XGBoost（极端梯度提升）算法

一、课前准备

- 了解GBDT的原理
- 了解Boosting的算法思想

二、课堂主题

- 在大数据竞赛中，XGBoost霸占了文本图像等领域外几乎80%以上的大数据竞赛。当然不仅是在竞赛圈,很多大公司也都将XGBoost作为核心模块使用,好奇的人肯定都很想揭开这个神奇的盒子的幕布。
- XGBoost是GBDT的一种高效实现，但是里面也加入了很多独有的思路和方法，需要细致的了解一下。

三、课堂目标

- 能够掌握XGBoost的计算策略
- 能够掌握XGBoost的正则项信息
- 能够推导XGBoost的目标函数

四、知识要点

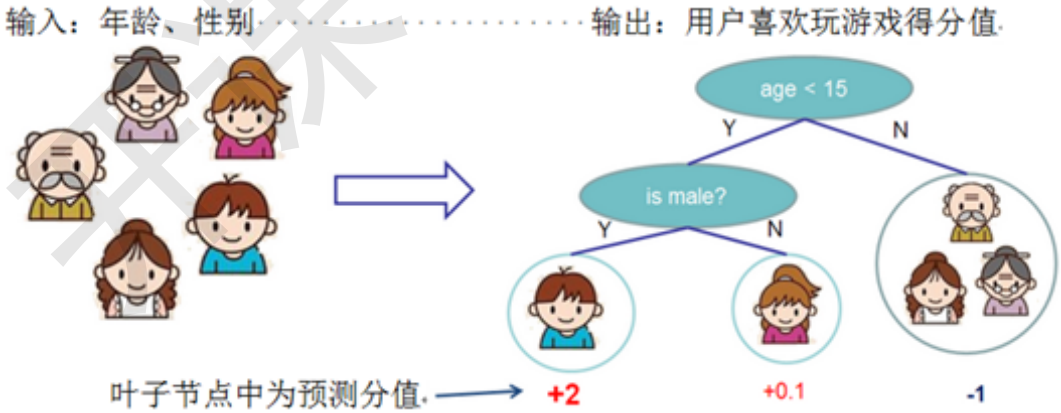
4.1 树模型的发展历程

4.2 XGBoost是什么?

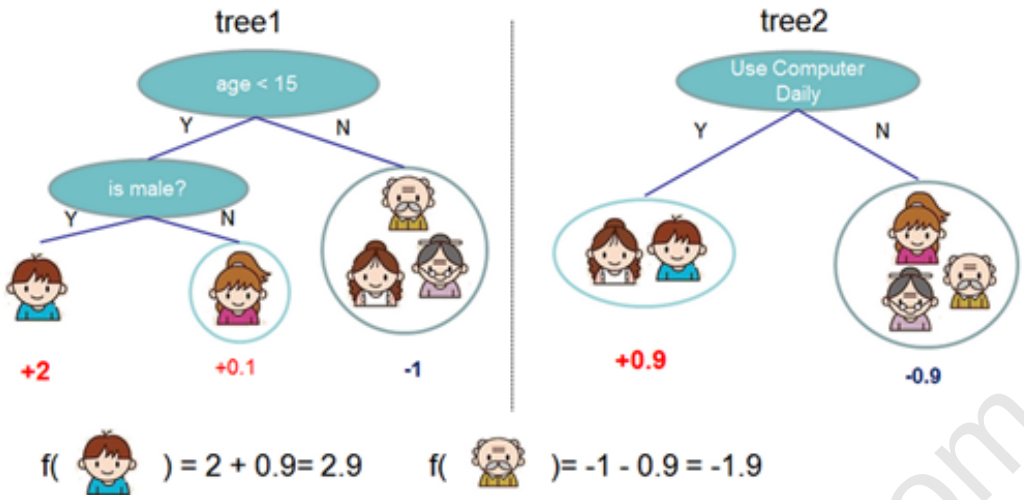
XGBoost全称eXtreme Gradient Boosting，是一种基于决策树的集成机器学习算法，使用梯度上升框架，适用于分类和回归问题。优点是速度快、效果好、能处理大规模数据、支持多种语言、支持自定义损失函数等。

4.3 XGBoost的定义

举个例子



把上图中得到的树记为tree1，同样我们可以根据日常是否使用电脑来进行新一次的打分，如下图所示：



男孩的得分是2+0.9=2.9

爷爷的得分是-1-0.9=-1.9。

4.4 XGBoost的目标函数

XGBoost的核心其实就是不断的添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数，去拟合上次预测的**残差**。用数学来表示这个模型，如下所示：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

这里的K就是树的棵树，F表示所有可能的CART树，f表示一颗具体的CART树，这个模型由K棵CART树组成。

目标函数

要使得树群的预测值$\hat{y}_i$尽量接近真实值y_i，而且要有尽量大的泛化能力。从数学的角度看，目标就变成了一个泛函数的最优化问题，目标函数可简化成如下的形式：

$$obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

这个目标函数分为两部分：损失函数和正则化项。且**损失函数揭示训练误差**（即预测分数和真实分数的差距），这里的正则化项由K棵树的正则化项相加而来，也可以说**正则化定义复杂度。**

4.5 XGBoost的训练

得知了XGBoost的损失函数是加法的形式，不妨使用加法训练的方式分步骤对目标函数进行优化，首先优化第一棵树，接下来是第二棵.....直至K棵树全被优化完成。优化的过程如下所示：

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

.....

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

在第t次迭代的时候，添加了一棵最优的CART树f_t。

$$\begin{aligned} obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

课堂问答

根据化简后的式子，我们知道里面的损失函数，也知道里面的正则项，那么constant又是哪来的呢?

考虑当损失函数是均方误差（MSE）的情况（也就是：$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$）

目标函数可以改写成如下的形式：

$$\begin{aligned} obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) + constant \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant \end{aligned}$$

如何得到的?

我们的目标函数如下：

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

泰勒二阶展开式的作用

泰勒公式的几何意义是利用多项式函数来逼近原函数，由于多项式函数可以任意次求导，易于计算，且便于求解极值或者判断函数的性质，因此可以通过泰勒公式获取函数的信息。

首先我们给出泰勒二阶展开的近似式：

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

由于目标函数中的y_i是真实值，而且通常真实值都是已知的，所以我们不去考虑，对其他的项我们来看一下目标函数和泰勒二阶展开式的对应关系：

- 泰勒二阶展开中f里的x对应目标函数的$\hat{y}_i^{(t-1)}$
- 泰勒二阶展开中f里的Δx对应目标函数里的$f_t(x_i)$

得到目标函数的近似式：

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

其中：gi和hi的表示如下：

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

$$h_i=\partial^2_{\hat{y}^{(t-1)}}\ l(y_i,\hat{y}_i^{(t-1)})$$

如果是MSE则结果如下：

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

$$h_i=\partial^2_{\hat{y}^{(t-1)}}\ l(y_i,\hat{y}_i^{(t-1)})=2$$

考虑到第t棵回归树是根据前面的t-1棵回归树的残差得来的，相当于t-1颗树的预测值$\hat{y}_i^{(t-1)}$是已知的，也就是说，$l(y_i,\hat{y}_i^{(t-1)})$对目标函数的优化不影响，可以直接去掉，同样常数项也可以直接移除，从而得到比较统一的目标函数：

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

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

$$h_i=\partial^2_{\hat{y}^{(t-1)}}\ l(y_i,\hat{y}_i^{(t-1)})$$

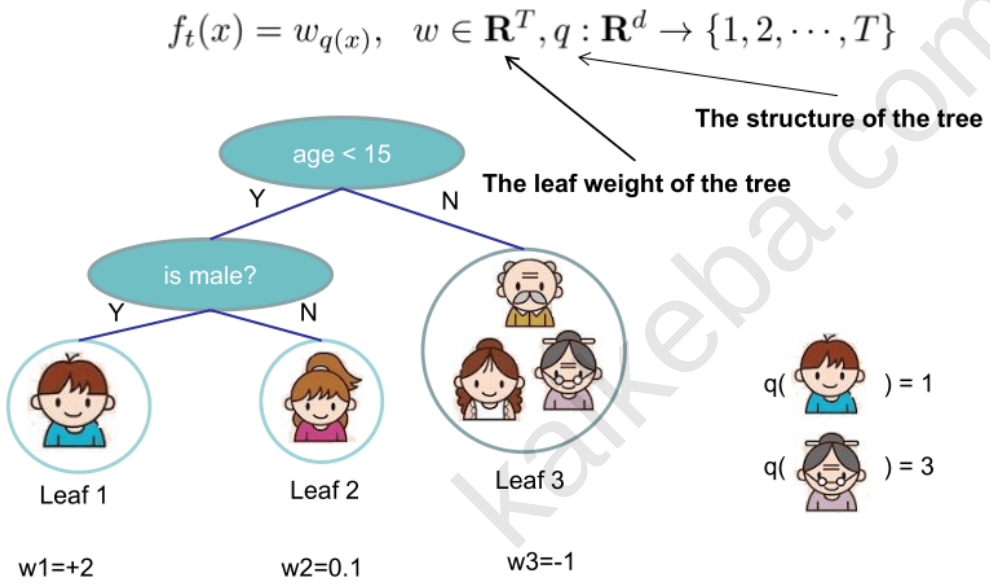
4.6 XGBoost的正则项

正则项也可以看作是**树的复杂度**。

CART树

对CART做一个新的定义，式子及结构表示如下：

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



正则项

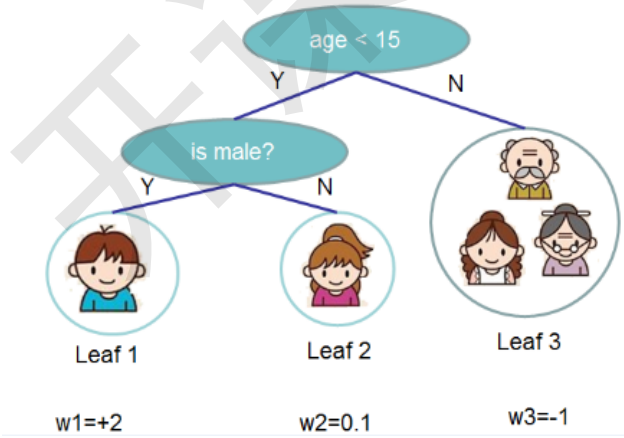
设定XGBoost的正则项包含两个部分：

1. 一个是树里面叶子节点的个数T
2. 一个是树上叶子节点的得分w的L2模平方（对w进行L2正则化，相当于针对每个叶结点的得分增加L2平滑，目的是为了避免过拟合）

最终表示成下图所示的形式：

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

叶子的个数　　w的L2模平方



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

这里出现了两个参数γ和λ，可以设定他们的值去调节树的结构，显然γ越大，表示越希望获得结构简单的树，因为此时对较多叶子节点的树的惩罚越大，λ越大也是希望获得结构越简单的树。

4.7 XGBoost的最终形态

把目标函数和正则项组合在一起得到最终的结果表达：

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

$$obj^{(t)} = \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \lambda\frac{1}{2}\sum_{j=1}^{T} w_j^2$$

$$obj^{(t)} = \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T$$

上式子中的I_j它代表一个集合$I_j = \{i|q(x_i) = j\}$，集合中每个值代表一个训练样本的序号，整个集合就是被第t棵CART树分到了第j个叶子节点上的训练样本（这个定义里的$q(x_i)$要表达的是：每个样本值x_i都能通过函数$q(x_i)$映射到树上的某个叶子节点，从而通过这个定义把两种累加统一到了一起）。叶子结点的个数计做T，叶子结点的分数计做W。

这样加了正则项的目标函数里就出现了两种累加：

- i->n：样本数
- j->T：叶子结点数

进一步对式子进行简化，定义：

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

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

简化后的式子如下：

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

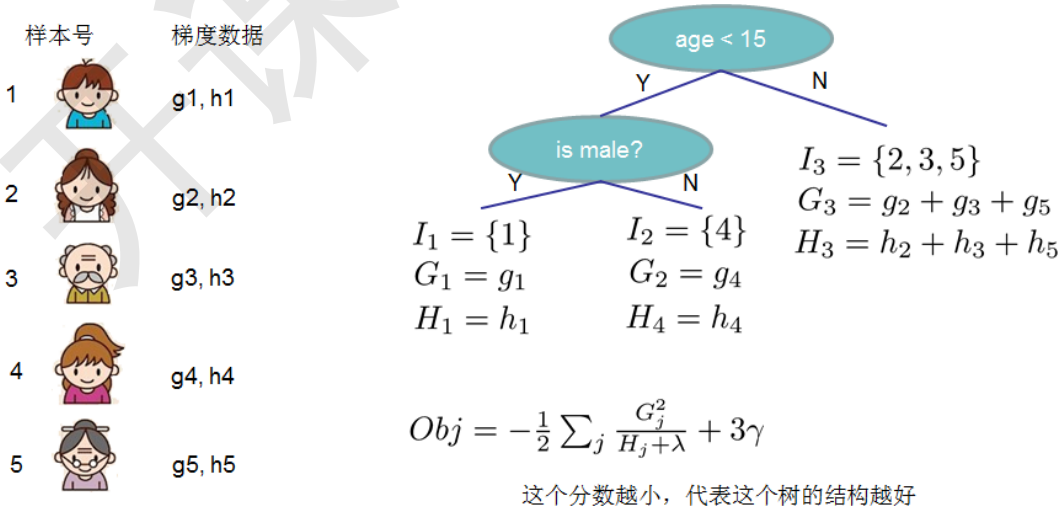
通过对w_j求导等于0，可以得到：

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

然后把最优解带入得到：

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

打分方式



这个分数越小，代表这个树的结构越好

换个角度理解一下w_j^*

假设分到j这个叶子结点上的样本只有一个，则有如下的形式：

$$w_j^* = (\frac{1}{h_j+\lambda})(-g_j)$$

此时第一个括号中表示的内容可以看作是学习率，第二个括号中表示的内容可以看作是反向梯度。

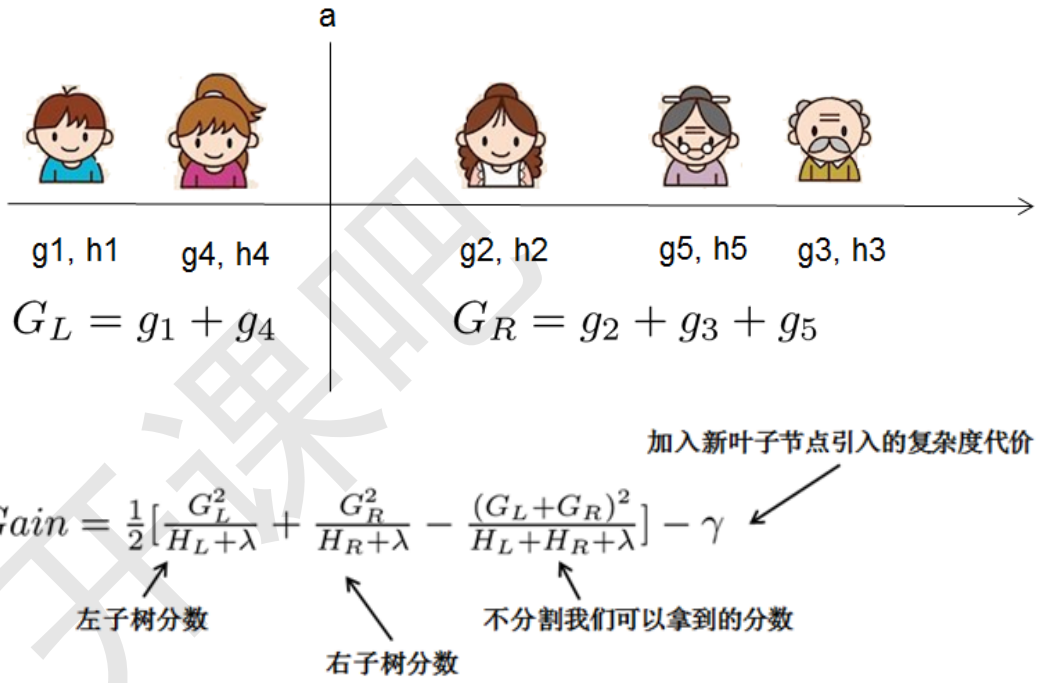
4.8 XGBoost的最优树结构

对于是否喜欢电子游戏的例子，最简单的树结构就是一个结点的树，可以计算出这棵单结点树的好坏obj^*，假设现在想按照年龄将这棵单节点树进行分叉，我们需要知道：

1、按照年龄分是否有效，也就是是否减少了obj的值；

2、如果可分，那么以哪个年龄值来分。

按照年龄进行排序，找出所有的切分点，对于每一个切分点去衡量切分的好坏。示例图和计算方式如下表示：



这个Gain实际上就是单节点的obj^*减去切分后的两个节点的树obj^*，Gain如果是正的，并且值越大，表示切分后obj^*越小于单节点的obj^*，就越值得切分。同时，我们还可以观察到，Gain的左半部分如果小于右侧的γ，则Gain就是负的，表明切分后obj^*反而变大了。γ在这里实际上是一个临界值，它的值越大，表示我们对切分后obj^*下降幅度要求越严。这个值也是可以在XGBoost中设定的。

扫描结束后，我们就可以确定是否切分，如果切分，对切分出来的两个节点，递归地调用这个切分过程，我们就能获得一个相对较好的树结构。

注意：xgboost的切分操作和普通的决策树切分过程是不一样的。普通的决策树在切分的时候并不考虑树的复杂度，而依赖后续的剪枝操作来控制。xgboost在切分的时候就已经考虑了树的复杂度，就是那个γ参数。所以，它不需要进行单独的剪枝操作。

4.9 XGBoost问题小结

- XGBoost为什么用泰勒展开?

Xgboost官网上有说，当目标函数是MSE时，展开是一阶项（残差）+二阶项的形式（官网说这是一个nice form），而其他目标函数，如logloss的展开式就没有这样的形式。为了能有个统一的形式，所以采用泰勒展开来得到二阶项，这样就能把MSE推导的那套直接复用到其他自定义损失函数上。简短来说，就是为了统一损失函数求导的形式以支持自定义损失函数。这是从为什么会想到引入泰勒二阶的角度来说的

二阶信息本身就能让梯度收敛更快更准确。这一点在优化算法里的牛顿法里已经证实了。可以简单认为一阶导指引梯度方向，二阶导指引梯度方向如何变化。这是从二阶导本身的性质，也就是为什么要用泰勒二阶展开的角度来说的

- XGBoost如何寻找最优特征？是有放回还是无放回的呢?

XGBoost在训练的过程中给出各个特征的评分，从而表明每个特征对模型训练的重要性。

XGBoost利用梯度优化模型算法, 样本是不放回的，想象一个样本连续重复抽出,梯度来回踏步，这显然不利于收敛。

XGBoost支持子采样, 也就是每轮计算可以不使用全部样本。

4.10 XGBoost的数据形式

XGBoost可以加在多种格式的训练数据：

- libsvm格式的文本数据
- Numpy的二维数组
- XGBoost的二进制的缓存文件加载的数据存储在对象Dmatrix中

```
# 示例代码，作为参考，不可运行
# 加载libsvm格式的数据
dtrain1 = xgb.DMatrix('train.svm.txt')

# 加载二进制的缓存文件
dtrain2 = xgb.DMatrix('train.svm.buffer')

# 加载numpy数组

data = np.random.rand(5, 10)  # 5行10列数据集
label = np.random.randint(2, size=5)  # 二分类目标值
dtrain = xgb.DMatrix(data, label=label)  # 组成训练集

# 将Dmatrix格式的数据保存成Xgboost的二进制格式，在下次加载时可以提高加载速度
dtrain = xgb.DMatrix('train.svm.txt')
dtrain.save_binary("train.buffer")

# 处理DMatrix中的缺失值
dtrain = xgb.DMatrix( data, label=label, missing = -999.0)

# 给定样本权重的方式
w = np.random.rand(5,1)
dtrain = xgb.DMatrix( data, label=label, missing = -999.0, weight=w)
```

4.11 XGBoost的参数调节

xgboost的训练过程中，我们主要用到xgboost.train()方法。

基本方法：

```
xgboost.train(params,
              dtrain,
              num_boost_round=10,
              evals(),
              obj=None,
              feval=None,
              maximize=False,
              early_stopping_rounds=None,
              evals_result=None,
              verbose_eval=True,
              learning_rates=None,
              xgb_model=None)
```

默认参数：

- parms：这是一个字典，里面包含着训练中的参数关键字和对应的值，形式是parms = {'booster':'gbtree','eta':0.1}
- dtrain：训练的数据
- num_boost_round：这是指提升迭代的个数
- evals：这是一个列表，用于对训练过程中进行评估列表中的元素。形式是evals = [(dtrain,'train'),(dval,'val')] 或者是 evals =[(dtrain,'train')] ，对于第一种情况，它使得我们可以在训练过程中观察验证集的效果。
- obj ： 自定义目的函数
- feval：自定义评估函数
- maximize：是否对评估函数进行最大化
- early_stopping_rounds：早起停止次数，假设为100，验证集的误差迭代到一定程度在100次内不能再继续降低，就停止迭代。这要求evals里至少有一个元素，如果有多个，按照最后一个去执行。返回的是最后的迭代次数（不是最好的）。如果early_stopping_rounds存在，则模型会生成三个属性，bst.best_score ,bst.best_iteration和bst.best_ntree_limit
- evals_result：字典，存储在watchlist中的元素的评估结果
- verbose_eval（可以输入布尔型或者数值型）：也要求evals里至少有一个元素，如果为True，则对evals中元素的评估结果会输出在结果中；如果输入数字，假设为5，则每隔5个迭代输出一次。
- learning_rates：每一次提升的学习率的列表
- xgb_model：在训练之前用于加载的xgb_model

params参数说明：

```
# xgboost模型
params = {
    'booster':'gbtree',
    'objective':'multi:softmax',   # 多分类问题
    'num_class':10,  # 类别数，与multi softmax并用
    'gamma':0.1,     # 用于控制是否后剪枝的参数，越大越保守，一般0.1 0.2的样子
    'max_depth':12,  # 构建树的深度，越大越容易过拟合
    'lambda':2,  # 控制模型复杂度的权重值的L2 正则化项参数，参数越大，模型越不容易过拟合
    'subsample':0.7, # 随机采样训练样本
    'colsample_bytree':3,# 这个参数默认为1，是每个叶子里面h的和至少是多少
    # 对于正负样本不均衡时的0-1分类而言，假设h在0.01附近，min_child_weight为1
```

```
    # 意味着叶子节点中最少需要包含100个样本。这个参数非常影响结果，
    # 控制叶子节点中二阶导的和的最小值，该参数值越小，越容易过拟合
    'silent':0,  # 设置成1 则没有运行信息输入，最好是设置成0
    'eta':0.007,  # 如同学习率
    'seed':1000,
    'nthread':7,  # CPU线程数
    #'eval_metric':'auc'
}
```

通用参数：

- booster [default=gbtree]
  - 有两种模型可以选择gbtree和gblinear。gbtree使用基于树的模型进行提升计算，gblinear使用线性模型进行提升计算。缺省值为gbtree
- silent [default=0]
  - 取0时表示打印出运行时信息，取1时表示以缄默方式运行，不打印运行时的信息。缺省值为0
    建议取0，过程中的输出数据有助于理解模型以及调参。另外实际上我设置其为1也通常无法缄默运行。。
- nthread [default to maximum number of threads available if not set]
  - XGBoost运行时的线程数。缺省值是当前系统可以获得的最大线程数
    如果你希望以最大速度运行，建议不设置这个参数，模型将自动获得最大线程
- num_pbuffer [set automatically by xgboost, no need to be set by user]
  - 预测缓冲区的大小，通常设置为训练实例的数量。缓冲器用于保存最后一个提升步骤的预测结果。
- num_feature [set automatically by xgboost, no need to be set by user]
  - boosting过程中用到的特征维数，设置为特征个数。XGBoost会自动设置，不需要手工设置

tree booster参数

- eta [default=0.3]
  - 为了防止过拟合，更新过程中用到的收缩步长。在每次提升计算之后，算法会直接获得新特征的权重。 eta通过缩减特征的权重使提升计算过程更加保守。缺省值为0.3
  - 取值范围为：[0,1]
  - 通常最后设置eta为0.01~0.2
- gamma [default=0]
  - 在树的叶子节点上进行进一步分区所需的最小损失减少。越大，算法就越保守。
  - range: [0,∞]
  - 模型在默认情况下，对于一个节点的划分只有在其loss function 得到结果大于0的情况下才进行，而gamma 给定了所需的最低loss function的值gamma值使得算法更conservation，且其值依赖于loss function ，在模型中应该进行调参。
- max_depth [default=6]
  - 树的最大深度。缺省值为6
  - 取值范围为：[1,∞]
  - 指树的最大深度
  - 树的深度越大，则对数据的拟合程度越高（过拟合程度也越高）。即该参数也是控制过拟合
  - 建议通过交叉验证（xgb.cv) 进行调参

    - 通常取值：3-10
- min_child_weight [default=1]
    - 孩子节点中最小的样本权重和。如果一个叶子节点的样本权重和小于min_child_weight则拆分过程结束。在线性回归模型中，这个参数是指建立每个模型所需要的最小样本数。该成熟越大算法越conservative。即调大这个参数能够控制过拟合。
    - 取值范围为: [0,∞]
- max_delta_step [default=0]
    - 我们允许每棵树的权值估计为。如果该值设置为0，则表示不存在约束。如果将其设置为正值，则有助于使更新步骤更加保守。这个参数通常是不需要的，但它可能有助于逻辑回归时，类是非常不平衡。将其设置为1-10可能有助于控制更新
    - 取值范围为：[0,∞]
    - 如果取值为0，那么意味着无限制。如果取为正数，则其使得xgboost更新过程更加保守。
    - 通常不需要设置这个值，但在使用logistics 回归时，若类别极度不平衡，则调整该参数可能有效果
- subsample [default=1]
    - 用于训练模型的子样本占整个样本集合的比例。如果设置为0.5则意味着XGBoost将随机的从整个样本集合中抽取出50%的子样本建立树模型，这能够防止过拟合。
    - 取值范围为：(0,1]
- colsample_bytree [default=1]
    - 在建立树时对特征随机采样的比例。缺省值为1
    - 取值范围：(0,1]
- colsample_bylevel[default=1]
    - 决定每次节点划分时子样例的比例
    - 通常不使用，因为subsample和colsample_bytree已经可以起到相同的作用了
- scale_pos_weight[default=0]
    - 大于0的取值可以处理类别不平衡的情况。帮助模型更快收敛

Linear Booster参数

- lambda [default=0]
    - L2 正则的惩罚系数
    - 用于处理XGBoost的正则化部分。通常不使用，但可以用来降低过拟合
- alpha [default=0]
    - L1 正则的惩罚系数
    - 当数据维度极高时可以使用，使得算法运行更快。
- lambda_bias
    - 在偏置上的L2正则。缺省值为0（在L1上没有偏置项的正则，因为L1时偏置不重要）

学习目标参数

- objective [default=reg:linear]
    - 定义学习任务及相应的学习目标，可选的目标函数如下：
    - “reg:linear” –线性回归。
    - “reg:logistic” –逻辑回归。
    - “binary:logistic” –二分类的逻辑回归问题，输出为概率。

  - “binary:logitraw” –二分类的逻辑回归问题，输出的结果为wTx。
  - “count:poisson” –计数问题的poisson回归，输出结果为poisson分布。
  - 在poisson回归中，max_delta_step的缺省值为0.7。(used to safeguard optimization)
  - “multi:softmax” –让XGBoost采用softmax目标函数处理多分类问题，同时需要设置参数num_class（类别个数）
  - “multi:softprob” –和softmax一样，但是输出的是ndata * nclass的向量，可以将该向量reshape成ndata行nclass列的矩阵。每行数据表示样本所属于每个类别的概率。
  - “rank:pairwise” – 通过最小化pairwise损失，将XGBoost设置为执行排序任务
- seed [default=0]
  - 随机数的种子。缺省值为0
  - 可以用于产生可重复的结果（每次取一样的seed即可得到相同的随机划分）

4.12 XGBoost调参数经验

调参步骤：

- 1，选择较高的学习速率（learning rate）。一般情况下，学习速率的值为0.1.但是，对于不同的问题，理想的学习速率有时候会在0.05~0.3之间波动。选择对应于此学习速率的理想决策树数量。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。
- 2，对于给定的学习速率和决策树数量，进行决策树特定参数调优（max_depth , min_child_weight , gamma , subsample,colsample_bytree）在确定一棵树的过程中，我们可以选择不同的参数。
- 3，Xgboost的正则化参数的调优。（lambda , alpha）。这些参数可以降低模型的复杂度，从而提高模型的表现。
- 4，降低学习速率，确定理想参数。

4.13 XGBoost代码展示

- 原生XGBoost需要先把数据集按输入特征部分，输出部分分开，然后放到一个DMatrix数据结构里面，这个DMatrix我们不需要关心里面的细节，使用我们的训练集X和y初始化即可。

```
# -*- encoding:utf-8 -*-

import xgboost as xgb
import numpy as np
from sklearn.model_selection import train_test_split  # cross_validation


def iris_type(s):
    it = {b'Iris-setosa': 0, b'Iris-versicolor': 1, b'Iris-virginica': 2}
    return it[s]


path = './data/iris.data'  # 数据文件路径
data = np.loadtxt(path, dtype=float, delimiter=',', converters={4: iris_type})
x, y = np.split(data, (4, ), axis=1)
```

```
# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    random_state=1,
                                                    test_size=50)

data_train = xgb.DMatrix(x_train, label=y_train)
data_test = xgb.DMatrix(x_test, label=y_test)
watch_list = [(data_test, 'eval'), (data_train, 'train')]

param = {
    # 树的深度
    'max_depth': 4,
    # 收缩步长
    'eta': 0.1,
    # 分类器选择：softmax
    'objective': 'multi:softmax',
    # 类别数
    'num_class': 3
}
# 参数，训练数据，提升迭代的个数，评估元素
bst = xgb.train(param, data_train, num_boost_round=4, evals=watch_list)
y_hat = bst.predict(data_test)
result = y_test.reshape(1, -1) == y_hat
print('正确率:\t', float(np.sum(result)) / len(y_hat))
```

```
[0] eval-merror:0.02  train-merror:0.02
[1] eval-merror:0.02  train-merror:0.02
[2] eval-merror:0.02  train-merror:0.02
[3] eval-merror:0.02  train-merror:0.02
正确率:   0.98
```

```
data
```

五、总结

- XGBoost原理
- XGBoost泰勒2阶展开的作用
- XGBoost的常用参数调节

六、作业

- 熟悉XGBoost算法的应用（常用参数）

- 熟悉XGBoost的数学原理
- 预习高潜用户购买画像