1. 安装 XGBOOST

使用 pip install xgboost==0.8

安装 graphviz

1.1XGBOOST 案例实战

```
#导入 xgb 的包
import xgboost as xgb
import numpy as np
#xgb 读取数据
# read in data
print(xgb.__version__)
data = np.random.rand(5, 10)    # 5 entities, each contains 10 features
label = np.random.randint(2, size=5)    # binary target
dtrain = xgb.DMatrix(data, label=label)

print(dtrain.feature_names)
print(dtrain.num_col())
#                                                dtrain                                                =
xgb.DMatrix('D:\\BigData\\Workspace\\PycharmProjects\\MachineLearning2\\BigDa
taClass10\\Day11\\agaricus.txt.train')
#                                                 dtest                                                =
xgb.DMatrix('D:\\BigData\\Workspace\\PycharmProjects\\MachineLearning2\\BigDa
taClass10\\Day11\\agaricus.txt.test')
dtrain = xgb.DMatrix('agaricus.txt.train')
dtest = xgb.DMatrix('agaricus.txt.test')
#                                                dtrain                                                =
xgb.DMatrix('C:\\Users\\zhao-chj\\AnacondaProjects\\XGBOOST\\agaricus.txt.train')
```

```
# dtest = xgb.DMatrix('C:\\Users\\zhao-chj\\AnacondaProjects\\XGBOOST\\agaricus.txt.test')
# specify parameters via map
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }
num_round = 2
bst = xgb.train(param, dtrain, num_round)

# make prediction
preds = bst.predict(dtest)
print(preds)
test_value=[round(value) for value in preds]
print(test_value)

from sklearn.metrics import accuracy_score
test_accuracy=accuracy_score(dtest.get_label(),test_value)
print("data train set score is : %.2f%%"%(test_accuracy*100.0))

# import    matplotlib.pyplot as plt
# xgb.plot_tree(bst)
# plt.show()
```

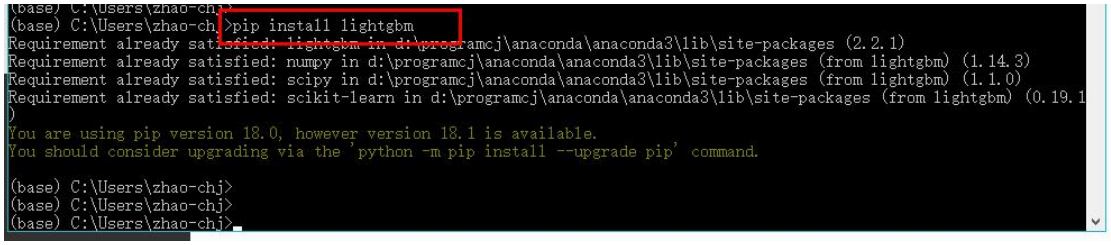
1.2 结合 Sklearn 的 XGBOOST 模型实战

```
from xgboost import XGBClassifier
from sklearn.datasets import load_svmlight_file
X_train,y_train=load_svmlight_file("./agaricus.txt.train")
bst=XGBClassifier(max_depth=3, learning_rate=0.1,
                  n_estimators=100, silent=True,
                  objective="binary:logistic", booster='gbtree',)
print(bst.fit(X_train,y_train))
X_test,y_test=load_svmlight_file("./agaricus.txt.test")
print(X_test)
print(y_test)
y_pred=bst.predict(X_test)
print(y_pred)
y_each_pred=[round(value) for value in y_pred]

y_train_pred=bst.predict(X_train)
y_train_each_pred=[round(value) for value in y_train_pred]
from    sklearn.metrics import accuracy_score

print("model in train score
```

```
is: %.2f%%"%(accuracy_score(y_train,y_train_each_pred)*100.0))
print("model in test score is: %.2f%%"%(accuracy_score(y_test,y_each_pred)*100.0))
```

2. 安装 LightGBM

https://github.com/apachecn/lightgbm-doc-zh 中文文档
http://lightgbm.apachecn.org/cn/latest/Installation-Guide.html
官方案例：
https://github.com/Microsoft/LightGBM/tree/master/examples/python-guide



2.1LightGBM 实战 1

```
import lightgbm as lgb
import pandas as pd
from sklearn.metrics import mean_squared_error

print('Loading data...')
# load or create your dataset
df_train = pd.read_csv('./regression.train', header=None, sep='\t')
df_test = pd.read_csv('./regression.test', header=None, sep='\t')

y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)

# create dataset for lightgbm
# '如果这是数据集进行验证,应该使用训练数据作为参考'
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

# specify your configurations as a dict
params = {
        'boosting_type': 'gbdt',
        'objective': 'regression',
        'metric': {'l2', 'l1'},
```

```
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': 0
}

print('Starting training...')
# train
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=20,
                valid_sets=lgb_eval,
                early_stopping_rounds=5)

print('Saving model...')
# save model to file
gbm.save_model('model.txt')

print('Starting predicting...')
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
# eval
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
```

2.2LightGBM 和 SKlearn 结合实战

```
import numpy as np
import pandas as pd
import lightgbm as lgb

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

print('Loading data...')
# load or create your dataset
df_train = pd.read_csv('./regression.train', header=None, sep='\t')
df_test = pd.read_csv('./regression.test', header=None, sep='\t')

y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
```

```
X_test = df_test.drop(0, axis=1)

print('Starting training...')
# train
gbm = lgb.LGBMRegressor(num_leaves=31,
                        learning_rate=0.05,
                        n_estimators=20)
gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='l1',
        early_stopping_rounds=5)

print('Starting predicting...')
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)
# eval
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)

# feature importances
print('Feature importances:', list(gbm.feature_importances_))
print("="*100)
#
==========================================================================
===========
# self-defined eval metric
# f(y_true: array, y_pred: array) -> name: string, eval_result: float, is_higher_better:
bool
# Root Mean Squared Logarithmic Error (RMSLE)
def rmsle(y_true, y_pred):
    return 'RMSLE', np.sqrt(np.mean(np.power(np.log1p(y_pred) - np.log1p(y_true),
2))), False



print('Starting training with custom eval function...')
# train
gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric=rmsle,
        early_stopping_rounds=5)

print('Starting predicting...')
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)
# eval
```

```
print('The rmsle of prediction is:', rmsle(y_test, y_pred)[1])
#====================================================================
========
print("="*100)
# other scikit-learn modules
estimator = lgb.LGBMRegressor(num_leaves=31)

param_grid = {
    'learning_rate': [0.01, 0.1, 1],
    'n_estimators': [20, 40]
}

gbm = GridSearchCV(estimator, param_grid, cv=3)
gbm.fit(X_train, y_train)

print('Best parameters found by grid search are:', gbm.best_params_)
```

2.3LightGBM 高级主题

```
import json
import lightgbm as lgb
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error

try:
    import cPickle as pickle
except BaseException:
    import pickle

print('Loading data...')
# load or create your dataset
df_train = pd.read_csv('./binary_classification/binary.train', header=None, sep='\t')
df_test = pd.read_csv('./binary_classification/binary.test', header=None, sep='\t')
W_train = pd.read_csv('./binary_classification/binary.train.weight', header=None)[0]
W_test = pd.read_csv('./binary_classification/binary.test.weight', header=None)[0]

y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)

num_train, num_feature = X_train.shape
```

```
# create dataset for lightgbm
# if you want to re-use data, remember to set free_raw_data=False
lgb_train = lgb.Dataset(X_train, y_train,
                                    weight=W_train, free_raw_data=False)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train,
                                   weight=W_test, free_raw_data=False)

# specify your configurations as a dict
params = {
        'boosting_type': 'gbdt',
        'objective': 'binary',
        'metric': 'binary_logloss',
        'num_leaves': 31,
        'learning_rate': 0.05,
        'feature_fraction': 0.9,
        'bagging_fraction': 0.8,
        'bagging_freq': 5,
        'verbose': 0
}

# generate feature names-feature_1---feature_2----feature_3
feature_name = ['feature_' + str(col) for col in range(num_feature)]

print('Starting training...')
# feature_name and categorical_feature
gbm = lgb.train(params,
                            lgb_train,
                            num_boost_round=10,
                            valid_sets=lgb_train,    # eval training data
                            feature_name=feature_name,
                            categorical_feature=[21])

print('Finished first 10 rounds...')
# check feature name
print('7th feature name is:', lgb_train.feature_name[6])

print('Saving model...')
# 4.1save model to file
gbm.save_model('model.txt')

print('Dumping model to JSON...')
# 4.2 dump model to JSON (and save to file)
model_json = gbm.dump_model()
```

```
with open('model.json', 'w+') as f:
    json.dump(model_json, f, indent=4)

# feature names
print('Feature names:', gbm.feature_name())

# feature importances
print('Feature importances:', list(gbm.feature_importance()))

print('Loading model to predict...')
# #4.3 load model to predict
bst = lgb.Booster(model_file='model.txt')
# can only predict with the best iteration (or the saving iteration)
y_pred = bst.predict(X_test)
# eval with loaded model
print("The rmse of loaded model's prediction is:", mean_squared_error(y_test,
y_pred) ** 0.5)

print('Dumping and loading model with pickle...')
# dump model with pickle
with open('model.pkl', 'wb') as fout:
    pickle.dump(gbm, fout)
# load model with pickle to predict
with open('model.pkl', 'rb') as fin:
    pkl_bst = pickle.load(fin)
# can predict with any iteration when loaded in pickle way
y_pred = pkl_bst.predict(X_test, num_iteration=7)
# eval with loaded model
print("The rmse of pickled model's prediction is:", mean_squared_error(y_test,
y_pred) ** 0.5)

#=================================================================
========================
# continue training
# init_model accepts:
# 1. model file name
# 2. Booster()
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=10,
                init_model='model.txt',
                valid_sets=lgb_eval)
```

```
print('Finished 10 - 20 rounds with model file...')

# decay learning rates
# learning_rates accepts:
# 1. list/tuple with length = num_boost_round
# 2. function(curr_iter)
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=10,
                init_model=gbm,
                learning_rates=lambda iter: 0.05 * (0.99 ** iter),
                valid_sets=lgb_eval)

print('Finished 20 - 30 rounds with decay learning rates...')

# change other parameters during training
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=10,
                init_model=gbm,
                valid_sets=lgb_eval,
                callbacks=[lgb.reset_parameter(bagging_fraction=[0.7] * 5 + [0.6]
* 5)])

print('Finished 30 - 40 rounds with changing bagging_fraction...')


# self-defined objective function
# f(preds: array, train_data: Dataset) -> grad: array, hess: array
# log likelihood loss
def loglikelihood(preds, train_data):
    labels = train_data.get_label()
    preds = 1. / (1. + np.exp(-preds))
    grad = preds - labels
    hess = preds * (1. - preds)
    return grad, hess


# self-defined eval metric
# f(preds: array, train_data: Dataset) -> name: string, eval_result: float,
is_higher_better: bool
# binary error
def binary_error(preds, train_data):
    labels = train_data.get_label()
```

```
    return 'error', np.mean(labels != (preds > 0.5)), False


gbm = lgb.train(params,
                lgb_train,
                num_boost_round=10,
                init_model=gbm,
                fobj=loglikelihood,
                feval=binary_error,
                valid_sets=lgb_eval)

print('Finished 40 - 50 rounds with self-defined objective function and eval metric...')

print("=====*100)
#==========================================================================
======

print('Starting a new training job...')


# callback
def reset_metrics():
    def callback(env):
        lgb_eval_new = lgb.Dataset(X_test, y_test, reference=lgb_train)
        if env.iteration - env.begin_iteration == 5:
            print('Add a new valid dataset at iteration 5...')
            env.model.add_valid(lgb_eval_new, 'new_valid')
    callback.before_iteration = True
    callback.order = 0
    return callback


gbm = lgb.train(params,
                lgb_train,
                num_boost_round=10,
                valid_sets=lgb_train,
                callbacks=[reset_metrics()])

print('Finished first 10 rounds with callback function...')
```

2.4LogisticRegression 案例

```
import time
```

```
import lightgbm as lgb
import numpy as np
import pandas as pd
from scipy.special import expit

##################
# Simulate some binary data with a single categorical and
#      single continuous predictor
np.random.seed(0)
N = 1000
X = pd.DataFrame({
       'continuous': range(N),
       'categorical': np.repeat([0, 1, 2, 3, 4], N / 5)
})
CATEGORICAL_EFFECTS = [-1, -1, -2, -2, 2]
LINEAR_TERM = np.array([
       -0.5 + 0.01 * X['continuous'][k]
       + CATEGORICAL_EFFECTS[X['categorical'][k]] for k in range(X.shape[0])
]) + np.random.normal(0, 1, X.shape[0])
TRUE_PROB = expit(LINEAR_TERM)
Y = np.random.binomial(1, TRUE_PROB, size=N)
DATA = {
       'X': X,
       'probability_labels': TRUE_PROB,
       'binary_labels': Y,
       'lgb_with_binary_labels': lgb.Dataset(X, Y),
       'lgb_with_probability_labels': lgb.Dataset(X, TRUE_PROB),
}


##################
# Set up a couple of utilities for our experiments
def log_loss(preds, labels):
       """Logarithmic loss with non-necessarily-binary labels."""
       log_likelihood = np.sum(labels * np.log(preds)) / len(preds)
       return -log_likelihood


def experiment(objective, label_type, data):
       """Measure performance of an objective.
       Parameters
       ----------
       objective : string 'binary' or 'xentropy'
```

```
        Objective function.
    label_type : string 'binary' or 'probability'
        Type of the label.
    data : dict
        Data for training.
    Returns
    -------
    result : dict
        Experiment summary stats.
    """
    np.random.seed(0)
    nrounds = 5
    lgb_data = data['lgb_with_' + label_type + '_labels']
    params = {
        'objective': objective,
        'feature_fraction': 1,
        'bagging_fraction': 1,
        'verbose': -1
    }
    time_zero = time.time()
    gbm = lgb.train(params, lgb_data, num_boost_round=nrounds)
    y_fitted = gbm.predict(data['X'])
    y_true = data[label_type + '_labels']
    duration = time.time() - time_zero
    return {
        'time': duration,
        'correlation': np.corrcoef(y_fitted, y_true)[0, 1],
        'logloss': log_loss(y_fitted, y_true)
    }


##################
# Observe the behavior of `binary` and `xentropy` objectives
print('Performance of `binary` objective with binary labels:')
print(experiment('binary', label_type='binary', data=DATA))

print('Performance of `xentropy` objective with binary labels:')
print(experiment('xentropy', label_type='binary', data=DATA))

print('Performance of `xentropy` objective with probability labels:')
print(experiment('xentropy', label_type='probability', data=DATA))

# Trying this throws an error on non-binary values of y:
#     experiment('binary', label_type='probability', DATA)
```

```
# The speed of `binary` is not drastically different than
#      `xentropy`. `xentropy` runs faster than `binary` in many cases, although
#      there are reasons to suspect that `binary` should run faster when the
#      label is an integer instead of a float
K = 10
A = [experiment('binary', label_type='binary', data=DATA)['time']
        for k in range(K)]
B = [experiment('xentropy', label_type='binary', data=DATA)['time']
        for k in range(K)]
print('Best `binary` time: ' + str(min(A)))
print('Best `xentropy` time: ' + str(min(B)))
```

2.5 结合 pyplot 绘图

```
import lightgbm as lgb
import pandas as pd

if lgb.compat.MATPLOTLIB_INSTALLED:
    import matplotlib.pyplot as plt
else:
    raise ImportError('You need to install matplotlib for plot_example.py.')

print('Loading data...')
# load or create your dataset
df_train = pd.read_csv('./regression.train', header=None, sep='\t')
df_test = pd.read_csv('./regression.test', header=None, sep='\t')

y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)

# create dataset for lightgbm
lgb_train = lgb.Dataset(X_train, y_train)
lgb_test = lgb.Dataset(X_test, y_test, reference=lgb_train)

# specify your configurations as a dict
params = {
    'num_leaves': 5,
    'metric': ('l1', 'l2'),
    'verbose': 0
}
```

```
evals_result = {}    # to record eval results for plotting

print('Starting training...')
# train
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=100,
                valid_sets=[lgb_train, lgb_test],
                feature_name=['f' + str(i + 1) for i in range(X_train.shape[-1])],
                categorical_feature=[21],
                evals_result=evals_result,
                verbose_eval=10)

print('Plotting metrics recorded during training...')
ax = lgb.plot_metric(evals_result, metric='l1')
plt.show()

print('Plotting feature importances...')
ax = lgb.plot_importance(gbm, max_num_features=10)
plt.show()

print('Plotting 84th tree...')    # one tree use categorical feature to split
ax = lgb.plot_tree(gbm, tree_index=83, figsize=(20, 8), show_info=['split_gain'])
plt.show()

print('Plotting 84th tree with graphviz...')
graph = lgb.create_tree_digraph(gbm, tree_index=83, name='Tree84')
graph.render(view=True)
```

LightGBM 的 Python 接口文档！

LightGBM 是一种使用基于树的学习算法的梯度提升框架。它具有以下优点，具有分布式和高效性：

- 更快的培训速度和更高的效率。
- 降低内存使用率。
- 更准确。
- 支持并行和 GPU 学习。
- 能够处理大规模数据。

内容：

安装

安装 Python 包的依赖关系， `setuptools`， `wheel`， `numpy` 和 `scipy` 是必需的，`scikit-learn` 需要 sklearn 接口，并建议：

```
pip install setuptools wheel numpy scipy scikit-learn -U
```

有关安装指南，请参阅 Python-package 文件夹。

要验证您的安装，请尝试使用 Python： `import lightgbm`

```
import lightgbm as lgb
```

数据接口

LightGBM Python 模块可以从以下位置加载数据：

- libsvm / tsv / csv / txt 格式文件
- NumPy 2D 数组，pandas DataFrame，SciPy 稀疏矩阵
- LightGBM 二进制文件

数据存储在 `Dataset` 对象中。

要将 libsvm 文本文件或 LightGBM 二进制文件加载到数据集中：

```
train_data = lgb.Dataset('train.svm.bin')
```

要将 numpy 数组加载到数据集中：

```
data = np.random.rand(500, 10)  # 500 entities, each contains 10 featureslabel = np.random.randint(2, size=500)  # binary targettrain_data = lgb.Dataset(data, label=label)
```

要将 scpiy.sparse.csr_matrix 数组加载到数据集中：

```
csr = scipy.sparse.csr_matrix((dat, (row, col)))train_data = lgb.Dataset(csr)
```

将数据集保存到 LightGBM 二进制文件将使加载更快：

```
train_data = lgb.Dataset('train.svm.txt')train_data.save_binary('train.bin')
```

创建验证数据：

```
test_data = train_data.create_valid('test.svm')
```

要么

```
test_data = lgb.Dataset('test.svm', reference=train_data)
```

在 LightGBM 中，验证数据应与训练数据保持一致。

具体功能名称和分类功能：

```
train_data = lgb.Dataset(data, label=label, feature_name=['c1', 'c2', 'c3'], categorical_feature=['c3'])
```

LightGBM 可以直接使用分类功能作为输入。它不需要转换为单热编码，并且比单热编码快得多（大约 8 倍加速）。

注意：您应该 `int` 在构造之前将分类要素转换为类型 `Dataset`。

可在需要时设置重量：

```
w = np.random.rand(500, )train_data = lgb.Dataset(data, label=label, weight=w)
```

要么

```
train_data = lgb.Dataset(data, label=label)w =
np.random.rand(500, )train_data.set_weight(w)
```

您可以使用 `Dataset.set_init_score()`设置初始分数，以及 `Dataset.set_group()`为排名任务设置组/查询数据。

内存有效使用：

`Dataset`LightGBM 中的对象非常节省内存，因为它只需要保存离散的 bin。但是，Numpy / Array / Pandas 对象是内存开销。如果您担心内存消耗，可以根据以下内容节省内存：

- 1.构造时让 `free_raw_data=True`（默认为 `True`）`Dataset`
- 2.在构造 `raw_data=None` 之后显式设置 `Dataset`
- 3.呼叫 `gc`

设定参数

LightGBM 可以使用对列表或字典来设置参数。例如：

助推器参数：

```
param = {'num_leaves':31, 'num_trees':100, 'objective':'binary'}param['metric'] = 'auc'
```

您还可以指定多个 eval 指标：

```
param['metric'] = ['auc', 'binary_logloss']
```

训练

训练模型需要参数列表和数据集：

```
num_round = 10bst = lgb.train(param, train_data, num_round, valid_sets=[test_data])
```

训练后，可以保存模型：

```
bst.save_model('model.txt')
```

训练的模型也可以转储为 JSON 格式：

```
json_model = bst.dump_model()
```

可以加载已保存的模型：

```
bst = lgb.Booster(model_file='model.txt')  #init model
```

简历

5 则 CV 训练：

```
num_round = 10lgb.cv(param, train_data, num_round, nfold=5)
```

提前停止

如果您有验证集，则可以使用提前停止来查找最佳提升次数。提前停车需要至少一套 `valid_sets`。如果有多个，除了训练数据外，它将使用所有这些：

```
bst = lgb.train(param, train_data, num_round, valid_sets=valid_sets,
early_stopping_rounds=10)
bst.save_model('model.txt', num_iteration=bst.best_iteration)
```

该模型将进行训练，直到验证分数停止改善。验证分数至少需要改进每一项 `early_stopping_rounds` 才能继续培训。

`best_iteration` 如果通过设置启用了早期停止逻辑，则具有最佳性能的迭代索引将保存在字段中 `early_stopping_rounds`。请注意，`train()`将从最佳迭代返回模型。

这适用于两种指标，以最小化（L2，日志丢失等）和最大化（NDCG，AUC 等）。请注意，如果您指定了多个评估指标，则所有评估指标都将用于提前停止。

预测

已训练或加载的模型可以对数据集执行预测：

```
# 7 entities, each contains 10 features
data = np.random.rand(7, 10)ypred = bst.predict(data)
```

如果在训练期间启用了早期停止，您可以通过以下方式从最佳迭代中获得预测 `bst.best_iteration`：

```
ypred = bst.predict(data, num_iteration=bst.best_iteration)
```

LightGBM-Api

数据结构

class lightgbm.Dataset（data，label = None，reference = None，weight = None，group = None，init_score = None，silent = False，feature_name =’auto’，categorical_feature =’auto’，params = None，free_raw_data = True ）[来源]

基地： `object`

LightGBM 中的数据集。

初始化数据集。

参数：

- **data**（*字符串，numpy 数组，pandas DataFrame ，scipy.sparse 或 numpy 数组列表*） - 数据集的数据源。如果是 string，则表示 txt 文件的路径。
- **label**（*list ，numpy 1-D 数组，pandas Series / one-column DataFrame 或 None ，optional （default = None ）*） - 数据的标签。
- **reference**（数据集*或无，可选（默认=无）*） - 如果这是用于验证的数据集，则应使用训练数据作为参考。
- **权重**（*列表，numpy 1-D 数组，pandas 系列或无，可选（默认=无）*） - 每个实例的权重。
- **group**（*list ，numpy 1-D 数组，pandas Series 或 None ，可选（默认= None ）*） - 数据集的组/查询大小。
- **init_score**（*list ，numpy 1-D 数组，pandas Series 或 None ，可选（默认=无）*） - 数据集的初始分数。
- **silent**（*bool ，optional （default = False ）*） - 是否在构造期间打印消息。
- **feature_name**（*字符串列表或'auto' ，可选（默认=“auto”）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int ，或'auto' ，可选（默认=“auto”）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值

（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。

- **params**（*dict 或 None，可选（默认=无）*） - 数据集的其他参数。
- **free_raw_data**（*bool，optional （default = True ）*） - 如果为 True，则在构造内部数据集后释放原始数据。

construct（）[来源]

懒惰初始化。

返回： 自构造的数据集对象。

返回类型： 数据集

create_valid（data，label = None，weight = None，group = None，init_score = None，silent = False，params = None ）[source]

创建与当前数据集对齐的验证数据。

参数：

- **data**（*字符串，numpy 数组，pandas DataFrame，scipy.sparse 或 numpy 数组列表*） - 数据集的数据源。如果是 string，则表示 txt 文件的路径。
- **label**（*list，numpy 1-D 数组，pandas Series / one-column DataFrame 或 None，optional （default = None ）*） - 数据的标签。
- **权重**（*列表，numpy 1-D 数组，pandas 系列或无，可选（默认=无）*） - 每个实例的权重。
- **group**（*list，numpy 1-D 数组，pandas Series 或 None，可选（默认= None ）*） - 数据集的组/查询大小。
- **init_score**（*list，numpy 1-D 数组，pandas Series 或 None，可选（默认=无）*） - 数据集的初始分数。
- **silent**（*bool，optional （default = False ）*） - 是否在构造期间打印消息。
- **params**（*dict 或 None，可选（默认=无）*） - 验证数据集的其他参数。

返回： 有效 - 参考 self 的验证数据集。

返回类型： 数据集

get_field（field_name ）[来源]

从数据集获取财产。

参数： field_name（*string*） - 信息的字段名称。

返回： info - 包含数据集信息的 numpy 数组。

返回类型： numpy 数组

get_group（）[来源]

获取数据集的组。

返回： group - 每组的组大小。

返回类型： numpy 数组或无

get_init_score（）[来源]

获取数据集的初始分数。

返回： init_score - Booster 的初始分数。

返回类型： numpy 数组或无

get_label（）[来源]

获取数据集的标签。

返回： label - 数据集中的标签信息。

返回类型： numpy 数组或无

get_ref_chain（ref_limit = 100 ）[来源]

获取数据集对象链。

从 r 开始，然后转到 r.reference（如果存在），然后转到 r.reference.reference 等，直到我们命中 `ref_limit` 或引用循环。

参数： ref_limit（*int* ，*optional* （*default = 100* ））- 引用的限制数。

返回： ref_chain - 数据集的引用链。

返回类型： 数据集

get_weight（）[来源]

获得数据集的权重。

返回： weight - 数据集中每个数据点的**权**重。

返回类型： numpy 数组或无

num_data（）[来源]

获取数据集中的行数。

返回： number_of_rows - 数据集中的行数。

返回类型： INT

num_feature（）[来源]

获取数据集中的列数（要素）。

返回： number_of_columns - 数据集中的列数（要素）。

返回类型： INT

save_binary（文件名）[来源]

将数据集保存到二进制文件。

参数： filename（*string*） - 输出文件的名称。

返回： **自我** - 回归自我。

返回类型： 数据集

set_categorical_feature（categorical_feature ）[来源]

设置分类功能。

参数： categorical_feature（*int 或 strings 的列表*） - 分类特征的名称或索引。

返回： self - 具有设置分类功能的数据集。

返回类型： 数据集

set_feature_name（feature_name ）[来源]

设置功能名称。

参数： feature_name（*字符串列表*） - 功能名称。

返回： self - 具有设置功能名称的数据集。

返回类型： 数据集

set_field（field_name，data ）[来源]

将属性设置为数据集。

参数：

- **field_name**（*string*） - 信息的字段名称。
- **data**（*list，numpy 1-D array，pandas Series 或 None*） - 要设置的数据数组。

返回： **self** - 具有 set 属性的数据集。

返回类型： 数据集

set_group（组）[来源]

设置数据集的组大小（用于排名）。

参数： **group**（*list ，numpy 1-D array ，pandas Series 或 None*） - 每组的组大小。

返回： **self** - 具有 set group 的数据集。

返回类型： 数据集

set_init_score（init_score ）[来源]

将 Booster 的初始分数设置为从。

参数： **init_score**（*list ，numpy 1-D 数组，pandas Series 或 None*） - Booster 的初始分数。

返回： **self** - 具有 set init 分数的数据集。

返回类型： 数据集

set_label（标签）[来源]

设置数据集的标签。

参数： **label**（*列表，numpy 1-D 数组，pandas Series / one-column DataFrame 或 None*） - 要设置为 Dataset 的标签信息。

返回： **self** - 带有 set label 的数据集。

返回类型： 数据集

set_reference（参考）[来源]

设置参考数据集。

参数： **reference**（Dataset） - 用作构造当前数据集的模板的引用。

返回： **self** - 具有 set reference 的数据集。

返回类型： 数据集

set_weight（重量）[来源]

设置每个实例的权重。

参数： **权重**（*列表，numpy 1-D 数组，pandas 系列或无*） - 要为每个数据点设置

权重。

返回： self - 设置权重的数据集。

返回类型： 数据集

subset（used_indices，params = None ）[来源]

获取当前数据集的子集。

参数：
- **used_indices**（*int 列表*） - 用于创建子集的索引。
- **params**（*dict 或 None，可选（默认=无）*） - 这些参数将传递给 Dataset 构造函数。

返回： **subset** - 当前数据集的子集。

返回类型： 数据集

class lightgbm.Booster（params = None，train_set = None，model_file = None，silent = False ）[source]

基地： object

LightGBM 中的助推器。

初始化助推器。

参数：
- **params**（*dict 或 None，可选（默认=无）*） - Booster 的参数。
- **train_set**（数据集*或无，可选（默认=无）*） - 训练数据集。
- **model_file**（*字符串或无，可选（默认=无）*） - 模型文件的路径。
- **silent**（*bool，optional （default = False ）*） - 是否在构造期间打印消息。

add_valid（数据，名称）[来源]

添加验证数据。

参数：
- **data**（数据集） - 验证数据。
- **name**（*字符串*） - 验证数据的名称。

返回： **self** - Booster，带有设置验证数据。

返回类型： 加速器

attr（关键）[来源]

从 Booster 获取属性字符串。

参数： key（*string*） - 属性的名称。

返回： value - 属性值。如果属性不存在，则返回 None。

返回类型： 字符串或无

current_iteration（）[来源]

获取当前迭代的索引。

返回： cur_iter - 当前迭代的索引。

返回类型： INT

dump_model（num_iteration = None，start_iteration = 0 ）[来源]

Dump Booster 转换为 JSON 格式。

参数：

- **num_iteration**（*int 或 None ， optional （default = None ）*） - 应转储的迭代索引。如果为 None，如果存在最佳迭代，则将其转储; 否则，所有迭代都被转储。如果<= 0，则转储所有迭代。
- **start_iteration**（*int ， optional （default = 0 ）*） - 启动应转储的迭代索引。

返回： **json_repr** - Booster 的 JSON 格式。

返回类型： 字典

eval（数据，名称，feval =无）[来源]

评估数据。

参数：

- **data**（Dataset） - 用于评估的数据。
- **name**（*字符串*） - 数据的名称。
- **feval**（*可调用或无，可选（默认=无）*） - 自定义评估功能。应该接受两个参数：preds，train_data 和 return（eval_name，eval_result，is_higher_better）或这样的元组列表。对于多类任务，preds 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 类获得第 i 行 pred，则访问方式是 preds [j * num_data + i]。

返回： **结果** - 列出评估结果。

返回类型： 名单

eval_train（feval =无）[来源]

评估培训数据。

参数：
feval（*可调用或无，可选（默认=无）*） - 自定义评估功能。应该接受两个参数：preds，train_data 和 return（eval_name，eval_result，is_higher_better）或这样的元组列表。对于多类任务，preds 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 类获得第 i 行 pred，则访问方式是 preds [j * num_data + i]。

返回：
结果 - 列出评估结果。

返回类型：
名单

eval_valid（feval =无）[来源]

评估验证数据。

参数：
feval（*可调用或无，可选（默认=无）*） - 自定义评估功能。应该接受两个参数：preds，train_data 和 return（eval_name，eval_result，is_higher_better）或这样的元组列表。对于多类任务，preds 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 类获得第 i 行 pred，则访问方式是 preds [j * num_data + i]。

返回：
结果 - 列出评估结果。

返回类型：
名单

feature_importance（importance_type ='split'，iteration = None ）[来源]

获取功能重要性。

参数：
- **importance_type**（*string ，optional （default ="split" ）*） - 如何计算重要性。如果"拆分"，则结果包含在模型中使用该要素的次数。如果"获得"，则结果包含使用该功能的分割的总增益。
- **iteration**（*int 或 None ，optional （default = None ）*） - 限制特征重要性计算中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。

返回：
result - 具有要素重要性的数组。

返回类型：
numpy 数组

feature_name（）[来源]

获取功能名称。

返回：
result - 包含要素名称的列表。

返回类型：
名单

free_dataset（）[来源]

免费助推器的数据集。

返回： 没有数据集的**自我**助推器。

返回类型： 加速器

free_network（）[来源]

免费助推器的网络。

返回： **自我**增强与自由网络。

返回类型： 加速器

get_leaf_output（tree_id，leaf_id ）[来源]

获取叶子的输出。

参数：

- **tree_id**（*int*） - 树的索引。
- **leaf_id**（*int*） - 树中叶子的索引。

返回： **result** - 叶子的输出。

返回类型： 浮动

model_from_string（model_str，verbose = True ）[来源]

从字符串加载 Booster。

参数：

- **model_str**（*string*） - 将从此字符串加载模型。
- **verbose**（*bool*，*optional*（*default = True*）） - 是否在加载模型时打印消息。

返回： **自我**加载的助推器对象。

返回类型： 加速器

model_to_string（num_iteration = None，start_iteration = 0 ）[来源]

将 Booster 保存到字符串。

参数：

- **num_iteration**（*int 或 None*，*optional*（*default = None*）） - 应保存的迭代索引。如果为 None，如果存在最佳迭代，则保存; 否则，将保存所有迭代。如果<= 0，则保存所有迭代。
- **start_iteration**（*int*，*optional*（*default = 0*）） - 启动应保存的迭代索引。

返回： **str_repr** - Booster 的字符串表示形式。

返回类型： 串

num_feature（）[来源]

获取功能数量。

返回： num_feature - **要素**数量。

返回类型： INT

num_model_per_iteration（）[来源]

获取每次迭代的模型数量。

返回： model_per_iter - 每次迭代的模型数。

返回类型： INT

num_trees（）[来源]

获取弱子模型的数量。

返回： num_trees - 弱子模型的数量。

返回类型： INT

predict（data，num_iteration = None，raw_score = False，pred_leaf = False，pred_contrib = False，data_has_header = False，is_reshape = True，** kwargs ）[来源]

做一个预测。

参数：

- **data**（*string，numpy array，pandas DataFrame 或 scipy.sparse*） - 用于预测的数据源。如果是 string，则表示 txt 文件的路径。
- **num_iteration**（*int 或 None，optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有迭代。如果<= 0，则使用所有迭代（无限制）。
- **raw_score**（*bool，optional （default = False ）*） - 是否预测原始分数。
- **pred_leaf**（*bool，optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool，optional （default = False ）*） -

是否预测功能贡献。

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- **data_has_header**（*bool，optional （default = False）*） - 数据是否有标题。仅在数据为字符串时使用。
- **is_reshape**（*bool，optional （default = True）*） - 如果为 True，则结果重新赋值为[nrow，ncol]。
- ** **kwargs** - 预测的其他参数。

返回： **结果** - 预测结果。

返回类型： numpy 数组

refit（data，label，decay_rate = 0.9，** kwargs ）[来源]

通过新数据重新安装现有的 Booster。

参数：

- **data**（*string，numpy array，pandas DataFrame 或 scipy.sparse*） - refit 的数据源。如果是 string，则表示 txt 文件的路径。
- **label**（*list，numpy 1-D array 或 pandas Series / one-column DataFrame*） - refit 标签。
- **decay_rate**（*float，optional （默认值= 0.9）*） - 改装的衰减率，将用于改装树。

```
leaf_output = decay_rate * old_leaf_output + (1.0 - decay_rate) * new_leaf_output
```

- ** **kwargs** - 改装的其他参数。这些参数将传递给 `predict` 方法。

返回： **结果** - 改装助推器。

返回类型： 加速器

reset_parameter（参数）[来源]

重置 Booster 的参数。

参数： params（*dict*） - Booster 的新参数。

返回： 具有新参数的**自我**助推器。

返回类型： 加速器

rollback_one_iter（）[来源]

回滚一次迭代。

返回： **自我**助推器，回滚一次迭代。

返回类型： 加速器

save_model（filename，num_iteration = None，start_iteration = 0 ）[来源]

将 Booster 保存到文件中。

参数：

- **filename**（*string*） - 保存 Booster 的文件名。
- **num_iteration** (*int or None, optional (default=None)*) – Index of the iteration that should be saved. If None, if the best iteration exists, it is saved; otherwise, all iterations are saved. If <= 0, all iterations are saved.
- **start_iteration** (*int, optional (default=0)*) – Start index of the iteration that should be saved.

Returns: **self** – Returns self.

Return type: Booster

set_attr(**kwargs)[source]

Set attributes to the Booster.

Parameters: **kwargs - The attributes to set. Setting a value to None deletes an attribute.

Returns: self - Booster with set attributes.

Return type: Booster

set_network(machines, local_listen_port=12400, listen_time_out=120, num_machines=1)[source]

Set the network configuration.

Parameters:

- **machines** (*list, set or string*) – Names of machines.
- **local_listen_port** (*int, optional (default=12400)*) – TCP listen port for

local machines.

- **listen_time_out** (*int, optional (default=120)*) – Socket time-out in minutes.
- **num_machines** (*int, optional (default=1)*) – The number of machines for parallel learning application.

Returns: **self** – Booster with set network.

Return type: Booster

set_train_data_name(name)[source]

Set the name to the training Dataset.

Parameters: **name** (*string*) – Name for the training Dataset.

Returns: **self** – Booster with set training Dataset name.

Return type: Booster

shuffle_models(start_iteration=0, end_iteration=-1)[source]

Shuffle models.

Parameters:

- **start_iteration** (*int, optional (default=0)*) – The first iteration that will be shuffled.
- **end_iteration** (*int, optional (default=-1)*) – The last iteration that will be shuffled. If <= 0, means the last available iteration.

Returns: **self** – Booster with shuffled models.

Return type: Booster

update(train_set=None, fobj=None)[source]

Update Booster for one iteration.

Parameters:

- **train_set** (Dataset *or None, optional (default=None)*) – Training data. If None, last training data is used.
- **fobj** (*callable or None, optional (default=None)*) –

Customized objective function.

-

  For multi-class task, the score is group by class_id first, then group by row_id. If you want to get i-th row score in j-th class, the access way is score[j * num_data + i] and you should group grad and hess in this way as well.

-

Returns: **is_finished** – Whether the update was successfully finished.

Return type: bool

训练

lightgbm.train（params，train_set，num_boost_round = 100，valid_sets = None，valid_names = None，fobj = None，feval = None，init_model = None，feature_name =’auto’，categorical_feature =’auto’，early_stopping_rounds = None，evals_result = None，verbose_eval = True，learning_rates = None，keep_training_booster = False，callbacks = None ）[source]

使用给定参数执行培训。

参数：

- **params**（*dict*） - 训练参数。
- **train_set**（数据集） - 要训练的数据。
- **num_boost_round**（*int，optional（default = 100）*） - 增强迭代次数。
- **valid_sets**（*数据集列表或无，可选（默认=无）*） - 训练期间要评估的数据列表。
- **valid_names**（*字符串列表或无，可选（默认=无）*） - 名称 `valid_sets`。
- **fobj**（*可调用或无，可选（默认=无）*） - 自定义目标函数。
- **feval**（*可调用或无，可选（默认=无）*） - 自定义评估功能。应该接受两个参数：preds，train_data 和 return（eval_name，eval_result，is_higher_better）或这样的元组列表。对于多类任务，preds 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 类获得第 i 行 pred，则访问方式是 preds [j * num_data + i]。忽略对应于使用目的的默认度量，则设置 `metric` 参数为字符串 `"None"` 在 `params`。
- **init_model**（*string，* Booster *或 None，可选（默认=无）*） - 用于继续

训练的 LightGBM 模型或 Booster 实例的文件名。

- **feature_name**（*字符串列表或'auto'，可选（默认=“auto”）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int，或'auto'，可选（默认=“auto”）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **early_stopping_rounds**（*int 或 None，可选（默认=无）*） - 激活提前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。`best_iteration` 如果通过设置启用了早期停止逻辑，则具有最佳性能的迭代索引将保存在字段中 `early_stopping_rounds`。
- **evals_result**（*dict 或 None，可选（默认=无）*） -

该词典用于存储所有项目的所有评估结果 `valid_sets`。

-

例

-

使用 a `valid_sets`= [valid_set，train_set]， `valid_names`= ['eval'，'train'] 和 a `params`= {'metric'：'logloss'}返回{'train'：{'logloss'：['0.48253'，'0.35953' ，...]}，'eval'：{'logloss'：['0.480385'，'0.357756'，...]}}。

-

- **verbose_eval**（*bool 或 int，optional （default = True ）*） -

至少需要一个验证数据。如果为 True，则在每个提升阶段打印有效集上的 eval 度量。如果为 int，则在每个 `verbose_eval` 提升阶段打印有效集上的 eval 度量。`early_stopping_rounds` 还打印了通过使用找到的最后一个增强阶段或增强阶段。

-

例

-

在 verbose_eval= 4 且至少有一个项目的情况下 valid_sets，每 4 个（而不是 1 个）增强阶段打印评估度量。

-
- **learning_rates**（*列表，可调用或无，可选（默认=无）*） - 每个助推轮的学习率列表或 learning_rate 根据当前轮数计算的自定义函数（例如，产生学习率衰减）。
- **keep_training_booster**（*bool，optional （default = False ）*） - 返回的 Booster 是否会用于继续训练。如果为 False，则返回值将在返回之前转换为 _InnerPredictor。您仍然可以使用_InnerPredictor 作为 init_model 将来的继续培训。
- **回调**（*callables 列表或 None，可选（default = None ）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **助推器** - 训练有素的助推器模型。

返回类型： 加速器

---

lightgbm.cv（params，train_set，num_boost_round = 100，folds = None，nfold = 5，stratified = True，shuffle = True，metrics = None，fobj = None，feval = None，init_model = None，feature_name =’auto’，categorical_feature =’auto’，early_stopping_rounds = None，fpreproc = None，verbose_eval = None，show_stdv = True，seed = 0，callbacks = None ）[source]

使用给定的参数执行交叉验证。

参数：

- **params**（*dict*） - 助推器的参数。
- **train_set**（数据集） - 要训练的数据。
- **num_boost_round**（*int，optional （default = 100 ）*） - 增强迭代次数。
- **folds**（*（train_idx，test_idx ）元组的生成器或迭代器，scikit-learn splitter 对象或 None，可选（默认=无）*） - 如果是生成器或迭代器，它应该为每个折叠产生训练和测试索引。如果是 object，它应该是 scikit-learn splitter 类之一（http://scikit-learn.org/stable/modules/classes.html#splitter-classes）并且有方法。此参数优先于其他数据拆分参数。split
- **nfold**（*int，optional （default = 5 ）*） - CV 中的折叠数。
- **分层**（*bool，optional （默认= True ）*） - 是否执行分层抽样。
- **shuffle**（*bool，optional （default = True ）*） - 是否在分割数据之前进行

随机播放。

- **metrics**（*字符串，字符串列表或 None，可选（默认=无）*） - CV 时要监控的评估指标。如果不是 None，`params` 则将覆盖度量标准。
- **fobj**（*可调用或无，可选（默认=无）*） - 自定义目标函数。
- **feval**（*可调用或无，可选（默认=无）*） - 自定义评估功能。应该接受两个参数：preds，train_data 和 return（eval_name，eval_result，is_higher_better）或这样的元组列表。对于多类任务，preds 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 类获得第 i 行 pred，则访问方式是 preds [j * num_data + i]。要忽略与使用目标对应的默认度量标准，请设置 `metrics` 为字符串 `"None"`。
- **init_model**（*string，* Booster *或 None，可选（默认=无）*） - 用于继续训练的 LightGBM 模型或 Booster 实例的文件名。
- **feature_name**（*字符串列表或'auto'，可选（默认="auto"）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int，或'auto'，可选（默认="auto"）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **early_stopping_rounds**（*int 或 None，可选（默认=无）*） - 激活提前停止。CV 评分需要至少每 `early_stopping_rounds` 一轮都要改进才能继续。至少需要一个指标。如果有多个，将检查所有这些。评估历史记录中的最后一个条目是最佳迭代中的条目。
- **fpreproc**（*可调用或无，可选（默认=无）*） - 采用（dtrain，dtest，params）并返回其转换版本的预处理函数。
- **verbose_eval**（*bool，int 或 None，可选（默认=无）*） - 是否显示进度。如果为 None，则返回 np.ndarray 时将显示进度。如果为 True，则每个提升阶段都会显示进度。如果为 int，则会在每个给定的 `verbose_eval` 提升阶段显示进度。
- **show_stdv**（*bool，optional （default = True）*） - 是否显示正在进行的标准偏差。结果不受此参数的影响，并且始终包含 std。
- **seed**（*int，optional （default = 0）*） - 用于生成折叠的种子（传递给 numpy.random.seed）。
- **回调**（*callables 列表或 None，可选（default = None）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **eval_hist** - 评估历史。字典具有以下格式：{'metric1-mean'：[values]，'metric1-stdv'：[values]，'metric2-mean'：[values]，'metric2-stdv'：[values]，...} 。

返回类型： 字典

Scikit-learnAPI

class **lightgbm.LGBMModel**（boosting_type ='gbdt'，num_leaves = 31，max_depth = -1，learning_rate = 0.1，n_estimators = 100，subsample_for_bin = 200000，objective = None，class_weight = None，min_split_gain = 0.0，min_child_weight = 0.001，min_child_samples = 20，subsample = 1.0，subsample_freq = 0，colsample_bytree = 1.0，reg_alpha = 0.0，reg_lambda = 0.0，random_state = None，n_jobs = -1，silent = True，importance_type ='split'，** kwargs ）[来源]

基地： `object`

为 LightGBM 实施 scikit-learn API。

构建梯度增强模型。

参数：

- **boosting_type**（*string*，*optional*（*default ='gbdt'*））- 'gbdt'，传统的 Gradient Boosting 决策树。'dart'，Dropouts 遇到多个加法回归树。'高斯'，基于渐变的单面采样。'rf'，随机森林。
- **num_leaves**（*int*，*optional*（*default = 31*））- 基础学习者的最大树叶。
- **max_depth**（*int*，*optional*（*default = -1*））- 基础学习者的最大树深度，-1 表示没有限制。
- **learning_rate**（*float*，*optional*（*默认值= 0.1*））- 提高学习率。您可以使用方法的 `callbacks` 参数 `fit` 来缩小/调整使用 `reset_parameter` 回调的训练中的学习率。请注意，这将忽略 `learning_rate` 训练中的参数。
- **n_estimators**（*int*，*optional*（*default = 100*））- 要适应的提升树数。
- **subsample_for_bin**（*int*，*optional*（*default = 200000*））- 构造箱的样本数。
- **objective**（*字符串，可调用或无，可选（默认=无）*）- 指定学习任务和相应的学习目标或要使用的自定义目标函数（请参阅下面的注释）。默认值：LGBMRegressor 的'regression'，LGBMClassifier 的'binary'或'multiclass'，LGBMRanker 的'lambdarank'。
- **class_weight**（*dict*，*'balanced' 或 None*，*可选（默认=无）*）- 与表单中的类关联的权重。仅将此参数用于多类分类任务；对于二进制分类任务，您可以使用或参数。“平衡”模式使用 y 的值自动调整与输入数据中的类频率成反比的权

重。如果为 None，则所有类都应该具有权重 1。注意，如果指定，这些权重将乘以（通过该方法）。

```
{class_label: weight}is_unbalancescale_pos_weightn_samples / (n_classes * np.bincount(y))sample_weightfitsample_weight
```

- **min_split_gain**（*float，optional （default = 0。）*） - 在树的叶节点上进行进一步分区所需的最小损耗减少。
- **min_child_weight**（*float，optional （default = 1e-3 ）*） - 子（叶）中所需的实例权重（粗体）的最小总和。
- **min_child_samples**（*int，optional （default = 20 ）*） - 子**节点**（叶子）中所需的最小数据量。
- **subsample**（*float，optional （default = 1。）*） - 训练实例的子采样率。
- **subsample_freq**（*int，optional （default = 0 ）*） - 子样本的频率，<= 0 表示不启用。
- **colsample_bytree**（*float，optional （default = 1。）*） - 构造每个树时列的子采样率。
- **reg_alpha**（*float，optional （default = 0。）*） - 权重上的 L1 正则化项。
- **reg_lambda**（*float，optional （default = 0。）*） - 权重的 L2 正则项。
- **random_state**（*int 或 None，可选（默认=无）*） - 随机数种子。如果为 None，将使用 C ++代码中的默认种子。
- **n_jobs**（*int，optional （default = -1 ）*） - 并行线程数。
- **silent**（*bool，optional （default = True ）*） - 是否在运行 boost 时打印消息。
- **importance_type**（*string，optional （default ='split' ）*） - 要填充的要素重要性的类型 `feature_importances_`。如果是“拆分”，则结果包含在模型中使用该要素的次数。如果为'gain'，则结果包含使用该功能的分割的总增益。
- ** **kwargs** -

模型的其他参数。查看 http://lightgbm.readthedocs.io/en/latest/Parameters.html 以获取更多参数。

-

注意

-

sklearn 不支持** kwargs，可能会导致意外问题。

-

n_features_

int - 拟合模型的特征数。

classes_

shape of shape = [n_classes] - 类标签数组（仅用于分类问题）。

n_classes_

int - 类的数量（仅用于分类问题）。

best_score_

dict 或 None - 拟合模型的最佳分数。

best_iteration_

int 或 None - `early_stopping_rounds` 已指定拟合模型的最佳迭代。

objective_

string 或 callable - 适合此模型时使用的具体目标。

booster_

助推器 - 这种模式的潜在助推器。

evals_result_

dict 或 None - 如果 `early_stopping_rounds` 已指定评估结果。

feature_importances_

shape of array = [n_features] - 要素重要性（越高，功能越重要）。

注意

可以为 `objective` 参数提供自定义目标函数。在这种情况下，它应该有签名 或 ：`objective(y_true, y_pred) -> grad, hessobjective(y_true, y_pred, group) -> grad, hess`

y_true ：shape-like = [n_samples]

目标值。

y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）

预测值。

group ：类似于 array

组/查询数据，用于排名任务。

grad ：类似于 shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）

每个采样点的梯度值。

hess ： shape-like of shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）

每个样本点的二阶导数的值。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 节获得第 i 行 y_pred，那么访问方式是 y_pred [j * num_data + i]，你也应该以这种方式对 grad 和 hess 进行分组。

best_iteration_

获得拟合模型的最佳迭代。

best_score_

获得合适模型的最佳分数。

booster_

获得此模型的底层 lightgbm Booster。

evals_result_

获得评估结果。

feature_importances_

获取功能重要性。

注意

用于标准化为 1 的 sklearn 接口中的特征重要性，它在 2.0.4 之后被弃用，现在与 Booster.feature_importance（）相同。 `importance_type` 属性被传递给函数以配置要提取的重要性值的类型。

fit（X，y，sample_weight =无，init_score =无，组=无，eval_set =无，eval_names =无，eval_sample_weight =无，eval_class_weight =无，eval_init_score =无，eval_group =无，eval_metric =无，early_stopping_rounds =无，详细= True，feature_name ='auto'，categorical_feature ='auto'，callbacks = None ）[source]

从训练集（X，y）构建梯度增强模型。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入要素矩阵。
- **y**（*array-like of shape = [n_samples]*） - 目标值（分类中的类标签，回归中的实数）。
- **sample_weight**（*shape = array = [n_samples]或 None ，可选（默认=无）*） - 训练数据的权重。
- **init_score**（*shape = array = [n_samples]或 None ，可选（默认=无）*）

- 训练数据的初始分数。

- **group**（*array-like 或 None ， optional （default = None ）*） - 训练数据的组数据。
- **eval_set**（*list 或 None ， optional （default = None ）*） - 用作验证集的（X，y）元组对的列表。
- **eval_names**（*字符串列表或 None ， 可选（默认=无）*） - eval_set 的名称。
- **eval_sample_weight**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的权重。
- **eval_class_weight**（*列表或无，可选（默认=无）*） - eval 数据的类权重。
- **eval_init_score**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的初始分数。
- **eval_group**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的组数据。
- **eval_metric**（*字符串，字符串列表，可调用或无，可选（默认=无）*） - 如果是字符串，它应该是要使用的内置评估指标。如果是可调用的，则应该是自定义评估指标，有关详细信息，请参阅下面的注释。在任何一种情况下，`metric` 都将评估和使用模型参数。默认值：LGBMRegressor 为'l2'，LGBMClassifier 为'logloss'，LGBMRanker 为'ndcg'。
- **early_stopping_rounds**（*int 或 None ， 可选（默认=无）*） - 激活提前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。
- **verbose**（*bool 或 int ， 可选（默认= True ）*） -

至少需要一个评估数据。如果为 True，则在每个提升阶段打印 eval 集上的 eval 度量。如果为 int，则在每个 `verbose` 提升阶段打印 eval 集上的 eval 度量。`early_stopping_rounds` 还打印了通过使用找到的最后一个增强阶段或增强阶段。

-

例

-

在 `verbose`= 4 且至少有一个项目的情况下 `eval_set`，每 4 个（而不是 1 个）增强阶段打印评估度量。

-
- **feature_name**（*字符串列表或'auto'，可选（默认='auto'）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int，或'auto'，可选（默认='auto'）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **回调**（*回调函数列表或无，可选（默认=无）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **自我** - 回归自我。

返回类型： 宾语

注意

自定义 eval 函数需要具有以下签名的可调用： ，或者 返回（eval_name，eval_result，is_bigger_better）或（eval_name，eval_result，is_bigger_better）列表：

```
func(y_true, y_pred)func(y_true, y_pred, weight)func(y_true, y_pred, weight, group)
```

> y_true ：shape-like = [n_samples]
>
> 目标值。
>
> y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> 重量：形状的数组= [n_samples]
>
> 样品的重量。
>
> group ：类似于 array
>
> 组/查询数据，用于排名任务。
>
> eval_name ：string
>
> 评价的名称。
>
> eval_result ：浮动

评估结果。

is_bigger_better : bool

eval 结果更好，例如 AUC 更大更好。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 个类中获得第 i 行 y_pred，则访问方式是 y_pred [j * num_data + i]。

get_params（深=真）[来源]

获取此估算工具的参数。

参数： deep（*bool ，optional （default = True ）*） - 如果为 True，将返回此估计器的参数并包含作为估算器的子对象。

返回： params - 映射到其值的参数名称。

返回类型： 字典

n_features_

获取适合型号的功能数量。

objective_

获得适合此模型时使用的具体目标。

predict（X，raw_score = False，num_iteration = None，pred_leaf = False，pred_contrib = False，** kwargs ）[来源]

返回每个样本的预测值。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入*要素*矩阵。
- **raw_score**（*bool ， optional （default = False ）*） - 是否预测原始分数。
- **num_iteration**（*int 或 None ， optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。
- **pred_leaf**（*bool ， optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool ， optional （default = False ）*） - 是否预测功能贡献。

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- ** **kwargs** - 预测的其他参数。

返回：

- **predicted_result**（*阵列状的= [N_SAMPLES 次]或形状= [N_SAMPLES 次，n_classes]*）-的预测值。
- **X_leaves**（*shape = array = [n_samples，n_trees]或 shape [n_samples，n_trees * n_classes]*） - 如果 `pred_leaf=True`，每个样本的每个树的预测叶。
- **X_SHAP_values**（*shape = array = [n_samples， n_features + 1]或 shape [n_samples， （n_features + 1）* n_classes]*） - 如果 `pred_contrib=True`，每个样本的每个特征贡献。

set_params（** params ）[来源]

设置此估算器的参数。

参数： ** params - 带有新值的参数名称。

返回： **自我** - 回归自我。

返回类型： 宾语

---

class**lightgbm.LGBMClassifier**（boosting_type ='gbdt'，num_leaves = 31，max_depth = -1，learning_rate = 0.1，n_estimators = 100，subsample_for_bin = 200000，objective = None，class_weight = None，min_split_gain = 0.0，min_child_weight = 0.001，min_child_samples = 20，subsample = 1.0，subsample_freq = 0，colsample_bytree = 1.0，reg_alpha = 0.0，reg_lambda = 0.0，random_state = None，n_jobs = -1，silent = True，importance_type ='split'，** kwargs ）[来源]

基地：**lightgbm.sklearn.LGBMModel**，**object**

LightGBM 分类器。

构建梯度增强模型。

参数：

- **boosting_type**（*string，optional（default ='gbdt'）*） - 'gbdt'，传统的 Gradient Boosting 决策树。'dart'，Dropouts 遇到多个加法回归树。'高斯'，基于渐变的单面采样。'rf'，随机森林。
- **num_leaves**（*int， optional （default = 31 ）*） - 基础学习者的最大树叶。
- **max_depth**（*int， optional （default = -1 ）*） - 基础学习者的最大树深度，

-1 表示没有限制。

- **learning_rate**（*float，optional （默认值= 0.1 ）*） - 提高学习率。您可以使用方法的 `callbacks` 参数 `fit` 来缩小/调整使用 `reset_parameter` 回调的训练中的学习率。请注意，这将忽略 `learning_rate` 训练中的参数。
- **n_estimators**（*int，optional （default = 100 ）*） - 要适应的提升树数。
- **subsample_for_bin**（*int，optional （default = 200000 ）*） - 构造箱的样本数。
- **objective**（*字符串，可调用或无，可选（默认=无）*） - 指定学习任务和相应的学习目标或要使用的自定义目标函数（请参阅下面的注释）。默认值：LGBMRegressor 的'regression'，LGBMClassifier 的'binary'或'multiclass'，LGBMRanker 的'lambdarank'。
- **class_weight**（*dict，'balanced' 或 None，可选（默认=无）*） - 与表单中的类关联的权重。仅将此参数用于多类分类任务; 对于二进制分类任务，您可以使用或参数。“平衡”模式使用 y 的值自动调整与输入数据中的类频率成反比的权重。如果为 None，则所有类都应该具有权重 1。注意，如果指定，这些权重将乘以（通过该方法）。

  `{class_label: weight}is_unbalancescale_pos_weightn_samples / (n_classes * np.bincount(y))sample_weightfitsample_weight`
- **min_split_gain**（*float，optional （default = 0。）*） - 在树的叶节点上进行进一步分区所需的最小损耗减少。
- **min_child_weight**（*float，optional （default = 1e-3 ）*） - 子（叶）中所需的实例权重（粗体）的最小总和。
- **min_child_samples**（*int，optional （default = 20 ）*） - 子**节点**（叶子）中所需的最小数据量。
- **subsample**（*float，optional （default = 1。）*） - 训练实例的子采样率。
- **subsample_freq**（*int，optional （default = 0 ）*） - 子样本的频率，<= 0 表示不启用。
- **colsample_bytree**（*float，optional （default = 1。）*） - 构造每个树时列的子采样率。
- **reg_alpha**（*float，optional （default = 0。）*） - 权重上的 L1 正则化项。
- **reg_lambda**（*float，optional （default = 0。）*） - 权重的 L2 正则项。
- **random_state**（*int 或 None，可选（默认=无）*） - 随机数种子。如果为 None，将使用 C ++代码中的默认种子。
- **n_jobs**（*int，optional （default = -1 ）*） - 并行线程数。
- **silent**（*bool，optional （default = True ）*） - 是否在运行 boost 时打印消息。
- **importance_type**（*string，optional （default ='split' ）*） - 要填充的要素重要性的类型 `feature_importances_`。如果是“拆分”，则结果包含在模型中使用该

要素的次数。如果为'gain'，则结果包含使用该功能的分割的总增益。

- ** kwargs -
模型的其他参数。查看 http://lightgbm.readthedocs.io/en/latest/Parameters.html 以获取更多参数。

-

注意

-
sklearn 不支持** kwargs，可能会导致意外问题。

-

n_features_

int - 拟合模型的特征数。

classes_

shape of shape = [n_classes] - 类标签数组（仅用于分类问题）。

n_classes_

int - 类的数量（仅用于分类问题）。

best_score_

dict 或 None - 拟合模型的最佳分数。

best_iteration_

int 或 None - `early_stopping_rounds` 已指定拟合模型的最佳迭代。

objective_

string 或 callable - 适合此模型时使用的具体目标。

booster_

助推器 - 这种模式的潜在助推器。

evals_result_

dict 或 None - 如果 `early_stopping_rounds` 已指定评估结果。

feature_importances_

shape of array = [n_features] - 要素重要性（越高，功能越重要）。

注意

可以为 `objective` 参数提供自定义目标函数。在这种情况下，它应该有签名 或 ：`objective(y_true, y_pred) -> grad, hessobjective(y_true, y_pred, group) -> grad, hess`

> y_true ：shape-like = [n_samples]
>
> 目标值。
>
> y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> group ：类似于 array
>
> 组/查询数据，用于排名任务。
>
> grad ：类似于 shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 每个采样点的梯度值。
>
> hess ：shape-like of shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 每个样本点的二阶导数的值。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 节获得第 i 行 y_pred，那么访问方式是 y_pred [j * num_data + i]，你也应该以这种方式对 grad 和 hess 进行分组。

best_iteration_

获得拟合模型的最佳迭代。

best_score_

获得合适模型的最佳分数。

booster_

获得此模型的底层 lightgbm Booster。

classes_

获取类标签数组。

evals_result_

获得评估结果。

feature_importances_

获取功能重要性。

注意

用于标准化为 1 的 sklearn 接口中的特征重要性，它在 2.0.4 之后被弃用，现在与 Booster.feature_importance（）相同。 `importance_type` 属性被传递给函数以配置要提取的重要性值的类型。

fit（X，y，sample_weight =无，init_score =无，eval_set =无，eval_names =无，eval_sample_weight =无，eval_class_weight =无，eval_init_score =无，eval_metric =无， early_stopping_rounds =无， verbose = True， feature_name =’auto’，categorical_feature =’auto’，callbacks = None ）[来源]

从训练集（X，y）构建梯度增强模型。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入要素矩阵。
- **y**（*array-like of shape = [n_samples]*） - 目标值（分类中的类标签，回归中的实数）。
- **sample_weight**（*shape = array = [n_samples]或 None ，可选（默认=无）*） - 训练数据的权重。
- **init_score**（*shape = array = [n_samples]或 None ，可选（默认=无）*） - 训练数据的初始分数。
- **group**（*array-like 或 None ，optional （default = None ）*） - 训练数据的组数据。
- **eval_set**（*list 或 None ，optional （default = None ）*） - 用作验证集的（X，y）元组对的列表。
- **eval_names**（*字符串列表或 None ，可选（默认=无）*） - eval_set 的名称。
- **eval_sample_weight**（*数组列表或 None ，可选（默认=无）*） - eval 数据的权重。
- **eval_class_weight**（*列表或无，可选（默认=无）*） - eval 数据的类权重。
- **eval_init_score**（*数组列表或 None ，可选（默认=无）*） - eval 数据的初始分数。
- **eval_group**（*数组列表或 None ，可选（默认=无）*） - eval 数据的组数据。
- **eval_metric**（*字符串，字符串列表，可调用或无，可选（默认=无）*） - 如果是字符串，它应该是要使用的内置评估指标。如果是可调用的，则应该是自定义评估指标，有关详细信息，请参阅下面的注释。在任何一种情况下，`metric` 都将评估和使用模型参数。默认值：LGBMRegressor 为'l2'，LGBMClassifier 为'logloss'，LGBMRanker 为'ndcg'。
- **early_stopping_rounds**（*int 或 None ，可选（默认=无）*） - 激活提

前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。

- **verbose**（*bool 或 int ， 可选（默认= True ）*） -

至少需要一个评估数据。如果为 True，则在每个提升阶段打印 eval 集上的 eval 度量。如果为 int，则在每个 `verbose` 提升阶段打印 eval 集上的 eval 度量。`early_stopping_rounds` 还打印了通过使用找到的最后一个增强阶段或增强阶段。

-

例

-

在 `verbose`= 4 且至少有一个项目的情况下 `eval_set`，每 4 个（而不是 1 个）增强阶段打印评估度量。

-
- **feature_name**（*字符串列表或'auto' ，可选（默认='auto' ）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int ，或'auto' ，可选（默认='auto' ）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **回调**（*回调函数列表或无，可选（默认=无）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **自我** - 回归自我。

返回类型： 宾语

注意

自定义 eval 函数需要具有以下签名的可调用： ，或者 返回（eval_name，eval_result，is_bigger_better）或（eval_name，eval_result，

is_bigger_better）列表：

func(y_true, y_pred)func(y_true, y_pred, weight)func(y_true, y_pred, weight, group)

> y_true ：shape-like = [n_samples]
>
> 目标值。
>
> y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> 重量：形状的数组= [n_samples]
>
> 样品的重量。
>
> group ：类似于 array
>
> 组/查询数据，用于排名任务。
>
> eval_name ：string
>
> 评价的名称。
>
> eval_result ：浮动
>
> 评估结果。
>
> is_bigger_better ：bool
>
> eval 结果更好，例如 AUC 更大更好。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 个类中获得第 i 行 y_pred，则访问方式是 y_pred [j * num_data + i]。

get_params（深=真）

获取此估算工具的参数。

参数：	**deep**（*bool ，optional （default = True ）*） - 如果为 True，将返回此估计器的参数并包含作为估算器的子对象。
返回：	**params** - 映射到其值的参数名称。
返回类型：	字典

n_classes_

获取课程数量。

n_features_

获取适合型号的功能数量。

objective_

获得适合此模型时使用的具体目标。

predict(X, raw_score = False, num_iteration = None, pred_leaf = False, pred_contrib = False, ** kwargs) [来源]

返回每个样本的预测值。

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入*要素*矩阵。
- **raw_score**（*bool ， optional （default = False ）*） - 是否预测原始分数。
- **num_iteration**（*int 或 None ， optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。
- **pred_leaf**（*bool ， optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool ， optional （default = False ）*） -

是否预测功能贡献。

参数：

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- ** **kwargs** - 预测的其他参数。

返回：

- **predicted_result**（*阵列状的= [N_SAMPLES 次]或形状= [N_SAMPLES 次，n_classes]* ）-的预测值。
- **X_leaves**（*shape = array = [n_samples，n_trees]或 shape [n_samples，n_trees * n_classes]*） - 如果 `pred_leaf=True`，每个样本的每个树的预测叶。
- **X_SHAP_values**（*shape = array = [n_samples， n_features + 1]或 shape [n_samples， （n_features + 1 ）* n_classes]*） - 如果 `pred_contrib=True`，每个样本的每个特征贡献。

predict_proba(X, raw_score = False, num_iteration = None, pred_leaf = False, pred_contrib = False, ** kwargs) [来源]

返回每个样本的每个类的预测概率。

- **X**（*形状的数组或稀疏矩阵= [n_samples，n_features]*） - 输入*要素*矩阵。
- **raw_score**（*bool，optional （default = False ）*） - 是否预测原始分数。
- **num_iteration**（*int 或 None，optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。
- **pred_leaf**（*bool，optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool，optional （default = False ）*） -

是否预测功能贡献。

参数：

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- ** **kwargs** - 预测的其他参数。

返回：

- **predicted_probability**（*阵列状=的[N_SAMPLES 次，n_classes]*）-针对每个类别对每个样品的预测概率。
- **X_leaves**（*shape = array = [n_samples，n_trees * n_classes]*） - 如果 `pred_leaf=True`，每个样本的每个树的预测叶。
- **X_SHAP_values**（*shape = array = [n_samples，（n_features + 1）* n_classes]*） - 如果 `pred_contrib=True`，每个样本的每个特征贡献。

set_params（** params ）

设置此估算器的参数。

参数： ** params - 带有新值的参数名称。

返回： **自我** - 回归自我。

返回类型： 宾语

```
classlightgbm.LGBMRegressor（boosting_type ='gbdt'，num_leaves = 31，max_depth = -1，learning_rate = 0.1，n_estimators = 100，subsample_for_bin = 200000，objective = None，class_weight = None，min_split_gain = 0.0，min_child_weight = 0.001，
```

```
min_child_samples = 20, subsample = 1.0, subsample_freq = 0, colsample_bytree =
1.0, reg_alpha = 0.0, reg_lambda = 0.0, random_state = None, n_jobs = -1, silent
= True, importance_type ='split', ** kwargs  )[来源]
```

基地：**lightgbm.sklearn.LGBMModel**，**object**

LightGBM 回归量。

构建梯度增强模型。

参数：

- **boosting_type**（*string，optional（default ='gbdt'）*）- 'gbdt'，传统的 Gradient Boosting 决策树。'dart'，Dropouts 遇到多个加法回归树。'高斯'，基于渐变的单面采样。'rf'，随机森林。
- **num_leaves**（*int，optional（default = 31）*）- 基础学习者的最大树叶。
- **max_depth**（*int，optional（default = -1）*）- 基础学习者的最大树深度，-1 表示没有限制。
- **learning_rate**（*float，optional（默认值= 0.1）*）- 提高学习率。您可以使用方法的 `callbacks` 参数 `fit` 来缩小/调整使用 `reset_parameter` 回调的训练中的学习率。请注意，这将忽略 `learning_rate` 训练中的参数。
- **n_estimators**（*int，optional（default = 100）*）- 要适应的提升树数。
- **subsample_for_bin**（*int，optional（default = 200000）*）- 构造箱的样本数。
- **objective**（*字符串，可调用或无，可选（默认=无）*）- 指定学习任务和相应的学习目标或要使用的自定义目标函数（请参阅下面的注释）。默认值：LGBMRegressor 的'regression'，LGBMClassifier 的'binary'或'multiclass'，LGBMRanker 的'lambdarank'。
- **class_weight**（*dict，'balanced' 或 None，可选（默认=无）*）- 与表单中的类关联的权重。仅将此参数用于多类分类任务；对于二进制分类任务，您可以使用或参数。“平衡”模式使用 y 的值自动调整与输入数据中的类频率成反比的权重。如果为 None，则所有类都应该具有权重 1。注意，如果指定，这些权重将乘以（通过该方法）。

  `{class_label: weight}is_unbalancescale_pos_weightn_samples / (n_classes * np.bincount(y))sample_weightfitsample_weight`
- **min_split_gain**（*float，optional（default = 0。）*）- 在树的叶节点上进行进一步分区所需的最小损耗减少。
- **min_child_weight**（*float，optional（default = 1e-3）*）- 子（叶）中所需的实例权重（粗体）的最小总和。
- **min_child_samples**（*int，optional（default = 20）*）- 子**节点**（叶子）中所需的最小数据量。

- **subsample**（*float，optional （default = 1。）*） - 训练实例的子采样率。
- **subsample_freq**（*int，optional （default = 0 ）*） - 子样本的频率，<= 0 表示不启用。
- **colsample_bytree**（*float，optional （default = 1。）*） - 构造每个树时列的子采样率。
- **reg_alpha**（*float，optional （default = 0。）*） - 权重上的 L1 正则化项。
- **reg_lambda**（*float，optional （default = 0。）*） - 权重的 L2 正则项。
- **random_state**（*int 或 None，可选（默认=无）*） - 随机数种子。如果为 None，将使用 C ++代码中的默认种子。
- **n_jobs**（*int，optional （default = -1 ）*） - 并行线程数。
- **silent**（*bool，optional （default = True ）*） - 是否在运行 boost 时打印消息。
- **importance_type**（*string，optional （default ='split' ）*） - 要填充的要素重要性的类型 `feature_importances_`。如果是“拆分”，则结果包含在模型中使用该要素的次数。如果为'gain'，则结果包含使用该功能的分割的总增益。
- ** **kwargs** -

模型的其他参数。查看 http://lightgbm.readthedocs.io/en/latest/Parameters.html 以获取更多参数。

-

注意

- sklearn 不支持** kwargs，可能会导致意外问题。

-

n_features_

int - 拟合模型的特征数。

classes_

shape of shape = [n_classes] - 类标签数组（仅用于分类问题）。

n_classes_

int - 类的数量（仅用于分类问题）。

best_score_

dict 或 None - 拟合模型的最佳分数。

best_iteration_

int 或 None - `early_stopping_rounds` 已指定拟合模型的最佳迭代。

objective_

string 或 callable - 适合此模型时使用的具体目标。

booster_

助推器 - 这种模式的潜在助推器。

evals_result_

dict 或 None - 如果 `early_stopping_rounds` 已指定评估结果。

feature_importances_

shape of array = [n_features] - 要素重要性（越高，功能越重要）。

注意

可以为 `objective` 参数提供自定义目标函数。在这种情况下，它应该有签名 或 `objective(y_true, y_pred) -> grad, hessobjective(y_true, y_pred, group) -> grad, hess`

> **y_true** : shape-like = [n_samples]
>
> 目标值。
>
> **y_pred** : shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> **group** : 类似于 array
>
> 组/查询数据，用于排名任务。
>
> **grad** : 类似于 shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 每个采样点的梯度值。
>
> **hess** : shape-like of shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 每个样本点的二阶导数的值。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 节获得第 i 行 y_pred，那么访问方式是 y_pred [j * num_data + i]，你也应该以这种方式对 grad 和 hess 进行分组。

best_iteration_

获得拟合模型的最佳迭代。

best_score_

获得合适模型的最佳分数。

booster_

获得此模型的底层 lightgbm Booster。

evals_result_

获得评估结果。

feature_importances_

获取功能重要性。

注意

用于标准化为 1 的 sklearn 接口中的特征重要性，它在 2.0.4 之后被弃用，现在与 Booster.feature_importance（）相同。 `importance_type` 属性被传递给函数以配置要提取的重要性值的类型。

fit（X，y，sample_weight =无，init_score =无，eval_set =无，eval_names =无，eval_sample_weight =无，eval_init_score =无，eval_metric =无，early_stopping_rounds =无，verbose = True，feature_name =’auto’，categorical_feature =’auto’，callbacks = None ）[来源]

从训练集（X，y）构建梯度增强模型。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入要素矩阵。
- **y**（*array-like of shape = [n_samples]*） - 目标值（分类中的类标签，回归中的实数）。
- **sample_weight**（*shape = array = [n_samples]或 None ，可选（默认=无）*） - 训练数据的权重。
- **init_score**（*shape = array = [n_samples]或 None ，可选（默认=无）*） - 训练数据的初始分数。
- **group**（*array-like 或 None ，optional （default = None ）*） - 训练数据的组数据。
- **eval_set**（*list 或 None ，optional （default = None ）*） - 用作验证集的（X，y）元组对的列表。
- **eval_names**（*字符串列表或 None ，可选（默认=无）*） - eval_set 的名称。
- **eval_sample_weight**（*数组列表或 None ，可选（默认=无）*） - eval 数据的权重。
- **eval_init_score**（*数组列表或 None ，可选（默认=无）*） - eval 数据的初始分数。

- **eval_group**（*数组列表或 None，可选（默认=无）*） - eval 数据的组数据。
- **eval_metric**（*字符串，字符串列表，可调用或无，可选（默认=无）*） - 如果是字符串，它应该是要使用的内置评估指标。如果是可调用的，则应该是自定义评估指标，有关详细信息，请参阅下面的注释。在任何一种情况下，`metric` 都将评估和使用模型参数。默认值：LGBMRegressor 为'l2'，LGBMClassifier 为'logloss'，LGBMRanker 为'ndcg'。
- **early_stopping_rounds**（*int 或 None，可选（默认=无）*） - 激活提前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。
- **verbose**（*bool 或 int，可选（默认= True ）*） -

至少需要一个评估数据。如果为 True，则在每个提升阶段打印 eval 集上的 eval 度量。如果为 int，则在每个 `verbose` 提升阶段打印 eval 集上的 eval 度量。`early_stopping_rounds` 还打印了通过使用找到的最后一个增强阶段或增强阶段。

-

例

-

在 `verbose`= 4 且至少有一个项目的情况下 `eval_set`，每 4 个（而不是 1 个）增强阶段打印评估度量。

-

- **feature_name**（*字符串列表或'auto'，可选（默认='auto'）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int，或'auto'，可选（默认='auto'）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **回调**（*回调函数列表或无，可选（默认=无）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **自我** - 回归自我。

返回类型： 宾语

注意

自定义 eval 函数需要具有以下签名的可调用： ，或者 返回（eval_name，eval_result，is_bigger_better）或（eval_name，eval_result，is_bigger_better）列表：

```
func(y_true, y_pred)func(y_true, y_pred, weight)func(y_true, y_pred, weight, group)
```

> y_true ：shape-like = [n_samples]
>
> 目标值。
>
> y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> 重量：形状的数组= [n_samples]
>
> 样品的重量。
>
> group ：类似于 array
>
> 组/查询数据，用于排名任务。
>
> eval_name ：string
>
> 评价的名称。
>
> eval_result ：浮动
>
> 评估结果。
>
> is_bigger_better ：bool
>
> eval 结果更好，例如 AUC 更大更好。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 个类中获得第 i 行 y_pred，则访问方式是 y_pred [j * num_data + i]。

get_params（深=真）

获取此估算工具的参数。

参数： deep（*bool* ，*optional* （*default = True* ））- 如果为 True，将返回此估计器的参数并包含作为估算器的子对象。

返回： params - 映射到其值的参数名称。

返回类型： 字典

n_features_

获取适合型号的功能数量。

objective_

获得适合此模型时使用的具体目标。

predict(X, raw_score=False, num_iteration=None, pred_leaf=False, pred_contrib=False, ** kwargs)

返回每个样本的预测值。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples，n_features]*） - 输入要素矩阵。
- **raw_score**（*bool，optional （default = False ）*） - 是否预测原始分数。
- **num_iteration**（*int 或 None，optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。
- **pred_leaf**（*bool，optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool，optional （default = False ）*） -

  是否预测功能贡献。

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- ** **kwargs** - 预测的其他参数。

返回：

- **predicted_result**（*阵列状的= [N_SAMPLES 次]或形状= [N_SAMPLES 次，n_classes]* ）-的预测值。
- **X_leaves**（*shape = array = [n_samples，n_trees]或 shape [n_samples，n_trees * n_classes]*） - 如果 `pred_leaf=True`，每个样本的每个树的预测叶。
- **X_SHAP_values**（*shape = array = [n_samples，n_features + 1]或 shape [n_samples，（n_features + 1）* n_classes]*） - 如果 `pred_contrib=True`，每

个样本的每个特征贡献。

set_params（** params ）

设置此估算器的参数。

参数： ** params - 带有新值的参数名称。

返回： **自我** - 回归自我。

返回类型： 宾语

---

class**lightgbm.LGBMRanker**（boosting_type ='gbdt'，num_leaves = 31，max_depth = -1，learning_rate = 0.1，n_estimators = 100，subsample_for_bin = 200000，objective = None，class_weight = None，min_split_gain = 0.0，min_child_weight = 0.001，min_child_samples = 20，subsample = 1.0，subsample_freq = 0，colsample_bytree = 1.0，reg_alpha = 0.0，reg_lambda = 0.0，random_state = None，n_jobs = -1，silent = True，importance_type ='split'，** kwargs ）[来源]

基地： **lightgbm.sklearn.LGBMModel**

LightGBM 排名。

构建梯度增强模型。

参数：

- **boosting_type**（*string，optional（default ='gbdt'）*） - 'gbdt'，传统的 Gradient Boosting 决策树。'dart'，Dropouts 遇到多个加法回归树。'高斯'，基于渐变的单面采样。'rf'，随机森林。
- **num_leaves**（*int，optional（default = 31）*） - 基础学习者的最大树叶。
- **max_depth**（*int，optional（default = -1）*） - 基础学习者的最大树深度，-1 表示没有限制。
- **learning_rate**（*float，optional（默认值= 0.1）*） - 提高学习率。您可以使用方法的 `callbacks` 参数 `fit` 来缩小/调整使用 `reset_parameter` 回调的训练中的学习率。请注意，这将忽略 `learning_rate` 训练中的参数。
- **n_estimators**（*int，optional（default = 100）*） - 要适应的提升树数。
- **subsample_for_bin**（*int，optional（default = 200000）*） - 构造箱的样本数。
- **objective**（*字符串，可调用或无，可选（默认=无）*） - 指定学习任务和相应的学习目标或要使用的自定义目标函数（请参阅下面的注释）。默认值：LGBMRegressor 的'regression'，LGBMClassifier 的'binary'或'multiclass'，LGBMRanker 的'lambdarank'。
- **class_weight**（*dict，'balanced' 或 None，可选（默认=无）*） - 与表单中的类关联的权重。仅将此参数用于多类分类任务；对于二进制分类任务，您可以使

用或参数。“平衡”模式使用 y 的值自动调整与输入数据中的类频率成反比的权重。如果为 None，则所有类都应该具有权重 1。注意，如果指定，这些权重将乘以（通过该方法）。

```
{class_label: weight}is_unbalancescale_pos_weightn_samples / (n_classes * np.bincount(y))sample_weightfitsample_weight
```

- **min_split_gain**（*float，optional （default = 0。）*） - 在树的叶节点上进行进一步分区所需的最小损耗减少。
- **min_child_weight**（*float，optional （default = 1e-3 ）*） - 子（叶）中所需的实例权重（粗体）的最小总和。
- **min_child_samples**（*int，optional （default = 20 ）*） - 子**节点**（叶子）中所需的最小数据量。
- **subsample**（*float，optional （default = 1。）*） - 训练实例的子采样率。
- **subsample_freq**（*int，optional （default = 0 ）*） - 子样本的频率，<= 0 表示不启用。
- **colsample_bytree**（*float，optional （default = 1。）*） - 构造每个树时列的子采样率。
- **reg_alpha**（*float，optional （default = 0。）*） - 权重上的 L1 正则化项。
- **reg_lambda**（*float，optional （default = 0。）*） - 权重的 L2 正则项。
- **random_state**（*int 或 None，可选（默认=无）*） - 随机数种子。如果为 None，将使用 C ++代码中的默认种子。
- **n_jobs**（*int，optional （default = -1 ）*） - 并行线程数。
- **silent**（*bool，optional （default = True ）*） - 是否在运行 boost 时打印消息。
- **importance_type**（*string，optional （default ='split' ）*） - 要填充的要素重要性的类型 `feature_importances_`。如果是“拆分”，则结果包含在模型中使用该要素的次数。如果为'gain'，则结果包含使用该功能的分割的总增益。
- ** **kwargs** -

模型的其他参数。查看 http://lightgbm.readthedocs.io/en/latest/Parameters.html 以获取更多参数。

-

注意

-

sklearn 不支持** kwargs，可能会导致意外问题。

-

```
n_features_
```

int - 拟合模型的特征数。

classes_

shape of shape = [n_classes] - 类标签数组（仅用于分类问题）。

n_classes_

int - 类的数量（仅用于分类问题）。

best_score_

dict 或 None - 拟合模型的最佳分数。

best_iteration_

int 或 None - `early_stopping_rounds` 已指定拟合模型的最佳迭代。

objective_

string 或 callable - 适合此模型时使用的具体目标。

booster_

助推器 - 这种模式的潜在助推器。

evals_result_

dict 或 None - 如果 `early_stopping_rounds` 已指定评估结果。

feature_importances_

shape of array = [n_features] - 要素重要性（越高，功能越重要）。

注意

可以为 `objective` 参数提供自定义目标函数。在这种情况下，它应该有签名 或 ：`objective(y_true, y_pred) -> grad, hessobjective(y_true, y_pred, group) -> grad, hess`

> y_true : shape-like = [n_samples]
>
> 目标值。
>
> y_pred : shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> group : 类似于 array
>
> 组/查询数据，用于排名任务。

grad ： 类似于 shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）

每个采样点的梯度值。

hess ： shape-like of shape = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）

每个样本点的二阶导数的值。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 节获得第 i 行 y_pred，那么访问方式是 y_pred [j * num_data + i]，你也应该以这种方式对 grad 和 hess 进行分组。

best_iteration_

获得拟合模型的最佳迭代。

best_score_

获得合适模型的最佳分数。

booster_

获得此模型的底层 lightgbm Booster。

evals_result_

获得评估结果。

feature_importances_

获取功能重要性。

注意

用于标准化为 1 的 sklearn 接口中的特征重要性，它在 2.0.4 之后被弃用，现在与 Booster.feature_importance（）相同。 `importance_type` 属性被传递给函数以配置要提取的重要性值的类型。

fit（X，y，sample_weight =无，init_score =无，组=无，eval_set =无，eval_names =无，eval_sample_weight =无，eval_init_score =无，eval_group =无，eval_metric =无，eval_at = [1]，early_stopping_rounds =无，verbose = True，feature_name ='auto'，categorical_feature ='auto'，callbacks = None ）[来源]

从训练集（X，y）构建梯度增强模型。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入要素矩阵。
- **y**（*array-like of shape = [n_samples]*） - 目标值（分类中的类标签，回归中的实数）。
- **sample_weight**（*shape = array = [n_samples]或 None ，可选（默认=*

无）） - 训练数据的权重。

- **init_score**（*shape = array = [n_samples]或 None ， 可选（默认=无）*） - 训练数据的初始分数。
- **group**（*array-like 或 None ， optional （default = None ）*） - 训练数据的组数据。
- **eval_set**（*list 或 None ， optional （default = None ）*） - 用作验证集的（X，y）元组对的列表。
- **eval_names**（*字符串列表或 None ， 可选（默认=无）*） - eval_set 的名称。
- **eval_sample_weight**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的权重。
- **eval_init_score**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的初始分数。
- **eval_group**（*数组列表或 None ， 可选（默认=无）*） - eval 数据的组数据。
- **eval_metric**（*字符串，字符串列表，可调用或无，可选（默认=无）*） - 如果是字符串，它应该是要使用的内置评估指标。如果是可调用的，则应该是自定义评估指标，有关详细信息，请参阅下面的注释。在任何一种情况下，`metric` 都将评估和使用模型参数。默认值：LGBMRegressor 为'l2'，LGBMClassifier 为'logloss'，LGBMRanker 为'ndcg'。
- **eval_at**（*int 列表，可选（默认= [1]*）） - 指定度量的评估位置。
- **early_stopping_rounds**（*int 或 None ， 可选（默认=无）*） - 激活提前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。
- **verbose**（*bool 或 int ， 可选（默认= True ）*） -

至少需要一个评估数据。如果为 True，则在每个提升阶段打印 eval 集上的 eval 度量。如果为 int，则在每个 `verbose` 提升阶段打印 eval 集上的 eval 度量。`early_stopping_rounds` 还打印了通过使用找到的最后一个增强阶段或增强阶段。

-

例

-

在 `verbose`= 4 且至少有一个项目的情况下 `eval_set`，每 4 个（而不

是 1 个）增强阶段打印评估度量。

-
- **feature_name**（*字符串列表或'auto'，可选（默认='auto'）*） - 功能名称。如果'auto'和数据是 pandas DataFrame，则使用数据列名称。
- **categorical_feature**（*字符串列表或 int，或'auto'，可选（默认='auto'）*） - 分类功能。如果是 int 列表，则解释为索引。如果是字符串列表，则解释为要素名称（也需要指定 `feature_name`）。如果'auto'和数据是 pandas DataFrame，则使用 pandas 分类列。分类特征中的所有值都应小于 int32 最大值（2147483647）。大值可能是内存消耗。考虑使用从零开始的连续整数。分类要素中的所有负值都将被视为缺失值。
- **回调**（*回调函数列表或无，可选（默认=无）*） - 每次迭代时应用的回调函数列表。有关更多信息，请参阅 Python API 中的回调。

返回： **自我** - 回归自我。

返回类型： 宾语

注意

自定义 eval 函数需要具有以下签名的可调用： ，或者 返回（eval_name，eval_result，is_bigger_better）或（eval_name，eval_result，is_bigger_better）列表：

```
func(y_true, y_pred)func(y_true, y_pred, weight)func(y_true, y_pred, weight, group)
```

> y_true ：shape-like = [n_samples]
>
> 目标值。
>
> y_pred ：shape = array = [n_samples]或 shape = [n_samples * n_classes]（用于多类任务）
>
> 预测值。
>
> 重量：形状的数组= [n_samples]
>
> 样品的重量。
>
> group ：类似于 array
>
> 组/查询数据，用于排名任务。
>
> eval_name ：string
>
> 评价的名称。

eval_result ：浮动

评估结果。

is_bigger_better ：bool

eval 结果更好，例如 AUC 更大更好。

对于多类任务，y_pred 首先按 class_id 分组，然后按 row_id 分组。如果你想在第 j 个类中获得第 i 行 y_pred，则访问方式是 y_pred [j * num_data + i]。

get_params（深=真）

获取此估算工具的参数。

参数： deep（*bool ，optional （default = True ）*） - 如果为 True，将返回此估计器的参数并包含作为估算器的子对象。

返回： params - 映射到其值的参数名称。

返回类型： 字典

n_features_

获取适合型号的功能数量。

objective_

获得适合此模型时使用的具体目标。

predict（X，raw_score = False，num_iteration = None，pred_leaf = False，pred_contrib = False，** kwargs ）

返回每个样本的预测值。

参数：

- **X**（*形状的数组或稀疏矩阵= [n_samples ，n_features]*） - 输入*要素*矩阵。
- **raw_score**（*bool ， optional （default = False ）*） - 是否预测原始分数。
- **num_iteration**（*int 或 None ， optional （default = None ）*） - 限制预测中的迭代次数。如果为 None，如果存在最佳迭代，则使用它; 否则，使用所有树木。如果<= 0，则使用所有树（无限制）。
- **pred_leaf**（*bool ， optional （default = False ）*） - 是否预测叶子索引。
- **pred_contrib**（*bool ， optional （default = False ）*） -

是否预测功能贡献。

-

注意

-

如果您想使用 SHAP 交互值等 SHAP 值获得有关模型预测的更多解释，可以安装 shap 包（https://github.com/slundberg/shap）。

-
- ** **kwargs** - 预测的其他参数。

返回：

- **predicted_result**（*阵列状的= [N_SAMPLES 次]或形状= [N_SAMPLES 次，n_classes]* ）-的预测值。
- **X_leaves**（*shape = array = [n_samples，n_trees]或 shape [n_samples，n_trees * n_classes]*） - 如果 `pred_leaf=True`，每个样本的每个树的预测叶。
- **X_SHAP_values**（*shape = array = [n_samples， n_features + 1]或 shape [n_samples， （n_features + 1）* n_classes]*） - 如果 `pred_contrib=True`，每个样本的每个特征贡献。

set_params（** params ）

设置此估算器的参数。

参数： ** params - 带有新值的参数名称。

返回： **自我** - 回归自我。

返回类型： 宾语

回调

lightgbm.early_stopping（stops_rounds，verbose = True ）[来源]

创建一个激活提前停止的回调。

注意

激活提前停止。该模型将进行训练，直到验证分数停止改善。验证分数至少需要在每 `early_stopping_rounds` 一轮都有所改进才能继续培训。至少需要一个验证数据和一个指标。如果有多个，将检查所有这些。但是无论如何都会忽略训练数据。

参数：

- **stopping_rounds**（*int*） - 没有趋势发生的可能轮次数。
- **verbose**（*bool，optional （默认= True ）*） - 是否使用提前停止信息打印消息。

返回： **callback** - 激活提前停止的回调。

返回类型： 功能

lightgbm.print_evaluation（句号= 1，show_stdv = True ）[来源]

创建一个打印评估结果的回调。

参数：
- **period**（*int*，*optional*（*default = 1*））- 打印评估结果的期间。
- **show_stdv**（*bool*，*optional*（*default = True*））- 是否显示 stdv（如果提供）。

返回： **callback** - 每次 `period` 迭代打印评估结果的回调。

返回类型： 功能

lightgbm.record_evaluation（eval_result ）[来源]

创建一个记录评估历史记录的回调 `eval_result`。

参数： eval_result（*dict*）- 用于存储评估结果的字典。

返回： callback - 将评估历史记录记录到传递的字典中的回调。

返回类型： 功能

lightgbm.reset_parameter（** kwargs ）[来源]

创建一个在第一次迭代后重置参数的回调。

注意

初始参数仍将在第一次迭代时生效。

参数： ** kwargs（*值应为列表或函数*）- 每个助推轮的参数*列表或*根据当前轮数计算参数的自定义函数（例如，产生学习率衰减）。如果列出 lst，参数= lst [current_round]。如果是函数 func，则参数= func（current_round）。

返回： callback - 在第一次迭代后重置参数的回调。

返回类型： 功能

绘图

lightgbm.plot_importance（booster，ax = None，height = 0.2，xlim = None，ylim = None，title ='Feature importance'，xlabel ='Feature importance'，ylabel

='Features', importance_type ='split', max_num_features = None, ignore_zero = True, figsize = None, grid = True, precision = None, ** kwargs) [source]

绘制模型的特征重要性。

参数：

- **booster**（Booster 或 LGBMModel） - 应该绘制具有重要性的 Booster 或 LGBMModel 实例。
- **ax**（*matplotlib.axes.Axes 或 None，可选（默认=无）*） - 目标轴实例。如果为 None，则将创建新的图形和轴。
- **height**（*float，optional （默认值= 0.2 ）*） - 传递给的条形高度 `ax.barh()`。
- **xlim**（*2 个元素的元组或无，可选（默认=无）*） - 传递给元组 `ax.xlim()`。
- **ylim**（*2 个元素的元组或无，可选（默认=无）*） - 元组传递给 `ax.ylim()`。
- **title**（*字符串或无，可选（默认="功能重要性"）*） - 轴标题。如果为 None，则禁用标题。
- **xlabel**（*字符串或无，可选（默认="功能重要性"）*） - X 轴标题标签。如果为 None，则禁用标题。
- **ylabel**（*字符串或无，可选（默认="功能"）*） - Y 轴标题标签。如果为 None，则禁用标题。
- **importance_type**（*string，optional （default ="split"）*） - 如何计算重要性。如果"拆分"，则结果包含在模型中使用该要素的次数。如果"获得"，则结果包含使用该功能的分割的总增益。
- **max_num_features**（*int 或 None，可选（默认=无）*） - 绘图上显示的最大顶部**要素**数。如果为 None 或<1，则将显示所有功能。
- **ignore_zero**（*bool，optional （default = True ）*） - 是否忽略具有零重要性的**要素**。
- **figsize**（*2 个元素的元组或无，可选（默认=无）*） - 图形大小。
- **grid**（*bool，optional （default = True ）*） - 是否为轴添加网格。
- **precision**（*int 或 None，可选（默认=无）*） - 用于将浮点值的显示限制为特定精度。
- ** **kwargs** - 传递给的其他参数 `ax.barh()`。

返回：**ax** - 具有模型特征重要性的图。

返回类型：matplotlib.axes.Axes

lightgbm.plot_metric（booster，metric = None，dataset_names = None，ax = None，xlim = None，ylim = None，title =’训练时的公制’，xlabel =’迭代’，ylabel =’auto’，figsize = None，grid = True ）[来源]

在培训期间绘制一个指标。

参数：

- **booster**（*dict 或* LGBMModel） - 从 `lightgbm.train()`LGBMModel 实例返回的字典。
- **metric**（*字符串或无，可选（默认=无）*） - 要绘制的度量标准名称。仅支持一个指标，因为不同的指标具有不同的比例 如果为 None，则从字典中选择第一个度量（根据哈希码）。
- **dataset_names**（*字符串列表或 None ，可选（默认=无）*） - 用于计算要绘制的度量标准的数据集名称列表。如果为 None，则使用所有数据集。
- **ax**（*matplotlib.axes.Axes 或 None ，可选（默认=无）*） - 目标轴实例。如果为 None，则将创建新的图形和轴。
- **xlim**（*2 个元素的元组或无，可选（默认=无）*） - 传递给元组 `ax.xlim()`。
- **ylim**（*2 个元素的元组或无，可选（默认=无）*） - 元组传递给 `ax.ylim()`。
- **title**（*字符串或无，可选（默认=“训练期间的度量”）*） - 轴标题。如果为 None，则禁用标题。
- **xlabel**（*字符串或无，可选（默认=“迭代”）*） - X 轴标题标签。如果为 None，则禁用标题。
- **ylabel**（*字符串或无，可选（默认=“自动”）*） - Y 轴标题标签。如果是“auto”，则使用度量标准名称。如果为 None，则禁用标题。
- **figsize**（*2 个元素的元组或无，可选（默认=无）*） - 图形大小。
- **grid**（*bool ， optional （default = True ）*） - 是否为轴添加网格。

返回： **ax** - 具有度量标准的训练历史图。

返回类型： matplotlib.axes.Axes

lightgbm.plot_tree（booster， ax = None， tree_index = 0， figsize = None，old_graph_attr = None，old_node_attr = None，old_edge_attr = None，show_info = None，precision = None，** kwargs ）[source]

绘制指定的树。

注意

最好使用它，`create_tree_digraph()`因为它具有无损质量，并且返回的对象也可以直接在 Jupyter 笔记本中呈现和显示。

参数：

- **booster**（Booster *或* LGBMModel）- 要绘制的 Booster 或 LGBMModel 实例。
- **ax**（*matplotlib.axes.Axes 或 None，可选（默认=无）*） - 目标轴实例。如果为 None，则将创建新的图形和轴。
- **tree_index**（*int，optional （default = 0 ）*） - 要绘制的目标树的索引。
- **figsize**（*2 个元素的元组或无，可选（默认=无）*） - 图形大小。
- **show_info**（*字符串列表或无，可选（默认=无）*） - 应在节点中显示哪些信息。列表项的可能值：'split_gain'，'internal_value'，'internal_count'，'leaf_count'。
- **precision**（*int 或 None，可选（默认=无）*） - 用于将浮点值的显示限制为特定精度。
- ** **kwargs** - 传递给 `Digraph` 构造函数的其他参数。有关支持的参数的完整列表，请访问 https://graphviz.readthedocs.io/en/stable/api.html#digraph。

返回： **ax** - 单树的情节。

返回类型： matplotlib.axes.Axes

lightgbm.create_tree_digraph（booster，tree_index = 0，show_info = None，precision = None，old_name = None，old_comment = None，old_filename = None，old_directory = None，old_format = None，old_engine = None，old_encoding = None，old_graph_attr = None，old_node_attr = None，old_edge_attr =无，old_body =无，old_strict = False，** kwargs ）[来源]

创建指定树的有向图表示。

注意

有关更多信息，请访问 https://graphviz.readthedocs.io/en/stable/api.html#digraph。

参数：

- **booster**（Booster *或* LGBMModel）- 要转换的 Booster 或 LGBMModel 实例。
- **tree_index**（*int，optional （default = 0 ）*） - 要转换的目标树的索引。
- **show_info**（*字符串列表或无，可选（默认=无）*） - 应在节点中显示哪些信息。列表项的可能值：'split_gain'，'internal_value'，

'internal_count'，'leaf_count'。

- **precision**（*int 或 None，可选（默认=无）*） - 用于将浮点值的显示限制为特定精度。
- ** **kwargs** - 传递给 `Digraph` 构造函数的其他参数。有关支持的参数的完整列表，请访问 https://graphviz.readthedocs.io/en/stable/api.html#digraph。

返回： **graph** - 指定树的有向图表示。

返回类型： graphviz.Digraph

参数调整

此页面包含针对不同方案的参数调整指南。

其他有用链接列表

- 参数
- Python API

叶子（最佳优先）树的调整参数

LightGBM 使用叶子树生长算法，而许多其他流行的工具使用深度树生长。与深度增长相比，叶子算法可以更快地收敛。然而，如果不与适当的参数一起使用，叶片生长可能过度拟合。

为了使用叶子树获得良好的结果，这些是一些重要的参数：

1. `num_leaves`。这是控制树模型复杂性的主要参数。从理论上讲，我们可以设置获得与深度树相同数量的叶子。但是，这种简单的转换在实践中并不好。原因在于，对于固定数量的叶子，叶子树通常比深度方式树深得多。无约束的深度会导致过度拟合。因此，当试图调整时，我们应该让它小于。例如，当深度方向树可以获得良好的准确度，但设置为可能导致过度拟合，并将其设置为或者可能比深度方式获得更好的准确性。

   `num_leaves = 2^(max_depth)num_leaves2^(max_depth)max_depth=7num_leaves1277080`
2. `min_data_in_leaf`。这是防止叶子树过度拟合的一个非常重要的参数。其最佳值取决于训练样本的数量和 `num_leaves`。将其设置为较大的值可以避免树长

得太深，但可能会导致不合适。实际上，将其设置为数百或数千对于大型数据集就足够了。

3. `max_depth`。您还可以使用 `max_depth` 显式限制树深度。

更快的速度

- 通过设置 `bagging_fraction` 和使用装袋 `bagging_freq`
- 通过设置使用要素子采样 `feature_fraction`
- 使用小 `max_bin`
- 使用 `save_binary` 加快今后的学习数据加载
- 使用并行学习，请参阅并行学习指南

为了更好的准确性

- 使用大 `max_bin`（可能会慢）
- 使用小 `learning_rate` 而大 `num_iterations`
- 使用大 `num_leaves`（可能导致过度配合）
- 使用更大的训练数据
- 尝试 `dart`

处理过度拟合

- 使用小 `max_bin`
- 使用小 `num_leaves`
- 使用 `min_data_in_leaf` 和 `min_sum_hessian_in_leaf`
- 使用套装 `bagging_fraction` 和套装 `bagging_freq`
- 按集使用特征子采样 `feature_fraction`
- 使用更大的训练数据
- 尝试 `lambda_l1`， `lambda_l2` 并 `min_gain_to_split` 进行正规化
- 尽量 `max_depth` 避免长树生长

下一个　以前

参数

此页面包含 LightGBM 中所有参数的说明。

其他有用链接列表

- Python API
- 参数调整

外部链接

- Laurae ++交互式文档

参数格式

参数格式是。可以在配置文件和命令行中设置参数。通过使用命令行，参数之前和之后不应有空格。通过使用配置文件，一行只能包含一个参数。您可以用来评论。`key1=value1 key2=value2 ...=#`

如果命令行和配置文件中都出现一个参数，LightGBM 将使用命令行中的参数。

核心参数

- `config`，default =`""`，type = string，aliases：`config_file`
- 配置文件的路径
- **注意**：只能在 CLI 版本中使用

- `task`，默认=`train`，类型=枚举，选项：`train`，`predict`，`convert_model`，`refit`，别名：`task_type`
- `train`，用于训练，别名：`training`
- `predict`，为预测，别名：`prediction`，`test`
- `convert_model`，要将模型文件转换为 if-else 格式，请参阅 IO 参数中的更多信息
- `refit`，用于使用新数据重新定义现有模型，别名：`refit_tree`
- **注意**：只能在 CLI 版本中使用; 对于特定于语言的包，您可以使用相应的功能

- `objective`，默认=`regression`，类型=枚举，选项：`regression`，`regression_l1`，`huber`，`fair`，`poisson`，`quantile`，`mape`，`gammma`，`tweedie`，`binary`，`multiclass`，`multiclassova`，`xentropy`，`xentlambda`，`lambdarank`，别名：`objective_type`，`app`，`application`
- 回归应用

- regression_l2，L2 损失，别名：regression，mean_squared_error，mse，l2_root，root_mean_squared_error，rmse
- regression_l1，L1 损失，别名：mean_absolute_error，mae
- huber，胡贝尔损失
- fair，公平的损失
- poisson，泊松回归
- quantile，分位数回归
- mape，MAPE 丢失，别名：mean_absolute_percentage_error
- gamma，Gamma 回归与日志链接。它可能是有用的，例如，用于建模保险索赔严重性，或任何可能是伽玛分布的目标
- tweedie，带有日志链接的 Tweedie 回归。它可能是有用的，例如，用于对保险中的总损失进行建模，或者对于可能是 tweedie-distributed 的任何目标进行建模

- binary，二元对数损失分类（或逻辑回归）。需要{0,1}中的标签; 参见 cross-entropy[0,1]中一般概率标签的申请
- 多类分类应用

- multiclass，softmax 目标函数，别名：softmax
- multiclassova，一 VS-所有二进制目标函数，别名：multiclass_ova，ova，ovr
- num_class 也应该设置

- 交叉熵应用
- xentropy，交叉熵的目标函数（可选线性权重），别名：cross_entropy
- xentlambda，交叉熵的替代参数化，别名：cross_entropy_lambda
- 标签是区间[0,1]中的任何内容
- lambdarank，lambdarank 申请

- label 应该 int 输入 lambdarank 任务，而更大的数字代表更高的相关性（例如 0：坏，1：公平，2：好，3：完美）
- label_gain 可用于设置 int 标签的增益（重量）
- 所有值 label 必须小于 in 中的元素数 label_gain

- boosting，默认=gbdt，类型=枚举，选项：gbdt，gbrt，rf，random_forest，dart，goss，别名：boosting_type，boost

- gbdt，传统的 Gradient Boosting 决策树，别名：gbrt
- rf，随机森林，别名：random_forest

- dart，Dropouts 符合多个加法回归树
- goss，基于梯度的单面采样

- data，默认=""，类型=串，别名：train，train_data，train_data_file，data_filename

- 训练数据的路径，LightGBM 将从这些数据进行训练
- **注意**：只能在 CLI 版本中使用

- valid，默认=""，类型=串，别名：test，valid_data，valid_data_file，test_data，test_data_file，valid_filenames

- 验证/测试数据的路径，LightGBM 将输出这些数据的指标
- 支持多个验证数据，由...分隔，
- **注意**：只能在 CLI 版本中使用

- num_iterations，默认=100，类型= INT，别名：num_iteration，n_iter，num_tree，num_trees，num_round，num_rounds，num_boost_round，n_estimators，约束：num_iterations >= 0

- 增强迭代次数
- **注意**：在内部，LightGBM 构造树以解决多类分类问题 num_class * num_iterations

- learning_rate，默认=0.1，类型=双，别名：shrinkage_rate，eta，约束：learning_rate > 0.0

- 收缩率
- 在 dart，它还影响掉落树木的标准化权重

- num_leaves，默认=31，类型= INT，别名：num_leaf，max_leaves，max_leaf，约束：num_leaves > 1

- 一棵树中叶子的最大数量

- tree_learner，默认=serial，类型=枚举，选项：serial，feature，data，voting，别名：tree，tree_type，tree_learner_type

- serial，单机树学习者
- feature，功能并行树学习器，别名：feature_parallel
- data，数据并行树学习器，别名：data_parallel

- `voting` 投票并行树学习器，别名： `voting_parallel`
- 请参阅并行学习指南以获取更多详细信息

- `num_threads` ，默认=`0`，类型= INT，别名：`num_thread`，`nthread`，`nthreads`，`n_jobs`

- LightGBM 的线程数
- `0` 表示 OpenMP 中的默认线程数
- 为了获得最佳速度，请将其设置为**实际 CPU 核心数**，而不是线程数（大多数 CPU 使用超线程为每个 CPU 核心生成 2 个线程）
- 如果数据集很小，请不要将其设置得太大（例如，对于 10,000 行的数据集，不要使用 64 个线程）
- 请注意，任务管理器或任何类似的 CPU 监视工具可能会报告核心未被充分利用。**这个是正常的**
- 对于并行学习，请勿使用所有 CPU 内核，因为这会导致网络通信性能下降

- `device_type` ，默认=`cpu`，类型=枚举，选项：`cpu`，`gpu`，别名：`device`

- 用于树学习的设备，可以使用 GPU 实现更快的学习
- **注意**：建议使用较小的 `max_bin`（例如 63）以获得更好的加速
- **注意**：对于更快的速度，GPU 默认使用 32 位浮点进行求和，因此这可能会影响某些任务的准确性。您可以设置 `gpu_use_dp=true` 为启用 64 位浮点数，但它会减慢训练速度
- **注意**：请参阅安装指南以构建具有 GPU 支持的 LightGBM

- `seed` ，default =`0`，type = int，aliases :`random_seed`,`random_state`

- 该种子用于生成其他种子，例如 `data_random_seed` ，`feature_fraction_seed`
- 如果你设置其他种子，将被覆盖

学习控制参数

- `max_depth` ，default =`-1`，type = int

- 限制树模型的最大深度。这用于处理过小时的过度拟合`#data`。树仍然生长叶子
- `< 0` 意味着没有限制

- `min_data_in_leaf`，默认=`20`，类型= INT，别名：`min_data_per_leaf`，`min_data`，`min_child_samples`，约束：`min_data_in_leaf >= 0`
- 一片叶子中的数据量最小。可以用来处理过度拟合
- `min_sum_hessian_in_leaf`，默认=`1e-3`，类型=双，别名：`min_sum_hessian_per_leaf`，`min_sum_hessian`，`min_hessian`，`min_child_weight`，约束：`min_sum_hessian_in_leaf >= 0.0`
- 一片叶子中的最小和粗麻布。比如`min_data_in_leaf`，它可以用来处理过度拟合
- `bagging_fraction`，默认=`1.0`，类型=双，别名：`sub_row`，`subsample`，`bagging`，约束：`0.0 < bagging_fraction <= 1.0`
- 喜欢`feature_fraction`，但这将随机选择部分数据而无需重新采样
- 可以用来加速训练
- 可以用来处理过度拟合
- **注意**：要启用装袋，也`bagging_freq`应将其设置为非零值
- `bagging_freq`，default =`0`，type = int，aliases：`subsample_freq`
- 装袋频率
- `0`意味着禁用装袋; `k`意味着在每次`k`迭代时执行装袋
- **注意**：启用装袋，`bagging_fraction`应设置为价值小于`1.0`以及
- `bagging_seed`，default =`3`，type = int，aliases：`bagging_fraction_seed`
- 套袋随机种子
- `feature_fraction`，默认=`1.0`，类型=双，别名：`sub_feature`，`colsample_bytree`，约束：`0.0 < feature_fraction <= 1.0`
- 如果`feature_fraction`小于，则 LightGBM 将在每次迭代时随机选择部分特征`1.0`。例如，如果将其设置为`0.8`，LightGBM 将在训练每棵树之前选择 80％的功能
- 可以用来加速训练
- 可以用来处理过度拟合
- `feature_fraction_seed`，default =`2`，type = int

- 随机种子 feature_fraction
- early_stopping_round ， default =0 ， type = int ， aliases :early_stopping_rounds,early_stopping
- 如果一个验证数据的一个度量标准在上 early_stopping_round 一轮中没有改善，则将停止训练
- <= 0 意味着禁用
- max_delta_step ，默认=0.0，类型=双，别名：max_tree_output，max_leaf_output
- 用于限制树叶的最大输出
- <= 0 意味着没有约束
- 叶子的最终最大输出是 learning_rate * max_delta_step
- lambda_l1 ，default =0.0，type = double，aliases：reg_alpha，constraints：lambda_l1 >= 0.0
- L1 正则化
- lambda_l2 ， 默认=0.0， 类型=双， 别名： reg_lambda， lambda， 约束： lambda_l2 >= 0.0
- L2 正规化
- min_gain_to_split ， default =0.0，type = double，aliases：min_split_gain，constraints：min_gain_to_split >= 0.0
- 执行拆分的最小增益
- drop_rate ，default =0.1，type = double，aliases：rate_drop，constraints：0.0 <= drop_rate <= 1.0
- 仅用于 dart
- 辍学率：辍学期间先前树木的一小部分
- max_drop ， default =50，type = int
- 仅用于 dart
- 在一次增强迭代期间丢弃的最大树数
- <=0 意味着没有限制

- skip_drop，默认=0.5，类型=双，约束：0.0 <= skip_drop <= 1.0
  - 仅用于 dart
  - 在增强迭代期间跳过退出程序的概率
- xgboost_dart_mode，默认=false，类型= bool
  - 仅用于 dart
  - true 如果要使用 xgboost dart 模式，请将此设置为
- uniform_drop，默认=false，类型= bool
  - 仅用于 dart
  - 将此设置为 true，如果要使用均匀丢弃
- drop_seed，default =4，type = int
  - 仅用于 dart
  - 随机种子选择下降模型
- top_rate，默认=0.2，类型=双，约束：0.0 <= top_rate <= 1.0
  - 仅用于 goss
  - 大梯度数据的保留率
- other_rate，默认=0.1，类型=双，约束：0.0 <= other_rate <= 1.0
  - 仅用于 goss
  - 小梯度数据的保留率
- min_data_per_group，default =100，type = int，constraints：min_data_per_group > 0
  - 每个分类组的最小数据量
- max_cat_threshold，default =32，type = int，constraints：max_cat_threshold > 0
  - 用于分类功能
  - 限制分类特征中的最大阈值点
- cat_l2，默认=10.0，类型=双，约束：cat_l2 >= 0.0

- 用于分类功能
- 分类分裂中的 L2 正则化

- `cat_smooth`，默认=`10.0`，类型=双，约束：`cat_smooth >= 0.0`

- 用于分类功能
- 这可以减少分类特征中噪声的影响，特别是对于数据很少的类别

- `max_cat_to_onehot`，default =`4`，type = int，constraints：`max_cat_to_onehot > 0`

- 当一个特征的类别数小于或等于时 `max_cat_to_onehot`，将使用一对另一分割算法

- `top_k`，default =`20`，type = int，aliases :`topk`, constraints：`top_k > 0`

- 用于投票平行
- 将此值设置为较大的值可获得更准确的结果，但会降低训练速度

- `monotone_constraints`，default =`None`，type = multi-int，aliases :`mc`,`monotone_constraint`

- 用于限制单调特征
- `1` 意味着增加，`-1` 意味着减少，`0` 意味着非约束
- 您需要按顺序指定所有功能。例如，`mc=-1,0,1` 表示第 1 个特征减少，第 2 个特征不受约束，第 3 个特征增加

- `feature_contri`，默认=`None`，键入=多双，别名：`feature_contrib`，`fc`，`fp`，`feature_penalty`

- 用于控制特征的分割增益，将用于替换第 i 个特征的分割增益 `gain[i] = max(0, feature_contri[i]) * gain[i]`
- 您需要按顺序指定所有功能

- `forcedsplits_filename`，默认=`""`，类型=串，别名：`fs`，`forced_splits_filename`，`forced_splits_file`，`forced_splits`

- `.json` 文件的路径，指定在最佳第一次学习开始之前在每个决策树的顶部强制拆分
- `.json` 文件可以任意嵌套的，并且每个分割包括 `feature`，`threshold` 字段，以及 `left` 与 `right` 表示 subsplits 字段

- 分类拆分以一种热门方式强制执行，`left` 表示包含特征值并 `right` 表示其他值的拆分
- **注意**：如果拆分使收益变差，则强制拆分逻辑将被忽略
- 看到这个文件作为一个例子

- `refit_decay_rate`，默认=`0.9`，类型=双，约束：`0.0 <= refit_decay_rate <= 1.0`

- `refit` 任务的衰减率，将用于改装树木 `leaf_output = refit_decay_rate * old_leaf_output + (1.0 - refit_decay_rate) * new_leaf_output`
- 仅用于 `refit`CLI 版本的任务或作为 `refit` 特定于语言的包中的函数的参数

IO 参数

- `verbosity`，default =`1`，type = int，aliases：`verbose`

- 控制 LightGBM 的详细程度
- `< 0`：致命，：错误（警告），：信息，：调试`= 0= 1> 1`

- `max_bin`，default =`255`，type = int，constraints：`max_bin > 1`

- 特征值的最大二进制数将被打包
- 少量的垃圾箱可能会降低训练精度，但可能会增加一般功率（处理过度配合）
- LightGBM 将根据自动压缩内存 `max_bin`。例如，LightGBM 将 `uint8_t` 用于特征值 if`max_bin=255`

- `min_data_in_bin`，default =`3`，type = int，constraints：`min_data_in_bin > 0`

- 一个 bin 中的最小数据量
- 使用它来避免单数据一个 bin（潜在的过度拟合）

- `bin_construct_sample_cnt`，default =`200000`，type = int，aliases :`subsample_for_bin`, constraints：`bin_construct_sample_cnt > 0`

- 采样以构建直方图箱的数据数量
- 将此值设置为较大值将提供更好的训练结果，但会增加数据加载时间
- 如果数据非常稀疏，则将此值设置为较大值

- histogram_pool_size，默认=-1.0，类型=双，别名：hist_pool_size
- 历史直方图的最大缓存大小（MB）
- < 0 意味着没有限制
- data_random_seed，default =1，type = int，aliases：data_seed
- 并行学习中的数据分区的随机种子（不包括 feature_parallel 模式）
- output_model，default =LightGBM_model.txt，type = string，aliases :model_output,model_out
- 训练中输出模型的文件名
- **注意**：只能在 CLI 版本中使用
- snapshot_freq，default =-1，type = int，aliases：save_period
- 保存模型文件快照的频率
- 将此值设置为正值以启用此功能。例如，模型文件将在每次迭代时进行快照 snapshot_freq=1
- **注意**：只能在 CLI 版本中使用
- input_model，default ="""，type = string，aliases :model_input,model_in
- 输入模型的文件名
- 对于 prediction 任务，该模型将应用于预测数据
- 对于 train 任务，将继续从该模型进行培训
- **注意**：只能在 CLI 版本中使用
- output_result，默认=LightGBM_predict_result.txt，类型=串，别名：predict_result，prediction_result，predict_name，prediction_name，pred_name，name_pred
- prediction 任务中预测结果的文件名
- **注意**：只能在 CLI 版本中使用
- initscore_filename，默认=""，类型=串，别名：init_score_filename，init_score_file，init_score，input_init_score
- 训练初始分数的文件路径
- 如果""，将使用 train_data_file+ .init（如果存在）

- **注意**：只能在 CLI 版本中使用

- `valid_data_initscores`，默认=""，类型=串，别名：`valid_data_init_scores`，`valid_init_score_file`，`valid_init_score`

- 具有验证初始分数的文件路径
- 如果""，将使用 `valid_data_file` + `.init`（如果存在）
- 单独,用于多验证数据
- **注意**：只能在 CLI 版本中使用

- `pre_partition`，默认=`false`，类型= bool，别名：`is_pre_partition`

- 用于并行学习（不包括 `feature_parallel` 模式）
- `true` 如果训练数据是预分区的，并且不同的机器使用不同的分区

- `enable_bundle`，默认=`true`，类型= bool，别名：`is_enable_bundle`，`bundle`

- 将此设置 `false` 为禁用独占功能捆绑（EFB），LightGBM 中描述了这一功能：一个高效的梯度提升决策树
- **注意**：禁用此选项可能会导致稀疏数据集的训练速度变慢

- `max_conflict_rate`，默认=`0.0`，类型=双，约束：`0.0 <= max_conflict_rate < 1.0`

- EFB 中捆绑包的最大冲突率
- 将此设置 `0.0` 为禁止冲突并提供更准确的结果
- 将此值设置为更大的值以实现更快的速度

- `is_enable_sparse`，默认=`true`，类型=布尔，别名：`is_sparse`，`enable_sparse`，`sparse`

- 用于启用/禁用稀疏优化

- `sparse_threshold`，默认=`0.8`，类型=双，约束：`0.0 < sparse_threshold <= 1.0`

- 将要素视为稀疏要素的零要素百分比阈值

- `use_missing`，默认=`true`，类型= bool

- 将此设置 `false` 为禁用缺失值的特殊句柄

- `zero_as_missing`，默认=`false`，类型= bool

- 将此设置 `true` 为将所有零视为缺失值（包括 libsvm /稀疏矩阵中未显示的值）
- 将其设置为 `false` 使用 `na` 为代表缺失值

- `two_round`，默认=`false`，类型= bool，别名：`two_round_loading`，`use_two_round_loading`

- `true` 如果数据文件太大而无法放入内存，请将此设置为
- 默认情况下，LightGBM 会将数据文件映射到内存并从内存加载功能。这将提供更快的数据加载速度，但是当数据文件非常大时可能导致内存不足错误

- `save_binary`，默认=`false`，类型= bool，别名：`is_save_binary`，`is_save_binary_file`

- 如果 `true`，LightGBM 将数据集（包括验证数据）保存到二进制文件。这样可以加快下次的数据加载速度

- `enable_load_from_binary_file`，默认=`true`，类型= 布尔，别名：`load_from_binary_file`，`binary_load`，`load_binary`

- 将此设置 `true` 为启用以前保存的二进制数据集的自动加载
- 将其设置 `false` 为忽略二进制数据集

- `header`，默认=`false`，类型= bool，别名：`has_header`

- `true` 如果输入数据有标题，则将其设置为

- `label_column`，default =`""`，type = int 或 string，aliases：`label`

- 用于指定标签列
- 使用索引编号，例如 `label=0` 表示 column_0 是标签
- `name:`为列名添加前缀，例如 `label=name:is_click`

- `weight_column`，default =`""`，type = int 或 string，aliases：`weight`

- 用于指定重量列
- 使用索引编号，例如 `weight=0` 表示 column_0 是权重
- `name:`为列名添加前缀，例如 `weight=name:weight`
- **注意**：索引从中开始，`0` 并且在传递类型时不计算标签列 `int`，例如，当 label 为 column_0 且 weight 为 column_1 时，正确的参数为 `weight=0`

- `group_column`，默认=""，类型= int 或字符串，别名：`group`，`group_id`，`query_column`，`query`，`query_id`

- 用于指定查询/组 ID 列
- 使用索引编号，例如 `query=0` 表示 column_0 是查询 ID
- `name:`为列名添加前缀，例如 `query=name:query_id`
- **注意**：数据应按 query_id 分组
- **注意**：索引从中开始，`0` 并且在传递类型时不计算标签列 `int`，例如，当 label 为 column_0 且 query_id 为 column_1 时，正确的参数为 `query=0`

- `ignore_column`，default =""，type = multi-int 或 string，aliases :`ignore_feature`,`blacklist`

- 用于在训练中指定一些忽略列
- 使用索引编号，例如 `ignore_column=0,1,2` 表示 column_0，column_1 和 column_2 将被忽略
- `name:`为列名添加前缀，例如 `ignore_column=name:c1,c2,c3`，将忽略 c1，c2 和 c3
- **注意**：仅在直接从文件加载数据时有效
- **注意**：索引从中开始，`0` 并且在传递类型时不计算标签列 `int`

- `categorical_feature`，默认=""，键入=多 int 或字符串，别名：`cat_feature`，`categorical_column`，`cat_column`

- 用于指定分类功能
- 使用索引编号，例如 `categorical_feature=0,1,2` 表示 column_0，column_1 和 column_2 是分类功能
- `name:`为列名添加前缀，例如，`categorical_feature=name:c1,c2,c3` 意味着 c1，c2 和 c3 是分类特征
- **注意**：仅支持带 `int` 类型的分类
- **注意**：索引从中开始，`0` 并且在传递类型时不计算标签列 `int`
- **注意**：所有值都应小于 `Int32.MaxValue`（2147483647）
- **注意**：使用大值可能会占用内存。当分类特征由从零开始的连续整数表示时，树决策规则最有效
- **注意**：所有负值都将被视为**缺失值**

- `predict_raw_score`，默认=`false`，类型=布尔，别名：`is_predict_raw_score`，`predict_rawscore`，`raw_score`

- 仅用于 `prediction` 任务

- 将此设置 `true` 为仅预测原始分数
- 将其设置 `false` 为预测转换得分

- `predict_leaf_index`，默认=`false`，类型= bool，别名：`is_predict_leaf_index`，`leaf_index`

- 仅用于 `prediction` 任务
- 将此设置 `true` 为使用所有树的叶索引进行预测

- `predict_contrib`，默认=`false`，类型= bool，别名：`is_predict_contrib`，`contrib`

- 仅用于 `prediction` 任务
- 将其设置 `true` 为估计 SHAP 值，表示每个特征对每个预测的贡献
- 生成值，其中最后一个值是训练数据上模型输出的预期值 `#features + 1`
- **注意**：如果您想使用 SHAP 交互值等 SHAP 值获得模型预测的更多解释，可以安装 shap 包

- `num_iteration_predict`，default =`-1`，type = int

- 仅用于 `prediction` 任务
- 用于指定将在预测中使用多少经过训练的迭代
- `<= 0` 意味着没有限制

- `pred_early_stop`，默认=`false`，类型= bool

- 仅用于 `prediction` 任务
- 如果 `true`，将使用提前停止来加速预测。可能会影响准确性

- `pred_early_stop_freq`，default =`10`，type = int

- 仅用于 `prediction` 任务
- 检查早期预测的频率

- `pred_early_stop_margin`，默认=`10.0`，类型=双

- 仅用于 `prediction` 任务
- 早期预测的边际门槛

- `convert_model_language`，默认=`""`，类型=字符串

- 仅用于 convert_model 任务
- 仅 cpp 支持
- if convert_model_language 设置后 task=train，模型也将被转换
- **注意**：只能在 CLI 版本中使用

- convert_model ， default =gbdt_prediction.cpp， type = string， aliases：convert_model_file

- 仅用于 convert_model 任务
- 输出转换模型的文件名
- **注意**：只能在 CLI 版本中使用

客观参数

- num_class ， default =1， type = int， aliases :num_classes, constraints：num_class > 0

- 仅用于 multi-class 分类申请

- is_unbalance ， 默认=false，类型= bool，别名：unbalance，unbalanced_sets

- 仅用于 binary 应用程序
- true 如果训练数据不平衡，则设置此项
- **注意**：此参数不能在同一时间使用 scale_pos_weight，只能选择一个人

- scale_pos_weight ， 默认=1.0，类型=双，约束：scale_pos_weight > 0.0

- 仅用于 binary 应用程序
- 具有正面等级的标签的重量
- **注意**：此参数不能在同一时间使用 is_unbalance，只能选择一个人

- sigmoid ， 默认=1.0，类型=双，约束：sigmoid > 0.0

- 只有在使用 binary 和 multiclassova 分类和 lambdarank 应用
- sigmoid 函数的参数

- boost_from_average ， 默认=true， 类型= bool

- 只用在 regression， binary 和 cross-entropy 应用程序

- 将初始分数调整为标签的均值以便更快收敛

- `reg_sqrt`，默认=`false`，类型= bool

- 仅用于 `regression` 应用程序
- 用于拟合 `sqrt(label)` 而不是原始值，预测结果也将自动转换为 `prediction^2`
- 在大范围标签的情况下可能有用

- `alpha`，默认=`0.9`，类型=双，约束：`alpha > 0.0`

- 仅用于 `huber` 和 `quantile regression` 应用程序
- 参数胡伯损失和分位数回归

- `fair_c`，默认=`1.0`，类型=双，约束：`fair_c > 0.0`

- 仅用于 `fair regression` 应用程序
- 公平损失的参数

- `poisson_max_delta_step`，默认=`0.7`，类型=双，约束：`poisson_max_delta_step > 0.0`

- 仅用于 `poisson regression` 应用程序
- 泊松回归的参数以保证优化

- `tweedie_variance_power`，默认=`1.5`，类型=双，约束：`1.0 <= tweedie_variance_power < 2.0`

- 仅用于 `tweedie regression` 应用程序
- 用于控制 tweedie 分布的方差
- 将其设置得更接近于 `2` 向 **Gamma** 分布转变
- 设置此更接近 `1` 朝向换档**泊松**分布

- `max_position`，default =`20`，type = int，constraints：`max_position > 0`

- 仅用于 `lambdarank` 应用程序
- 在这个位置优化 NDCG

- `label_gain`，默认=`0,1,3,7,15,31,63,...,2^30-1`，type = multi-double

- 仅用于 `lambdarank` 应用程序

- 标签的相关收益。例如，标签的增益 2 是 3 在默认标签增益的情况下
- 分开 ,

度量参数

- metric ， default =""， type = multi-enum， aliases :metrics,metric_types
- 在评估集上评估的度量标准
- ""（空字符串或未指定）表示 objective 将使用与指定对应的度量标准（这仅适用于预定义的目标函数，否则不会添加评估度量标准）
- "None"（字符串，不一个 None 值）意味着没有度量将被注册，别名： na， null， custom
- l1，绝对的损失，别名：mean_absolute_error，mae，regression_l1
- l2，平方损失，别名：mean_squared_error，mse，regression_l2，regression
- l2_root，根方损失，别名：root_mean_squared_error，rmse
- quantile，分位数回归
- mape，MAPE 丢失，别名：mean_absolute_percentage_error
- huber，胡贝尔损失
- fair，公平的损失
- poisson，泊松回归的负对数似然
- gamma，Gamma 回归的负对数似然
- gamma_deviance，Gamma 回归的剩余偏差
- tweedie，Tweedie 回归的负对数似然
- ndcg，NDCG，别名：lambdarank
- map，MAP，别名：mean_average_precision
- auc，AUC
- binary_logloss，日志丢失，别名：binary
- binary_error，对于一个样本：0 正确分类，1 用于错误分类
- multi_logloss，日志多类分类的损失，别名： multiclass，softmax， multiclassova， multiclass_ova， ova， ovr
- multi_error，多级分类的错误率
- xentropy，交叉熵（带可选的线性权重），别名： cross_entropy
- xentlambda，“强度加权”交叉熵，别名： cross_entropy_lambda
- kldiv，Kullback-Leibler 分歧，别名： kullback_leibler
- 支持多个指标，以。分隔 ,

- `metric_freq`，default =`1`，type = int，aliases :`output_freq`, constraints：`metric_freq > 0`
- 公制输出的频率
- `is_provide_training_metric`，默认=`false`，类型=布尔，别名：`training_metric`，`is_training_metric`，`train_metric`
- 将此设置为 `true` 在训练数据集上输出度量标准结果
- **注意**：只能在 CLI 版本中使用
- `eval_at`，默认=`1,2,3,4,5`，键入=多 INT，别名：`ndcg_eval_at`，`ndcg_at`，`map_eval_at`，`map_at`
- 仅用于 `ndcg` 和 `map` 指标
- NDCG 和 MAP 评估位置，由，

网络参数

- `num_machines`，default =`1`，type = int，aliases :`num_machine`, constraints：`num_machines > 0`
- 并行学习应用程序的机器数量
- 需要在 **socket** 和 **mpi** 版本中设置此参数
- `local_listen_port`，默认=`12400`，类型= INT，别名：`local_port`，`port`，约束：`local_listen_port > 0`
- 本地计算机的 TCP 侦听端口
- **注意**：在培训之前，请不要忘记在防火墙设置中允许此端口
- `time_out`，default =`120`，type = int，constraints：`time_out > 0`
- 套接字超时，以分钟为单位
- `machine_list_filename`，默认=`""`，类型=串，别名：`machine_list_file`，`machine_list`，`mlist`
- 列出此并行学习应用程序的计算机的文件路径
- 每行包含一台 IP 和一台机器的一个端口。格式是（空格作为分隔符）`ip port`

- `machines`，default =`""`，type = string，aliases :`workers`,`nodes`
- 以下格式的机器列表：`ip1:port1,ip2:port2`

GPU 参数

- `gpu_platform_id`，default =`-1`，type = int
- OpenCL 平台 ID。通常每个 GPU 供应商都会公开一个 OpenCL 平台
- `-1` 表示系统范围的默认平台
- **注意**：有关详细信息，请参阅 GPU 目标

- `gpu_device_id`，default =`-1`，type = int
- 指定平台中的 OpenCL 设备 ID。所选平台中的每个 GPU 都具有唯一的设备 ID
- `-1` 表示所选平台中的默认设备
- **注意**：有关详细信息，请参阅 GPU 目标

- `gpu_use_dp`，默认=`false`，类型= bool
- 将此设置为 `true` 在 GPU 上使用双精度数学运算（默认情况下使用单精度）

其他

使用输入分数继续训练

LightGBM 支持初始分数的持续训练。它使用附加文件来存储这些初始分数，如下所示：

```
0.5-0.10.9...
```

这意味着第一个数据行的初始分数是`0.5`，第二个是`-0.1`，依此类推。初始分数文件逐行对应于数据文件，并且每行具有每个分数。

如果数据文件的名称是 `train.txt`，则初始分数文件应命名为 `train.txt.init` 与数据文件在同一文件夹中。在这种情况下，LightGBM 将自动加载初始分数文件（如果存在）。

否则，您应该通过 `initscore_filename` 参数指定自定义命名文件的路径以及初始分数。

重量数据

LightGBM 支持加权训练。它使用附加文件来存储权重数据，如下所示：

```
1.00.50.8...
```

这意味着第一个数据行的权重是 `1.0`，第二个是 `0.5`，依此类推。权重文件逐行对应于数据文件，并且每行具有权重。

如果数据文件的名称是 `train.txt`，则权重文件应该命名为 `train.txt.weight` 并放在与数据文件相同的文件夹中。在这种情况下，LightGBM 将自动加载权重文件（如果存在）。

此外，您可以在数据文件中包含权重列。请参考上面的 `weight_column` 参数。

查询数据

对于 LambdaRank 学习，它需要用于训练数据的查询信息。LightGBM 使用附加文件来存储查询数据，如下所示：

```
271867...
```

这意味着第一 `27` 行样本属于一个查询，下一 `18` 行属于另一个，依此类推。

注意：数据应按查询排序。

如果数据文件的名称是 `train.txt`，则查询文件应该命名为 `train.txt.query` 并放在与数据文件相同的文件夹中。在这种情况下，LightGBM 将自动加载查询文件（如果存在）。

此外，您可以在数据文件中包含查询/组 ID 列。请参考上面的 `group_column` 参数。

功能

这是 LightGBM 如何工作的概念性概述[1]。我们假设熟悉决策树提升算法，而不是关注可能与其他提升包不同的 LightGBM 方面。有关详细算法，请参阅引文或源代码。

速度和内存使用的优化

许多增强工具使用基于预排序的算法[2,3]（例如 xgboost 中的默认算法）用于决策树学习。这是一个简单的解决方案，但不容易优化。

LightGBM 使用基于直方图的算法[4,5,6]，它将连续特征（属性）值存储到离散区间。这加快了培训速度并减少了内存使用量。基于直方图的算法的优点包括：

- **降低计算每次拆分增益的成本**
- 基于预排序的算法具有时间复杂性 `O(#data)`
- 计算直方图具有时间复杂度 `O(#data)`，但这仅涉及快速总结操作。构建直方图后，基于直方图的算法具有时间复杂度 `O(#bins)`，并且`#bins` 远小于`#data`。
- **使用直方图减法进一步加速**
- 要在二叉树中获取一个叶子的直方图，请使用其父级及其邻居的直方图减法
- 因此，它需要仅为一个叶子构建直方图（小于`#data` 其邻居）。然后它可以通过直方图减法获得其邻居的直方图，成本较低（`O(#bins)`）
- **减少内存使用量**
- 用离散箱替换连续值。如果`#bins` 很小，可以使用小数据类型，例如 uint8_t 来存储训练数据
- 无需存储用于预排序特征值的其他信息
- **降低并行学习的通信成本**

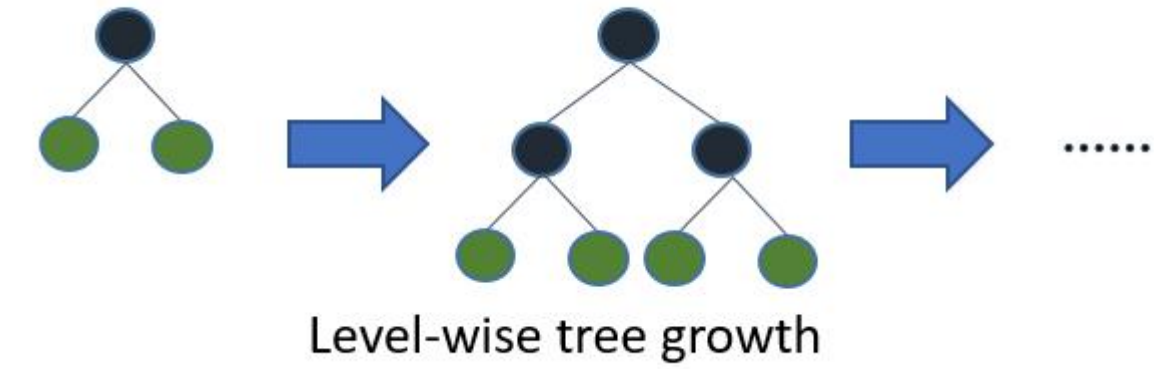
稀疏优化

- 只需要为稀疏特征构造直方图 `O(2 * #non_zero_data)`

精度优化

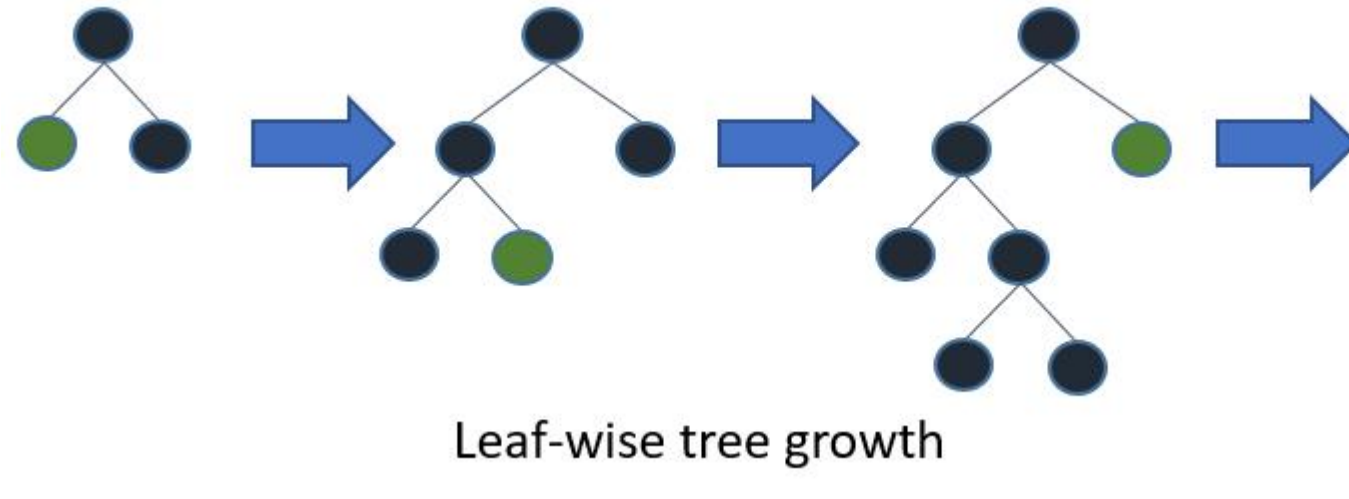
叶子（最好的）树生长

大多数决策树学习算法按级别（深度）方向生成树，如下图所示：



LightGBM 以叶子方式生长树木（最佳优先）[7]。它将选择具有最大增量损失的叶子来生长。保持`#leaf`固定的叶子算法往往比水平算法实现更低的损失。

叶子方式可能会导致过度拟合`#data`，因此 LightGBM 包含 `max_depth` 限制树深度的参数。然而，即使 `max_depth` 指定了树木，树木仍然会逐渐生长。



分类特征的最佳分割

通常使用单热编码来表示分类特征，但这种方法对于树学习者来说是次优的。特别是对于高基数分类特征，基于单热特征构建的树往往是不平衡的，并且需要非常深地生长以实现良好的准确性。

最佳解决方案是通过将其类别划分为 2 个子集来分割分类特征，而不是单热编码。如果要素具有 `k` 类别，则可能存在分区。但是回归树有一个有效的解决方案[8]。它需要找到最佳分区。`2^(k-1) - 1O(k * log(k))`

基本思想是根据每次拆分的培训目标对类别进行排序。更具体地说，LightGBM 根据其累积值（）对直方图（对于分类特征）进行排序，然后在排序的直方图上找到最佳分割。`sum_gradient / sum_hessian`

网络通信的优化

在 LightGBM 的并行学习中，它只需要使用一些集体通信算法，如“All reduce”，“All gather”和“Reduce scatter”。LightGBM 实现了最先进的算法[9]。这些集体通信算法可以提供比点对点通信更好的性能。

并行学习中的优化

LightGBM 提供以下并行学习算法。

特征并行

传统算法

并行特征旨在并行化决策树中的“查找最佳拆分”。传统特征并行的过程是：

1. 垂直分区数据（不同的机器具有不同的功能集）。
2. 工作人员在本地功能集上找到本地最佳分割点{feature，threshold}。
3. 彼此沟通本地最好的分裂，并获得最好的分裂。
4. 具有最佳拆分的工作人员执行拆分，然后将拆分的数据结果发送给其他工作人员。
5. 其他工作人员根据收到的数据分割数据。

传统特征并行的缺点：

- 有计算开销，因为它无法加速时间复杂度的“分裂” `O(#data)`。因此，并行特征在`#data`大时不能很好地加速。
- 需要分割结果的通信，其成本约为（一个数据一位）。`O(#data / 8)`

LightGBM 中的并行特征

由于特征并行在`#data`大的时候不能很好地加速，我们做了一点改变：不是垂直分割数据，而是每个工人都拥有完整的数据。因此，LightGBM 不需要为分割数据结果进行通信，因为每个工作人员都知道如何分割数据。并且`#data`不会更大，因此在每台机器中保存完整数据是合理的。

LightGBM 中并行功能的过程：

1. 工作人员在本地功能集上找到本地最佳分割点{feature，threshold}。
2. 彼此沟通本地最好的分裂，并获得最好的分裂。
3. 执行最佳分割。

但是，这个特征并行算法仍然会受到“分裂”时计算开销的影响`#data`。因此，当`#data`大的时候使用并行数据会更好。

数据并行

传统算法

数据并行旨在并行化整个决策学习。数据并行的过程是：

1. 水平分区数据。
2. 工人使用本地数据构建局部直方图。
3. 合并来自所有局部直方图的全局直方图。
4. 从合并的全局直方图中找到最佳拆分，然后执行拆分。

传统数据并行的缺点：

- 通信成本高。如果使用点对点通信算法，则一台机器的通信成本约为。如果使用集体通信算法（例如“All Reduce”），通信成本约为（在[9]的第 4.5 章中检查“All Reduce”的成本）。`O(#machine * #feature * #bin)O(2 * #feature * #bin)`

LightGBM 中的数据并行

我们降低了 LightGBM 中并行数据的通信成本：

1. LightGBM 使用“Reduce Scatter”来合并不同工人的不同（非重叠）特征的直方图，而不是“从所有局部直方图中合并全局直方图”。然后，工作人员在本地合并直方图上找到本地最佳分割，并同步全局最佳分割。
2. 如前所述，LightGBM 使用直方图减法来加速训练。基于此，我们只能为一片叶子传达直方图，并通过减法得到其邻居的直方图。

考虑到所有因素，LightGBM 中的数据并行具有时间复杂性。

```
O(0.5 * #feature * #bin)
```

投票并行

投票并行进一步降低了数据并行中的通信成本，使其成本不变。它使用两阶段投票来降低特征直方图的通信成本[10]。

GPU 支持

感谢@ huanzhang12 提供此功能。请阅读[11]以获取更多详细信息。

- GPU 安装
- GPU 教程

应用程序和指标

LightGBM 支持以下应用程序：

- 回归，目标函数是 L2 损失
- 二进制分类，目标函数是 logloss
- 多分类
- 交叉熵，目标函数是 logloss，支持非二进制标签的训练
- lambdarank，目标函数是使用 NDCG 的 lambdarank

LightGBM 支持以下指标：

- L1 损失
- L2 损失
- 记录丢失
- 分类错误率
- AUC
- NDCG
- 地图
- 多级日志丢失
- 多级错误率
- 公平
- 胡伯
- 泊松
- 位数
- MAPE
- 库勒巴克-莱布勒
- 伽玛
- 特威迪

有关更多详细信息，请参阅参数。

其他功能

- `max_depth` 树的限制，同时生长树叶
- 镖
- L1 / L2 正则化
- 套袋
- 列（特征）子样本
- 继续列车输入 GBDT 模型
- 继续训练输入得分文件
- 加权培训
- 培训期间的验证度量输出
- 多验证数据
- 多指标
- 提前停止（训练和预测）
- 叶指数的预测

有关更多详细信息，请参阅参数。

高级主题

缺少价值处理

- LightGBM 默认启用缺失值句柄。通过设置禁用它 `use_missing=false`。
- LightGBM 默认使用 NA（NaN）表示缺失值。通过设置将其更改为使用零 `zero_as_missing=true`。
- 当 `zero_as_missing=false`（默认）时，稀疏矩阵（和 LightSVM）中未显示的值被视为零。
- 何时 `zero_as_missing=true`，NA 和零（包括稀疏矩阵（和 LightSVM）中未显示的值）被视为缺失。

分类特征支持

- LightGBM 通过整数编码的分类功能提供良好的准确性。LightGBM 应用 Fisher（1958） 来找到这里描述的最佳分类 。这通常比单热编码表现更好。
- 使用 `categorical_feature` 指定的类别特征。请参阅参数 `categorical_feature` 中的参数。
- 必须将分类特征编码为 `int` 小于 `Int32.MaxValue`（2147483647）的非负整数（）。最好使用从零开始的连续范围的整数。
- 使用 `min_data_per_group`，`cat_smooth` 来处理过度拟合（当`#data` 小或`#category` 大时）。
- 对于具有高基数（`#category` 大）的分类特征，通常最好将该特征视为数字，或者通过简单地忽略整数的分类解释或通过将类别嵌入到低维数字空间中。

LambdaRank

- 标签应该是类型 `int`，使得较大的数字对应于较高的相关性（例如 0：差，1：公平，2：良好，3：完美）。
- 使用 `label_gain` 设定的增益（权重） `int` 标签。
- 使用 `max_position` 设置 NDCG 优化位置。

参数调整

案例

lightGBM 简介

xgboost 的出现，让数据民工们告别了传统的机器学习算法们：RF、GBM、SVM、LASSO……..。现在微软推出了一个新的 boosting 框架，想要挑战 xgboost 的江湖地位。

顾名思义，lightGBM 包含两个关键点：light 即轻量级，GBM 梯度提升机。

LightGBM 是一个梯度 boosting 框架，使用基于学习算法的决策树。它可以说是分布式的，高效的，有以下优势：

- 更快的训练效率
- 低内存使用
- 更高的准确率
- 支持并行化学习
- 可处理大规模数据

xgboost 缺点

其缺点，或者说不足之处：

每轮迭代时，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。

预排序方法（pre-sorted）：首先，空间消耗大。这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如排序后的索引，为了后续快速的计算分割点），这里需要消耗训练数据两倍的内存。其次时间上也有较大的开销，在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。

对 cache 优化不友好。在预排序后，特征对梯度的访问是一种随机访问，并且不同的特征访问的顺序不一样，无法对 cache 进行优化。同时，在每一层长树的时候，需要随机访问一个行索引到叶子索引的数组，并且不同特征访问的顺序也不一样，也会造成较大的 cache miss。

lightGBM 特点

以上与其说是 xgboost 的不足，倒不如说是 lightGBM 作者们构建新算法时着重瞄准的点。解决了什么问题，那么原来模型没解决就成了原模型的缺点。

- 概括来说，lightGBM 主要有以下特点：
- 基于 Histogram 的决策树算法
- 带深度限制的 Leaf-wise 的叶子生长策略
- 直方图做差加速
- 直接支持类别特征(Categorical Feature)
- Cache 命中率优化
- 基于直方图的稀疏特征优化
- 多线程优化
- 前 2 个特点使我们尤为关注的。
- Histogram 算法

直方图算法的基本思想：先把连续的浮点特征值离散化成 k 个整数，同时构造一个宽度为 k 的直方图。遍历数据时，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。

带深度限制的 Leaf-wise 的叶子生长策略

Level-wise 过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。但实际上 Level-wise 是一种低效算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

Leaf-wise 则是一种更为高效的策略：每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。因此同 Level-wise 相比，在分裂次数相同的情况下，Leaf-wise 可以降低更多的误差，得到更好的精度。

Leaf-wise 的缺点：可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在 Leaf-wise 之上增加了一个最大深度限制，在保证高效率的同时防止过拟合。

xgboost 和 lightgbm

决策树算法

XGBoost 使用的是 pre-sorted 算法，能够更精确的找到数据分隔点；

首先，对所有特征按数值进行预排序。
其次，在每次的样本分割时，用 O(# data)的代价找到每个特征的最优分割点。
最后，找到最后的特征以及分割点，将数据分裂成左右两个子节点。

优缺点：

这种 pre-sorting 算法能够准确找到分裂点，但是在空间和时间上有很大的开销。

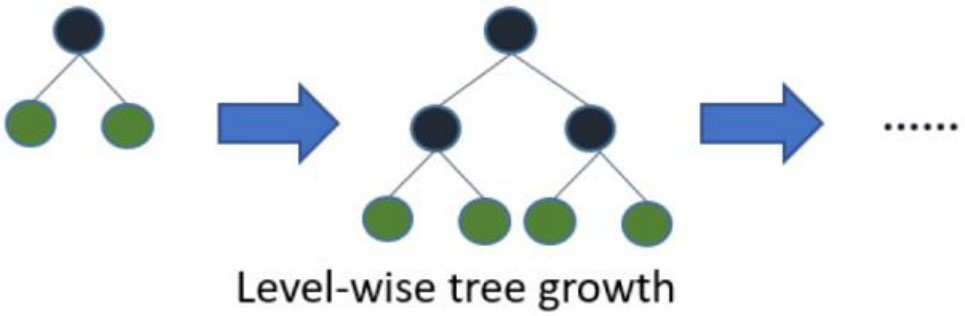
i. 由于需要对特征进行预排序并且需要保存排序后的索引值（为了后续快速的计算分裂点），因此内存需要训练数据的两倍。
ii. 在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。
LightGBM 使用的是 histogram 算法，占用的内存更低，数据分隔的复杂度更低。

其思想是将连续的浮点特征离散成 k 个离散值，并构造宽度为 k 的 Histogram。然后遍历训练数据，统计每个离散值在直方图中的累计统计量。在进行特征选择时，只需要根据直方图的离散值，遍历寻找最优的分割点。

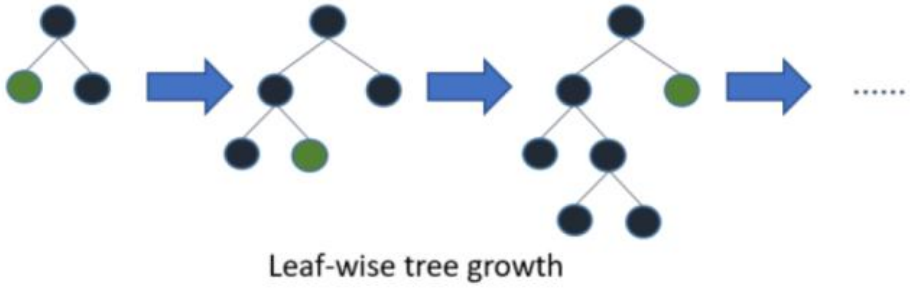
Histogram 算法的优缺点：

Histogram 算法并不是完美的。由于特征被离散化后，找到的并不是很精确的分割点，所以会对结果产生影响。但在实际的数据集上表明，离散化的分裂点对最终的精度影响并不大，甚至会好一些。原因在于 decision tree 本身就是一个弱学习器，采用 Histogram 算法会起到正则化的效果，有效地防止模型的过拟合。
时间上的开销由原来的 O(#data * #features)降到 O(k * #features)。由于离散化，#bin 远小于#data，因此时间上有很大的提升。
Histogram 算法还可以进一步加速。一个叶子节点的 Histogram 可以直接由父节点的 Histogram 和兄弟节点的 Histogram 做差得到。一般情况下，构造 Histogram 需要遍历该叶子上的所有数据，通过该方法，只需要遍历 Histogram 的 k 个捅。速度提升了一倍。
决策树生长策略

XGBoost 采用的是按层生长 level（depth）-wise 生长策略，如 Figure 1 所示，能够同时分裂同一层的叶子，从而进行多线程优化，不容易过拟合；但不加区分的对待同一层的叶子，带来了很多没必要的开销。因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。



LightGBM 采用 leaf-wise 生长策略，如 Figure 2 所示，每次从当前所有叶子中找到分裂增益最大（一般也是数据量最大）的一个叶子，然后分裂，如此循环。因此同 Level-wise 相比，在分裂次数相同的情况下，Leaf-wise 可以降低更多的误差，得到更好的精度。Leaf-wise 的缺点是可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在 Leaf-wise 之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



网络通信优化

XGBoost 由于采用 pre-sorted 算法，通信代价非常大，所以在并行的时候也是采用 histogram 算法；LightGBM 采用的 histogram 算法通信代价小，通过使用集合通信算法，能够实现并行计算的线性加速。

LightGBM 支持类别特征

实际上大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征，转化 one-hotting 特征，降低了空间和时间的效率。而类别特征的使用是在实践中很常用的。基于这个考虑，LightGBM 优化了对类别特征的支持，可以直接输入类别特征，不需要额外的 0/1 展开。并在决策树算法上增加了类别特征的决策规则。

lightGBM 调参

所有的参数含义，参考：http://lightgbm.apachecn.org/cn/latest/Parameters.html

调参过程：

（1）num_leaves

LightGBM 使用的是 leaf-wise 的算法，因此在调节树的复杂程度时，使用的是 num_leaves 而不是 max_depth。

大致换算关系：num_leaves = 2^(max_depth)

（2）样本分布非平衡数据集：可以 param[‘is_unbalance’]=’true’

（3）Bagging 参数：bagging_fraction+bagging_freq（必须同时设置）、feature_fraction

（4）min_data_in_leaf、min_sum_hessian_in_leaf

实战 iris-sklearn 接口形式的 LightGBM 示例

这里主要以 sklearn 的使用形式来使用 lightgbm 算法，包含建模，训练，预测，网格参数优化。

```
import lightgbm as lgb
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.datasets import    make_classification
#  加载数据
print('Load data...')

iris = load_iris()
data=iris.data
target = iris.target
X_train,X_test,y_train,y_test =train_test_split(data,target,test_size=0.2)

# df_train = pd.read_csv('../regression/regression.train', header=None, sep='\t')
```

```
# df_test = pd.read_csv('../regression/regression.test', header=None, sep='\t')
# y_train = df_train[0].values
# y_test = df_test[0].values
# X_train = df_train.drop(0, axis=1).values
# X_test = df_test.drop(0, axis=1).values

print('Start training...')
# 创建模型，训练模型
gbm = lgb.LGBMRegressor(objective='regression',num_leaves=31,learning_rate=0.05,n_estimators=20)
gbm.fit(X_train, y_train,eval_set=[(X_test, y_test)],eval_metric='l1',early_stopping_rounds=5)

print('Start predicting...')
# 测试机预测
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)
# 模型评估
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)

# feature importances
print('Feature importances:', list(gbm.feature_importances_))

# 网格搜索，参数优化
estimator = lgb.LGBMRegressor(num_leaves=31)

param_grid = {
    'learning_rate': [0.01, 0.1, 1],
    'n_estimators': [20, 40]
}

gbm = GridSearchCV(estimator, param_grid)

gbm.fit(X_train, y_train)

print('Best parameters found by grid search are:', gbm.best_params_)
```

实战 iris-原生形式使用 lightgbm

```
# coding: utf-8
# pylint: disable = invalid-name, C0111
import json
import lightgbm as lgb
```

```
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.datasets import    make_classification

iris = load_iris()
data=iris.data
target = iris.target
X_train,X_test,y_train,y_test =train_test_split(data,target,test_size=0.2)


# 加载你的数据
# print('Load data...')
# df_train = pd.read_csv('../regression/regression.train', header=None, sep='\t')
# df_test = pd.read_csv('../regression/regression.test', header=None, sep='\t')
#
# y_train = df_train[0].values
# y_test = df_test[0].values
# X_train = df_train.drop(0, axis=1).values
# X_test = df_test.drop(0, axis=1).values

# 创建成 lgb 特征的数据集格式
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

# 将参数写成字典下形式
params = {
    'task': 'train',
    'boosting_type': 'gbdt',    # 设置提升类型
    'objective': 'regression', # 目标函数
    'metric': {'l2', 'auc'},    # 评估函数
    'num_leaves': 31,     # 叶子节点数
    'learning_rate': 0.05,    # 学习速率
    'feature_fraction': 0.9, # 建树的特征选择比例
    'bagging_fraction': 0.8, # 建树的样本采样比例
    'bagging_freq': 5,    # k 意味着每 k 次迭代执行 bagging
    'verbose': 1 # <0 显示致命的, =0 显示错误 (警告), >0 显示信息
}

print('Start training...')
# 训练 cv and train
gbm                                                                                    =
lgb.train(params,lgb_train,num_boost_round=20,valid_sets=lgb_eval,early_stopping
```

```
_rounds=5)

print('Save model...')
# 保存模型到文件
gbm.save_model('model.txt')

print('Start predicting...')
# 预测数据集
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
# 评估模型
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
```