# 第6章 提升树模型

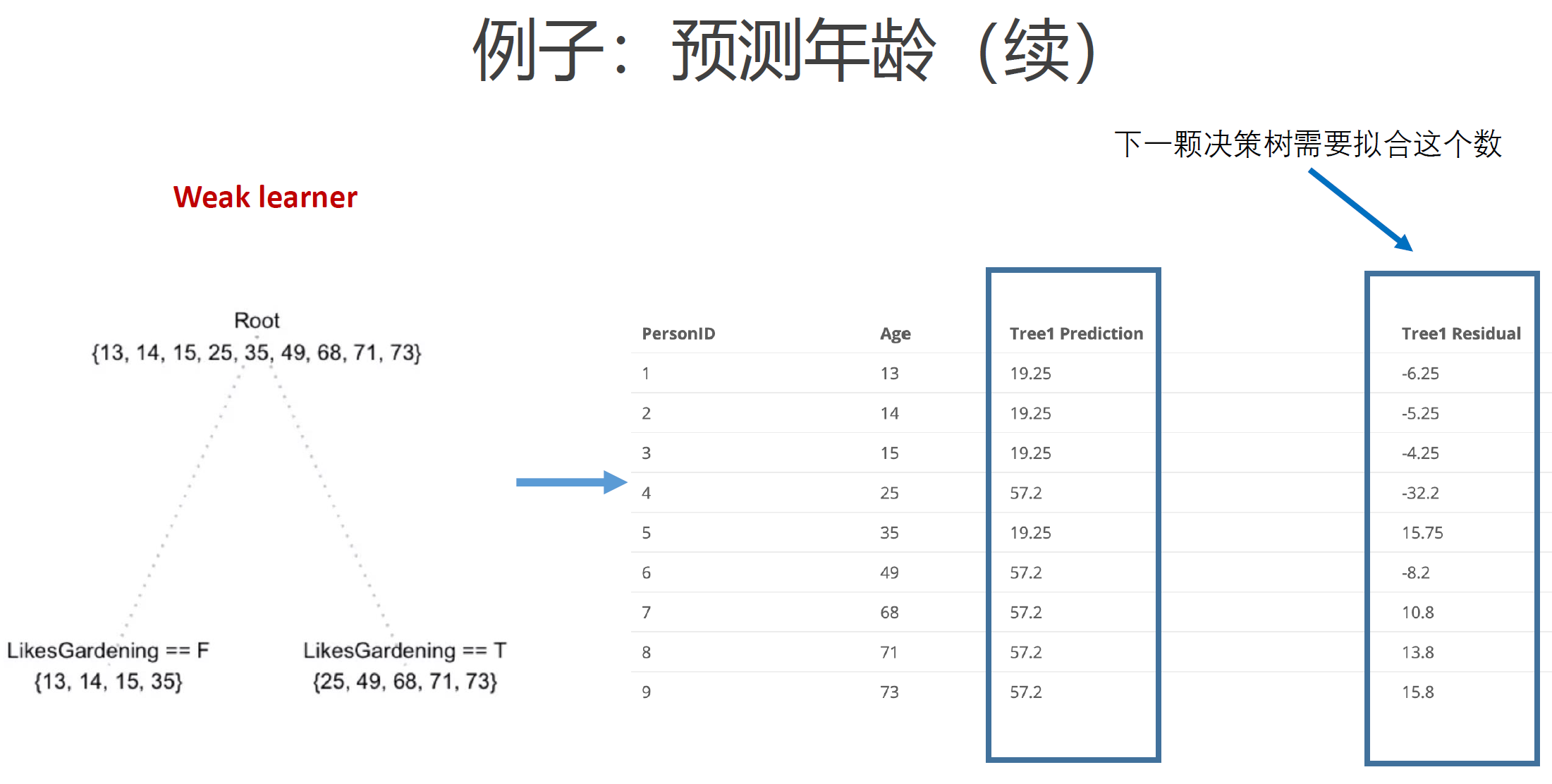
## 6.1 XGBoost原理

### 6.1.1 从一个简单的例子出发

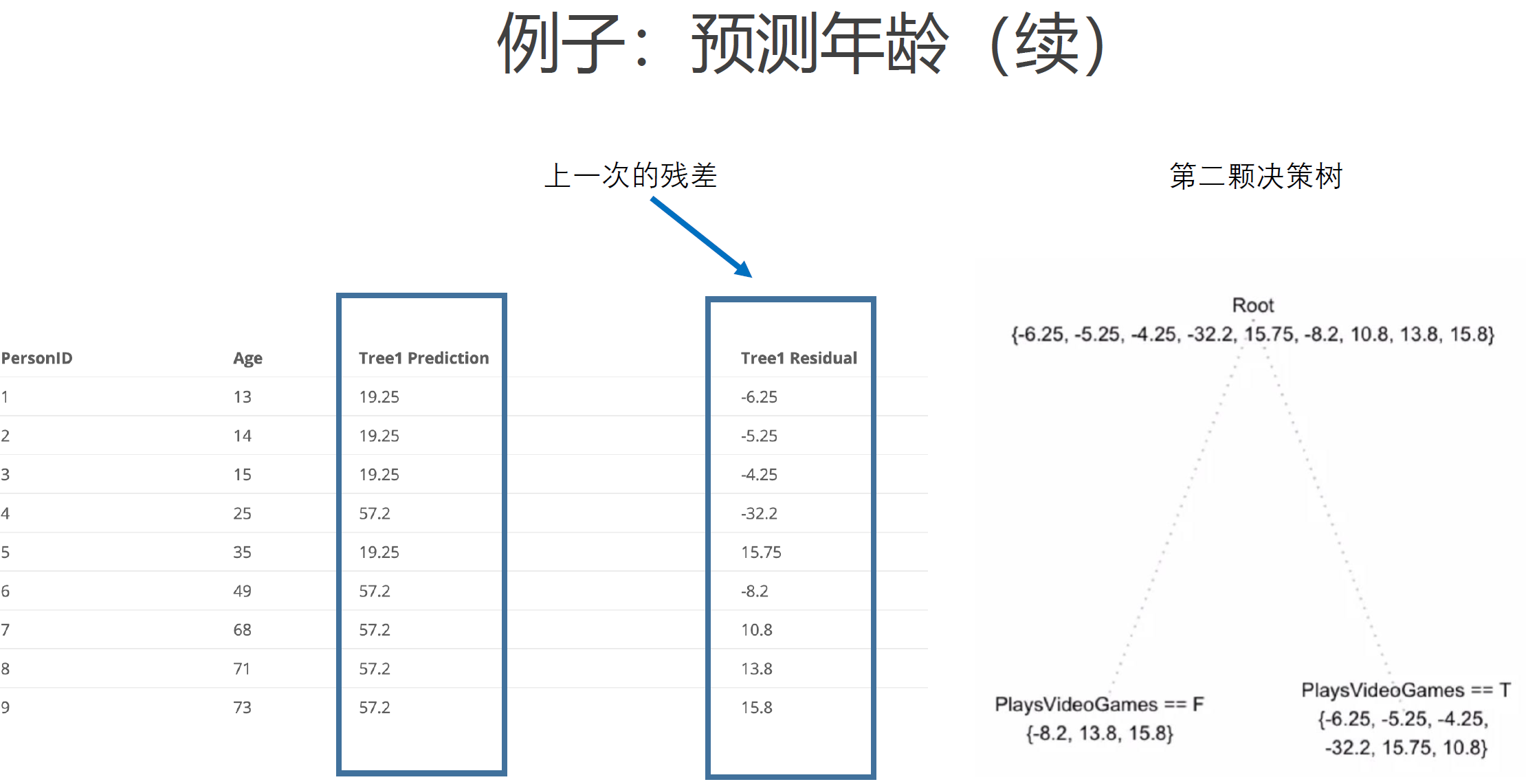
我们从一个最简单的例子出发，总览一下XGBoost算法的预测过程。比如我们要预测用户的年龄，X包括喜欢花园、打游戏、喜欢帽子三个特征，Y是客户的年龄，如下：



首先，我们使用一个弱分类器进行预测，比如弱分类器就是一个两层的决策树，采用“喜欢花园”这个特征进行一次划分，因此我们的第一颗树的预测结果就是下图中的Tree1 Prediction，残差为Tree1 Residual，这就是下一颗决策树需要拟合的目标。



接下来，第二颗决策树采用“打游戏”这个特征进行划分，拟合上一棵树的残差，预测结果为Tree2 Prediction。



如果说XGB只进行两轮训练，那么两个子模型的预测结果如下图所示，两个模型预测结果叠加（Combined Prediction），即为最终的预测结果。最终模型的输出结果与实际值之间的残差为Final Residual，同时可以计算出预测结果的SSE，评估模型的好坏。

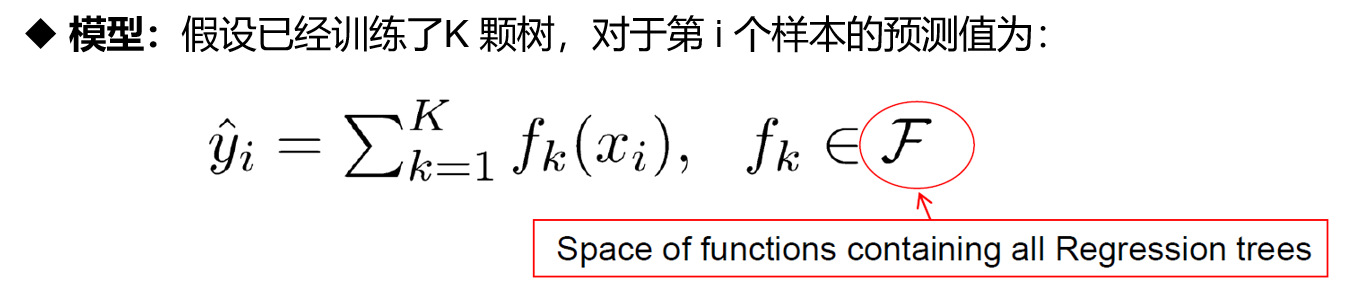


如上所述，XGB训练就是这样的过程，不断对残差进行拟合，每一次预测都在减小模型的残差，直到模型的预测结果足够好为止。

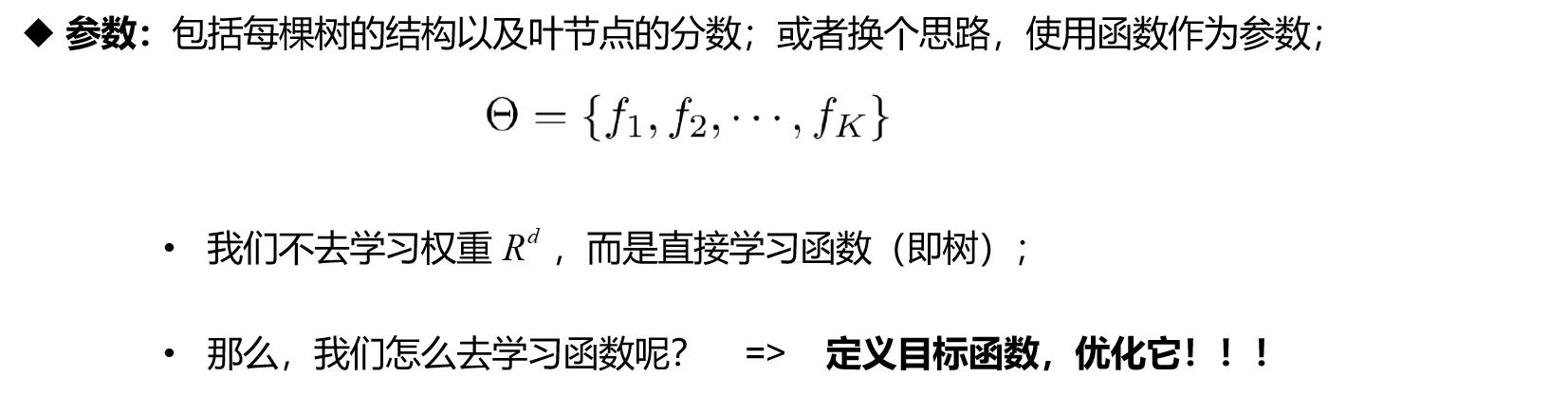
### 6.1.2 原理

#### 6.1.2.1 定义模型

正如前面的例子看到的，XGB就是串行训练出很多颗回归树叠加起来得到的，因此，XGB模型的表达式可以如下表述：



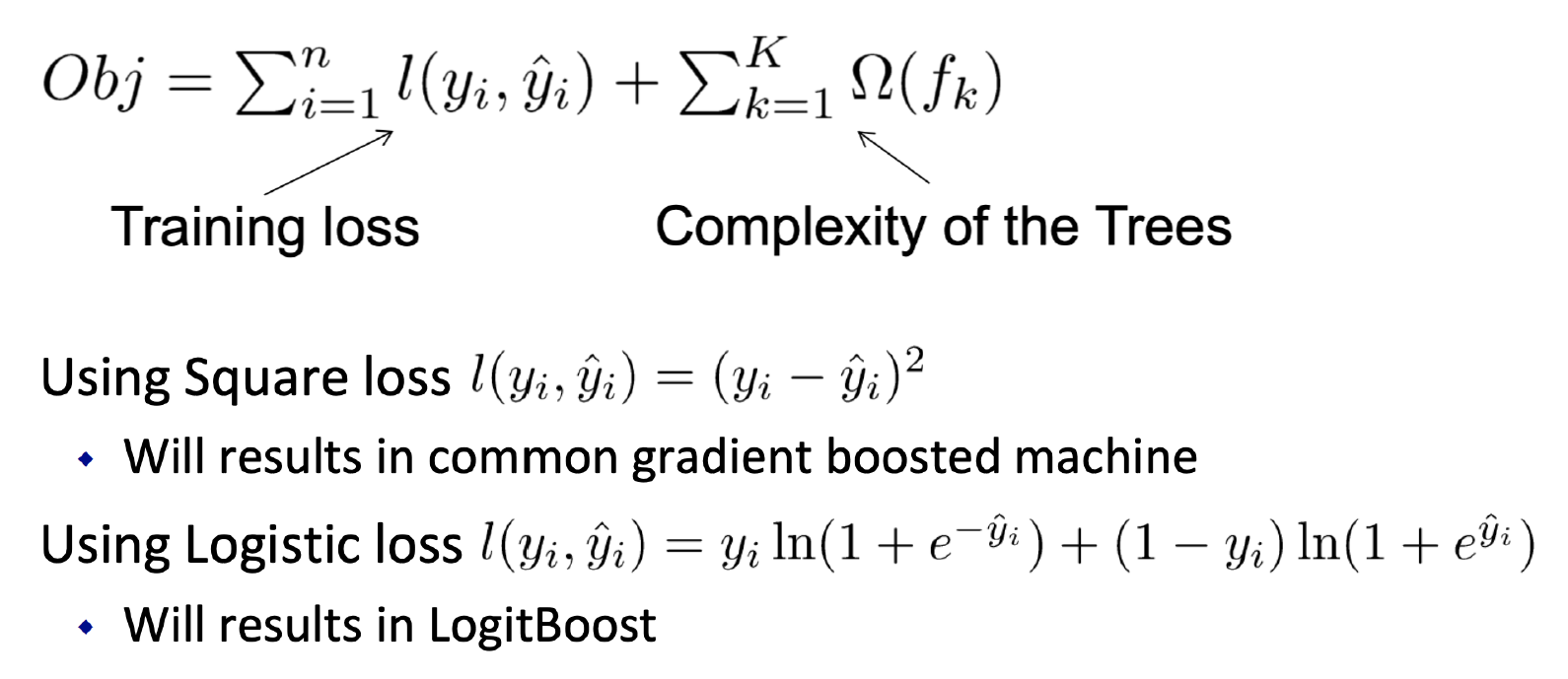
跟所有的机器学习模型一样，我们想要找到一组参数使得模型的预测结果尽可能接近实际值y。既然XGB模型是用很多颗回归树组成的，那么参数应该就是每棵树的结构以及样本划分到叶节点的分数，当然这么考虑会比较复杂，我们简化一下，直接把每个回归树当做一个整体来看（先当做黑箱），将回归树函数作为参数，不去学习权重，直接去学习函数就可以了。



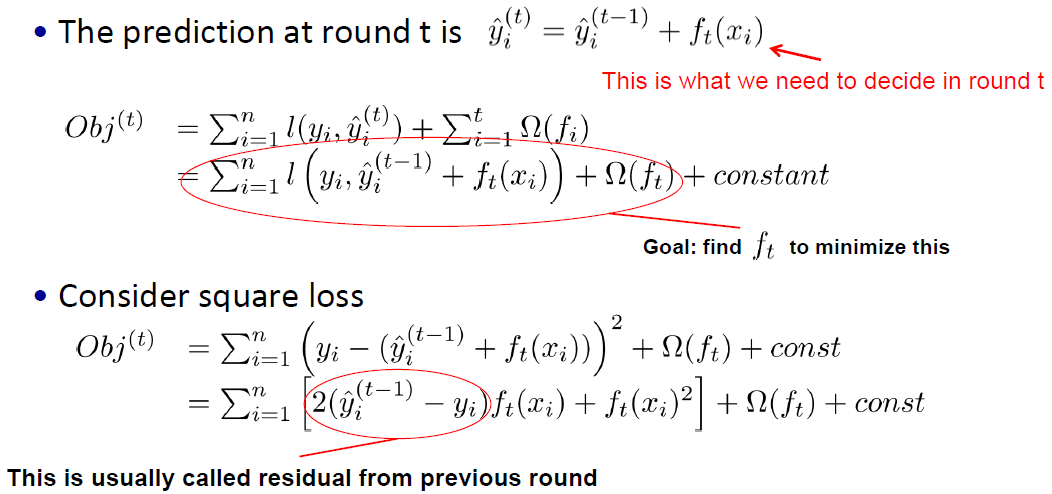
这样的话，我们怎么去学习函数呢？其实学习函数和学习参数是一样的，就是定义一个目标函数，然后去优化它。

#### 6.1.2.2 构造目标函数

参考一般的机器学习模型的目标函数，我们定义的目标函数包括两个部分：训练损失项和惩罚项。对于损失项，如果是回归问题可以使用平方损失，如果是分类问题可以使用交叉熵损失。

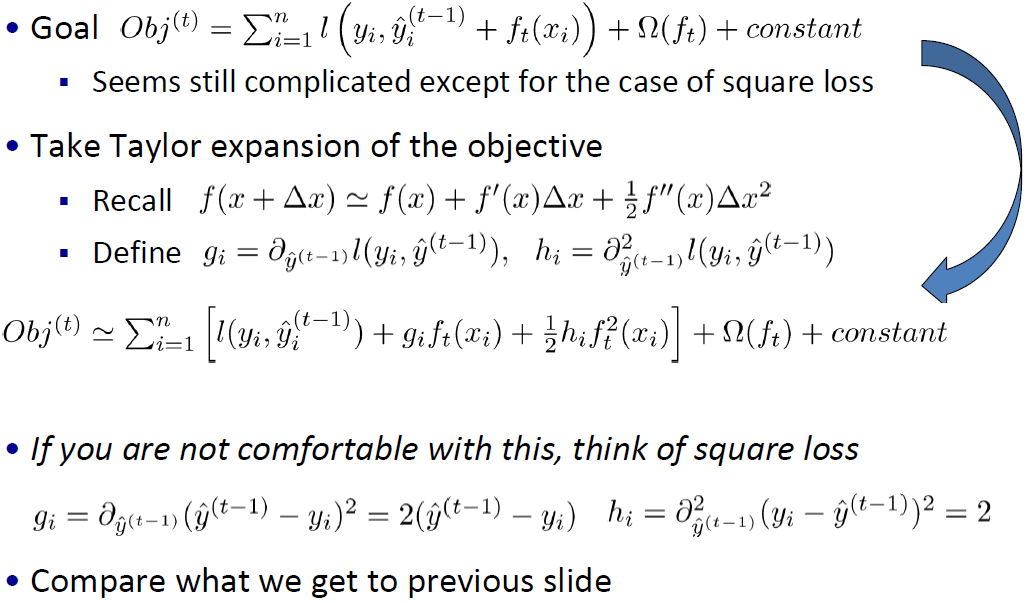


目标函数的形式确定了，我们把第t轮模型的预测值放进去，便可以得到第t轮模型的目标函数Obj(t)。在t-1轮已经确定的情况下，我们的目标就是找到一个ft使得Obj(t)最小。

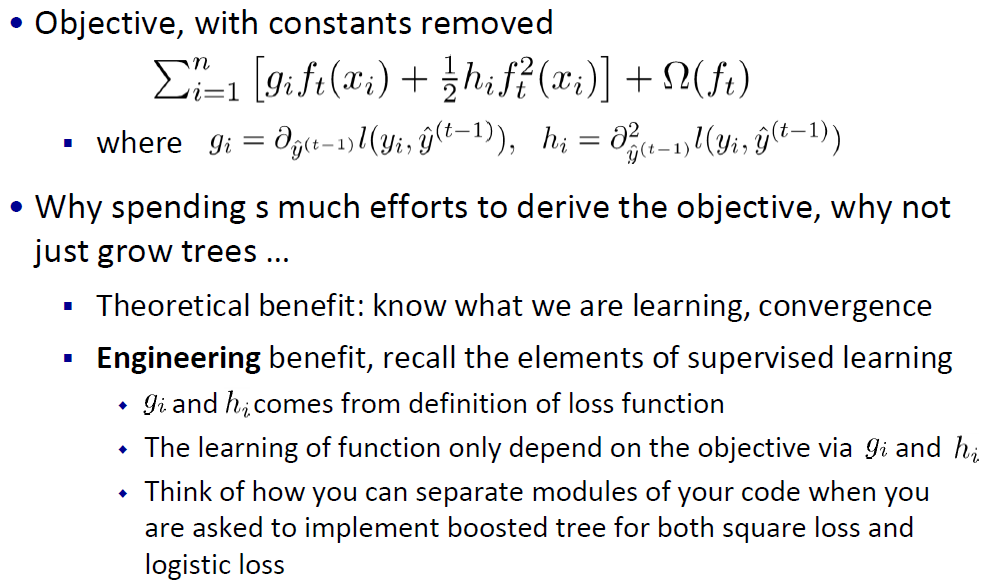


#### 6.1.2.3 使用泰勒级数近似目标函数

对于这个目标函数，如果损失项使用平方损失，那这个函数就很简单，但是如果不是平方损失，目标函数就会有些复杂，所以我们进行一定程度的简化，使用泰勒级数近似这个目标函数：



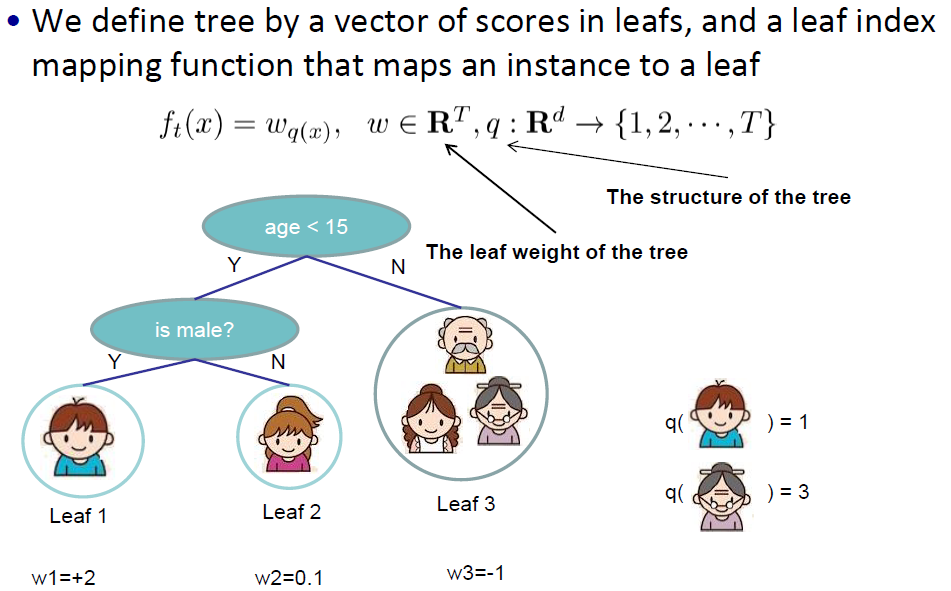
进行二阶泰勒展开后，目标函数变成了下面的新形式：



g, h分别是一阶导数和二阶导数，这两个值是很容易计算的，但f(x)如何去表示呢？因为它是一棵树，目标函数里面有棵树，这就有点尴尬了，我们如何将这棵树表示成函数的形式呢？

#### 6.1.2.4 重新定义一棵树

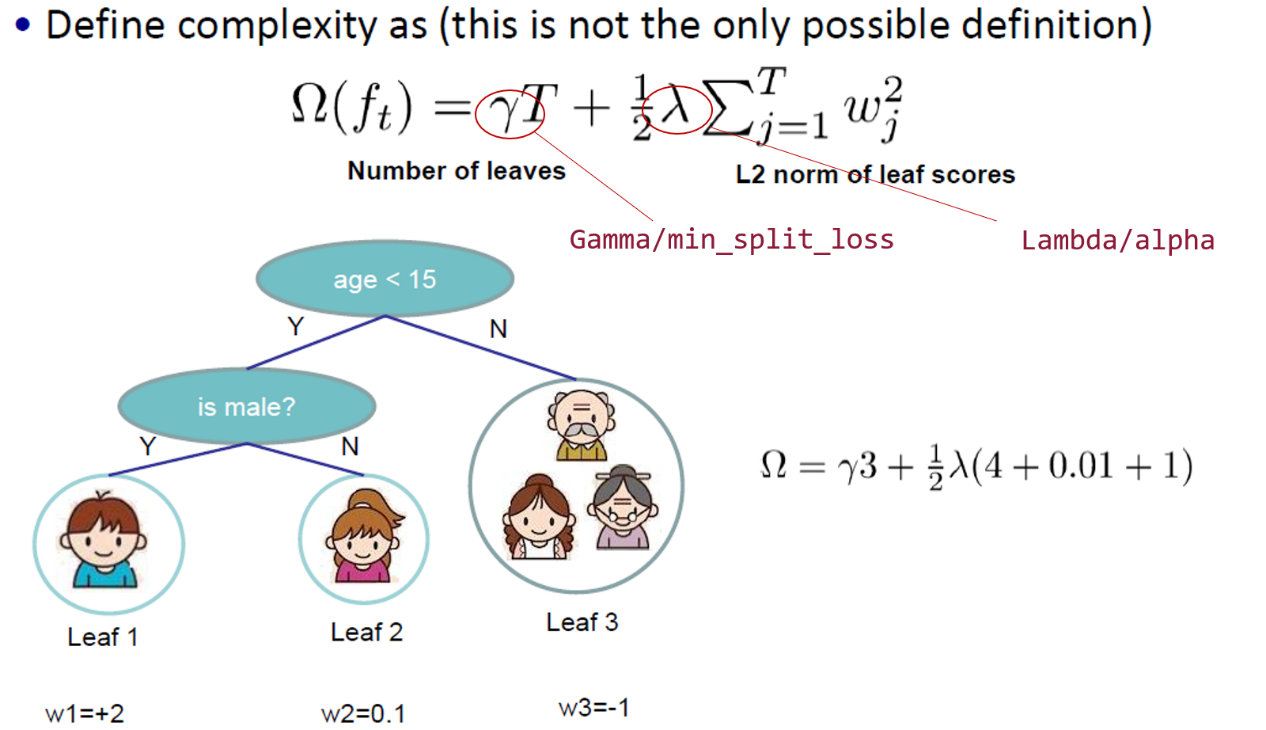
对于一棵树f(x)，假如我们已经知道树的结构（后面会说结构的事，先认为已经知道了），给定一个样本，我们只要知道这个样本分配到哪个节点上，以及节点的分数就可以了，因此，我们就可以把这棵树f(x)定义为一个函数，函数包括两部分：样本映射到叶节点的函数和叶节点的分数。比如下面这个例子，小男孩被映射到第一个叶节点，第一个叶节点的分数为2，那么一个小男孩进入这棵树，最终输出结果就是2。



到此为止，损失项定义完了，目标函数中还有一部分惩罚项需要定义。

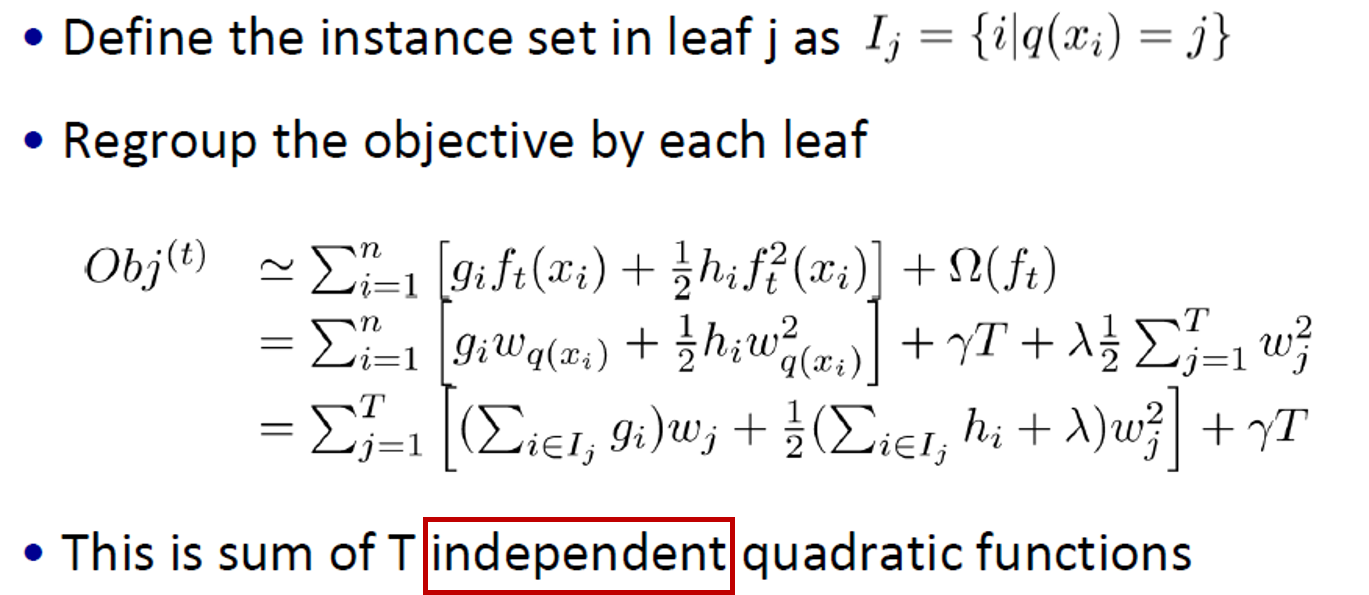
#### 6.1.2.5 定义树的复杂度

惩罚项就是在限制树的复杂度，对于目标函数，我们既希望训练损失尽可能小，又希望树尽可能简单，减小过拟合。因此，我们定义的惩罚项包括两部分：对叶节点分数ω的L2正则化（λ对应XGB中的参数Lambda/alpha），对叶节点个数的惩罚（γ对应XGB中的参数Gamma/min\_split\_loss）。



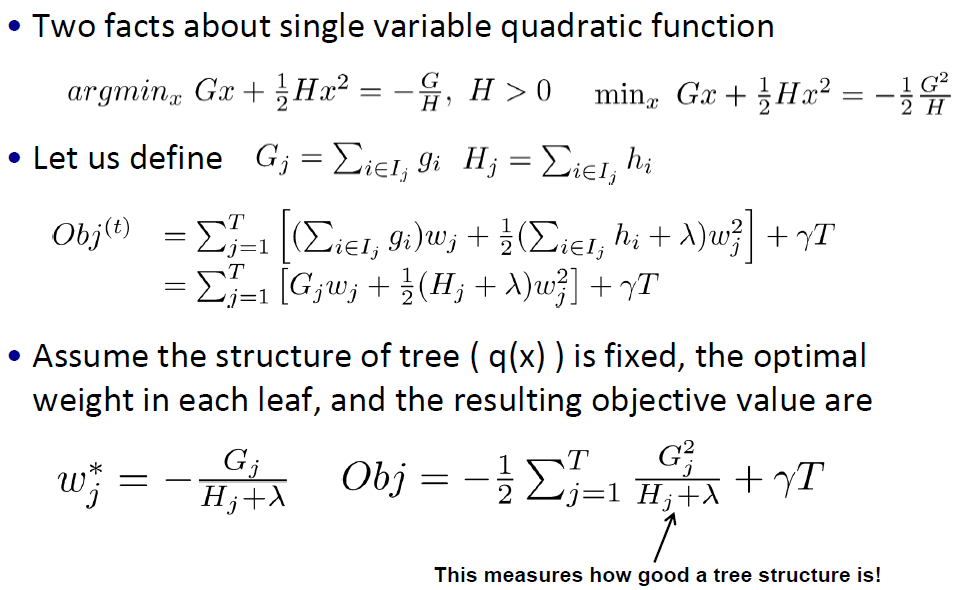
#### 6.1.2.6 新的目标函数

将上面的定义都放到目标函数中，便可得到新的目标函数。目标函数经过化简可以发现：这是T个无依赖的二次函数的加法，也就是说，一旦树结构确定下来，树的T个叶节点上的损失互不影响（并行计算）。

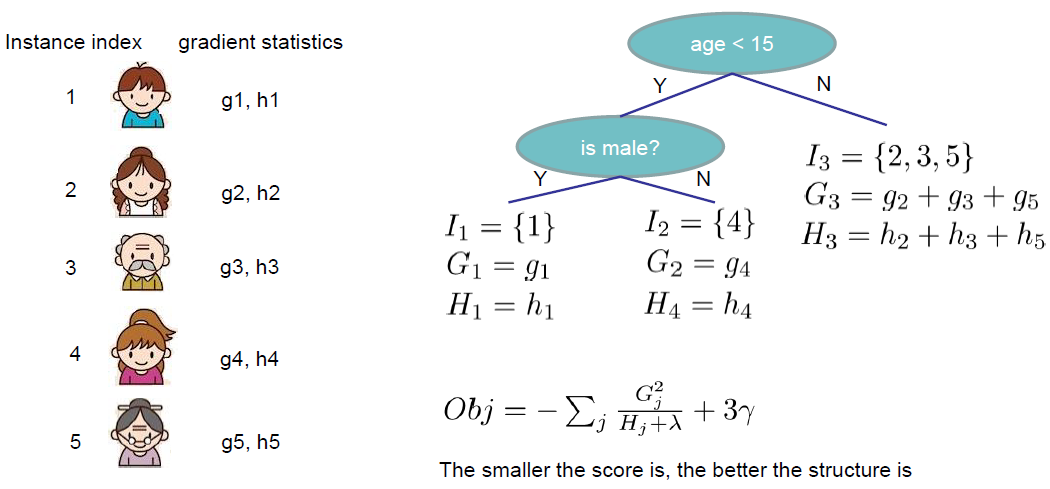


#### 6.1.2.7 树结构的打分

既然损失函数的形式为二次函数，我们便可以直接使用求根公式得到参数ω和目标函数的最小值，这个目标函数的值可以用来衡量一个树的结构有多好。



对于树结构分数的计算可以参考下面这个例子，在已知样本的梯度统计量的情况下（统计量怎么得到的？），代入表达式便可以算出损失函数的分数，这个分数越小，说明树的结构越好。

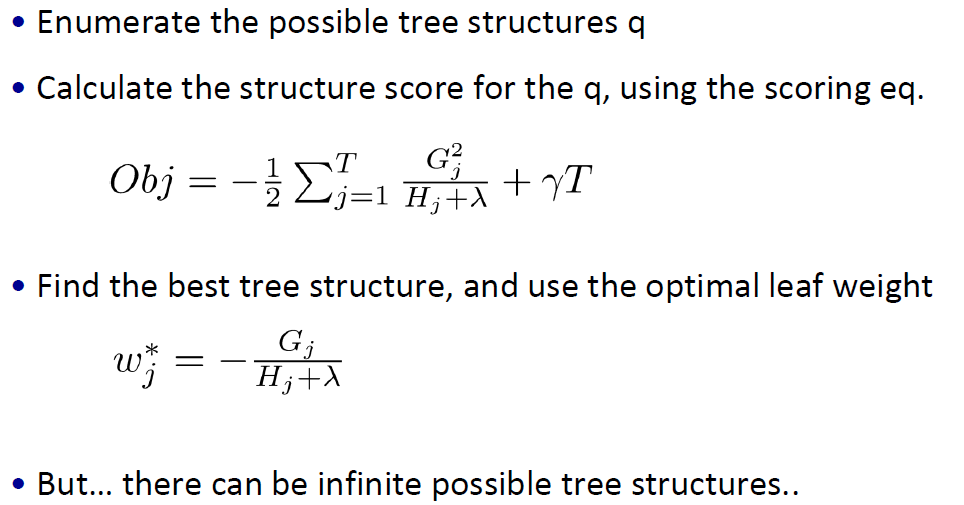


#### 6.1.2.8 如何寻找树的形状

我们回过头来解决前面遗留的一个问题，前面的推导我们是假设树的结构是已知的情况下推导的，但是如何寻找树的形状？现在来解决一下，这里说明三种解决方法：

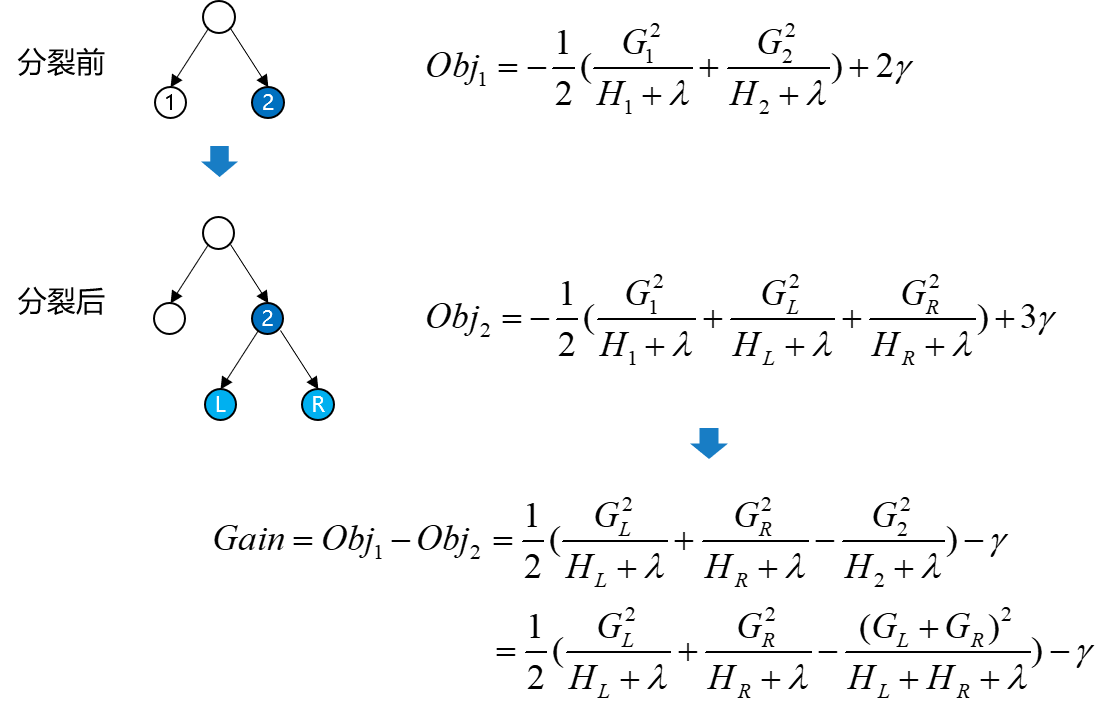
**（1）暴力解法**

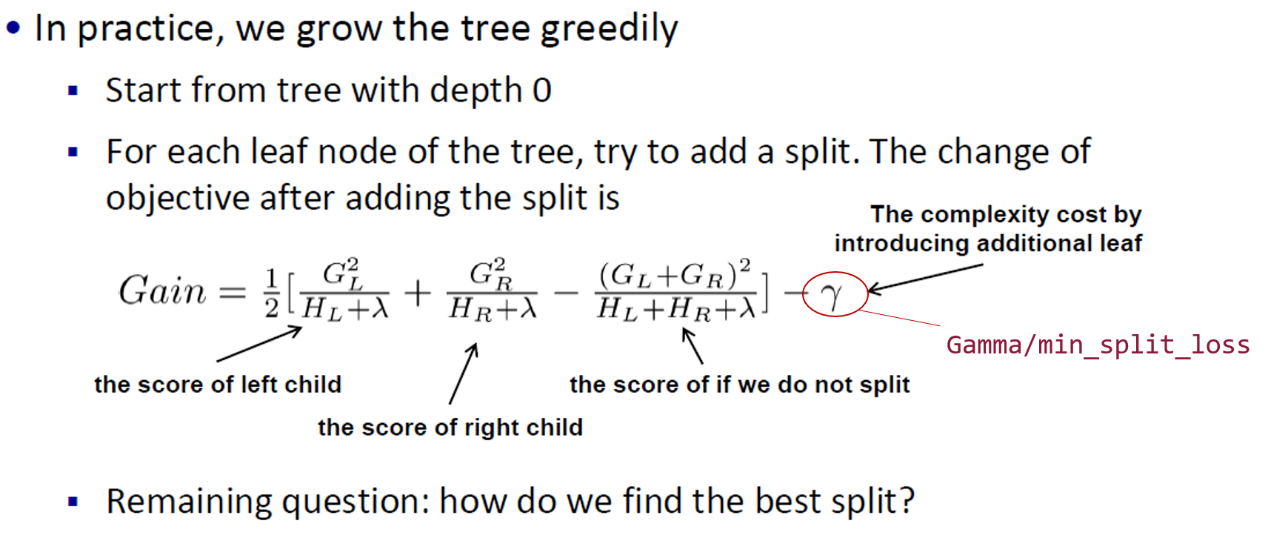
第一种方法，枚举所有的树结构，然后计算每种结构的分数，选择其中最好的树结构，得到最优的叶节点权重。这种想法很直观，但是，我们做不到。



**（2）贪心算法**

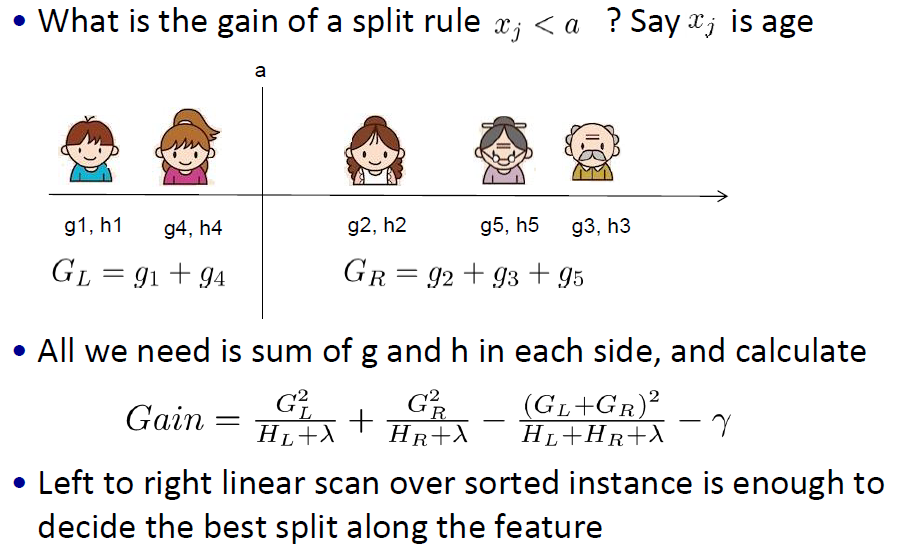
第二种方法，采用贪心算法执行树的生长。从根节点开始，对树的每一个节点试着用特征去分裂，如下图所示，对叶节点2进行分裂，分裂前目标函数为Obj1，分裂后为Obj2，Gain为分裂后目标函数的减小量。出于贪心的想法，我们当然希望这个减小量越大越好，因此这个减小量Gain就可以衡量分裂的好坏，用作寻找最好split的标准。





注：从Gain的表达式，我们可以知道为什么λ会有一个别名min\_split\_loss，因为分裂后损失项的减小量（Gain的第一部分）至少要大于γ，不然Gain为负值，导致目标函数不仅没有减小，反而变大了。此外，该分裂增益可以作为特征重要性输出的重要依据（feature\_importances\_默认度量方式就是gain）。

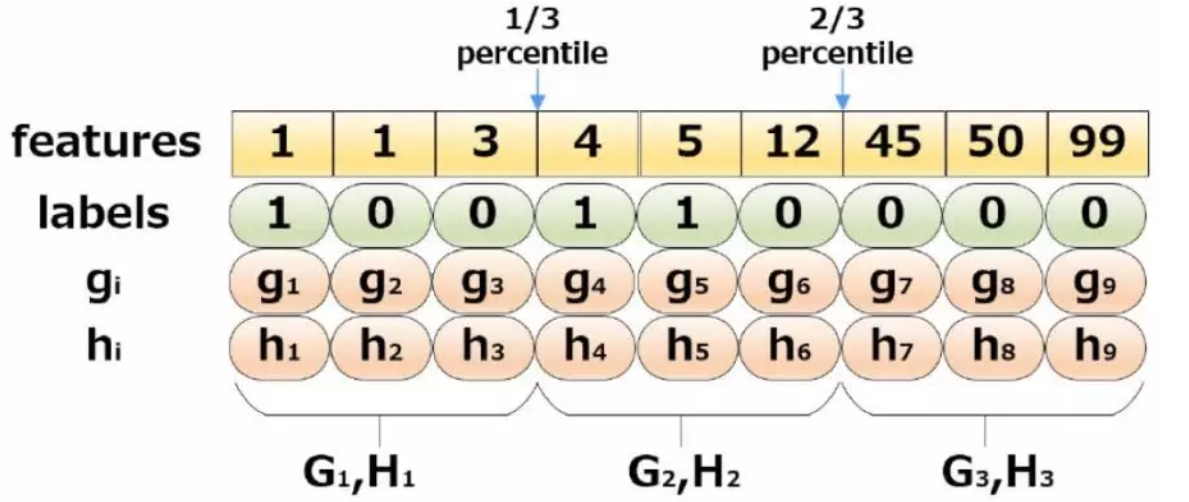
既然如此，我们怎么寻找最好的split呢？对于一个特征，我们可以选择不同的a值来进行划分，然后比较Gain的大小，最终选择Gain最大的特征和划分点。一般情况下，将特征值进行排列，然后从左到右线性搜索即可找到特征的最优切分点。



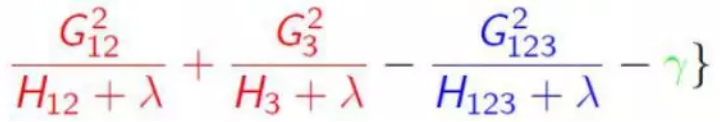
**（3）近似算法**

贪心算法可以得到最优解，但当数据量太大时则无法读入内存进行计算，近似算法主要针对贪心算法这一缺点给出近似最优解。对于每个特征，只考察**分位点**可以减少计算复杂度。

该算法先根据特征分布的分位数提出候选划分点，然后将连续型特征映射到由这些候选点划分的桶里，然后聚合统计信息找到所有区间的最佳分裂点。



该算法的操作过程如图，根据样本特征进行排序，然后基于分位数进行划分，并统计三个桶内的G，H值，最终求解节点划分的增益Gain：



如上所述寻找树形状的算法，在XGB中由参数tree\_method决定，默认为auto，支持的方法包括exact、approx和hist。

## 6.2 XGB用于评分卡

本节主要介绍使用XGBoost模型生成评分卡。由于XGBoost模型对输入没有逻辑回归模型那么敏感，因此其特征处理过程会有所简化。但XGBoost模型的参数调整远比逻辑回归模型复杂，因此需要比较完善的自动化调参策略。

### 6.3.1 特征工程

特征工程指的是通过数据处理方法，从源数据中抽取关键信息进行组合，挖掘出更加深入的信息的过程。在传统的机器学习领域，特征工程对模型的帮助非常大。多数算法最终结果的好坏很大程度上取决于特征工程的优劣。

工业界的两种主流建模思路是复杂特征加简单算法和简单特征加复杂算法。整体来看这两者并没有优劣之分，但是算法越简单可解释性越强。在传统信用评分建模中，业务人员更希望得到一个解释性更强的模型，所以使用复杂的特征工程加逻辑回归算法进行建模。然而随着机器学习模型在各个领域的普及，风控领域使用的模型正在慢慢地向更复杂的模型过渡。

目前实用性较强的复杂算法的代表为卷积神经网络（CNN）、循环神经网络（RNN）等表征学习算法，这些算法对特征工程的要求非常低，建立一个优秀的模型不一定要有大量业务经验的支撑，也就意味着对新手更友好。美中不足的是，复杂算法对数据量和计算资源的要求更高。

现阶段风控领域部分业务仍追求一定的解释性，并且考虑到模型融合的整体复杂度，将复杂模型作为一种特征构造方法，使用复杂模型的输出作为传统评分卡的输入，最终仍使用传统逻辑回归进行训练。

### 6.2.2 特征衍生

业界常用的特征衍生方案有以下两种：

* **通过算法自动进行特征交叉。**虽然不可以解释，但是可以将特征挖掘得较为深入和透彻。可以很轻松地从基础的几百维度衍生至任意维度，比如可以通过XGBoost对特征进行离散，或者通过FM算法进行特征交叉，也可以通过神经网络进行表征学习，然后将内部的参数取出来作为模型的输入（如使用word2vec算法的权重作为表征特征）。总之，只要升高了特征维度，再和原始特征合并一起建模，都可以看成是特征衍生。
* **通过一些跨时间维度的计算逻辑对特征进行时间维度的比较，从而衍生出具有业务含义的特定字段。**这种做法会具有更强的解释性，是传统的银行或者信用卡中心惯用的衍生方法之一。

### 6.2.3 离散特征处理

使用XGBoost算法只能接收连续变量的输入，因此需要对实际业务中的分类型变量进行映射，得到数值型变量。常用的有两种编码方法：

#### 6.2.3.1 one-hot编码

one-hot编码的就是用N位状态寄存器对分类型特征的N个状态进行编码。比如：某个特征为国家，包括["中国", "美国", "日本"]三个国家，如果进行one-hot编码，"中国"的编码是[1, 0, 0]，"美国"的编码是[0, 1, 0]，"日本"的编码是[0, 0, 1]。类别特征转化为数值特征，一个维度的特征转化为三个维度的特征。

one-hot编码的优点：

* one-hot编码可以对特征进行扩充；
* 连续变量经过编码后，从一个权重变为多个权重，提升了模型的非线性能力；
* 不需要多参数进行归一化处理；
* 随着将大权重拆分成几个小权重处理特征，降低了异常值对模型的影响，增加了模型稳定性；

one-hot编码的缺点：

* 生成了维度较大的稀疏矩阵；

#### 6.2.3.2 WOE编码

WOE（Weight of Evidence，证据权重）是一种对原始自变量进行编码的形式。WOE编码是一种有监督的编码方式，将预测类别的集中度的属性作为编码的数值。对于自变量第i箱的WOE值为：



其中， 是第i箱中坏客户占所有坏客户的比例；

是第i箱中好客户占所有好客户的比例；

是第i箱中坏客户的人数；

是第i箱中好客户的人数；

是所有坏客户的人数；

是所有好客户的人数；

WOE编码是标准评分卡模型不可或缺的一个环节，无论是对字符型变量还是对数值型变量都要进行WOE映射。对数值型变量进行WOE映射只要是为了弱化极值影响，增加模型鲁棒性。但树模型对极值和变量分布波动并不敏感，因此XGBoost评分卡中只对字符型变量进行WOE映射。

##### 6.2.3.2.1 WOE预分箱

在实现WOE映射的过程中，最重要的一点是分箱的逻辑，显然分箱不同，得到的WOE映射值会有很大不同。这里使用**基于负样本占比差异最大化的分箱原则**。所期望得到的分箱结果应该是：箱的总数在5箱以内（可以适当调整，通常不超过10箱），并且每一箱之间的负样本占比差值尽可能大（箱合并原则），每一箱的样本量不能小于整体样本的5%（可以根据分箱结果调整，原则是不要太小）。换言之，主要通过控制划分后的总箱数，来迭代进行箱的合并。



图1 负样本占比差异最大化分箱

将离散变量转化为数值编码后，就可以开始正式建模了。因为XGBoost等树模型只关心数值的排序，对变量的分布和取值范围并不敏感，所以不需要过多地进行归一化处理。为保证树模型的精度，数值型变量也不必做分箱处理。

##### 6.2.3.2.2 WOE转换

对每个分类型变量采用WOE转换，但是不进行预分箱，这样WOE值相近的变量类别会被划分到一个分箱，转换后的数值量与数值型变量一起送入XGBoost自动化分箱。

##### 6.2.3.2.3 badrate转换

对每个分类型变量采用badrate转换，这样XGBoost分箱时会基于负样本占比差异最大化进行分箱，转换后的数值量与数值型变量一起送入XGBoost自动化分箱。

### 6.2.4 特征筛选

#### 6.2.4.1 迭代特征重要度筛选

在特征筛选阶段，使用集成模型中效果最好的XGBoost算法进行特征的初步筛选。需要注意的是，直接根据XGBoost算法的特征重要度小于某一阈值对特征进行筛选，有一定的不合理性：当某些低重要度特征被删除后，其余低重要度特征的重要性会有所上升。因此使用一种基于迭代思想的特征筛选方法来完成特征筛选的第一环节，这样做的目的是削弱特征间的相互影响。

迭代特征筛选方法的步骤为：

1. 设置特征重要度阈值min\_score和单次迭代最多删除特征个数max\_del\_var\_nums；
2. 使用特征集合var\_set训练XGBoost模型，得到所有特征的重要度feature\_importance；
3. 对于每个特征：如果特征重要度小于min\_score，将其加入待删除特征集合var\_del，直到var\_del的特征数量大于等于max\_del\_var\_nums，break；
4. 判断var\_del集合是否为空。如果不为空，从var\_set集合删除特征var\_del，回到第（2）步，重复这个过程；如果为空，说明已经没有能够删除的特征，迭代特征筛选结束。

本初步筛选方案的精华在于，使用min\_score参数控制每一次删除的特征重要性，使用max\_del\_var\_nums控制每一次循环删除特征的个数。这在一定程度上避免了特征之间的干扰。

#### 6.2.4.2 特征稳定性筛选

除了基于模型贡献度的筛选方式外，业务同样需要模型具备一定的稳定性。因为信用评分模型的稳定性很大程度上取决于模型中每个变量分布的稳定性，所谓特征稳定性，就是关注该特征的取值随着时间的推移会不会发生大的波动。为保证模型上线后的稳定性，需要对模型中稳定性较差的变量进行筛选。

对特征稳定性的关注，一定要在建模之前完成，从一开始就避免将那些本身不太稳定的特征选入模型。如果建模前没有筛选不稳定的特征，直到模型临近上线，才意识到应该去看看有没有不太稳定的特征，一旦发现有特征稳定性不满足要求，则需要对其进行剔除后重新建模，从而导致不必要的重复性劳动。

在传统评分卡中，通常对每一个特征的信息值IV、最大信息系数MIC、PSI等指标进行特征筛选，XGBoost模型同样可以使用这些方法：

##### 6.2.4.2.1 IV筛选

IV称为信息价值(information value)，是目前评分卡模型中筛选变量最常用的指标之一，自变量的IV值越大，表示自变量的预测能力越强。类似的指标还有信息增益、基尼(gini)系数等。常用判断标准如表1：

表1 IV值的判断标准

|  |  |
| --- | --- |
| **IV范围** | **预测能力** |
| <0.02 | 无效 |
| 0.02-0.10 | 弱预测能力 |
| 0.10-0.20 | 中预测能力 |
| >0.20 | 强预测能力 |

计算变量中第i个分箱对应的 IV 值的计算公式为：



变量对应的IV值为所有分箱对应的 IV 值之和：



##### 6.2.4.2.2 MIC筛选

经典的互信息（mutual information）也是评价定性自变量对定性因变量的相关性的，互信息表示为随机变量X与Y之间的互信息I(X;Y)为单个事件之间互信息的数学期望，互信息计算公式如下：



互信息直接用于特征筛选其实不是太方便：1、它不属于度量方式，也没有办法归一化，在不同数据集上的结果无法做比较；2、对于连续变量的计算不是很方便（X和Y都是集合，x，y都是离散的取值），通常变量需要先离散化，而互信息的结果对离散化的方式很敏感。

　　最大信息系数（maximal information coefficient, MIC）克服了这两个问题。它首先寻找一种最优的离散化方式，然后把互信息取值转换成一种度量方式，取值区间在[0,1]。minepy提供了MIC功能。

##### 6.2.4.2.3 PSI筛选

最常用的就是使用PSI（PopulationStability Index，群体稳定性指数）指标评估特征稳定性。计算公式如下：



其中：PSI是对两个日期的特征数据进行计算，可以任选其一作为base集，另一则是test集（也可称为expected集和actual集），用字母i表示第i个分段区间。

特征的PSI计算过程如下：

1. 特征取值等频分段：对这个特征在base集的取值进行等频划分（通常等频分10份即可）。
2. 计算：统计落在每个分段区间内的目标数量，进一步得到数量占比，表示该特征在base集中第i个取值分段中的数量占比。
3. 计算：按照（2）中的方式计算test集的。需要注意的是，分段还是采用第1步得到的分段（依据base集的分段）。
4. 根据公式即可计算得到该特征基于这两个日期的PSI。

通常，如果一个特征跨度6个月的PSI取值小于0.1，那么这个特征被认为是稳定的（当然，也可以根据具体情况适当放宽0.1的标准）。

注意：并非所有PSI值很高的特征都不能用于建模，如果一个特征区分度很好但PSI值不满足预期（比如跨度6个月的PSI大于0.1），但同时，该特征的取值波动性从业务的角度可以解释得通，那么这样的特征用于建模也是可以的。

### 6.2.5 自动化调参

业内普遍使用的调参策略是基于随机搜索、遗传算法、贝叶斯优化等形式实现的，这里介绍一种基于业务指标实现调参的思路，并通过代码实现自动化的参数搜索。

1. 自动化调参策略

业务期望模型的训练集KS值和时间外样本集KS值足够接近，且时间外样本集的KS值足够大。前者用于保证模型的跨时间稳定性不会很差，而后者用于保证模型的精度足够高。因此给出调参目标为两者的组合，如下：



其中，offks为时间外样本集KS值，devks训练集KS值，w为分配的权重。

注意，KS值得分配权重w可以根据实际情况调节。比如当业务稳定性较差时，应更多关注两者KS值的差值，因此需要将w从默认的0.2改为一个更大的值。

1. 参数搜索方案

参数搜索方案使用的是一种针对目标KS值的贪心搜索方法。每一次只考虑单个参数，进行前向和后向搜索，当对目标KS值有提高时，继续搜索，否则停止该方向的搜索。

### 6.2.6 递归特征删除

经过模型的筛选之后，并不能保证当前模型的入模变量已是最优变量组合，因此我们使用一种对变量进行精细化调整的递归特征删除方案。最有可能得到最优解的方案为，遍历每一种变量组合，得到KS值最大的一组特征组合。但考虑到这种方案实现效果较低，使用逐个特征删除的方案代替。

首先将一个特征从变量组合中去掉，观察模型KS值和PSI的变化：如果变量模型没有明显变化或者模型表现较好，则可以删除该特征；否则，保留该特征，继续去掉下一个特征并观察。

### 6.2.7 评分卡制作

#### 6.2.7.1 基本原理

根据XGBoost原理所述，我们训练的模型包括K颗树，模型为：



其中为包含所有回归树的函数空间，回归树是一个将属性映射到分数的函数。

对于二分类问题，我们需要把模型的分数映射到[0,1]之间，因此我们可以使用sigmoid函数，其二分类模型可以表达为：



即：



该模型的参数为每棵树的结构和叶子节点的分数，我们将函数作为参数，即：

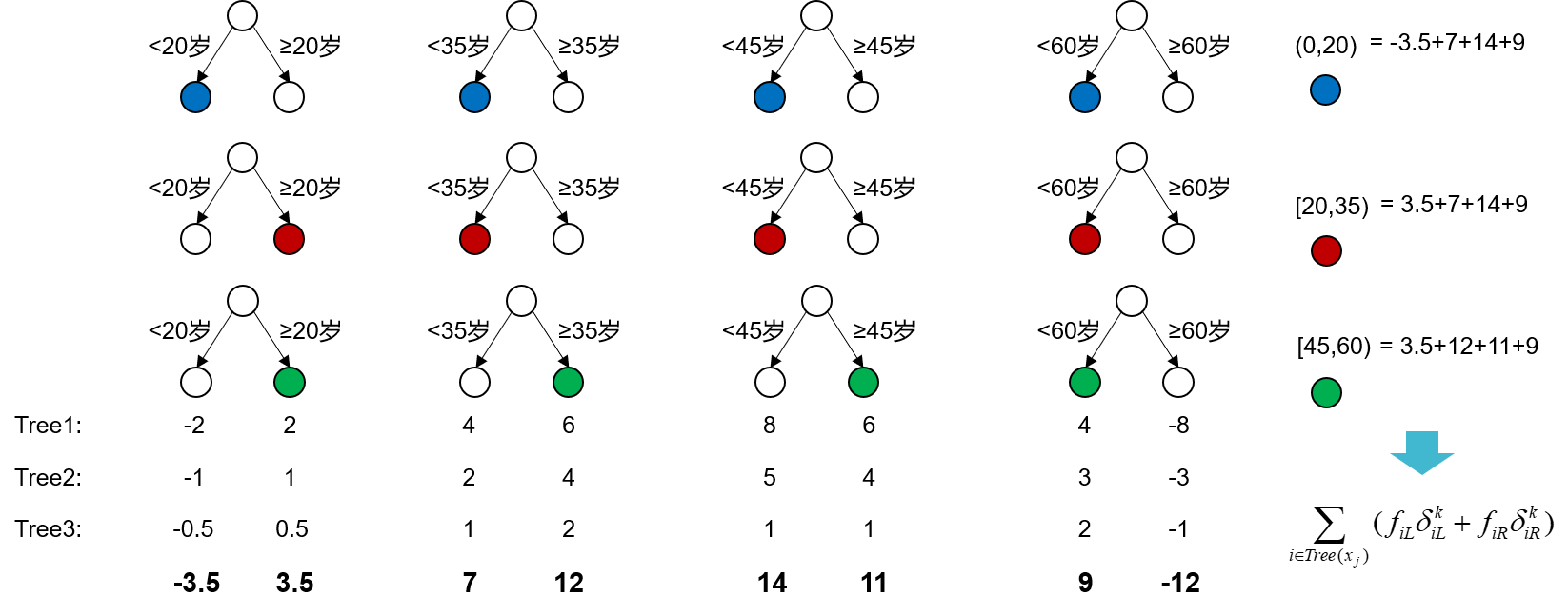


所以，该模型没有学习训练样本的权重，而是直接学习树的函数。

#### 6.2.7.2 输出分箱的方法

为了采用xgboost构建评分卡，我们使用仅有两个叶子的树在前一步的残差上训练梯度提升树（max\_depth=1），因此损失函数为：。对于每一个特征，我们找到在这些特征上分叉的所有的树和它们对应的分数，将这些分数相加，我们可以得到一个新的分段函数。例如，对于特征i，我们有一组，其中ik指使用i特征的第k棵树。由于我们使用的是两个叶子节点的树，除了i特征不包含其他的特征，此时我们将这棵决策树看作是将这个变量分箱为两个类，并给出相应分箱的分数。我们将这一组决策树的结果相加可以得到，同理，这也是一个分段函数，可以用分箱来表示最后的结果，因此它可以用来生成评分卡分数。

下面举例来说明：假设我们训练得到一个XGB模型，共有100颗子树。其中用年龄这个特征进行分裂的子树有12个，对年龄进行划分的节点有四个：20岁，35岁，45岁，60岁。为了便于说明，假设每个节点均包括3颗子树，子树的每个叶节点的分数如下：



对于如上子树，将相同分裂点的子树的叶节点分数相加（Tree1+Tree2+Tree3），得到不同子树及其叶节点分数。按照各个子树对应的特征值从小到大的顺序，对同一特征所对应的回归树进行排序，并将排序后的第一个子树的左侧叶子节点表示的数值区间、排序后的最后一个子树的右侧叶子节点表示的数值区间以及不同子树的相邻两个叶子节点表示的数值区间的交集作为特征分箱，叶节点分数的和就是各个分箱的分数。

#### 6.2.7.3 构建评分卡

我们可以定义比率来表示客户违约的相对概率：



将odds带入可得：



评分卡的分值可以定义为比率对数的线性表达，即：



其中，A与B是常数，B前面的负号可以使得违约概率越低，得分越高。通常情况下，即高分值代表低风险，低分值代表高风险。

A、B的值可以通过将两个已知或假设的分值带入计算得到。通常情况下，需要设定两个假设：

* 某个特定的违约概率下的预期评分，即比率odds为时的分数为；
* 该违约概率翻倍的评分（PDO）；

根据以上的分析，则odds为时的分数为，代入以上线性表达式，可得：



解该方程组，可得：



在实际的应用中，我们会计算出每个变量的各分箱对应的分值。新用户产生时，对应到每个分箱的值，将这些值相加，最后加上初始基础分，得到最终的结果。评分公式如下：



其中，变量是样本在每棵树中的得分。由于我们使用的是两个叶子节点的树，所以样本在每棵树上都会分到其中一个节点，所以可以得到如下形式：



其中，分别为梯度提升树中第i颗树左节点和右节点的得分；是0,1逻辑变量，表示样本在第i颗树中被划分到左节点，与此同时右节点的逻辑变量，有且仅有一个节点取1。

最终得到下表所示的评分卡模型的评分表：

表 评分卡模型的评分表

|  |  |  |
| --- | --- | --- |
| **变量** | **分箱类别** | **分值** |
| 基准分 | -- |  |
|  | 1 |  |
| 2 |  |
| … | … |
|  |  |
|  | 1 |  |
| 2 |  |
| … | … |
|  |  |
| … | … | … |

其中，表示采用第j个特征进行分裂的第i颗树；表示某个特征的第k个分箱在第i颗树的左节点上，表示某个特征的第k个分箱在第i颗树的右节点上。

从以上公式中，我们发现不同变量在每个分箱上的评分都可以表示为，也就是说影响每个分箱的因素包括三部分，分别为参数B、样本所在叶子节点以及样本所在叶子节点的得分。

#### 6.2.7.4 评分映射

评分映射的公式为：



由于，所以：



其中，p为集成模型的输出，A为基础分，B为。

### 6.2.8 自定义损失函数

我们常用的损失函数有MSE、MAE、LogLoss、HingeLoss等，但是，在实际业务中，这些"现成的"损失函数有时无法很好地适应我们试图解决的业务问题，这时就需要我们自定义损失函数。

XGBoost模型支持自定义评价函数和损失函数。只要保证损失函数二阶可导，通过评价函数的最大化即可对模型参数进行求解。可以考虑根据业务目标对二者进行调整。

在讨论之前，首先明确一下训练损失和验证损失的概念：

* **训练损失**。这是在训练数据上进行优化的函数。例如，在神经网络的二分类器中，这通常是交叉熵；对于随机森林分类器，这是基尼系数。训练损失通常也称为"目标函数"。
* **验证损失**。这是我们用来评估我们训练模型在看不见的数据上的性能的函数。这通常与训练损失不同。例如，在分类任务中，这通常是ROC曲线下的区域（AUC）——尽管这从未直接优化，因为它是不可微分的。这通常称为"性能或评估指标"。

在许多情况下，自定义这些损失对于构建更好的模型非常有效。而且对于梯度提升类算法特别简单。

#### 6.2.8.1 自定义训练损失

我们在训练期间优化训练损失。某些算法很难自定义，比如随机森林，但对其他算法来说比较容易，比如梯度提升和神经网络。因为这类方法都是使用梯度下降类算法进行优化，所以训练损失通常需要一个具有凸梯度（一阶导数）和海森（二阶导数）的函数。它最好是连续的，有限的和非零的，非零的要求很重要，因为函数为零的部分会冻结梯度下降。

在梯度提升的背景下，训练损失是使用梯度下降优化的函数，例如梯度提升模型的"梯度"部分。训练损失的梯度用于改变每个连续树的目标变量。这里需要注意，即使训练损失定义了"梯度"，每个树仍然使用与此自定义损失函数无关的贪婪分割算法生长。

自定义训练损失通常需要我们做一些微积分来找到梯度和海森。一般来说，首先更改验证损失更容易一些，因为它不需要那么多的开销。

#### 6.2.8.2 自定义验证损失

验证损失用于调整超参数。它通常更容易自定义，因为它没有像训练损失那么多的功能要求。验证损失可以是非凸的，不可微分的和不连续的。因此，从自定义验证损失开始通常是一个更容易的地方。

例如，在XGBoost中，一个重要的超参数是n\_estimators的数量。验证损失可用于找到最佳数量的boosting次数。XGBoost中的验证损失称为eval\_metric。我们可以使用库中可用的验证损失之一（rmse/logloss/auc等），也可以自定义函数。由于自定义函数非常简单，如果它对我们的业务问题很重要，那么一定要自定义验证函数。

具体而言，我们通常使用early\_stopping\_rounds变量，而不是直接优化boosting的轮数。当给定数量的早期停止轮次的验证损失开始增加时，它会停止提升。实际上，它通过监视样本外验证集的验证损失来防止过度拟合。如下图所示，设置更高的停止轮次会导致模型运行更多boosting轮次。

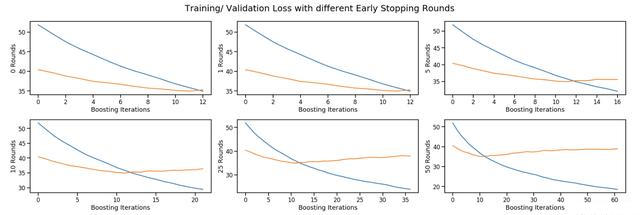


图2 不同早停轮次的训练损失和验证损失

注：其中，蓝色表示训练损失，橙色表示验证损失。训练和验证都使用相同的自定义损失功能。

如果适用于业务问题，我们希望使用自定义函数来进行训练和验证损失。在某些情况下，由于自定义损失的函数形式，可能无法将其用作训练损失。在这种情况下，仅更新验证损失并使用像MSE这样的默认训练损失也是有意义的。我们仍然可以从中获益，因为超参数将使用所需的自定义验证损失进行调整。

#### 6.2.8.3 如何自定义损失

这里我们看一个例子。首先，让我们假设过高估计比低估更糟糕。另外，假设平方损失是我们在任一方向上的误差的良好模型。为了对其进行编码，我们定义了一个自定义MSE函数，它对正残差的惩罚比负残差多10倍。下图说明了我们的自定义损失函数与标准MSE损失函数的对比情况。

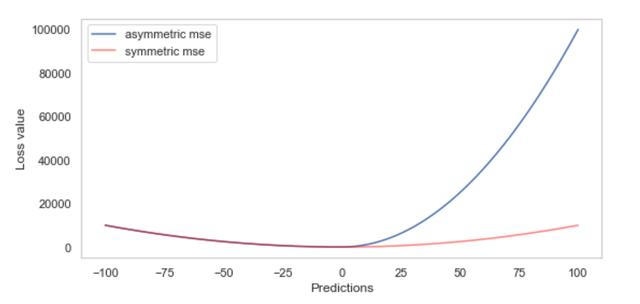


图3 自定义损失函数与标准MSE损失函数的对比

根据定义，非对称MSE很好，因为它具有易于计算的梯度和海森，如下图所示。请注意，海森在两个不同的值上是常量，左边是2，右边是20，尽管在下面的图中很难看到。

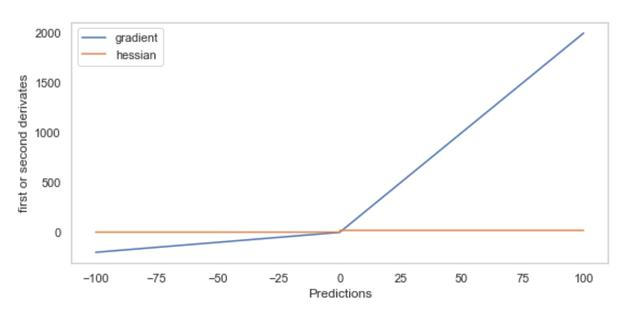


图4 梯度和海森

XGBoost提供了一种直接的方式来实现自定义训练和验证损失。其他梯度提升包，包括LightGBM和Catboost，也提供这个功能。

* 训练损失：在XGBoost中自定义训练损失需要定义一个函数，该函数包含两个数组，即训练数据集及其预测。反过来，该函数应该返回每个样本的梯度和海森数组。如上所述，我们需要使用微积分来导出梯度和海森，然后在Python中实现它。
* 验证损失：自定义XGBoost中的验证损失需要定义一个函数，该函数接受相同的两个数组，同样返回两个值：一个是metric名称的字符串，另一个是损失本身。

自定义训练和验证损失函数后，将两个函数分别赋值给XGBoost模型的obj和feval两个参数即可，其他过程不变。

对于上面的问题，其自定义损失分别为：

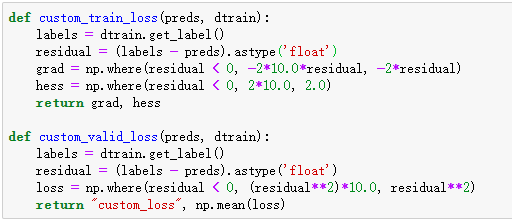


图5 自定义训练和验证损失函数

在XGBoost模型中使用自定义的训练和验证损失：

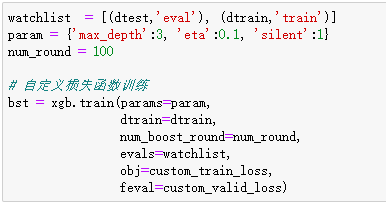


图6 XGBoost模型中使用自定义的训练和验证损失

#### 6.2.8.4 何时自定义损失

虽然有时候自定义验证和训练损失函数可以提高模型效果，但是，我们不应该直接使用自定义损失函数，最好采用迭代的方法。首先从像随机森林这样的简单基线模型开始；在下一次迭代中，你可以使用更复杂的模型，如XGBoost或LightGBM，并进行超参数优化。只有在这些基线模型稳定后，才有必要自定义验证和训练损失函数。其原因在于不和基线模型做比较，我们很难知道自定义损失函数是否会对模型有提升。