LightGBM 中文文档

LightGBM 是一个梯度 boosting 框架, 使用基于学习算法的决策树. 它是分布式的, 高效的, 装逼的, 它具有以下优势: * 速度和内存使用的优化 * 减少分割增益的计算量 * 通过直方图的相减来进行进一步的加速 * 减少内存的使用 减少并行学习的通信代价 * 稀疏优化 * 准确率的优化 * Leaf-wise (Best-first) 的决策树生长策略 * 类别特征值的最优分割 * 网络通信的优化 * 并行学习的优化 * 特征并行 * 数据并行 * 投票并行 * GPU 支持可处理大规模数据

更多有关 LightGBM 特性的详情, 请参阅: LightGBM 特性.

文档地址

- 在线阅读
- 在线阅读（Gitee）

项目负责人

- @那伊抹微笑

项目贡献者

- @那伊抹微笑
- @陈瑶
- @胡世昌
- @王金树
- @谢家柯
- @方振影
- @臧艺
- @冯斐
- @黄志浩
- @刘陆琛
- @周立刚
- @陈洪
- @孙永杰

- @王贤才

下载

Docker

```
docker pull apachecn0/lightgbm-doc-zh
docker run -tid -p <port>:80 apachecn0/lightgbm-doc-zh
# 访问 http://localhost:{port} 查看文档
```

PYPI

```
pip install lightgbm-doc-zh
lightgbm-doc-zh <port>
# 访问 http://localhost:{port} 查看文档
```

NPM

```
npm install -g lightgbm-doc-zh
lightgbm-doc-zh <port>
# 访问 http://localhost:{port} 查看文档
```

贡献指南

为了使项目更加便于维护，我们将文档格式全部转换成了 Markdown，同时更换了页面生成器。后续维护工作将完全在 Markdown 上进行。

小部分格式仍然存在问题，主要是链接和表格。需要大家帮忙找到，并提 PullRequest 来修复。

建议反馈

- 联系项目负责人 @那伊抹微笑.
- 在我们的 apachecn/lightgbm-doc-zh github 上提 issue.
- 发送到 Email: lightgbm#apachecn.org（#替换成@）.
- 在我们的 组织学习交流群 中联系群主/管理员即可.

组织学习交流群

机器学习交流群: 629470233 （2000人）

大数据交流群: 214293307 （2000人）

了解我们: http://www.apachecn.org/organization/209.html

加入组织: http://www.apachecn.org/organization/209.html

更多信息请参阅: http://www.apachecn.org/organization/348.html

安装指南

该页面是 LightGBM CLI 版本的构建指南.

要构建 Python 和 R 的软件包, 请分别参阅 Python-package 和 R-package 文件夹.

Windows

LightGBM 可以使用 Visual Studio, MSBuild 与 CMake 或 MinGW 来在 Windows 上构建.

Visual Studio (or MSBuild)

使用 GUI

1. 安装 Visual Studio (2015 或更新版本).
2. 下载 zip archive 并且 unzip（解压）它.
3. 定位到 `LightGBM-master/windows` 文件夹.
4. 使用 Visual Studio 打开 `LightGBM.sln` 文件, 选择 `Release` 配置并且点击 `BUILD` -> `Build Solution (Ctrl+Shift+B)`.

   如果出现有关 **Platform Toolset** 的错误, 定位到 `PROJECT` -> `Properties` -> `Configuration Properties` -> `General` 然后选择 toolset 安装到你的机器.

该 exe 文件可以在 `LightGBM-master/windows/x64/Release` 文件夹中找到.

使用命令行

1. 安装 Git for Windows, CMake (3.8 或更新版本) 以及 MSBuild (**MSBuild** 是非必要的, 如果已安装 **Visual Studio** (2015 或更新版本) 的话).
2. 运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM
cd LightGBM
mkdir build
cd build
cmake -DCMAKE_GENERATOR_PLATFORM=x64 ..
cmake --build . --target ALL_BUILD --config Release
```

这些 exe 和 dll 文件可以在 `LightGBM/Release` 文件夹中找到.

MinGW64

1. 安装 Git for Windows, CMake 和 MinGW-w64.
2. 运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM
cd LightGBM
mkdir build
cd build
cmake -G "MinGW Makefiles" ..
mingw32-make.exe -j4
```

这些 exe 和 dll 文件可以在 `LightGBM/` 文件夹中找到.

注意: 也许你需要再一次运行 `cmake -G "MinGW Makefiles" ..` 命令, 如果遇到 `sh.exe was found in your PATH` 错误的话.

也许你还想要参阅 gcc 建议.

Linux

LightGBM 使用 **CMake** 来构建. 运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM ; cd LightGBM
mkdir build ; cd build
cmake ..
make -j4
```

注意: glibc >= 2.14 是必须的.

也许你还想要参阅 gcc 建议.

OSX

LightGBM 依赖于 **OpenMP** 进行编译, 然而 Apple Clang 不支持它.

请使用以下命令来安装 **gcc/g++** :

```
brew install cmake
brew install gcc --without-multilib
```

然后安装 LightGBM:

```
git clone --recursive https://github.com/Microsoft/LightGBM ; cd LightGBM
export CXX=g++-7 CC=gcc-7
mkdir build ; cd build
```

```
cmake ..
make -j4
```

也许你还想要参阅 gcc 建议.

Docker

请参阅 Docker 文件夹.

Build MPI 版本

LightGBM 默认的构建版本是基于 socket 的的. LightGBM 也支持 MPI. MPI 是一种与 RDMA 支持的高性能通信方法.

如果您需要运行具有高性能通信的并行学习应用程序, 则可以构建带有 MPI 支持的 LightGBM.

Windows

使用 GUI

1. 需要先安装 MS MPI . 需要 `msmpisdk.msi` 和 `MSMpiSetup.exe` .
2. 安装 Visual Studio (2015 或更新版本).
3. 下载 zip archive 并且 unzip（解压）它.
4. 定位到 `LightGBM-master/windows` 文件夹.
5. 使用 Visual Studio 打开 `LightGBM.sln` 文件, 选择 `Release_mpi` 配置并且点击 `BUILD` -> `Build Solution (Ctrl+Shift+B)` .

   如果遇到有关 **Platform Toolset** 的错误, 定位到 `PROJECT` -> `Properties` -> `Configuration Properties` -> `General` 并且选择安装 toolset 到你的机器上.

该 exe 文件可以在 `LightGBM-master/windows/x64/Release_mpi` 文件夹中找到.

使用命令行

1. 需要先安装 MS MPI . 需要 `msmpisdk.msi` 和 `MSMpiSetup.exe` .
2. 安装 Git for Windows, CMake (3.8 或更新版本) 和 MSBuild (MSBuild 是非必要的, 如果已安装 **Visual Studio** (2015 或更新版本)).
3. 运行以下命令:

   ```
   git clone --recursive https://github.com/Microsoft/LightGBM
   cd LightGBM
   mkdir build
   ```

```
cd build
cmake -DCMAKE_GENERATOR_PLATFORM=x64 -DUSE_MPI=ON ..
cmake --build . --target ALL_BUILD --config Release
```

这些 exe 和 dll 文件可以在 `LightGBM/Release` 文件夹中找到.

注意: Build MPI version 通过 **MinGW** 来构建 MPI 版本的不支持的, 由于它里面缺失了 MPI 库.

Linux

需要先安装 Open MPI .

然后运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM ; cd LightGBM
mkdir build ; cd build
cmake -DUSE_MPI=ON ..
make -j4
```

Note: glibc >= 2.14 是必要的.

OSX

先安装 **gcc** 和 **Open MPI** :

```
brew install openmpi
brew install cmake
brew install gcc --without-multilib
```

然后运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM ; cd LightGBM
export CXX=g++-7 CC=gcc-7
mkdir build ; cd build
cmake -DUSE_MPI=ON ..
make -j4
```

Build GPU 版本

Linux

在编译前应该先安装以下依赖:

- OpenCL 1.2 headers and libraries, 它们通常由 GPU 制造商提供.

The generic OpenCL ICD packages (for example, Debian package `cl-icd-libopencl1` and `cl-icd-opencl-dev`) can also be used.

- libboost 1.56 或更新版本 (1.61 或最新推荐的版本).

  We use Boost.Compute as the interface to GPU, which is part of the Boost library since version 1.61. However, since we include the source code of Boost.Compute as a submodule, we only require the host has Boost 1.56 or later installed. We also use Boost.Align for memory allocation. Boost.Compute requires Boost.System and Boost.Filesystem to store offline kernel cache.

  The following Debian packages should provide necessary Boost libraries: `libboost-dev`, `libboost-system-dev`, `libboost-filesystem-dev`.

- CMake 3.2 或更新版本.

要构建 LightGBM GPU 版本, 运行以下命令:

```
git clone --recursive https://github.com/Microsoft/LightGBM ; cd LightGBM
mkdir build ; cd build
cmake -DUSE_GPU=1 ..
# if you have installed the NVIDIA OpenGL, please using following instead
# sudo cmake -DUSE_GPU=1 -DOpenCL_LIBRARY=/usr/local/cuda/lib64/libOpenCL.so -
OpenCL_INCLUDE_DIR=/usr/local/cuda/include/ ..
make -j4
```

Windows

如果使用 **MinGW**, 该构建过程类似于 Linux 上的构建. 相关的更多细节请参阅 GPU Windows 平台上的编译 .

以下构建过程适用于 MSVC (Microsoft Visual C++) 构建.

1. 安装 Git for Windows, CMake (3.8 or higher) 和 MSBuild (MSBuild 是非必要的, 如果已安装 **Visual Studio** (2015 或更新版本)).
2. 针对 Windows 平台安装 **OpenCL** . 安装取决于你的 GPU 显卡品牌 (NVIDIA, AMD, Intel).
   - 要运行在 Intel 上, 获取 Intel SDK for OpenCL.
   - 要运行在 AMD 上, 获取 AMD APP SDK.
   - 要运行在 NVIDIA 上, 获取 CUDA Toolkit.
   - 安装 Boost Binary.

   注意: 要匹配你的 Visual C++ 版本:

   Visual Studio 2015 -> `msvc-14.0-64.exe`,

   Visual Studio 2017 -> `msvc-14.1-64.exe`.
3. 运行以下命令:

```
Set BOOST_ROOT=C:\local\boost_1_64_0\
Set BOOST_LIBRARYDIR=C:\local\boost_1_64_0\lib64-msvc-14.0
git clone --recursive https://github.com/Microsoft/LightGBM
cd LightGBM
mkdir build
cd build
cmake -DCMAKE_GENERATOR_PLATFORM=x64 -DUSE_GPU=1 ..
cmake --build . --target ALL_BUILD --config Release
```

注意: `C:\local\boost_1_64_0\` 和 `C:\local\boost_1_64_0\lib64-msvc-14.0` 是你 Boost 二进制文件的位置. 你还可以将它们设置为环境变量, 以在构建时避免 `Set ...` 命令.

Protobuf 支持

如果想要使用 protobuf 来保存和加载模型, 请先安装 protobuf c++ version . 然后使用 USE_PROTO=ON 配置来运行 cmake 命令, 例如:

```
cmake -DUSE_PROTO=ON ..
```

然后在保存或加载模型时, 可以在参数中使用 `model_format=proto`.

注意: 针对 windows 用户, 它只对 mingw 进行了测试.

Docker

请参阅 GPU Docker 文件夹.

快速入门指南

本文档是 LightGBM CLI 版本的快速入门指南。

参考 安装指南 先安装 LightGBM 。

其他有帮助的链接列表

- 参数
- 参数调整
- Python 包快速入门
- Python API

训练数据格式

LightGBM 支持 CSV, TSV 和 LibSVM 格式的输入数据文件。

Label 是第一列的数据，文件中是不包含 header（标题） 的。

类别特征支持

12/5/2016 更新:

LightGBM 可以直接使用 categorical feature（类别特征）（不需要单独编码）。 Expo data 实验显示，与 one-hot 编码相比，其速度提高了 8 倍。

有关配置的详细信息，请参阅 参数 章节。

权重和 Query/Group 数据

LightGBM 也支持加权训练，它需要一个额外的 加权数据 。 它需要额外的 query 数据 用于排名任务。

11/3/2016 更新:

1. 现在支持 header（标题）输入
2. 可以指定 label 列，权重列和 query/group id 列。 索引和列都支持
3. 可以指定一个被忽略的列的列表

参数快速查看

参数格式是 `key1=value1 key2=value2 ...` 。 参数可以在配置文件和命令行中。

一些重要的参数如下 :

- `config`, 默认= `""`, type（类型）=string, alias（别名）= `config_file`
  - 配置文件的路径
- `task`, 默认= `train`, type（类型）=enum, options（可选）= `train`, `predict`, `convert_model`
  - `train`, alias（别名）= `training`, 用于训练
  - `predict`, alias（别名）= `prediction`, `test`, 用于预测。
  - `convert_model`, 用于将模型文件转换为 if-else 格式， 在 转换模型参数 中了解更多信息
- `application`, 默认= `regression`, 类型=enum, 可选= `regression`, `regression_l1`, `huber`, `fair`, `poisson`, `quantile`, `quantile_l2`, `binary`, `multiclass`, `multiclassova`, `xentropy`, `xentlambda`, `lambdarank`, 别名= `objective`, `app`
  - 回归 application
    - `regression_l2`, L2 损失, 别名= `regression`, `mean_squared_error`, `mse`
    - `regression_l1`, L1 损失, 别名= `mean_absolute_error`, `mae`
    - `huber`, Huber loss
    - `fair`, Fair loss
    - `poisson`, Poisson regression
    - `quantile`, Quantile regression
    - `quantile_l2`, 与 `quantile` 类似, 但是使用 L2 损失
  - `binary`, 二进制 `log loss` _ 分类 application
  - 多类别分类 application
    - `multiclass`, softmax 目标函数, `num_class` 也应该被设置
    - `multiclassova`, One-vs-All 二元目标函数, `num_class` 也应该被设置
  - 交叉熵 application
    - `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`
    - label 是在 [0, 1] 间隔中的任何东西

    - `lambdarank`, lambdarank application
        - 在 lambdarank 任务中 label 应该是 `int` 类型，而较大的数字表示较高的相关性（例如，0:bad, 1:fair, 2:good, 3:perfect）
        - `label_gain` 可以用来设置 `int` label 的 gain(weight)（增益（权重））
- `boosting`, 默认= `gbdt`, type=enum, 选项= `gbdt`, `rf`, `dart`, `goss`, 别名= `boost`, `boosting_type`
    - `gbdt`, traditional Gradient Boosting Decision Tree（传统梯度提升决策树）
    - `rf`, 随机森林
    - `dart`, Dropouts meet Multiple Additive Regression Trees
    - `goss`, Gradient-based One-Side Sampling（基于梯度的单面采样）
- `data`, 默认= `""`, 类型=string, 别名= `train`, `train_data`
    - 训练数据， LightGBM 将从这个数据训练
- `valid`, 默认= `""`, 类型=multi-string, 别名= `test`, `valid_data`, `test_data`
    - 验证/测试 数据，LightGBM 将输出这些数据的指标
    - 支持多个验证数据，使用 `,` 分开
- `num_iterations`, 默认= `100`, 类型=int, 别名= `num_iteration`, `num_tree`, `num_trees`, `num_round`, `num_rounds`, `num_boost_round`
    - boosting iterations/trees 的数量
- `learning_rate`, 默认= `0.1`, 类型=double, 别名= `shrinkage_rate`
    - shrinkage rate（收敛率）
- `num_leaves`, 默认= `31`, 类型=int, 别名= `num_leaf`
    - 在一棵树中的叶子数量
- `tree_learner`, 默认= `serial`, 类型=enum, 可选= `serial`, `feature`, `data`, `voting`, 别名= `tree`
    - `serial`, 单个 machine tree 学习器
    - `feature`, 别名= `feature_parallel`, feature parallel tree learner（特征并行树学习器）
    - `data`, 别名= `data_parallel`, data parallel tree learner（数据并行树学习器）
    - `voting`, 别名= `voting_parallel`, voting parallel tree learner（投票并行树学习器）
    - 参考 Parallel Learning Guide（并行学习指南） 来了解更多细节
- `num_threads`, 默认= `OpenMP_default`, 类型=int, 别名= `num_thread`, `nthread`
    - LightGBM 的线程数
    - 为了获得最好的速度，将其设置为 **real CPU cores（真实 CPU 内核）** 数量，而不是线程数（大多数 CPU 使用 hyper-threading 来为每个 CPU core 生成 2 个线程）

    - 对于并行学习，不应该使用全部的 CPU cores ，因为这会导致网络性能不佳
- `max_depth`, 默认= `-1`, 类型=int
    - 树模型最大深度的限制。 当 `#data` 很小的时候，这被用来处理 overfit（过拟合）。 树仍然通过 leaf-wise 生长
    - `< 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`
    - 一个叶子中的最小数据量。可以用这个来处理过拟合。
- `min_sum_hessian_in_leaf`, 默认= `1e-3`, 类型=double, 别名= `min_sum_hessian_per_leaf`, `min_sum_hessian`, `min_hessian`, `min_child_weight`
    - 一个叶子节点中最小的 sum hessian 。类似于 `min_data_in_leaf` ，它可以用来处理过拟合。

想要了解全部的参数， 请参阅 Parameters（参数）.

运行 LightGBM

对于 Windows:

```
lightgbm.exe config=your_config_file other_args ...
```

对于 Unix:

```
./lightgbm config=your_config_file other_args ...
```

参数既可以在配置文件中，也可以在命令行中，命令行中的参数优先于配置文件。例如下面的命令行会保留 `num_trees=10` ，并忽略配置文件中的相同参数。

```
./lightgbm config=train.conf num_trees=10
```

示例

- Binary Classification（二元分类）
- Regression（回归）
- Lambdarank
- Parallel Learning（并行学习）

Python 包的相关介绍

该文档给出了有关 LightGBM Python 软件包的基本演练.

其它有用的链接列表

- Python 例子
- Python API
- 参数优化

安装

安装 Python 软件包的依赖, `setuptools`, `wheel`, `numpy` 和 `scipy` 是必须的, `scikit-learn` 对于 sklearn 接口和推荐也是必须的:

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

参考 Python-package 安装指南文件夹.

为了验证是否安装成功, 可以在 Python 中 `import lightgbm` 试试:

```
import lightgbm as lgb
```

数据接口

LightGBM Python 模块能够使用以下几种方式来加载数据:

- libsvm/tsv/csv txt format file（libsvm/tsv/csv 文本文件格式）
- Numpy 2D array, pandas object（Numpy 2维数组, pandas 对象）
- LightGBM binary file（LightGBM 二进制文件）

加载后的数据存在 `Dataset` 对象中.

要加载 ligsvm 文本文件或 LightGBM 二进制文件到 Dataset 中:

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

要加载 numpy 数组到 Dataset 中:

```
data = np.random.rand(500, 10)  # 500 个样本, 每一个包含 10 个特征
label = np.random.randint(2, size=500)  # 二元目标变量,  0 和 1
train_data = lgb.Dataset(data, label=label)
```

要加载 scpiy.sparse.csr_matrix 数组到 Dataset 中:

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

保存 Dataset 到 LightGBM 二进制文件将会使得加载更快速:

```
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')
```

or

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

在 LightGBM 中, 验证数据应该与训练数据一致（格式一致）.

指定 feature names（特征名称）和 categorical features（分类特征）:

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

LightGBM 可以直接使用 categorical features（分类特征）作为 input（输入）. 它不需要被转换成 one-hot coding（独热编码）, 并且它比 one-hot coding（独热编码）更快（约快上 8 倍）

注意: 在你构造 `Dataset` 之前, 你应该将分类特征转换为 `int` 类型的值.

当需要时可以设置权重:

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

或者

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

并且你也可以使用 `Dataset.set_init_score()` 来初始化 score（分数）, 以及使用 `Dataset.set_group()` ； 来设置 group/query 数据以用于 ranking（排序）任务.

内存的高使用:

LightGBM 中的 `Dataset` 对象由于只需要保存 discrete bins（离散的数据块）, 因此它具有很好的内存效率. 然而, Numpy/Array/Pandas 对象的内存开销较大. 如果你关心你的内存消耗. 您可以根据以下方式来节省内存:

1. 在构造 `Dataset` 时设置 `free_raw_data=True` （默认为 `True`）
2. 在 `Dataset` 被构造完之后手动设置 `raw_data=None`
3. 调用 `gc`

设置参数

LightGBM 可以使用一个 pairs 的 list 或一个字典来设置 参数. 例如:

- Booster（提升器）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

训练一个模型时, 需要一个 parameter list（参数列表）和 data set（数据集）:

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

在训练完成后, 可以使用如下方式来存储模型:

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

训练后的模型也可以转存为 JSON 的格式:

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

以保存模型也可以使用如下的方式来加载.

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

交叉验证

使用 5-折 方式的交叉验证来进行训练（4 个训练集, 1 个测试集）：

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

提前停止

如果您有一个验证集, 你可以使用提前停止找到最佳数量的 boosting rounds（梯度次数）. 提前停止需要在 `valid_sets` 中至少有一个集合. 如果有多个，它们都会被使用:

```
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)
```

该模型将开始训练, 直到验证得分停止提高为止. 验证错误需要至少每个 <cite>early_stopping_rounds</cite> 减少以继续训练.

如果提前停止, 模型将有 1 个额外的字段: <cite>bst.best_iteration</cite>. 请注意 <cite>train()</cite> 将从最后一次迭代中返回一个模型, 而不是最好的一个.

This works with both metrics to minimize (L2, log loss, etc.) and to maximize (NDCG, AUC). Note that if you specify more than one evaluation metric, all of them will be used for early stopping.

这与两个度量标准一起使用以达到最小化（L2, 对数损失, 等等）和最大化（NDCG, AUC）. 请注意, 如果您指定多个评估指标, 则它们都会用于提前停止.

预测

已经训练或加载的模型都可以对数据集进行预测:

```
# 7 个样本, 每一个包含 10 个特征
data = np.random.rand(7, 10)
ypred = bst.predict(data)
```

如果在训练过程中启用了提前停止, 可以用 <cite>bst.best_iteration</cite> 从最佳迭代中获得预测结果:

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

特性

这篇文档是对 LightGBM 的特点和其中用到的算法的简短介绍

本页不包含详细的算法，如果你对这些算法感兴趣可以查阅引用的论文或者源代码

速度和内存使用的优化

许多提升工具对于决策树的学习使用基于 pre-sorted 的算法 [1, 2] (例如，在xgboost中默认的算法) ，这是一个简单的解决方案，但是不易于优化。

LightGBM 利用基于 histogram 的算法 [3, 4, 5]，通过将连续特征（属性）值分段为 discrete bins 来加快训练的速度并减少内存的使用。 如下的是基于 histogram 算法的优点：

- **减少分割增益的计算量**
  - Pre-sorted 算法需要 `O(#data)` 次的计算
  - Histogram 算法只需要计算 `O(#bins)` 次, 并且 `#bins` 远少于 `#data`
    - 这个仍然需要 `O(#data)` 次来构建直方图, 而这仅仅包含总结操作
- **通过直方图的相减来进行进一步的加速**
  - 在二叉树中可以通过利用叶节点的父节点和相邻节点的直方图的相减来获得该叶节点的直方图
  - 所以仅仅需要为一个叶节点建立直方图 (其 `#data` 小于它的相邻节点)就可以通过直方图的相减来获得相邻节点的直方图，而这花费的代价(`O(#bins)`)很小。
- **减少内存的使用**
  - 可以将连续的值替换为 discrete bins。 如果 `#bins` 较小, 可以利用较小的数据类型来存储训练数据, 如 uint8_t。
  - 无需为 pre-sorting 特征值存储额外的信息
- **减少并行学习的通信代价**

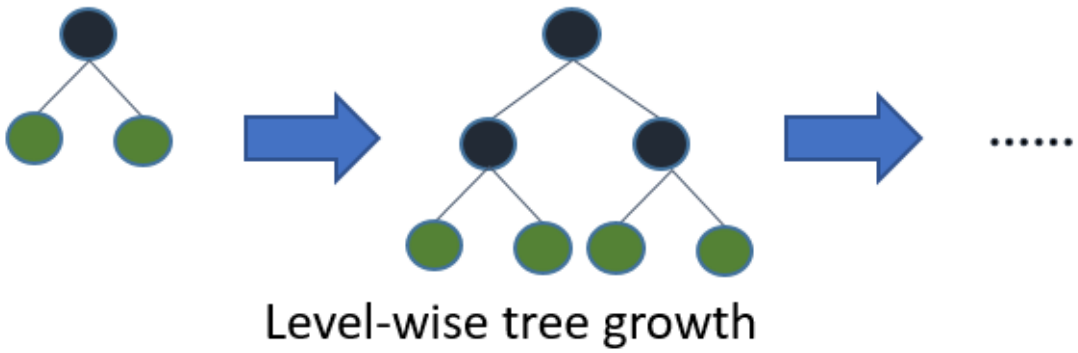
稀疏优化

- 对于稀疏的特征仅仅需要 `O(2 * #non_zero_data)` 来建立直方图

准确率的优化

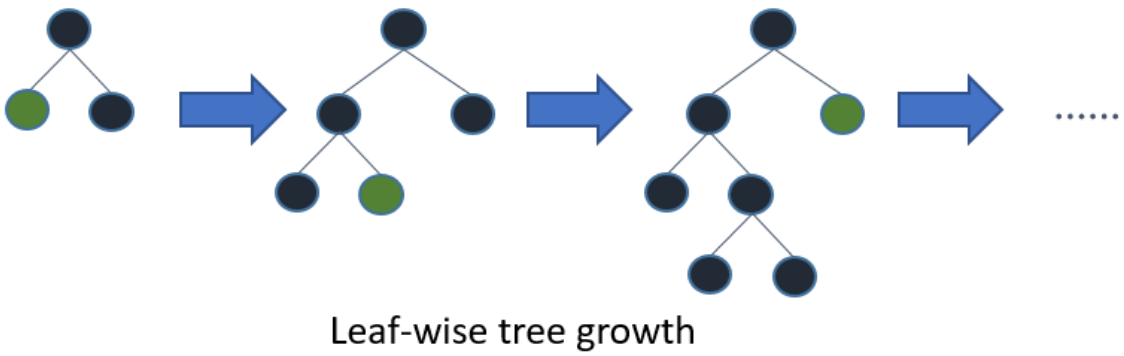
Leaf-wise (Best-first) 的决策树生长策略

大部分决策树的学习算法通过 level(depth)-wise 策略生长树，如下图一样：



LightGBM 通过 leaf-wise (best-first)[6] 策略来生长树。它将选取具有最大 delta loss 的叶节点来生长。 当生长相同的 `#leaf`，leaf-wise 算法可以比 level-wise 算法减少更多的损失。

当 `#data` 较小的时候，leaf-wise 可能会造成过拟合。 所以，LightGBM 可以利用额外的参数 `max_depth` 来限制树的深度并避免过拟合（树的生长仍然通过 leaf-wise 策略）。



类别特征值的最优分割

我们通常将类别特征转化为 one-hot coding。 然而，对于学习树来说这不是个好的解决方案。原因是，对于一个基数较大的类别特征，学习树会生长的非常不平衡，并且需要非常深的深度才能来达到较好的准确率。

事实上，最好的解决方案是将类别特征划分为两个子集，总共有 `2^(k-1) - 1` 种可能的划分 但是对于回归树 [7] 有个有效的解决方案。为了寻找最优的划分需要大约 `k * log(k)` .

基本的思想是根据训练目标的相关性对类别进行重排序。 更具体的说，根据累加值（`sum_gradient / sum_hessian`）重新对（类别特征的）直方图进行排序，然后在排好序的直方图中寻找最好的分割点。

网络通信的优化

LightGBM 中的并行学习，仅仅需要使用一些聚合通信算法，例如 “All reduce”, “All gather” 和 “Reduce scatter”. LightGBM 实现了 state-of-art 算法 [8] . 这些聚合通信算法可以提供比点对点通信更好的性能。

并行学习的优化

LightGBM 提供以下并行学习优化算法：

特征并行

传统算法

传统的特征并行算法旨在于在并行化决策树中的“ `Find Best Split` .主要流程如下:

1. 垂直划分数据（不同的机器有不同的特征集）
2. 在本地特征集寻找最佳划分点 {特征, 阈值}
3. 本地进行各个划分的通信整合并得到最佳划分
4. 以最佳划分方法对数据进行划分，并将数据划分结果传递给其他线程
5. 其他线程对接受到的数据进一步划分

传统的特征并行方法主要不足:

- 存在计算上的局限，传统特征并行无法加速 “split”（时间复杂度为 “O（#data）”）。 因此，当数据量很大的时候，难以加速。
- 需要对划分的结果进行通信整合，其额外的时间复杂度约为 “O（#data/8）”（一个数据一个字节）

LightGBM 中的特征并行

既然在数据量很大时，传统数据并行方法无法有效地加速，我们做了一些改变：不再垂直划分数据，即每个线程都持有全部数据。 因此，LighetGBM中没有数据划分结果之间通信的开销，各个线程都知道如何划分数据。 而且，“#data” 不会变得更大，所以，在使每天机器都持有全部数据是合理的。

LightGBM 中特征并行的流程如下：

1. 每个线程都在本地数据集上寻找最佳划分点｛特征， 阈值｝
2. 本地进行各个划分的通信整合并得到最佳划分
3. 执行最佳划分

然而，该特征并行算法在数据量很大时仍然存在计算上的局限。因此，建议在数据量很大时使用数据并行。

数据并行

传统算法

数据并行旨在于并行化整个决策学习过程。数据并行的主要流程如下：

1. 水平划分数据
2. 线程以本地数据构建本地直方图
3. 将本地直方图整合成全局整合图
4. 在全局直方图中寻找最佳划分，然后执行此划分

传统数据划分的不足：

- 高通讯开销。 如果使用点对点的通讯算法，一个机器的通讯开销大约为 “O(#machine * #feature * #bin)” 。 如果使用集成的通讯算法（例如， “All Reduce”等），通讯开销大约为 “O(2 * #feature * #bin)”[8] 。

LightGBM中的数据并行

LightGBM 中采用以下方法较少数据并行中的通讯开销：

1. 不同于“整合所有本地直方图以形成全局直方图”的方式，LightGBM 使用分散规约(Reduce scatter)的方式对不同线程的不同特征（不重叠的）进行整合。 然后线程从本地整合直方图中寻找最佳划分并同步到全局的最佳划分中。
2. 如上所述。LightGBM 通过直方图做差法加速训练。 基于此，我们可以进行单叶子的直方图通讯，并且在相邻直方图上使用做差法。

通过上述方法，LightGBM 将数据并行中的通讯开销减少到 “O(0.5 * #feature * #bin)”。

投票并行

投票并行未来将致力于将“数据并行”中的通讯开销减少至常数级别。 其将会通过两阶段的投票过程较少特征直方图的通讯开销 [9] .

GPU 支持

感谢 “@huanzhang12 <https://github.com/huanzhang12>” 对此项特性的贡献。相关细节请阅读 [10] 。

- GPU 安装
- GPU 训练

应用和度量

支持以下应用:

- 回归，目标函数为 L2 loss
- 二分类， 目标函数为 logloss（对数损失）
- 多分类
- lambdarank, 目标函数为基于 NDCG 的 lambdarank

支持的度量

- L1 loss
- L2 loss
- Log loss
- Classification error rate
- AUC
- NDCG
- Multi class log loss
- Multi class error rate

获取更多详情，请至 Parameters。

其他特性

- Limit `max_depth` of tree while grows tree leaf-wise
- DART
- L1/L2 regularization
- Bagging
- Column(feature) sub-sample

- Continued train with input GBDT model
- Continued train with the input score file
- Weighted training
- Validation metric output during training
- Multi validation data
- Multi metrics
- Early stopping (both training and prediction)
- Prediction for leaf index

获取更多详情，请参阅 参数。

References

[1] Mehta, Manish, Rakesh Agrawal, and Jorma Rissanen. “SLIQ: A fast scalable classifier for data mining.” International Conference on Extending Database Technology. Springer Berlin Heidelberg, 1996.

[2] Shafer, John, Rakesh Agrawal, and Manish Mehta. “SPRINT: A scalable parallel classifier for data mining.” Proc. 1996 Int. Conf. Very Large Data Bases. 1996.

[3] Ranka, Sanjay, and V. Singh. “CLOUDS: A decision tree classifier for large datasets.” Proceedings of the 4th Knowledge Discovery and Data Mining Conference. 1998.

[4] Machado, F. P. “Communication and memory efficient parallel decision tree construction.” (2003).

[5] Li, Ping, Qiang Wu, and Christopher J. Burges. “Mcrank: Learning to rank using multiple classification and gradient boosting.” Advances in neural information processing systems. 2007.

[6] Shi, Haijian. “Best-first decision tree learning.” Diss. The University of Waikato, 2007.

[7] Walter D. Fisher. “On Grouping for Maximum Homogeneity.” Journal of the American Statistical Association. Vol. 53, No. 284 (Dec., 1958), pp. 789-798.

[8] Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. “Optimization of collective communication operations in MPICH.” International Journal of High Performance Computing Applications 19.1 (2005): 49-66.

[9] Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, Tieyan Liu. “A Communication-Efficient Parallel Algorithm for Decision Tree.” Advances in Neural Information Processing Systems 29 (NIPS 2016).

[10] Huan Zhang, Si Si and Cho-Jui Hsieh. “GPU Acceleration for Large-scale Tree Boosting.” arXiv:1706.08359, 2017.

实验

对比实验

详细的实验脚本和输出日志部分请参考 repo.

数据集

我们使用4个数据集进行对比实验，有关数据的细节在下表列出：

数据集	任务	链接	训练集	特征	注释
Higgs	二分类	link	10,500,000	28	使用余下50万个样本作为测试集
Yahoo LTR	机器学习排序	link	473,134	700	set1.train为训练集，set1.test为测试集
MS LTR	机器学习排序	link	2,270,296	137	{S1,S2,S3}为训练集，{S5}为测试集
Expo	二分类	link	11,000,000	700	使用余下100W个样本作为测试集
Allstate	二分类	link	13,184,290	4228	使用余下100W个样本作为测试集

硬件环境

我们使用一台Linux服务器作为实验平台，具体配置如下：

OS	CPU	Memory
Ubuntu 14.04 LTS	2 * E5-2670 v3	DDR4 2133Mhz, 256GB

底层

我们使用 xgboost 作为底层算法。

并且 xgboost 和 LightGBM 都基于 OpenMP 构建。

设置

我们为该实验建立了3个设置， 这些设置的参数如下:

1. xgboost:

```
eta = 0.1
max_depth = 8
num_round = 500
nthread = 16
tree_method = exact
min_child_weight = 100
```

2. xgboost_hist (使用直方图算法):

```
eta = 0.1
num_round = 500
nthread = 16
tree_method = approx
min_child_weight = 100
tree_method = hist
grow_policy = lossguide
max_depth = 0
max_leaves = 255
```

3. LightGBM:

```
learning_rate = 0.1
num_leaves = 255
num_trees = 500
num_threads = 16
min_data_in_leaf = 0
min_sum_hessian_in_leaf = 100
```

xgboost 通过 `max_depth` 对建树进行深度限制与模型复杂度控制 。

LightGBM 通过 `num_leaves` 执行带深度限制的 leaf-wise 叶子生长策略与模型复杂度控制。

因此我们无法设置完全相同的模型进行比较。为了相对权衡， 我们在xgboost中设置 `max_depth=8` 以使叶子数量达到最大数量 255 与 LightGBM 中设置 `num_leves=255` 进行比较。

其他参数皆为默认值

结论

效率

为了比较效率, 我们只运行没有任何测试或者度量输出的训练进程，并且我们不计算 IO 的时间。

如下是耗时的对比表格：

Data	xgboost	xgboost_hist	LightGBM
Higgs	3794.34 s	551.898 s	**238.505513 s**
Yahoo LTR	674.322 s	265.302 s	**150.18644 s**
MS LTR	1251.27 s	385.201 s	**215.320316 s**
Expo	1607.35 s	588.253 s	**138.504179 s**
Allstate	2867.22 s	1355.71 s	**348.084475 s**

我们发现在所有数据集上 LightGBM 都比 xgboost 快。

准确率

为了比较准确率, 我们使用数据集测试集部分的准确率进行公平比较。

Data	Metric	xgboost	xgboost_hist	LightGBM
Higgs	AUC	0.839593	0.845605	0.845154
Yahoo LTR	NDCG₁	0.719748	0.720223	0.732466
NDCG₃	0.717813	0.721519	0.738048	
NDCG₅	0.737849	0.739904	0.756548	
NDCG₁₀	0.78089	0.783013	0.796818	
MS LTR	NDCG₁	0.483956	0.488649	0.524255

Data	Metric	xgboost	xgboost_hist	LightGBM
NDCG₃	0.467951	0.473184	0.505327	
NDCG₅	0.472476	0.477438	0.510007	
NDCG₁₀	0.492429	0.496967	0.527371	
Expo	AUC	0.756713	0.777777	0.777543
Allstate	AUC	0.607201	0.609042	0.609167

内存消耗

我们在运行训练任务时监视 RES，并在 LightGBM 中设置 `two_round=true` （将增加数据载入时间，但会减少峰值内存使用量，不影响训练速度和准确性）以减少峰值内存使用量。

Data	xgboost	xgboost_hist	LightGBM
Higgs	4.853GB	3.784GB	**0.868GB**
Yahoo LTR	1.907GB	1.468GB	**0.831GB**
MS LTR	5.469GB	3.654GB	**0.886GB**
Expo	1.553GB	1.393GB	**0.543GB**
Allstate	6.237GB	4.990GB	**1.027GB**

并行测试

数据集

我们使用 `terabyte click log` 数据集进行并行测试，详细信息如下表：

数据	任务	链接	数据集	特征
Criteo	二分类	link	1,700,000,000	67

该数据集包含了 24 天点击记录，其中有 13 个整数特征与 26 个类别特征。

我们统计了该数据集 26 个类别前十天的点击率和计数，使用接下来十天的数据作为训练集并且该训练集中类别已与点击率和计数相对应。

处理后的训练集共有 17 亿条数据和 67 个特征。

环境

我们使用了 16 台 Windows 服务器作为实验平台，详细信息如下表：

OS	CPU	Memory	Network Adapter
Windows Server 2012	2 * E5-2670 v2	DDR3 1600Mhz, 256GB	Mellanox ConnectX-3, 54Gbps, RDMA support

设置：

```
learning_rate = 0.1
num_leaves = 255
num_trees = 100
num_thread = 16
tree_learner = data
```

我们在此使用并行数据，因为该数据集数据量大但是特征少。

其他参数皆为默认值

结论

#Machine	Time per Tree	Memory Usage(per Machine)
1	627.8 s	176GB
2	311 s	87GB
4	156 s	43GB
8	80 s	22GB
16	42 s	11GB

从结果看，我们发现 LightGBM 在分布式学习中可以做到线性加速。

GPU 实验

参考 GPU 性能.

参数

这个页面包含了 LightGBM 的所有参数.

一些有用的链接列表

- Python API
- Parameters Tuning

外部链接

- Laurae++ Interactive Documentation

更新于 08/04/2017

以下参数的default已经修改:

- `min_data_in_leaf` = 100 => 20
- `min_sum_hessian_in_leaf` = 10 => 1e-3
- `num_leaves` = 127 => 31
- `num_iterations` = 10 => 100

参数格式

参数的格式为 `key1=value1 key2=value2 ...`. 并且, 在配置文件和命令行中均可以设置参数. 使用命令行设置参数时, 在 `=` 前后都不应该有空格. 使用配置文件设置参数时, 一行只能包含一个参数. 你可以使用 `#` 进行注释.

如果一个参数在命令行和配置文件中均出现了, LightGBM 将会使用命令行中的该参数.

核心参数

- `config`, default= `""`, type=string, alias= `config_file`
  - 配置文件的路径
- `task`, default= `train`, type=enum, options= `train`, `predict`, `convert_model`
  - `train`, alias= `training`, for training
  - `predict`, alias= `prediction`, `test`, for prediction.
  - `convert_model`, 要将模型文件转换成 if-else 格式, 可以查看这个链接获取更多信息 Convert model parameters

- `objective`, default=`regression`, type=enum, options=`regression`, `regression_l1`, `huber`, `fair`, `poisson`, `quantile`, `quantile_l2`, `binary`, `multiclass`, `multiclassova`, `xentropy`, `xentlambda`, `lambdarank`, alias=`objective`, `app`, `application`
  - regression application
    - `regression_l2`, L2 loss, alias=`regression`, `mean_squared_error`, `mse`
    - `regression_l1`, L1 loss, alias=`mean_absolute_error`, `mae`
    - `huber`, Huber loss
    - `fair`, Fair loss
    - `poisson`, Poisson regression
    - `quantile`, Quantile regression
    - `quantile_l2`, 类似于 `quantile`, 但是使用了 L2 loss
  - `binary`, binary log loss classification application
  - multi-class classification application
    - `multiclass`, softmax 目标函数, 应该设置好 `num_class`
    - `multiclassova`, One-vs-All 二分类目标函数, 应该设置好 `num_class`
  - cross-entropy application
    - `xentropy`, 目标函数为 cross-entropy (同时有可选择的线性权重), alias=`cross_entropy`
    - `xentlambda`, 替代参数化的 cross-entropy, alias=`cross_entropy_lambda`
    - 标签是 [0, 1] 间隔内的任意值
  - `lambdarank`, lambdarank application
    - 在 lambdarank 任务中标签应该为 `int` type, 数值越大代表相关性越高 (e.g. 0:bad, 1:fair, 2:good, 3:perfect)
    - `label_gain` 可以被用来设置 `int` 标签的增益 (权重)
- `boosting`, default=`gbdt`, type=enum, options=`gbdt`, `rf`, `dart`, `goss`, alias=`boost`, `boosting_type`
  - `gbdt`, 传统的梯度提升决策树
  - `rf`, Random Forest (随机森林)
  - `dart`, Dropouts meet Multiple Additive Regression Trees
  - `goss`, Gradient-based One-Side Sampling (基于梯度的单侧采样)
- `data`, default=`""`, type=string, alias=`train`, `train_data`
  - 训练数据, LightGBM 将会使用这个数据进行训练

- `valid`, default= `""`, type=multi-string, alias= `test`, `valid_data`, `test_data`
  - 验证/测试 数据, LightGBM 将输出这些数据的度量
  - 支持多验证数据集, 以 `,` 分割
- `num_iterations`, default= `100`, type=int, alias= `num_iteration`, `num_tree`, `num_trees`, `num_round`, `num_rounds`, `num_boost_round`
  - boosting 的迭代次数
  - **Note**: 对于 Python/R 包, **这个参数是被忽略的**, 使用 `train` and `cv` 的输入参数 `num_boost_round` (Python) or `nrounds` (R) 来代替
  - **Note**: 在内部, LightGBM 对于 `multiclass` 问题设置 `num_class * num_iterations` 棵树
- `learning_rate`, default= `0.1`, type=double, alias= `shrinkage_rate`
  - shrinkage rate (收缩率)
  - 在 `dart` 中, 它还影响了 dropped trees 的归一化权重
- `num_leaves`, default= `31`, type=int, alias= `num_leaf`
  - 一棵树上的叶子数
- `tree_learner`, default= `serial`, type=enum, options= `serial`, `feature`, `data`, `voting`, alias= `tree`
  - `serial`, 单台机器的 tree learner
  - `feature`, alias= `feature_parallel`, 特征并行的 tree learner
  - `data`, alias= `data_parallel`, 数据并行的 tree learner
  - `voting`, alias= `voting_parallel`, 投票并行的 tree learner
  - 请阅读 并行学习指南 来了解更多细节
- `num_threads`, default= `OpenMP_default`, type=int, alias= `num_thread`, `nthread`
  - LightGBM 的线程数
  - 为了更快的速度, 将此设置为真正的 CPU 内核数, 而不是线程的数量 (大多数 CPU 使用超线程来使每个 CPU 内核生成 2 个线程)
  - 当你的数据集小的时候不要将它设置的过大 (比如, 当数据集有 10,000 行时不要使用 64 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未被充分利用的内核. **这是正常的**
  - 对于并行学习, 不应该使用全部的 CPU 内核, 因为这会导致网络性能不佳
- `device`, default= `cpu`, options= `cpu`, `gpu`
  - 为树学习选择设备, 你可以使用 GPU 来获得更快的学习速度

    - **Note**: 建议使用较小的 `max_bin` (e.g. 63) 来获得更快的速度
    - **Note**: 为了加快学习速度, GPU 默认使用32位浮点数来求和. 你可以设置 `gpu_use_dp=true` 来启用64位浮点数, 但是它会使训练速度降低
    - **Note**: 请参考 安装指南 来构建 GPU 版本

用于控制模型学习过程的参数

- `max_depth`, default= `-1`, type=int
    - 限制树模型的最大深度. 这可以在 `#data` 小的情况下防止过拟合. 树仍然可以通过 leaf-wise 生长.
    - `< 0` 意味着没有限制.
- `min_data_in_leaf`, default= `20`, type=int, alias= `min_data_per_leaf` , `min_data`, `min_child_samples`
    - 一个叶子上数据的最小数量. 可以用来处理过拟合.
- `min_sum_hessian_in_leaf`, default= `1e-3`, type=double, alias= `min_sum_hessian_per_leaf`, `min_sum_hessian`, `min_hessian`, `min_child_weight`
    - 一个叶子上的最小 hessian 和. 类似于 `min_data_in_leaf`, 可以用来处理过拟合.
- `feature_fraction`, default= `1.0`, type=double, `0.0 < feature_fraction < 1.0`, alias= `sub_feature`, `colsample_bytree`
    - 如果 `feature_fraction` 小于 `1.0`, LightGBM 将会在每次迭代中随机选择部分特征. 例如, 如果设置为 `0.8`, 将会在每棵树训练之前选择 80% 的特征
    - 可以用来加速训练
    - 可以用来处理过拟合
- `feature_fraction_seed`, default= `2`, type=int
    - `feature_fraction` 的随机数种子
- `bagging_fraction`, default= `1.0`, type=double, `0.0 < bagging_fraction < 1.0`, alias= `sub_row`, `subsample`
    - 类似于 `feature_fraction`, 但是它将在不进行重采样的情况下随机选择部分数据
    - 可以用来加速训练
    - 可以用来处理过拟合
    - **Note**: 为了启用 bagging, `bagging_freq` 应该设置为非零值
- `bagging_freq`, default= `0`, type=int, alias= `subsample_freq`
    - bagging 的频率, `0` 意味着禁用 bagging. `k` 意味着每 `k` 次迭代执行bagging
    - **Note**: 为了启用 bagging, `bagging_fraction` 设置适当

- `bagging_seed` , default= `3` , type=int, alias= `bagging_fraction_seed`
  - bagging 随机数种子
- `early_stopping_round` , default= `0` , type=int, alias= `early_stopping_rounds` , `early_stopping`
  - 如果一个验证集的度量在 `early_stopping_round` 循环中没有提升, 将停止训练
- `lambda_l1` , default= `0` , type=double, alias= `reg_alpha`
  - L1 正则
- `lambda_l2` , default= `0` , type=double, alias= `reg_lambda`
  - L2 正则
- `min_split_gain` , default= `0` , type=double, alias= `min_gain_to_split`
  - 执行切分的最小增益
- `drop_rate` , default= `0.1` , type=double
  - 仅仅在 `dart` 时使用
- `skip_drop` , default= `0.5` , type=double
  - 仅仅在 `dart` 时使用, 跳过 drop 的概率
- `max_drop` , default= `50` , type=int
  - 仅仅在 `dart` 时使用, 一次迭代中删除树的最大数量
  - `<=0` 意味着没有限制
- `uniform_drop` , default= `false` , type=bool
  - 仅仅在 `dart` 时使用, 如果想要均匀的删除, 将它设置为 `true`
- `xgboost_dart_mode` , default= `false` , type=bool
  - 仅仅在 `dart` 时使用, 如果想要使用 xgboost dart 模式, 将它设置为 `true`
- `drop_seed` , default= `4` , type=int
  - 仅仅在 `dart` 时使用, 选择 dropping models 的随机数种子
- `top_rate` , default= `0.2` , type=double
  - 仅仅在 `goss` 时使用, 大梯度数据的保留比例
- `other_rate` , default= `0.1` , type=int
  - 仅仅在 `goss` 时使用, 小梯度数据的保留比例
- `min_data_per_group` , default= `100` , type=int
  - 每个分类组的最小数据量
- `max_cat_threshold` , default= `32` , type=int
  - 用于分类特征

    - 限制分类特征的最大阈值
- `cat_smooth`, default=`10`, type=double
    - 用于分类特征
    - 这可以降低噪声在分类特征中的影响, 尤其是对数据很少的类别
- `cat_l2`, default=`10`, type=double
    - 分类切分中的 L2 正则
- `max_cat_to_onehot`, default=`4`, type=int
    - 当一个特征的类别数小于或等于 `max_cat_to_onehot` 时, one-vs-other 切分算法将会被使用
- `top_k`, default=`20`, type=int, alias=`topk`
    - 被使用在 Voting parallel 中
    - 将它设置为更大的值可以获得更精确的结果, 但会减慢训练速度

IO 参数

- `max_bin`, default=`255`, type=int
    - 工具箱的最大数特征值决定了容量 工具箱的最小数特征值可能会降低训练的准确性, 但是可能会增加一些一般的影响（处理过度学习）
    - LightGBM 将根据 `max_bin` 自动压缩内存。 例如, 如果 maxbin=255, 那么 LightGBM 将使用 uint8t 的特性值
- `max_bin`, default=`255`, type=int
- `min_data_in_bin`, default=`3`, type=int - 单个数据箱的最小数, 使用此方法避免 one-data-one-bin（可能会过度学习）
- `data_r和om_seed`, default=`1`, type=int
    - 并行学习数据分隔中的随机种子 (不包括并行功能)
- `output_model`, default=`LightGBM_model.txt`, type=string, alias=`model_output`, `model_out`
    - 培训中输出的模型文件名
- `input_model`, default=`""`, type=string, alias=`model_input`, `model_in`
    - 输入模型的文件名
    - 对于 `prediction` 任务, 该模型将用于预测数据
    - 对于 `train` 任务, 培训将从该模型继续

- `output_result`, default= `LightGBM_predict_result.txt`, type=string, alias= `predict_result`, `prediction_result`
  - `prediction` 任务的预测结果文件名
- `model_format`, default= `text`, type=multi-enum, 可选项= `text`, `proto`
  - 保存和加载模型的格式
  - `text`, 使用文本字符串
  - `proto`, 使用协议缓冲二进制格式
  - 您可以通过使用逗号来进行多种格式的保存, 例如 `text,proto`. 在这种情况下, `model_format` 将作为后缀添加 `output_model`
  - **Note**: 不支持多种格式的加载
  - **Note**: 要使用这个参数, 您需要使用 build 版本 <./Installation-Guide.rst#protobuf-support>`__
- `pre_partition`, default= `false`, type=bool, alias= `is_pre_partition`
  - 用于并行学习(不包括功能并行)
  - `true` 如果训练数据 pre-partitioned, 不同的机器使用不同的分区
- `is_sparse`, default= `true`, type=bool, alias= `is_enable_sparse`, `enable_sparse`
  - 用于 enable/disable 稀疏优化. 设置 `false` 就禁用稀疏优化
- `two_round`, default= `false`, type=bool, alias= `two_round_loading`, `use_two_round_loading`
  - 默认情况下, LightGBM 将把数据文件映射到内存, 并从内存加载特性。 这将提供更快的数据加载速度。但当数据文件很大时, 内存可能会耗尽
  - 如果数据文件太大, 不能放在内存中, 就把它设置为 `true`
- `save_binary`, default= `false`, type=bool, alias= `is_save_binary`, `is_save_binary_file`
  - 如果设置为 `true` LightGBM 则将数据集(包括验证数据)保存到二进制文件中。 可以加快数据加载速度。
- `verbosity`, default= `1`, type=int, alias= `verbose`
  - `<0` = 致命的, `=0` = 错误 (警告), `>0` = 信息
- `header`, default= `false`, type=bool, alias= `has_header`
  - 如果输入数据有标识头, 则在此处设置 `true`
- `label`, default= `""`, type=string, alias= `label_column`
  - 指定标签列
  - 用于索引的数字, e.g. `label=0` 意味着 column_0 是标签列
  - 为列名添加前缀 `name:`, e.g. `label=name:is_click`

- `weight`, default=`""`, type=string, alias=`weight_column`
  - 列的指定
  - 用于索引的数字, e.g. `weight=0` 表示 column_0 是权重点
  - 为列名添加前缀 `name:`, e.g. `weight=name:weight`
  - **Note**: 索引从 `0` 开始. 当传递 type 为索引时, 它不计算标签列, 例如当标签为 0 时, 权重为列 1, 正确的参数是权重值为 0
- `query`, default=`""`, type=string, alias=`query_column`, `group`, `group_column`
  - 指定 query/group ID 列
  - 用数字做索引, e.g. `query=0` 意味着 column_0 是这个查询的 Id
  - 为列名添加前缀 `name:` , e.g. `query=name:query_id`
  - **Note**: 数据应按照 query_id. 索引从 `0` 开始. 当传递 type 为索引时, 它不计算标签列, 例如当标签为列 0, 查询 id 为列 1 时, 正确的参数是查询 =0
- `ignore_column`, default=`""`, type=string, alias=`ignore_feature`, `blacklist`
  - 在培训中指定一些忽略的列
  - 用数字做索引, e.g. `ignore_column=0,1,2` 意味着 column_0, column_1 和 column_2 将被忽略
  - 为列名添加前缀 `name:` , e.g. `ignore_column=name:c1,c2,c3` 意味着 c1, c2 和 c3 将被忽略
  - **Note**: 只在从文件直接加载数据的情况下工作
  - **Note**: 索引从 `0` 开始. 它不包括标签栏
- `categorical_feature`, default=`""`, type=string, alias=`categorical_column`, `cat_feature`, `cat_column`
  - 指定分类特征
  - 用数字做索引, e.g. `categorical_feature=0,1,2` 意味着 column_0, column_1 和 column_2 是分类特征
  - 为列名添加前缀 `name:`, e.g. `categorical_feature=name:c1,c2,c3` 意味着 c1, c2 和 c3 是分类特征
  - **Note**: 只支持分类与 `int` type. 索引从 `0` 开始. 同时它不包括标签栏
  - **Note**: 负值的值将被视为 **missing values**
- `predict_raw_score`, default=`false`, type=bool, alias=`raw_score`, `is_predict_raw_score`
  - 只用于 `prediction` 任务
  - 设置为 `true` 只预测原始分数

    - 设置为 `false` 只预测分数
- `predict_leaf_index`, default= `false`, type=bool, alias= `leaf_index`, `is_predict_leaf_index`
    - 只用于 `prediction` 任务
    - 设置为 `true` to predict with leaf index of all trees
- `predict_contrib`, default= `false`, type=bool, alias= `contrib`, `is_predict_contrib`
    - 只用于 `prediction` 任务
    - 设置为 `true` 预估 SHAP values, 这代表了每个特征对每个预测的贡献. 生成的特征+1的值, 其中最后一个值是模型输出的预期值, 而不是训练数据
- `bin_construct_sample_cnt`, default= `200000`, type=int, alias= `subsample_for_bin`
    - 用来构建直方图的数据的数量
    - 在设置更大的数据时, 会提供更好的培训效果, 但会增加数据加载时间
    - 如果数据非常稀疏, 则将其设置为更大的值
- `num_iteration_predict`, default= `-1`, type=int
    - 只用于 `prediction` 任务
    - 用于指定在预测中使用多少经过培训的迭代
    - `<= 0` 意味着没有限制
- `pred_early_stop`, default= `false`, type=bool
    - 如果 `true` 将使用提前停止来加速预测。可能影响精度
- `pred_early_stop_freq`, default= `10`, type=int
    - 检查早期early-stopping的频率
- `pred_early_stop_margin`, default= `10.0`, type=double
    - t提前early-stopping的边际阈值
- `use_missing`, default= `true`, type=bool
    - 设置为 `false` 禁用丢失值的特殊句柄
- `zero_as_missing`, default= `false`, type=bool
    - 设置为 `true` 将所有的0都视为缺失的值 (包括 libsvm/sparse 矩阵中未显示的值)
    - 设置为 `false` 使用 `na` 代表缺失值
- `init_score_file`, default= `""`, type=string
    - 训练初始分数文件的路径, `""` 将使用 `train_data_file` + `.init` (如果存在)
- `valid_init_score_file`, default= `""`, type=multi-string
    - 验证初始分数文件的路径, `""` 将使用 `valid_data_file` + `.init` (如果存在)

    - 通过 `,` 对multi-validation进行分离

目标参数

- `sigmoid`, default= `1.0`, type=double
    - sigmoid 函数的参数. 将用于 `binary` 分类 和 `lambdarank`
- `alpha`, default= `0.9`, type=double
    - Huber loss 和 Quantile regression 的参数. 将用于 `regression` 任务
- `fair_c`, default= `1.0`, type=double
    - Fair loss 的参数. 将用于 `regression` 任务
- `gaussian_eta`, default= `1.0`, type=double
    - 控制高斯函数的宽度的参数. 将用于 `regression_l1` 和 `huber` losses
- `poisson_max_delta_step`, default= `0.7`, type=double
    - Poisson regression 的参数用于维护优化
- `scale_pos_weight`, default= `1.0`, type=double
    - 正值的权重 `binary` 分类 任务
- `boost_from_average`, default= `true`, type=bool
    - 只用于 `regression` 任务
    - 将初始分数调整为更快收敛速度的平均值
- `is_unbalance`, default= `false`, type=bool, alias= `unbalanced_sets`
    - 用于 `binary` 分类
    - 如果培训数据不平衡 设置为 `true`
- `max_position`, default= `20`, type=int
    - 用于 `lambdarank`
    - 将在这个 NDCG 位置优化
- `label_gain`, default= `0,1,3,7,15,31,63,...`, type=multi-double
    - 用于 `lambdarank`
    - 有关获得标签. 列如, 如果使用默认标签增益 这个 `2` 的标签则是 `3`
    - 使用 `,` 分隔
- `num_class`, default= `1`, type=int, alias= `num_classes`
    - 只用于 `multiclass` 分类

- `reg_sqrt`, default= `false`, type=bool
  - 只用于 `regression`
  - 适合 `sqrt(label)` 相反, 预测结果也会自动转换成 `pow2(prediction)`

度量参数

- `metric`, default={ `l2` for regression}, { `binary_logloss` for binary classification}, { `ndcg` for lambdarank}, type=multi-enum, options= `l1`, `l2`, `ndcg`, `auc`, `binary_logloss`, `binary_error` ...
  - `l1`, absolute loss, alias= `mean_absolute_error`, `mae`
  - `l2`, square loss, alias= `mean_squared_error`, `mse`
  - `l2_root`, root square loss, alias= `root_mean_squared_error`, `rmse`
  - `quantile`, Quantile regression
  - `huber`, Huber loss
  - `fair`, Fair loss
  - `poisson`, Poisson regression
  - `ndcg`, NDCG
  - `map`, MAP
  - `auc`, AUC
  - `binary_logloss`, log loss
  - `binary_error`, 样本: `0` 的正确分类, `1` 错误分类
  - `multi_logloss`, mulit-class 损失日志分类
  - `multi_error`, error rate for mulit-class 出错率分类
  - `xentropy`, cross-entropy (与可选的线性权重), alias= `cross_entropy`
  - `xentlambda`, “intensity-weighted” 交叉熵, alias= `cross_entropy_lambda`
  - `kldiv`, Kullback-Leibler divergence, alias= `kullback_leibler`
  - 支持多指标, 使用 `,` 分隔
- `metric_freq`, default= `1`, type=int
  - 频率指标输出
- `train_metric`, default= `false`, type=bool, alias= `training_metric`, `is_training_metric`
  - 如果你需要输出训练的度量结果则设置 `true`
- `ndcg_at`, default= `1,2,3,4,5`, type=multi-int, alias= `ndcg_eval_at`, `eval_at`
  - NDCG 职位评估, 使用 `,` 分隔

网络参数

以下参数用于并行学习, 只用于基本(socket)版本。

- `num_machines`, default= `1`, type=int, alias= `num_machine`
  - 用于并行学习的并行学习应用程序的数量
  - 需要在socket和mpi版本中设置这个
- `local_listen_port`, default= `12400`, type=int, alias= `local_port`
  - 监听本地机器的TCP端口
  - 在培训之前, 您应该再防火墙设置中放开该端口
- `time_out`, default= `120`, type=int
  - 允许socket几分钟内超时
- `machine_list_file`, default= `""`, type=string, alias= `mlist`
  - 为这个并行学习应用程序列出机器的文件
  - 每一行包含一个IP和一个端口为一台机器。格式是ip port, 由空格分隔

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。
  - default为 `-1`, 意味着整个系统平台
- `gpu_device_id`, default= `-1`, type=int
  - OpenCL设备ID在指定的平台上。 在选定的平台上的每一个GPU都有一个唯一的设备ID
  - default为 `-1`, 这个default意味着选定平台上的设备
- `gpu_use_dp`, default= `false`, type=bool
  - 设置为 `true` 在GPU上使用双精度GPU (默认使用单精度)

模型参数

该特性仅在命令行版本中得到支持。

- `convert_model_language`, default= `""`, type=string
  - 只支持 `cpp`
  - 如果 `convert_model_language` 设置为 `task``时 该模型也将转换为 ``train`,

- `convert_model`, default=`"gbdt_prediction.cpp"`, type=string
    - 转换模型的输出文件名

其他

持续训练输入分数

LightGBM支持对初始得分进行持续的培训。它使用一个附加的文件来存储这些初始值, 如下:

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

它意味着最初的得分第一个数据行是 `0.5`,第二个是 `-0.1` 等等。 初始得分文件与数据文件逐行对应, 每一行有一个分数。 如果数据文件的名称是 `train.txt`, 最初的分数文件应该被命名为 train.txt.init` 与作为数据文件在同一文件夹。 在这种情况下, LightGBM 将自动加载初始得分文件, 如果它存在的话。

权重数据

LightGBM 加权训练。它使用一个附加文件来存储权重数据, 如下:

```
1.0
0.5
0.8
...
```

它意味的重压着第一个数据行是 `1.0`, 第二个是 `0.5`, 等等. 权重文件按行与数据文件行相对应, 每行的权重为. 如果数据文件的名称是 `train.txt`, 应该将重量文件命名为 `train.txt.weight` 与数据文件相同的文件夹. 在这种情况下, LightGBM 将自动加载权重文件, 如果它存在的话.

update: 现在可以在数据文件中指定 `weight` 列。请参阅以上参数的参数.

查询数据

对于 LambdaRank 的学习, 它需要查询信息来训练数据. LightGBM 使用一个附加文件来存储查询数据, 如下:

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

它意味着第一个 `27` 行样本属于一个查询和下一个 `18` 行属于另一个, 等等. **Note**: 数据应该由查询来排序.

如果数据文件的名称是`train.txt`, 这个查询文件应该被命名为 train.txt.query``查询在相同的培训数据文件夹中。 在这种情况下, LightGBM将自动加载查询文件, 如果它存在的话。

update: 现在可以在数据文件中指定特定的 query/group id。请参阅上面的参数组。

参数优化

该页面包含了 LightGBM 中所有的参数.

其他有用链接列表

- 参数
- Python API

针对 Leaf-wise (最佳优先) 树的参数优化

LightGBM uses the leaf-wise tree growth algorithm, while many other popular tools use depth-wise tree growth. Compared with depth-wise growth, the leaf-wise algorithm can convenge much faster. However, the leaf-wise growth may be over-fitting if not used with the appropriate parameters.

LightGBM 使用 leaf-wise 的树生长策略, 而很多其他流行的算法采用 depth-wise 的树生长策略. 与 depth-wise 的树生长策略相较, leaf-wise 算法可以收敛的更快. 但是, 如果参数选择不当的话, leaf-wise 算法有可能导致过拟合.

To get good results using a leaf-wise tree, these are some important parameters:

想要在使用 leaf-wise 算法时得到好的结果, 这里有几个重要的参数值得注意:

1. `num_leaves`. This is the main parameter to control the complexity of the tree model. Theoretically, we can set `num_leaves = 2^(max_depth)` to convert from depth-wise tree. However, this simple conversion is not good in practice. The reason is, when number of leaves are the same, the leaf-wise tree is much deeper than depth-wise tree. As a result, it may be over-fitting. Thus, when trying to tune the `num_leaves`, we should let it be smaller than `2^(max_depth)`. For example, when the `max_depth=6` the depth-wise tree can get good accuracy, but setting `num_leaves` to `127` may cause over-fitting, and setting it to `70` or `80` may get better accuracy than depth-wise. Actually, the concept `depth` can be forgotten in leaf-wise tree, since it doesn't have a correct mapping from `leaves` to `depth`.

2. `num_leaves`. 这是控制树模型复杂度的主要参数. 理论上, 借鉴 depth-wise 树, 我们可以设置 `num_leaves = 2^(max_depth)` 但是, 这种简单的转化在实际应用中表现不佳. 这是因为, 当叶子数目相同时, leaf-wise 树要比 depth-wise 树深得多, 这就有可能导致过拟合. 因此, 当我们试着调整 `num_leaves` 的取值时, 应该让其小于 `2^(max_depth)`. 举个例子, 当 `max_depth=6` 时(这里译者认为例子中, 树的最大深度应为7), depth-wise 树可以达到较高的准确率.但是如果设置 `num_leaves` 为 `127` 时, 有可能会导致过拟合, 而将其设置为 `70` 或 `80` 时可能会得到比 depth-wise 树更高的准确率. 其实, `depth` 的概念在 leaf-wise 树中并没有多大作用, 因为并不存在一个从 `leaves` 到 `depth` 的合理映射.
3. `min_data_in_leaf`. This is a very important parameter to deal with over-fitting in leaf-wise tree. Its value depends on the number of training data and `num_leaves`. Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset.
4. `min_data_in_leaf`. 这是处理 leaf-wise 树的过拟合问题中一个非常重要的参数. 它的值取决于训练数据的样本个树和 `num_leaves`. 将其设置的较大可以避免生成一个过深的树, 但有可能导致欠拟合. 实际应用中, 对于大数据集, 设置其为几百或几千就足够了.
5. `max_depth`. You also can use `max_depth` to limit the tree depth explicitly.
6. `max_depth`. 你也可以利用 `max_depth` 来显式地限制树的深度.

针对更快的训练速度

- Use bagging by setting `bagging_fraction` and `bagging_freq`
- Use feature sub-sampling by setting `feature_fraction`
- Use small `max_bin`
- Use `save_binary` to speed up data loading in future learning
- Use parallel learning, refer to 并行学习指南
- 通过设置 `bagging_fraction` 和 `bagging_freq` 参数来使用 bagging 方法
- 通过设置 `feature_fraction` 参数来使用特征的子抽样
- 使用较小的 `max_bin`
- 使用 `save_binary` 在未来的学习过程对数据加载进行加速
- 使用并行学习, 可参考 并行学习指南

针对更好的准确率

- Use large `max_bin` (may be slower)
- Use small `learning_rate` with large `num_iterations`
- Use large `num_leaves` (may cause over-fitting)

- Use bigger training data
- Try `dart`
- 使用较大的 `max_bin` （学习速度可能变慢）
- 使用较小的 `learning_rate` 和较大的 `num_iterations`
- 使用较大的 `num_leaves` （可能导致过拟合）
- 使用更大的训练数据
- 尝试 `dart`

处理过拟合

- Use small `max_bin`
- Use small `num_leaves`
- Use `min_data_in_leaf` and `min_sum_hessian_in_leaf`
- Use bagging by set `bagging_fraction` and `bagging_freq`
- Use feature sub-sampling by set `feature_fraction`
- Use bigger training data
- Try `lambda_l1`, `lambda_l2` and `min_gain_to_split` for regularization
- Try `max_depth` to avoid growing deep tree
- 使用较小的 `max_bin`
- 使用较小的 `num_leaves`
- 使用 `min_data_in_leaf` 和 `min_sum_hessian_in_leaf`
- 通过设置 `bagging_fraction` 和 `bagging_freq` 来使用 bagging
- 通过设置 `feature_fraction` 来使用特征子抽样
- 使用更大的训练数据
- 使用 `lambda_l1`, `lambda_l2` 和 `min_gain_to_split` 来使用正则
- 尝试 `max_depth` 来避免生成过深的树

Python API

Data Structure API

class lightgbm.Dataset(data, label=None, max_bin=None, reference=None, weight=None, group=None, init_score=None, silent=False, feature_name='auto', categorical_feature='auto', params=None, free_raw_data=True)

Bases: `object`

Dataset in LightGBM.

Constract Dataset.

- Parameters:
- **data** (*string_,_ numpy array or scipy.sparse*) – Data source of Dataset. If string, it represents the path to txt file.
- **label** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Label of the data.
- **max_bin** (*int or None_,_ optional (__default=None_)_*) – Max number of discrete bins for features. If None, default value from parameters of CLI-version will be used.
- **reference** (*Dataset or None_,_ optional (__default=None_)_*) – If this is Dataset for validation, training data should be used as reference.
- **weight** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Weight for each instance.
- **group** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Group/query size for Dataset.
- **init_score** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Init score for Dataset.
- **silent** (*bool_,_ optional (__default=False_)_*) – Whether to print messages during construction.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_, or_ 'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **params** (*dict or None_,_ optional (__default=None_)_*) – Other parameters.

- **free_raw_data** (*bool_,_ optional (__default=True_)_*) – If True, raw data is freed after constructing inner Dataset.

construct()

Lazy init.

- Returns:
- **self** – Returns self.
- Return type:
- Dataset

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

Create validation data align with current Dataset.

- Parameters:
- **data** (*string_,_ numpy array or scipy.sparse*) – Data source of Dataset. If string, it represents the path to txt file.
- **label** (*list or numpy 1-D array_,_ optional (__default=None_)_*) – Label of the training data.
- **weight** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Weight for each instance.
- **group** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Group/query size for Dataset.
- **init_score** (*list_,_ numpy 1-D array or None_,_ optional (__default=None_)_*) – Init score for Dataset.
- **silent** (*bool_,_ optional (__default=False_)_*) – Whether to print messages during construction.
- **params** (*dict or None_,_ optional (__default=None_)_*) – Other parameters.
- Returns:
- **self** – Returns self
- Return type:
- Dataset

get_field(field_name)

Get property from the Dataset.

- Parameters:
- **field_name** (*string*) – The field name of the information.

- Returns:
- **info** – A numpy array with information from the Dataset.
- Return type:
- numpy array

get_group()

Get the group of the Dataset.

- Returns:
- **group** – Group size of each group.
- Return type:
- numpy array

get_init_score()

Get the initial score of the Dataset.

- Returns:
- **init_score** – Init score of Booster.
- Return type:
- numpy array

get_label()

Get the label of the Dataset.

- Returns:
- **label** – The label information from the Dataset.
- Return type:
- numpy array

get_ref_chain(ref_limit=100)

Get a chain of Dataset objects, starting with r, then going to r.reference if exists, then to r.reference.reference, etc. until we hit `ref_limit` or a reference loop.

- Parameters:
- **ref_limit** (*int_,_ optional (__default=100_)_*) – The limit number of references.
- Returns:
- **ref_chain** – Chain of references of the Datasets.
- Return type:

- set of Dataset

get_weight()

Get the weight of the Dataset.

- Returns:
- **weight** – Weight for each data point from the Dataset.
- Return type:
- numpy array

num_data()

Get the number of rows in the Dataset.

- Returns:
- **number_of_rows** – The number of rows in the Dataset.
- Return type:
- int

num_feature()

Get the number of columns (features) in the Dataset.

- Returns:
- **number_of_columns** – The number of columns (features) in the Dataset.
- Return type:
- int

save_binary(filename)

Save Dataset to binary file.

- Parameters:
- **filename** (*string*) – Name of the output file.

set_categorical_feature(categorical_feature)

Set categorical features.

- Parameters:
- **categorical_feature** (*list of int or strings*) – Names or indices of categorical features.

set_feature_name(feature_name)

Set feature name.

- Parameters:
- **feature_name** (*list of strings*) – Feature names.

set_field(field_name, data)

Set property into the Dataset.

- Parameters:
- **field_name** (*string*) – The field name of the information.
- **data** (*list_,_ numpy array or None*) – The array of data to be set.

set_group(group)

Set group size of Dataset (used for ranking).

- Parameters:
- **group** (*list_,_ numpy array or None*) – Group size of each group.

set_init_score(init_score)

Set init score of Booster to start from.

- Parameters:
- **init_score** (*list_,_ numpy array or None*) – Init score for Booster.

set_label(label)

Set label of Dataset

- Parameters:
- **label** (*list_,_ numpy array or None*) – The label information to be set into Dataset.

set_reference(reference)

Set reference Dataset.

- Parameters:
- **reference** (*Dataset*) – Reference that is used as a template to consturct the current Dataset.

set_weight(weight)

Set weight of each instance.

- Parameters:

- **weight** (*list_,_ numpy array or None*) – Weight to be set for each data point.

subset(used_indices, params=None)

Get subset of current Dataset.

- Parameters:
- **used_indices** (*list of int*) – Indices used to create the subset.
- **params** (*dict or None_,_ optional (__default=None_)_*) – Other parameters.
- Returns:
- **subset** – Subset of the current Dataset.
- Return type:
- Dataset

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

Bases: `object`

Booster in LightGBM.

Initialize the Booster.

- Parameters:
- **params** (*dict or None_,_ optional (__default=None_)_*) – Parameters for Booster.
- **train_set** (*Dataset or None_,_ optional (__default=None_)_*) – Training dataset.
- **model_file** (*string or None_,_ optional (__default=None_)_*) – Path to the model file.
- **silent** (*bool_,_ optional (__default=False_)_*) – Whether to print messages during construction.

add_valid(data, name)

Add validation data.

- Parameters:
- **data** (*Dataset*) – Validation data.
- **name** (*string*) – Name of validation data.

attr(key)

Get attribute string from the Booster.

- Parameters:
- **key** (*string*) – The name of the attribute.

- Returns:
- **value** – The attribute value. Returns None if attribute do not exist.
- Return type:
- string or None

current_iteration()

Get the index of the current iteration.

- Returns:
- **cur_iter** – The index of the current iteration.
- Return type:
- int

dump_model(num_iteration=-1)

Dump Booster to json format.

- Parameters:
- **num_iteration** (*int_,_ optional (__default=-1_)_*) – Index of the iteration that should to dumped. If <0, the best iteration (if exists) is dumped.
- Returns:
- **json_repr** – Json format of Booster.
- Return type:
- dict

eval(data, name, feval=None)

Evaluate for data.

- Parameters:
- **data** (*Dataset*) – Data for the evaluating.
- **name** (*string*) – Name of the data.
- **feval** (*callable or None_,_ optional (__default=None_)_*) – Custom evaluation function.
- Returns:
- **result** – List with evaluation results.
- Return type:
- list

eval_train(feval=None)

Evaluate for training data.

- Parameters:
- **feval** (*callable or None_,_ optional (__default=None_)_*) – Custom evaluation function.
- Returns:
- **result** – List with evaluation results.
- Return type:
- list

eval_valid(feval=None)

Evaluate for validation data.

- Parameters:
- **feval** (*callable or None_,_ optional (__default=None_)_*) – Custom evaluation function.
- Returns:
- **result** – List with evaluation results.
- Return type:
- list

feature_importance(importance_type='split', iteration=-1)

Get feature importances.

- Parameters:
- **importance_type** (*string_,_ optional (__default="split"_)_*) – How the importance is calculated. If "split", result contains numbers of times the feature is used in a model. If "gain", result contains total gains of splits which use the feature.
- Returns:
- **result** – Array with feature importances.
- Return type:
- numpy array

feature_name()

Get names of features.

- Returns:
- **result** – List with names of features.
- Return type:
- list

free_dataset()

Free Booster's Datasets.

free_network()

Free Network.

get_leaf_output(tree_id, leaf_id)

Get the output of a leaf.

- Parameters:
- **tree_id** (*int*) – The index of the tree.
- **leaf_id** (*int*) – The index of the leaf in the tree.
- Returns:
- **result** – The output of the leaf.
- Return type:
- float

num_feature()

Get number of features.

- Returns:
- **num_feature** – The number of features.
- Return type:
- int

predict(data, num_iteration=-1, raw_score=False, pred_leaf=False, pred_contrib=False, data_has_header=False, is_reshape=True, pred_parameter=None)

Make a prediction.

- Parameters:
- **data** (*string_,_ numpy array or scipy.sparse*) – Data source for prediction. If string, it represents the path to txt file.
- **num_iteration** (*int_,_ optional (__default=-1_)_*) – Iteration used for prediction. If <0, the best iteration (if exists) is used for prediction.
- **raw_score** (*bool_,_ optional (__default=False_)_*) – Whether to predict raw scores.
- **pred_leaf** (*bool_,_ optional (__default=False_)_*) – Whether to predict leaf index.
- **pred_contrib** (*bool_,_ optional (__default=False_)_*) – Whether to predict feature contributions.

- **data_has_header** (*bool_,_ optional (__default=False_)_*) – Whether the data has header. Used only if data is string.
- **is_reshape** (*bool_,_ optional (__default=True_)_*) – If True, result is reshaped to [nrow, ncol].
- **pred_parameter** (*dict or None_,_ optional (__default=None_)_*) – Other parameters for the prediction.
- Returns:
- **result** – Prediction result.
- Return type:
- numpy array

reset_parameter(params)

Reset parameters of Booster.

- Parameters:
- **params** (*dict*) – New parameters for Booster.

rollback_one_iter()

Rollback one iteration.

save_model(filename, num_iteration=-1)

Save Booster to file.

- Parameters:
- **filename** (*string*) – Filename to save Booster.
- **num_iteration** (*int_,_ optional (__default=-1_)_*) – Index of the iteration that should to saved. If <0, the best iteration (if exists) is saved.

set_attr(**kwargs)

Set the attribute of the Booster.

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

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

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.

set_train_data_name(name)

Set the name to the training Dataset.

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

update(train_set=None, fobj=None)

Update 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

Training API

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)

Perform the training with given parameters.

- Parameters:
  - **params** (*dict*) – Parameters for training.
  - **train_set** (*Dataset*) – Data to be trained.
  - **num_boost_round** (*int_,_ optional (__default=100_)_*) – Number of boosting iterations.

- **valid_sets** (*list of Datasets or None_,_ optional (__default=None_)_*) – List of data to be evaluated during training.
- **valid_names** (*list of string or None_,_ optional (__default=None_)_*) – Names of `valid_sets`.
- **fobj** (*callable or None_,_ optional (__default=None_)_*) – Customized objective function.
- **feval** (*callable or None_,_ optional (__default=None_)_*) – Customized evaluation function. Note: should return (eval_name, eval_result, is_higher_better) or list of such tuples.
- **init_model** (*string or None_,_ optional (__default=None_)_*) – Filename of LightGBM model or Booster instance used for continue training.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_, or_ 'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – Activates early stopping. The model will train until the validation score stops improving. Requires at least one validation data and one metric. If there's more than one, will check all of them. If early stopping occurs, the model will add `best_iteration` field.
- **evals_result** (*dict or None_,_ optional (__default=None_)_*) – This dictionary used to store all evaluation results of all the items in `valid_sets`. Example With a `valid_sets` = [valid_set, train_set], `valid_names` = ['eval', 'train'] and a `params` = ('metric':'logloss') returns: {'train': {'logloss': ['0.48253', '0.35953', …]}, 'eval': {'logloss': ['0.480385', '0.357756', …]}}.
- **verbose_eval** (*bool or int_,_ optional (__default=True_)_*) – Requires at least one validation data. If True, the eval metric on the valid set is printed at each boosting stage. If int, the eval metric on the valid set is printed at every `verbose_eval` boosting stage. The last boosting stage or the boosting stage found by using `early_stopping_rounds` is also printed. Example With `verbose_eval` = 4 and at least one item in evals, an evaluation metric is printed every 4 (instead of 1) boosting stages.
- **learning_rates** (*list_,_ callable or None_,_ optional (__default=None_)_*) – List of learning rates for each boosting round or a customized function that calculates `learning_rate` in terms of current number of round (e.g. yields learning rate decay).
- **keep_training_booster** (*bool_,_ optional (__default=False_)_*) – Whether the returned Booster will be used to keep training. If False, the returned value will be converted into _InnerPredictor before returning. You can still use _InnerPredictor as `init_model` for future continue training.

- **callbacks** (*list of callables or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.

- Returns:
  - **booster** – The trained Booster model.
- Return type:
  - Booster

lightgbm.cv(params, train_set, num_boost_round=10, 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)

Perform the cross-validation with given paramaters.

- Parameters:
- **params** (*dict*) – Parameters for Booster.
- **train_set** (*Dataset*) – Data to be trained on.
- **num_boost_round** (*int_,_ optional (__default=10_)_*) – Number of boosting iterations.
- **folds** (*a generator or iterator of (__train_idx_,_ test_idx_)_ tuples or None_,_ optional (__default=None_)_*) – The train and test indices for the each fold. This argument has highest priority over other data split arguments.
- **nfold** (*int_,_ optional (__default=5_)_*) – Number of folds in CV.
- **stratified** (*bool_,_ optional (__default=True_)_*) – Whether to perform stratified sampling.
- **shuffle** (*bool_,_ optional (__default=True_)_*) – Whether to shuffle before splitting data.
- **metrics** (*string_,_ list of strings or None_,_ optional (__default=None_)_*) – Evaluation metrics to be monitored while CV. If not None, the metric in `params` will be overridden.
- **fobj** (*callable or None_,_ optional (__default=None_)_*) – Custom objective function.
- **feval** (*callable or None_,_ optional (__default=None_)_*) – Custom evaluation function.
- **init_model** (*string or None_,_ optional (__default=None_)_*) – Filename of LightGBM model or Booster instance used for continue training.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_, or_ 'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.

- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – Activates early stopping. CV error needs to decrease at least every `early_stopping_rounds` round(s) to continue. Last entry in evaluation history is the one from best iteration.
- **fpreproc** (*callable or None_,_ optional (__default=None_)_*) – Preprocessing function that takes (dtrain, dtest, params) and returns transformed versions of those.
- **verbose_eval** (*bool_,_ int_, or_ None_,_ optional (__default=None_)_*) – Whether to display the progress. If None, progress will be displayed when np.ndarray is returned. If True, progress will be displayed at every boosting stage. If int, progress will be displayed at every given `verbose_eval` boosting stage.
- **show_stdv** (*bool_,_ optional (__default=True_)_*) – Whether to display the standard deviation in progress. Results are not affected by this parameter, and always contains std.
- **seed** (*int_,_ optional (__default=0_)_*) – Seed used to generate the folds (passed to numpy.random.seed).
- **callbacks** (*list of callables or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.
- Returns:
- **eval_hist** – Evaluation history. The dictionary has the following format: {'metric1-mean': [values], 'metric1-stdv': [values], 'metric2-mean': [values], 'metric1-stdv': [values], ...}.
- Return type:
- dict

Scikit-learn API

class lightgbm.LGBMModel(boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=10, max_bin=255, subsample_for_bin=200000, objective=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=1, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=-1, silent=True, **kwargs)

Bases: `object`

Implementation of the scikit-learn API for LightGBM.

Construct a gradient boosting model.

- Parameters:
  - **boosting_type** (*string_,_ optional (__default="gbdt"_)_*) – 'gbdt', traditional Gradient Boosting Decision Tree. 'dart', Dropouts meet Multiple Additive Regression Trees. 'goss', Gradient-based One-Side Sampling. 'rf', Random Forest.

- **num_leaves** (*int_,_ optional (__default=31_)_*) – Maximum tree leaves for base learners.
- **max_depth** (*int_,_ optional (__default=-1_)_*) – Maximum tree depth for base learners, -1 means no limit.
- **learning_rate** (*float_,_ optional (__default=0.1_)_*) – Boosting learning rate.
- **n_estimators** (*int_,_ optional (__default=10_)_*) – Number of boosted trees to fit.
- **max_bin** (*int_,_ optional (__default=255_)_*) – Number of bucketed bins for feature values.
- **subsample_for_bin** (*int_,_ optional (__default=50000_)_*) – Number of samples for constructing bins.
- **objective** (*string_,_ callable or None_,_ optional (__default=None_)_*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below). default: 'regression' for LGBMRegressor, 'binary' or 'multiclass' for LGBMClassifier, 'lambdarank' for LGBMRanker.
- **min_split_gain** (*float_,_ optional (__default=0._)_*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*float_,_ optional (__default=1e-3_)_*) – Minimum sum of instance weight(hessian) needed in a child(leaf).
- **min_child_samples** (*int_,_ optional (__default=20_)_*) – Minimum number of data need in a child(leaf).
- **subsample** (*float_,_ optional (__default=1._)_*) – Subsample ratio of the training instance.
- **subsample_freq** (*int_,_ optional (__default=1_)_*) – Frequence of subsample, <=0 means no enable.
- **colsample_bytree** (*float_,_ optional (__default=1._)_*) – Subsample ratio of columns when constructing each tree.
- **reg_alpha** (*float_,_ optional (__default=0._)_*) – L1 regularization term on weights.
- **reg_lambda** (*float_,_ optional (__default=0._)_*) – L2 regularization term on weights.
- **random_state** (*int or None_,_ optional (__default=None_)_*) – Random number seed. Will use default seeds in c++ code if set to None.
- **n_jobs** (*int_,_ optional (__default=-1_)_*) – Number of parallel threads.
- **silent** (*bool_,_ optional (__default=True_)_*) – Whether to print messages while running boosting.
- ****kwargs** (*other parameters*) – Check http://lightgbm.readthedocs.io/en/latest/Parameters.html for more parameters. Note **kwargs is not supported in sklearn, it may cause unexpected issues.

n_features_

int – The number of features of fitted model.

classes_

array of shape = [n_classes] – The class label array (only for classification problem).

n_classes_

int – The number of classes (only for classification problem).

best_score_

dict or None – The best score of fitted model.

best_iteration_

int or None – The best iteration of fitted model if `early_stopping_rounds` has been specified.

objective_

string or callable – The concrete objective used while fitting this model.

booster_

Booster – The underlying Booster of this model.

evals_result_

dict or None – The evaluation results if `early_stopping_rounds` has been specified.

feature_importances_

array of shape = [n_features] – The feature importances (the higher, the more important the feature).

Note

A custom objective function can be provided for the `objective` parameter. In this case, it should have the signature `objective(y_true, y_pred) -> grad, hess` or `objective(y_true, y_pred, group) -> grad, hess`:

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class task)
```

The predicted values.

```
group: array-like
```

Group/query data, used for ranking task.

```
grad: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the gradient for each sample point.

```
hess: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the second derivative for each sample point.

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

apply(X, num_iteration=0)

Return the predicted leaf every tree for each sample.

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_*) – Input features matrix.
- **num_iteration** (*int_,_ optional (__default=0_)_*) – Limit number of iterations in the prediction; defaults to 0 (use all trees).
- Returns:
- **X_leaves** – The predicted leaf every tree for each sample.
- Return type:
- array-like of shape = [n_samples, n_trees]

best_iteration_

Get the best iteration of fitted model.

best_score_

Get the best score of fitted model.

booster_

Get the underlying lightgbm Booster of this model.

evals_result_

Get the evaluation results.

feature_importances_

Get feature importances.

Note

Feature importance in sklearn interface used to normalize to 1, it's deprecated after 2.0.4 and same as Booster.feature_importance() now.

fit(X, y, sample_weight=None, init_score=None, group=None, eval_set=None, eval_names=None, eval_sample_weight=None, eval_init_score=None, eval_group=None, eval_metric=None, early_stopping_rounds=None, verbose=True, feature_name='auto', categorical_feature='auto', callbacks=None)

Build a gradient boosting model from the training set (X, y).

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_*) – Input feature matrix.
- **y** (*array-like of shape = [__n_samples_]_*) – The target values (class labels in classification, real numbers in regression).
- **sample_weight** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Weights of training data.
- **init_score** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Init score of training data.
- **group** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Group data of training data.
- **eval_set** (*list or None_,_ optional (__default=None_)_*) – A list of (X, y) tuple pairs to use as a validation sets for early-stopping.
- **eval_names** (*list of strings or None_,_ optional (__default=None_)_*) – Names of eval_set.
- **eval_sample_weight** (*list of arrays or None_,_ optional (__default=None_)_*) – Weights of eval data.
- **eval_init_score** (*list of arrays or None_,_ optional (__default=None_)_*) – Init score of eval data.
- **eval_group** (*list of arrays or None_,_ optional (__default=None_)_*) – Group data of eval data.
- **eval_metric** (*string_,_ list of strings_,_ callable or None_,_ optional (__default=None_)_*) – If string, it should be a built-in evaluation metric to use. If callable, it should be a custom evaluation metric, see note for more details.

- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – Activates early stopping. The model will train until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` round(s) to continue training.
- **verbose** (*bool_,_ optional (__default=True_)_*) – If True and an evaluation set is used, writes the evaluation progress.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_, or_ 'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **callbacks** (*list of callback functions or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.
- Returns:
- **self** – Returns self.
- Return type:
- object Note Custom eval function expects a callable with following functions: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)`. Returns (eval_name, eval_result, is_bigger_better) or list of (eval_name, eval_result, is_bigger_better)

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class)
```

The predicted values.

```
weight: array-like of shape = [n_samples]
```

The weight of samples.

```
group: array-like
```

Group/query data, used for ranking task.

```
eval_name: str
```

The name of evaluation.

```
eval_result: float
```

The eval result.

```
is_bigger_better: bool
```

Is eval result bigger better, e.g. AUC is bigger_better.

For multi-class task, the y_pred is group by class_id first, then group by row_id. If you want to get i-th row y_pred in j-th class, the access way is y_pred[j * num_data + i].

n_features_

Get the number of features of fitted model.

objective_

Get the concrete objective used while fitting this model.

predict(X, raw_score=False, num_iteration=0)

Return the predicted value for each sample.

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_) – Input features matrix.
- **raw_score** (*bool_,_ optional (__default=False_)_) – Whether to predict raw scores.
- **num_iter ation** (*int_,_ optional (__default=0_)_) – Limit number of iterations in the prediction; defaults to 0 (use all trees).
- Returns:
- **predicted_result** – The predicted values.
- Return type:
- array-like of shape = [n_samples] or shape = [n_samples, n_classes]

class lightgbm.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=10, max_bin=255, subsample_for_bin=200000, objective=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=1, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=-1, silent=True, **kwargs)

Bases: `lightgbm.sklearn.LGBMModel`, `object`

LightGBM classifier.

Construct a gradient boosting model.

- Parameters:
  - **boosting_type** (*string_,_ optional (__default="gbdt"_)_*) – 'gbdt', traditional Gradient Boosting Decision Tree. 'dart', Dropouts meet Multiple Additive Regression Trees. 'goss', Gradient-based One-Side Sampling. 'rf', Random Forest.
  - **num_leaves** (*int_,_ optional (__default=31_)_*) – Maximum tree leaves for base learners.
  - **max_depth** (*int_,_ optional (__default=-1_)_*) – Maximum tree depth for base learners, -1 means no limit.
  - **learning_rate** (*float_,_ optional (__default=0.1_)_*) – Boosting learning rate.
  - **n_estimators** (*int_,_ optional (__default=10_)_*) – Number of boosted trees to fit.
  - **max_bin** (*int_,_ optional (__default=255_)_*) – Number of bucketed bins for feature values.
  - **subsample_for_bin** (*int_,_ optional (__default=50000_)_*) – Number of samples for constructing bins.
  - **objective** (*string_,_ callable or None_,_ optional (__default=None_)_*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below). default: 'regression' for LGBMRegressor, 'binary' or 'multiclass' for LGBMClassifier, 'lambdarank' for LGBMRanker.
  - **min_split_gain** (*float_,_ optional (__default=0._)_*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
  - **min_child_weight** (*float_,_ optional (__default=1e-3_)_*) – Minimum sum of instance weight(hessian) needed in a child(leaf).
  - **min_child_samples** (*int_,_ optional (__default=20_)_*) – Minimum number of data need in a child(leaf).
  - **subsample** (*float_,_ optional (__default=1._)_*) – Subsample ratio of the training instance.
  - **subsample_freq** (*int_,_ optional (__default=1_)_*) – Frequence of subsample, <=0 means no enable.
  - **colsample_bytree** (*float_,_ optional (__default=1._)_*) – Subsample ratio of columns when constructing each tree.
  - **reg_alpha** (*float_,_ optional (__default=0._)_*) – L1 regularization term on weights.
  - **reg_lambda** (*float_,_ optional (__default=0._)_*) – L2 regularization term on weights.
  - **random_state** (*int or None_,_ optional (__default=None_)_*) – Random number seed. Will use default seeds in c++ code if set to None.
  - **n_jobs** (*int_,_ optional (__default=-1_)_*) – Number of parallel threads.

- **silent** (*bool_,_ optional (__default=True_)_*) – Whether to print messages while running boosting.
- ****kwargs** (*other parameters*) – Check http://lightgbm.readthedocs.io/en/latest/Parameters.html for more parameters. Note **kwargs is not supported in sklearn, it may cause unexpected issues.

n_features_

int – The number of features of fitted model.

classes_

array of shape = [n_classes] – The class label array (only for classification problem).

n_classes_

int – The number of classes (only for classification problem).

best_score_

dict or None – The best score of fitted model.

best_iteration_

int or None – The best iteration of fitted model if `early_stopping_rounds` has been specified.

objective_

string or callable – The concrete objective used while fitting this model.

booster_

Booster – The underlying Booster of this model.

evals_result_

dict or None – The evaluation results if `early_stopping_rounds` has been specified.

feature_importances_

array of shape = [n_features] – The feature importances (the higher, the more important the feature).

Note

A custom objective function can be provided for the `objective` parameter. In this case, it should have the signature `objective(y_true, y_pred) -> grad, hess` or `objective(y_true, y_pred, group) -> grad, hess`:

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class task)
```

The predicted values.

```
group: array-like
```

Group/query data, used for ranking task.

```
grad: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the gradient for each sample point.

```
hess: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the second derivative for each sample point.

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

classes_

Get the class label array.

fit(X, y, sample_weight=None, init_score=None, eval_set=None, eval_names=None, eval_sample_weight=None, eval_init_score=None, eval_metric='logloss', early_stopping_rounds=None, verbose=True, feature_name='auto', categorical_feature='auto', callbacks=None)

Build a gradient boosting model from the training set (X, y).

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_*) – Input feature matrix.
- **y** (*array-like of shape = [__n_samples_]_*) – The target values (class labels in classification, real numbers in regression).
- **sample_weight** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Weights of training data.

- **init_score** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Init score of training data.
- **group** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Group data of training data.
- **eval_set** (*list or None_,_ optional (__default=None_)_*) – A list of (X, y) tuple pairs to use as a validation sets for early-stopping.
- **eval_names** (*list of strings or None_,_ optional (__default=None_)_*) – Names of eval_set.
- **eval_sample_weight** (*list of arrays or None_,_ optional (__default=None_)_*) – Weights of eval data.
- **eval_init_score** (*list of arrays or None_,_ optional (__default=None_)_*) – Init score of eval data.
- **eval_group** (*list of arrays or None_,_ optional (__default=None_)_*) – Group data of eval data.
- **eval_metric** (*string_,_ list of strings_,_ callable or None_,_ optional (__default="logloss"_)_*) – If string, it should be a built-in evaluation metric to use. If callable, it should be a custom evaluation metric, see note for more details.
- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – Activates early stopping. The model will train until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` round(s) to continue training.
- **verbose** (*bool_,_ optional (__default=True_)_*) – If True and an evaluation set is used, writes the evaluation progress.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_*, or_ *'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **callbacks** (*list of callback functions or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.
- Returns:
- **self** – Returns self.
- Return type:
- object

Note

Custom eval function expects a callable with following functions: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)`. Returns (eval_name, eval_result, is_bigger_better) or list of (eval_name, eval_result, is_bigger_better)

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class)
```

The predicted values.

```
weight: array-like of shape = [n_samples]
```

The weight of samples.

```
group: array-like
```

Group/query data, used for ranking task.

```
eval_name: str
```

The name of evaluation.

```
eval_result: float
```

The eval result.

```
is_bigger_better: bool
```

Is eval result bigger better, e.g. AUC is bigger_better.

For multi-class task, the y_pred is group by class_id first, then group by row_id. If you want to get i-th row y_pred in j-th class, the access way is y_pred[j * num_data + i].

n_classes_

Get the number of classes.

predict_proba(X, raw_score=False, num_iteration=0)

Return the predicted probability for each class for each sample.

- Parameters:

- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_) – Input features matrix.
- **raw_score** (*bool_,_ optional (__default=False_)_) – Whether to predict raw scores.
- **num_iteration** (*int_,_ optional (__default=0_)_) – Limit number of iterations in the prediction; defaults to 0 (use all trees).
- Returns:
- **predicted_probability** – The predicted probability for each class for each sample.
- Return type:
- array-like of shape = [n_samples, n_classes]

class lightgbm.LGBMRegressor(boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=10, max_bin=255, subsample_for_bin=200000, objective=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=1, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=-1, silent=True, **kwargs)

Bases: `lightgbm.sklearn.LGBMModel`, `object`

LightGBM regressor.

Construct a gradient boosting model.

- Parameters:
  - **boosting_type** (*string_,_ optional (__default="gbdt"_)_) – 'gbdt', traditional Gradient Boosting Decision Tree. 'dart', Dropouts meet Multiple Additive Regression Trees. 'goss', Gradient-based One-Side Sampling. 'rf', Random Forest.
  - **num_leaves** (*int_,_ optional (__default=31_)_) – Maximum tree leaves for base learners.
  - **max_depth** (*int_,_ optional (__default=-1_)_) – Maximum tree depth for base learners, -1 means no limit.
  - **learning_rate** (*float_,_ optional (__default=0.1_)_) – Boosting learning rate.
  - **n_estimators** (*int_,_ optional (__default=10_)_) – Number of boosted trees to fit.
  - **max_bin** (*int_,_ optional (__default=255_)_) – Number of bucketed bins for feature values.
  - **subsample_for_bin** (*int_,_ optional (__default=50000_)_) – Number of samples for constructing bins.
  - **objective** (*string_,_ callable or None_,_ optional (__default=None_)_) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below). default: 'regression' for LGBMRegressor, 'binary' or 'multiclass' for LGBMClassifier, 'lambdarank' for LGBMRanker.

- **min_split_gain** (*float_,_ optional (__default=0._)_*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*float_,_ optional (__default=1e-3_)_*) – Minimum sum of instance weight(hessian) needed in a child(leaf).
- **min_child_samples** (*int_,_ optional (__default=20_)_*) – Minimum number of data need in a child(leaf).
- **subsample** (*float_,_ optional (__default=1._)_*) – Subsample ratio of the training instance.
- **subsample_freq** (*int_,_ optional (__default=1_)_*) – Frequence of subsample, <=0 means no enable.
- **colsample_bytree** (*float_,_ optional (__default=1._)_*) – Subsample ratio of columns when constructing each tree.
- **reg_alpha** (*float_,_ optional (__default=0._)_*) – L1 regularization term on weights.
- **reg_lambda** (*float_,_ optional (__default=0._)_*) – L2 regularization term on weights.
- **random_state** (*int or None_,_ optional (__default=None_)_*) – Random number seed. Will use default seeds in c++ code if set to None.
- **n_jobs** (*int_,_ optional (__default=-1_)_*) – Number of parallel threads.
- **silent** (*bool_,_ optional (__default=True_)_*) – Whether to print messages while running boosting.
- ****kwargs** (*other parameters*) – Check http://lightgbm.readthedocs.io/en/latest/Parameters.html for more parameters. Note **kwargs is not supported in sklearn, it may cause unexpected issues.

n_features_

int – The number of features of fitted model.

classes_

array of shape = [n_classes] – The class label array (only for classification problem).

n_classes_

int – The number of classes (only for classification problem).

best_score_

dict or None – The best score of fitted model.

best_iteration_

int or None – The best iteration of fitted model if `early_stopping_rounds` has been specified.

objective_

string or callable – The concrete objective used while fitting this model.

booster_

Booster – The underlying Booster of this model.

evals_result_

dict or None – The evaluation results if `early_stopping_rounds` has been specified.

feature_importances_

array of shape = [n_features] – The feature importances (the higher, the more important the feature).

Note

A custom objective function can be provided for the `objective` parameter. In this case, it should have the signature `objective(y_true, y_pred) -> grad, hess` or `objective(y_true, y_pred, group) -> grad, hess`:

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class task)
```

The predicted values.

```
group: array-like
```

Group/query data, used for ranking task.

```
grad: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the gradient for each sample point.

```
hess: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the second derivative for each sample point.

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

fit(X, y, sample_weight=None, init_score=None, eval_set=None, eval_names=None, eval_sample_weight=None, eval_init_score=None, eval_metric='l2', early_stopping_rounds=None, verbose=True, feature_name='auto', categorical_feature='auto', callbacks=None)

Build a gradient boosting model from the training set (X, y).

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_*) – Input feature matrix.
- **y** (*array-like of shape = [__n_samples_]_*) – The target values (class labels in classification, real numbers in regression).
- **sample_weight** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Weights of training data.
- **init_score** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Init score of training data.
- **group** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Group data of training data.
- **eval_set** (*list or None_,_ optional (__default=None_)_*) – A list of (X, y) tuple pairs to use as a validation sets for early-stopping.
- **eval_names** (*list of strings or None_,_ optional (__default=None_)_*) – Names of eval_set.
- **eval_sample_weight** (*list of arrays or None_,_ optional (__default=None_)_*) – Weights of eval data.
- **eval_init_score** (*list of arrays or None_,_ optional (__default=None_)_*) – Init score of eval data.
- **eval_group** (*list of arrays or None_,_ optional (__default=None_)_*) – Group data of eval data.
- **eval_metric** (*string_,_ list of strings_,_ callable or None_,_ optional (__default="l2"_)_*) – If string, it should be a built-in evaluation metric to use. If callable, it should be a custom evaluation metric, see note for more details.
- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – A ctivates early stopping. The model will train until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` round(s) to continue training.
- **verbose** (*bool_,_ optional (__default=True_)_*) – If True and an evaluation set is used, writes the evaluation progress.

- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
- **categorical_feature** (*list of strings or int_, or_ 'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **callbacks** (*list of callback functions or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.
- Returns:
- **self** – Returns self.
- Return type:
- object

Note

Custom eval function expects a callable with following functions: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)`. Returns (eval_name, eval_result, is_bigger_better) or list of (eval_name, eval_result, is_bigger_better)

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class)
```

The predicted values.

```
weight: array-like of shape = [n_samples]
```

The weight of samples.

```
group: array-like
```

Group/query data, used for ranking task.

```
eval_name: str
```

The name of evaluation.

```
eval_result: float
```

The eval result.

```
is_bigger_better: bool
```

Is eval result bigger better, e.g. AUC is bigger_better.

For multi-class task, the y_pred is group by class_id first, then group by row_id. If you want to get i-th row y_pred in j-th class, the access way is y_pred[j * num_data + i].

class lightgbm.LGBMRanker(boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=10, max_bin=255, subsample_for_bin=200000, objective=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=1, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=-1, silent=True, **kwargs)

Bases: `lightgbm.sklearn.LGBMModel`

LightGBM ranker.

Construct a gradient boosting model.

- Parameters:
  - **boosting_type** (*string_,_ optional (__default="gbdt"_)_*) – 'gbdt', traditional Gradient Boosting Decision Tree. 'dart', Dropouts meet Multiple Additive Regression Trees. 'goss', Gradient-based One-Side Sampling. 'rf', Random Forest.
  - **num_leaves** (*int_,_ optional (__default=31_)_*) – Maximum tree leaves for base learners.
  - **max_depth** (*int_,_ optional (__default=-1_)_*) – Maximum tree depth for base learners, -1 means no limit.
  - **learning_rate** (*float_,_ optional (__default=0.1_)_*) – Boosting learning rate.
  - **n_estimators** (*int_,_ optional (__default=10_)_*) – Number of boosted trees to fit.
  - **max_bin** (*int_,_ optional (__default=255_)_*) – Number of bucketed bins for feature values.
  - **subsample_for_bin** (*int_,_ optional (__default=50000_)_*) – Number of samples for constructing bins.
  - **objective** (*string_,_ callable or None_,_ optional (__default=None_)_*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below). default: 'regression' for LGBMRegressor, 'binary' or 'multiclass' for LGBMClassifier, 'lambdarank' for LGBMRanker.
  - **min_split_gain** (*float_,_ optional (__default=0._)_*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
  - **min_child_weight** (*float_,_ optional (__default=1e-3_)_*) – Minimum sum of instance weight(hessian) needed in a child(leaf).

- **min_child_samples** (*int_,_ optional (__default=20_)_*) – Minimum number of data need in a child(leaf).
- **subsample** (*float_,_ optional (__default=1._)_*) – Subsample ratio of the training instance.
- **subsample_freq** (*int_,_ optional (__default=1_)_*) – Frequence of subsample, <=0 means no enable.
- **colsample_bytree** (*float_,_ optional (__default=1._)_*) – Subsample ratio of columns when constructing each tree.
- **reg_alpha** (*float_,_ optional (__default=0._)_*) – L1 regularization term on weights.
- **reg_lambda** (*float_,_ optional (__default=0._)_*) – L2 regularization term on weights.
- **random_state** (*int or None_,_ optional (__default=None_)_*) – Random number seed. Will use default seeds in c++ code if set to None.
- **n_jobs** (*int_,_ optional (__default=-1_)_*) – Number of parallel threads.
- **silent** (*bool_,_ optional (__default=True_)_*) – Whether to print messages while running boosting.
- ****kwargs** (*other parameters*) – Check http://lightgbm.readthedocs.io/en/latest/Parameters.html for more parameters. Note **kwargs is not supported in sklearn, it may cause unexpected issues.

n_features_

int – The number of features of fitted model.

classes_

array of shape = [n_classes] – The class label array (only for classification problem).

n_classes_

int – The number of classes (only for classification problem).

best_score_

dict or None – The best score of fitted model.

best_iteration_

int or None – The best iteration of fitted model if `early_stopping_rounds` has been specified.

objective_

string or callable – The concrete objective used while fitting this model.

booster_

Booster – The underlying Booster of this model.

evals_result_

dict or None – The evaluation results if `early_stopping_rounds` has been specified.

feature_importances_

array of shape = [n_features] – The feature importances (the higher, the more important the feature).

Note

A custom objective function can be provided for the `objective` parameter. In this case, it should have the signature `objective(y_true, y_pred) -> grad, hess` or `objective(y_true, y_pred, group) -> grad, hess`:

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class task)
```

The predicted values.

```
group: array-like
```

Group/query data, used for ranking task.

```
grad: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the gradient for each sample point.

```
hess: array-like of shape = [n_samples] or shape = [n_samples * n_classes]
(for multi-class task)
```

The value of the second derivative for each sample point.

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

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

Build a gradient boosting model from the training set (X, y).

- Parameters:
- **X** (*array-like or sparse matrix of shape = [__n_samples_,_ n_features_]_) – Input feature matrix.
- **y** (*array-like of shape = [__n_samples_]_) – The target values (class labels in classification, real numbers in regression).
- **sample_weight** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Weights of training data.
- **init_score** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Init score of training data.
- **group** (*array-like of shape = [__n_samples_]* or_ *None_,_ optional (__default=None_)_*) – Group data of training data.
- **eval_set** (*list or None_,_ optional (__default=None_)_*) – A list of (X, y) tuple pairs to use as a validation sets for early-stopping.
- **eval_names** (*list of strings or None_,_ optional (__default=None_)_*) – Names of eval_set.
- **eval_sample_weight** (*list of arrays or None_,_ optional (__default=None_)_*) – Weights of eval data.
- **eval_init_score** (*list of arrays or None_,_ optional (__default=None_)_*) – Init score of eval data.
- **eval_group** (*list of arrays or None_,_ optional (__default=None_)_*) – Group data of eval data.
- **eval_metric** (*string_,_ list of strings_,_ callable or None_,_ optional (__default="ndcg"_)_*) – If string, it should be a built-in evaluation metric to use. If callable, it should be a custom evaluation metric, see note for more details.
- **eval_at** (*list of int_,_ optional (__default=_[**1**]__)_*) – The evaluation positions of NDCG.
- **early_stopping_rounds** (*int or None_,_ optional (__default=None_)_*) – Activates early stopping. The model will train until the validation score stops improving. Validation error needs to decrease at least every `early_stopping_rounds` round(s) to continue training.
- **verbose** (*bool_,_ optional (__default=True_)_*) – If True and an evaluation set is used, writes the evaluation progress.
- **feature_name** (*list of strings or 'auto'_,_ optional (__default="auto"_)_*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.

- **categorical_feature** (*list of strings or int_, or_ *'auto'_,_ optional (__default="auto"_)_*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names (need to specify `feature_name` as well). If 'auto' and data is pandas DataFrame, pandas categorical columns are used.
- **callbacks** (*list of callback functions or None_,_ optional (__default=None_)_*) – List of callback functions that are applied at each iteration. See Callbacks in Python API for more information.
- Returns:
- **self** – Returns self.
- Return type:
- object

Note

Custom eval function expects a callable with following functions: `func(y_true, y_pred)`, `func(y_true, y_pred, weight)` or `func(y_true, y_pred, weight, group)`. Returns (eval_name, eval_result, is_bigger_better) or list of (eval_name, eval_result, is_bigger_better)

```
y_true: array-like of shape = [n_samples]
```

The target values.

```
y_pred: array-like of shape = [n_samples] or shape = [n_samples *
n_classes] (for multi-class)
```

The predicted values.

```
weight: array-like of shape = [n_samples]
```

The weight of samples.

```
group: array-like
```

Group/query data, used for ranking task.

```
eval_name: str
```

The name of evaluation.

```
eval_result: float
```

The eval result.

```
is_bigger_better: bool
```

Is eval result bigger better, e.g. AUC is bigger_better.

For multi-class task, the y_pred is group by class_id first, then group by row_id. If you want to get i-th row y_pred in j-th class, the access way is y_pred[j * num_data + i].

Callbacks

lightgbm.early_stopping(stopping_rounds, verbose=True)

Create a callback that activates early stopping.

Note

Activates early stopping. Requires at least one validation data and one metric. If there's more than one, will check all of them.

- Parameters:
- **stopping_rounds** (*int*) – The possible number of rounds without the trend occurrence.
- **verbose** (*bool_,_ optional (__default=True_)_*) – Whether to print message with early stopping information.
- Returns:
- **callback** – The callback that activates early stopping.
- Return type:
- function

lightgbm.print_evaluation(period=1, show_stdv=True)

Create a callback that prints the evaluation results.

- Parameters:
- **period** (*int_,_ optional (__default=1_)_*) – The period to print the evaluation results.
- **show_stdv** (*bool_,_ optional (__default=True_)_*) – Whether to show stdv (if provided).
- Returns:
- **callback** – The callback that prints the evaluation results every `period` iteration(s).
- Return type:
- function

lightgbm.record_evaluation(eval_result)

Create a callback that records the evaluation history into `eval_result`.

- Parameters:
- **eval_result** (*dict*) – A dictionary to store the evaluation results.
- Returns:
- **callback** – The callback that records the evaluation history into the passed dictionary.
- Return type:
- function

lightgbm.reset_parameter(**kwargs)

Create a callback that resets the parameter after the first iteration.

Note

The initial parameter will still take in-effect on first iteration.

- Parameters:
- **kwargs** (*value should be list or function*) – List of parameters for each boosting round or a customized function that calculates the parameter in terms of current number of round (e.g. yields learning rate decay). If list lst, parameter = lst[current_round]. If function func, parameter = func(current_round).
- Returns:
- **callback** – The callback that resets the parameter after the first iteration.
- Return type:
- function

Plotting

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, **kwargs)

Plot model's feature importances.

- Parameters:
- **booster** (*Booster or LGBMModel*) – Booster or LGBMModel instance which feature importance should be plotted.
- **ax** (*matplotlib.axes.Axes or None_,_ optional (__default=None_)_*) – Target axes instance. If None, new figure and axes will be created.

- **height** (*float_,_ optional (__default=0.2_)_*) – Bar height, passed to `ax.barh()`.
- **xlim** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Tuple passed to `ax.xlim()`.
- **ylim** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Tuple passed to `ax.ylim()`.
- **title** (*string or None_,_ optional (__default="Feature importance"_)_*) – Axes title. If None, title is disabled.
- **xlabel** (*string or None_,_ optional (__default="Feature importance"_)_*) – X-axis title label. If None, title is disabled.
- **ylabel** (*string or None_,_ optional (__default="Features"_)_*) – Y-axis title label. If None, title is disabled.
- **importance_type** (*string_,_ optional (__default="split"_)_*) – How the importance is calculated. If "split", result contains numbers of times the feature is used in a model. If "gain", result contains total gains of splits which use the feature.
- **max_num_features** (*int or None_,_ optional (__default=None_)_*) – Max number of top features displayed on plot. If None or <1, all features will be displayed.
- **ignore_zero** (*bool_,_ optional (__default=True_)_*) – Whether to ignore features with zero importance.
- **figsize** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Figure size.
- **grid** (*bool_,_ optional (__default=True_)_*) – Whether to add a grid for axes.
- ****kwargs** (*other parameters*) – Other parameters passed to `ax.barh()`.
- Returns:
- **ax** – The plot with model's feature importances.
- Return type:
- matplotlib.axes.Axes

lightgbm.plot_metric(booster, metric=None, dataset_names=None, ax=None, xlim=None, ylim=None, title='Metric during training', xlabel='Iterations', ylabel='auto', figsize=None, grid=True)

Plot one metric during training.

- Parameters:
- **booster** (*dict or LGBMModel*) – Dictionary returned from `lightgbm.train()` or LGBMModel instance.
- **metric** (*string or None_,_ optional (__default=None_)_*) – The metric name to plot. Only one metric supported because different metrics have various scales. If None, first metric picked from dictionary (according to hashcode).

- **dataset_names** (*list of strings or None_,_ optional (__default=None_)_*) – List of the dataset names which are used to calculate metric to plot. If None, all datasets are used.
- **ax** (*matplotlib.axes.Axes or None_,_ optional (__default=None_)_*) – Target axes instance. If None, new figure and axes will be created.
- **xlim** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Tuple passed to `ax.xlim()`.
- **ylim** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Tuple passed to `ax.ylim()`.
- **title** (*string or None_,_ optional (__default="Metric during training"_)_*) – Axes title. If None, title is disabled.
- **xlabel** (*string or None_,_ optional (__default="Iterations"_)_*) – X-axis title label. If None, title is disabled.
- **ylabel** (*string or None_,_ optional (__default="auto"_)_*) – Y-axis title label. If 'auto', metric name is used. If None, title is disabled.
- **figsize** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Figure size.
- **grid** (*bool_,_ optional (__default=True_)_*) – Whether to add a grid for axes.
- Returns:
- **ax** – The plot with metric's history over the training.
- Return type:
- matplotlib.axes.Axes

lightgbm.plot_tree(booster, ax=None, tree_index=0, figsize=None, graph_attr=None, node_attr=None, edge_attr=None, show_info=None)

Plot specified tree.

- Parameters:
- **booster** (*Booster* or *LGBMModel*) – Booster or LGBMModel instance to be plotted.
- **ax** (*matplotlib.axes.Axes or None_,_ optional (__default=None_)_*) – Target axes instance. If None, new figure and axes will be created.
- **tree_index** (*int_,_ optional (__default=0_)_*) – The index of a target tree to plot.
- **figsize** (*tuple of 2 elements or None_,_ optional (__default=None_)_*) – Figure size.
- **graph_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for the graph.
- **node_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for all nodes.
- **edge_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for all edges.

- **show_info** (*list or None_,_ optional (__default=None_)_*) – What information should be showed on nodes. Possible values of list items: ‘split_gain’, ‘internal_value’, ‘internal_count’, ‘leaf_count’.
- Returns:
- **ax** – The plot with single tree.
- Return type:
- matplotlib.axes.Axes

lightgbm.create_tree_digraph(booster, tree_index=0, show_info=None, name=None, comment=None, filename=None, directory=None, format=None, engine=None, encoding=None, graph_attr=None, node_attr=None, edge_attr=None, body=None, strict=False)

Create a digraph representation of specified tree.

Note

For more information please visit http://graphviz.readthedocs.io/en/stable/api.html#digraph.

- Parameters:
- **booster** (*Booster or LGBMModel*) – Booster or LGBMModel instance.
- **tree_index** (*int_,_ optional (__default=0_)_*) – The index of a target tree to convert.
  - **show_info** (*list or None_,_ optional (__default=None_)_*) – What information should be showed on nodes. Possible values of list items: ‘split_gain’, ‘internal_value’, ‘internal_count’, ‘leaf_count’.
- **name** (*string or None_,_ optional (__default=None_)_*) – Graph name used in the source code.
- **comment** (*string or None_,_ optional (__default=None_)_*) – Comment added to the first line of the source.
- **filename** (*string or None_,_ optional (__default=None_)_*) – Filename for saving the source. If None, `name` + ‘.gv’ is used.
- **directory** (*string or None_,_ optional (__default=None_)_*) – (Sub)directory for source saving and rendering.
- **format** (*string or None_,_ optional (__default=None_)_*) – Rendering output format (‘pdf’, ‘png’, …).
- **engine** (*string or None_,_ optional (__default=None_)_*) – Layout command used (‘dot’, ‘neato’, …).
- **encoding** (*string or None_,_ optional (__default=None_)_*) – Encoding for saving the source.
- **graph_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for the graph.

- **node_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for all nodes.
- **edge_attr** (*dict or None_,_ optional (__default=None_)_*) – Mapping of (attribute, value) pairs set for all edges.
- **body** (*list of strings or None_,_ optional (__default=None_)_*) – Lines to add to the graph body.
- **strict** (*bool_,_ optional (__default=False_)_*) – Whether rendering should merge multi-edges.
- Returns:
- **graph** – The digraph representation of specified tree.
- Return type:
- graphviz.Digraph

并行学习指南

这是一篇 LightGBM 的并行学习教程.

点击 快速入门 来学习怎样使用 LightGBM.

选择合适的并行算法

LightGBM 现已提供了以下并行学习算法.

Parallel Algorithm	How to Use
Data parallel	`tree_learner=data`
Feature parallel	`tree_learner=feature`
Voting parallel	`tree_learner=voting`

这些算法适用于不同场景,如下表所示:

	#data is small	#data is large
#feature is small	Feature Parallel	Data Parallel
#feature is large	Feature Parallel	Voting Parallel

在 optimization in parallel learning 了解更多并行算法的细节.

构建并行版本

默认的并行版本支持基于 socket 的并行学习.

如果你想构建基于 MPI 的并行版本, 请参考 安装指南.

准备工作

Socket 版本

它需要收集所有想要运行并行学习的机器的所有 IP 并且指定一个 TCP 端口号 (假设是 12345) , 更改防火墙使得这个端口可以被访问 (12345). 然后把这些 IP 和端口写入一个文件中 (假设是 `mlist.txt`), 如下所示:

```
machine1_ip 12345
machine2_ip 12345
```

MPI 版本

它需要收集所有想要运行并行学习机器的 IP (或 hostname) . 然后把这些IP写入一个文件中 (例如 `mlist.txt`) 如下所示:

```
machine1_ip
machine2_ip
```

Note: 对于 windows 用户, 需要安装 “smpd” 来开启 MPI 服务. 更多细节点击 here.

运行并行学习

Socket 版本

1. 在配置文件中编辑以下参数:

   `tree_learner=your_parallel_algorithm` , 在这里编辑 `your_parallel_algorithm` (e.g. feature/data) .

   `num_machines=your_num_machines` , 在这里编辑 `your_num_machines` (e.g. 4) .

   `machine_list_file=mlist.txt` , `mlist.txt` 在 准备工作 生成.

   `local_listen_port=12345` , `12345` 在 准备工作 分配.

2. 拷贝数据文件, 可执行文件, 配置文件和 `mlist.txt` 到所有机器.

3. 在所有机器上运行以下命令, 你需要更改 `your_config_file` 为真实配置文件.

   Windows: `lightgbm.exe config=your_config_file`

   Linux: `./lightgbm config=your_config_file`

MPI 版本

1. 在配置中编辑以下参数:

`tree_learner=your_parallel_algorithm`, 在这里编辑 `your_parallel_algorithm` (e.g. feature/data) .

`num_machines=your_num_machines`, 在这里编辑 `your_num_machines` (e.g. 4) .

2. 拷贝数据文件, 可执行文件, 配置文件和 `mlist.txt` 到所有机器.

   Note: MPI 需要运行在 **所有机器的相同路径上**.

3. 在机器上运行以下命令 (不需要运行所有机器), 需要更改 `your_config_file` 为真实的配置文件.

   Windows:

   ```
   mpiexec.exe /machinefile mlist.txt lightgbm.exe config=your_config_file
   ```

   Linux:

   ```
   mpiexec --machinefile mlist.txt ./lightgbm config=your_config_file
   ```

例子

- A simple parallel example

LightGBM GPU 教程

本文档的目的在于一步步教你快速上手 GPU 训练。

对于 Windows, 请参阅 GPU Windows 教程.

我们将用 Microsoft Azure cloud computing platform 上的 GPU 实例做演示， 但你可以使用具有现代 AMD 或 NVIDIA GPU 的任何机器。

GPU 安装

你需要在 Azure （East US, North Central US, South Central US, West Europe 以及 Southeast Asia 等区域都可用）上启动一个 `NV` 类型的实例 并选择 Ubuntu 16.04 LTS 作为操作系统。

经测试, `NV6` 类型的虚拟机是满足最小需求的, 这种虚拟机包括 ½ M60 GPU， 8 GB 内存, 180 GB/s 的内存带宽以及 4,825 GFLOPS 的峰值计算能力。 不要使用 `NC` 类型的实例，因为这些 GPU (K80) 是基于较老的架构 (Kepler).

首先我们需要安装精简版的 NVIDIA 驱动和 OpenCL 开发环境：

```
sudo apt-get update
sudo apt-get install --no-install-recommends nvidia-375
sudo apt-get install --no-install-recommends nvidia-opencl-icd-375 nvidia-
opencl-dev opencl-headers
```

安装完驱动以后需要重新启动服务器。

```
sudo init 6
```

大约 30 秒后，服务器可以重新运转。

如果你正在使用 AMD GPU, 你需要下载并安装 AMDGPU-Pro 驱动，同时安装 `ocl-icd-libopencl1` 和 `ocl-icd-opencl-dev` 两个包。

编译 LightGBM

现在安装必要的生成工具和依赖:

```
sudo apt-get install --no-install-recommends git cmake build-essential
libboost-dev libboost-system-dev libboost-filesystem-dev
```

`NV6` GPU 实例自带一个 320 GB 的极速 SSD，挂载在 `/mnt` 目录下。 我们把它作为我们的工作环境（如果你正在使用自己的机器，可以跳过该步）：

```
sudo mkdir -p /mnt/workspace
sudo chown $(whoami):$(whoami) /mnt/workspace
cd /mnt/workspace
```

现在我们可以准备好校验 LightGBM 并使用 GPU 支持来编译它:

```
git clone --recursive https://github.com/Microsoft/LightGBM
cd LightGBM
mkdir build ; cd build
cmake -DUSE_GPU=1 ..
# if you have installed the NVIDIA OpenGL, please using following instead
# sudo cmake -DUSE_GPU=1 -DOpenCL_LIBRARY=/usr/local/cuda/lib64/libOpenCL.so -
OpenCL_INCLUDE_DIR=/usr/local/cuda/include/ ..
make -j$(nproc)
cd ..
```

你可以看到有两个二进制文件生成了， `lightgbm` 和 `lib_lightgbm.so`

如果你正在 OSX 系统上编译，你可能需要在 `src/treelearner/gpu_tree_learner.h` 中移除 `BOOST_COMPUTE_USE_OFFLINE_CACHE` 宏指令以避免 Boost.Compute 中的冲突错误。

安装 Python 接口 (可选)

如果你希望使用 LightGBM 的 Python 接口，你现在可以安装它（同时包括一些必要的 Python 依赖包）：

```
sudo apt-get -y install python-pip
sudo -H pip install setuptools numpy scipy scikit-learn -U
cd python-package/
sudo python setup.py install --precompile
cd ..
```

你需要设置一个额外的参数 `"device" : "gpu"` （同时也包括其他选项如 `learning_rate`, `num_leaves`, 等等）来在 Python 中使用 GPU.

你可以阅读我们的 Python Package Examples 来获取更多关于如何使用 Python 接口的信息。

数据集准备

使用如下命令来准备 Higgs 数据集

```
git clone https://github.com/guolinke/boosting_tree_benchmarks.git
cd boosting_tree_benchmarks/data
```

```
wget "https://archive.ics.uci.edu/ml/machine-learning-databases/00280/
HIGGS.csv.gz"
gunzip HIGGS.csv.gz
python higgs2libsvm.py
cd ../..
ln -s boosting_tree_benchmarks/data/higgs.train
ln -s boosting_tree_benchmarks/data/higgs.test
```

现在我们可以通过运行如下命令来为 LightGBM 创建一个配置文件（请复制整段代码块并作为一个整体来运行它）：

```
cat > lightgbm_gpu.conf <<EOF
max_bin = 63
num_leaves = 255
num_iterations = 50
learning_rate = 0.1
tree_learner = serial
task = train
is_training_metric = false
min_data_in_leaf = 1
min_sum_hessian_in_leaf = 100
ndcg_eval_at = 1,3,5,10
sparse_threshold = 1.0
device = gpu
gpu_platform_id = 0
gpu_device_id = 0
EOF
echo "num_threads=$(nproc)" >> lightgbm_gpu.conf
```

我们可以通过在配置文件中设置 `device=gpu` 来使 GPU 处于可用状态。 默认将使用系统安装的第一个 GPU (`gpu_platform_id=0` 以及 `gpu_device_id=0`).

在 GPU 上运行你的第一个学习任务

现在我们可以准备开始用 GPU 做训练了！

首先我们希望确保 GPU 能够正确工作。 运行如下代码来在 GPU 上训练，并记录下 50 次迭代后的 AUC。

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train valid=higgs.test
objective=binary metric=auc
```

现在用如下代码在 CPU 上训练相同的数据集. 你应该能观察到相似的 AUC：

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train valid=higgs.test
objective=binary metric=auc device=cpu
```

现在我们可以不计算 AUC，每次迭代后进行 GPU 上的速度测试。

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train objective=binary
metric=auc
```

CPU 的速度测试:

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train objective=binary
metric=auc device=cpu
```

你可以观察到在该 GPU 上加速了超过三倍.

GPU 加速也可以用于其他任务/指标上（回归，多类别分类器，排序，等等）. 比如，我们可以在一个回归任务下训练 Higgs 数据集:

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train objective=regression_l2
metric=l2
```

同样地, 你也可以比较 CPU 上的训练速度:

```
./lightgbm config=lightgbm_gpu.conf data=higgs.train objective=regression_l2
metric=l2 device=cpu
```

进一步阅读

- GPU 优化指南和性能比较
- GPU SDK Correspondence and Device Targeting Table
- GPU Windows 教程

参考

如果您觉得 GPU 加速很有用，希望您在著作中能够引用如下文章；

Huan Zhang, Si Si and Cho-Jui Hsieh. “GPU Acceleration for Large-scale Tree Boosting.” arXiv: 1706.08359, 2017.

进阶主题

缺失值的处理

- LightGBM 通过默认的方式来处理缺失值，你可以通过设置 `use_missing=false` 来使其无效。
- LightGBM 通过默认的的方式用 NA (NaN) 去表示缺失值，你可以通过设置 `zero_as_missing=true` 将其变为零。
- 当设置 `zero_as_missing=false` （默认）时，在稀疏矩阵里 (和LightSVM) ，没有显示的值视为零。
- 当设置 `zero_as_missing=true` 时， NA 和 0 （包括在稀疏矩阵里，没有显示的值）视为缺失。

分类特征的支持

- 当使用本地分类特征，LightGBM 能提供良好的精确度。不像简单的 one-hot 编码，LightGBM 可以找到分类特征的最优分割。 相对于 one-hot 编码结果，LightGBM 可以提供更加准确的最优分割。
- 用 `categorical_feature` 指定分类特征 参考 Parameters 的参数 `categorical_feature`
- 首先需要转换为 int 类型，并且只支持非负数。 将其转换为连续范围更好。
- 使用 `min_data_per_group`, `cat_smooth` 去处理过拟合（当 `#data` 比较小，或者 `#category` 比较大）
- 对于具有高基数的分类特征(`#category` 比较大), 最好把它转化为数字特征。

LambdaRank

- 标签应该是 int 类型，较大的数字代表更高的相关性（例如：0：坏，1：公平，2：好，3：完美）。
- 使用 `label_gain` 设置增益（重量）的 `int` 标签。
- 使用 `max_position` 设置 NDCG 优化位置。

参数优化

- 参考 参数优化 .

并行学习

- 参考 并行学习指南 .

GPU 的支持

- 参考 GPU 教程 和 GPU Targets

GCC 用户的建议 (MinGW, *nix)

- 参考 gcc 建议.

LightGBM FAQ 常见问题解答

内容

关键问题

在使用 LightGBM 遇到坑爹的问题时（程序奔溃，预测结果错误，无意义输出...）,你应该联系谁?

如果你的问题不是那么紧急，可以把问题放到 Microsoft/LightGBM repository.

如果你的问题急需要解决，首先要明确你有哪些错误:

- 你认为问题会不会复现在 CLI（命令行接口），R 或者 Python上？
- 还是只会在某个特定的包（R 或者 Python）上出现?
- 还是会在某个特定的编译器（gcc 或者 MinGW）上出现?
- 还是会在某个特定的操作系统（Windows 或者 Linux）上出现?
- 你能用一个简单的例子复现这些问题吗?
- 你能（或者不能）在去掉所有的优化信息和在 debug 模式下编译 LightGBM 时复现这些问题吗?

当出现问题的时候，根据上述答案，随时可以@我们（不同的问题可以@不同的人，下面是各种不同类型问题的负责人），这样我们就能更快地帮助你解决问题。

- @guolinke (C++ code / R-package / Python-package)
- @chivee (C++ code / Python-package)
- @Laurae2 (R-package)
- @wxchan (Python-package)
- @henry0312 (Python-package)
- @StrikerRUS (Python-package)

- @huanzhang12 (GPU support)

记住这是一个免费的/开放的社区支持，我们可能不能做到全天候的提供帮助.

LightGBM

- **问题 1**：我可以去哪里找到关于LightGBM参数的更多详细内容?
- **方法 1**：可以看一下这个 Parameters and Laurae++/Parameters 网站。

- **问题 2**：在一个有百万个特征的数据集中，（要在很长一段时间后才开始训练或者）训练根本没有开始。
- **方法 2**：对 `bin_construct_sample_cnt` 用一个较小的值和对 `min_data` 用一个较大的值。

- **问题 3**：当在一个很大的数据集上使用LightGBM，我的电脑会耗尽内存。
- **方法 3**：很多方法啊：将 `histogram_pool_size` 参数设置成你想为LightGBM分配的MB(histogram_pool_size + dataset size = approximately RAM used), 减少 `num_leaves` 或减少 `max_bin` （点这里 Microsoft/LightGBM#562）。

- **问题 4**：我使用Windows系统。我应该使用Visual Studio或者MinGW编译LightGBM吗?
- **方法 4**：推荐使用 Visual Studio，因为它的性能更好。

- **问题 5**：当使用LightGBM，我每次运行得到的结果都不同（结果不能复现）。
- **方法 5**：这是一个很正常的问题，我们/你也无能为力。 你可以试试使用 `gpu_use_dp = true` 来复现结果（点这里 Microsoft/LightGBM#560）。 你也可以使用CPU的版本试试。

- **问题 6**：Bagging在改变线程的数量时，是不能复现的。
- **方法 6**：由于LightGBM Bagging是多线程运行的，它的输出依赖于使用线程的数量。 There is no workaround currently。

- **问题 7**：我试过使用随机森林模式，LightGBM崩溃啦!
- **方法 7**：这是设计的问题。 你必须使用 `bagging_fraction` 和 `feature_fraction` 与1不同，要和 `bagging_freq` 结合使用。 看这个例子 this thread。

- **问题 8**：当在一个很大的数据集上和很多核心系统使用LightGBMWindows系统时，CPU不是满负荷运行（例如只使用了10%的CPU）。
- **方法 8**：请使用 Visual Studio， 因为Visual Studio可能 10x faster than MinGW，尤其是在很大的树上。

R 包

- **问题 1**：在训练先前的LightGBM模型时一个错误出现后，任何使用LightGBM的训练命令都不会起作用。
- **方法 1**：在R控制台中运行 `lgb.unloader(wipe = TRUE)`，再重新创建LightGBM数据集（这会消除所有与LightGBM相关的变量）。 由于这些指针，选择不去消除这些变量不会修复这些错误。 这是一个已知的问题: Microsoft/LightGBM#698。

- **问题 2**：我使用过 `setinfo` ,试过打印我的 `lgb.Dataset` ,结果R控制台无响应。
- **方法 2**：在使用 `setinfo` 后避免打印 `lgb.Dataset`. 这是一个已知的bug：Microsoft/LightGBM#539。

Python 包

- **问题 1**：当从GitHub使用 `python setup.py install` 安装，我看到如下错误信息。

```
error: 错误: 安装脚本指定绝对路径:
/Users/Microsoft/LightGBM/python-package/lightgbm/../../lib_lightgbm.so
setup()参数必须 *一直* 是/-分离路径相对于setup.py目录， *从不* 是绝对路径。
```

- **方法 1**：这个错误在新版本中应该会被解决。 如果你还会遇到这个问题，试着在你的Python包中去掉 `lightgbm.egg-info` 文件夹，再重装一下， 或者对照一下这个 this thread on stackoverflow。

- **问题 2**：我看到错误信息如下

```
在构建数据集前不能 get/set label/weight/init_score/group/num_data/
num_feature。
```

但是我已经使用下面的代码构建数据集

```
train = lightgbm.Dataset(X_train, y_train)
```

或如下错误信息

```
在释放原始数据后，不能设置predictor/reference/categorical特征。可以在创建数据集时设置free_raw_data=False避免上面的问题。
```

- **方法2**: 因为LightGBM创建bin mappers来构建树，在一个Booster内的train和valid数据集共享同一个bin mappers，类别特征和特征名等信息，数据集对象在创建Booster时候被创建。如果你设置 `free_raw_data=True` (默认)，原始数据（在Python数据结构中的）将会被释放。 所以，如果你想要：
  - 在创建数据集前get label(or weight/init_score/group)，这和get `self.label` 操作相同。
  - 在创建数据集前set label(or weight/init_score/group)，这和 `self.label=some_label_array` 操作相同。
  - 在创建数据集前get num_data(or num_feature)，你可以使用 `self.data` 得到数据，然后如果你的数据是 `numpy.ndarray`，使用一些类似 `self.data.shape` 的代码。
  - 在构建数据集之后set predictor(or reference/categorical feature)，你应该设置 `free_raw_data=False` 或使用同样的原始数据初始化数据集对象。

Development Guide

Algorithms

Refer to Features to understand important algorithms used in LightGBM.

Classes and Code Structure

Important Classes

Class	Description
`Application`	The entrance of application, including training and prediction logic
`Bin`	Data structure used for store feature discrete values(converted from float values)
`Boosting`	Boosting interface, current implementation is GBDT and DART
`Config`	Store parameters and configurations
`Dataset`	Store information of dataset
`DatasetLoader`	Used to construct dataset
`Feature`	Store One column feature
`Metric`	Evaluation metrics
`Network`	Network interfaces and communication algorithms
`ObjectiveFunction`	Objective function used to train
`Tree`	Store information of tree model
`TreeLearner`	Used to learn trees

Code Structure

Path	Description
./include	Header files
./include/utils	Some common functions
./src/application	Implementations of training and prediction logic
./src/boosting	Implementations of Boosting
./src/io	Implementations of IO relatived classes, including `Bin`, `Config`, `Dataset`, `DatasetLoader`, `Feature` and `Tree`
./src/metric	Implementations of metrics
./src/network	Implementations of network functions
./src/objective	Implementations of objective functions
./src/treelearner	Implementations of tree learners

Documents API

Refer to docs README.

C API

Refere to the comments in c_api.h.

High Level Language Package

See the implementations at Python-package and R-package.

Questions

Refer to FAQ.

Also feel free to open issues if you met problems.