

Package: catboost (via r-universe)

October 11, 2026

Title Gradient Boosting on Decision Trees

Version 1.2.10-1

Description Open-source gradient boosting on decision trees with categorical features support out of the box. A fork of the upstream CatBoost R package packaged for CRAN-style installation; the R API is a strict superset of upstream's catboost.* interface.

License Apache License (== 2.0)

Imports jsonlite, parallel

Suggests caret, testthat, tibble, e1071, Matrix

SystemRequirements CMake (>= 3.15), C++20, clang and clang++ compiler, Python3, Cython, Perl, GNU Make or Ninja

URL <https://github.com/davdittrich/catboost>

BugReports <https://github.com/davdittrich/catboost/issues>

Biarch FALSE

Encoding UTF-8

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make perl python3

Repository <https://davdittrich.r-universe.dev>

Date/Publication 2026-08-12 07:53:29 UTC

RemoteUrl <https://github.com/davdittrich/catboost>

RemoteRef HEAD

RemoteSha 15b2b989c22e680dc5de975741d43312d2e45e2e

Contents

catboost.calc_feature_statistics	4
catboost.calc_leaf_indexes	6
catboost.caret	7

catboost.compare	7
catboost.compute_training_options	9
catboost.create_metric_calcer	10
catboost.cv	11
catboost.dataset_statistics	12
catboost.Dictionary	13
catboost.dictionary.apply	15
catboost.dictionary.end_of_sentence_token_id	16
catboost.dictionary.fit	16
catboost.dictionary.get_token	17
catboost.dictionary.get_tokens	17
catboost.dictionary.get_top_tokens	18
catboost.dictionary.load	18
catboost.dictionary.min_unused_token_id	19
catboost.dictionary.save	19
catboost.dictionary.size	20
catboost.dictionary.unknown_token_id	21
catboost.drop_unused_features	21
catboost.eval_feature	22
catboost.eval_metric	24
catboost.eval_metrics	25
catboost.features_data.get_cat_feature_count	26
catboost.features_data.get_feature_count	27
catboost.features_data.get_feature_names	27
catboost.features_data.get_num_feature_count	28
catboost.features_data.get_object_count	28
catboost.FeaturesData	29
catboost.get_best_iteration	30
catboost.get_best_score	30
catboost.get_borders	31
catboost.get_evals_result	31
catboost.get_feature_importance	32
catboost.get_leaf_values	33
catboost.get_leaf_weights	34
catboost.get_metadata	34
catboost.get_model_feature_names	35
catboost.get_model_params	35
catboost.get_object_importance	36
catboost.get_plain_params	37
catboost.get_probability_threshold	38
catboost.get_roc_curve	38
catboost.get_scale_and_bias	39
catboost.get_test_eval	40
catboost.get_test_evals	40
catboost.get_tree_leaf_counts	41
catboost.grid_search	41
catboost.iterate_leaf_indexes	44
catboost.load_model	44

catboost.load_pool	45
catboost.model_based_eval	48
catboost.normalize_model_from_pool	50
catboost.plot_partial_dependence	51
catboost.plot_predictions	52
catboost.plot_tree	53
catboost.pool.get_baseline	54
catboost.pool.get_cat_feature_indices	55
catboost.pool.get_embedding_feature_indices	55
catboost.pool.get_feature_names	56
catboost.pool.get_features	56
catboost.pool.get_group_id_hash	57
catboost.pool.get_label	57
catboost.pool.get_text_feature_indices	58
catboost.pool.get_weight	58
catboost.pool.has_label	59
catboost.pool.is_empty	59
catboost.pool.is_quantized	60
catboost.pool.num_col	60
catboost.pool.num_pairs	61
catboost.pool.num_row	61
catboost.pool.quantize	62
catboost.pool.save	62
catboost.pool.save_quantization_borders	63
catboost.pool.set_baseline	64
catboost.pool.set_feature_names	64
catboost.pool.set_group_id	65
catboost.pool.set_group_weight	65
catboost.pool.set_pairs	66
catboost.pool.set_pairs_weight	66
catboost.pool.set_subgroup_id	67
catboost.pool.set_timestamp	67
catboost.pool.set_weight	68
catboost.pool.shape	68
catboost.pool.slice	69
catboost.pool.train_eval_split	69
catboost.predict	70
catboost.predict_log_proba	73
catboost.randomized_search	74
catboost.restore_handle	76
catboost.run_worker	77
catboost.sample_gaussian_process	78
catboost.save_borders	80
catboost.save_model	81
catboost.save_pool	82
catboost.select_features	83
catboost.set_leaf_values	85
catboost.set_metadata	86

catboost.set_probability_threshold	86
catboost.set_scale_and_bias	87
catboost.shrink	88
catboost.staged_predict	88
catboost.staged_predict_log_proba	90
catboost.sum_models	91
catboost.Tokenizer	92
catboost.tokenizer.tokenize	93
catboost.train	94
catboost.virtual_ensembles_predict	118
dim.catboost.Pool	119
dimnames.catboost.Pool	120
head.catboost.Pool	120
print.catboost.Model	121
print.catboost.Pool	121
summary.catboost.Model	122
tail.catboost.Pool	122

Index 124

catboost.calc_feature_statistics

Calculate feature statistics.

Description

Get statistics for a feature using the model, dataset and target (see https://catboost.ai/docs/concepts/python-reference_catboost_calc_feature_statistics.html). The catboost model has borders for the float features used in it. The borders divide feature values into bins, and the model's prediction depends on the number of the bin where the feature value falls in.

For float features this function takes the model's borders and computes 1) Mean target value for every bin; 2) Mean model prediction for every bin; 3) The number of objects in the dataset which fall into each bin; 4) Predictions on varying feature: for every object, varies the feature value so that it falls into bin #0, bin #1, ... and counts model predictions, then averages that over each bin.

For categorical features (one-hot encoded only – pass `one_hot_max_size` at training time to keep a feature one-hot) does the same, but with the feature's unique values (from `pool`, or `cat_feature_values`) taking the role of bins.

Usage

```
catboost.calc_feature_statistics(
    model,
    pool,
    feature = NULL,
    prediction_type = NULL,
```

```

    cat_feature_values = NULL,
    thread_count = -1
)

```

Arguments

model	The model obtained as the result of training. Default value: Required argument
pool	A catboost.Pool. Provides both the dataset to compute statistics on and (via catboost.pool.get_label) the target. Default value: Required argument
feature	NULL, a single feature name (character scalar) or 0-based feature index (numeric scalar), or a vector/list of names/indices. NULL computes statistics for every feature in pool . A single name/index returns that feature's statistics list directly; anything else returns a named list of per-feature statistics lists. Default value: NULL
prediction_type	Prediction type used for mean_prediction : one of 'Class', 'Probability', 'RawFormulaVal' or 'Exponent'. If NULL, derived from the model's loss function ('Probability' for CrossEntropy/Logloss, 'RawFormulaVal' otherwise). Default value: NULL
cat_feature_values	A named list (feature name -> vector of values) of the categorical feature values to compute statistics on. When feature names/selects a single categorical feature, a plain (non-list) vector is also accepted for that feature. If NULL, the feature's unique values already present in pool are used. Default value: NULL
thread_count	The number of threads to use for getting statistics. If -1, then the number of threads is set to the number of CPU cores. Default value: -1

Value

A list (single feature) or a named list of lists (multiple features). For a float feature, the list has **borders**, **binarized_feature**, **mean_target**, **mean_weighted_target**, **mean_prediction**, **objects_per_bin**, **predictions_on_varying_feature**. A one-hot categorical feature has the same, but **cat_values** instead of **borders**.

See Also

https://catboost.ai/docs/concepts/python-reference_catboost_calc_feature_statistics.html

```
catboost.calc_leaf_indexes
```

Calculate the leaf index each object falls into, per tree

Description

R equivalent of Python's `model.calc_leaf_indexes(data, ntree_start, ntree_end, thread_count, verbose)` (`core.py:3157-3195`).

Usage

```
catboost.calc_leaf_indexes(  
    model,  
    pool,  
    ntree_start = 0,  
    ntree_end = 0,  
    thread_count = -1,  
    verbose = FALSE  
)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>pool</code>	The input data on which to calculate leaf indexes. Should be a <code>catboost.Pool</code> object.
<code>ntree_start</code>	Index of the first tree for which leaf indexes will be calculated (zero-based indexing). Default value: 0
<code>ntree_end</code>	Index of the tree after the last tree for which leaf indexes will be calculated (zero-based indexing). If 0, resolves to <code>catboost.ntrees(model)</code> . Default value: 0
<code>thread_count</code>	The number of threads to use. If -1, then the number of threads is set to the number of CPU cores. Default value: -1
<code>verbose</code>	If TRUE, writes debug logging. Default value: FALSE

Value

An integer matrix of shape (number of objects in `pool`) x (`ntree_end` - `ntree_start`); row `i` is the array of leaf indexes for the `i`-th object.

catboost.caret	<i>Support caret interface</i>
----------------	--------------------------------

Description

Support caret interface

Usage

```
catboost.caret
```

catboost.compare	<i>Compare metrics of two models.</i>
------------------	---------------------------------------

Description

Evaluate `metrics` for `model` and `other` on the same `pool` and return both models' per-iteration metric values for comparison.

Python's `CatBoost.compare(model, data, metrics, ...)` only draws an interactive Jupyter widget from this data (it returns `None`); there is no headless R equivalent of that widget, so `catboost.compare` exposes the widget's underlying metrics-diff data structure instead – what `compare()` computes internally via `self._eval_metrics(...)` and `model._eval_metrics(...)` on the same pool/metrics (both funnel into the same `TMetricsPlotCalcer` vendor entry point as `catboost.eval_metrics`, which this function calls once per model).

Named `catboost.compare` (not `model.compare`, and not an S3 method) because a dotted name would collide with R's existing S3 dispatch on `catboost.Model` (see [predict.catboost.Model](#)).

Usage

```
catboost.compare(
  model,
  other,
  pool,
  metrics,
  ntree_start = 0L,
  ntree_end = 0L,
  eval_period = 1,
  thread_count = -1,
  tmp_dir = NULL
)
```

Arguments

model	The first model obtained as a result of training. Default value: Required argument
other	The second (other) model to compare against model . Default value: Required argument
pool	A catboost.Pool to evaluate both models' metrics on. Default value: Required argument
metrics	A list of metric names to be calculated. (Full list of supported metrics: https://catboost.ai/docs/references/custom-metric___supported-metrics.html) Default value: Required argument
ntree_start	Each model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0
ntree_end	Each model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). If value equals 0, this parameter is ignored and ntree_end is set to that model's own tree_count. Default value: 0
eval_period	Each model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 1
thread_count	The number of threads to use when applying each model. If -1, then the number of threads is set to the number of CPU cores. Default value: -1
tmp_dir	The name of the temporary directory for intermediate results. If NULL, the name is generated with <code>tempdir()</code> . Default value: NULL

Value

An object of class `catboost.compare` (a list): **model** and **other**, each the same named-list-of-numeric-vectors (metric name -> per-iteration values) that `catboost.eval_metrics` returns for that model on **pool**.

See Also

https://catboost.ai/docs/concepts/python-reference_catboost_compare.html

`catboost.compute_training_options`*Resolve training options without training a model*

Description

R equivalent of Python's `catboost.utils.compute_training_options()`: resolves a plain `params` list (the same shape `catboost.train` accepts) to its final, fully-resolved training options, without fitting a model. Some defaults (e.g. `depth`, `border_count`) depend on the shape of the training data, so a minimal description of that shape must be supplied via `train_meta_info` instead of an actual pool.

Usage

```
catboost.compute_training_options(  
  params,  
  train_meta_info,  
  test_meta_info = NULL  
)
```

Arguments

<code>params</code>	Training parameters, as passed to <code>catboost.train</code> .
<code>train_meta_info</code>	A list describing the learn dataset shape: • <code>object_count</code> – number of rows. • <code>feature_count</code> – number of features. • <code>max_cat_features_uniq_values_on_learn</code> – (optional, default 0) max number of unique values across categorical features on learn. • <code>target_min_value</code>, <code>target_max_value</code> – (optional) target value range; omit both to leave target statistics unresolved. • <code>has_pairs</code> – (optional, default FALSE) whether the dataset has pairs.
<code>test_meta_info</code>	Optional, same shape as <code>train_meta_info</code> , describing the test dataset. Defaults to NULL (no test dataset).

Value

A list object with the fully-resolved training options.

```
catboost.create_metric_calcer
```

Create a batch metric calcer

Description

R equivalent of Python's `CatBoost.create_metric_calcer()` (`core.py:3354-3374`): incrementally aggregate metrics across several pool chunks (e.g. a dataset too large to hold as a single Pool at once). Python's `BatchMetricCalcer` streams sufficient statistics natively; this R implementation instead buffers each `$add()`-ed `catboost.Pool` and, on `$eval_metrics()`, concatenates the raw feature/label/weight data of all buffered pools into one fresh Pool and delegates to `catboost.eval_metrics` – numerically identical to streaming aggregation for standard per-object metrics (they are sums/ means over all objects regardless of how those objects were chunked), but **does not preserve group_id/pairs across chunks**: passing several chunks of a grouped/ranking pool will not reproduce Python's per-group behavior. Add the whole pool in one `$add()` call for grouped/ranking metrics.

Usage

```
catboost.create_metric_calcer(
  model,
  metrics,
  ntree_start = 0,
  ntree_end = 0,
  eval_period = 1,
  thread_count = -1,
  tmp_dir = NULL
)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>metrics</code>	A list (or single string) of metric names, same format as <code>catboost.eval_metrics</code> 's <code>metrics</code> argument.
<code>ntree_start</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> . Default value: 0
<code>ntree_end</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> . If value equals to 0 this parameter is ignored and <code>ntree_end</code> equal to <code>tree_count</code> . Default value: 0
<code>eval_period</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> with the given step. Default value: 1
<code>thread_count</code>	The number of threads to use when applying the model. Default value: -1
<code>tmp_dir</code>	Directory for temporary files used by <code>catboost.eval_metrics</code> . Default value: NULL

Value

A `catboost.BatchMetricCalcer` object with `$add(pool)` and `$eval_metrics()` methods.

<code>catboost.cv</code>	<i>Cross-validate model.</i>
--------------------------	------------------------------

Description

Estimate model performance using cross-validation.

Usage

```
catboost.cv(
  pool,
  params = list(),
  fold_count = 3,
  type = "Classical",
  partition_random_seed = 0,
  shuffle = TRUE,
  stratified = FALSE,
  early_stopping_rounds = NULL,
  custom_objective = NULL,
  custom_eval_metric_object = NULL
)
```

Arguments

<code>pool</code>	Data to cross-validate on
<code>params</code>	Parameters for <code>catboost.train</code>
<code>fold_count</code>	Folds count.
<code>type</code>	is type of cross-validation.
<code>partition_random_seed</code>	The random seed used for splitting pool into folds.
<code>shuffle</code>	Shuffle the dataset objects before splitting into folds.
<code>stratified</code>	Perform stratified sampling.
<code>early_stopping_rounds</code>	Activates Iter overfitting detector with <code>od_wait</code> set to <code>early_stopping_rounds</code> .
<code>custom_objective</code>	A user-defined loss function, used instead of <code>params\$loss_function</code> . See catboost.train 's <code>custom_objective</code> argument for the full contract. Default value: <code>NULL</code> (use <code>params\$loss_function</code> as-is)
<code>custom_eval_metric_object</code>	A user-defined eval metric, used instead of <code>params\$eval_metric</code> . See catboost.train 's <code>custom_eval_metric_object</code> argument for the full contract. Default value: <code>NULL</code> (use <code>params\$eval_metric</code> as-is)

Value

A data.frame of evaluation results from cross-validation.

```
catboost.dataset_statistics
```

Calculate dataset statistics

Description

R equivalent of CatBoost CLI's `dataset-statistics` mode: computes per-feature statistics (and, unless `only_light_statistics` is set, per-float-feature histograms) for a dataset read from disk. Calls the same core library entry point the CLI mode itself calls (`catboost/private/libs/app_helpers/NCB::CalculateDatasetStatisticsSingleHost`) directly, in-process – no CLI binary shell-out.

Usage

```
catboost.dataset_statistics(  
  pool_path,  
  cd_path = "",  
  pairs_path = "",  
  delimiter = "\t",  
  has_header = FALSE,  
  thread_count = -1,  
  border_count = 254,  
  only_group_statistics = FALSE,  
  only_light_statistics = FALSE,  
  not_convert_string_targets = TRUE,  
  custom_feature_limits = "",  
  spot_size = 0,  
  spot_count = 0  
)
```

Arguments

<code>pool_path</code>	Path to the dataset file (same format <code>catboost.from_file</code> reads). Default value: Required argument
<code>cd_path</code>	Path to the column description file. Default value: "" (no column description)
<code>pairs_path</code>	Path to the pairs file. Default value: "" (no pairs)
<code>delimiter</code>	Column delimiter in the dataset file. Default value: "\t"
<code>has_header</code>	Whether the dataset file has a header row. Default value: FALSE

<code>thread_count</code>	Number of threads to use. -1 means use all cores. Default value: -1
<code>border_count</code>	Number of histogram bins per float feature. Default value: 254
<code>only_group_statistics</code>	Only compute group-related statistics. Default value: FALSE
<code>only_light_statistics</code>	Skip the second-pass histogram computation. Default value: FALSE
<code>not_convert_string_targets</code>	If TRUE, a string-typed target column is kept as-is (reported as a category distribution). If FALSE, it is converted to a float target instead (fails if the strings are not numeric). Default value: TRUE
<code>custom_feature_limits</code>	Comma-separated list of custom float-feature clip ranges, in the CLI's "<feature_id>:<min>:<max>,..." format (min/max may be "-inf"/"inf"). Values outside [min, max] are reported as underflow/overflow instead of being binned. Default value: "" (no custom limits)
<code>spot_size</code>	Size of each sampled "spot" (contiguous row range) to process instead of the whole dataset. Must be set together with <code>spot_count</code> . Default value: 0 (process the whole dataset)
<code>spot_count</code>	Number of sampled spots to process instead of the whole dataset. Must be set together with <code>spot_size</code> . Default value: 0 (process the whole dataset)

Value

A named list with elements `statistics` and `histograms` (each the parsed contents of the corresponding CLI JSON output file; `histograms` is NULL if `only_light_statistics` was set).

`catboost.Dictionary` *Create text Dictionary*

Description

Builds a token/id vocabulary from text data. Native bridge to Python's `catboost.text_processing.Dictionary`, backed directly by vendor's `NTextProcessing::NDictionary::TDictionary/ TDictionaryBuilder/TBpeDictionary`. Use `catboost.dictionary.fit` to train it and `catboost.dictionary.apply` to convert text to token ids.

Usage

```
catboost.Dictionary(
    token_level_type = NULL,
    gram_order = NULL,
    start_token_id = NULL,
    end_of_word_policy = NULL,
    end_of_sentence_policy = NULL,
    occurrence_lower_bound = NULL,
    max_dictionary_size = NULL,
    dictionary_type = NULL,
    num_bpe_units = NULL,
    skip_unknown = NULL,
    skip_step = NULL
)
```

Arguments

token_level_type	"Word" or "Letter". Default value: "Word"
gram_order	The number of words/letters joined into each token (multigrams). Default value: 1
start_token_id	Initial shift for assigned token identifiers. Default value: 0
end_of_word_policy	"Skip" or "Insert"; used with Letter-level dictionaries. Default value: "Insert"
end_of_sentence_policy	"Skip" or "Insert"; whether <code>catboost.dictionary.apply</code> appends an end-of-sentence token id after a tokenized line. Default value: "Skip"
occurrence_lower_bound	Minimum corpus occurrence count for a token to be kept in the dictionary. Default value: 50
max_dictionary_size	Maximum number of tokens to keep, or -1 for unlimited. Default value: -1
dictionary_type	"FrequencyBased" or "Bpe". Default value: "FrequencyBased"
num_bpe_units	Number of token-pair merges to perform (Bpe type). Default value: 0
skip_unknown	Skip unknown tokens when building a Bpe dictionary. Default value: FALSE

skip_step Number of words/letters skipped when joining them into tokens; only takes effect when `gram_order > 1`.
 Default value: 0

Value

A mutable `catboost.Dictionary` object (`fit/load` update it in place).

```
catboost.dictionary.apply
```

Apply a Dictionary

Description

Converts `data` into token ids.

Usage

```
catboost.dictionary.apply(  
    dictionary,  
    data,  
    tokenizer = NULL,  
    unknown_token_policy = NULL  
)
```

Arguments

dictionary A fitted `catboost.Dictionary` object.
 Default value: Required argument

data A string, character vector, or list mixing strings and already-tokenized character vectors.
 Default value: Required argument

tokenizer An optional `catboost.Tokenizer`, see `catboost.dictionary.fit`.
 Default value: NULL

unknown_token_policy
 "Skip" (drop unknown tokens) or "Insert" (emit `unknown_token_id`).
 Default value: "Skip"

Value

An integer vector of token ids if `data` is a single string, otherwise a list of integer vectors (one per document).

```
catboost.dictionary.end_of_sentence_token_id
```

Identifier of the end-of-sentence token

Description

Identifier of the end-of-sentence token

Usage

```
catboost.dictionary.end_of_sentence_token_id(dictionary)
```

Arguments

dictionary	A fitted catboost.Dictionary object. Default value: Required argument
-------------------	--

Value

An integer id.

```
catboost.dictionary.fit
```

Train a Dictionary

Description

Builds the token/id vocabulary from **data**.

Usage

```
catboost.dictionary.fit(dictionary, data, tokenizer = NULL, verbose = FALSE)
```

Arguments

dictionary	A catboost.Dictionary object. Default value: Required argument
data	A character vector, or a list mixing strings and already-tokenized character vectors. Default value: Required argument
tokenizer	An optional catboost.Tokenizer used to split each length-1 string element of data into tokens; character vector elements of length > 1 are treated as already tokenized regardless. Default value: NULL
verbose	Unused; accepted for interface parity with the Python API. Default value: FALSE

Value

The (mutated) catboost.Dictionary, invisibly.

```
catboost.dictionary.get_token
```

Get a single token by id

Description

Get a single token by id

Usage

```
catboost.dictionary.get_token(dictionary, token_id)
```

Arguments

<code>dictionary</code>	A fitted catboost.Dictionary object. Default value: Required argument
<code>token_id</code>	A token id. Default value: Required argument

Value

A character string.

```
catboost.dictionary.get_tokens
```

Get tokens by id

Description

Get tokens by id

Usage

```
catboost.dictionary.get_tokens(dictionary, token_ids)
```

Arguments

<code>dictionary</code>	A fitted catboost.Dictionary object. Default value: Required argument
<code>token_ids</code>	A vector of token ids. Default value: Required argument

Value

A character vector of tokens.

```
catboost.dictionary.get_top_tokens
```

Most frequent tokens

Description

Most frequent tokens

Usage

```
catboost.dictionary.get_top_tokens(dictionary, top_size = NULL)
```

Arguments

<code>dictionary</code>	A fitted catboost.Dictionary object. Default value: Required argument
<code>top_size</code>	Number of tokens to return. Default value: 10

Value

A character vector of the most frequent tokens, most frequent first.

```
catboost.dictionary.load
```

Load a Dictionary

Description

Reads a dictionary file written by `catboost.dictionary.save` or Python's `Dictionary.save`.

Usage

```
catboost.dictionary.load(dictionary, frequency_dict_path, bpe_path = NULL)
```

Arguments

<code>dictionary</code>	A catboost.Dictionary object to load into. Default value: Required argument
<code>frequency_dict_path</code>	Input file path for the frequency-based alphabet. Default value: Required argument
<code>bpe_path</code>	Input file path for Bpe merge data; when supplied, the loaded dictionary is treated as "Bpe". Default value: NULL

Value

The (mutated) catboost.Dictionary, invisibly.

```
catboost.dictionary.min_unused_token_id
```

Smallest unused token identifier

Description

Smallest unused token identifier

Usage

```
catboost.dictionary.min_unused_token_id(dictionary)
```

Arguments

dictionary A fitted catboost.Dictionary object.
Default value: Required argument

Value

An integer id.

```
catboost.dictionary.save
```

Save a Dictionary

Description

Writes the vendor "id_count_token" text format (a JSON options header line, a token-count line, then one "id\tcount\ttoken" line per token) for "FrequencyBased" dictionaries – byte-identical to the pinned Python oracle's Dictionary.save because both call the same vendor TDictionary::Save. "Bpe" dictionaries additionally require bpe_path and are written via vendor's TBpeDictionary::Save.

Usage

```
catboost.dictionary.save(dictionary, frequency_dict_path, bpe_path = NULL)
```

Arguments

<code>dictionary</code>	A fitted catboost.Dictionary object. Default value: Required argument
<code>frequency_dict_path</code>	Output file path for the frequency-based alphabet. Default value: Required argument
<code>bpe_path</code>	Output file path for Bpe merge data; required when <code>dictionary\$dictionary_type == "Bpe"</code> . Default value: NULL

Value

The catboost.Dictionary, invisibly.

<code>catboost.dictionary.size</code>	<i>Dictionary size</i>
---------------------------------------	------------------------

Description

Dictionary size

Usage

```
catboost.dictionary.size(dictionary)
```

Arguments

<code>dictionary</code>	A fitted catboost.Dictionary object. Default value: Required argument
-------------------------	--

Value

An integer.

`catboost.dictionary.unknown_token_id`
Identifier of the unknown-token placeholder

Description

Identifier of the unknown-token placeholder

Usage

```
catboost.dictionary.unknown_token_id(dictionary)
```

Arguments

`dictionary` A fitted `catboost.Dictionary` object.
Default value: Required argument

Value

An integer id.

`catboost.drop_unused_features`
Drop unused features information from model

Description

Drop unused features information from model

Usage

```
catboost.drop_unused_features(model)
```

Arguments

`model` The model obtained as the result of training.

Value

Status, the result of dropping feature. TRUE if this succeeded, FALSE otherwise.

catboost.eval_feature
Evaluate the impact of feature sets.

Description

R equivalent of the CatBoost CLI's **eval-feature** mode: repeated cross-validated training that measures how much each tested set of features changes the loss, and reports a Wilcoxon test p-value per set.

This calls the same core entry point the CLI mode calls (`EvaluateFeatures`, `catboost/libs/train_lib/eval_feature` in process – no CLI binary is required. There is no Python-side counterpart of this mode; `CatBoost.select_features` is a different algorithm.

Usage

```
catboost.eval_feature(
    pool,
    features_to_evaluate = list(),
    params = list(),
    eval_mode = "OneVsNone",
    offset = 0,
    fold_count = 3,
    fold_size_unit = "Object",
    fold_size = 0,
    relative_fold_size = 0,
    timesplit_quantile = 0.5,
    custom_objective = NULL,
    custom_eval_metric_object = NULL
)
```

Arguments

pool	The dataset to evaluate on (a <code>catboost.Pool</code>). Test sets are not supported by this mode, matching the CLI. Default value: Required argument
features_to_evaluate	Feature sets to test, as a list of integer vectors of 0-based feature indices (CLI: <code>--features-to-evaluate</code>). An empty list evaluates the baseline only, which is meaningful with <code>eval_mode = "OneVsNone"</code> . Default value: <code>list()</code>
params	Parameters for <code>catboost.train</code> . Default value: <code>list()</code>
eval_mode	One of <code>"OneVsNone"</code> , <code>"OneVsOthers"</code> , <code>"OneVsAll"</code> , <code>"OthersVsAll"</code> (CLI: <code>--feature-eval-mode</code>). Default value: <code>"OneVsNone"</code>

<code>offset</code>	First fold used for feature evaluation (CLI: <code>--offset</code>). Default value: 0
<code>fold_count</code>	Number of folds used for feature evaluation (CLI: <code>--fold-count</code>). Default value: 3
<code>fold_size_unit</code>	"Object" or "Group" (CLI: <code>--fold-size-unit</code>). Default value: "Object"
<code>fold_size</code>	Fold size, in <code>fold_size_unit</code> units (CLI: <code>--fold-size</code>). Exactly one of <code>fold_size</code> and <code>relative_fold_size</code> must be non-zero. Default value: 0
<code>relative_fold_size</code>	Fold size as a fraction of the dataset (CLI: <code>--relative-fold-size</code>). Default value: 0
<code>timesplit_quantile</code>	Quantile for the time split (CLI: <code>--timesplit-quantile</code>). Default value: 0.5
<code>custom_objective</code>	A user-defined loss function, used instead of <code>params\$loss_function</code> . See <code>catboost.train</code> 's <code>custom_objective</code> argument for the full contract. Default value: NULL (use <code>params\$loss_function</code> as-is)
<code>custom_eval_metric_object</code>	A user-defined eval metric, used instead of <code>params\$eval_metric</code> . See <code>catboost.train</code> 's <code>custom_eval_metric_object</code> argument for the full contract. Default value: NULL (use <code>params\$eval_metric</code> as-is)

Value

A list with one entry per tested feature set, mirroring the columns of the CLI's `--feature-eval-output-file` TSV but at full double precision:

- `p_value` – numeric, Wilcoxon test p-value per feature set.
- `best_iterations` – list of integer vectors, best baseline iteration per fold.
- `metric_names` – character vector of evaluated metric names.
- `metric_delta` – numeric matrix, feature sets by metrics; average metric change, signed so that positive always means improvement.
- `feature_sets` – list of integer vectors, the evaluated sets echoed back.

`catboost.eval_metric` *Evaluate a metric directly on raw label/approx arrays*

Description

R equivalent of Python's `catboost.utils.eval_metric()` (`utils.py:271`): evaluates a single named metric directly on raw label and approx (prediction) arrays, needing neither a fitted model nor a `catboost.Pool` – unlike `catboost.eval_metrics`, which requires both.

Usage

```
catboost.eval_metric(
  label,
  approx,
  metric,
  weight = NULL,
  group_id = NULL,
  group_weight = NULL,
  subgroup_id = NULL,
  pairs = NULL,
  thread_count = -1
)
```

Arguments

<code>label</code>	Object labels: a numeric vector of length <code>n_objects</code> , or an <code>n_objects</code> x <code>n_target_dimension</code> numeric matrix. Default value: Required argument
<code>approx</code>	Object approxes (raw predictions): a numeric vector of length <code>n_objects</code> , or an <code>n_objects</code> x <code>n_approx_dimension</code> numeric matrix. Default value: Required argument
<code>metric</code>	Metric name, e.g. "Logloss" or "RMSE". Default value: Required argument
<code>weight</code>	Object weights, a numeric vector of length <code>n_objects</code> . Default value: NULL
<code>group_id</code>	Object group ids, a numeric or character vector of length <code>n_objects</code> . Required by groupwise metrics (e.g. "PairLogit", "QueryRMSE"). Default value: NULL
<code>group_weight</code>	Group weights, a numeric vector of length <code>n_objects</code> (one value per object, constant within each group). Default value: NULL
<code>subgroup_id</code>	Subgroup ids, a numeric or character vector of length <code>n_objects</code> . Default value: NULL

pairs	An (N x 2) or (N x 3) numeric matrix of (winner_id, loser_id[, weight]) rows, same convention as <code>catboost.pool.set_pairs</code> . Default value: NULL
thread_count	The number of threads to use. Set to -1 to use all available CPU cores. Default value: -1

Value

A numeric vector with the metric's value(s).

`catboost.eval_metrics`

Calculate metrics.

Description

Calculate the specified metrics for the specified dataset.

Usage

```
catboost.eval_metrics(
  model,
  pool,
  metrics,
  ntree_start = 0L,
  ntree_end = 0L,
  eval_period = 1,
  thread_count = -1,
  tmp_dir = NULL
)
```

Arguments

model	The model obtained as the result of training. Default value: Required argument
pool	The pool for which you want to evaluate the metrics. Default value: Required argument
metrics	The list of metrics to be calculated. (Supported metrics https://catboost.ai/docs/references/custom-metric__supported-metrics.html) Default value: Required argument
ntree_start	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0

ntree_end	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0 (if value equals to 0 this parameter is ignored and ntree_end equal to tree_count)
eval_period	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 1
thread_count	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: -1
tmp_dir	The name of the temporary directory for intermediate results. If NULL, then the name will be generated. Default value: NULL

Value

dict: metric -> array of shape [(ntree_end - ntree_start) / eval_period].

See Also

https://catboost.ai/docs/concepts/python-reference_catboost_eval-metrics.html

catboost.features_data.get_cat_feature_count

Number of categorical features in a FeaturesData

Description

Get the number of categorical features in a catboost.FeaturesData.

Usage

```
catboost.features_data.get_cat_feature_count(features_data)
```

Arguments

features_data A catboost.FeaturesData object.
Default value: Required argument

Value

The number of categorical features.

`catboost.features_data.get_feature_count`*Total number of features in a FeaturesData*

Description

Get the total number of features (numerical + categorical) in a catboost.FeaturesData.

Usage

```
catboost.features_data.get_feature_count(features_data)
```

Arguments

`features_data` A catboost.FeaturesData object.
Default value: Required argument

Value

The total number of features.

`catboost.features_data.get_feature_names`*Get feature names from a FeaturesData*

Description

Get the names of the features of a catboost.FeaturesData: numerical feature names followed by categorical feature names. Unnamed features are empty strings.

Usage

```
catboost.features_data.get_feature_names(features_data)
```

Arguments

`features_data` A catboost.FeaturesData object.
Default value: Required argument

Value

A character vector of feature names, length equal to `catboost.features_data.get_feature_count(features_data)`.

`catboost.features_data.get_num_feature_count`*Number of numerical features in a FeaturesData*

Description

Get the number of numerical features in a catboost.FeaturesData.

Usage

```
catboost.features_data.get_num_feature_count(features_data)
```

Arguments

`features_data` A catboost.FeaturesData object.
Default value: Required argument

Value

The number of numerical features.

`catboost.features_data.get_object_count`*Number of objects in a FeaturesData*

Description

Get the number of objects (rows) in a catboost.FeaturesData.

Usage

```
catboost.features_data.get_object_count(features_data)
```

Arguments

`features_data` A catboost.FeaturesData object.
Default value: Required argument

Value

The number of objects.

`catboost.FeaturesData`*Create a FeaturesData container*

Description

Store features data in a form that can be passed directly to `catboost.load_pool/catboost.from_matrix` as the `data/float_and_cat_features_data` argument, as an alternative to a plain matrix or `data.frame`. Numerical features are given as a numeric matrix, categorical features are given separately as a character matrix, each with optional column names.

Usage

```
catboost.FeaturesData(  
  num_feature_data = NULL,  
  cat_feature_data = NULL,  
  num_feature_names = NULL,  
  cat_feature_names = NULL  
)
```

Arguments

num_feature_data
A numeric matrix of numerical feature values, or `NULL`.
Default value: `NULL`

cat_feature_data
A character matrix of categorical feature values, or `NULL`.
Default value: `NULL`

num_feature_names
A list/vector of names for the numerical features. Must be `NULL` if `num_feature_data` is `NULL`. If not specified, empty strings are used.
Default value: `NULL`

cat_feature_names
A list/vector of names for the categorical features. Must be `NULL` if `cat_feature_data` is `NULL`. If not specified, empty strings are used.
Default value: `NULL`

Value

`catboost.FeaturesData`

```
catboost.get_best_iteration
```

Get best iteration

Description

Return the 0-based iteration index of the best model seen during training, as determined by early stopping / `use_best_model` against the (last) eval set, or NULL if the model was fitted without an eval set (or has no recorded training history). R equivalent of Python's `model.get_best_iteration()/model.best_iteration_`.

Usage

```
catboost.get_best_iteration(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A single integer, or NULL.

```
catboost.get_best_score
```

Get best score

Description

Return the best metric value(s) seen during training for the learn set and each eval set. R equivalent of Python's `model.get_best_score()/model.best_score_`.

Usage

```
catboost.get_best_score(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A named list, one element per dataset (`learn`, plus `validation` if there is exactly one eval set, or `validation_0`, `validation_1`, ... if there are several), each itself a named list of metric name to best value. Empty list if the model has no recorded training history.

`catboost.get_borders` *Get a model's float feature borders*

Description

R equivalent of Python's `model.get_borders()` (`core.py:3822-3827`, defined once on the `CatBoost` base class and inherited unchanged by `CatBoostClassifier`/`CatBoostRegressor`/`CatBoostRanker`): returns the quantization borders the model's float features were split on during training.

Usage

```
catboost.get_borders(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A named list keyed by flat feature index (as a string); each element is the numeric vector of that feature's borders (empty for a feature the model does not use as a float feature).

`catboost.get_evals_result`
Get evaluation results

Description

Return the full per-iteration metric history recorded during training for the learn set and each eval set. R equivalent of Python's `model.get_evals_result()/model.evals_result_`.

Usage

```
catboost.get_evals_result(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A named list, one element per dataset (`learn`, plus `validation` if there is exactly one eval set, or `validation_0`, `validation_1`, ... if there are several), each itself a named list of metric name to a numeric vector (one value per iteration where the metric was recorded – shorter than the total iteration count when `metric_period > 1`). Empty list if the model has no recorded training history.

```
catboost.get_feature_importance
```

Calculate the feature importances

Description

Calculate the feature importances (see <https://catboost.ai/docs/concepts/fstr.html#fstr>) (Regular feature importance, ShapValues, and Feature interaction strength).

Usage

```
catboost.get_feature_importance(
    model,
    pool = NULL,
    type = "FeatureImportance",
    thread_count = -1,
    fstr_type = NULL
)
```

Arguments

<code>model</code>	The model obtained as the result of training. Default value: Required argument
<code>pool</code>	The input dataset. The feature importance for the training dataset is calculated if this argument is not specified. Models with ranking metrics require pool argument to calculate feature importance. Default value: NULL
<code>type</code>	The feature importance type. Possible values: 'PredictionValuesChange' Calculate score for every feature. 'LossFunctionChange' Calculate score for every feature for groupwise model. 'FeatureImportance' 'LossFunctionChange' in case of groupwise model and 'PredictionValuesChange' otherwise. 'Interaction' Calculate pairwise score between every feature. 'ShapValues' Calculate SHAP Values for every object. 'ShapInteractionValues' Calculate SHAP Interaction Values between each pair of features for every object. <code>pool</code> is required.

	• 'PredictionDiff' Calculate the most important features explaining the difference in predictions for a pair of documents. <code>pool</code> is required and must contain exactly 2 rows. • 'SageValues' Calculate SAGE (Shapley Additive Global importanceE) values, a global feature importance measure based on Shapley values of the model's loss function. <code>pool</code> is required. Not supported for multiclass models.
	Default value: 'FeatureImportance'
<code>thread_count</code>	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: -1
<code>fstr_type</code>	Deprecated parameter, use 'type' instead.

Value

Feature importances

See Also

<https://catboost.ai/docs/features/feature-importances-calculation.html>

`catboost.get_leaf_values`
Get the model's tree leaf values

Description

R equivalent of Python's `model.get_leaf_values()` (`core.py:2121-2129`, defined once on `_CatBoostBase` and inherited unchanged by `CatBoost/CatBoostClassifier/CatBoostRegressor/CatBoostRank`).

Usage

```
catboost.get_leaf_values(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A numeric vector of leaf values for all trees, of length `sum(catboost.get_tree_leaf_counts(model)) * ApproxDimension` (`ApproxDimension` is 1 except for multiclass/multi-dimensional losses, e.g. `MultiClass`). Each leaf occupies `ApproxDimension` consecutive entries: the value for dimension `k` (0-based) of the `j`-th leaf (0-based) of the `i`-th tree (0-based) is at position `(sum(catboost.get_tree_leaf_counts(model)[seq_len(i)]) + j) * ApproxDimension + k + 1` (1-based R indexing).

```
catboost.get_leaf_weights
```

Get the model's tree leaf weights

Description

R equivalent of Python's `model.get_leaf_weights()` (`core.py:2131-2139`, defined once on `_CatBoostBase` and inherited unchanged by `CatBoost`/`CatBoostClassifier`/`CatBoostRegressor`/`CatBoostRanker`).

Usage

```
catboost.get_leaf_weights(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A numeric vector of leaf weights for all trees, of length `sum(catboost.get_tree_leaf_counts(model))` – one weight per leaf (unlike `catboost.get_leaf_values`, this is never multiplied by `ApproxDimension`: a leaf has one weight regardless of how many value dimensions it stores).

```
catboost.get_metadata
```

Get model metadata

Description

Return all key/value string metadata pairs stored in the model (training params, custom user data, etc). R equivalent of Python's `model.get_metadata()` (returned as a plain named character vector rather than a dict-like proxy) and of the CLI's `metadata dump` mode. To read a single key, index the result: `catboost.get_metadata(model)[["my_key"]]` errors on a missing key (matching Python's `KeyError` for `metadata["my_key"]`), while `catboost.get_metadata(model)["my_key"]` returns `NA` (matching `metadata.get("my_key")`).

Usage

```
catboost.get_metadata(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A named character vector of all metadata key/value pairs.

`catboost.get_model_feature_names`

Get the feature names used by a model

Description

Return the names of the features used by the model (falling back to their string indices for features that have no name). R equivalent of Python's `model.feature_names_` property and of the CLI's `metadata dump-feature-names` mode.

Usage

```
catboost.get_model_feature_names(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A character vector of feature names, ordered as in the model.

`catboost.get_model_params`

Model parameters

Description

Return the model parameters.

Usage

```
catboost.get_model_params(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A list object with model parameters.

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-get_model_params.html

```
catboost.get_object_importance
```

Calculate the object importances

Description

Calculate the object importances (see <https://catboost.ai/docs/concepts/ostr.html>).

This is the implementation of the LeafInfluence algorithm from the following paper: <https://arxiv.org/pdf/1802.066>

Usage

```
catboost.get_object_importance(
    model,
    pool,
    train_pool,
    top_size = -1,
    type = "Average",
    update_method = "SinglePoint",
    thread_count = -1,
    ostr_type = NULL
)
```

Arguments

<code>model</code>	The model obtained as the result of training. Default value: Required argument
<code>pool</code>	The pool for which you want to evaluate the object importances. Default value: Required argument
<code>train_pool</code>	The pool on which the model has been trained. Default value: Required argument
<code>top_size</code>	Method returns the result of the <code>top_size</code> most important train objects. If -1, then the top size is not limited. Default value: -1
<code>type</code>	Possible values: • 'Average' Method returns the mean train objects scores for all input objects. • 'PerObject' Method returns the train objects scores for every input object.

	Default value: 'Average'
<code>update_method</code>	Description of the update set methods are given in section 3.1.3 of the paper. Possible values: • 'SinglePoint' • 'TopKLeaves' It is possible to set top size : TopKLeaves:top=2. • 'AllPoints' Default value: 'SinglePoint'
<code>thread_count</code>	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: -1
<code>ostr_type</code>	Deprecated parameter, use 'type' instead.

Value

List with elements "indices" and "scores".

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-get_object_importance.html

`catboost.get_plain_params`

Plain Model parameters

Description

Return the plain model parameters.

Usage

```
catboost.get_plain_params(model)
```

Arguments

`model` the model obtained as the result of training.

Value

A list object with model parameters.

```
catboost.get_probability_threshold
```

Get the binary-classification probability threshold

Description

Get the threshold used for class separation in a binary classification model, R equivalent of Python's `CatBoostClassifier.get_probability_threshold()` (`core.py:5909-5914`). Backed by the same model metadata key (`binclass_probability_threshold`) [catboost.get_metadata/catboost.set_metadata](#) expose; defaults to 0.5 (`DEFAULT_BINCLASS_PROBABILITY_THRESHOLD`, `model.h:74`) when the key has never been set.

Usage

```
catboost.get_probability_threshold(model)
```

Arguments

`model` The model obtained as the result of training.

Value

A single numeric value in $[0, 1]$.

```
catboost.get_roc_curve
```

Build a ROC curve

Description

Build the points of the ROC curve for a binary classification model, matching Python's `catboost.utils.get_roc_curve` and the CLI's `roc` mode. Ports the same engine both wrap (`catboost/private/libs/algo/roc_curve.cpp TRocCurve`): raw model predictions are converted to probabilities, sorted in descending order, and swept once to accumulate false positive / false negative rates, inserting a synthetic point wherever the FPR and FNR curves cross.

Usage

```
catboost.get_roc_curve(model, pool, label_border = 0.5)
```

Arguments

<code>model</code>	The model obtained as the result of training on a binary classification task. Default value: Required argument
<code>pool</code>	A <code>catboost.Pool</code> (or list of <code>catboost.Pools</code>) with label data, used to build the curve. Labels are binarized: values <code>> label_border</code> count as the positive class. Labels that do not round to 0 or 1 (i.e. the pool is not a binary classification target) make the call fail with an error. Default value: Required argument
<code>label_border</code>	The label binarization border: a label counts as the positive class if it is strictly greater than this value. Matches the CLI's <code>roc</code> mode <code>--label-border/-b</code> option. Default value: 0.5

Value

A list with three numeric vectors of equal length, sorted by decreasing **threshold**: **fpr** (false positive rate), **tpr** (true positive rate), and **threshold** (probability decision boundary, in `[0, 1]`).

See Also

https://catboost.ai/docs/concepts/python-reference_utils_get_roc_curve.html

`catboost.get_scale_and_bias`
Get model scale and bias

Description

Return the model's scale and bias, used to compute the final formula as `Scale * sumTrees + Bias`. R equivalent of Python's `model.get_scale_and_bias()` and of the CLI's `normalize-model --print-scale-and-bias` mode.

Usage

```
catboost.get_scale_and_bias(model)
```

Arguments

<code>model</code>	The model obtained as the result of training.
--------------------	---

Value

A list with **scale** (a single number) and **bias** (a numeric vector, one value per model output dimension; empty for the zero-bias default).

```
catboost.get_test_eval
```

Get the model's raw predictions on its eval set

Description

R equivalent of Python's `CatBoost.get_test_eval()` (`core.py:1830-1840`): returns the trained model's raw (`RawFormulaVal`) predictions on an evaluation set. Python caches these values in-process at training time (`self._object.__test_evals`, populated by `TrainModel()`'s `TEvalResult` output argument); R's `catboost.Model` object does not retain a reference to the `test_pool` a model was trained with, so this function takes the eval pool explicitly and recomputes the identical values via `catboost.predict(model, eval_pool, prediction_type = "RawFormulaVal")` – mathematically the same quantity Python's cached `TEvalResult` holds, since both are the deterministic output of applying the same fully-trained trees to the same eval set.

Usage

```
catboost.get_test_eval(model, eval_pool)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>eval_pool</code>	The evaluation <code>catboost.Pool</code> the model was (or will be) scored against.

Value

A vector (single-dimension models) or matrix (multiclass models) of raw predictions, one row per object in `eval_pool`.

```
catboost.get_test_evals
```

Get the model's raw predictions on all its eval sets

Description

R equivalent of Python's `CatBoost.get_test_evals()` (`core.py:1842-1850`). Python supports fitting against a list of several eval sets, so `get_test_evals()` returns one entry per set; `catboost.train` only accepts a single `test_pool`, so this always returns a length-1 list wrapping `catboost.get_test_eval`'s result.

Usage

```
catboost.get_test_evals(model, eval_pool)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>eval_pool</code>	The evaluation <code>catboost.Pool</code> the model was (or will be) scored against.

Value

A length-1 list containing the same result `catboost.get_test_eval` would return.

```
catboost.get_tree_leaf_counts
```

Get the number of leaves in each tree of the model

Description

R equivalent of Python's `model.get_tree_leaf_counts()` (`core.py:2112-2119`, defined once on `_CatBoostBase` and inherited unchanged by `CatBoost/CatBoostClassifier/CatBoostRegressor/CatBo`

Usage

```
catboost.get_tree_leaf_counts(model)
```

Arguments

<code>model</code>	The model obtained as the result of training.
--------------------	---

Value

An integer vector of size `catboost.ntrees(model)`; element `i` is the number of leaves in the `i`-th tree (1-based R indexing).

```
catboost.grid_search
```

Exhaustive search over specified parameter values.

Description

R equivalent of Python's `CatBoost.grid_search`: calls the same native search entry point (`NCB::GridSearch`, vendored at `catboost/private/libs/hyperparameter_tuning/hyperparameter_tuning.h`) that Python's `CatBoost._tune_hyperparams` calls via its Cython wrapper, rather than approximating the search with an R-side loop over `catboost.cv` – the grid/quantization-parameter enumeration order and train/test-split reuse live entirely in that native code, so calling it directly is what makes best-params/best-score match the Python oracle.

Usage

```
catboost.grid_search(
  param_grid,
  pool,
  params = list(),
  cv = 3,
  partition_random_seed = 0,
  calc_cv_statistics = TRUE,
  search_by_train_test_split = TRUE,
  refit = TRUE,
  shuffle = TRUE,
  stratified = FALSE,
  train_size = 0.8,
  verbose = TRUE,
  custom_objective = NULL,
  custom_eval_metric_object = NULL
)
```

Arguments

param_grid	Named list of parameter name -> vector of values to try, or an unnamed list of such named lists (multiple grids, explored independently). Default value: Required argument
pool	Data to search on (a <code>catboost.Pool</code>). Default value: Required argument
params	Fixed parameters for <code>catboost.train</code> , held constant across the search. Default value: <code>list()</code>
cv	Number of cross-validation folds. Default value: 3
partition_random_seed	The random seed used for splitting the data. Default value: 0
calc_cv_statistics	Whether to estimate quality via cross-validation with the found best parameters. Only used when <code>search_by_train_test_split = TRUE</code> . Default value: <code>TRUE</code>
search_by_train_test_split	If <code>TRUE</code> , the dataset is split into train/test parts, candidates are trained on the train part and compared by loss on the test part. If <code>FALSE</code> , every candidate is evaluated with cross-validation instead. Default value: <code>TRUE</code>
refit	If <code>TRUE</code> , fit a model on <code>pool</code> with the best found parameters (via <code>catboost.train</code>) and return it as <code>\$model</code> . Default value: <code>TRUE</code>
shuffle	Shuffle the dataset objects before searching. Default value: <code>TRUE</code>

stratified	Perform stratified sampling for the cross-validation statistics. Unlike Python (which auto-detects this from the loss function), R always defaults to FALSE , matching <code>catboost.cv</code> 's own precedent of leaving stratification to the caller. Default value: FALSE
train_size	Proportion of the dataset used for the train split (used when <code>search_by_train_test_split = TRUE</code>). Default value: 0.8
verbose	Whether to print search progress. Default value: TRUE
custom_objective	A user-defined loss function, used instead of <code>params\$loss_function</code> for both the search itself and (when <code>refit = TRUE</code>) the final refit. See <code>catboost.train</code> 's <code>custom_objective</code> argument for the full contract. Default value: NULL (use <code>params\$loss_function</code> as-is)
custom_eval_metric_object	A user-defined eval metric, used instead of <code>params\$eval_metric</code> for both the search itself and (when <code>refit = TRUE</code>) the final refit. See <code>catboost.train</code> 's <code>custom_eval_metric_object</code> argument for the full contract. Default value: NULL (use <code>params\$eval_metric</code> as-is)

Value

A list with `$params` (best found parameters, as a named list), `$cv_results` (a `data.frame` of cross-validation results with the same columns `catboost.cv` returns), and, if `refit = TRUE`, `$model` (a `catboost.Model` fit with the best parameters).

Examples

```
# (c) refit = TRUE correctly reuses the supplied custom_objective for the
# final refit, not just for the search itself.
## Not run:
n <- 50
features <- data.frame(x1 = rnorm(n), x2 = rnorm(n))
label <- with(features, x1 - x2 + rnorm(n, sd = 0.3))
pool <- catboost.load_pool(features, label = label)
rmse_custom_objective <- list(
  calc_ders_range = function(approx, target, weight) {
    w <- if (is.null(weight)) 1 else weight
    cbind(w * (target - approx), w * (-1))
  })
result <- catboost.grid_search(
  list(depth = c(3, 4)), pool,
  params = list(iterations = 10, logging_level = "Silent",
    eval_metric = "RMSE", boost_from_average = FALSE),
  cv = 3, refit = TRUE, verbose = FALSE,
  custom_objective = rmse_custom_objective)
# result$model was refit with rmse_custom_objective, not a built-in loss.

## End(Not run)
```

```
catboost.iterate_leaf_indexes
```

Iterate the leaf index each object falls into, per tree

Description

R equivalent of Python's `model.iterate_leaf_indexes(data, ntree_start, ntree_end)` (`core.py:3132-3155`). Python streams results one object at a time via `_leaf_indexes_iterator` purely to bound memory for very large pools; the underlying leaf-index computation is identical to `catboost.calc_leaf_indexes` (same `CalcLeafIndexesMulti` native call). R has no lazy generator protocol in the base language, so this is a thin wrapper that computes the full result via `catboost.calc_leaf_indexes` and splits it into one list element per object – identical numeric output, not an approximation.

Usage

```
catboost.iterate_leaf_indexes(model, pool, ntree_start = 0, ntree_end = 0)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>pool</code>	The input data on which to calculate leaf indexes. Should be a <code>catboost.Pool</code> object.
<code>ntree_start</code>	Index of the first tree for which leaf indexes will be calculated (zero-based indexing). Default value: 0
<code>ntree_end</code>	Index of the tree after the last tree for which leaf indexes will be calculated (zero-based indexing). If 0, resolves to <code>catboost.ntrees(model)</code> . Default value: 0

Value

A list with one element per object in `pool`; each element is an integer vector of leaf indexes for that object.

```
catboost.load_model
```

Load the model

Description

Load the model from a file.

Note: Feature importance (see <https://catboost.ai/docs/concepts/fstr.html#fstr>) is not saved when using this function.

Usage

```
catboost.load_model(model_path, file_format = "cbm")
```

Arguments

<code>model_path</code>	The path to the model. Default value: Required argument
<code>file_format</code>	Format of the model file. Default value: 'cbm'

Value

A model object.

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-load_model.html

<code>catboost.load_pool</code>	<i>Create a dataset</i>
---------------------------------	-------------------------

Description

Create a dataset from the given file, matrix or data.frame.

Usage

```
catboost.load_pool(
  data,
  label = NULL,
  cat_features = NULL,
  column_description = NULL,
  pairs = NULL,
  delimiter = "\t",
  has_header = FALSE,
  weight = NULL,
  group_id = NULL,
  group_weight = NULL,
  subgroup_id = NULL,
  pairs_weight = NULL,
  baseline = NULL,
  feature_names = NULL,
  thread_count = -1,
  graph = NULL,
  embedding_features = NULL,
  timestamp = NULL,
  feature_tags = NULL
)
```

Arguments

data	A file path, matrix, sparse matrix (any <code>Matrix</code> package <code>sparseMatrix</code>, e.g. <code>dgCMatrix</code>) or <code>data.frame</code> with features. A sparse matrix is densified before being handed to the native Pool builder, so the resulting Pool is identical to the one built from <code>as.matrix(data)</code> (stored zeros are real zeros). Caveat: because the sparse layout is not preserved, training on such a Pool reproduces Python's <i>dense</i> Pool, not its native sparse one. CatBoost breaks ties between equal-scoring split candidates differently in its sparse and dense column layouts, so on degenerate/tie-heavy data the two can differ (a delta of 0.0298 was measured inside Python itself between its own dense and sparse Pools). The following column types are supported: • double • factor. It is assumed that categorical features are given in this type of columns. A standard CatBoost processing procedure is applied to this type of columns: 1. The values are converted to strings. 2. The <code>ConvertCatFeatureToFloat</code> function is applied to the resulting string. Default value: Required argument
label	The label vector or label matrix. A plain (non-factor, non-character) integer label <i>matrix</i> with more than one column (e.g. a multi-target 0/1 matrix for <code>MultiLogloss</code>) is automatically read as a float target, matching Python's <code>Pool</code>. A single-column integer label vector keeps its existing integer-target behavior.
cat_features	A vector of categorical features indices. The indices are zero based and can differ from the given in the Column descriptions file. If data parameter is <code>data.frame</code> don't use <code>cat_features</code>, categorical features are determined automatically from <code>data.frame</code> column types.
column_description	The path to the input file that contains the column descriptions.
pairs	A file path, matrix or <code>data.frame</code> that contains the pairs descriptions. The shape should be <code>Nx2</code>, where <code>N</code> is the pairs' count. The first element of pair is the index of winner document in training set. The second element of pair is the index of loser document in training set.
delimiter	Delimiter character to use to separate features in a file.
has_header	Read column names from first line, if this parameter is set to <code>True</code>.
weight	The weights of the objects.
group_id	The group ids of the objects.
group_weight	The group weight of the objects.
subgroup_id	The subgroup ids of the objects.
pairs_weight	The weights of the pairs.
baseline	Vector of initial (raw) values of the objective function. Used in the calculation of final values of trees.

<code>feature_names</code>	A list of names for each feature in the dataset.
<code>thread_count</code>	The number of threads to use while reading the data. Optimizes reading time. This parameter doesn't affect results.
<code>graph</code>	A file path, matrix or data.frame that contains the pairs of indices of objects for graph features. The shape should be Nx2, where N is the pairs of indices count. If -1, then the number of threads is set to the number of CPU cores. Known limitation: passing a matrix or data.frame here (in-memory <code>graph</code>) fails with a native "Unimplemented" error. The in-memory Pool builder <code>catboost.load_pool()/catboost.from_matrix()</code> use (<code>TRawFeaturesOrderDataProviderBuilder</code> in vendor/catboost's <code>catboost/libs/data/data_p</code> has never implemented <code>SetGraph()</code> ; only the sibling builder used by the file-based path (<code>TRawObjectsOrderDataProviderBuilder</code>) does. There is no patch layer over the vendored native sources in this package, so this cannot be fixed from the R/C++ bridge alone. Documented workaround: write <code>graph</code> to a file and pass a file data path instead (<code>catboost.load_pool(pool_path, ..., graph = graph_path)</code>), which routes through the working file-based builder.
<code>embedding_features</code>	A named list of numeric matrices, one per embedding feature; each matrix has one row per object and one column per embedding dimension. R matrix cells cannot hold vectors, so embedding features are supplied beside <code>data</code> instead of inside it (Python's <code>Pool(data, embedding_features = [...])</code> indexes columns of a data frame holding arrays). The embedding features are appended after the columns of <code>data</code> , so their flat feature indices are <code>ncol(data)</code> , <code>ncol(data) + 1</code> , ...; list names become their feature names. Caveat: the <code>embedding_processing</code> training parameter (given to <code>catboost.train</code> , not here) defaults to <code>list(default = list("LDA", "KNN"))</code> . The LDA calcer's output is not bit-reproducible against the Python package: LDA solves a float32 symmetric eigenproblem whose eigenvector signs and near-degenerate eigenvalue ordering depend on the LAPACK/BLAS build, so this package and the Python wheel legitimately disagree. The KNN calcer agrees exactly. For parity-sensitive use, pass <code>embedding_processing = list(default = list("KNN"))</code> . The CLI's <code>--embedding-calcers</code> flag is a mutually exclusive alternate encoding of the same default-slot calcer list, also accepted as a <code>catboost.train</code> params key: <code>embedding_calcers = list("KNN")</code> is equivalent to the <code>embedding_processing</code> call above (giving <code>embedding_calcers</code> and <code>embedding_processing</code> together is an error, matching the CLI).
<code>timestamp</code>	A numeric vector of per-object timestamps, length equal to the number of objects. Convenience wrapper around <code>catboost.pool.set_timestamp</code> : applied to the constructed Pool before it is returned, equivalent to calling <code>catboost.pool.set_timestamp(pool, timestamp)</code> afterward.
<code>feature_tags</code>	A named list grouping features for <code>catboost.select_features</code> 's grouping = "ByTags" mode (mirrors Python's <code>Pool(feature_tags=...)</code>). Each element is itself a list with a required <code>features</code> entry – a vector of 0-based feature indices or feature names (requires <code>feature_names</code> to be supplied) – and an optional <code>cost</code> (default 1.0), e.g. <code>list(group_a =</code>

`list(features = c(0, 1)), group_b = list(features = c("f2", "f3"), cost = 2))`. Only supported when `data` is a matrix, sparse matrix or `catboost.FeaturesData`; `NULL` (the default) attaches no tags. Loading a pool from a file or a `data.frame` does not support feature tags; passing a non-`NULL` value in those cases raises an error.

Value

`catboost.Pool`

Examples

```
## Not run:
# From file
pool_path <- system.file("extdata", "adult_train.1000", package = "catboost")
cd_path <- system.file("extdata", "adult.cd", package = "catboost")
pool <- catboost.load_pool(pool_path, column_description = cd_path)
print(pool)

# From matrix
target <- 1
data_matrix <- matrix(runif(18), 6, 3)
pool <- catboost.load_pool(data_matrix[, -target], label = data_matrix[, target])
print(pool)

# From data.frame
nonsense <- factor(c('A', 'B', 'C'))
data_frame <- data.frame(value = runif(10), category = nonsense[(1:10) %% 3 + 1])
label = (1:10) %% 2
pool <- catboost.load_pool(data_frame, label = label)
print(pool)

## End(Not run)
```

`catboost.model_based_eval`

Model-based feature evaluation.

Description

R equivalent of the CatBoost CLI's `model-based-eval` mode: continue a pre-trained baseline model's training in repeated short experiments, with and without the tested feature sets, and write the resulting per-experiment error logs into `train_dir`.

This calls the same core entry point the CLI mode calls (`ModelBasedEval`, `catboost/libs/train_lib/train_model` in process – no CLI binary is required. There is no Python-side counterpart of this mode.

This mode is GPU-only. CatBoost's CPU trainer refuses it outright ("Model based eval is not implemented for CPU"), and the mode's options are only recognised when `task_type = "GPU"`. On a build or machine without a working CUDA device the call fails with a specific error rather than falling back to CPU.

Unlike `catboost.eval_feature` this function takes dataset *file paths* rather than a `catboost.Pool`: the only public core entry point for this mode is file-driven, it reads the baseline model's training snapshot from disk, and it writes its results to disk.

Usage

```
catboost.model_based_eval(
  learn_set,
  test_set,
  features_to_evaluate,
  baseline_model_snapshot = "baseline_model_snapshot",
  column_description = NULL,
  params = list(task_type = "GPU"),
  offset = 1000,
  experiment_count = 200,
  experiment_size = 5,
  use_evaluated_features_in_baseline_model = FALSE,
  delimiter = "\t",
  has_header = FALSE
)
```

Arguments

- | | |
|--------------------------------------|--|
| <code>learn_set</code> | Path to the learn dataset file (CLI: <code>--learn-set</code>).
Default value: Required argument |
| <code>test_set</code> | Path to the test dataset file (CLI: <code>--test-set</code>). Required: the mode evaluates the change in <i>test</i> error.
Default value: Required argument |
| <code>features_to_evaluate</code> | Feature sets to test, in the CLI's <code>--features-to-evaluate</code> syntax: sets separated by <code>;</code> , each set a comma-separated list of 0-based indices, index ranges (4,78-89,312), or feature names. <code>#tag</code> references are not supported by this R wrapper: this function never sets <code>poolLoadParams.PoolMetaInfoPath</code> , which tag resolution requires, so any <code>#tag</code> value errors with "There is no tag '#x' in pool metainfo".
Default value: Required argument |
| <code>baseline_model_snapshot</code> | Path to the snapshot of the baseline model's training (CLI: <code>--baseline-model-snapshot</code>).
Default value: "baseline_model_snapshot" |
| <code>column_description</code> | Path to the column description file (CLI: <code>--column-description</code>). NULL means none.
Default value: NULL |
| <code>params</code> | Parameters for <code>catboost.train</code> . Must include <code>task_type = "GPU"</code> ; <code>train_dir</code> selects where the results are written.
Default value: <code>list(task_type = "GPU")</code> |

<code>offset</code>	Number of last iterations of the baseline model's training to evaluate over (CLI: <code>--offset</code>). Must be at least <code>experiment_count * experiment_size</code> . Default value: 1000
<code>experiment_count</code>	Number of experiments (CLI: <code>--experiment-count</code>). Default value: 200
<code>experiment_size</code>	Number of iterations in one experiment (CLI: <code>--experiment-size</code>). Default value: 5
<code>use_evaluated_features_in_baseline_model</code>	Keep the evaluated features in the baseline model instead of zeroing them out (CLI: <code>--use-evaluated-features-in-baseline-model</code>). Default value: FALSE
<code>delimiter</code>	Field delimiter of the dataset files (CLI: <code>--delimiter</code>). Default value: <code>"\t"</code>
<code>has_header</code>	Whether the dataset files have a header line (CLI: <code>--has-header</code>). Default value: FALSE

Value

The `train_dir` the per-experiment error logs were written to, invisibly.

See Also

[catboost.eval_feature](#) for the CPU-capable `eval-feature` mode.

`catboost.normalize_model_from_pool`

Rescale a model so its raw predictions on a pool span $[0, 1]$

Description

CLI-only capability: R equivalent of the CLI's `normalize-model --input-path/-i mode (mode_normalize_model.cpp)`, which has no Python `get_scale_and_bias/set_scale_and_bias` counterpart. Resets the model to identity scale/bias, computes the min and max of its raw (`RawFormulaVal`) predictions over `pool`, then sets `scale = 1 / (max - min)`, `bias = -scale * min` so the rescaled raw predictions span exactly $[0, 1]$ – mirroring `mode_normalize_model.cpp`'s `CalcMinMaxOnAllPools + model.SetScaleAndBias({scale, {bias}})` in-place.

Usage

```
catboost.normalize_model_from_pool(model, pool)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>pool</code>	A <code>catboost.Pool</code> (or list of pools) to compute the min/max raw prediction range over.

Value

No return value, called for side effects.

```
catboost.plot_partial_dependence
```

Compute the partial dependence of the model on one or two features

Description

R equivalent of Python's `CatBoost.plot_partial_dependence()` (`core.py:3947-4020`), minus the optional plotly figure. Computes, per bucket of the given float feature(s), the model's mean prediction over `pool` when that feature is perturbed to a representative value of the bucket – the standard partial-dependence definition, and mathematically identical to CatBoost's native tree-traversal implementation (`GetPartialDependence()`, `catboost/libs/fstr/partial_dependence.cpp`) for CPU (oblivious-tree) models: since every float-feature split is at one of that feature's borders, the prediction is constant within a bucket, so averaging predictions at any one representative point per bucket equals the true expectation the native algorithm computes via leaf-weight redistribution.

Usage

```
catboost.plot_partial_dependence(
    model,
    pool,
    features,
    prediction_type = "RawFormulaVal",
    ntree_start = 0,
    ntree_end = 0,
    thread_count = -1
)
```

Arguments

<code>model</code>	The model obtained as the result of training. Only supported for single-dimensional (non-multiclass), all-numeric-feature, oblivious-tree models – the same constraints <code>GetPartialDependence()</code> itself enforces (<code>partial_dependence.cpp:187-1</code>).
<code>pool</code>	The input dataset. Should be a <code>catboost.Pool</code> object.
<code>features</code>	A single feature, or a vector/list of 2 features (0-based integer index or feature name string), for float features only.
<code>prediction_type</code>	The prediction type passed to <code>catboost.predict</code> for the perturbed data. Default value: "RawFormulaVal"
<code>ntree_start</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> . Default value: 0

ntree_end	Model is applied on the interval [ntree_start, ntree_end). If value equals to 0 this parameter is ignored and ntree_end equal to tree_count. Default value: 0
thread_count	The number of threads to use when applying the model. Default value: -1

Value

If one feature was given: a numeric vector, one value per bucket. If two features were given: a numeric matrix, rows indexed by the first feature's buckets and columns by the second's.

catboost.plot_predictions

Vary one feature's value and observe the prediction

Description

R equivalent of Python's `CatBoost.plot_predictions()` (`core.py:3847-3945`), minus the optional plotly figure: for each object in `pool` and each feature in `features_to_change`, the feature's value is perturbed across every bucket of the model's float- feature borders (`catboost.predict` is then called on the perturbed data) and the resulting predictions are returned as the primary, required contract of this function – `core.py:3847-4020`'s own evidence (see this ticket) established these are real numeric prediction arrays, not plotting-only output.

Usage

```
catboost.plot_predictions(
    model,
    pool,
    features_to_change,
    prediction_type = "RawFormulaVal",
    ntree_start = 0,
    ntree_end = 0,
    thread_count = -1
)
```

Arguments

model	The model obtained as the result of training.
pool	The input dataset. Should be a <code>catboost.Pool</code> object.
features_to_change	A vector/list of 0-based integer feature indices or feature name strings, for float features only.

prediction_type	The prediction type passed to <code>catboost.predict</code> for the perturbed data. Note on R-vs-Python parity: Python's <code>plot_predictions()</code> calls <code>self.predict(doc)</code> with no explicit <code>prediction_type</code>, so its oracle value depends on which subclass' <code>predict()</code> default applies ('RawFormulaVal' for the base class and <code>CatBoostRanker</code>, loss-dependent for <code>CatBoostRegressor</code>, 'Class' for <code>CatBoostClassifier</code>); R has no per-subclass default to hang this on, so it must be passed explicitly to match a given Python class's behavior (same rationale as <code>catboost.predict</code>'s own parity note). Default value: "RawFormulaVal"
ntree_start	Model is applied on the interval <code>[ntree_start, ntree_end)</code>. Default value: 0
ntree_end	Model is applied on the interval <code>[ntree_start, ntree_end)</code>. If value equals to 0 this parameter is ignored and <code>ntree_end</code> equal to <code>tree_count</code>. Default value: 0
thread_count	The number of threads to use when applying the model. Default value: -1

Value

A list with one entry per object in `pool`; each entry is a named list keyed by the (string) 0-based feature index, holding the vector (or matrix, for multi-dimensional `prediction_types`) of predictions across that feature's buckets.

<code>catboost.plot_tree</code>	<i>Plot a single tree's structure.</i>
---------------------------------	--

Description

Return the node/edge structure of one tree in `model` (see https://catboost.ai/docs/concepts/python-reference_catboost_plot_tree.html).

Python's `plot_tree` returns a `graphviz.Digraph` built from the model's internal per-tree splits and leaf values (`_get_tree_splits/_get_tree_leaf_values` in `catboost/python-package/catboost/core` reading the same vendor `TFullModel` tree layout that `catboost.save_model`'s "json" export format serializes). `catboost.plot_tree` reuses that existing JSON export (no new native entry point) to read the same split/leaf data and reconstructs the DOT-text graph with base R string building – no `DiagrammeR/graphviz` R package dependency is added, since none is required for either the structural differential test (spec Sec 4.3's "Structural" row targets canonical node/edge JSON, not a rendered image) or the "object of the expected class" smoke test.

Only oblivious (symmetric) trees with `FloatFeature/OneHotFeature` splits are supported; other tree/split kinds stop with an explicit error rather than silently mis-rendering.

Usage

```
catboost.plot_tree(model, tree_idx, pool = NULL)
```

Arguments

<code>model</code>	The model obtained as the result of training. Default value: Required argument
<code>tree_idx</code>	0-based index of the tree to plot. Default value: Required argument
<code>pool</code>	A catboost.Pool used to resolve feature names and to decode categorical split values. Required if the tree splits on any categorical feature (mirrors Python's own <code>plot_tree</code> , which raises if a categorical split is present and no pool is given); optional for float-only trees, in which case node labels fall back to the 0-based flat feature index – again mirroring Python's own <code>pool = NULL</code> fallback. Categorical split values are decoded by re-hashing the pool's raw string values with <code>CatBoostCalcCatFeatureHash_R</code> (the same raw <code>CalcCatFeatureHash()</code> stored in the exported JSON model's <code>TOneHotSplit::Value</code>) until the split's hash is matched; when pool was built by <code>catboost.load_pool/catboost.from_matrix</code> (which pre-hash categorical columns into floats before the vendor pool is built, so no hash-to-string dictionary survives – see <code>catboost.calc_feature_statistics</code> 's docs) the original string cannot be recovered and the label falls back to " <code><hash:...></code> ". Default value: NULL

Value

An object of class `catboost.plot_tree`: a list with `dot` (character scalar, DOT-language source text), `nodes` (data.frame: `id`, `label`, `color`, `shape`) and `edges` (data.frame: `from`, `to`, `label`).

See Also

https://catboost.ai/docs/concepts/python-reference_catboost_plot_tree.html

```
catboost.pool.get_baseline
```

Get baseline from a Pool

Description

Get the baseline data of a Pool.

Usage

```
catboost.pool.get_baseline(pool)
```

Arguments

<code>pool</code>	A catboost.Pool object. Default value: Required argument
-------------------	---

Value

A (rows x baseline__count) numeric matrix. `baseline__count` is 0 if the Pool has no baseline data.

`catboost.pool.get_cat_feature_indices`*Get categorical feature indices from a Pool*

Description

Get the (0-based) column indices of the categorical features of a Pool.

Usage

```
catboost.pool.get_cat_feature_indices(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

An integer vector of 0-based categorical feature indices.

`catboost.pool.get_embedding_feature_indices`*Get embedding feature indices from a Pool*

Description

Get the (0-based) column indices of the embedding features of a Pool. Always `integer(0)`: this package does not support building Pools with embedding features.

Usage

```
catboost.pool.get_embedding_feature_indices(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

An integer vector of 0-based embedding feature indices.

`catboost.pool.get_feature_names`*Get feature names from a Pool*

Description

Get the names of the features (columns) of a Pool.

Usage

```
catboost.pool.get_feature_names(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

A character vector of feature names, length equal to `catboost.pool.num_col(pool)`. Unnamed features are empty strings.

`catboost.pool.get_features`*Get the feature matrix from a Pool*

Description

Get the raw numeric feature matrix of a Pool. Only supported for Pools whose features are all numeric (no categorical or text features).

Usage

```
catboost.pool.get_features(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

A (rows x columns) numeric matrix of feature values.

`catboost.pool.get_group_id_hash`*Get group id hashes from a Pool*

Description

Get the hashes generated from a Pool's group ids.

Usage

```
catboost.pool.get_group_id_hash(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

A character vector of decimal-formatted 64-bit hash values (one per object), or `NULL` if the Pool has no group ids. Returned as character rather than numeric because R's double cannot represent the full 64-bit hash range exactly.

`catboost.pool.get_label`*Get labels from a Pool*

Description

Get the label (target) data of a Pool.

Usage

```
catboost.pool.get_label(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

A vector of labels if the target is one-dimensional, a (rows x targets) matrix otherwise. `numeric(0)` if the Pool has no label data.

`catboost.pool.get_text_feature_indices`*Get text feature indices from a Pool*

Description

Get the (0-based) column indices of the text features of a Pool.

Usage

```
catboost.pool.get_text_feature_indices(pool)
```

Arguments

<code>pool</code>	A catboost.Pool object. Default value: Required argument
-------------------	---

Value

An integer vector of 0-based text feature indices.

`catboost.pool.get_weight`*Get weights from a Pool*

Description

Get the per-object weight of a Pool. Objects with no weight set default to weight 1.

Usage

```
catboost.pool.get_weight(pool)
```

Arguments

<code>pool</code>	A catboost.Pool object. Default value: Required argument
-------------------	---

Value

A numeric vector of per-object weights.

`catboost.pool.has_label`*Has the Pool got label data*

Description

Check whether the Pool has label (target) data.

Usage

```
catboost.pool.has_label(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

TRUE if the Pool has label data, FALSE otherwise.

`catboost.pool.is_empty`*Is the Pool empty*

Description

Check whether the Pool has no objects.

Usage

```
catboost.pool.is_empty(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

TRUE if the Pool has zero rows, FALSE otherwise.

`catboost.pool.is_quantized`
Is the Pool quantized

Description

Check whether the Pool's feature data has already been quantized (either by `catboost.pool.quantize` or as a side effect of `catboost.train`).

Usage

```
catboost.pool.is_quantized(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

TRUE if the Pool is quantized, FALSE otherwise.

`catboost.pool.num_col`
Number of columns in a Pool

Description

Get the number of features (columns) in a Pool.

Usage

```
catboost.pool.num_col(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

The number of feature columns.

`catboost.pool.num_pairs`
Number of pairs in a Pool

Description

Get the number of pairs in a Pool.

Usage

```
catboost.pool.num_pairs(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

An integer, the number of pairs.

`catboost.pool.num_row`
Number of rows in a Pool

Description

Get the number of objects (rows) in a Pool.

Usage

```
catboost.pool.num_row(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

The number of rows.

```
catboost.pool.quantize
```

Quantize a Pool

Description

Quantize this Pool in place: build the binarized (quantized) feature data used by training, same as Python's `Pool.quantize()` (`catboost/libs/data/quantization.h`'s `ConstructQuantizedPoolFromRaw` the same core entry point `catboost.train` itself uses).

Usage

```
catboost.pool.quantize(pool, params = list())
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Must not already be quantized. Default value: Required argument
<code>params</code>	A named list of quantization parameters (e.g. <code>border_count</code> , <code>feature_border_type</code> , <code>nan_mode</code> , <code>per_float_feature_quantization</code> , <code>ignored_features</code>). Same names/semantics as <code>catboost.train</code> 's <code>params</code> . Default value: <code>list()</code>

Value

Nothing. The Pool is quantized in place.

```
catboost.pool.save
```

Save a quantized Pool to CatBoost's binary quantized-pool format

Description

R equivalent of Python's `Pool.save()`: saves an already-quantized Pool to CatBoost's own binary quantized-pool format (`catboost/private/libs/quantized_pool/serialization.h`'s `SaveQuantizedPool`, the same core entry point Python's `Pool.save()/_save()` calls). This is a different, binary format from `catboost.save_pool` (R/`catboost.R`), which writes CD/TSV files read back by `catboost.load_pool`'s `column_description` path – `catboost.save_pool` does NOT already satisfy Python's `Pool.save()`.

Usage

```
catboost.pool.save(pool, fname)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Must already be quantized (see <code>catboost.pool.quantize</code>). Default value: Required argument
<code>fname</code>	Output file path. Default value: Required argument

Value

Nothing. Writes `fname` as a side effect.

```
catboost.pool.save_quantization_borders
```

Save a Pool's quantization borders to a file

Description

Save the borders used in numeric feature quantization to a file, so they can be reused to quantize another Pool identically (via upstream's `input_borders` mechanism). File format is described at https://catboost.ai/docs/concepts/input-data_custom-borders.html.

Usage

```
catboost.pool.save_quantization_borders(pool, output_file)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Must already be quantized. Default value: Required argument
<code>output_file</code>	Output file path. Default value: Required argument

Value

Nothing. Writes `output_file` as a side effect.

```
catboost.pool.set_baseline
```

Set baseline on a Pool

Description

Set the baseline data of a Pool.

Usage

```
catboost.pool.set_baseline(pool, baseline)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>baseline</code>	A (rows x baseline_count) numeric matrix, <code>rows</code> equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_feature_names
```

Set feature names on a Pool

Description

Set the names of the features (columns) of a Pool.

Usage

```
catboost.pool.set_feature_names(pool, feature_names)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>feature_names</code>	A character vector of feature names, length equal to <code>catboost.pool.num_col(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

`catboost.pool.set_group_id`
Set group ids on a Pool

Description

Set the group ids of a Pool. Each id is hashed the same way Python's `Pool.set_group_id` hashes it, so the resulting group id hashes (see [catboost.pool.get_group_id_hash](#)) match the Python oracle for the same input values.

Usage

```
catboost.pool.set_group_id(pool, group_id)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>group_id</code>	A character or integral-numeric vector, length equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

`catboost.pool.set_group_weight`
Set group weights on a Pool

Description

Set the per-object group weight of a Pool (weights must be equal within each group).

Usage

```
catboost.pool.set_group_weight(pool, group_weight)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>group_weight</code>	A numeric vector of per-object group weights, length equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_pairs
```

Set pairs on a Pool

Description

Set the pairwise comparison data of a Pool.

Usage

```
catboost.pool.set_pairs(pool, pairs)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>pairs</code>	An (N x 2) or (N x 3) numeric matrix of (<code>winner_id</code> , <code>loser_id</code> [, <code>weight</code>]) rows. <code>winner_id/loser_id</code> are 0-indexed object row numbers, matching the <code>pairs</code> argument of <code>catboost.from_matrix</code> . <code>weight</code> defaults to 1.0 when the third column is omitted. Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_pairs_weight
```

Set pair weights on a Pool

Description

Set the per-pair weight of a Pool's existing pairs, keeping the (`winner_id`, `loser_id`) ids unchanged.

Usage

```
catboost.pool.set_pairs_weight(pool, pairs_weight)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>pairs_weight</code>	A numeric vector of per-pair weights, length equal to <code>catboost.pool.num_pairs(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_subgroup_id
```

Set subgroup ids on a Pool

Description

Set the subgroup ids of a Pool. Each id is hashed the same way Python's `Pool.set_subgroup_id` hashes it.

Usage

```
catboost.pool.set_subgroup_id(pool, subgroup_id)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>subgroup_id</code>	A character or integral-numeric vector, length equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_timestamp
```

Set timestamps on a Pool

Description

Set the per-object timestamp of a Pool.

Usage

```
catboost.pool.set_timestamp(pool, timestamp)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>timestamp</code>	A numeric vector of per-object timestamps, length equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.set_weight
```

Set weights on a Pool

Description

Set the per-object weight of a Pool.

Usage

```
catboost.pool.set_weight(pool, weight)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>weight</code>	A numeric vector of per-object weights, length equal to <code>nrow(pool)</code> . Default value: Required argument

Value

Nothing. Mutates `pool` in place.

```
catboost.pool.shape
```

Shape of a Pool

Description

Get the (rows, columns) shape of a Pool.

Usage

```
catboost.pool.shape(pool)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
-------------------	--

Value

An integer vector of length 2: `c(num_row, num_col)`.

`catboost.pool.slice` *Slice a Pool*

Description

Return a new Pool containing a contiguous range of rows from `pool`: R equivalent of Python's `Pool.slice()`. Like Python's `Pool.slice()` (which calls `_take_slice()` in `catboost/python-package/catboost/_catboost.pyx`), this builds the new Pool with the core `TDataProvider::GetSubset` machinery via `CatBoostPoolSliceSubset_R` (`src/catboost.cpp`), so all column kinds – numeric, categorical, text and embedding features, every target column, weights, group ids, subgroup ids, baseline, pairs and feature names – are preserved in the sliced Pool.

The only difference from Python is the row selector: Python accepts an arbitrary row-index array (`rindex`), whereas this exposes a contiguous `[offset, offset + size)` range.

Usage

```
catboost.pool.slice(pool, offset, size)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>offset</code>	Zero-based index of the first row to include. Default value: Required argument
<code>size</code>	Number of rows to include. Default value: Required argument

Value

A new `catboost.Pool` object containing the sliced rows.

`catboost.pool.train_eval_split`
Split a Pool into train/eval subsets

Description

R equivalent of Python's `Pool.train_eval_split()`: splits `pool` into a train and (optionally) an eval Pool, with the same shuffle-then-split(-then-stratify) semantics as the Python method. Python's own implementation (`TrainEvalSplit()` in `catboost/python-package/catboost/helpers.py`) lives in the python-package tree (not a core lib) and includes `Python.h`, so it is not directly callable from R; `CatBoostPoolTrainEvalSplit_R` (`src/catboost.cpp`) reimplements its body against the same core entry points it itself calls.

Usage

```
catboost.pool.train_eval_split(
    pool,
    has_time = FALSE,
    is_classification = FALSE,
    eval_fraction = 0.2,
    save_eval_pool = TRUE
)
```

Arguments

<code>pool</code>	A <code>catboost.Pool</code> object. Default value: Required argument
<code>has_time</code>	If <code>TRUE</code> , disables shuffling before the split (preserves row order, e.g. for time-ordered data). Default value: <code>FALSE</code>
<code>is_classification</code>	If <code>TRUE</code> , performs a stratified split using the Pool's label as the class column. Default value: <code>FALSE</code>
<code>eval_fraction</code>	Fraction of rows (in $(0, 1)$) to hold out for eval. Default value: <code>0.2</code>
<code>save_eval_pool</code>	If <code>FALSE</code> , the eval Pool is not built and <code>eval</code> is <code>NULL</code> in the returned list. Default value: <code>TRUE</code>

Value

A named list with elements `train` and `eval` (each a `catboost.Pool`, or `NULL` for `eval` if `save_eval_pool` is `FALSE`).

<code>catboost.predict</code>	<i>Get predictions from a CatBoost model</i>
-------------------------------	--

Description

Get predictions from a CatBoost model on new data.

In case of multiclassification the prediction is returned in the form of a matrix. Each row of this matrix contains the predictions for one row of the input dataset.

Usage

```
## S3 method for class 'catboost.Model'
predict(
  object,
  newdata,
  verbose = FALSE,
  prediction_type = "RawFormulaVal",
  ntree_start = 0,
  ntree_end = 0,
  thread_count = -1
)

catboost.predict(
  model,
  pool,
  verbose = FALSE,
  prediction_type = "RawFormulaVal",
  ntree_start = 0,
  ntree_end = 0,
  thread_count = -1
)
```

Arguments

- | | |
|------------------------|--|
| object | The model obtained as the result of training.
Default value: Required argument |
| newdata | The input data on which to make predictions. Should be a 'catboost.Pool' object.
Default value: Required argument |
| verbose | Verbose output to stdout.
Default value: FALSE (not used) |
| prediction_type | <p>The format for displaying approximated values in output data (see https://catboost.ai/docs/concepts/output-data.html).</p> <p>Possible values:</p> <ul style="list-style-type: none"> • 'Probability' • 'LogProbability' • 'Class' • 'RawFormulaVal' • 'Exponent' • 'RMSEWithUncertainty' <p>Default value: 'RawFormulaVal'</p> <p>Note on R-vs-Python parity: R exposes a single <code>catboost.Model</code> type and a single <code>catboost.predict()</code> entry point for every trained model, always defaulting to 'RawFormulaVal' regardless of the loss function used</p> |

to train it. This intentionally mirrors Python's base `CatBoost.predict()` class, whose default is likewise `'RawFormulaVal'` (`core.py:2932` in the vendored Python package). Python additionally offers separate scikit-learn-style estimator subclasses – `CatBoostClassifier`, `CatBoostRegressor`, `CatBoostRanker` – that have no equivalent object type in R and that override the default independently: `CatBoostClassifier.predict()` defaults to `'Class'` (`core.py:5552`); `CatBoostRegressor.predict()` resolves its default via `_get_default_prediction_type()` (`core.py:6183, 6320-6329`) to `'Exponent'` for Poisson*/Tweedie* losses, `'RMSEWithUncertainty'` for that loss, and `'RawFormulaVal'` otherwise; `CatBoostRanker.predict()` hard-codes `'RawFormulaVal'` and does not accept a `prediction_type` argument at all. Since R has no per-task subclass to hang a different default off of, and since guessing the intended default from the training loss function would silently change output type/shape for existing callers (e.g. returning integer class labels where a numeric score was previously returned), R always defaults to `'RawFormulaVal'` and requires classification/ranking callers to pass `prediction_type` explicitly (e.g. `'Class'` or `'Probability'`) to obtain the same value Python's estimator subclasses return by default.

<code>ntree_start</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing). Default value: 0
<code>ntree_end</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing). Default value: 0 (if value equals to 0 this parameter is ignored and <code>ntree_end</code> equal to <code>tree_count</code>)
<code>thread_count</code>	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: 1
<code>model</code>	The model obtained as the result of training. Default value: Required argument
<code>pool</code>	The input data on which to make predictions. Should be a <code>'catboost.Pool'</code> object. Default value: Required argument

Details

The function `'catboost.predict'` is a synonym for `'predict.catboost.Model'`, which is an S3 method (i.e. called like `'predict(model, newdata)'`).

Value

Vector of predictions (matrix for multi-class classification).

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-predict.html

catboost.predict_log_proba

Predict class log-probabilities

Description

Predict class log-probabilities, R equivalent of Python's `CatBoostClassifier.predict_log_proba()` (`core.py:5655-5697`). A thin wrapper over `catboost.predict` with `prediction_type = "Probability"` followed by `log()` – exactly Python's own implementation (`self._predict(data, 'LogProbability', ...)` resolves, for a binary classifier, to `log()` of the same two-column `[P(class 0), P(class 1)]` representation `predict_proba()` returns), not a separately re-derived computation.

R's `catboost.predict(..., prediction_type = "Probability")` collapses the binary case to a single `P(class 1)` column (see `catboost.predict`), so for a binary model this function reconstructs the two-column `[log(1 - p), log(p)]` matrix Python always returns; for a multiclass model `catboost.predict()` already returns the full probability matrix and `log()` is applied directly.

Usage

```
catboost.predict_log_proba(
  model,
  pool,
  ntree_start = 0,
  ntree_end = 0,
  thread_count = -1
)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>pool</code>	The input dataset. Should be a <code>catboost.Pool</code> object.
<code>ntree_start</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing). Default value: 0
<code>ntree_end</code>	Model is applied on the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing). If value equals to 0 this parameter is ignored and <code>ntree_end</code> equal to <code>tree_count</code> . Default value: 0
<code>thread_count</code>	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Default value: -1

Value

Matrix of per-class log-probabilities (rows = objects, columns = classes).

`catboost.randomized_search`

Randomized search on hyper parameters.

Description

R equivalent of Python's `CatBoost.randomized_search`: calls the same native search entry point (`NCB::RandomizedSearch`, vendored at `catboost/private/libs/hyperparameter_tuning/hyperparameter_tuning`) that Python's `CatBoost._tune_hyperparams` calls via its Cython wrapper. In contrast to `catboost.grid_search`, not all parameter values are tried: a fixed number (`n_iter`) of settings is sampled uniformly from `param_distributions` (the native sampler seeds its shuffle from a fixed constant, so the sampled combinations – not just the winner – are reproducible across calls with the same grid and `n_iter`).

Sampling from continuous distributions (Python's `scipy.stats`-style `rvs()` objects) is not supported: `param_distributions` entries must be plain value vectors, sampled uniformly by the native code.

Usage

```
catboost.randomized_search(
    param_distributions,
    pool,
    params = list(),
    cv = 3,
    n_iter = 10,
    partition_random_seed = 0,
    calc_cv_statistics = TRUE,
    search_by_train_test_split = TRUE,
    refit = TRUE,
    shuffle = TRUE,
    stratified = FALSE,
    train_size = 0.8,
    verbose = TRUE,
    custom_objective = NULL,
    custom_eval_metric_object = NULL
)
```

Arguments

`param_distributions`

Named list of parameter name -> vector of values to sample from, or an unnamed list of such named lists.

Default value: Required argument

<code>pool</code>	Data to search on (a <code>catboost.Pool</code>). Default value: Required argument
<code>params</code>	Fixed parameters for <code>catboost.train</code> , held constant across the search. Default value: <code>list()</code>
<code>cv</code>	Number of cross-validation folds. Default value: 3
<code>n_iter</code>	Number of parameter settings sampled. Default value: 10
<code>partition_random_seed</code>	The random seed used for splitting the data. Default value: 0
<code>calc_cv_statistics</code>	Whether to estimate quality via cross-validation with the found best parameters. Only used when <code>search_by_train_test_split = TRUE</code> . Default value: <code>TRUE</code>
<code>search_by_train_test_split</code>	If <code>TRUE</code> , the dataset is split into train/test parts, candidates are trained on the train part and compared by loss on the test part. If <code>FALSE</code> , every candidate is evaluated with cross-validation instead. Default value: <code>TRUE</code>
<code>refit</code>	If <code>TRUE</code> , fit a model on <code>pool</code> with the best found parameters (via <code>catboost.train</code>) and return it as <code>\$model</code> . Default value: <code>TRUE</code>
<code>shuffle</code>	Shuffle the dataset objects before searching. Default value: <code>TRUE</code>
<code>stratified</code>	Perform stratified sampling for the cross-validation statistics. Unlike Python (which auto-detects this from the loss function), R always defaults to <code>FALSE</code> , matching <code>catboost.cv</code> 's own precedent of leaving stratification to the caller. Default value: <code>FALSE</code>
<code>train_size</code>	Proportion of the dataset used for the train split (used when <code>search_by_train_test_split = TRUE</code>). Default value: 0.8
<code>verbose</code>	Whether to print search progress. Default value: <code>TRUE</code>
<code>custom_objective</code>	A user-defined loss function, used instead of <code>params\$loss_function</code> for both the search itself and (when <code>refit = TRUE</code>) the final refit. See <code>catboost.train</code> 's <code>custom_objective</code> argument for the full contract. Default value: <code>NULL</code> (use <code>params\$loss_function</code> as-is)
<code>custom_eval_metric_object</code>	A user-defined eval metric, used instead of <code>params\$eval_metric</code> for both the search itself and (when <code>refit = TRUE</code>) the final refit. See <code>catboost.train</code> 's <code>custom_eval_metric_object</code> argument for the full contract. Default value: <code>NULL</code> (use <code>params\$eval_metric</code> as-is)

Value

A list with `$params` (best found parameters, as a named list), `$cv_results` (a `data.frame` of cross-validation results with the same columns `catboost.cv` returns), and, if `refit = TRUE`, `$model` (a `catboost.Model` fit with the best parameters).

`catboost.restore_handle`

Restore or complete model handle after de-serializing

Description

After de-serializing a model object through R base's functions (`'readRDS'`, `'load'`), its underlying object will not exist in the computer's memory anymore, and needs to be restored from the raw bytes that the model stores.

This is automatically done internally when calling functions such as `catboost.predict`, but the process is repeated at each call, which makes them slower than if using a fresh model object and increases memory usage inbetween calls to the garbage collector. This function allows restoring the internal object beforehand so as to avoid restoring the object multiple times.

Note that the model object needs to be re-assigned as the output of this function, as the modifications are not done in-place.

Usage

```
catboost.restore_handle(model)
```

Arguments

<code>model</code>	The model obtained as the result of training which has been serialized and is now de-serialized.
--------------------	--

Value

The model object with its handle pointing to a valid object in memory.

`catboost.run_worker` *Run a distributed-training worker.*

Description

R equivalent of the CatBoost CLI's **run-worker** mode: blocks the calling process, acting as a distributed-training worker awaiting commands from a master node over the network, until stopped. There is no R-level stop mechanism beyond what the CLI mode itself provides – killing the process, or the master sending a stop-slave command through the par protocol.

Call this from a separate R process (or session) on each worker machine, *before* starting the master-side training call. On the master side, pass `params = list(node_type = "Master", file_with_hosts = <path>, node_port = <port>)` to `catboost.train` (see its **Distributed training** section) or to `catboost.select_features`. `file_with_hosts` is a plain text file with one `host:port` line per worker, where each `port` matches that worker's own `node_port` below.

Usage

```
catboost.run_worker(node_port, thread_count = parallel::detectCores())
```

Arguments

<code>node_port</code>	TCP port for this worker. Must match this worker's line in the master's <code>file_with_hosts</code> file. Default value: Required argument
<code>thread_count</code>	The number of threads used by this worker. Default value: number of CPU cores (<code>parallel::detectCores()</code>), falling back to 1 when that cannot be determined (returns NA on some platforms).

Value

No value is returned; this call blocks until the worker stops.

See Also

<https://catboost.ai/docs/features/distributed-training.html>

Examples

```
## Not run:
# Run on each worker machine; blocks until killed or stopped by the master.
catboost.run_worker(node_port = 12345)

## End(Not run)
```

```
catboost.sample_gaussian_process
```

Sample from a Gaussian Process posterior

Description

R port of Python's module-level `catboost.sample_gaussian_process`. Implements Gaussian Process posterior sampling (Kernel Gradient Boosting / Algorithm 4) from "Gradient Boosting Performs Gaussian Process Inference" (<https://arxiv.org/abs/2206.05608>). Produces samples from the GP posterior under the prior assumption $f \sim \text{GP}(0, \sigma^2 K + \delta^2 I)$.

Takes a `catboost.Pool` rather than the separate `X/y/ cat_features/text_features/embedding_features` arguments Python's version takes – every other R training entry point (`catboost.train`, `catboost.cv`, ...) is Pool-based, and the Pool already carries that information.

Reproduces the Python algorithm's two-stage fit – a near-zero-signal "prior" model whose leaf values are then randomized by direct JSON model surgery, then a "posterior" model fit to the residual – with the same fitting hyperparameters. Both stages need to fit `learn_pool`'s features against a boosting label other than its own (pure noise for the prior, a residual for the posterior); this package's Pool object has no `set_label` entry point (unlike Python's `X/y` arrays, which Python simply resubmits with a different `y` each stage), so each stage instead builds a throwaway `catboost.Pool` from `learn_pool`'s features (`catboost.pool.get_features`) plus the desired label, carrying over `learn_pool`'s weight (`catboost.pool.get_weight`) and group id (`catboost.pool.get_group_id_hash/ catboost.pool.set_group_id`) if either is set – the same "extract features, rebuild pool with something else changed" idiom already used by `catboost.plot_predictions/ catboost.plot_partial_depe`.

As with those, this restricts `learn_pool` to all-numeric features (`get_features`'s own restriction); an earlier revision instead kept one shared pool and retargeted its label via a temporary `catboost.pool.set_baseline` (which supports categorical/text/ embedding features), but CatBoost's native trainer silently zeroes out `model_shrink_rate` whenever a baseline column is present ("Model shrinkage in combination with baseline column is not implemented yet"), which would silently drop a term the paper's Algorithm 4 requires – not an acceptable substitution. `learn_pool`'s own baseline and pairs (if either is set) are NOT propagated onto the rebuilt Pools and trigger a warning, not a silent drop; group weight/subgroup id have no accessor anywhere in this package to even detect, so they can't be checked either.

RNG note: draws a seeded, deterministic sequence from `random_seed` (default 0), scoped to this call only – the caller's global RNG stream is saved before and restored after, so this function has no observable RNG side effect (unlike Python's local `np.random.default_rng`, R has no non-global RNG object, so this is done by save/restore of `.Random.seed` instead). Reproducible within this package, but not byte-identical to the Python implementation – R has no generator matching numpy's PCG64 `default_rng` bit-for-bit (same caveat as this package's `embedding_processing` LDA calcer; see `catboost.train`).

Usage

```
catboost.sample_gaussian_process(
```

```

learn_pool,
test_pool = NULL,
random_seed = NULL,
samples = 10,
posterior_iterations = 900,
prior_iterations = 100,
learning_rate = 0.1,
depth = 6,
sigma = 0.1,
delta = 0,
random_strength = 0.1,
random_score_type = "Gumbel",
eps = 0.0001,
verbose = FALSE
)

```

Arguments

learn_pool	A catboost.Pool with a single-dimensional label and all-numeric features (no categorical/text/embedding features – the same restriction catboost.pool.get_features itself enforces). Must be non-empty. Default value: Required argument
test_pool	A catboost.Pool or NULL. Validation dataset for metrics/early stopping during posterior training only, as in catboost.train 's own <code>test_pool</code> . Default value: NULL
random_seed	NULL (mapped to 0) or a single integer. Default value: NULL
samples	Number of posterior samples (models) to produce. Range: $[1, +\infty)$. Default value: 10
posterior_iterations	Number of boosting iterations for each posterior model. Range: $[1, +\infty)$. Default value: 900
prior_iterations	Number of boosting iterations for each prior model. Range: $[1, +\infty)$. Default value: 100
learning_rate	Learning rate of the posterior model. Range: $(0, 1]$. Default value: 0.1
depth	Tree depth. Range: $[1, 16]$. Default value: 6
sigma	Scale of the GP kernel (lower values give lower posterior variance). Range: $(0, +\infty)$. Default value: 0.1
delta	Scale of homogeneous noise added to the kernel (label noise). Range: $[0, +\infty)$. Default value: 0

<code>random_strength</code>	Corresponds to beta in the paper; higher values give faster posterior convergence. Range: (0,+inf). Default value: 0.1
<code>random_score_type</code>	Type of random noise added to split scores. Possible values: 'Gumbel' – Gumbel-distributed, as in the paper. 'NormalWithModelSizeDecrease' – normally-distributed, deviation decreasing with model iteration count. Default value: 'Gumbel'
<code>eps</code>	Technical parameter controlling precision of the prior estimate. Range: (0,1]. Default value: 1e-4
<code>verbose</code>	Verbosity of posterior model training output, same convention as <code>catboost.train</code> 's <code>verbose</code> param. The prior model always trains silently. Default value: FALSE

Value

A list of `samples` trained `catboost.Model` objects, each the sum (via `catboost.sum_models`) of that sample's prior and posterior model.

See Also

<https://arxiv.org/abs/2206.05608>

`catboost.save_borders`

Save a model's float feature borders to a file

Description

R equivalent of Python's `model.save_borders(fname)` (`core.py:3809-3820`, `_save_borders()`): writes the model's float feature quantization borders to a file in the format described at <https://catboost.ai/docs/concepts/input-data-custom-borders.html>, so they can be reused to quantize another Pool identically (upstream's `input_borders` mechanism). Unlike `catboost.pool.save_quantization_borders` (which requires an already-quantized Pool), this reads borders directly off a trained model.

Usage

```
catboost.save_borders(model, output_file)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>output_file</code>	Output file path.

Value

No return value, called for side effects.

<code>catboost.save_model</code>	<i>Save the model</i>
----------------------------------	-----------------------

Description

Save the model to a file.

Note: Feature importance (see <https://catboost.ai/docs/concepts/fstr.html#fstr>) is not saved when using this function.

Usage

```
catboost.save_model(
    model,
    model_path,
    file_format = "cbm",
    export_parameters = NULL,
    pool = NULL
)
```

Arguments

<code>model</code>	The model to be saved. Default value: Required argument
<code>model_path</code>	The path to the resulting binary file with the model description. Used for solving other machine learning problems (for instance, applying a model). Default value: Required argument
<code>file_format</code>	specified format model from a file. Possible values: • 'cbm' For catboost binary format • 'coreml' To export into Apple CoreML format • 'onnx' To export into ONNX-ML format • 'pmml' To export into PMML format • 'cpp' To export as C++ code • 'python' To export as Python code. Default value: 'cbm'
<code>export_parameters</code>	are a parameters for CoreML or PMML export.
<code>pool</code>	is training pool.

Value

Status, the result of model shrinking. TRUE if shrinking succeeded, FALSE otherwise.

See Also

<https://catboost.ai/docs/features/export-model-to-core-ml.html>

catboost.save_pool	<i>Save the dataset</i>
--------------------	-------------------------

Description

Save the dataset to the CatBoost format. Files with the following data are created:

- Dataset description
- Column descriptions

Use the catboost.load_pool function to read the resulting files. These files can also be used in the [Command-line version](#) and the [Python library](#).

Usage

```
catboost.save_pool(
    data,
    label = NULL,
    weight = NULL,
    baseline = NULL,
    pool_path = "data.pool",
    cd_path = "cd.pool"
)
```

Arguments

data	A data.frame with features. The following column types are supported: • double • factor. It is assumed that categorical features are given in this type of columns. A standard CatBoost processing procedure is applied to this type of columns: 1. The values are converted to strings. 2. The ConvertCatFeatureToFloat function is applied to the resulting string.
label	Default value: Required argument
weight	The label vector.
baseline	The weights of the label vector.
pool_path	Vector of initial (raw) values of the label function for the object. Used in the calculation of final values of trees.
cd_path	The path to the output file that contains the dataset description.
	The path to the output file that contains the column descriptions.

Value

Nothing. This method writes a dataset to disk.

`catboost.select_features`

Select the best features by recursive elimination.

Description

R equivalent of Python's `CatBoost.select_features`: trains repeatedly while eliminating the weakest features, and reports which features survived.

This calls the same native entry point Python's `select_features` calls (NCB::SelectFeatures, vendored at `catboost/libs/features_selection/select_features.h`), so the elimination loop, its per-step retraining, the SHAP-based feature strengths and the final model all come from vendor code rather than an R-side reimplementation.

This is a different algorithm from `catboost.eval_feature`, which scores caller-supplied feature sets by cross-validation instead of eliminating features.

Feature selection by feature *tags* (Python's `grouping = "ByTags"`) is supported via `grouping/features_tags_for_num_features_tags_to_select`: attach tags to `learn_pool` with `catboost.load_pool`'s `feature_tags` argument, then select among them here.

Usage

```
catboost.select_features(
  learn_pool,
  features_for_select = NULL,
  num_features_to_select = NULL,
  test_pool = NULL,
  params = list(),
  algorithm = NULL,
  steps = NULL,
  shap_calc_type = NULL,
  train_final_model = TRUE,
  custom_eval_metric_object = NULL,
  grouping = NULL,
  features_tags_for_select = NULL,
  num_features_tags_to_select = NULL
)
```

Arguments

`learn_pool` The dataset to select features on (a `catboost.Pool`).
 Default value: Required argument

<code>features_for_select</code>	(for <code>grouping = "Individual"</code>, the default) Which features may be eliminated. A vector of 0-based feature indices or of feature names, or a single string in the CLI's range syntax ("0,2-4,17", both ends of a range inclusive). Vectors are collapsed with commas, matching Python's <code>",".join(map(str, features_for_select))</code>. Must be NULL when <code>grouping = "ByTags"</code>. Default value: Required argument when <code>grouping = "Individual"</code>
<code>num_features_to_select</code>	(for <code>grouping = "Individual"</code>) How many features to keep out of <code>features_for_select</code>. Must be NULL when <code>grouping = "ByTags"</code>. Default value: Required argument when <code>grouping = "Individual"</code>
<code>test_pool</code>	Validation dataset used to measure the loss during elimination (a <code>catboost.Pool</code>), or NULL to measure it on <code>learn_pool</code>. Only one validation dataset is supported. Default value: NULL
<code>params</code>	Parameters for <code>catboost.train</code>. Default value: <code>list()</code>
<code>algorithm</code>	One of "RecursiveByPredictionValuesChange", "RecursiveByLossFunctionChange", "RecursiveByShapValues". NULL leaves the vendor default ("RecursiveByShapValues"). Default value: NULL
<code>steps</code>	How many times a full model is trained during the elimination. More steps give more accurate results. NULL leaves the vendor default (1). Default value: NULL
<code>shap_calc_type</code>	One of "Regular", "Approximate", "Exact". NULL leaves the vendor default ("Regular"). Default value: NULL
<code>train_final_model</code>	Whether to fit a model on the selected features and return it. Default value: TRUE
<code>custom_eval_metric_object</code>	A user-defined eval metric, used instead of <code>params\$eval_metric</code>. See <code>catboost.train</code>'s <code>custom_eval_metric_object</code> argument for the full contract. There is no <code>custom_objective</code> argument here: the underlying native engine (<code>NCB::SelectFeatures</code>, <code>catboost/libs/features_selection/select_features.h</code>) has no custom-objective-descriptor parameter at all (verified against that header and <code>recursive_features_elimination.*</code> – zero <code>TCustomObjectiveDescriptor</code> references), so a <code>custom_objective</code> argument here would be dead: it would have nothing to wire it to. Default value: NULL (use <code>params\$eval_metric</code> as-is)
<code>grouping</code>	One of "Individual" (the default) or "ByTags". "Individual" selects among <code>features_for_select</code>; "ByTags" selects among the tags attached to <code>learn_pool</code> via <code>catboost.load_pool</code>'s <code>feature_tags</code> argument, using <code>features_tags_for_select/num_features_tags_to_select</code> instead. Default value: "Individual"

features_tags_for_select
 (for `grouping = "ByTags"`) A character vector of tag names (from `learn_pool`'s `feature_tags`) that may be eliminated as a group. Must be `NULL` when `grouping = "Individual"`.
 Default value: Required argument when `grouping = "ByTags"`

num_features_tags_to_select
 (for `grouping = "ByTags"`) How many tags to keep out of `features_tags_for_select`. Must be `NULL` when `grouping = "Individual"`.
 Default value: Required argument when `grouping = "ByTags"`

Value

A list with the fields of the vendor's selection summary:

- `selected_features` – 0-based indices of the kept features.
- `selected_features_names` – their names.
- `eliminated_features` – 0-based indices of the dropped features.
- `eliminated_features_names` – their names.
- `selected_features_tags/eliminated_features_tags` – present only when `grouping = "ByTags"`: the kept/dropped tag names.
- `loss_graph` – list with `removed_features_count`, `loss_values` and `main_indices` (the graph points whose loss was measured by fitting a model rather than estimated from `fstr`).
- `model` – the fitted `catboost.Model`, present only when `train_final_model = TRUE`.

`catboost.set_leaf_values`

Set the model's tree leaf values

Description

R equivalent of Python's `model.set_leaf_values(new_leaf_values)` (`core.py:2141-2152`, defined once on `_CatBoostBase` and inherited unchanged by `CatBoost/CatBoostClassifier/CatBoostRegressor/CatBoostRanker`). The change is held in memory only; call `catboost.save_model` to persist it, same calling convention as `catboost.set_scale_and_bias`.

Usage

```
catboost.set_leaf_values(model, new_leaf_values)
```

Arguments

model The model obtained as the result of training.

new_leaf_values
 A numeric vector with new leaf values for all trees. Its length must equal `length(catboost.get_leaf_values(model))` (i.e. `sum(catboost.get_tree_leaf_counts(model)) * ApproxDimension`, not just the leaf count – see `catboost.get_leaf_values`), indexed the same way as `catboost.get_leaf_values`.

Value

No return value, called for side effects.

```
catboost.set_metadata
```

Set model metadata

Description

Set a single string metadata key/value pair on the model, in place. R equivalent of Python's `model.get_metadata()[key] = value` and of the CLI's `metadata set --key --value` mode. The change is held in memory only; call `catboost.save_model` to persist it, matching Python's calling convention.

Usage

```
catboost.set_metadata(model, key, value)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>key</code>	The metadata key name.
<code>value</code>	The metadata value.

Value

No return value, called for side effects.

```
catboost.set_probability_threshold
```

Set the binary-classification probability threshold

Description

Set (or discard) the threshold used for class separation in a binary classification model, R equivalent of Python's `CatBoostClassifier.set_probability_threshold()` (`core.py:5891-5906`). Held in the same model metadata key (`binclass_probability_threshold`) `catboost.get_metadata/` `catboost.set_metadata` expose; the change is in-memory only, same calling convention as `catboost.set_metadata`.

Usage

```
catboost.set_probability_threshold(
    model,
    binclass_probability_threshold = NULL
)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>binclass_probability_threshold</code>	A single numeric value in $[0, 1]$, or <code>NULL</code> to discard a previously-set threshold (falls back to the 0.5 default). Default value: <code>NULL</code>

Value

No return value, called for side effects.

<code>catboost.set_scale_and_bias</code>
<i>Set model scale and bias</i>

Description

Set the model's scale and bias, in place. R equivalent of Python's `model.set_scale_and_bias(scale, bias)` and of the CLI's `normalize-model --set-scale --set-bias` mode. The change is held in memory only; call `catboost.save_model` to persist it, matching Python's calling convention.

Usage

```
catboost.set_scale_and_bias(model, scale, bias)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>scale</code>	The model scale, a single number.
<code>bias</code>	The model bias: a single number, or a numeric vector with one value per model output dimension.

Value

No return value, called for side effects.

catboost.shrink	<i>Shrink the model</i>
-----------------	-------------------------

Description

Shrink the model

Usage

```
catboost.shrink(model, ntree_end, ntree_start = 0)
```

Arguments

<code>model</code>	The model obtained as the result of training.
<code>ntree_end</code>	Leave the trees with indices from the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing).
<code>ntree_start</code>	Leave the trees with indices from the interval <code>[ntree_start, ntree_end)</code> (zero-based indexing).

Value

Status, the result of model shrinking. TRUE if shrinking succeeded, FALSE otherwise.

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-shrink.html

catboost.staged_predict	<i>Apply the model for each tree</i>
-------------------------	--------------------------------------

Description

Apply the model to the given dataset and calculate the results for each *i*-th tree of the model taking into consideration only the trees in the range `[1;i]`.

Peculiarities: In case of multiclassification the prediction is returned in the form of a matrix. Each line of this matrix contains the predictions for one object of the input dataset.

Usage

```
catboost.staged_predict(
    model,
    pool,
    verbose = FALSE,
    prediction_type = "RawFormulaVal",
    ntree_start = 0L,
    ntree_end = 0L,
    eval_period = 1,
    thread_count = -1
)
```

Arguments

model	The model obtained as the result of training. Default value: Required argument
pool	The input dataset. Default value: Required argument
verbose	Verbose output to stdout. Default value: FALSE (not used)
prediction_type	The format for displaying approximated values in output data (see https://catboost.ai/docs/concepts/output-data.html). Possible values: • 'Probability' • 'Class' • 'RawFormulaVal' Default value: 'RawFormulaVal' Note on R-vs-Python parity: same rationale as <code>catboost.predict</code> – R has a single <code>catboost.staged_predict()</code> entry point for every model, always defaulting to 'RawFormulaVal' regardless of loss function, which mirrors Python's base <code>CatBoost.staged_predict()</code> default. Python's <code>CatBoostClassifier.staged_predict()</code> overrides its default to 'Class' (<code>core.py:5699</code>); R does not replicate this because it has no classifier-specific subclass to attach a different default to. Pass <code>prediction_type = 'Class'</code> (or 'Probability') explicitly to match Python's classifier default.
ntree_start	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0
ntree_end	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0 (if value equals to 0 this parameter is ignored and ntree_end equal to tree_count)

eval_period	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 1
thread_count	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: 1

Value

List object with predictions from one iteration.

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-staged_predict.html

catboost.staged_predict_log_proba

Predict class log-probabilities at each stage

Description

Predict class log-probabilities at each iteration, R equivalent of Python's `CatBoostClassifier.staged_predict_log_proba` (core.py:5796-5835). A thin wrapper over `catboost.staged_predict` with `prediction_type = "Probability"`: each iteration's result is `log()`-transformed the same way `catboost.predict_log_proba` transforms a single `catboost.predict()` call.

Usage

```
catboost.staged_predict_log_proba(
    model,
    pool,
    ntree_start = 0L,
    ntree_end = 0L,
    eval_period = 1,
    thread_count = -1
)
```

Arguments

model	The model obtained as the result of training.
pool	The input dataset. Should be a <code>catboost.Pool</code> object.
ntree_start	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 0

ntree_end	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). If value equals to 0 this parameter is ignored and ntree_end equal to tree_count. Default value: 0
eval_period	Model is applied on the interval [ntree_start, ntree_end) with the step eval_period (zero-based indexing). Default value: 1
thread_count	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Default value: -1

Value

Iterator object; each `nextElem()` call returns the matrix of per-class log-probabilities for the corresponding iteration.

`catboost.sum_models` *Sum models.*

Description

Blend trees and counters of two or more trained CatBoost models into a new model. Leaf values can be individually weighted for each input model. For example, it may be useful to blend models trained on different validation datasets.

Usage

```
catboost.sum_models(
    models,
    weights = NULL,
    ctr_merge_policy = "IntersectingCountersAverage"
)
```

Arguments

models	Models for the summation. Default value: Required argument
weights	The weights of the models. Default value: NULL (use weight 1 for every model)
ctr_merge_policy	The counters merging policy. Possible values: • 'FaillfCtrIntersects' Ensure that the models have zero intersecting counters • 'LeaveMostDiversifiedTable' Use the most diversified counters by the count of unique hash values • 'IntersectingCountersAverage' Use the average ctr counter values in the intersecting bins Default value: 'IntersectingCountersAverage'

Value

Model object.

<code>catboost.Tokenizer</code>	<i>Create text Tokenizer</i>
---------------------------------	------------------------------

Description

Splits a string into tokens ahead of `catboost.Dictionary` fitting/application. Native bridge to Python's `catboost.text_processing.Tokenizer`, backed directly by vendor's `NTextProcessing::NTokenizer::TTokenizer`.

Usage

```
catboost.Tokenizer(
    lowercasing = NULL,
    number_process_policy = NULL,
    number_token = NULL,
    separator_type = NULL,
    delimiter = NULL,
    split_by_set = NULL,
    skip_empty = NULL,
    token_types = NULL,
    sub_tokens_policy = NULL,
    languages = NULL,
    lemmatizing = NULL
)
```

Arguments

<code>lowercasing</code>	Lowercase each token. Default value: <code>FALSE</code>
<code>number_process_policy</code>	One of "Skip", "LeaveAsIs", "Replace". Default value: "LeaveAsIs"
<code>number_token</code>	Replacement token used when <code>number_process_policy</code> is "Replace". Default value: " " (matches vendor default)
<code>separator_type</code>	Tokenization method: "ByDelimiter" or "BySense". Default value: "ByDelimiter"
<code>delimiter</code>	Delimiter string used to split tokens (ByDelimiter mode). Default value: " "
<code>split_by_set</code>	If <code>TRUE</code> , each individual character in <code>delimiter</code> is treated as its own delimiter (ByDelimiter mode). Default value: <code>FALSE</code>

<code>skip_empty</code>	Skip empty tokens adjacent to delimiters (ByDelimiter mode). Default value: TRUE
<code>token_types</code>	Character vector of token types kept after tokenization (BySense mode). Possible values: "Word", "Digit", "Punctuation", "SentenceBreak", "ParagraphBreak", "Unknown". Default value: NULL (vendor default: c("Word", "Digit", "Unknown"))
<code>sub_tokens_policy</code>	Subtoken processing policy (BySense mode): one of "SingleToken", "SeveralTokens". Default value: "SingleToken"
<code>languages</code>	Character vector of language names used for lemmatizing/BySense tokenization. Default value: NULL (all languages)
<code>lemmatizing</code>	Apply lemmatization to tokens. Default value: FALSE

Value

catboost.Tokenizer

`catboost.tokenizer.tokenize`

Tokenize a string

Description

Split `string` into tokens using a `catboost.Tokenizer`.

Usage

```
catboost.tokenizer.tokenize(tokenizer, string, types = FALSE)
```

Arguments

<code>tokenizer</code>	A <code>catboost.Tokenizer</code> object. Default value: Required argument
<code>string</code>	Input string. Default value: Required argument
<code>types</code>	If TRUE, also return token types. Default value: FALSE

Value

A character vector of tokens, or (if `types = TRUE`) a `data.frame` with `token` and `type` columns.

catboost.train	<i>Train the model</i>
----------------	------------------------

Description

Train the model using a CatBoost dataset.

The list of parameters

- Common parameters
 - fold_permutation_block

Objects in the dataset are grouped in blocks before the random permutations. This parameter defines the size of the blocks. The smaller is the value, the slower is the training. Large values may result in quality degradation.

Default value:

Default value differs depending on the dataset size and ranges from 1 to 256 inclusively
 - ignored_features

Identifiers of features to exclude from training. The non-negative indices that do not match any features are successfully ignored. For example, if five features are defined for the objects in the dataset and this parameter is set to "42", the corresponding non-existing feature is successfully ignored.

The identifier corresponds to the feature's index. Feature indices used in train and feature importance are numbered from 0 to featureCount-1. If a file is used as input data then any non-feature column types are ignored when calculating these indices. For example, each row in the input file contains data in the following order: "categorical feature<\t>label<\t>numerical feature". So for the row "rock<\t>0<\t>42", the identifier for the "rock" feature is 0, and for the "42" feature it is 1.

The identifiers of features to exclude should be enumerated at vector.

For example, if training should exclude features with the identifiers 1, 2, 7, 42, 43, 44, 45, the value of this parameter should be set to c(1,2,7,42,43,44,45).

Default value:

None (use all features)
 - use_best_model

If this parameter is set, the number of trees that are saved in the resulting model is defined as follows:

Build the number of trees defined by the training parameters.

 - * Identify the iteration with the optimal loss function value.
 - * No trees are saved after this iteration.

This option requires a test dataset to be provided.

Default value:

FALSE (not used)
 - loss_function

The loss function (see <https://catboost.ai/docs/concepts/loss-functions.html#loss-functions>) to use in training. The specified value also determines the machine learning problem to solve.

Format:

<Loss function 1>[:<parameter 1>=<value>...<parameter N>=<value>:]

Supported loss functions:

- * 'Logloss'
- * 'CrossEntropy'
- * 'MultiClass'
- * 'MultiClassOneVsAll'
- * 'RMSE'
- * 'MAE'
- * 'Quantile'
- * 'LogLinQuantile'
- * 'MAPE'
- * 'Poisson'
- * 'Lq'
- * 'PairLogit'
- * 'PairLogitPairwise'
- * 'YetiRank'
- * 'YetiRankPairwise'
- * 'QueryCrossEntropy'
- * 'QueryRMSE'
- * 'QuerySoftMax'

Supported parameters:

- * alpha - The coefficient used in quantile-based losses ('Quantile' and 'LogLinQuantile'). The default value is 0.5.
For example, if you need to calculate the value of Quantile with the coefficient $\alpha = 0.1$, use the following construction:
'Quantile:alpha=0.1'

Default value:

'RMSE'

— custom_loss

Loss function (see <https://catboost.ai/docs/concepts/loss-functions.html#loss-functions>) values to output during training. These functions are not used for optimization and are displayed for informational purposes only.

Format:

c(<Loss function 1>[:<parameter>=<value>],<Loss function 2>[:<parameter>=<value>],...,<Loss function N>[:<parameter>=<value>])

Supported loss functions:

- * 'Logloss'
- * 'CrossEntropy'
- * 'Precision'
- * 'Recall'
- * 'F1'
- * 'F'
- * 'BalancedAccuracy'

- * 'BalancedErrorRate'
- * 'MCC'
- * 'Accuracy'
- * 'CtrFactor'
- * 'AUC'
- * 'BrierScore'
- * 'HingeLoss'
- * 'HammingLoss'
- * 'ZeroOneLoss'
- * 'Kappa'
- * 'WKappa'
- * 'LogLikelihoodOfPrediction'
- * 'MultiClass'
- * 'MultiClassOneVsAll'
- * 'TotalF1'
- * 'MAE'
- * 'MAPE'
- * 'Poisson'
- * 'Quantile'
- * 'RMSE'
- * 'LogLinQuantile'
- * 'Lq'
- * 'NumErrors'
- * 'SMAPE'
- * 'R2'
- * 'MSLE'
- * 'MedianAbsoluteError'
- * 'PairLogit'
- * 'PairLogitPairwise'
- * 'PairAccuracy'
- * 'QueryCrossEntropy'
- * 'QueryRMSE'
- * 'QuerySoftMax'
- * 'PFound'
- * 'NDCG'
- * 'AverageGain'
- * 'PrecisionAt'
- * 'RecallAt'
- * 'MAP'
- * 'MRR'
- * 'ERR'

Supported parameters:

- * alpha - The coefficient used in quantile-based losses ('Quantile' and 'LogLinQuantile'). The default value is 0.5.

For example, if you need to calculate the value of CrossEntropy and Quantile with the coefficient $\alpha = 0.1$, use the following construction:

c('CrossEntropy') or simply 'CrossEntropy'.

Values of all custom loss functions for learning and test datasets are saved to the Loss function (see https://catboost.ai/docs/concepts/output-data_loss-function.html#output-data_loss-function) output files (learn_error.tsv and test_error.tsv respectively). The catalog for these files is specified in the train-dir (train_dir) parameter.

Default value:

None (use one of the loss functions supported by the library)

– eval_metric

The loss function used for overfitting detection (if enabled) and best model selection (if enabled).

Supported loss functions:

- * 'Logloss'
- * 'CrossEntropy'
- * 'Precision'
- * 'Recall'
- * 'F1'
- * 'F'
- * 'BalancedAccuracy'
- * 'BalancedErrorRate'
- * 'MCC'
- * 'Accuracy'
- * 'CtrFactor'
- * 'AUC'
- * 'BrierScore'
- * 'HingeLoss'
- * 'HammingLoss'
- * 'ZeroOneLoss'
- * 'Kappa'
- * 'WKappa'
- * 'LogLikelihoodOfPrediction'
- * 'MultiClass'
- * 'MultiClassOneVsAll'
- * 'TotalF1'
- * 'MAE'
- * 'MAPE'
- * 'Poisson'
- * 'Quantile'
- * 'RMSE'
- * 'LogLinQuantile'

```

* 'Lq'
* 'NumErrors'
* 'SMAPE'
* 'R2'
* 'MSLE'
* 'MedianAbsoluteError'
* 'PairLogit'
* 'PairLogitPairwise'
* 'PairAccuracy'
* 'QueryCrossEntropy'
* 'QueryRMSE'
* 'QuerySoftMax'
* 'PFound'
* 'NDCG'
* 'AverageGain'
* 'PrecisionAt'
* 'RecallAt'
* 'MAP'
* 'MRR'
* 'ERR'

```

Format:

metric_name:param=Value

Examples:

```
'R2'
```

```
'Quantile:alpha=0.3'
```

Default value:

Optimized objective is used

– iterations

The maximum number of trees that can be built when solving machine learning problems.

When using other parameters that limit the number of iterations, the final number of trees may be less than the number specified in this parameter.

Default value:

1000

– border

The target border. If the value is strictly greater than this threshold, it is considered a positive class. Otherwise it is considered a negative class.

The parameter is obligatory if the Logloss function is used, since it uses borders to transform any given target to a binary target.

Used in binary classification.

Default value:

0.5

– leaf_estimation_iterations

The number of gradient steps when calculating the values in leaves.

Default value:

1

- depth
Depth of the trees.
The value can be any integer up to 16. It is recommended to use values in the range [1; 10].
Default value:
6
- learning_rate
The learning rate.
Used for reducing the gradient step.
Default value:
0.03
- rsm
Random subspace method. The percentage of features to use at each iteration of building trees. At each iteration, features are selected over again at random.
The value must be in the range [0;1].
Default value:
1
- random_seed
The random seed used for training.
Default value:
0
- nan_mode
Way to process missing values.
Possible values:
 - * 'Min'
 - * 'Max'
 - * 'Forbidden'Default value:
'Min'
- od_pval
Use the Overfitting detector (see <https://catboost.ai/docs/concepts/overfitting-detector.html#overfitting-detector>) to stop training when the threshold is reached.
Requires that a test dataset was input.
For best results, it is recommended to set a value in the range $[10^{-10}; 10^{-2}]$.
The larger the value, the earlier overfitting is detected.
Default value:
The overfitting detection is turned off
- od_type
The method used to calculate the values in leaves.
Possible values:
 - * IncToDec
 - * IterRestriction. Do not specify the overfitting detector threshold when using the Iter type.
Default value:
'IncToDec'

- `od_wait`
The number of iterations to continue the training after the iteration with the optimal loss function value. The purpose of this parameter differs depending on the selected overfitting detector type:
 - * `IncToDec` - Ignore the overfitting detector when the threshold is reached and continue learning for the specified number of iterations after the iteration with the optimal loss function value.
 - * `Iter` - Consider the model overfitted and stop training after the specified number of iterations since the iteration with the optimal loss function value.
 Default value:
20
- `leaf_estimation_method`
The method used to calculate the values in leaves.
Possible values:
 - * `Newton`
 - * `Gradient`
 Default value:
Default value depends on the selected loss function
- `grow_policy`
GPU only. The tree growing policy. It describes how to perform greedy tree construction.
Possible values:
 - * `SymmetricTree`
 - * `Lossguide`
 - * `Depthwise`
 Default value:
`SymmetricTree`
- `min_data_in_leaf`
GPU only. The minimum training samples count in leaf. CatBoost will not search for new splits in leaves with samples count less than `min_data_in_leaf`. This parameter is used only for `Depthwise` and `Lossguide` growing policies.
Default value:
1
- `max_leaves`
GPU only. The maximum leaf count in resulting tree. Used only for `Lossguide` growing policy. This parameter is used only for `Lossguide` growing policy.
Default value:
31
- `score_function` GPU only. Score that is used during tree construction to select the next tree split.
Possible values:
 - * `L2`
 - * `Cosine`
 - * `NewtonL2`
 - * `NewtonCosine`

- Default value:
Cosine
For growing policy Lossguide default is NewtonL2.
- l2_leaf_reg
L2 regularization coefficient. Used for leaf value calculation.
Any positive values are allowed.
Default value:
3
- model_size_reg
Model size regularization coefficient. The influence coefficient of the model size for choosing tree structure. To get a smaller model size - increase this coefficient.
Any positive values are allowed.
Default value:
0.5
- has_time
Use the order of objects in the input data (do not perform a random permutation of the dataset at the preprocessing stage)
Default value:
FALSE (not used; permute input dataset)
- allow_const_label
To allow the constant label value in the dataset.
Default value:
FALSE
- name
The experiment name to display in visualization tools (see <https://catboost.ai/docs/features/visualization.html#visualization>).
Default value:
experiment
- prediction_type
The format for displaying approximated values in output data.
Possible values:
 - * 'Probability'
 - * 'Class'
 - * 'RawFormulaVal'
 Default value:
'RawFormulaVal'
- fold_len_multiplier
Coefficient for changing the length of folds.
The value must be greater than 1. The best validation result is achieved with minimum values.
With values close to 1 (for example, $1 + \epsilon$), each iteration takes a quadratic amount of memory and time for the number of objects in the iteration. Thus, low values are possible only when there is a small number of objects.
Default value:
2
- class_weights
Classes weights. The values are used as multipliers for the object weights.

For example, for 3 class classification you could use:

`c(0.85, 1.2, 1)`

Default value:

None (the weight for all classes is set to 1)

- `classes_count`
The upper limit for the numeric class label. Defines the number of classes for multiclassification.
Only non-negative integers can be specified. The given integer should be greater than any of the target values.
If this parameter is specified the labels for all classes in the input dataset should be smaller than the given value.
Default value:
maximum class label + 1
- `one_hot_max_size`
Convert the feature to float if the number of different values that it takes exceeds the specified value. Ctrs are not calculated for such features.
The one-vs.-all delimiter is used for the resulting float features.
Default value:
FALSE
Do not convert features to float based on the number of different values
- `random_strength`
Score standard deviation multiplier.
Default value:
1
- `bootstrap_type`
Bootstrap type. Defines the method for sampling the weights of documents.
Possible values:
 - * 'Bayesian'
 - * 'Bernoulli'
 - * 'Poisson'
 - * 'MVS'
 - * 'No'
 Poisson bootstrap is supported only on GPU.
Default value:
'Bayesian'
- `bagging_temperature`
Controls intensity of Bayesian bagging. The higher the temperature the more aggressive bagging is.
Typical values are in the range $[0, 1]$ (0 is for no bagging).
Possible values are in the range $[0, +\infty)$.
Default value:
1
- `subsample`
Sample rate for bagging. This parameter can be used if one of the following bootstrap types is defined:
 - * 'Bernoulli'

- Default value:
0.66
 - `sampling_unit`
The parameter allows to specify the sampling scheme: sample weights for each object individually or for an entire group of objects together.
Possible values:
 - * 'Object'
 - * 'Group'
Default value:
'Object'
 - `sampling_frequency`
Frequency to sample weights and objects when building trees.
Possible values:
 - * 'PerTree'
 - * 'PerTreeLevel'
Default value:
'PerTreeLevel'
 - `model_shrink_rate`
For $i > 0$ at the start of i -th iteration multiplies model by $(1 - \text{model_shrink_rate} / i)$.
Possible values: $[0, 1)$.
Default value: 0
- CTR settings
 - `simple_ctr`
Binarization settings for categorical features (see https://catboost.ai/docs/concepts/algorithm-main-stages_cat-to-numeric.html).
Format:
`c(CtrType[:TargetBorderCount=BorderCount][:TargetBorderType=BorderType][:CtrBorderCount`
Components:
 - * CTR types for training on CPU:
 - 'Borders'
 - 'Buckets'
 - 'BinarizedTargetMeanValue'
 - 'Counter'
 - * CTR types for training on GPU:
 - 'Borders'
 - 'Buckets'
 - 'FeatureFreq'
 - 'FloatTargetMeanValue'
 - * The number of borders for label value binarization. (see <https://catboost.ai/docs/concepts/quantization.html>) Only used for regression problems. Allowed values are integers from 1 to 255 inclusively. The default value is 1. This option is available for training on CPU only.
 - * The binarization (see <https://catboost.ai/docs/concepts/quantization.html>) type for the label value. Only used for regression problems.
Possible values:

- 'Median'
- 'Uniform'
- 'UniformAndQuantiles'
- 'MaxLogSum'
- 'MinEntropy'
- 'GreedyLogSum'

By default, 'MinEntropy' This option is available for training on CPU only.

- * The number of splits for categorical features. Allowed values are integers from 1 to 255 inclusively.

- * The binarization type for categorical features. Supported values for training on CPU:

- 'Uniform'

Supported values for training on GPU:

- 'Median'
- 'Uniform'
- 'UniformAndQuantiles'
- 'MaxLogSum'
- 'MinEntropy'
- 'GreedyLogSum'

- * Priors to use during training (several values can be specified) Possible formats:

- 'One number - Adds the value to the numerator.'
- 'Two slash-delimited numbers (for GPU only) - Use this format to set a fraction. The number is added to the numerator and the second is added to the denominator.'

– combinations_ctr

Binarization settings for combinations of categorical features (see https://catboost.ai/docs/concepts/algorithm-main-stages_cat-to-numeric.html).

Format:

c(CtrType[:TargetBorderCount=BorderCount][:TargetBorderType=BorderType][:CtrBorderCount

Components:

- * CTR types for training on CPU:

- 'Borders'
- 'Buckets'
- 'BinarizedTargetMeanValue'
- 'Counter'

- * CTR types for training on GPU:

- 'Borders'
- 'Buckets'
- 'FeatureFreq'
- 'FloatTargetMeanValue'

- * The number of borders for target binarization. (see <https://catboost.ai/docs/concepts/quantization.html>) Only used for regression problems. Allowed values are integers from 1 to 255 inclusively. The default value is 1. This option is available for training on CPU only.

- * The binarization (see <https://catboost.ai/docs/concepts/quantization.html>) type for the target. Only used for regression problems.
Possible values:
 - 'Median'
 - 'Uniform'
 - 'UniformAndQuantiles'
 - 'MaxLogSum'
 - 'MinEntropy'
 - 'GreedyLogSum'
 By default, 'MinEntropy' This option is available for training on CPU only.
- * The number of splits for categorical features. Allowed values are integers from 1 to 255 inclusively.
- * The binarization type for categorical features. Supported values for training on CPU:
 - 'Uniform'
 Supported values for training on GPU:
 - 'Median'
 - 'Uniform'
 - 'UniformAndQuantiles'
 - 'MaxLogSum'
 - 'MinEntropy'
 - 'GreedyLogSum'
- * Priors to use during training (several values can be specified) Possible formats:
 - 'One number - Adds the value to the numerator.'
 - 'Two slash-delimited numbers (for GPU only) - Use this format to set a fraction. The number is added to the numerator and the second is added to the denominator.'
- ctr_target_border_count
Maximum number of borders used in target binarization for categorical features that need it. If TargetBorderCount is specified in 'simple_ctr', 'combinations_ctr' or 'per_feature_ctr' option it overrides this value.
Default value:
1
- counter_calc_method
The method for calculating the Counter CTR type for the test dataset.
Possible values:
 - * 'Full'
 - * 'FullTest'
 - * 'PrefixTest'
 - * 'SkipTest'
 Default value: 'PrefixTest'
- max_ctr_complexity
The maximum number of categorical features that can be combined.
Default value:
4

- `ctr_leaf_count_limit`
 The maximum number of leaves with categorical features. If the number of leaves exceeds the specified limit, some leaves are discarded. The value must be positive (for zero limit use `ignored_features` parameter).
 The leaves to be discarded are selected as follows:
 1. The leaves are sorted by the frequency of the values.
 2. The top N leaves are selected, where N is the value specified in the parameter.
 3. All leaves starting from N+1 are discarded.
 This option reduces the resulting model size and the amount of memory required for training. Note that the resulting quality of the model can be affected.
 Default value:
 None (The number of leaves with categorical features is not limited)
 - `store_all_simple_ctr`
 Ignore categorical features, which are not used in feature combinations, when choosing candidates for exclusion.
 Use this parameter with `ctr-leaf-count-limit` only.
 Default value:
 FALSE (Both simple features and feature combinations are taken in account when limiting the number of leaves with categorical features)
- Binarization settings
 - `border_count`
 The number of splits for numerical features. Allowed values are integers from 1 to 255 inclusively.
 Default value:
 254 for training on CPU or 128 for training on GPU
 - `feature_border_type`
 The binarization mode (see <https://catboost.ai/docs/concepts/quantization.html>) for numerical features.
 Possible values:
 - * 'Median'
 - * 'Uniform'
 - * 'UniformAndQuantiles'
 - * 'MaxLogSum'
 - * 'MinEntropy'
 - * 'GreedyLogSum'
 Default value:
 'MinEntropy'
- Performance settings
 - `thread_count`
 The number of threads to use when applying the model.
 Allows you to optimize the speed of execution. This parameter doesn't affect results.
 Default value:
 The number of CPU cores.
- Output settings

- logging_level
Possible values:
 - * 'Silent'
 - * 'Verbose'
 - * 'Info'
 - * 'Debug'Default value:
'Silent'
- metric_period
The frequency of iterations to print the information to stdout. The value should be a positive integer.
Default value:
1
- train_dir
The directory for storing the files generated during training.
Default value:
None (current catalog)
- save_snapshot
Enable snapshotting for restoring the training progress after an interruption.
Default value:
None
- snapshot_file
Settings for recovering training after an interruption (see <https://catboost.ai/docs/features/snapshots.html>).
Depending on whether the file specified exists in the file system:
 - * Missing - write information about training progress to the specified file.
 - * Exists - load data from the specified file and continue training from where it left off.Default value:
File can't be generated or read. If the value is omitted, the file name is experiment.cbsnapshot.
- snapshot_interval
Interval between saving snapshots (seconds)
Default value:
600
- allow_writing_files
If this flag is set to FALSE, no files with different diagnostic info will be created during training. With this flag set to FALSE no snapshotting can be done. Plus visualisation will not work, because visualisation uses files that are created and updated during training.
Default value:
TRUE
- approx_on_full_history
If this flag is set to TRUE, each approximated value is calculated using all the preceding rows in the fold (slower, more accurate). If this flag is set to FALSE, each approximated value is calculated using only the beginning 1/fold_len_multiplier fraction of the fold (faster, slightly less accurate).

- Default value:
FALSE
- `boosting_type`
Boosting scheme. Possible values: - 'Ordered' - Gives better quality, but may slow down the training. - 'Plain' - The classic gradient boosting scheme. May result in quality degradation, but does not slow down the training.
Default value:
Depends on object count and feature count in train dataset and on learning mode.
 - `dev_score_calc_obj_block_size`
CPU only. Size of block of samples in score calculation. Should be > 0 Used only for learning speed tuning. Changing this parameter can affect results in pairwise scoring mode due to numerical accuracy differences
Default value:
5000000
 - `dev_efb_max_buckets`
CPU only. Maximum bucket count in exclusive features bundle. Should be in an integer between 0 and 65536. Used only for learning speed tuning.
Default value:
1024
 - `sparse_features_conflict_fraction`
CPU only. Maximum allowed fraction of conflicting non-default values for features in exclusive features bundle. Should be a real value in $[0, 1)$ interval.
Default value:
0.0
 - `leaf_estimation_backtracking`
Type of backtracking during gradient descent. Possible values: - 'No' - never backtrack; supported on CPU and GPU - 'AnyImprovement' - reduce the descent step until the value of loss function is less than before the step; supported on CPU and GPU - 'Armijo' - reduce the descent step until Armijo condition is satisfied; supported on GPU only
Default value:
'AnyImprovement'

Usage

```
catboost.train(
    learn_pool,
    test_pool = NULL,
    params = list(),
    init_model = NULL,
    custom_objective = NULL,
    custom_eval_metric_object = NULL
)
```

Arguments

`learn_pool` The dataset used for training the model.
Default value: Required argument

<code>test_pool</code>	The dataset used for testing the quality of the model. Default value: NULL (not used)
<code>params</code>	The list of parameters to start training with. If omitted, default values are used (see The list of parameters). If set, the passed list of parameters overrides the default values. Default value: Required argument
<code>init_model</code>	Continue training starting from an existing model. Accepts a <code>catboost.Model</code> object (as returned by <code>catboost.train</code> or <code>catboost.load_model</code>), or a string/path to a model file on disk (loaded via <code>catboost.load_model</code> with the default "cbm" format). Default value: NULL (train a new model from scratch)

custom_objective

A user-defined loss function, used instead of `params$loss_function`. Must be a named `list` with one or both of the following elements (both are R closures; at least one is required):

- `calc_ders_range = function(approx, target, weight)` Called for single-dimension losses (regression, binary classification, ranking). `approx/target` are numeric vectors of the current batch's predictions/labels; `weight` is either a numeric vector of the same length or NULL (unweighted pool). Must return an $N \times 2$ numeric matrix ($N = \text{length}(\text{approx})$), column 1 = first derivative, column 2 = second derivative of the loss w.r.t. `approx`, one row per observation.
- `calc_ders_multi = function(approx, target, weight)` Called for multi-dimensional losses (multiclass, multi-target regression), once per observation. `approx` is a numeric vector of length K (the current observation's per-dimension predictions); `target` is a numeric vector (length 1 for multiclass, length K for multi-target); `weight` is a single number. Must return `list(der1 = <numeric vector, length K>, der2 = <K x K numeric matrix, or NULL>)`: `der1` is the gradient; `der2`, when required by the current `leaf_estimation_method`, is the Hessian (a symmetric matrix – only its upper triangle, including the diagonal, is read).

Reimplements the two-method contract of Python's `CatBoost(loss_function = <object with calc_ders_range/ calc_ders_multi>)` (see `_catboost.pyx`'s `_ObjectiveCalcDersRange/_ObjectiveCalcDersMultiClass/_ObjectiveCalcDersMultiTarget` adapted to R's lack of a scalar/length-1-vector distinction. Every call is marshaled back onto R's main thread (via the package's internal call-back bridge) so it is safe to call from a multi-threaded training run; `params$thread_count` is NOT forced to 1.

`params$loss_function` is set automatically to "PythonUserDefinedPerObject" (covers `calc_ders_range` and `calc_ders_multi` used for MultiClass-shaped losses) when left unset; set it explicitly to "PythonUserDefinedMultiTarget" yourself if `calc_ders_multi` implements a multi-target regression objective (vector target) instead – any other value is rejected. A custom objective also requires `params$eval_metric` to be set explicitly (there is no default metric to infer from an opaque R closure), and is incompatible

with `params$boost_from_average = TRUE` (native CatBoost only computes that data-dependent starting bias for a fixed allowlist of built-in losses).

Default value: `NULL` (use `params$loss_function` as-is)

`custom_eval_metric_object`

A user-defined eval metric, used instead of `params$eval_metric`. Must be a named list with:

- `evaluate = function(approx, target, weight)` Required. Called once per batch. `approx` is a numeric $N \times D$ matrix (N = batch size, D = number of approx dimensions – $D == 1$ for single-dimension losses). `target` is a numeric vector of length N (single-target metrics, `multi_target` unset/`FALSE`) or a numeric $N \times K$ matrix (multi-target metrics, `multi_target = TRUE`). `weight` is a numeric vector of length N , or `NULL` (unweighted pool). Must return `list(error = <numeric scalar>, weight = <numeric scalar>)`: the batch's weighted error sum and weight sum – CatBoost itself sums these across batches/folds.
- `is_max_optimal = function()` Required. Returns `TRUE/FALSE`: whether a larger metric value is better.
- `get_final_error = function(error)` Optional. `error` is the numeric length-2 vector `c(sum_error, sum_weight)` accumulated across all batches (the same two numbers `evaluate` reported, summed). Must return the final scalar metric value. Defaults to `sum_error / sum_weight` (or 0 if `sum_weight == 0`) when omitted, matching native `IMetric`'s own default (see `vendor/catboost/catboost/libs/metrics/metric.cpp`'s `TMetric::GetFinalError`).
- `is_additive = function()` Optional. Returns `TRUE/FALSE`: whether per-batch `error/weight` sums can be combined by plain addition. Defaults to `FALSE` when omitted (conservative; matches Python's own default).
- `multi_target` Optional logical scalar, default `FALSE`. Set `TRUE` if `evaluate`'s `target` argument is a multi-column matrix (a multi-target regression metric) rather than a single vector.

Reimplements the four-method contract of Python's `CustomMetric/MultiTargetCustomMetric` (`_catboost.pyx`'s `evaluate/is_max_optimal/get_final_error/is_additive`), adapted to R's lack of a class hierarchy: Python dispatches single- vs multi-target `evaluate` by subclassing `MultiTargetCustomMetric`; R uses the explicit `multi_target` list element instead. Every call is marshaled back onto R's main thread via the same callback bridge `custom_objective` uses (shared, not a second queue); `params$thread_count` is NOT forced to 1.

`params$eval_metric` is set automatically to `"PythonUserDefinedPerObject"` (`multi_target` unset/`FALSE`) or `"PythonUserDefinedMultiTarget"` (`multi_target = TRUE`) when left unset; any other value is rejected. This argument is a deliberate R-vs-Python API divergence: Python passes a metric object directly through `eval_metric = <object>`, while R keeps `eval_metric` as its existing native string/params-list key (see `custom_metric`, a *different*, pre-existing native parameter for reporting extra built-in metrics –

not a callback) and adds this separate formal argument instead, to avoid a naming collision between the two.

Default value: NULL (use `params$eval_metric` as-is)

Details

Custom objective / custom eval metric support matrix. Every entry point below shares the same `custom_objective/custom_eval_metric_object` argument contract documented under `catboost.train`'s own `custom_objective` and `custom_eval_metric_object` params:

- `catboost.train` Both `custom_objective` and `custom_eval_metric_object`.
- `catboost.cv` Both.
- `catboost.grid_search` Both – forwarded to the search itself and, when `refit = TRUE`, to the refit.
- `catboost.randomized_search` Both – same forwarding as `catboost.grid_search`.
- `catboost.eval_feature` Both.
- `catboost.select_features` **custom_eval_metric_object only**. There is no `custom_objective` argument: the native entry point it calls, `NCB::SelectFeatures` (vendored at `catboost/libs/features_select`) takes no `TCustomObjectiveDescriptor` parameter anywhere in its signature or in `recursive_features_elimination.*` – there is no C++-side slot to wire a custom objective into, so the omission is structural, not a missing feature.

Multithreading preservation. CatBoost's own engine parallelizes derivative/metric computation across TBB worker threads (`NPar::TTbbLocalExecutor`), but R's C API may only be called from R's own main thread. The naive fix – forcing `params$thread_count = 1` whenever a custom objective/metric is supplied – was rejected: it would silently kill CatBoost's own training parallelism for every custom-loss user, trading a correctness problem for a performance regression. Instead, training runs on a background `std::thread` while R's real main thread (the `.Call` frame the user invoked) runs a drain loop that services R-closure-invocation requests queued by any worker thread; each request blocks its own thread on a condition variable until the main thread has executed it and posted the result back (`src/r_callback_bridge.h`). TBB's own parallel scheduling of the surrounding derivative/metric computation is untouched – only the R closure invocation itself is serialized onto the main thread, not the whole computation. `params$thread_count` is therefore never forced to 1 for a custom objective/metric run.

Ctrl-C during a bridged run. Detecting the interrupt on the drain loop's polling thread consumes it – R clears its pending-interrupt flag before the drain loop observes it, so nothing is left for R to deliver afterwards. Ctrl-C during a custom-objective/metric run therefore surfaces as an ordinary R error, not an R interrupt condition: `tryCatch(..., interrupt = )` will not fire on it. Restoring true interrupt semantics is tracked as a follow-up, not pursued here due to CRAN R CMD check NOTE risk in the required mechanism.

Distributed training. `params` accepts the native `node_type`, `node_port` and `file_with_hosts` keys used to run multi-machine training: start a worker process on each remote machine with `catboost.run_worker`, then set `node_type = "Master"`, `file_with_hosts = <path to a "host:port"-per-line file>` and `node_port = <the par-framework port on this machine>` in the master-side `params` list. Setting any `node_type` other than `"SingleHost"` routes the call to the native distributed training engine (the same one the command-line

client's `--node-type Master` fit uses); the default single-host path is untouched by this and behaves exactly as before.

Distributed training shards the dataset across the workers and builds each split's histograms per shard, so a distributed model is not bit-for-bit identical to a single-host model trained on the same data with the same seed – it is statistically equivalent, not reproducible against it. `node_type = "Master"` requires `task_type = "CPU"` (the default); it is rejected for GPU training.

Value

Model object.

Full Parameter Reference (generated)

All 139 hyperparameters from the machine-generated capability inventory, and how catboost accepts each one.

Unrecognized `params` keys are rejected with an error naming the offending key(s). To use a newer vendor-core parameter not yet listed below, set `options(catboost.allow_unknown_params = TRUE)` to bypass this check.

X Supplied via `learn_pool/test_pool` (see `catboost.load_pool`), not the `params` list.

allow_const_label Native training parameter accepted in the `params` list.

allow_writing_files Native training parameter accepted in the `params` list.

approx_on_full_history Native training parameter accepted in the `params` list.

auto_class_weights Native training parameter accepted in the `params` list.

bagging_temperature Native training parameter accepted in the `params` list.

baseline Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

best_model_min_trees Native training parameter accepted in the `params` list.

boost_from_average Native training parameter accepted in the `params` list.

boosting_type Native training parameter accepted in the `params` list.

bootstrap_type Native training parameter accepted in the `params` list.

border_count Native training parameter accepted in the `params` list.

callback Python-only (a native training-progress convenience distinct from `callbacks`); no catboost equivalent.

callbacks A list of functions, each taking a single `info` argument (a list with `iteration` and `metrics` elements, the latter shaped like `catboost.get_evals_result`'s return value) and returning `TRUE` to continue training or `FALSE` to stop it early. Training continues only while every callback returns `TRUE`, matching Python's `callbacks` kwarg.

cat_features Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

class_names Native training parameter accepted in the `params` list.

class_weights Native training parameter accepted in the `params` list.

classes_count Native training parameter accepted in the `params` list.

colsample_bylevel Alias of `rsm` (see `process_synonyms`); resolved automatically.

column_description Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

combinations_ctr Native training parameter accepted in the `params` list.

counter_calc_method Native training parameter accepted in the `params` list.

ctr_description Native training parameter accepted in the `params` list.

ctr_history_unit Native training parameter accepted in the `params` list.

ctr_leaf_count_limit Native training parameter accepted in the `params` list.

ctr_target_border_count Native training parameter accepted in the `params` list.

custom_loss Native training parameter accepted in the `params` list.

custom_metric Native training parameter accepted in the `params` list.

data_partition Native training parameter accepted in the `params` list.

depth Native training parameter accepted in the `params` list.

dev_efb_max_buckets Native training parameter accepted in the `params` list.

dev_score_calc_obj_block_size Native training parameter accepted in the `params` list.

device_config Native training parameter accepted in the `params` list.

devices Native training parameter accepted in the `params` list.

dictionaries Native training parameter accepted in the `params` list.

diffusion_temperature Native training parameter accepted in the `params` list.

early_stopping_rounds Native training parameter accepted in the `params` list.

embedding_features Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

eta Alias of `learning_rate` (see `process_synonyms`); resolved automatically.

eval_fraction Native training parameter accepted in the `params` list.

eval_metric Native training parameter accepted in the `params` list.

eval_set Supplied via the `test_pool` argument of `catboost.train`, not the `params` list.

feature_border_type Native training parameter accepted in the `params` list.

feature_calcers Native training parameter accepted in the `params` list.

feature_weights Native training parameter accepted in the `params` list.

final_ctr_computation_mode Native training parameter accepted in the `params` list.

first_feature_use_penalties Native training parameter accepted in the `params` list.

fixed_binary_splits Native training parameter accepted in the `params` list.

fold_len_multiplier Native training parameter accepted in the `params` list.

fold_permutation_block Native training parameter accepted in the `params` list.

gpu_cat_features_storage Native training parameter accepted in the `params` list.

gpu_ram_part Native training parameter accepted in the `params` list.

graph Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

group_id Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

group_weight Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

grow_policy Native training parameter accepted in the `params` list.

has_time Native training parameter accepted in the `params` list.

ignored_features Native training parameter accepted in the `params` list.

init_model Supplied via the `init_model` argument of `catboost.train`, not the `params` list.

input_borders Native training parameter accepted in the `params` list.

iterations Native training parameter accepted in the `params` list.

l2_leaf_reg Native training parameter accepted in the `params` list.

langevin Native training parameter accepted in the `params` list.

leaf_estimation_backtracking Native training parameter accepted in the `params` list.

leaf_estimation_iterations Native training parameter accepted in the `params` list.

leaf_estimation_method Native training parameter accepted in the `params` list.

learning_rate Native training parameter accepted in the `params` list.

log_cerr Python-only; no catboost equivalent.

log_cout Python-only; no catboost equivalent.

logging_level Native training parameter accepted in the `params` list.

loss_function Native training parameter accepted in the `params` list.

max_bin Alias of `border_count` (see `process_synonyms`); resolved automatically.

max_ctr_complexity Native training parameter accepted in the `params` list.

max_depth Alias of `depth` (see `process_synonyms`); resolved automatically.

max_leaves Native training parameter accepted in the `params` list.

metadata Native training parameter accepted in the `params` list.

metric_period Native training parameter accepted in the `params` list.

min_child_samples Alias of `min_data_in_leaf` (see `process_synonyms`); resolved automatically.

min_data_in_leaf Native training parameter accepted in the `params` list.

model_shrink_mode Native training parameter accepted in the `params` list.

model_shrink_rate Native training parameter accepted in the `params` list.

model_size_reg Native training parameter accepted in the `params` list.

monotone_constraints Native training parameter accepted in the `params` list.

mvs_reg Native training parameter accepted in the `params` list.

n_estimators Alias of `iterations` (see `process_synonyms`); resolved automatically.

name Native training parameter accepted in the `params` list.

nan_mode Native training parameter accepted in the `params` list.

num_boost_round Alias of `iterations` (see `process_synonyms`); resolved automatically.

num_leaves Alias of **max_leaves** (see **process_synonyms**); resolved automatically.
num_trees Alias of **iterations** (see **process_synonyms**); resolved automatically.
objective Alias of **loss_function** (see **process_synonyms**); resolved automatically.
od_pval Native training parameter accepted in the **params** list.
od_type Native training parameter accepted in the **params** list.
od_wait Native training parameter accepted in the **params** list.
one_hot_max_size Native training parameter accepted in the **params** list.
output_borders Native training parameter accepted in the **params** list.
pairs Supplied via **catboost.load_pool** (Pool construction), not the **params** list.
pairs_weight Supplied via **catboost.load_pool** (Pool construction), not the **params** list.
params Refers to the **params** argument itself.
penalties_coefficient Native training parameter accepted in the **params** list.
per_feature_ctr Native training parameter accepted in the **params** list.
per_float_feature_quantization Native training parameter accepted in the **params** list.
per_object_feature_penalties Native training parameter accepted in the **params** list.
pinned_memory_size Native training parameter accepted in the **params** list.
plot Python-only; no catboost equivalent.
plot_file Python-only; no catboost equivalent.
posterior_sampling Native training parameter accepted in the **params** list.
random_score_type Native training parameter accepted in the **params** list.
random_seed Native training parameter accepted in the **params** list.
random_state Alias of **random_seed** (see **process_synonyms**); resolved automatically.
random_strength Native training parameter accepted in the **params** list.
reg_lambda Alias of **l2_leaf_reg** (see **process_synonyms**); resolved automatically.
rsm Native training parameter accepted in the **params** list.
sample_weight Supplied via **catboost.load_pool** (Pool construction), not the **params** list.
sampling_frequency Native training parameter accepted in the **params** list.
sampling_unit Native training parameter accepted in the **params** list.
save_snapshot Native training parameter accepted in the **params** list.
scale_pos_weight Native training parameter accepted in the **params** list.
score_function Native training parameter accepted in the **params** list.
silent Client-side convenience translated to **logging_level** ("Silent" if TRUE, "Verbose" if FALSE) before export to native, matching Python's **silent** kwarg. Mutually exclusive with **logging_level/verbose/verbose_eval**.
simple_ctr Native training parameter accepted in the **params** list.
snapshot_file Native training parameter accepted in the **params** list.

snapshot_interval Native training parameter accepted in the `params` list.

sparse_features_conflict_fraction Native training parameter accepted in the `params` list.

store_all_simple_ctr Native training parameter accepted in the `params` list.

subgroup_id Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

subsample Native training parameter accepted in the `params` list.

target_border Native training parameter accepted in the `params` list.

task_type Native training parameter accepted in the `params` list.

text_features Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

text_processing Native training parameter accepted in the `params` list.

thread_count Supplied via `catboost.load_pool` (Pool construction), not the `params` list.

tokenizers Native training parameter accepted in the `params` list.

train_dir Native training parameter accepted in the `params` list.

use_best_model Native training parameter accepted in the `params` list.

used_ram_limit Native training parameter accepted in the `params` list.

verbose Native training parameter accepted in the `params` list.

verbose_eval Alias of `verbose` (see `process_synonyms`); resolved automatically.

y Supplied via `learn_pool/test_pool` (see `catboost.load_pool`), not the `params` list.

Note

If `params$class_names` is set and `learn_pool/ test_pool` was built from a factor or character label, this function temporarily retargets that Pool object's internal target representation for the duration of this call (needed to match native's classification target converter against string class names), then restores it before returning – including on error – so the Pool is left exactly as it was and remains safe to retrain, read via `catboost.pool.get_label/train_eval_split`, or reuse as both `learn_pool` and `test_pool`. This `class_names` + string-label combination is currently only supported via `catboost.train`; `catboost.cv/catboost.grid_search/ catboost.select_features/etc.` do not yet apply the same promotion.

See Also

https://catboost.ai/docs/concepts/r-reference_catboost-train.html

Examples

```
## Not run:
train_pool_path <- system.file("extdata", "adult_train.1000", package = "catboost")
test_pool_path <- system.file("extdata", "adult_test.1000", package = "catboost")
cd_path <- system.file("extdata", "adult.cd", package = "catboost")
train_pool <- catboost.load_pool(train_pool_path, column_description = cd_path)
test_pool <- catboost.load_pool(test_pool_path, column_description = cd_path)
```

```

fit_params <- list(
  iterations = 100,
  loss_function = 'Logloss',
  ignored_features = c(4, 9),
  border_count = 32,
  depth = 5,
  learning_rate = 0.03,
  l2_leaf_reg = 3.5,
  train_dir = 'train_dir')
model <- catboost.train(train_pool, test_pool, fit_params)

## End(Not run)

# (a) custom_objective: reimplement RMSE's own derivatives (matches
# calc_ders_range's contract -- weight is applied by the closure itself,
# unlike the built-in dispatcher, which multiplies by weight for you).
## Not run:
n <- 50
features <- data.frame(x1 = rnorm(n), x2 = rnorm(n))
label <- with(features, x1 - x2 + rnorm(n, sd = 0.3))
pool <- catboost.load_pool(features, label = label)
rmse_custom_objective <- list(
  calc_ders_range = function(approx, target, weight) {
    w <- if (is.null(weight)) 1 else weight
    cbind(w * (target - approx), w * (-1))
  })
model <- catboost.train(
  pool,
  params = list(iterations = 10, depth = 3, logging_level = "Silent",
    eval_metric = "RMSE", boost_from_average = FALSE),
  custom_objective = rmse_custom_objective)

## End(Not run)

# (b) custom_eval_metric_object: reimplement RMSE's own Eval/GetFinalError.
## Not run:
n <- 50
features <- data.frame(x1 = rnorm(n), x2 = rnorm(n))
label <- with(features, x1 - x2 + rnorm(n, sd = 0.3))
pool <- catboost.load_pool(features, label = label)
rmse_custom_eval_metric <- list(
  evaluate = function(approx, target, weight) {
    w <- if (is.null(weight)) rep(1, length(target)) else weight
    diff <- approx[, 1] - target
    list(error = sum(w * diff^2), weight = sum(w))
  },
  is_max_optimal = function() FALSE,
  get_final_error = function(error) sqrt(error[1] / (error[2] + 1e-38)))
model <- catboost.train(
  pool,
  params = list(iterations = 10, depth = 3, logging_level = "Silent"),
  custom_eval_metric_object = rmse_custom_eval_metric)

```

```
## End(Not run)
```

```
catboost.virtual_ensembles_predict
```

Apply the model with several virtual ensembles

Description

Apply the model to the given dataset using several independent truncated models - virtual ensembles. Each tree in ensemble predicts its own value for each document from pool.

Peculiarities: Return value varies on prediction_type: array for 'VirtEnsembles' and matrix for 'TotalUncertainty'

Usage

```
catboost.virtual_ensembles_predict(
    model,
    pool,
    verbose = FALSE,
    prediction_type = "VirtEnsembles",
    ntree_end = 0L,
    virtual_ensembles_count = 10,
    thread_count = -1
)
```

Arguments

model	The model obtained as the result of training. Default value: Required argument
pool	The input dataset. Default value: Required argument
verbose	Verbose output to stdout. Default value: FALSE (not used)
prediction_type	The format for displaying approximated values in output data (see https://catboost.ai/docs/concepts/python-reference_virtual_ensembles_predict.html#python-reference_catboostclassifier_predict__output-format). Possible values: • 'VirtEnsembles' • 'TotalUncertainty' Default value: 'VirtEnsembles'
ntree_end	Index of the first tree not to be used when applying the model or calculating the metrics (zero-based indexing). Default value: 0 (the index of the last tree to use equals to the number of trees in the model minus one)

virtual_ensembles_count	Number of tree ensembles to use. Each virtual ensemble can be considered as truncated model. Default value: 10
thread_count	The number of threads to use when applying the model. If -1, then the number of threads is set to the number of CPU cores. Allows you to optimize the speed of execution. This parameter doesn't affect results. Default value: -1

Value

Matrix or Array of predictions (for 'TotalUncertainty' and 'VirtEnsembles' prediction_type correspondingly)

See Also

https://catboost.ai/docs/concepts/python-reference_virtual_ensembles_predict.html?lang=en

dim.catboost.Pool	<i>Dimensions of catboost.Pool</i>
-------------------	------------------------------------

Description

Get dimensions of a Pool.

Usage

```
## S3 method for class 'catboost.Pool'
dim(x)
```

Arguments

x	The input dataset. Default value: Required argument
----------	--

Value

Returns a vector of row numbers and column numbers in an catboost.Pool.

```
dimnames.catboost.Pool
```

Dimension names of catboost.Pool

Description

Get dimension names of a Pool.

Column names are read from the Pool's own (C++-side) feature layout, so they stay in sync with `catboost.pool.get_feature_names()` after a `catboost.pool.set_feature_names()` call, which mutates that layout in place and cannot update an R-side attribute of the caller's object.

Usage

```
## S3 method for class 'catboost.Pool'
dimnames(x)
```

Arguments

x The input dataset.
 Default value: Required argument

Value

A list with the two elements. The second element contains the column names, or `NULL` if the Pool has no feature names.

```
head.catboost.Pool      Head of catboost.Pool
```

Description

Return a list with the first `n` objects of the dataset.

Each line of this list contains the following information for each object:

- The label value.
- The weight value.
- The feature values.

Usage

```
## S3 method for class 'catboost.Pool'
head(x, n = 10, ...)
```

Arguments

x	The input dataset. Default value: Required argument
n	The quantity of the first objects in the dataset to be returned. Default value: 10
...	not currently used

Value

A matrix containing the first **n** objects of the dataset.

```
print.catboost.Model
```

Print basic information about model

Description

Displays the most general characteristics of a CatBoost model.

Usage

```
## S3 method for class 'catboost.Model'
print(x, ...)
```

Arguments

x	The model obtained as the result of training.
...	Not used

Value

The same model that was passed as input.

```
print.catboost.Pool
```

Print catboost.Pool

Description

Print dimensions of catboost.Pool.

Usage

```
## S3 method for class 'catboost.Pool'
print(x, ...)
```

Arguments

<code>x</code>	a catboost.Pool object Default value: Required argument
<code>...</code>	not currently used

Value

Nothing. This method prints pool dimensions.

```
summary.catboost.Model
```

Print basic information about model

Description

Displays the most general characteristics of a CatBoost model (same as 'print').

Usage

```
## S3 method for class 'catboost.Model'
summary(object, ...)
```

Arguments

<code>object</code>	The model obtained as the result of training.
<code>...</code>	Not used

Value

The same model that was passed as input.

```
tail.catboost.Pool
```

Tail of catboost.Pool

Description

Return a list with the last n objects of the dataset.

Each line of this list contains the following information for each object:

- The target value.
- The weight value.
- The feature values.

Usage

```
## S3 method for class 'catboost.Pool'  
tail(x, n = 10, ...)
```

Arguments

x	The input dataset. Default value: Required argument
n	The quantity of the last objects in the dataset to be returned. Default value: 10
...	not currently used

Value

A matrix containing the last **n** objects of the dataset.

Index

catboost.calc_feature_statistics, [4](#), [54](#)
catboost.calc_leaf_indexes, [6](#), [44](#)
catboost.caret, [7](#)
catboost.compare, [7](#)
catboost.compute_training_options, [9](#)
catboost.create_metric_calcer, [10](#)
catboost.cv, [11](#), [78](#)
catboost.dataset_statistics, [12](#)
catboost.Dictionary, [13](#)
catboost.dictionary.apply, [15](#)
catboost.dictionary.end_of_sentence_token_id, [16](#)
catboost.dictionary.fit, [16](#)
catboost.dictionary.get_token, [17](#)
catboost.dictionary.get_tokens, [17](#)
catboost.dictionary.get_top_tokens, [18](#)
catboost.dictionary.load, [18](#)
catboost.dictionary.min_unused_token_id, [19](#)
catboost.dictionary.save, [19](#)
catboost.dictionary.size, [20](#)
catboost.dictionary.unknown_token_id, [21](#)
catboost.drop_unused_features, [21](#)
catboost.eval_feature, [22](#), [49](#), [50](#), [83](#)
catboost.eval_metric, [24](#)
catboost.eval_metrics, [7](#), [8](#), [10](#), [24](#), [25](#)
catboost.features_data.get_cat_feature_count, [26](#)
catboost.features_data.get_feature_count, [27](#)
catboost.features_data.get_feature_names, [27](#)
catboost.features_data.get_num_feature_count, [28](#)
catboost.features_data.get_object_count, [28](#)
catboost.FeaturesData, [29](#)
catboost.from_matrix, [54](#), [66](#)
catboost.get_best_iteration, [30](#)
catboost.get_best_score, [30](#)
catboost.get_borders, [31](#)
catboost.get_evals_result, [31](#), [112](#)
catboost.get_feature_importance, [32](#)
catboost.get_leaf_values, [33](#), [34](#), [85](#)
catboost.get_leaf_weights, [34](#)
catboost.get_metadata, [34](#), [38](#), [86](#)
catboost.get_model_feature_names, [35](#)
catboost.get_model_params, [35](#)
catboost.get_object_importance, [36](#)
catboost.get_plain_params, [37](#)
catboost.get_probability_threshold, [38](#)
catboost.get_roc_curve, [38](#)
catboost.get_scale_and_bias, [39](#)
catboost.get_test_eval, [40](#), [40](#), [41](#)
catboost.get_test_evals, [40](#)
catboost.get_tree_leaf_counts, [41](#)
catboost.grid_search, [41](#)
catboost.iterate_leaf_indexes, [44](#)
catboost.load_model, [44](#)
catboost.load_pool, [45](#), [54](#), [83](#), [84](#)
catboost.model_based_eval, [48](#)
catboost.normalize_model_from_pool, [50](#)
catboost.plot_partial_dependence, [51](#), [78](#)
catboost.plot_predictions, [52](#), [78](#)
catboost.plot_tree, [53](#)
catboost.Pool, [24](#)
catboost.pool.get_baseline, [54](#)
catboost.pool.get_cat_feature_indices, [55](#)
catboost.pool.get_embedding_feature_indices, [55](#)
catboost.pool.get_feature_names, [56](#)

`catboost.pool.get_features`, 56, 78, 79
`catboost.pool.get_group_id_hash`, 57, 65, 78
`catboost.pool.get_label`, 5, 57
`catboost.pool.get_text_feature_indices`, 58
`catboost.pool.get_weight`, 58, 78
`catboost.pool.has_label`, 59
`catboost.pool.is_empty`, 59
`catboost.pool.is_quantized`, 60
`catboost.pool.num_col`, 60
`catboost.pool.num_pairs`, 61, 66
`catboost.pool.num_row`, 61
`catboost.pool.quantize`, 62
`catboost.pool.save`, 62
`catboost.pool.save_quantization_borders`, 63, 80
`catboost.pool.set_baseline`, 64, 78
`catboost.pool.set_feature_names`, 64
`catboost.pool.set_group_id`, 65, 78
`catboost.pool.set_group_weight`, 65
`catboost.pool.set_pairs`, 25, 66
`catboost.pool.set_pairs_weight`, 66
`catboost.pool.set_subgroup_id`, 67
`catboost.pool.set_timestamp`, 47, 67
`catboost.pool.set_weight`, 68
`catboost.pool.shape`, 68
`catboost.pool.slice`, 69
`catboost.pool.train_eval_split`, 69
`catboost.predict`, 40, 51–53, 70, 73, 76, 89
`catboost.predict_log_proba`, 73, 90
`catboost.randomized_search`, 74
`catboost.restore_handle`, 76
`catboost.run_worker`, 77, 111
`catboost.sample_gaussian_process`, 78
`catboost.save_borders`, 80
`catboost.save_model`, 53, 81, 85–87
`catboost.save_pool`, 82
`catboost.select_features`, 47, 77, 83
`catboost.set_leaf_values`, 85
`catboost.set_metadata`, 38, 86, 86
`catboost.set_probability_threshold`, 86
`catboost.set_scale_and_bias`, 85, 87
`catboost.shrink`, 88
`catboost.staged_predict`, 88, 90
`catboost.staged_predict_log_proba`, 90
`catboost.sum_models`, 80, 91
`catboost.Tokenizer`, 92
`catboost.tokenizer.tokenize`, 93
`catboost.train`, 9, 11, 23, 40, 43, 75, 77–80, 84, 94
`catboost.virtual_ensembles_predict`, 118

`dim.catboost.Pool`, 119
`dimnames.catboost.Pool`, 120

`head.catboost.Pool`, 120

`predict.catboost.Model`, 7
`predict.catboost.Model`
 (`catboost.predict`), 70
`print.catboost.Model`, 121
`print.catboost.Pool`, 121

`summary.catboost.Model`, 122

`tail.catboost.Pool`, 122