration was there previously will be lost).

Requesting a different prediction type or passing parameters to predict.lgb.Booster will cause it to ignore the fast-predict configuration and take the slow route instead (but be aware that an existing configuration might not always be overriden by supplying different parameters or prediction type, so make sure to check that the output is what was expected when a prediction is to be made on a single row for something different than what is configured).

Note that, if configuring a non-default prediction type (such as leaf indices), then that type must also be passed in the call to predict.lgb.Booster in order for it to use the configuration. This also applies for `start_iteration` and `num_iteration`, but **the** `params` **list must be empty** in the call to `predict`.

Predictions about feature contributions do not allow a fast route for CSR inputs, and as such, this function will produce an error if passing `csr=TRUE` and `type = "contrib"` together.

Value

The same `model` that was passed as input, invisibly, with the desired configuration stored inside it and available to be used in future calls to predict.lgb.Booster.

Examples

```
library(lightgbm)
data(mtcars)
X <- as.matrix(mtcars[, -1L])
y <- mtcars[, 1L]
dtrain <- lgb.Dataset(X, label = y, params = list(max_bin = 5L))
params <- list(
  min_data_in_leaf = 2L
  , num_threads = 2L
)
model <- lgb.train(
```

```
  params = params
 , data = dtrain
 , obj = "regression"
 , nrounds = 5L
 , verbose = -1L
)
lgb.configure_fast_predict(model)

x_single <- X[11L, , drop = FALSE]
predict(model, x_single)

# Will not use it if the prediction to be made
# is different from what was configured
predict(model, x_single, type = "leaf")
```

`lgb.convert_with_rules` *Data preparator for LightGBM datasets with rules (integer)*

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

```
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

```
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,
  serializable = TRUE,
  eval_train_metric = 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 and valids has been provided, also will disable the printing of evaluation during training

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

eval_freq — evaluation output frequency, only effective when verbose > 0 and valids has been provided

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 or 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

serializable — whether to make the resulting objects serializable through functions such as save or saveRDS (see section "Model serialization").

eval_train_metric — boolean, whether to add the cross validation results on the training data. This parameter defaults to FALSE. Setting it to TRUE will increase run time.

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
  , num_threads = 2L
)
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,
  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.
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

Value

constructed dataset

Examples

```
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)
```

`lgb.Dataset.construct` *Construct Dataset explicitly*

Description

Construct Dataset explicitly

Usage

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

Arguments

`dataset`	Object of class `lgb.Dataset`

Value

constructed dataset

Examples

```
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

```
lgb.Dataset.create.valid(
  dataset,
  data,
  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
`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`.

Value

constructed dataset

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)

# 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` *Save* `lgb.Dataset` *to a binary file*

Description

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

Usage

```
lgb.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` *Set categorical feature of* `lgb.Dataset`

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

```
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)
```

---

lgb.Dataset.set.reference

Set reference of `lgb.Dataset`

Description

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

Usage

```
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

```
# 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.drop_serialized *Drop serialized raw bytes in a LightGBM model object*

Description

If a LightGBM model object was produced with argument 'serializable=TRUE', the R object will keep a copy of the underlying C++ object as raw bytes, which can be used to reconstruct such object after getting serialized and de-serialized, but at the cost of extra memory usage. If these raw bytes are not needed anymore, they can be dropped through this function in order to save memory. Note that the object will be modified in-place.

New in version 4.0.0

Usage

```
lgb.drop_serialized(model)
```

Arguments

model	lgb.Booster object which was produced with 'serializable=TRUE'.

Value

lgb.Booster (the same 'model' object that was passed as input, as invisible).

See Also

lgb.restore_handle, lgb.make_serializable.

---

lgb.dump *Dump LightGBM model to json*

Description

Dump LightGBM model to json

Usage

```
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

```
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
  , num_threads = 2L
)
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

```
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
  , num_threads = 2L
)
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

```
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

```
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
  , num_threads = 2L
)
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

```
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
    , num_threads = 2L
)
model <- lgb.train(
    params = params
    , data = dtrain
    , nrounds = 3L
)

tree_interpretation <- lgb.interprete(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

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

Arguments

filename	path of model file
model_str	a str containing the model (as a character or raw vector)

Value

lgb.Booster

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
  , num_threads = 2L
)
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)
```

---

lgb.make_serializable *Make a LightGBM object serializable by keeping raw bytes*

Description

If a LightGBM model object was produced with argument 'serializable=FALSE', the R object will not be serializable (e.g. cannot save and load with `saveRDS` and `readRDS`) as it will lack the raw bytes needed to reconstruct its underlying C++ object. This function can be used to forcibly produce those serialized raw bytes and make the object serializable. Note that the object will be modified in-place.

New in version 4.0.0

Usage

```
lgb.make_serializable(model)
```

Arguments

model	lgb.Booster object which was produced with 'serializable=FALSE'.

Value

lgb.Booster (the same 'model' object that was passed as input, as invisible).

See Also

lgb.restore_handle, lgb.drop_serialized.

---

lgb.model.dt.tree *Parse a LightGBM model json dump*

Description

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

Usage

```
lgb.model.dt.tree(model, num_iteration = NULL)
```

Arguments

model	object of class lgb.Booster
num_iteration	number of iterations you want to predict with. NULL or <= 0 means use best iteration

Value

A data.table with detailed information about model trees' nodes and leafs.

The columns of the data.table are:

- tree_index: ID of a tree in a model (integer)
- split_index: ID of a node in a tree (integer)
- split_feature: for a node, it's a feature name (character); for a leaf, it simply labels it as "NA"
- node_parent: ID of the parent node for current node (integer)
- leaf_index: ID of a leaf in a tree (integer)
- leaf_parent: ID of the parent node for current leaf (integer)
- split_gain: Split gain of a node

- `threshold`: Splitting threshold value of a node
- `decision_type`: Decision type of a node
- `default_left`: Determine how to handle NA value, TRUE -> Left, FALSE -> Right
- `internal_value`: Node value
- `internal_count`: The number of observation collected by a node
- `leaf_value`: Leaf value
- `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
  , num_threads = 2L
)
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

```
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

```
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
    , num_threads = 2L
)

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

```
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

```
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
  , num_threads = 2L
)
model <- lgb.train(
  params = params
  , data = dtrain
  , nrounds = 5L
)

tree_interpretation <- lgb.interprete(
  model = model
  , data = agaricus.test$data
  , idxset = 1L:5L
)
lgb.plot.interpretation(
  tree_interpretation_dt = tree_interpretation[[1L]]
  , top_n = 3L
)
```

`lgb.restore_handle` *Restore the C++ component of a de-serialized LightGBM model*

Description

After a LightGBM model object is de-serialized through functions such as `save` or `saveRDS`, its underlying C++ object will be blank and needs to be restored to able to use it. Such object is restored automatically when calling functions such as `predict`, but this function can be used to forcibly restore it beforehand. Note that the object will be modified in-place.

New in version 4.0.0

Usage

```
lgb.restore_handle(model)
```

Arguments

model	lgb.Booster object which was de-serialized and whose underlying C++ object and R handle need to be restored.

Details

Be aware that fast single-row prediction configurations are not restored through this function. If you wish to make fast single-row predictions using a lgb.Booster loaded this way, call lgb.configure_fast_predict on the loaded lgb.Booster object.

Value

lgb.Booster (the same 'model' object that was passed as input, invisibly).

See Also

lgb.make_serializable, lgb.drop_serialized.

Examples

```
library(lightgbm)


data("agaricus.train")
model <- lightgbm(
  agaricus.train$data
  , agaricus.train$label
  , params = list(objective = "binary")
  , nrounds = 5L
  , verbose = 0
  , num_threads = 2L
)
fname <- tempfile(fileext="rds")
saveRDS(model, fname)

model_new <- readRDS(fname)
model_new$check_null_handle()
lgb.restore_handle(model_new)
model_new$check_null_handle()
```

---

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
  , num_threads = 2L
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params
  , data = dtrain
  , nrounds = 10L
  , valids = valids
  , early_stopping_rounds = 5L
)
lgb.save(model, tempfile(fileext = ".txt"))
```

---

lgb.train *Main training logic for LightGBM*

Description

Low-level R interface to train a LightGBM model. Unlike `lightgbm`, this function is focused on performance (e.g. speed, memory efficiency). It is also less likely to have breaking API changes in new releases than `lightgbm`.

Usage

```
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,
  serializable = TRUE
)
```

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 and `valids` has been provided, also will disable the printing of evaluation during training

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

`eval_freq` evaluation output frequency, only effective when verbose > 0 and `valids` has been provided

`init_model` path of model file or `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

`serializable` whether to make the resulting objects serializable through functions such as `save` or `saveRDS` (see section "Model serialization").

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
  , num_threads = 2L
)
valids <- list(test = dtest)
model <- lgb.train(
  params = params
  , data = dtrain
  , nrounds = 5L
  , valids = valids
  , early_stopping_rounds = 3L
)
```

`lightgbm` *Train a LightGBM model*

Description

High-level R interface to train a LightGBM model. Unlike `lgb.train`, this function is focused on compatibility with other statistics and machine learning interfaces in R. This focus on compatibility means that this interface may experience more frequent breaking API changes than `lgb.train`. For efficiency-sensitive applications, or for applications where breaking API changes across releases is very expensive, use `lgb.train`.

Usage

```
lightgbm(
  data,
  label = NULL,
  weights = NULL,
  params = list(),
  nrounds = 100L,
  verbose = 1L,
  eval_freq = 1L,
```

```
  early_stopping_rounds = NULL,
  init_model = NULL,
  callbacks = list(),
  serializable = TRUE,
  objective = "auto",
  init_score = NULL,
  num_threads = NULL,
  ...
)
```

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
weights	Sample / observation weights for rows in the input data. If NULL, will assume that all observations / rows have the same importance / weight. *Changed from 'weight', in version 4.0.0*
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 and valids has been provided, also will disable the printing of evaluation during training
eval_freq	evaluation output frequency, only effective when verbose > 0 and valids has been provided
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.
init_model	path of model file or lgb.Booster object, will continue training from this model
callbacks	List of callback functions that are applied at each iteration.
serializable	whether to make the resulting objects serializable through functions such as save or saveRDS (see section "Model serialization").
objective	Optimization objective (e.g. '"regression"', '"binary"', etc.). For a list of accepted objectives, see the "objective" item of the "Parameters" section of the documentation. If passing "auto" and data is not of type lgb.Dataset, the objective will be determined according to what is passed for label: • If passing a factor with two variables, will use objective "binary". • If passing a factor with more than two variables, will use objective "multiclass" (note that parameter num_class in this case will also be determined automatically from label).

- Otherwise (or if passing `lgb.Dataset` as input), will use objective `"regression"`.

New in version 4.0.0

`init_score` initial score is the base prediction lightgbm will boost from

New in version 4.0.0

`num_threads` Number of parallel threads to use. For best speed, this should be set to the number of physical cores in the CPU - in a typical x86-64 machine, this corresponds to half the number of maximum threads.

Be aware that using too many threads can result in speed degradation in smaller datasets (see the parameters documentation for more details).

If passing zero, will use the default number of threads configured for OpenMP (typically controlled through an environment variable `OMP_NUM_THREADS`).

If passing `NULL` (the default), will try to use the number of physical cores in the system, but be aware that getting the number of cores detected correctly requires package `RhpcBLASctl` to be installed.

This parameter gets overriden by `num_threads` and its aliases under `params` if passed there.

New in version 4.0.0

`...` 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

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`

New in version 4.0.0

Usage

```
## S3 method for class 'lgb.Booster'
predict(
  object,
  newdata,
  type = "response",
  start_iteration = NULL,
  num_iteration = NULL,
  header = FALSE,
  params = list(),
  ...
)
```

Arguments

`object`	Object of class `lgb.Booster`
`newdata`	a `matrix` object, a `dgCMatrix`, a `dgRMatrix` object, a `dsparseVector` object, or a character representing a path to a text file (CSV, TSV, or LibSVM). For sparse inputs, if predictions are only going to be made for a single row, it will be faster to use CSR format, in which case the data may be passed as either a single-row CSR matrix (class `dgRMatrix` from package `Matrix`) or as a sparse numeric vector (class `dsparseVector` from package `Matrix`). If single-row predictions are going to be performed frequently, it is recommended to pre-configure the model object for fast single-row sparse predictions through function lgb.configure_fast_predict. *Changed from 'data', in version 4.0.0*
`type`	Type of prediction to output. Allowed types are: • `"response"`: will output the predicted score according to the objective function being optimized (depending on the link function that the objective uses), after applying any necessary transformations - for example, for `objective="binary"`, it will output class probabilities.

- `"class"`: for classification objectives, will output the class with the highest predicted probability. For other objectives, will output the same as "response". Note that `"class"` is not a supported type for lgb.configure_fast_predict (see the documentation of that function for more details).
- `"raw"`: will output the non-transformed numbers (sum of predictions from boosting iterations' results) from which the "response" number is produced for a given objective function - for example, for `objective="binary"`, this corresponds to log-odds. For many objectives such as "regression", since no transformation is applied, the output will be the same as for "response".
- `"leaf"`: will output the index of the terminal node / leaf at which each observations falls in each tree in the model, outputted as integers, with one column per tree.
- `"contrib"`: will return the per-feature contributions for each prediction, including an intercept (each feature will produce one column).

Note that, if using custom objectives, types "class" and "response" will not be available and will default towards using "raw" instead.

If the model was fit through function lightgbm and it was passed a factor as labels, passing the prediction type through `params` instead of through this argument might result in factor levels for classification objectives not being applied correctly to the resulting output.

New in version 4.0.0

`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).

`header`
: only used for prediction for text file. True if text file has header

`params`
: a list of additional named parameters. See the "Predict Parameters" section of the documentation for a list of parameters and valid values. Where these conflict with the values of keyword arguments to this function, the values in `params` take precedence.

`...`
: ignored

Details

If the model object has been configured for fast single-row predictions through lgb.configure_fast_predict, this function will use the prediction parameters that were configured for it - as such, extra prediction parameters should not be passed here, otherwise the configuration will be ignored and the slow route will be taken.

Value

For prediction types that are meant to always return one output per observation (e.g. when predicting `type="response"` or `type="raw"` on a binary classification or regression objective), will return a vector with one element per row in `newdata`.

For prediction types that are meant to return more than one output per observation (e.g. when predicting `type="response"` or `type="raw"` on a multi-class objective, or when predicting `type="leaf"`, regardless of objective), will return a matrix with one row per observation in `newdata` and one column per output.

For `type="leaf"` predictions, will return a matrix with one row per observation in `newdata` and one column per tree. Note that for multiclass objectives, LightGBM trains one tree per class at each boosting iteration. That means that, for example, for a multiclass model with 3 classes, the leaf predictions for the first class can be found in columns 1, 4, 7, 10, etc.

For `type="contrib"`, will return a matrix of SHAP values with one row per observation in `newdata` and columns corresponding to features. For regression, ranking, cross-entropy, and binary classification objectives, this matrix contains one column per feature plus a final column containing the Shapley base value. For multiclass objectives, this matrix will represent `num_classes` such matrices, in the order "feature contributions for first class, feature contributions for second class, feature contributions for third class, etc.".

If the model was fit through function lightgbm and it was passed a factor as labels, predictions returned from this function will retain the factor levels (either as values for `type="class"`, or as column names for `type="response"` and `type="raw"` for multi-class objectives). Note that passing the requested prediction type under `params` instead of through `type` might result in the factor levels not being present in the output.

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
  , num_threads = 2L
)
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
    )
)
```

`print.lgb.Booster` *Print method for LightGBM model*

Description

Show summary information about a LightGBM model object (same as `summary`).

New in version 4.0.0

Usage

```
## S3 method for class 'lgb.Booster'
print(x, ...)
```

Arguments

`x`	Object of class `lgb.Booster`
`...`	Not used

Value

The same input `x`, returned as invisible.

`setLGBMThreads` *Set maximum number of threads used by LightGBM*

Description

LightGBM attempts to speed up many operations by using multi-threading. The number of threads used in those operations can be controlled via the `num_threads` parameter passed through `params` to functions like lgb.train and lgb.Dataset. However, some operations (like materializing a model from a text file) are done via code paths that don't explicitly accept thread-control configuration.

Use this function to set the maximum number of threads LightGBM will use for such operations.

This function affects all LightGBM operations in the same process.

So, for example, if you call `setLGBMthreads(4)`, no other multi-threaded LightGBM operation in the same process will use more than 4 threads.

Call `setLGBMthreads(-1)` to remove this limitation.

Usage

```
setLGBMthreads(num_threads)
```

Arguments

num_threads	maximum number of threads to be used by LightGBM in multi-threaded operations

See Also

getLGBMthreads

---

set_field *Set one attribute of a* lgb.Dataset *object*

Description

Set one attribute of a lgb.Dataset

Usage

```
set_field(dataset, field_name, data)

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

Arguments

dataset	Object of class lgb.Dataset
field_name	String with the name of the attribute to set. 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.
data	The data for the field. See examples.

Value

The lgb.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::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

```
slice(dataset, idxset)

## S3 method for class 'lgb.Dataset'
slice(dataset, idxset)
```

Arguments

`dataset`	Object of class `lgb.Dataset`
`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")
```

---

summary.lgb.Booster *Summary method for LightGBM model*

Description

Show summary information about a LightGBM model object (same as print).

New in version 4.0.0

Usage

```
## S3 method for class 'lgb.Booster'
summary(object, ...)
```

Arguments

object	Object of class lgb.Booster
...	Not used

Value

The same input object, returned as invisible.

Index