Branch: master ▾

Find file | Copy path

LightGBM / docs / Parameters.rst

StrikerRUS fixed minor typos (#2119)

24ad35f 9 days ago

15 contributors

Raw | Blame | History

962 lines (497 sloc) 52.4 KB

Parameters

This page contains descriptions of all parameters in LightGBM.

List of other helpful links

- Python API
- Parameters Tuning

External Links

- Laurae++ Interactive Documentation

Parameters Format

The parameters format is `key1=value1 key2=value2 ...` . Parameters can be set both in config file and command line. By using command line, parameters should not have spaces before and after `=` . By using config files, one line can only contain one parameter. You can use `#` to comment.

If one parameter appears in both command line and config file, LightGBM will use the parameter from the command line.

Core Parameters

- `config` 🔗, default = `""` , type = string, aliases: `config_file`
  - path of config file
  - **Note**: can be used only in CLI version
- `task` 🔗, default = `train` , type = enum, options: `train` , `predict` , `convert_model` , `refit` , aliases: `task_type`
  - `train` , for training, aliases: `training`
  - `predict` , for prediction, aliases: `prediction` , `test`
  - `convert_model` , for converting model file into if-else format, see more information in IO Parameters
  - `refit` , for refitting existing models with new data, aliases: `refit_tree`
  - **Note**: can be used only in CLI version; for language-specific packages you can use the correspondent functions
- `objective` 🔗, default = `regression` , type = enum, options: `regression` , `regression_l1` , `huber` , `fair` , `poisson` , `quantile` , `mape` , `gammma` , `tweedie` , `binary` , `multiclass` , `multiclassova` , `xentropy` , `xentlambda` , `lambdarank` , aliases: `objective_type` , `app` , `application`
  - regression application
    - `regression_l2` , L2 loss, aliases: `regression` , `mean_squared_error` , `mse` , `l2_root` , `root_mean_squared_error` , `rmse`
    - `regression_l1` , L1 loss, aliases: `mean_absolute_error` , `mae`
    - `huber` , Huber loss

        - `fair` , Fair loss
        - `poisson` , Poisson regression
        - `quantile` , Quantile regression
        - `mape` , MAPE loss, aliases: `mean_absolute_percentage_error`
        - `gamma` , Gamma regression with log-link. It might be useful, e.g., for modeling insurance claims severity, or for any target that might be gamma-distributed
        - `tweedie` , Tweedie regression with log-link. It might be useful, e.g., for modeling total loss in insurance, or for any target that might be tweedie-distributed
    - `binary` , binary log loss classification (or logistic regression). Requires labels in {0, 1}; see `cross-entropy` application for general probability labels in [0, 1]
    - multi-class classification application
        - `multiclass` , softmax objective function, aliases: `softmax`
        - `multiclassova` , One-vs-All binary objective function, aliases: `multiclass_ova` , `ova` , `ovr`
        - `num_class` should be set as well
    - cross-entropy application
        - `xentropy` , objective function for cross-entropy (with optional linear weights), aliases: `cross_entropy`
        - `xentlambda` , alternative parameterization of cross-entropy, aliases: `cross_entropy_lambda`
        - label is anything in interval [0, 1]
    - `lambdarank` , lambdarank application
        - label should be `int` type in lambdarank tasks, and larger number represents the higher relevance (e.g. 0:bad, 1:fair, 2:good, 3:perfect)
        - label_gain can be used to set the gain (weight) of `int` label
        - all values in `label` must be smaller than number of elements in `label_gain`

- `boosting` 🔗, default = `gbdt` , type = enum, options: `gbdt` , `gbrt` , `rf` , `random_forest` , `dart` , `goss` , aliases: `boosting_type` , `boost`
    - `gbdt` , traditional Gradient Boosting Decision Tree, aliases: `gbrt`
    - `rf` , Random Forest, aliases: `random_forest`
    - `dart` , Dropouts meet Multiple Additive Regression Trees
    - `goss` , Gradient-based One-Side Sampling
- `data` 🔗, default = `""` , type = string, aliases: `train` , `train_data` , `train_data_file` , `data_filename`
    - path of training data, LightGBM will train from this data
    - **Note**: can be used only in CLI version
- `valid` 🔗, default = `""` , type = string, aliases: `test` , `valid_data` , `valid_data_file` , `test_data` , `test_data_file` , `valid_filenames`
    - path(s) of validation/test data, LightGBM will output metrics for these data
    - support multiple validation data, separated by `,`
    - **Note**: can be used only in CLI version
- `num_iterations` 🔗, default = `100` , type = int, aliases: `num_iteration` , `n_iter` , `num_tree` , `num_trees` , `num_round` , `num_rounds` , `num_boost_round` , `n_estimators` , constraints: `num_iterations >= 0`
    - number of boosting iterations
    - **Note**: internally, LightGBM constructs `num_class * num_iterations` trees for multi-class classification problems
- `learning_rate` 🔗, default = `0.1` , type = double, aliases: `shrinkage_rate` , `eta` , constraints: `learning_rate > 0.0`
    - shrinkage rate
    - in `dart` , it also affects on normalization weights of dropped trees
- `num_leaves` 🔗, default = `31` , type = int, aliases: `num_leaf` , `max_leaves` , `max_leaf` , constraints: `num_leaves > 1`
    - max number of leaves in one tree
- `tree_learner` 🔗, default = `serial` , type = enum, options: `serial` , `feature` , `data` , `voting` , aliases: `tree` , `tree_type` , `tree_learner_type`
    - `serial` , single machine tree learner
    - `feature` , feature parallel tree learner, aliases: `feature_parallel`

  - `data` , data parallel tree learner, aliases: `data_parallel`
  - `voting` , voting parallel tree learner, aliases: `voting_parallel`
  - refer to Parallel Learning Guide to get more details
- `num_threads` , default = `0` , type = int, aliases: `num_thread` , `nthread` , `nthreads` , `n_jobs`
  - number of threads for LightGBM
  - `0` means default number of threads in OpenMP
  - for the best speed, set this to the number of **real CPU cores**, not the number of threads (most CPUs use hyper-threading to generate 2 threads per CPU core)
  - do not set it too large if your dataset is small (for instance, do not use 64 threads for a dataset with 10,000 rows)
  - be aware a task manager or any similar CPU monitoring tool might report that cores not being fully utilized. **This is normal**
  - for parallel learning, do not use all CPU cores because this will cause poor performance for the network communication
- `device_type` , default = `cpu` , type = enum, options: `cpu` , `gpu` , aliases: `device`
  - device for the tree learning, you can use GPU to achieve the faster learning
  - **Note**: it is recommended to use the smaller `max_bin` (e.g. 63) to get the better speed up
  - **Note**: for the faster speed, GPU uses 32-bit float point to sum up by default, so this may affect the accuracy for some tasks. You can set `gpu_use_dp=true` to enable 64-bit float point, but it will slow down the training
  - **Note**: refer to Installation Guide to build LightGBM with GPU support
- `seed` , default = `None` , type = int, aliases: `random_seed` , `random_state`
  - this seed is used to generate other seeds, e.g. `data_random_seed` , `feature_fraction_seed` , etc.
  - by default, this seed is unused in favor of default values of other seeds
  - this seed has lower priority in comparison with other seeds, which means that it will be overridden, if you set other seeds explicitly

Learning Control Parameters

- `max_depth` , default = `-1` , type = int
  - limit the max depth for tree model. This is used to deal with over-fitting when `#data` is small. Tree still grows leaf-wise
  - `< 0` means no limit
- `min_data_in_leaf` , default = `20` , type = int, aliases: `min_data_per_leaf` , `min_data` , `min_child_samples` , constraints: `min_data_in_leaf >= 0`
  - minimal number of data in one leaf. Can be used to deal with over-fitting
- `min_sum_hessian_in_leaf` , default = `1e-3` , type = double, aliases: `min_sum_hessian_per_leaf` , `min_sum_hessian` , `min_hessian` , `min_child_weight` , constraints: `min_sum_hessian_in_leaf >= 0.0`
  - minimal sum hessian in one leaf. Like `min_data_in_leaf` , it can be used to deal with over-fitting
- `bagging_fraction` , default = `1.0` , type = double, aliases: `sub_row` , `subsample` , `bagging` , constraints: `0.0 < bagging_fraction <= 1.0`
  - like `feature_fraction` , but this will randomly select part of data without resampling
  - can be used to speed up training
  - can be used to deal with over-fitting
  - **Note**: to enable bagging, `bagging_freq` should be set to a non zero value as well
- `bagging_freq` , default = `0` , type = int, aliases: `subsample_freq`
  - frequency for bagging
  - `0` means disable bagging; `k` means perform bagging at every `k` iteration
  - **Note**: to enable bagging, `bagging_fraction` should be set to value smaller than `1.0` as well
- `bagging_seed` , default = `3` , type = int, aliases: `bagging_fraction_seed`
  - random seed for bagging
- `feature_fraction` , default = `1.0` , type = double, aliases: `sub_feature` , `colsample_bytree` , constraints: `0.0 < feature_fraction <= 1.0`

  - LightGBM will randomly select part of features on each iteration if `feature_fraction` smaller than `1.0`. For example, if you set it to `0.8`, LightGBM will select 80% of features before training each tree
  - can be used to speed up training
  - can be used to deal with over-fitting
- `feature_fraction_seed`, default = `2`, type = int
  - random seed for `feature_fraction`
- `early_stopping_round`, default = `0`, type = int, aliases: `early_stopping_rounds`, `early_stopping`
  - will stop training if one metric of one validation data doesn't improve in last `early_stopping_round` rounds
  - `<= 0` means disable
- `max_delta_step`, default = `0.0`, type = double, aliases: `max_tree_output`, `max_leaf_output`
  - used to limit the max output of tree leaves
  - `<= 0` means no constraint
  - the final max output of leaves is `learning_rate * max_delta_step`
- `lambda_l1`, default = `0.0`, type = double, aliases: `reg_alpha`, constraints: `lambda_l1 >= 0.0`
  - L1 regularization
- `lambda_l2`, default = `0.0`, type = double, aliases: `reg_lambda`, `lambda`, constraints: `lambda_l2 >= 0.0`
  - L2 regularization
- `min_gain_to_split`, default = `0.0`, type = double, aliases: `min_split_gain`, constraints: `min_gain_to_split >= 0.0`
  - the minimal gain to perform split
- `drop_rate`, default = `0.1`, type = double, aliases: `rate_drop`, constraints: `0.0 <= drop_rate <= 1.0`
  - used only in `dart`
  - dropout rate: a fraction of previous trees to drop during the dropout
- `max_drop`, default = `50`, type = int
  - used only in `dart`
  - max number of dropped trees during one boosting iteration
  - `<=0` means no limit
- `skip_drop`, default = `0.5`, type = double, constraints: `0.0 <= skip_drop <= 1.0`
  - used only in `dart`
  - probability of skipping the dropout procedure during a boosting iteration
- `xgboost_dart_mode`, default = `false`, type = bool
  - used only in `dart`
  - set this to `true`, if you want to use xgboost dart mode
- `uniform_drop`, default = `false`, type = bool
  - used only in `dart`
  - set this to `true`, if you want to use uniform drop
- `drop_seed`, default = `4`, type = int
  - used only in `dart`
  - random seed to choose dropping models
- `top_rate`, default = `0.2`, type = double, constraints: `0.0 <= top_rate <= 1.0`
  - used only in `goss`
  - the retain ratio of large gradient data
- `other_rate`, default = `0.1`, type = double, constraints: `0.0 <= other_rate <= 1.0`
  - used only in `goss`
  - the retain ratio of small gradient data
- `min_data_per_group`, default = `100`, type = int, constraints: `min_data_per_group > 0`
  - minimal number of data per categorical group
- `max_cat_threshold`, default = `32`, type = int, constraints: `max_cat_threshold > 0`
  - used for the categorical features
  - limit the max threshold points in categorical features

- `cat_l2` , default = `10.0` , type = double, constraints: `cat_l2 >= 0.0`
  - used for the categorical features
  - L2 regularization in categorcial split
- `cat_smooth` , default = `10.0` , type = double, constraints: `cat_smooth >= 0.0`
  - used for the categorical features
  - this can reduce the effect of noises in categorical features, especially for categories with few data
- `max_cat_to_onehot` , default = `4` , type = int, constraints: `max_cat_to_onehot > 0`
  - when number of categories of one feature smaller than or equal to `max_cat_to_onehot` , one-vs-other split algorithm will be used
- `top_k` , default = `20` , type = int, aliases: `topk` , constraints: `top_k > 0`
  - used in Voting parallel
  - set this to larger value for more accurate result, but it will slow down the training speed
- `monotone_constraints` , default = `None` , type = multi-int, aliases: `mc` , `monotone_constraint`
  - used for constraints of monotonic features
  - `1` means increasing, `-1` means decreasing, `0` means non-constraint
  - you need to specify all features in order. For example, `mc=-1,0,1` means decreasing for 1st feature, non-constraint for 2nd feature and increasing for the 3rd feature
- `feature_contri` , default = `None` , type = multi-double, aliases: `feature_contrib` , `fc` , `fp` , `feature_penalty`
  - used to control feature's split gain, will use `gain[i] = max(0, feature_contri[i]) * gain[i]` to replace the split gain of i-th feature
  - you need to specify all features in order
- `forcedsplits_filename` , default = `""` , type = string, aliases: `fs` , `forced_splits_filename` , `forced_splits_file` , `forced_splits`
  - path to a `.json` file that specifies splits to force at the top of every decision tree before best-first learning commences
  - `.json` file can be arbitrarily nested, and each split contains `feature` , `threshold` fields, as well as `left` and `right` fields representing subsplits
  - categorical splits are forced in a one-hot fashion, with `left` representing the split containing the feature value and `right` representing other values
  - **Note**: the forced split logic will be ignored, if the split makes gain worse
  - see this file as an example
- `refit_decay_rate` , default = `0.9` , type = double, constraints: `0.0 <= refit_decay_rate <= 1.0`
  - decay rate of `refit` task, will use `leaf_output = refit_decay_rate * old_leaf_output + (1.0 - refit_decay_rate) * new_leaf_output` to refit trees
  - used only in `refit` task in CLI version or as argument in `refit` function in language-specific package
- `cegb_tradeoff` , default = `1.0` , type = double, constraints: `cegb_tradeoff >= 0.0`
  - cost-effective gradient boosting multiplier for all penalties
- `cegb_penalty_split` , default = `0.0` , type = double, constraints: `cegb_penalty_split >= 0.0`
  - cost-effective gradient-boosting penalty for splitting a node
- `cegb_penalty_feature_lazy` , default = `0,0,...,0` , type = multi-double
  - cost-effective gradient boosting penalty for using a feature
  - applied per data point
- `cegb_penalty_feature_coupled` , default = `0,0,...,0` , type = multi-double
  - cost-effective gradient boosting penalty for using a feature
  - applied once per forest

IO Parameters

- `verbosity` , default = `1` , type = int, aliases: `verbose`
  - controls the level of LightGBM's verbosity
  - `< 0` : Fatal, `= 0` : Error (Warning), `= 1` : Info, `> 1` : Debug

- `max_bin` , default = `255` , type = int, constraints: `max_bin > 1`
  - max number of bins that feature values will be bucketed in
  - small number of bins may reduce training accuracy but may increase general power (deal with over-fitting)
  - LightGBM will auto compress memory according to `max_bin` . For example, LightGBM will use `uint8_t` for feature value if `max_bin=255`
- `min_data_in_bin` , default = `3` , type = int, constraints: `min_data_in_bin > 0`
  - minimal number of data inside one bin
  - use this to avoid one-data-one-bin (potential over-fitting)
- `bin_construct_sample_cnt` , default = `200000` , type = int, aliases: `subsample_for_bin` , constraints: `bin_construct_sample_cnt > 0`
  - number of data that sampled to construct histogram bins
  - setting this to larger value will give better training result, but will increase data loading time
  - set this to larger value if data is very sparse
- `histogram_pool_size` , default = `-1.0` , type = double, aliases: `hist_pool_size`
  - max cache size in MB for historical histogram
  - `< 0` means no limit
- `data_random_seed` , default = `1` , type = int, aliases: `data_seed`
  - random seed for data partition in parallel learning (excluding the `feature_parallel` mode)
- `output_model` , default = `LightGBM_model.txt` , type = string, aliases: `model_output` , `model_out`
  - filename of output model in training
  - **Note**: can be used only in CLI version
- `snapshot_freq` , default = `-1` , type = int, aliases: `save_period`
  - frequency of saving model file snapshot
  - set this to positive value to enable this function. For example, the model file will be snapshotted at each iteration if `snapshot_freq=1`
  - **Note**: can be used only in CLI version
- `input_model` , default = `""` , type = string, aliases: `model_input` , `model_in`
  - filename of input model
  - for `prediction` task, this model will be applied to prediction data
  - for `train` task, training will be continued from this model
  - **Note**: can be used only in CLI version
- `output_result` , default = `LightGBM_predict_result.txt` , type = string, aliases: `predict_result` , `prediction_result` , `predict_name` , `prediction_name` , `pred_name` , `name_pred`
  - filename of prediction result in `prediction` task
  - **Note**: can be used only in CLI version
- `initscore_filename` , default = `""` , type = string, aliases: `init_score_filename` , `init_score_file` , `init_score` , `input_init_score`
  - path of file with training initial scores
  - if `""` , will use `train_data_file` + `.init` (if exists)
  - **Note**: works only in case of loading data directly from file
- `valid_data_initscores` , default = `""` , type = string, aliases: `valid_data_init_scores` , `valid_init_score_file` , `valid_init_score`
  - path(s) of file(s) with validation initial scores
  - if `""` , will use `valid_data_file` + `.init` (if exists)
  - separate by `,` for multi-validation data
  - **Note**: works only in case of loading data directly from file
- `pre_partition` , default = `false` , type = bool, aliases: `is_pre_partition`
  - used for parallel learning (excluding the `feature_parallel` mode)
  - `true` if training data are pre-partitioned, and different machines use different partitions
- `enable_bundle` , default = `true` , type = bool, aliases: `is_enable_bundle` , `bundle`

  - set this to `false` to disable Exclusive Feature Bundling (EFB), which is described in LightGBM: A Highly Efficient Gradient Boosting Decision Tree
  - **Note**: disabling this may cause the slow training speed for sparse datasets
- `max_conflict_rate`, default = `0.0`, type = double, constraints: `0.0 <= max_conflict_rate < 1.0`
  - max conflict rate for bundles in EFB
  - set this to `0.0` to disallow the conflict and provide more accurate results
  - set this to a larger value to achieve faster speed
- `is_enable_sparse`, default = `true`, type = bool, aliases: `is_sparse`, `enable_sparse`, `sparse`
  - used to enable/disable sparse optimization
- `sparse_threshold`, default = `0.8`, type = double, constraints: `0.0 < sparse_threshold <= 1.0`
  - the threshold of zero elements percentage for treating a feature as a sparse one
- `use_missing`, default = `true`, type = bool
  - set this to `false` to disable the special handle of missing value
- `zero_as_missing`, default = `false`, type = bool
  - set this to `true` to treat all zero as missing values (including the unshown values in libsvm/sparse matrices)
  - set this to `false` to use `na` for representing missing values
- `two_round`, default = `false`, type = bool, aliases: `two_round_loading`, `use_two_round_loading`
  - set this to `true` if data file is too big to fit in memory
  - by default, LightGBM will map data file to memory and load features from memory. This will provide faster data loading speed, but may cause run out of memory error when the data file is very big
  - **Note**: works only in case of loading data directly from file
- `save_binary`, default = `false`, type = bool, aliases: `is_save_binary`, `is_save_binary_file`
  - if `true`, LightGBM will save the dataset (including validation data) to a binary file. This speed ups the data loading for the next time
  - **Note**: can be used only in CLI version; for language-specific packages you can use the correspondent function
- `header`, default = `false`, type = bool, aliases: `has_header`
  - set this to `true` if input data has header
  - **Note**: works only in case of loading data directly from file
- `label_column`, default = `""`, type = int or string, aliases: `label`
  - used to specify the label column
  - use number for index, e.g. `label=0` means column_0 is the label
  - add a prefix `name:` for column name, e.g. `label=name:is_click`
  - **Note**: works only in case of loading data directly from file
- `weight_column`, default = `""`, type = int or string, aliases: `weight`
  - used to specify the weight column
  - use number for index, e.g. `weight=0` means column_0 is the weight
  - add a prefix `name:` for column name, e.g. `weight=name:weight`
  - **Note**: works only in case of loading data directly from file
  - **Note**: index starts from `0` and it doesn't count the label column when passing type is `int`, e.g. when label is column_0, and weight is column_1, the correct parameter is `weight=0`
- `group_column`, default = `""`, type = int or string, aliases: `group`, `group_id`, `query_column`, `query`, `query_id`
  - used to specify the query/group id column
  - use number for index, e.g. `query=0` means column_0 is the query id
  - add a prefix `name:` for column name, e.g. `query=name:query_id`
  - **Note**: works only in case of loading data directly from file
  - **Note**: data should be grouped by query_id
  - **Note**: index starts from `0` and it doesn't count the label column when passing type is `int`, e.g. when label is column_0 and query_id is column_1, the correct parameter is `query=0`
- `ignore_column`, default = `""`, type = multi-int or string, aliases: `ignore_feature`, `blacklist`
  - used to specify some ignoring columns in training

  - use number for index, e.g. `ignore_column=0,1,2` means column_0, column_1 and column_2 will be ignored
  - add a prefix `name:` for column name, e.g. `ignore_column=name:c1,c2,c3` means c1, c2 and c3 will be ignored
  - **Note**: works only in case of loading data directly from file
  - **Note**: index starts from `0` and it doesn't count the label column when passing type is `int`
  - **Note**: despite the fact that specified columns will be completely ignored during the training, they still should have a valid format allowing LightGBM to load file successfully
- `categorical_feature` , default = `""` , type = multi-int or string, aliases: `cat_feature` , `categorical_column` , `cat_column`
  - used to specify categorical features
  - use number for index, e.g. `categorical_feature=0,1,2` means column_0, column_1 and column_2 are categorical features
  - add a prefix `name:` for column name, e.g. `categorical_feature=name:c1,c2,c3` means c1, c2 and c3 are categorical features
  - **Note**: only supports categorical with `int` type
  - **Note**: index starts from `0` and it doesn't count the label column when passing type is `int`
  - **Note**: all values should be less than `Int32.MaxValue` (2147483647)
  - **Note**: using large values could be memory consuming. Tree decision rule works best when categorical features are presented by consecutive integers starting from zero
  - **Note**: all negative values will be treated as **missing values**
- `predict_raw_score` , default = `false` , type = bool, aliases: `is_predict_raw_score` , `predict_rawscore` , `raw_score`
  - used only in `prediction` task
  - set this to `true` to predict only the raw scores
  - set this to `false` to predict transformed scores
- `predict_leaf_index` , default = `false` , type = bool, aliases: `is_predict_leaf_index` , `leaf_index`
  - used only in `prediction` task
  - set this to `true` to predict with leaf index of all trees
- `predict_contrib` , default = `false` , type = bool, aliases: `is_predict_contrib` , `contrib`
  - used only in `prediction` task
  - set this to `true` to estimate SHAP values, which represent how each feature contributes to each prediction
  - produces `#features + 1` values where the last value is the expected value of the model output over the training data
  - **Note**: if you want to get more explanation for your model's predictions using SHAP values like SHAP interaction values, you can install shap package
  - **Note**: unlike the shap package, with `predict_contrib` we return a matrix with an extra column, where the last column is the expected value
- `num_iteration_predict` , default = `-1` , type = int
  - used only in `prediction` task
  - used to specify how many trained iterations will be used in prediction
  - `<= 0` means no limit
- `pred_early_stop` , default = `false` , type = bool
  - used only in `prediction` task
  - if `true` , will use early-stopping to speed up the prediction. May affect the accuracy
- `pred_early_stop_freq` , default = `10` , type = int
  - used only in `prediction` task
  - the frequency of checking early-stopping prediction
- `pred_early_stop_margin` , default = `10.0` , type = double
  - used only in `prediction` task
  - the threshold of margin in early-stopping prediction
- `convert_model_language` , default = `""` , type = string
  - used only in `convert_model` task

  - only `cpp` is supported yet
  - if `convert_model_language` is set and `task=train`, the model will be also converted
  - **Note**: can be used only in CLI version
- `convert_model`, default = `gbdt_prediction.cpp`, type = string, aliases: `convert_model_file`
  - used only in `convert_model` task
  - output filename of converted model
  - **Note**: can be used only in CLI version

Objective Parameters

- `num_class`, default = `1`, type = int, aliases: `num_classes`, constraints: `num_class > 0`
  - used only in `multi-class` classification application
- `is_unbalance`, default = `false`, type = bool, aliases: `unbalance`, `unbalanced_sets`
  - used only in `binary` application
  - set this to `true` if training data are unbalanced
  - **Note**: while enabling this should increase the overall performance metric of your model, it will also result in poor estimates of the individual class probabilities
  - **Note**: this parameter cannot be used at the same time with `scale_pos_weight`, choose only **one** of them
- `scale_pos_weight`, default = `1.0`, type = double, constraints: `scale_pos_weight > 0.0`
  - used only in `binary` application
  - weight of labels with positive class
  - **Note**: while enabling this should increase the overall performance metric of your model, it will also result in poor estimates of the individual class probabilities
  - **Note**: this parameter cannot be used at the same time with `is_unbalance`, choose only **one** of them
- `sigmoid`, default = `1.0`, type = double, constraints: `sigmoid > 0.0`
  - used only in `binary` and `multiclassova` classification and in `lambdarank` applications
  - parameter for the sigmoid function
- `boost_from_average`, default = `true`, type = bool
  - used only in `regression`, `binary` and `cross-entropy` applications
  - adjusts initial score to the mean of labels for faster convergence
- `reg_sqrt`, default = `false`, type = bool
  - used only in `regression` application
  - used to fit `sqrt(label)` instead of original values and prediction result will be also automatically converted to `prediction^2`
  - might be useful in case of large-range labels
- `alpha`, default = `0.9`, type = double, constraints: `alpha > 0.0`
  - used only in `huber` and `quantile` `regression` applications
  - parameter for Huber loss and Quantile regression
- `fair_c`, default = `1.0`, type = double, constraints: `fair_c > 0.0`
  - used only in `fair` `regression` application
  - parameter for Fair loss
- `poisson_max_delta_step`, default = `0.7`, type = double, constraints: `poisson_max_delta_step > 0.0`
  - used only in `poisson` `regression` application
  - parameter for Poisson regression to safeguard optimization
- `tweedie_variance_power`, default = `1.5`, type = double, constraints: `1.0 <= tweedie_variance_power < 2.0`
  - used only in `tweedie` `regression` application
  - used to control the variance of the tweedie distribution
  - set this closer to `2` to shift towards a **Gamma** distribution
  - set this closer to `1` to shift towards a **Poisson** distribution
- `max_position`, default = `20`, type = int, constraints: `max_position > 0`

    - used only in `lambdarank` application
    - optimizes NDCG at this position
- `label_gain`, default = `0,1,3,7,15,31,63,...,2^30-1`, type = multi-double
    - used only in `lambdarank` application
    - relevant gain for labels. For example, the gain of label `2` is `3` in case of default label gains
    - separate by `,`

Metric Parameters

- `metric`, default = `""`, type = multi-enum, aliases: `metrics`, `metric_types`
    - metric(s) to be evaluated on the evaluation set(s)
        - `""` (empty string or not specified) means that metric corresponding to specified `objective` will be used (this is possible only for pre-defined objective functions, otherwise no evaluation metric will be added)
        - `"None"` (string, **not** a `None` value) means that no metric will be registered, aliases: `na`, `null`, `custom`
        - `l1`, absolute loss, aliases: `mean_absolute_error`, `mae`, `regression_l1`
        - `l2`, square loss, aliases: `mean_squared_error`, `mse`, `regression_l2`, `regression`
        - `l2_root`, root square loss, aliases: `root_mean_squared_error`, `rmse`
        - `quantile`, Quantile regression
        - `mape`, MAPE loss, aliases: `mean_absolute_percentage_error`
        - `huber`, Huber loss
        - `fair`, Fair loss
        - `poisson`, negative log-likelihood for Poisson regression
        - `gamma`, negative log-likelihood for **Gamma** regression
        - `gamma_deviance`, residual deviance for **Gamma** regression
        - `tweedie`, negative log-likelihood for **Tweedie** regression
        - `ndcg`, NDCG, aliases: `lambdarank`
        - `map`, MAP, aliases: `mean_average_precision`
        - `auc`, AUC
        - `binary_logloss`, log loss, aliases: `binary`
        - `binary_error`, for one sample: `0` for correct classification, `1` for error classification
        - `multi_logloss`, log loss for multi-class classification, aliases: `multiclass`, `softmax`, `multiclassova`, `multiclass_ova`, `ova`, `ovr`
        - `multi_error`, error rate for multi-class classification
        - `xentropy`, cross-entropy (with optional linear weights), aliases: `cross_entropy`
        - `xentlambda`, "intensity-weighted" cross-entropy, aliases: `cross_entropy_lambda`
        - `kldiv`, Kullback-Leibler divergence, aliases: `kullback_leibler`
    - support multiple metrics, separated by `,`
- `metric_freq`, default = `1`, type = int, aliases: `output_freq`, constraints: `metric_freq > 0`
    - frequency for metric output
- `is_provide_training_metric`, default = `false`, type = bool, aliases: `training_metric`, `is_training_metric`, `train_metric`
    - set this to `true` to output metric result over training dataset
    - **Note**: can be used only in CLI version
- `eval_at`, default = `1,2,3,4,5`, type = multi-int, aliases: `ndcg_eval_at`, `ndcg_at`, `map_eval_at`, `map_at`
    - used only with `ndcg` and `map` metrics
    - NDCG and MAP evaluation positions, separated by `,`

Network Parameters

- `num_machines`, default = `1`, type = int, aliases: `num_machine`, constraints: `num_machines > 0`
  - the number of machines for parallel learning application
  - this parameter is needed to be set in both **socket** and **mpi** versions
- `local_listen_port`, default = `12400`, type = int, aliases: `local_port`, `port`, constraints: `local_listen_port > 0`
  - TCP listen port for local machines
  - **Note**: don't forget to allow this port in firewall settings before training
- `time_out`, default = `120`, type = int, constraints: `time_out > 0`
  - socket time-out in minutes
- `machine_list_filename`, default = `""`, type = string, aliases: `machine_list_file`, `machine_list`, `mlist`
  - path of file that lists machines for this parallel learning application
  - each line contains one IP and one port for one machine. The format is `ip port` (space as a separator)
- `machines`, default = `""`, type = string, aliases: `workers`, `nodes`
  - list of machines in the following format: `ip1:port1,ip2:port2`

GPU Parameters

- `gpu_platform_id`, default = `-1`, type = int
  - OpenCL platform ID. Usually each GPU vendor exposes one OpenCL platform
  - `-1` means the system-wide default platform
  - **Note**: refer to GPU Targets for more details
- `gpu_device_id`, default = `-1`, type = int
  - OpenCL device ID in the specified platform. Each GPU in the selected platform has a unique device ID
  - `-1` means the default device in the selected platform
  - **Note**: refer to GPU Targets for more details
- `gpu_use_dp`, default = `false`, type = bool
  - set this to `true` to use double precision math on GPU (by default single precision is used)

Others

Continued Training with Input Score

LightGBM supports continued training with initial scores. It uses an additional file to store these initial scores, like the following:

```
0.5
-0.1
0.9
...
```

It means the initial score of the first data row is `0.5`, second is `-0.1`, and so on. The initial score file corresponds with data file line by line, and has per score per line.

And if the name of data file is `train.txt`, the initial score file should be named as `train.txt.init` and in the same folder as the data file. In this case, LightGBM will auto load initial score file if it exists.

Otherwise, you should specify the path to the custom named file with initial scores by the `initscore_filename` parameter.

Weight Data

LightGBM supports weighted training. It uses an additional file to store weight data, like the following:

```
1.0
0.5
```

```
0.8
...
```

It means the weight of the first data row is `1.0` , second is `0.5` , and so on. The weight file corresponds with data file line by line, and has per weight per line.

And if the name of data file is `train.txt` , the weight file should be named as `train.txt.weight` and placed in the same folder as the data file. In this case, LightGBM will load the weight file automatically if it exists.

Also, you can include weight column in your data file. Please refer to the `weight_column` parameter in above.

Query Data

For LambdaRank learning, it needs query information for training data. LightGBM uses an additional file to store query data, like the following:

```
27
18
67
...
```

It means first `27` lines samples belong to one query and next `18` lines belong to another, and so on.

Note: data should be ordered by the query.

If the name of data file is `train.txt` , the query file should be named as `train.txt.query` and placed in the same folder as the data file. In this case, LightGBM will load the query file automatically if it exists.

Also, you can include query/group id column in your data file. Please refer to the `group_column` parameter in above.