

1、Xgboost介绍

1.1、Xgboost概述

XGBoost是陈天奇等人开发的一个开源机器学习项目，高效地实现了GBDT算法并进行了算法和工程上的许多改进，被广泛应用在Kaggle竞赛及其他许多机器学习竞赛中并取得了不错的成绩。

1.2、青出于蓝

说到XGBoost，不得不提GBDT(Gradient Boosting Decision Tree)。因为XGBoost本质上还是一个GBDT，但是力争把速度和效率发挥到极致，所以叫X (Extreme) GBoosted。两者都是boosting方法。

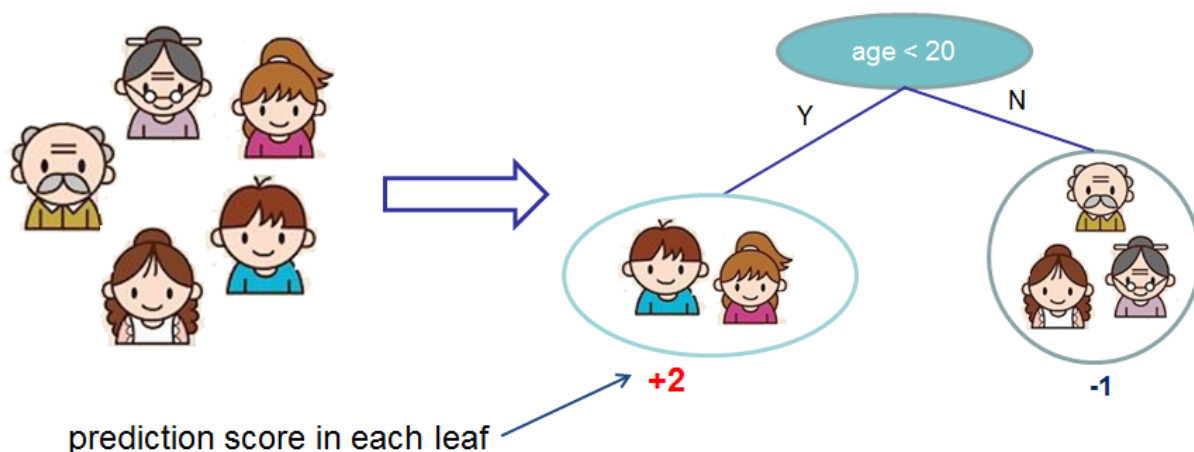
2、Xgboost树的定义

2.1、构造决策树

先来举个例子，我们要预测一家人对电子游戏的喜好程度，考虑到年轻和年老相比，年轻更可能喜欢电子游戏，以及男性和女性相比，男性更喜欢电子游戏，故先根据年龄大小区分小孩和大人，然后再通过性别区分开是男是女，逐一给各人在电子游戏喜好程度上打分，如下图所示。

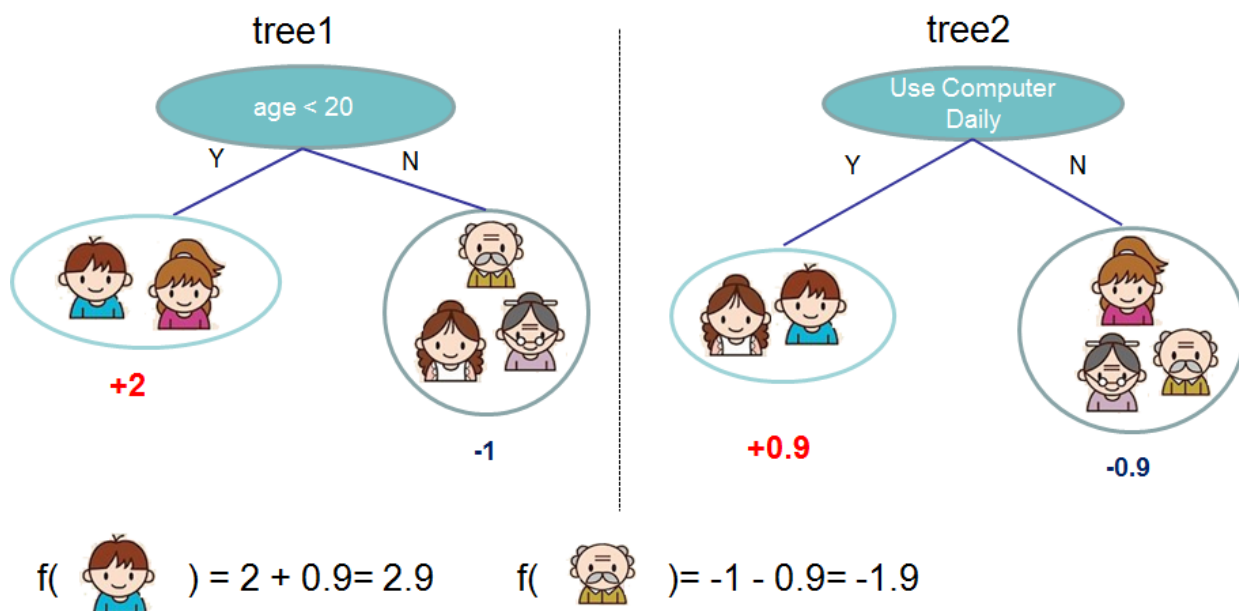
Input: age, gender, occupation, ...

Like the computer game X



2.2、决策树集成

就这样，训练出了2棵树tree1和tree2，类似之前gbdt的原理，两棵树的结论累加起来便是最终的结论，所以小孩的预测分数就是两棵树中小孩所落到的结点的分数相加： $2 + 0.9 = 2.9$ 。爷爷的预测分数同理： $-1 + (-0.9) = -1.9$ 。具体如下图所示：



恩，你可能要拍案而起了，惊呼，这不是跟之前介绍的GBDT乃异曲同工么？

事实上，如果不考虑工程实现、解决问题上的一些差异，XGBoost与GBDT比较大的不同仅仅在于目标函数的定义。

3、Xgboost目标函数

3.1、目标函数方程

对于Boosting算法我们知道，是将多个弱分类器的结果结合起来作为最终的结果来进行输出。 $f_t(x_i)$ 为第 t 棵树的输出结果， $\hat{y}_i^{(t)}$ 是模型当前的输出结果， y_i 是实际的结果。

那么：

$$\hat{y}_i^{(t)} = \sum_{t=1}^t f_t(x_i)$$

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

XGBoost的目标函数如下图所示：

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i) + \sum_{i=1}^t \Omega(f_t)$$

- 训练损失

$$\sum_{i=1}^n L(y_i, \hat{y}_i)$$

- 常见损失函数

平方损失函数： $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$

逻辑回归损失函数：

$$l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$$

- 树的复杂度

$$\sum_{i=1}^t \Omega(f_i)$$

Xgboost包含多棵树，定义每棵树的复杂度：

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

其中 T 为叶子节点的个数， w_j 为叶子节点向量的模。 γ 表示节点切分的难度， λ 表示L2正则化系数。

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{i=1}^t \Omega(f_i)$$

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \sum_{i=1}^{t-1} \Omega(f_i)$$

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

$\sum_{i=1}^{t-1} \Omega(f_i)$ 为前 t-1 棵树的复杂度，是常数项，求导时可忽略。目标函数简化为：

$$Obj^{(t)} = \sum_{i=1}^n L(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t)$$

3.2、目标函数泰勒展开

泰勒展开近似目标函数：

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

令：

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y, \hat{y}^{(t-1)})$$

$$h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y, \hat{y}^{(t-1)})$$

则：

$$Obj^t \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

- 方程中的 l 即为损失函数（比如平方损失函数： $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$ ，或者交叉熵Log-loss
- $\Omega(f_t)$ 的是正则项（包括L1正则、L2正则），防止过拟合，鲁棒性加强。

- 对于 $f(x)$ ，XGBoost利用二阶泰勒展开三项，做一个近似。 **$f(x)$ 表示的是其中一颗回归树。**

由于在第 t 步时 $\hat{y}_i^{(t-1)}$ 其实是一个已知的值，所以 $l(y_i, \hat{y}_i^{(t-1)})$ 是一个常数，其对函数的优化不会产生影响。因此，去掉全部的常数项，得到目标函数为：

$$Obj^t \approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y, \hat{y}^{(t-1)})$$

$$h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y, \hat{y}^{(t-1)})$$

所以我们只需要求出每一步损失函数的一阶导和二阶导的值（由于前一步的 $\hat{y}^{(t-1)}$ 是已知的，所以这两个值就是常数），然后最优化目标函数，就可以得到每一步的 $f(x)$ ，最后根据加法模型得到一个整体模型。

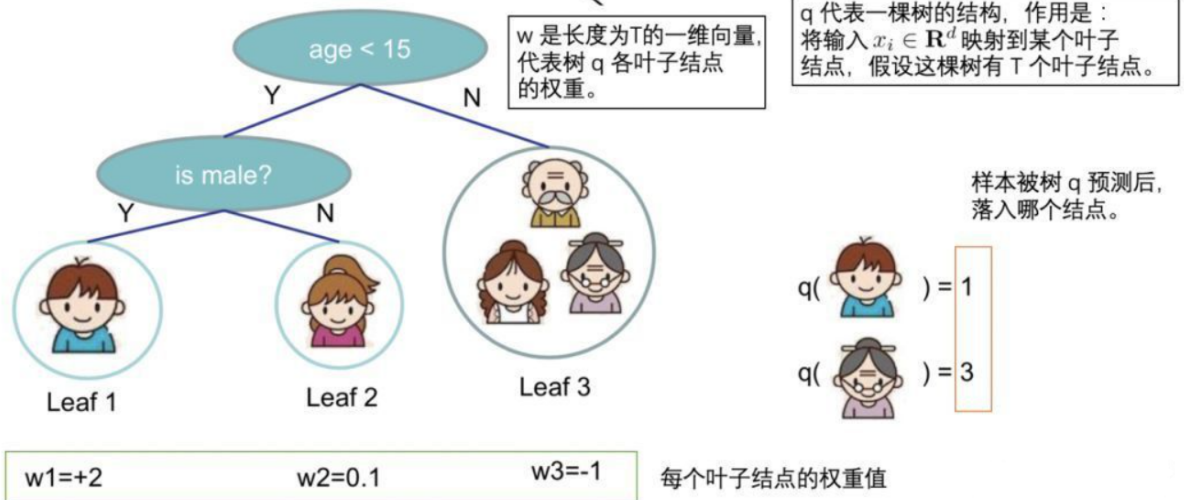
3.3、定义一棵树

我们重新定义一颗树，包括两个部分：

- 叶子结点的权重向量 w ；
- 实例 \rightarrow 叶子结点的映射关系 q （本质是树的分支结构）；

一棵树的表达形式定义如下：

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q: \mathbf{R}^d \rightarrow \{1, 2, \dots, T\}$$



3.4、定义树的复杂度

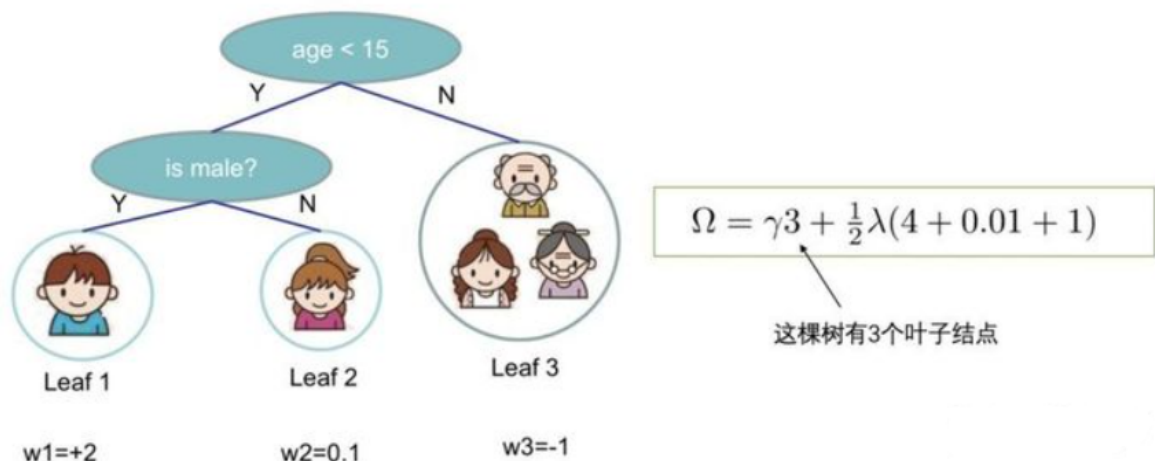
我们定义一颗树的复杂度 Ω ，它由两部分组成：

- 叶子结点的数量；
- 叶子结点权重向量的L2范数；

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

叶子结点的数量

叶子结点权重向量的L2范数



3.5、叶结点归组

我们将属于第 j 个叶子结点的所有样本 x_i ，划入到一个叶子结点样本集中，数学表示如下：

$$I_j = \{i | q(x_i) = j\}$$

然后，将【3】和【4】中一棵树及其复杂度的定义，带入到【2】中泰勒展开后的目标函数 Obj 中，具体推导如下：

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned}$$

所有的训练样本，按叶子结点进行了分组！

为了进一步简化上式，我们定义：

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

含义如下：

- G_j ：叶子结点 j 所包含样本的一阶偏导数累加和，是一个常量
- H_j ：叶子结点 j 所包含样本的二阶偏导数累加和，是一个常量

将 G_j 和 H_j 带入目标式 Obj ，得到我们**最终的目标函数**（注意，此时式中的变量只剩下第 t 棵树的权重向量 w ）

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

3.6、树结构得分

回忆一下高中数学知识。假设有一个一元二次函数，形式如下：

$$Gx + \frac{1}{2} Hx^2, H > 0$$

我们可以套用一元二次函数的最值公式轻易地求出最值点：

$$x^* = -\frac{b}{2a} = \frac{G}{H}$$

那么回到XGBoost的最终目标函数上 $Obj^{(t)}$ ，该如何求出它的最值呢？

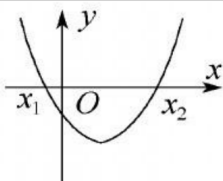
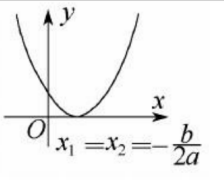
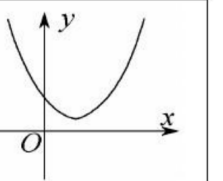
$$Obj^{(t)} = \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

我们先简单分析一下上面的式子：

- 对于每个叶子结点，可以将其从目标函数中拆解出来：

$$G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2$$

在【3.5、叶结点归组】中我们提到， G_j 和 H_j 相对于第 t 棵树来说是可以计算出来。那么，这个式子就是一个只包含一个变量**叶子结点权重** w_j 的一元二次函数，我们可以通过最值公式求出它的最值点。也可以参考[一元二次方程求根公式](#)。

判别式 $\Delta = b^2 - 4ac$	$\Delta > 0$	$\Delta = 0$	$\Delta < 0$
方程 $ax^2 + bx + c = 0$	有两不等实根 x_1 和 x_2 ，且 $x_1 < x_2$	有两相等实根 $x_1 = x_2$	无实根
二次函数 $y = ax^2 + bx + c$ ($a > 0$) 的图像			

- 两个不同实根： $x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- 两个相等实根： $x_{1,2} = -\frac{b}{2a}$

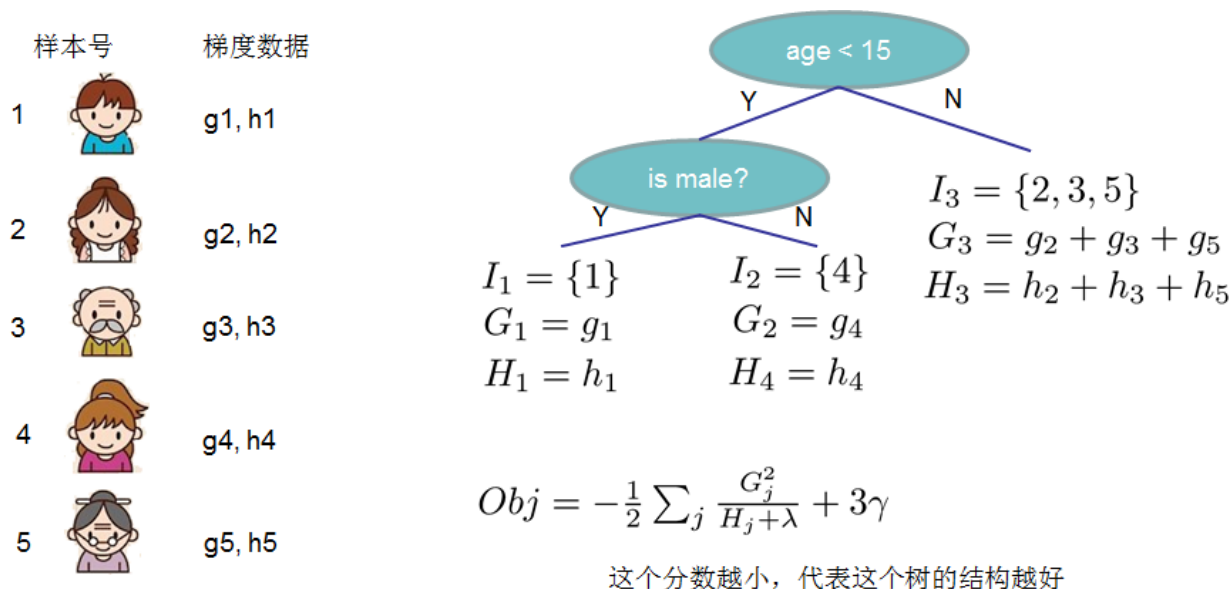
再次分析一下目标函数 $Obj^{(t)}$ ，可以发现，各个叶子结点的目标子式是相互独立的，也就是说，当每个叶子结点的子式都达到最值点时，整个目标函数 $Obj^{(t)}$ 才达到最值点。

那么，假设目前树的结构已经固定，套用一元二次函数的最值公式，将目标函数对 w_j 求一阶导，并令其等于 0，则可以求得叶子结点 j 对应的权值：

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

所以目标函数可以化简为：

$$\begin{aligned}
 Obj^{(t)} &= \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \\
 &= \sum_{j=1}^T \left[-\frac{G_j^2}{H_j + \lambda} + \frac{1}{2} \frac{G_j^2}{H_j + \lambda} \right] + \gamma T \\
 &= -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T
 \end{aligned}$$



这个分数越小，代表这个树的结构越好

上图给出目标函数计算的例子，求每个节点每个样本的一阶导数 g_j 和二阶导数 h_j ，然后针对每个节点对所含样本求和得到 G_j 和 H_j ，最后遍历决策树的节点即可得到目标函数。

3.7、XGBoost与GBDT差异

4、Xgboost模型使用

4.1、模型基本使用

参数说明

4.1.1、使用方式一

```
1 import numpy as np
2 import xgboost as xgb
3 from xgboost import XGBClassifier
4 from sklearn import datasets
5 from sklearn import tree
6 from sklearn.model_selection import train_test_split
7 X,y = datasets.load_wine(return_X_y=True)
8 X_train,X_test,y_train,y_test =
  train_test_split(X,y,test_size=0.2)
9 model = XGBClassifier(learning_rate =0.1,# 学习率，控制每
  次迭代更新权重时的步长，默认0.3。值越小，训练越慢。
10                        n_estimators=10,# 总共迭代的次数，即
  决策树的个数
11                        max_depth=5, # 深度
12                        min_child_weight= 1,# 默认值为1。值
  越大，越容易欠拟合；值越小，越容易过拟合
13                        gamma=0.3,# 惩罚项系数，指定节点分裂所
  需的最小损失函数下降值。
14                        subsample=0.8,# 训练每棵树时，使用的
  数据占全部训练集的比例。默认值为1，典型值为0.5-1。防止
  overfitting。
15                        colsample_bytree=0.8,
```

```

16         objective= 'binary:logistic',# 目
    标函数
17         eval_metric = ['merror'],# 验证数据
    集评判标准
18         nthread=4,)# 并行线程数
19 eval_set = [(X_test, y_test),(X_train,y_train)]
20 model.fit(X_train,y_train,eval_set = eval_set,verbose =
    True)
21 model.score(X_test,y_test)

```

XGBoost可视化

```

1 xgb.to_graphviz(model,
2                 condition_node_params={'shape': 'box',
3                                         'style':
4                                             'filled,rounded',
5                                             'fillcolor':
6                                                 '#78bceb'}},
7                 leaf_node_params={'shape': 'box',
8                                     'style':
9                                         'filled,rounded',
10                                        'fillcolor':
11                                            '#e48038'})

```

4.1.2、使用方式二

```

1 x,y = datasets.load_wine(return_X_y=True)
2 X_train,X_test,y_train,y_test =
    train_test_split(X,y,test_size=0.2)
3 param = {'learning_rate':0.1,
4           'n_estimators':10000,
5           'max_depth':5,
6           'min_child_weight':1,
7           'gamma':0.3,
8           'subsample':0.8,
9           'colsample_bytree':0.8,
10          'verbosity':0,
11          'objective':'multi:softprob',

```

```

12         'eval_metric': 'merror',
13         'early_stopping_rounds': 20}
14
15 model = xgb.XGBClassifier(**param)
16 model.fit(X_train, y_train, eval_set=[(X_test, y_test)])
17 model.score(X_test, y_test)

```

4.1.3、使用方式三

DMatrix是XGBoost中使用的数据矩阵。DMatrix是XGBoost使用的内部数据结构，它针对内存效率和训练速度进行了优化

```

1  import xgboost as xgb
2  from sklearn.metrics import accuracy_score
3  from sklearn import datasets
4  X,y = datasets.load_wine(return_X_y=True)
5  X_train,X_test,y_train,y_test =
    train_test_split(X,y,test_size=0.2)
6
7  # 创建数据
8  dtrain = xgb.DMatrix(data = X_train,label = y_train)
9  dtest = xgb.DMatrix(data = X_test,label = y_test)
10
11 # 指定参数
12 param = {'learning_rate':0.1,
13         'max_depth':5,
14         'min_child_weight':1,
15         'gamma':0.3,
16         'subsample':0.8,
17         'eval_metric':['merror','mlogloss'],
18         'colsample_bytree':0.1,
19         'verbosity':0,
20         'objective':'multi:softmax',
21         'num_class':3}
22 num_round = 1000
23 evals = [(dtrain,'train'),(dtest,'eval')]
24 bst = xgb.train(param,
25                 dtrain,
26                 num_round,

```

```
27         evals = evals,  
28         early_stopping_rounds=10)  
29 # 进行预测  
30 y_ = bst.predict(dtest)  
31 display(y_,accuracy_score(y_test,y_))
```