从 C++ 代码中构建共享库（针对 linux/osx 的是 `libxgboost.so` 然后针对 windows 的是 `libxgboost.dll` ）

一、下载源码

git clone --recursive https://github.com/dmlc/xgboost

对于window用户，打开git shell,输入以下命令

```
git submodule init

git submodule update

mkdir build

cd build

cmake .. -G"Visual Studio 15 2017 Win64"
```

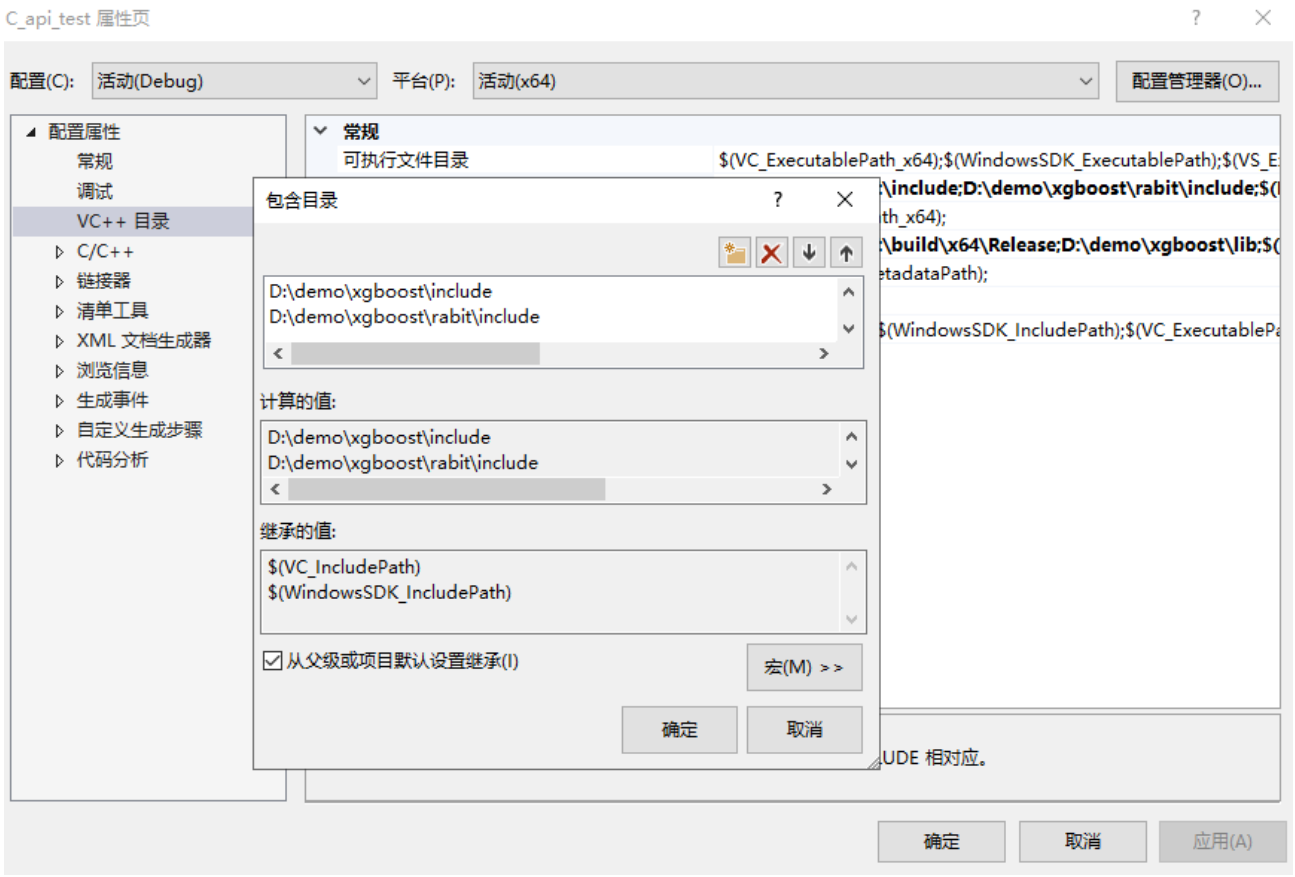
生成工程文件 .sln，使用vs2017打开xgboost.sln，点击Build,默认只有7项勾选，我们选择将所有的项勾选，点击Build Solution开始编译。生成解决方案

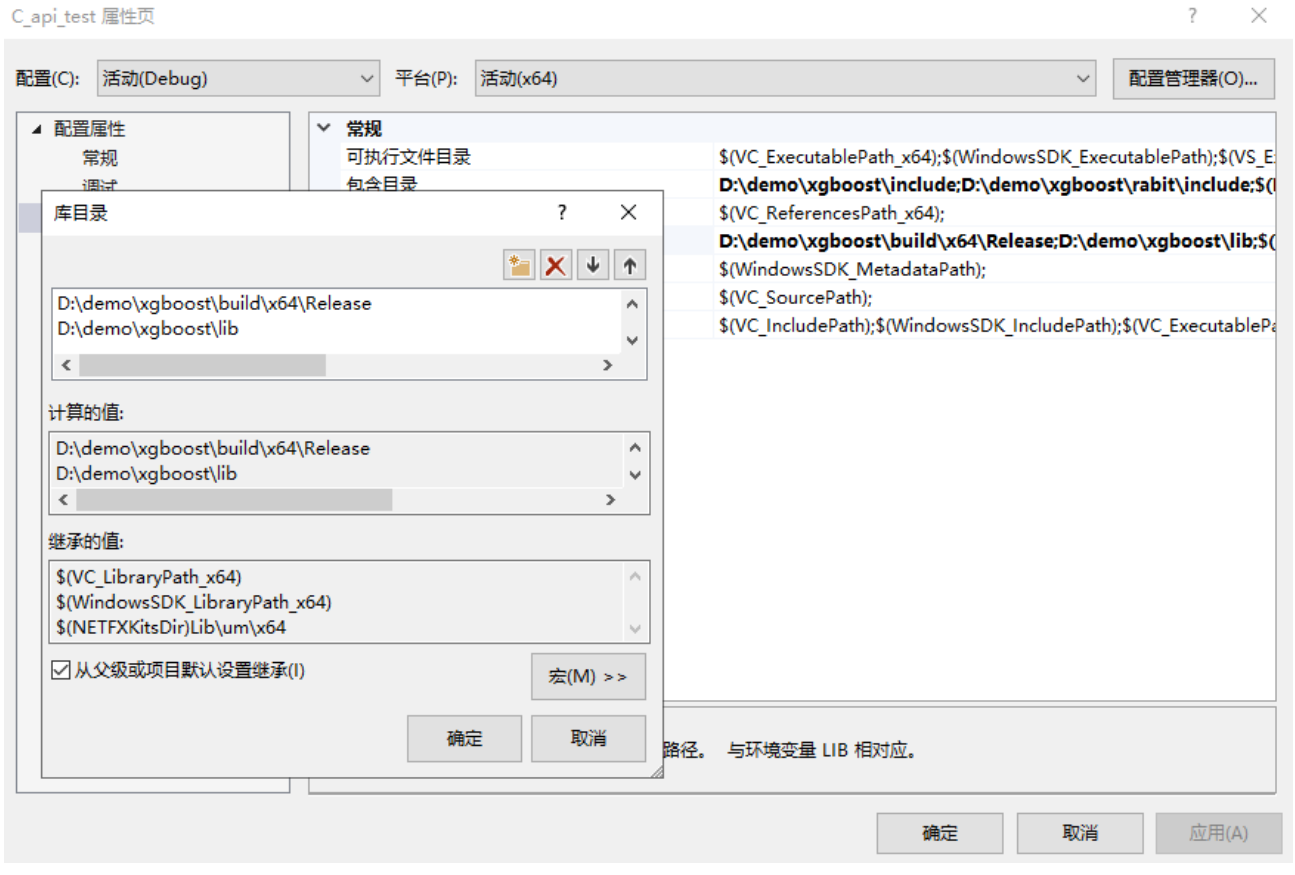
编译完成之后，将/xgboost/lib/下的xgboost.dll复制到C:\Windows\System32和C:\Windows\SysWOW64目录下（不执行此步可能会出现“应用程序无法正常启动 0x0000007b”错误）

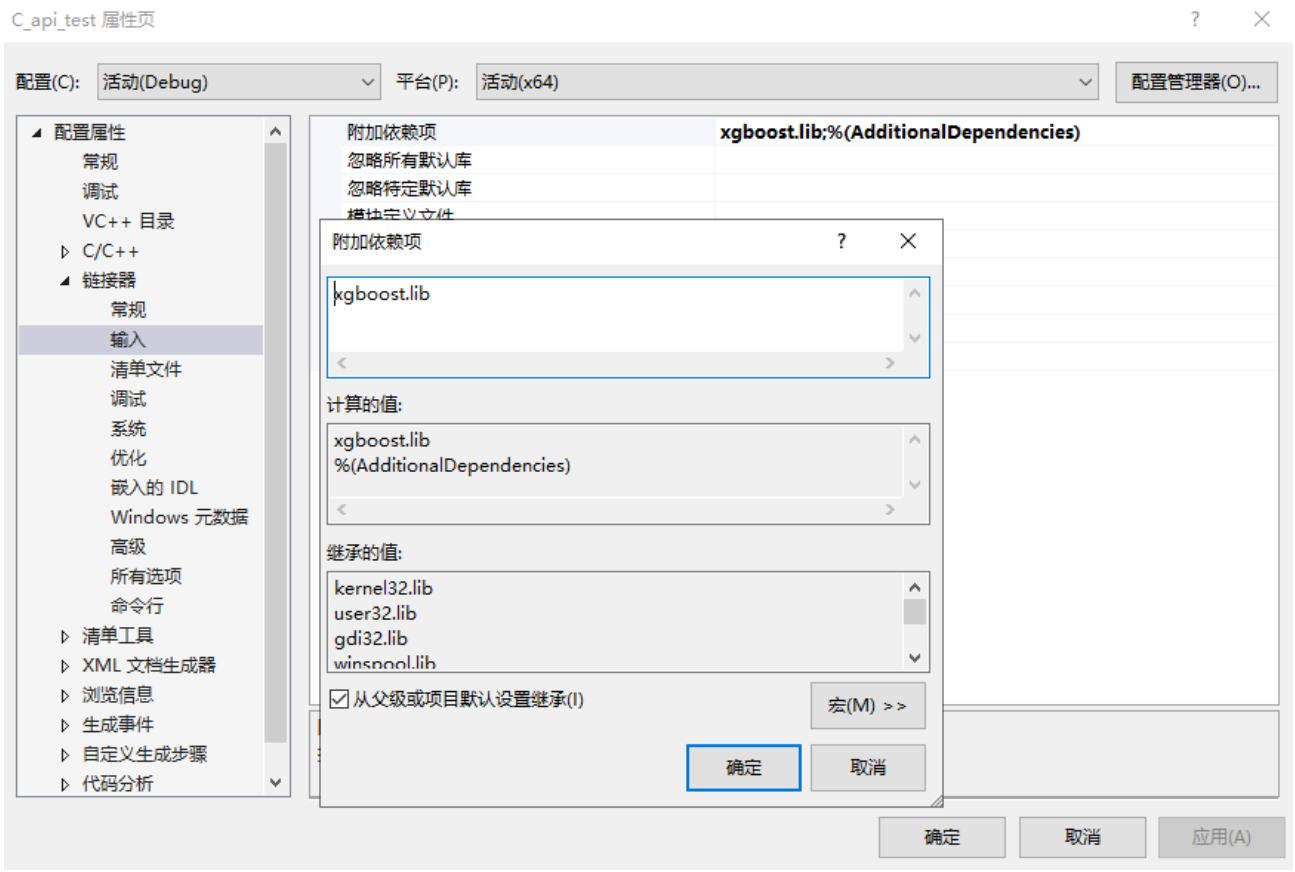
创建一个新的win64控制台应用程序tC_api_test进行测试
（**debug和release都可以进行以下配置**）

二、配置依赖库：

在创建好的项目上右键->Properties,配置包含目录和库目录，再在Linker->input中添加xgboost.lib与rabit.lib依赖


C_api_test 属性页
配置(C): 活动(Debug)
平台(P): 活动(x64)
配置管理器(O)...
配置属性
常规
调试
VC++ 目录
C/C++
链接器
清单工具
XML 文档生成器
浏览信息
生成事件
自定义生成步骤
代码分析
常规
可执行文件目录
$(VC_ExecutablePath_x64);$(WindowsSDK_ExecutablePath);$(VS_E
包含目录
D:\demo\xgboost\include
D:\demo\xgboost\rabit\include
计算的值:
D:\demo\xgboost\include
D:\demo\xgboost\rabit\include
继承的值:
$(VC_IncludePath)
$(WindowsSDK_IncludePath)
从父级或项目默认设置继承(I)
宏(M) >>
确定
取消
确定
取消
应用(A)


C_api_test 属性页
配置(C): 活动(Debug)
平台(P): 活动(x64)
配置管理器(O)...
配置属性
常规
常规
可执行文件目录
$(VC_ExecutablePath_x64);$(WindowsSDK_ExecutablePath);$(VS_E
包含目录
D:\demo\xgboost\include;D:\demo\xgboost\rabit\include;$(
$(VC_ReferencesPath_x64);
D:\demo\xgboost\build\x64\Release;D:\demo\xgboost\lib;$(
$(WindowsSDK_MetadataPath);
$(VC_SourcePath);
$(VC_IncludePath);$(WindowsSDK_IncludePath);$(VC_ExecutablePa
库目录
D:\demo\xgboost\build\x64\Release
D:\demo\xgboost\lib
计算的值:
D:\demo\xgboost\build\x64\Release
D:\demo\xgboost\lib
继承的值:
$(VC_LibraryPath_x64)
$(WindowsSDK_LibraryPath_x64)
$(NETFXKitsDir)Lib\um\x64
从父级或项目默认设置继承(I)
宏(M) >>
确定
取消
路径。 与环境变量 LIB 相对应。
确定
取消
应用(A)



三、添加代码：

```
#include<iostream>
#include<xgboost/c_api.h>
void test();
int main()
{
    test();
    system("pause");
    return 0;
}

void test()
{
    //创建训练数据和标签
    const int cols = 3, rows = 5;
    float train[rows][cols];
    for (int i = 0; i < rows; i++)
```

输出：

四、其他编译方式：

（未尝试）
下载并编译xgboost本身，放到和hw.c一个目录：

git clone --recursive https://github.com/dmlc/xgboost
cd xgboost; make -j4

编译指令：g++ hw.c -lxgboost/include -lxgboost/rabit/include xgboost/lib/libxgboost.a xgboost/rabit/lib/librabit.a xgboost/dmlc-core/libdmlc.a -fopenmp

五、数据

1. 数据打包格式

```
line1: 算子1特征值+算子1运行时间
line2:
......
```

2. 数据形式

单算子输入，有利于增加训练数据量和样本丰富度。
输入特征的取值是**缓存、数学计算、DMA次数和卷积次数，输出单算子运行的时间**

3. 标签映射表

（用于增加模型的可解释性）

```
0                2dconv
1                dwconv
2                add
......
```

六、C_api部分函数说明：

训练相关：

XGB_DLL int XGBoosterCreate	创建一个xgboost学习器
输入参数	
const DMatrixHandle dmats[]	被设置为缓存的多个矩阵（储存训练数据的矩阵）
bst_ulong len	矩阵的个数（几个训练矩阵）
BoosterHandle * out	指向booster的句柄
XGB_DLL int XGDMatrixCreateFromMat	从密集的矩阵中创建矩阵内容
输入参数	
const float * data	指向数据空间的指针
bst_ulong nrow	行数
bst_ulong ncol	列数
float missing	哪个值表示缺失的值
DmatrixHandle* out	创建的dmatrix
XGB_DLL int XGDMatrixSetFloatInfo	将float vector设置为info中的内容（设置标签信息）
输入参数	
DmatrixHandle* handle	数据矩阵的一个实例
const char* field	字符串，可以是label，weight
const float* array	指向float vector的指针，储存训练标签
bst_ulong len	指针的长度，标签个数
XGB_DLL int XGDMatrixGetFloatInfo	从矩阵中获取浮点信息向量（获取标签信息）
输入参数	

const DmatrixHandle handle	数据矩阵的一个实例
const char* field	字符串，可以是label，weight
bst_ulong* out_len	返回的结果长度
const float** out_dptr	指向结果的指针
XGB_DLL int XGBoosterSetParam	设置参数
输入参数	
BoosterHandle handle	句柄
const char* name	参数名
const char* value	参数值
XGB_DLL int XGBoosterUpdateOneIter	使用训练数据在一个回合内更新模型
输入参数	
BoosterHandle handle	句柄
int iter	当前迭代的回合数
DMatrixHandle dtrain	训练数据
XGB_DLL int XGBoosterPredictFromDMa	
输入参数	
BoosterHandle handle	booster句柄
DMatrixHandle dmat	DMatrix句柄
char const* c_json_config	JSON格式的字符串编码预测配置， JSON对象中有以下可用字段:"type": [0, 6] 0:正常的预测 1:输出权值累积值 2:预测各个特征的贡献 3:预测近似贡献 4:预测特征交互作用 5:预测近似的特征交互 6: predict leaf "training": bool 是否使用预测函数作为训练循环的一部分。不用于窗口显示预测。

bst_ulong const** out_shape	输出预测的形状
bst_ulong* out_dim	输出预测的维度
float const** out_result	储存预测值的缓存区
XGB_DLL int XGBoosterPredict	
输入参数	
BoosterHandle handle	booster句柄
DMatrixHandle dmat	DMatrix句柄
int option_mask	在预测中选择的位掩码，可能的值 1:输出余量而不是转换后的值 2:输出树的叶索引而不是叶值，注意每棵树的叶索引是唯一的 4:输出特征对单个预测的贡献
unsigned ntree_limit	限制用于预测的树的数量，这只对增强的树有效，当参数设置为0时，我们将使用所有的树
int training	是否将预测函数用作训练循环的一部分
bst_ulong* out_len	用于存储返回结果的长度
const float** out_result	用于设置指向数组的指针
XGB_DLL int XGBoosterDumpModel	转存模型
输入参数	
BoosterHandle handle	句柄
const char * fmap	可以是空字符
int with_states	是否使用统计信息dump
bst_ulong* out_len	输出阵列的长度
const char*** out_dump_array	表示每个模型存储的指针
XGB_DLL int XGBoosterLoadModel	从现有文件加载模型
输入参数	
BoosterHandle handle	句柄
const char * fname	文件地址或文件名

XGB_DLL int XGBoosterSaveModel	将模型保存到现有文件中
输入参数	
BoosterHandle handle	句柄
const char * fname	文件地址或文件名

XGB_DLL int XGBoosterCreate	创建xgboost学习器
输入参数	
const DmatrixHandle dmats[]	被设置为缓存的矩阵
bst_ulong len	dmats的长度
BoosterHandle* out	结果booster的句柄

调试事项：

XGBoost调试

Sklearn提供的常用数据集

- 一、自带的小数据集（packageddataset）：sklearn.datasets.load_
  - 鸢尾花数据集：load_iris（）：用于分类任务的数据集
  - 手写数字数据集： load_digits（）:用于分类任务或者降维任务的数据集
  - 乳腺癌数据集load-barest-cancer（）：简单经典的用于二分类任务的数据集
  - 糖尿病数据集：load-diabetes（）：经典的用于回归认为的数据集，值得注意的是，这10个特征中的每个特征都已经被处理成0均值，方差归一化的特征值。
  - 波士顿房价数据集：load-boston（）：经典的用于回归任务的数据集
  - 体能训练数据集： load-linnerud（）：经典的用于多变量回归任务的数据集。
- 可在线下载的数据集：sklearn.datasets.fetch_，一般规模较大（DownloadedDataset）
  - fetch_olivetti_faces(data_home=None, shuffle=False, random_state=0,download_if_missing=True)：Olivetti 脸部图片数据集。
- 计算机生成的数据集（GeneratedDataset）：sklearn.datasets.make_
  - make_blobs：多类单标签数据集，为每个类分配一个或多个正太分布的点集
  - make_classification：多类单标签数据集，为每个类分配一个或多个正太分布的点集，提供了为数据添加噪声的方式，包括维度相关性，无效特征以及冗余特征等
  - make_gaussian-quantiles：将一个单高斯分布的点集划分为两个数量均等的点集，作为两类
  - make_hastie-10-2：产生一个相似的二元分类数据集，有10个维度
  - make_circle和make_moom产生二维二元分类数据集来测试某些算法的性能，可以为数据集添加噪声，可以为二元分类器产生一些球形判决界面的数据。
- 四、svmlight/libsvm格式的数据集:sklearn.datasets.load_svmlight_file(...)
  - from sklearn.datasets importload_svmlight_file
  - x_train,y_train=load_svmlight_file("/path/to/train_dataset.txt","")#如果要加在多个数据的时候，可以用逗号隔开
  - svmlight/libsvm的每一行样本的存放格式：
- 五、data.org在线下载获取的数据集:sklearn.datasets.fetch_mldata(...)
  - 子主题 from sklearn.datasets.mldata import fetch_mldata
    import tempfile
    test_data_home = tempfile.mkdtemp()
    iris = fetch_mldata('iris', data_home=test_data_home)
    print(iris);print(iris.target.shape);print(iris.data.shape)

XGBoost参数

- 通用参数
  - booster
    - gbtree，dart使用基于树的模型
    - gblinear 使用线性函数
  - verbosity
    - 打印信息的等级
  - nthread
    - 运行XGBoost的并行计算线程
  - num_pbuffer
    - 预测缓冲区大小，XGBoost自动设置
  - num_feature
    - 最大特征维度，XGBoost自动设置
- Tree Booster参数
  - eta：[0-1]
    - 在更新中使用步长收缩，以防止过拟合。在每一步推进后，我们可以直接得到新特征的权值，eta缩小了特征权值，使得推进过程更加保守。
  - gamma：[default=0, alias: min_split_loss]
    - 在树的叶节点上进行进一步分区所需的最小损失减少。gamma越大，算法就越保守。
  - max_depth：[default=6]
    - 树的最大深度。增加这个值将使模型更复杂，更有可能过拟合。0只在lossguided growth policy中接受，当tree_method设置为hist或gpu_hist时，它表示没有深度限制。要注意，XGBoost在训练深度树时会大量消耗内存。
  - min_child_weight：[default=1]
    - 叶子节点所需要的最小样本实例。如果树分区步骤的结果是一个叶子节点的样本小于min_child_weight，那么构建过程将放弃进一步的分裂。在线性回归任务中，这只对应于每个节点中所需的最小样本数。min_child_weight值越大，算法保守性越强。
  - max_delta_step：[default=0]
    - 参数限制每棵树权重改变的最大步长。如果这个参数的值为0，那就意味着没有约束。如果它被赋予了某个正值，那么它会让这个算法更加保守。通常，这个参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。
  - subsample：[default=1]
    - 参数控制每棵树随机采样的比例。 减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。典型值：0.5-1
  - sampling_method：[default=uniform]
    - 对训练实例进行抽样的方法
      - uniform:
        - 每个训练实例被选中的概率相等
      - gradient_based:
        - 每个训练实例的选择概率与梯度的正则化绝对值成正比。注意，只有当tree_method设置为gpu_hist时，才支持这种采样方法;其他树形方法只支持uniform采样。
  - colsample_bytree, colsample_bylevel, colsample_bynode：[default=1]
    - 这是列子抽样的一系列参数。
      - colsample_bytree
        - 控制每棵随机采样的列数的占比(每一列是一个特征)。 典型值：0.5-1
      - colsample_bylevel
        - 控制树的每一级对列数的采样的占比。
      - colsample_bynode
        - 每一次分裂的列采样占比
  - lambda：[default=1, alias: reg_lambda]
    - L2正则化项
  - alpha：[default=0, alias: reg_alpha]
    - L1正则化项
- 命令行参数
  - num_round
    - boosting的回合数
  - data
    - 训练数据路径
  - test:data
    - 测试数据路径
  - save_period
- 学习任务参数
  - objective：[default=reg:squarederror]
    - reg:squarederror
      - 平方损失回归
    - reg:squaredlogerror
      - 平方log损失回归
      - 所有输入标签必须大于-1
    - reg:logistic
      - 逻辑回归
    - reg:pseudohubererror
      - 基于Pseudo Huber损失的回归，一种替代绝对损失的二次可微分方法
    - binary:logistic
      - 二元分类的逻辑回归，输出概率
    - binary:logitraw
      - 逻辑回归二值分类，输出逻辑转换前的分数
    - binary:hinge
      - 二值分类的hinge损失。预测值为0或1，而不是生成概率
    - count:poisson
      - 计数数据的泊松回归，输出泊松分布的均值
      - max_delta_step
        - 在泊松回归中默认设置为0.7(用于保障优化)
    - multi:softmax
      - 多分类
      - num_class
    - multi:softprob
      - 与softmax相同，但输出一个 ndata * nclass的向量，可以进一步重塑为ndata * nclass矩阵。结果包含了每个类中每个数据点的预测概率。
    - 不常用
      - survival:cox
      - survival:aft
      - rank:pairwise
      - rank:ndcg
      - rank:map
      - reg:gamma
      - reg:tweedie
  - base_score：[default=0.5]
    - 所有实例的初始预测得分，全局偏差
  - eval_metric：[default according to objective]
    - 验证数据的评价指标，默认的指标将根据objective分配（rmse用于回归， logloss用于分类，平均精度用于排名）
  - seed：[default=0]
  - seed_per_iteration：[default=false]
    - 通过迭代器编号确定PRNG的种子，这个选项将在分布式模式下自动打开。
- 预测
  - 预测参数
    - output_margin
      - 如果严格形状为True，除了multi:softmax由于跃落变换而具有等效的multi:softprobb输出，都输出二维数组；如果严格形状设置为False，那么输出可以有1或2维，这取决于使用的模型。
    - preds_contrib
      - 输出是一个3-dim数组，形状为(行、组、列+1)。是否使用approx_contribs并不会改变输出形状。如果没有设置严格形状参数，它可以是2维或3维数组，这取决于是否使用多类模型。
    - preds_interactions
      - 行，组，列+1，列+1)作为形状。与预测贡献的情况一样，是否使用approx_contribs并不会改变输出形状。如果严格形状设置为False，它可以有3或4个维，这取决于基础模型。
    - pred_leaf
      - 输出是一个4元数组，形状为(n_samples, n_iterations, n_classes, n_trees_in_forest)。N_trees_in_forest是在训练期间由num_parallel_tree指定的。当strict shape设置为False时，输出是一个2输的数组，最后3输连为1。如果最后一个维度等于1，它也会被删除。在scikit学习界面中使用apply方法时，默认为False。
  - Early Stopping

调参步骤

XGBoost有一个很有用的函数"cv"，这个函数可以在每一次迭代中使用交叉验证，并返回理想的决策树数量。

- 第一步：确定学习速率和tree_based 参数调优的估计器数目：
  - 确定学习速率以及对应的树的数量，这个数和学习率是用于调参时候的值，避免过小的学习率和过多的树导致调参速度慢
- 第二步： max_depth 和 min_child_weight 参数调优
  - 对最终结果有很大的影响
  - 进行高负荷的栅格搜索(grid search)：可以先粗分后细分
    - param_test1 = {
      'max_depth':range(3,10,2),
      'min_child_weight':range(1,6,2)
      }
    - ({mean: 0.83690, std: 0.00621, params: {'max_depth': 3, 'min_child_weight': 1},
      mean: 0.83728, std: 0.00858, params: {'max_depth': 3, 'min_child_weight': 3},
      mean: 0.83713, std: 0.00847, params: {'max_depth': 3, 'min_child_weight': 5},
      mean: 0.84051, std: 0.00748, params: {'max_depth': 5, 'min_child_weight': 1},
      mean: 0.84112, std: 0.00595, params: {'max_depth': 5, 'min_child_weight': 3},
      mean: 0.84123, std: 0.00619, params: {'max_depth': 5, 'min_child_weight': 5},
      mean: 0.83772, std: 0.00518, params: {'max_depth': 7, 'min_child_weight': 1},
      mean: 0.83673, std: 0.00579, params: {'max_depth': 7, 'min_child_weight': 3},
      mean: 0.83658, std: 0.00355, params: {'max_depth': 7, 'min_child_weight': 5},
      mean: 0.82695, std: 0.00622, params: {'max_depth': 9, 'min_child_weight': 1},
      mean: 0.82999, std: 0.00565, params: {'max_depth': 9, 'min_child_weight': 3},
      mean: 0.83211, std: 0.00707, params: {'max_depth': 9, 'min_child_weight': 5}},
      {'max_depth': 5, 'min_child_weight': 5},
      0.8412329302215709)
- 第三步：gamma参数调优
  - 调完后可以再确定一次树的深度
- 第四步：调整subsample 和 colsample_bytree 参数
  - 先粗后细
- 第五步：正则化参数调优 reg_alpha和reg_lambda(这里只调了 reg_alpha)
  - 可能会降低得分
- 第六步：降低学习率
  - 使用较低的学习速率，以及使用更多的决策树