对外经济贸易大学
UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

前向分步算法与Boosting

信息学院-黄浩

2022年3月14日星期一

主要内容

一、加法模型与前向分步算法

二、Adaboost

三、GBDT

四、XGBoost

五、CatBoost与LGBM

加法模型与前向分步算法

$$f(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m)$$

其中，$b(x;\gamma_m)$称为基函数，γ_m称为基函数的参数，β_m称为基函数的系数。

在给定训练数据及损失函数$L(y,f(x))$的条件下，学习加法模型$f(x)$成为经验风险极小化问题，即损失函数极小化问题：

$$\min_{\beta_m,\gamma_m}\sum_{i=1}^{N}L\left(y_i,\sum_{m=1}^{M}\beta_m b(x_i;\gamma_m)\right)$$

随后，该问题可以作如此简化：从前向后，每一步只学习一个基函数及其系数，逐步逼近上式，即：每步只优化如下损失函数：

$$\min_{\beta,\gamma}\sum_{i=1}^{N}L(y_i,\beta b(x_i;\gamma))$$

加法模型与前向分步算法

前向分步算法的算法流程：

- **输入：训练数据集** $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$
- **损失函数：** $L(y, f(x))$
- **基函数集：** $\{b(x; \gamma)\}$
- **输出：加法模型** $f(x)$
- **算法步骤：**
  - **1. 初始化** $f_0(x) = 0$
  - **2. 对于m=1,2,..M**
  - **a)极小化损失函数**

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

加法模型与前向分步算法

b、更新

$$f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$$

3、最终得到加法模型

$$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m)$$

就这样，前向分步算法将同时求解从m=1到M的所有参数（β_m、γ_m）的优化问题简化为逐次求解各个β_m、γ_m（1≤m≤M）的优化问题。

Adaboost算法

AdaBoost算法是提升算法中最具代表性的。其中AdaBoost是Adaptive Boosting的缩写，在AdaBoost算法中会提高前一轮分类器分类错误的样本的权值，而降低那些被分类正确样本的权值。对于弱分类器的组合，AdaBoost算法采取加权多数表决的方法。具体的说就是加大分类误差率小的弱分类器的权值，使其在表决中起到较大的作用；减小分类误差率大的弱分类器的权值，使其在表决中起较小的作用。

Adaboost算法

1. 为数据集里的每个样本赋予相同的权重（一般情况），然后开始依据一定的分类标准来对该数据集分类，从而得到第一个Classifier[1]（弱分类器）。同时由此可得此次分类中被分错的样本，提高他们的权重，用于下一个Classifier[2]的训练。
2. 依据上一次Classifier训练更新样本权重后的样本集，来训练此次Classifier[2]。其中可以依据样本权重来影响此次训练的错误率来使此次Classifier[2]足够重视之前错分的样本。这样我们可以得到对于上次分错的样本有较好分类能力的Classifier[2]，然后提高本次分类中被分错的样本的权重。重复此过程，得到若干的弱分类器Classifier[i]。
3. 为1、2步中得到的Classifier[i]确定在最终的强分类器中的权重（此权重的确定也可以在1、2步中确定）。Strong_Classifier = ∑（ weight[i] *Classifier[i]）（弱分类器的线性组合），计算Strong_Classifier的错误率，然后综合考虑程序迭代次数来判断是否继续重复1-3步迭代训练。

Adaboost算法

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t} \left[h_t(x_i) \neq y_i \right].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

Figure 1: The boosting algorithm AdaBoost.

Adaboost算法

对于这个算法需要说明的是：

1.算法开始前，需要将每个样本的权重初始化为1/m,这样一开始每个样本都是等概率的分布，每个分类器都会公正对待。

2.开始迭代后，需要计算每个弱分类器的分类错误的误差，误差等于各个分错样本的权重和，这里就体现了样本权重的作用。如果一个分类器正确分类了一个权重大的样本，那么这个分类器的误差就会小，否则就会大。这样就对分类错误的样本更大的关注。

3.获取最优分类器后，需要计算这个分类器的权重，然后再更新各个样本的权重，然后再归一化。

4.算法迭代的次数一般不超过弱分类器的个数，如果弱分类器的个数非常之多，那么可以权衡自己性价比来折中选择。

5.迭代完成后，最后的分类器是由迭代过程中选择的弱分类器线性加权得到的。

Adaboost算法

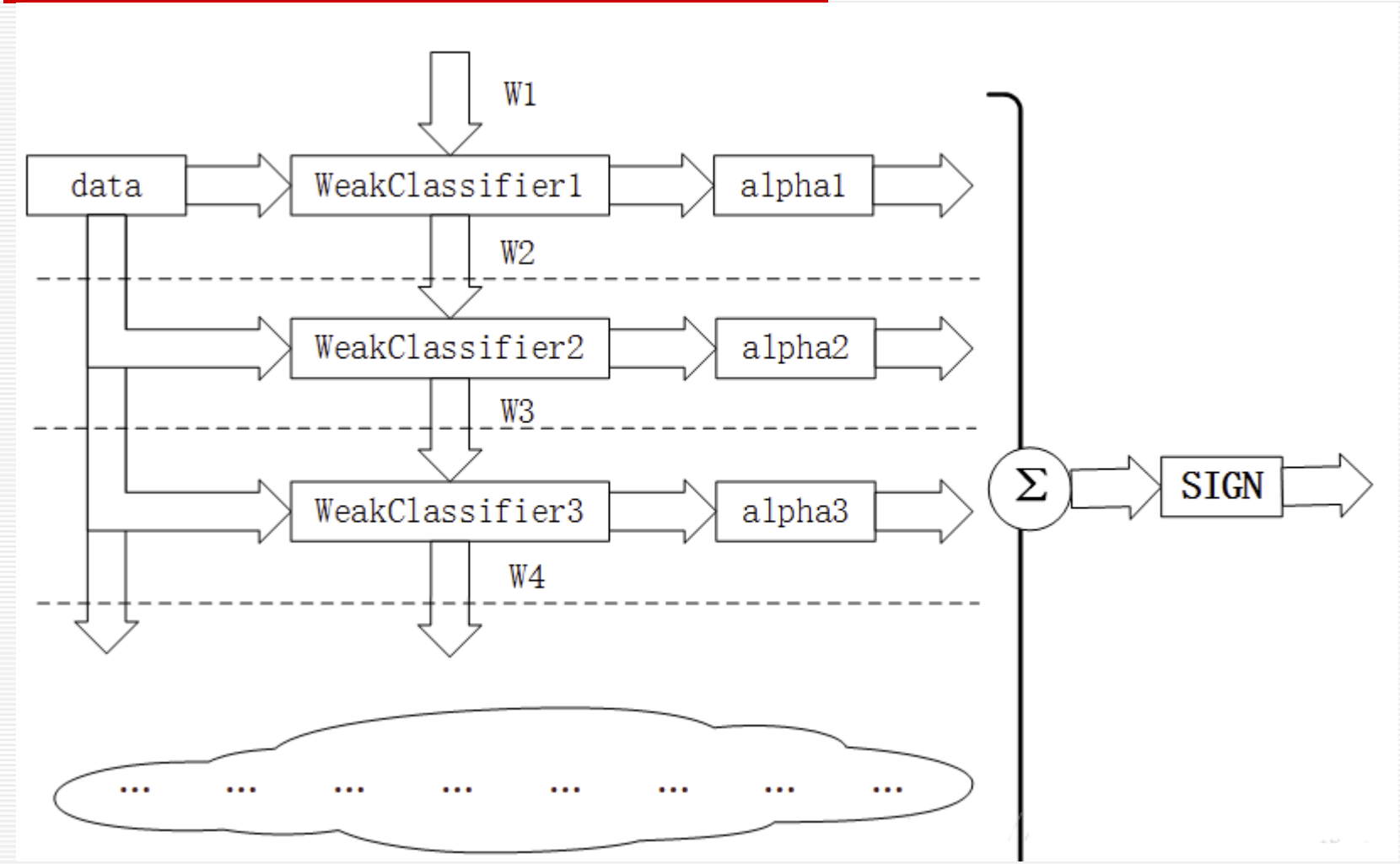
Adaboost算法中有两种权重：

- 一种是数据的权重
- 另一种是弱分类器的权重。

数据的权重主要用于弱分类器寻找其分类误差最小的决策点，找到之后用这个最小误差计算出该弱分类器的权重（发言权），分类器权重越大说明该弱分类器在最终决策时拥有更大的发言权。

在Adaboost算法中，每训练完一个弱分类器都就会调整权重，上一轮训练中被误分类的点的权重会增加，在本轮训练中，由于权重影响，本轮的弱分类器将更有可能把上一轮的误分类点分对，如果还是没有分对，那么分错的点的权重将继续增加，下一个弱分类器将更加关注这个点，尽量将其分对。这样，达到“你分不对的我来分”，下一个分类器主要关注上一个分类器没分对的点，每个分类器都各有侧重。

Adaboost算法


W1
data
WeakClassifier1
alpha1
W2
WeakClassifier2
alpha2
W3
WeakClassifier3
alpha3
W4
Σ
SIGN

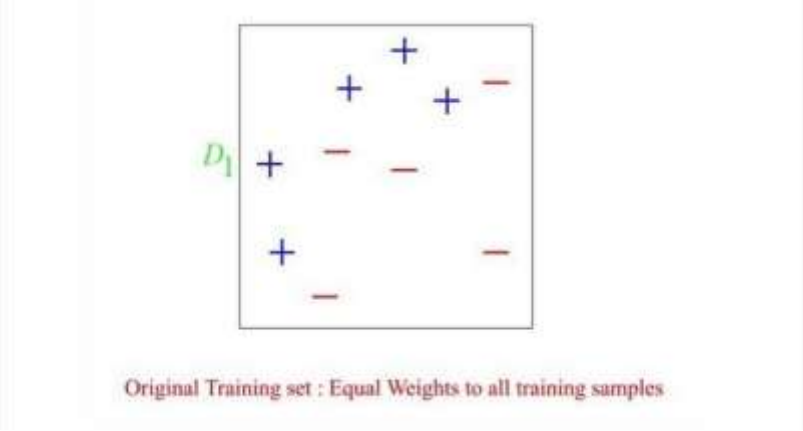
Adaboost算法

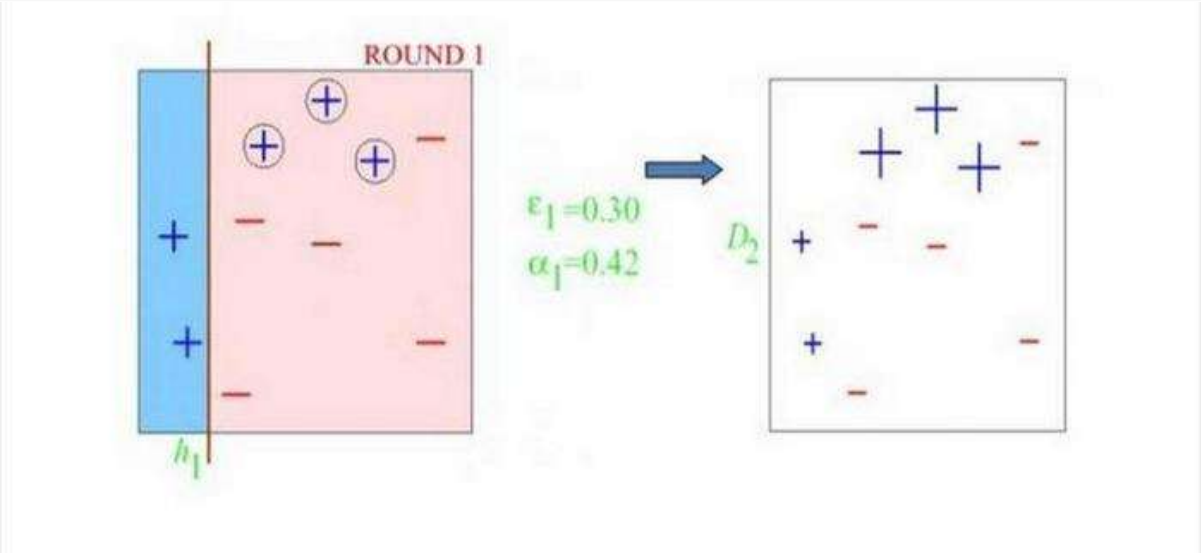
第i轮迭代要做这么几件事：
1.新增弱分类器WeakClassifier(i)与弱分类器权重alpha(i)
2.通过数据集data与数据权重W(i)训练弱分类器WeakClassifier(i)，并得出其分类错误率，以此计算出其弱分类器权重alpha(i)
3.通过加权投票表决的方法，让所有弱分类器进行加权投票表决的方法得到最终预测输出，计算最终分类错误率，如果最终错误率低于设定阈值（比如5%），那么迭代结束；如果最终错误率高于设定阈值，那么更新数据权重得到W(i+1)。

对于boosting算法，存在两个问题：
1.如何调整训练集，使得在训练集上训练的弱分类器得以进行；
2.如何将训练得到的各个弱分类器联合起来形成强分类器。

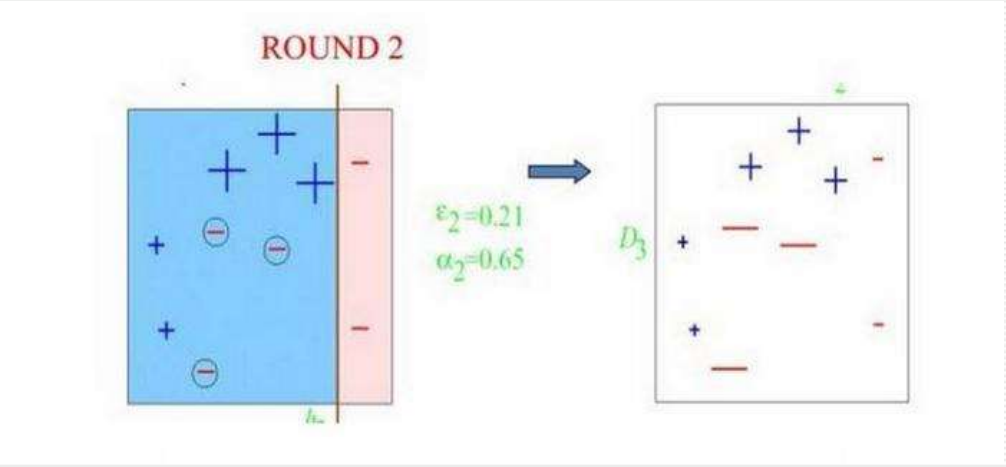
针对以上两个问题，AdaBoost算法进行了调整：
1.使用加权后选取的训练数据代替随机选取的训练样本，这样将训练的焦点集中在比较难分的训练数据样本上；
2.将弱分类器联合起来，使用加权的投票机制代替平均投票机制。让分类效果好的弱分类器具有较大的权重，而分类效果差的分类器具有较小的权重。

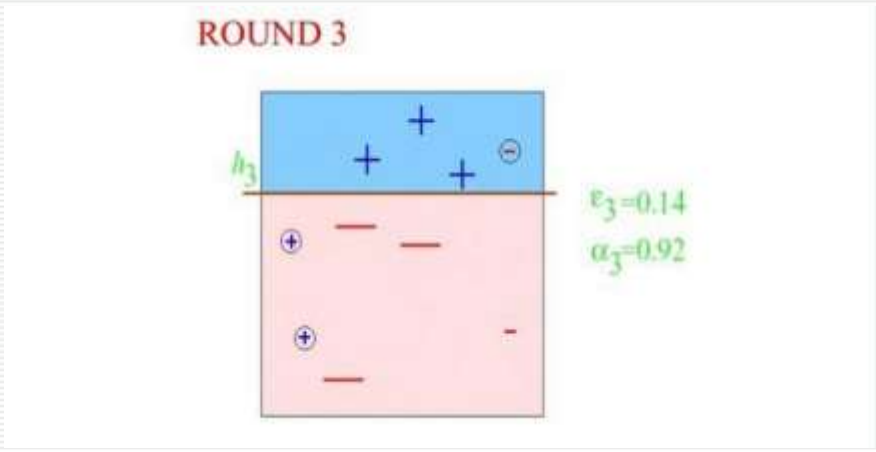
Adaboost算法

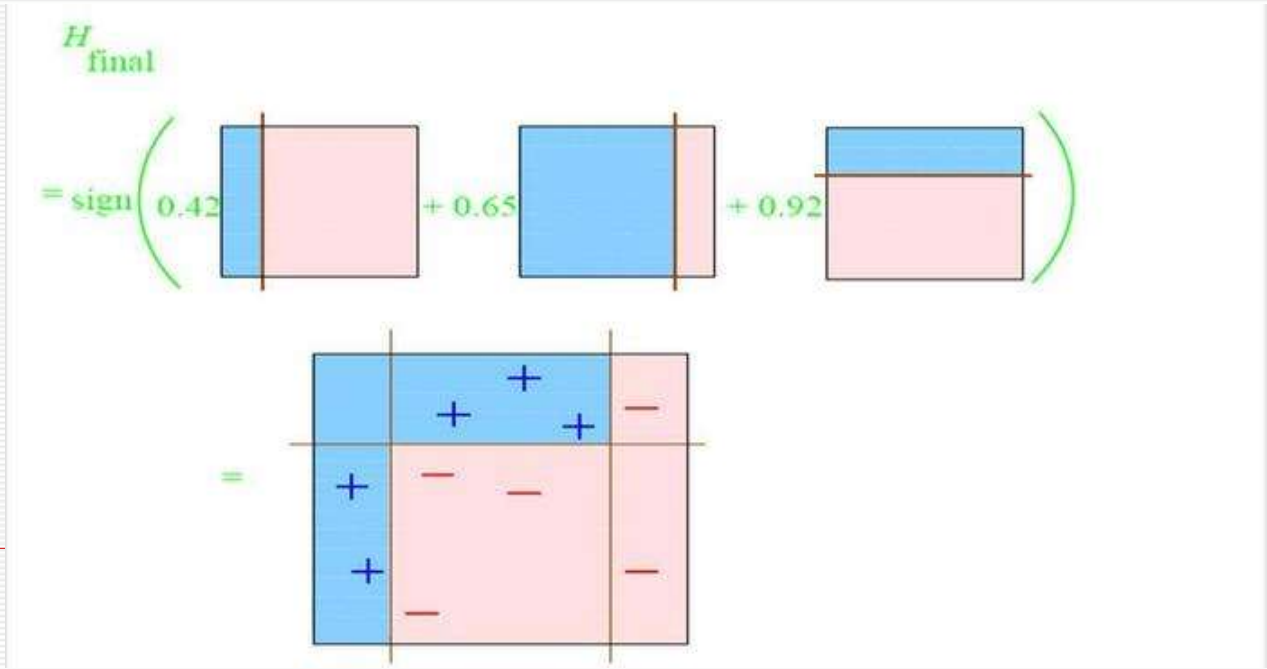

D_1
Original Training set : Equal Weights to all training samples


ROUND 1
h_1
ε_1 =0.30
α_1=0.42
D_2

Adaboost算法


ROUND 2
ε_2=0.21
α_2=0.65
D_3


ROUND 3
h_3
ε_3=0.14
α_3=0.92


H_{final}
= sign
0.42
+ 0.65
+ 0.92
=

Adaboost算法

- 优点：
- 可以使用各种方法构造子分类器，Adaboost算法提供的是框架
- 简单，不用做特征筛选
- 相比较于RF，更不用担心过拟合问题

- **缺点：**
- adboost对于噪音数据和异常数据是十分敏感的。Boosting方法本身对噪声点异常点很敏感，因此在每次迭代时候会给噪声点较大的权重，这不是系统所期望的。
- 运行速度慢，凡是涉及迭代的基本上都无法采用并行计算，Adaboost是一种“串行”算法，所以GBDT(Gradient Boosting Decision Tree)也非常慢。

GBDT算法

GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree)，是一种迭代的决策树算法，该算法由多棵回归树组成，所有树的结论累加起来做最终答案。它在被提出之初就和SVM一起被认为是泛化能力较强的算法。

GBDT是一种基于boosting集成学习（ensemble method）的算法，但和传统的Adaboost有很大不同。在Adaboost，我们是利用前一轮迭代弱学习器的误差率来更新训练集的权重，这样一轮轮的迭代下去。GBDT也是迭代，使用了前向分布算法，但是弱学习器限定了只能使用CART回归树模型，同时迭代思路和Adaboost也有所不同。GBDT的训练过程如下：

GBDT算法

提升树算法：

(1) 初始化$f_0(x) = 0$

(2) 对$m = 1, 2, ..., M$

(a) 计算残差

$$r_{mi} = y_i - f_{m-1}(x), i = 1, 2, ..., N$$

(b) 拟合残差r_{mi}学习一个回归树，得到$h_m(x)$

(c) 更新$f_m(x) = f_{m-1} + h_m(x)$

(3) 得到回归问题提升树

$$f_M(x) = \sum_{m=1}^{M} h_m(x)$$

GBDT算法

上面伪代码中的**残差**是什么?

在提升树算法中，假设我们前一轮迭代得到的强学习器是

$$f_{t-1}(x)$$

损失函数是

$$L(y, f_{t-1}(x))$$

我们本轮迭代的目标是找到一个弱学习器

$$h_t(x)$$

最小化让本轮的损失

$$L(y, f_t(x)) = L(y, f_{t-1}(x) + h_t(x))$$

当采用平方损失函数时

$$\begin{aligned} & L(y, f_{t-1}(x) + h_t(x)) \\ & = (y - f_{t-1}(x) - h_t(x))^2 \\ & = (r - h_t(x))^2 \end{aligned}$$

这里，

$$r = y - f_{t-1}(x)$$

是当前模型拟合数据的残差（residual）所以，对于提升树来说只需要简单地拟合当前模型的残差。

GBDT算法

当损失函数是平方损失和指数损失函数时，梯度提升树每一步优化是很简单的，但是对于一般损失函数而言，往往每一步优化起来不那么容易，针对这一问题，Freidman提出了梯度提升树算法，这是利用最速下降的近似方法，**其关键是利用损失函数的负梯度作为提升树算法中的残差的近似值。**

那么负梯度长什么样呢?

第t轮的第i个样本的损失函数的负梯度为:

$$-\left[\frac{\partial L(y, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)}$$

此时不同的损失函数将会得到不同的负梯度，如果选择平方损失

$$L(y, f(x_i)) = \frac{1}{2}(y - f(x_i))^2$$

负梯度为

$$-\left[\frac{\partial L(y, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)} = y - f(x_i)$$

此时我们发现GBDT的**负梯度就是残差**，所以说对于回归问题，我们要拟合的就是残差。

GBDT算法

GBDT算法：

（1）初始化弱学习器

$$f_0(x) = arg\ min_c \sum_{i=1}^{N} L(y_i, c)$$

（2）对$m = 1, 2, ..., M$有：

（a）对每个样本$i = 1, 2, ..., N$，计算负梯度，即残差

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

（b）将上步得到的残差作为样本新的真实值，并将数据(x_i, r_{im})，$i = 1, 2, ..N$作为下棵树的训练数据，得到一颗新的回归树$f_m(x)$其对应的叶子节点区域为$R_{jm}, j = 1, 2, ..., J$。其中J为回归树t的叶子节点的个数。

（c）对叶子区域$j = 1, 2, ..J$计算最佳拟合值

$$\Upsilon_{jm} = \underbrace{arg\ min}_{\Upsilon} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \Upsilon)$$

（d）更新强学习器

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} \Upsilon_{jm} I(x \in R_{jm})$$

（3）得到最终学习器

$$f(x) = f_M(x) = f_0(x) + \sum_{m=1}^{M} \sum_{j=1}^{J} \Upsilon_{jm} I(x \in R_{jm})$$

GBDT算法

GBDT中的树是回归树（不是分类树），GBDT用来做回归预测，调整后也可以用于分类。

GBDT主要由三个概念组成：Regression Decistion Tree（即DT)，Gradient Boosting（即GB)，Shrinkage (算法的一个重要演进分枝）。

GBDT算法

1、DT：回归树 Regression Decision Tree

提起决策树（DT, Decision Tree) 绝大部分人首先想到的就是C4.5分类决策树。但如果一开始就把GBDT中的树想成分类树，那就错了。千万不要以为GBDT是很多棵分类树。

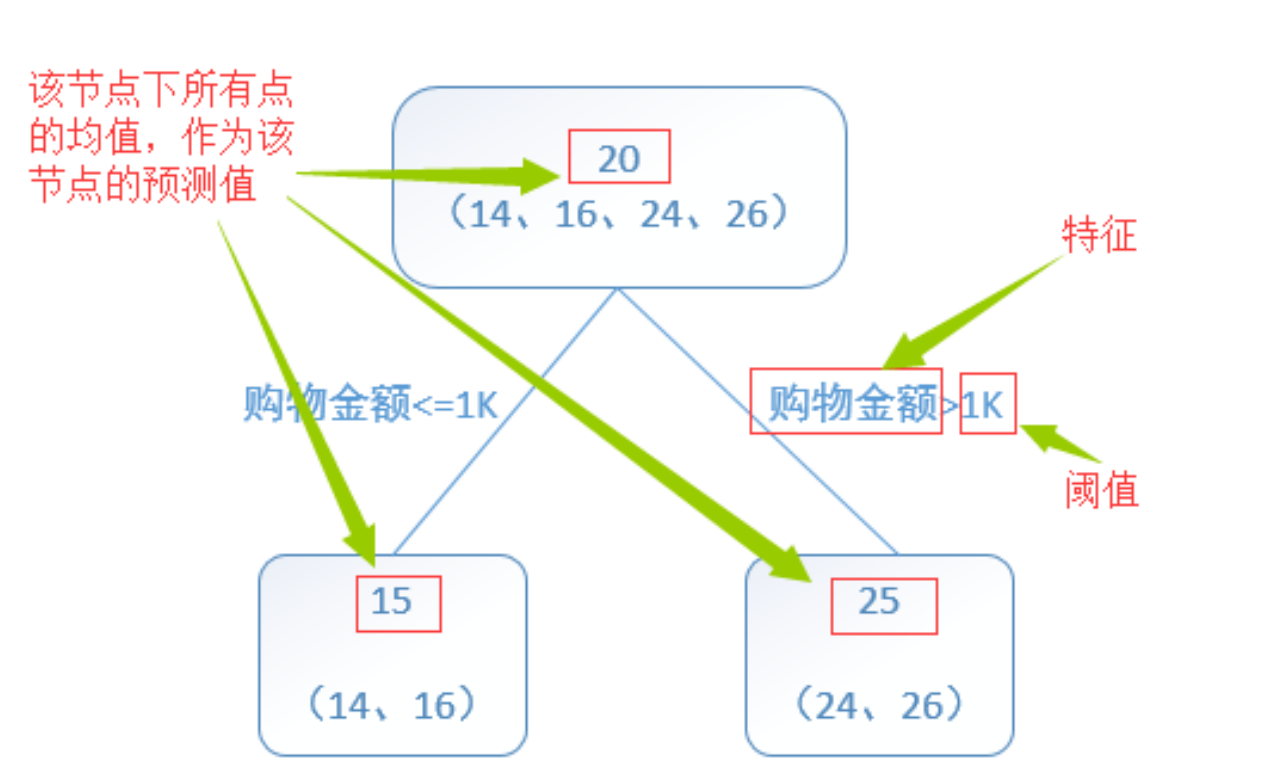
决策树分为两大类，回归树和分类树。前者用于预测实数值，如明天的温度、用户的年龄、网页的相关程度；后者用于分类标签值，如晴天/阴天/雾/雨、用户性别、网页是否是垃圾页面。

GBDT的核心在于累加所有树的结果作为最终结果，比如对年龄的累加（-3是加负3），而分类树的结果显然是没办法累加的，所以GBDT中的树都是回归树，不是分类树，尽管GBDT调整后也可用于分类但不代表GBDT的树是分类树。

GBDT算法

回归树总体流程类似于分类树，区别在于，回归树的每一个节点都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个feature的每个阈值找最好的分割点，但衡量最好的标准不再是最大熵，而是最小化平方误差。也就是被预测出错的人数越多，错的越离谱，平方误差就越大，通过最小化平方误差能够找到最可靠的分枝依据。分枝直到每个叶子节点上人的年龄都唯一或者达到预设的终止条件(如叶子个数上限)，若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。

GBDT算法


该节点下所有点的均值，作为该节点的预测值
20
（14、16、24、26）
特征
购物金额<=1K
购物金额>1K
阈值
15
（14、16）
25
（24、26）

GBDT算法

2、GB：梯度迭代 Gradient Boosting

梯度提升（Gradient boosting）是一种用于回归、分类和排序任务的机器学习技术，属于Boosting算法族的一部分。

Boosting是一族可将弱学习器提升为强学习器的算法，属于集成学习（ensemble learning）的范畴。

Boosting方法基于这样一种思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断要好。通俗地说，就是“三个臭皮匠顶个诸葛亮”的道理。

梯度提升同其他boosting方法一样，通过集成（ensemble）多个弱学习器，通常是决策树，来构建最终的预测模型。

GBDT算法

Boosting、bagging和stacking是集成学习的三种主要方法。

不同于bagging方法，boosting方法通过分步迭代（stage-wise）的方式来构建模型，在迭代的每一步构建的弱学习器都是为了弥补已有模型的不足。Boosting族算法的著名代表是AdaBoost。

AdaBoost算法通过给已有模型预测错误的样本更高的权重，使得先前的学习器做错的训练样本在后续受到更多的关注的方式来弥补已有模型的不足。

虽然同属于Boosting族，但是梯度提升方法的优点比较多。

GBDT算法

相比于AdaBoost，梯度提升方法的优点：

1、与AdaBoost算法不同，梯度提升方法在迭代的每一步构建一个能够沿着梯度最陡的方向降低损失（steepest-descent）的学习器来弥补已有模型的不足。
2、经典的AdaBoost算法只能处理采用指数损失函数的二分类学习任务，而梯度提升方法通过设置不同的可微损失函数可以处理各类学习任务（多分类、回归、Ranking等），应用范围大大扩展。
3、AdaBoost算法对异常点（outlier）比较敏感，而梯度提升算法通过引入bagging思想、加入正则项等方法能够有效地抵御训练数据中的噪音，具有更好的健壮性。

GBDT算法

提升树是迭代多棵回归树来共同决策。当采用平方误差损失函数时，每一棵回归树学习的是之前所有树的结论和残差，拟合得到一个当前的残差回归树，残差的意义如公式：残差 = 真实值 - 预测值 。提升树即是整个迭代过程生成的回归树的累加。

GBDT的核心就在于，每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。

GBDT算法

3. Gradient Boosting Decision Tree：

梯度提升决策树为什么梯度提升方法倾向于选择决策树（通常是CART树）作为基学习器呢？这与决策树算法自身的优点有很大的关系：

1、决策树可以认为是if-then规则的集合，易于理解，可解释性强，预测速度快；
2、决策树算法相比于其他的算法需要更少的特征工程，比如可以不用做特征标准化，可以很好的处理字段缺失的数据，也可以不用关心特征间是否相互依赖等
3、决策树能够自动组合多个特征，它可以毫无压力地处理特征间的交互关系并且是非参数化的，因此你不必担心异常值或者数据是否线性可分。举个例子，西瓜a（乌黑色、纹路清晰）可能是好瓜，西瓜b（青绿色，纹路清晰）的也可能是好瓜。决策树一样可以处理。

GBDT算法

决策树有优点，自然也有缺点，不过，可以通过梯度提升方法解决这个缺点。

单独使用决策树算法时，有容易过拟合缺点。

通过各种方法，抑制决策树的复杂性，降低单棵决策树的拟合能力通过梯度提升的方法集成多个决策树，则预测效果上来的同时，也能够很好的解决过拟合的问题。（这一点具有**bagging**的思想，降低单个学习器的拟合能力，提高方法的泛化能力。）

由此可见，梯度提升方法和决策树学习算法可以互相取长补短，是一对完美的搭档。

GBDT算法

抑制单颗决策树的复杂度的方法有很多：

1、限制树的最大深度、限制叶子节点的最少样本数量、限制节点分裂时的最少样本数量
2、吸收**bagging**的思想对训练样本采样（**subsample**），在学习单颗决策树时只使用一部分训练样本
3、借鉴随机森林的思路在学习单颗决策树时只采样一部分特征
4、在目标函数中添加正则项惩罚复杂的树结构等。

现在主流的**GBDT**算法实现中这些方法基本上都有实现，因此**GBDT**算法的超参数还是比较多的，应用过程中需要精心调参，并用交叉验证的方法选择最佳参数。
提升树利用加法模型和前向分步算法实现学习的优化过程。当损失函数是平方损失和指数损失函数时，每一步的优化很简单，如平方损失函数学习残差回归树。

GBDT算法

4. Shrinkage：

Shrinkage（缩减）的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。用方程来看更清晰，即
没用Shrinkage时：（yi表示第i棵树上y的预测值， y(1~i)表示前i棵树y的综合预测值）。

y(i+1) = 残差(y1~yi)， 其中： 残差(y1~yi) = y真实值 - y(1 ~ i)
y(1 ~ i) = SUM(y1, ..., yi)
Shrinkage不改变第一个方程，只把第二个方程改为：
y(1 ~ i) = y(1 ~ i-1) + step * yi

GBDT算法

4. Shrinkage：

即Shrinkage仍然以残差作为学习目标，但对于残差学习出来的结果，只累加一小部分（step残差）逐步逼近目标，step一般都比较小，如0.01~0.001（注意该step非gradient的step），导致各个树的残差是渐变的而不是陡变的。直觉上这也很好理解，不像直接用残差一步修复误差，而是只修复一点点，其实就是把大步切成了很多小步。本质上，Shrinkage为每棵树设置了一个weight，累加时要乘以这个weight，但和Gradient并没有关系*。 这个weight就是step。就像Adaboost一样，Shrinkage能减少过拟合发生也是经验证明的。

GBDT算法

实例详解：

如下表所示：一组数据，特征为年龄、体重，身高为标签值。共有5条数据，前四条为训练样本，最后一条为要预测的样本。

编号	年龄(岁)	体重（kg）	身高(m)(标签值)
1	5	20	1.1
2	7	30	1.3
3	21	70	1.7
4	30	60	1.8
5(要预测的)	25	65	?

参数设置：

- 学习率：learning_rate=0.1
- 迭代次数：n_trees=10
- 树的深度：max_depth=3

GBDT算法

1.初始化弱学习器:

$$f_0(x) = arg\ min_c \sum_{i=1}^{N} L(y_i, c)$$

损失函数为平方损失，因为平方损失函数是一个凸函数，直接求导，倒数等于零，得到c。

$$\sum_{i=1}^{N} \frac{\partial L(y_i, c))}{\partial c} = \sum_{i=1}^{N} \frac{\partial (\frac{1}{2}(y_i - c)^2)}{\partial c} = \sum_{i=1}^{N} c - y_i$$

令导数等于0

$$\sum_{i=1}^{N} c - y_i = 0$$

$$c = (\sum_{i=1}^{N} y_i)/N$$

所以初始化时，c取值为所有训练样本标签值的均值。$c = (1.1 + 1.3 + 1.7 + 1.8)/4 = 1.475$，此时得到初始学习器$f_0(x)$

$$f_0(x) = c = 1.475$$

GBDT算法

2.对迭代轮数m=1，2,...,M:

由于我们设置了迭代次数：n_trees=10，这里的$M = 10$。

计算负梯度，根据上文损失函数为平方损失时，负梯度就是残差残差，再直白一点就是 y与上一轮得到的学习器f_{m-1}的差值

$$r_{i1} = -\left[\frac{\partial L(y_i, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_0(x)}$$

残差在下表列出：

编号	真实值	$f_0(x)$	残差
1	1.1	1.475	-0.375
2	1.3	1.475	-0.175
3	1.7	1.475	0.225
4	1.8	1.475	0.325

GBDT算法

此时将残差作为样本的真实值来训练弱学习器$f_1(x)$，即下表数据

编号	年龄(岁)	体重（kg)	标签值
1	5	20	-0.375
2	7	30	-0.175
3	21	70	0.225
4	30	60	0.325

接着，寻找回归树的最佳划分节点，遍历每个特征的每个可能取值。从年龄特征的5开始，到体重特征的70结束，分别计算分裂后两组数据的平方损失（Square Error），SE_l左节点平方损失，SE_r右节点平方损失，找到使平方损失和$SE_{sum} = SE_l + SE_r$最小的那个划分节点，即为最佳划分节点。

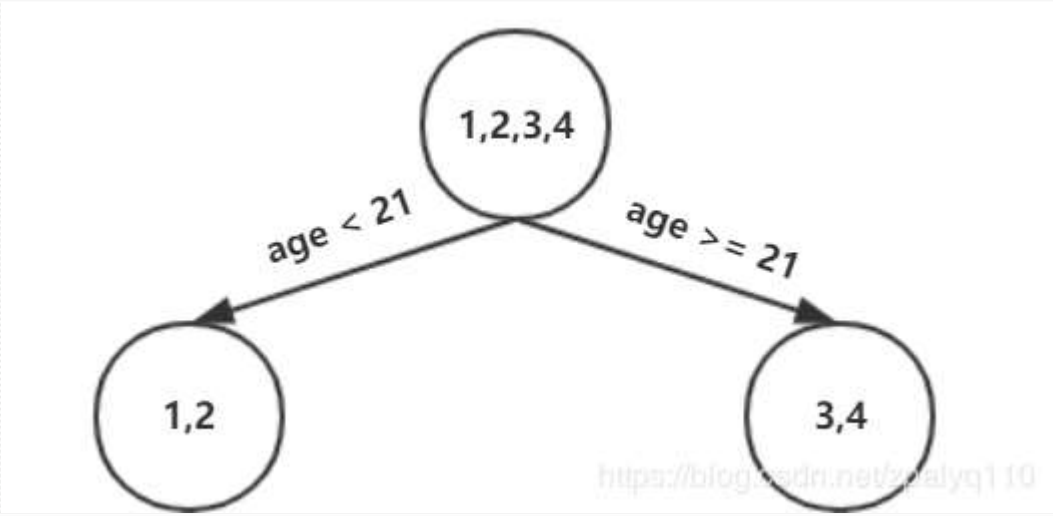
例如：以年龄7为划分节点，将小于7的样本划分为到左节点，大于等于7的样本划分为右节点。左节点包括x_1，右节点包括样本x_2, x_3, x_4，$SE_l = 0, SE_r = 0.047, SE_{sum} = 0.047$，所有可能划分情况如下表所示：

GBDT算法

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄5	/	1，2，3，4	0	0.327	**0.327**
年龄7	1	2，3，4	0	0.140	**0.140**
年龄21	1，2	3，4	0.020	0.005	**0.025**
年龄30	1，2，3	4	0.187	0	**0.187**
体重20	/	1，2，3，4	0	0.327	**0.327**
体重30	1	2，3，4	0	0.140	**0.140**
体重60	1，2	3，4	0.020	0.005	**0.025**
体重70	1，2，4	3	0.260	0	**0.260**

以上划分点是的总平方损失最小为**0.025**有两个划分点：年龄21和体重60，所以随机选一个作为划分点，这里我们选 **年龄21**
现在我们的第一棵树长这个样子：

GBDT算法



GBDT算法

我们设置的参数中树的深度max_depth=3，现在树的深度只有2，需要再进行一次划分，这次划分要对左右两个节点分别进行划分：

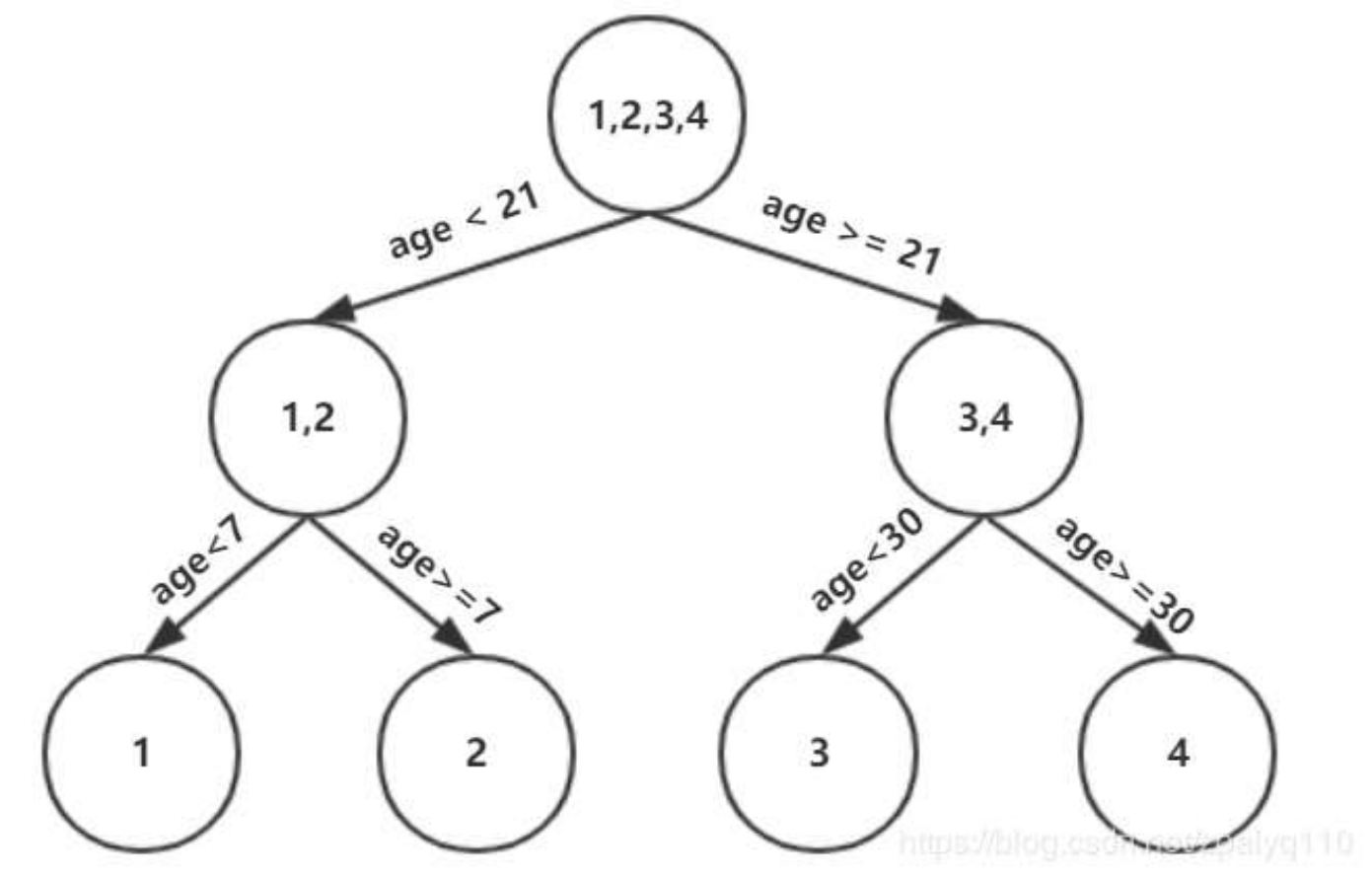
对于**左节点**，只含有1,2两个样本，根据下表我们选择**年龄7**划分

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄5	/	1，2	0	0.020	**0.020**
年龄7	1	2	0	0	**0**
体重20	/	1，2	0	0.020	**0.020**
体重30	1	2	0	0	**0**

对于**右节点**，只含有3,4两个样本，根据下表我们选择**年龄30**划分

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄21	1，2	3，4	0	0.005	**0.005**
年龄30	1，2，3	4	0	0	**0**
体重60	1，2	3，4	0	0.005	**0.005**
体重70	1，2，4	3	0	0	**0**

GBDT算法



GBDT算法

此时我们的树深度满足了设置，还需要做一件事情，给这每个叶子节点分别赋一个参数Υ，来拟合残差。

$$\Upsilon_{j1} = \underbrace{arg\ min}_{\Upsilon} \sum_{x_i \in R_{j1}} L(y_i, f_0(x_i) + \Upsilon)$$

这里其实和上面初始化学习器是一个道理，平方损失，求导，令导数等于零，化简之后得到每个叶子节点的参数Υ，其实就是标签值的均值。这个地方的标签值不是原始的 y，而是本轮要拟合的标残差 $y - f_0(x)$.

根据上述划分结果，为了方便表示，规定从左到右为第$1, 2, 3, 4$个叶子结点

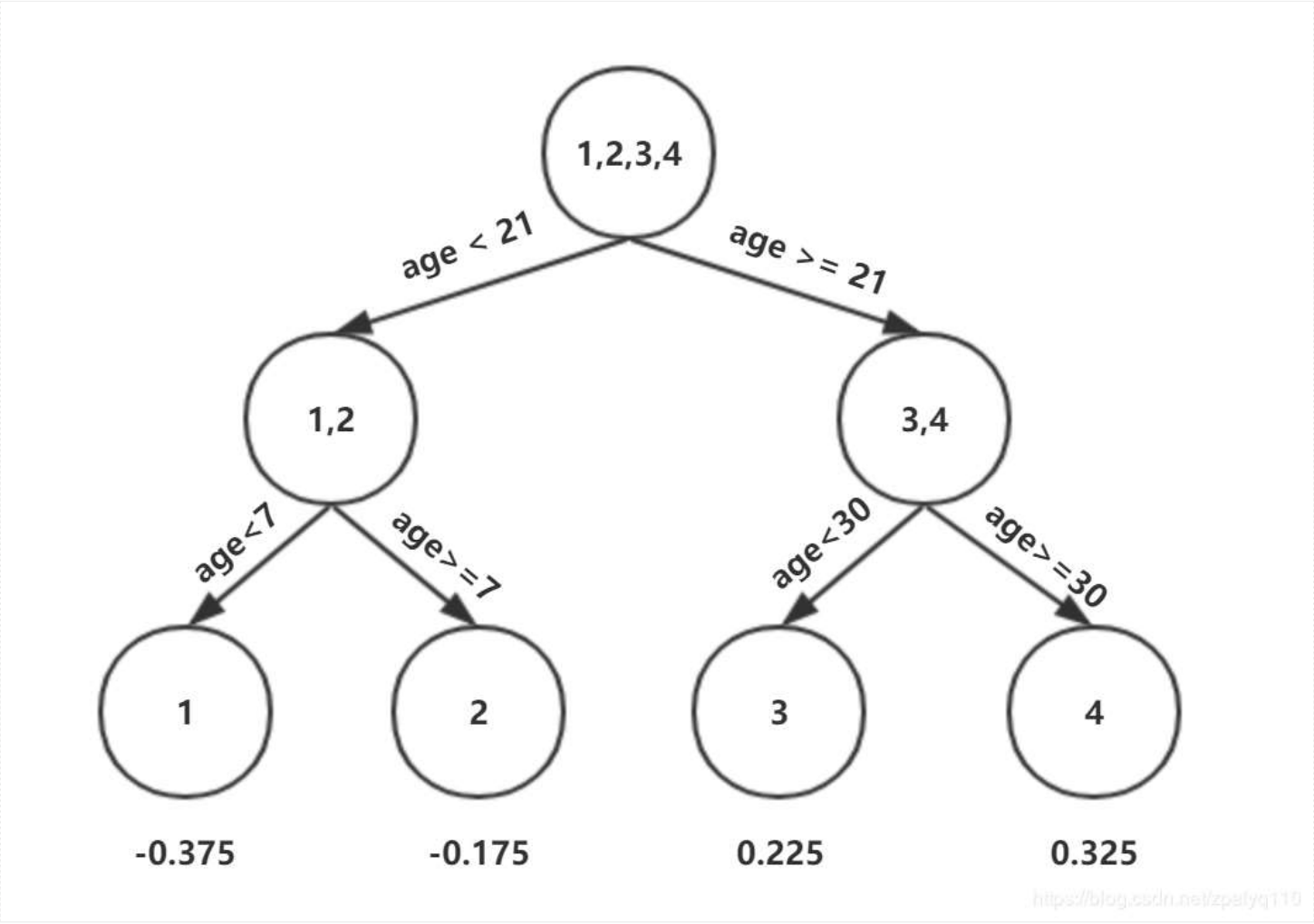
$$(x_1 \in R_{11}),\ \Upsilon_{11} = -0.375$$

$$(x_2 \in R_{21}),\ \Upsilon_{21} = -0.175$$

$$(x_3 \in R_{31}),\ \Upsilon_{31} = 0.225$$

$$(x_4 \in R_{41}),\ \Upsilon_{41} = 0.325$$

GBDT算法



GBDT算法

此时可更新强学习器，需要用到参数学习率：learning_rate=0.1，用lr表示。

$$f_1(x) = f_0(x) + lr * \sum_{j=1}^{4} \Upsilon_{j1} I(x \in R_{j1})$$

为什么要用学习率呢？这是**Shrinkage**的思想，如果每次都全部加上（学习率为1）很容易一步学到位导致过拟合。

重复此步骤，直到 $m > 10$ 结束，最后生成10棵树。

4.得到最后的强学习器：

$$f(x) = f_{10}(x) = f_0(x) + \sum_{m=1}^{10} \sum_{j=1}^{4} \Upsilon_{jm} I(x \in R_{jm})$$

GBDT算法

5.预测样本5：

$f_0(x) = 1.475$

在$f_1(x)$中，样本5的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.225**。

在$f_2(x)$中，样本5的...此处省略...所以被预测为**0.202**

为什么是0.202？这是根据第二颗树得到的，可以GitHub简单运行一下代码

在$f_3(x)$中，样本5的...此处省略...所以被预测为**0.182**

在$f_4(x)$中，样本5的...此处省略...所以被预测为**0.164**

在$f_5(x)$中，样本5的...此处省略...所以被预测为**0.147**

在$f_6(x)$中，样本5的...此处省略...所以被预测为**0.132**

在$f_7(x)$中，样本5的...此处省略...所以被预测为**0.119**

在$f_8(x)$中，样本5的...此处省略...所以被预测为**0.107**

在$f_9(x)$中，样本5的...此处省略...所以被预测为**0.096**

在$f_{10}(x)$中，样本5的...此处省略...所以被预测为**0.087**

最终预测结果：

$$f(x) = 1.475 + 0.1 * (0.225 + 0.202 + 0.182 + 0.164 + 0.147 + 0.132 + 0.119 + 0.107 + 0.096 + 0.087) = 1.621547$$

XGBoost算法

XGBoost在计算速度和准确率上，较**GBDT**有明显的提升，**xgboost** 的全称是**eXtreme Gradient Boosting**，它是**Gradient Boosting Machine**的一个**c++**实现，作者为正在华盛顿大学研究机器学习的大牛陈天奇最初开发的实现可扩展，便携，分布式 **gradient boosting (GBDT, GBRT or GBM)** 算法的一个库。

XGBoost最大的特点在于，它能够自动利用**CPU**的多线程进行并行，同时在算法上加以改进提高了精度。

XGBoost 所应用的算法就是 **gradient boosting decision tree**，既可以用于分类也可以用于回归问题中。

Gradient boosting 就是通过加入新的弱学习器，来努力纠正前面所有弱学习器的残差，最终这样多个学习器相加在一起用来进行最终预测，准确率就会比单独的一个要高。之所以称为 **Gradient**，是因为在添加新模型时使用了梯度下降算法来最小化的损失。

XGBoost算法

表现快是因为它具有这样的设计：
Parallelization：训练时可以用所有的 **CPU** 内核来并行化建树。
Distributed Computing ：用分布式计算来训练非常大的模型。
Out-of-Core Computing：对于非常大的数据集还可以进行 **Out-of-Core Computing**。
Cache Optimization of data structures and algorithms：更好地利用硬件。

XGBoost算法

XGBoost对GBDT的改进

Xgboost是GB算法的高效实现，xgboost中的基学习器除了可以是CART（gbtree）也可以是线性分类器（gblinear），传统GBDT以CART作为基分类器，xgboost支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

XGBoost算法

1．避免过拟合

目标函数之外加上了正则化项整体求最优解，用以权衡目标函数的下降和模型的复杂程度，避免过拟合。基学习为**CART**时，正则化项与树的叶子节点的数量**T**和叶子节点的值有关。

在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的**score**的**L2**模的平方和。从**Bias-variance trade-off**角度来讲，正则项降低了模型的**variance**，使学习出来的模型更加简单，防止过拟合，这也是**xgboost**优于传统**GBDT**的一个特性。

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

XGBoost算法

2．二阶的泰勒展开，精度更高

不同于传统的GBDT只利用了一阶的导数信息的方式，XGBoost对损失函数做了二阶的泰勒展开，精度更高。

第t次的损失函数：

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

对上式做二阶泰勒展开(g为一阶导数，h为二阶导数)：

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

$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

XGBoost算法

3．树节点分裂优化

选择候选分割点针对GBDT进行了多个优化。正常的树节点分裂时公式如下：

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

XGBoost算法

对于特征的值有缺失的样本，xgboost可以自动学习出它的分裂方向。XGBoost树节点分裂时，虽然也是通过计算分裂后的某种值减去分裂前的某种值，从而得到增益。但是相比GBDT，它做了如下改进：

1、通过添加阈值gamma进行了剪枝来限制树的生成
2、通过添加系数lambda对叶子节点的值做了平滑，防止过拟合。
3、在寻找最佳分割点时，考虑传统的枚举每个特征的所有可能分割点的贪心法效率太低，XGBoost实现了一种近似的算法，即：根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。
4、特征列排序后以块的形式存储在内存中，在迭代中可以重复使用；虽然boosting算法迭代必须串行，但是在处理每个特征列时可以做到并行。

XGBoost算法

xgboost算法的步骤和GB基本相同，都是首先初始化为一个常数，gb是根据一阶导数ri，xgboost是根据一阶导数gi和二阶导数hi，迭代生成基学习器，相加更新学习器。

xgboost与gdbt除了上述三点的不同，xgboost在实现时还做了许多优化：

1、在寻找最佳分割点时，考虑传统的枚举每个特征的所有可能分割点的贪心法效率太低，xgboost实现了一种近似的算法。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。
2、xgboost考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率。

XGBoost算法

xgboost与gdbt除了上述三点的不同，xgboost在实现时还做了许多优化：

3、特征列排序后以块的形式存储在内存中，在迭代中可以重复使用；虽然boosting算法迭代必须串行，但是在处理每个特征列时可以做到并行。
4、按照特征列方式存储能优化寻找最佳的分割点，但是当以行计算梯度数据时会导致内存的不连续访问，严重时会导致cache miss，降低算法效率。paper中提到，可先将数据收集到线程内部的buffer，然后再计算，提高算法的效率。
5、xgboost 还考虑了当数据量比较大，内存不够时怎么有效的使用磁盘，主要是结合多线程、数据压缩、分片的方法，尽可能的提高算法的效率。

XGBoost算法

不同的机器学习模型适用于不同类型的任务。深度神经网络通过对时空位置建模，能够很好地捕获图像、语音、文本等高维数据。而基于树模型的XGBoost则能很好地处理表格数据，同时还拥有一些深度神经网络所没有的特性（如：模型的可解释性、输入数据的不变性、更易于调参等）。

这两类模型都很重要，并广泛用于数据科学竞赛和工业界。举例来说，几乎所有采用机器学习技术的公司都在使用tree boosting，同时XGBoost已经给业界带来了很大的影响。

集成学习-XGBoost

- **基本构成**：boosted tree作为有监督学习算法有几个重要部分：模型、参数、目标函数、优化算法
- **模型**：模型指给定输入x如何去预测输出y
- **参数**：参数指需要学习的东西，在线性模型中，参数指线性系数w
- **目标函数**：目标函数：损失 + 正则，寻找一个比较好的参数

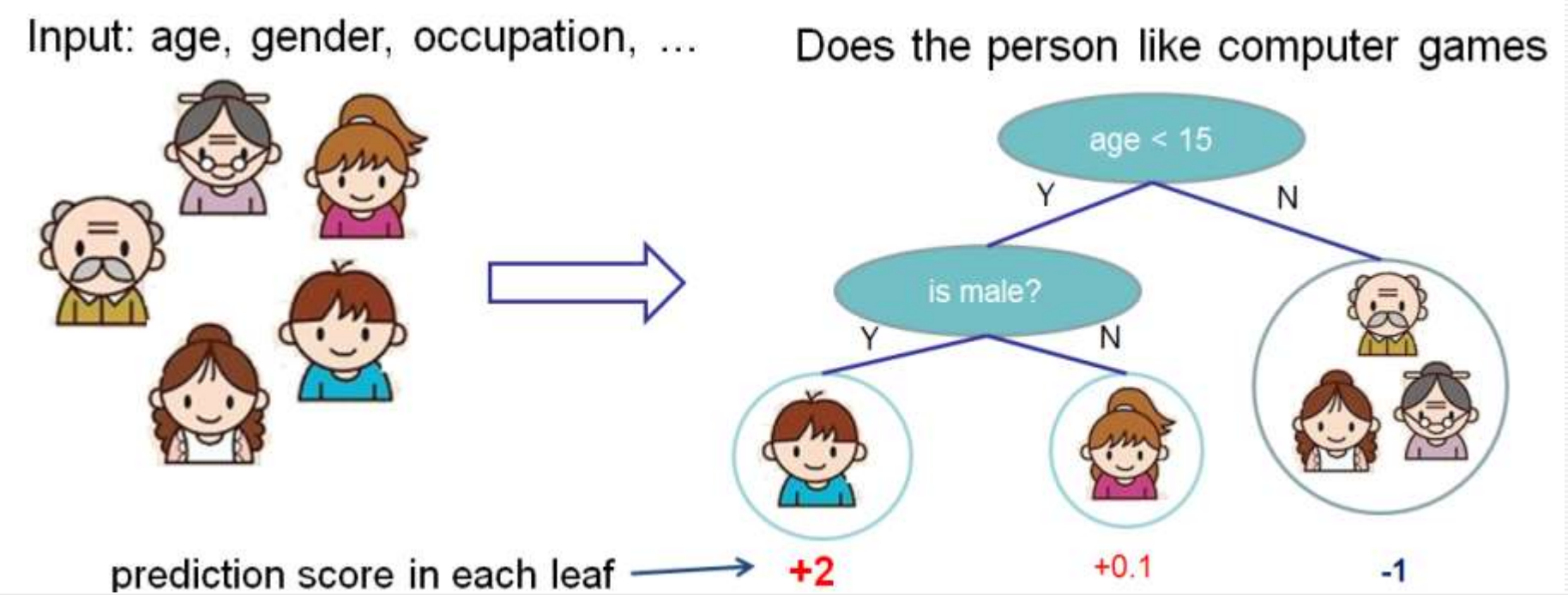
$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

误差函数： 我们的模型有多拟合数据

正则化项：惩罚复杂模型

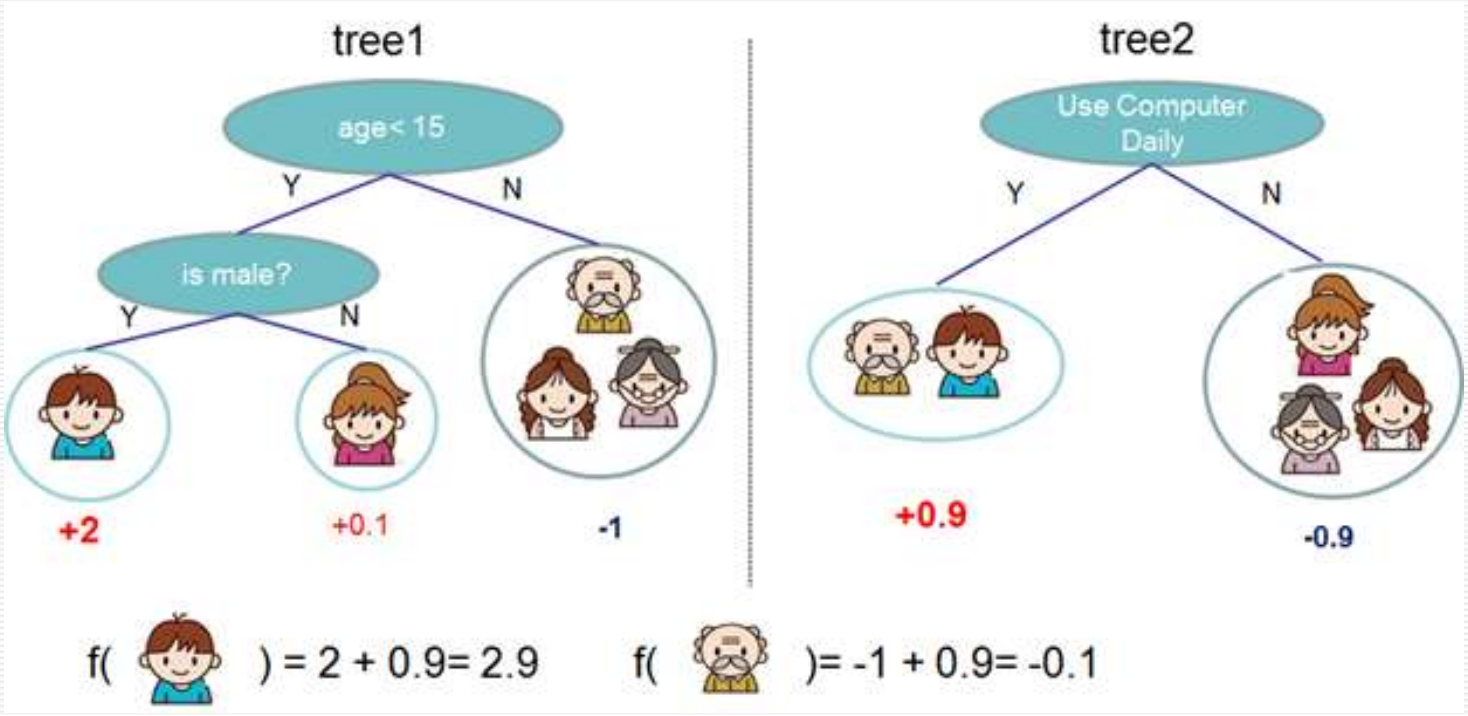
集成学习-XGBoost

基学习器：分类和回归树（CART）：CART会把输入根据输入的属性分配到各个叶子节点，而每个叶子节点上面都会对应一个实数分数。



集成学习-XGBoost

一个CART往往过于简单无法有效地预测，因此一个更加强力的模型叫做tree ensemble。用两棵树来进行预测。对于每个样本的预测结果就是每棵树预测分数的和。



集成学习-XGBoost

预测函数：

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

目标函数：

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

集成学习-XGBoost

第一部分是训练误差，第二部分是每棵树的复杂度的和。
每一次保留原来的模型不变，加入一个新的函数f到我们的模型中。

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

加入一个新的函数

第t轮的模型预测

保留前面t-1轮的模型预测

集成学习-XGBoost

选取一个f，使得目标函数尽量最大地降低(加入f后的预测结果与实际结果误差减少)。

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

目标：找到 f_t 来优化这一目标

对于*l*是平方误差时

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

一般叫做残差

集成学习-XGBoost

对于*l*不是平方误差的情况：
采用如下的泰勒展开近似来定义一个近似的目标函数

- 目标 $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
- 用泰勒展开来近似我们原来的目标
  - 泰勒展开：$f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
  - 定义：$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}),\quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

集成学习-XGBoost

移除常数项(真实值与上一轮的预测值之差)，目标函数只依赖于每个数据点的在误差函数上的一阶导数和二阶导数

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

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

集成学习-XGBoost

树的复杂度

- 以上是目标函数中训练误差的部分，接下来定义树的复杂度。
- 对于f的定义做一下细化，把树拆分成结构函数q(输入x输出叶子节点索引)和叶子权重部分w(输入叶子节点索引输出叶子节点分数)。
- 结构函数q把输入映射到叶子的索引号上面去，而w给定了每个索引号对应的叶子分数是什么。

集成学习-XGBoost

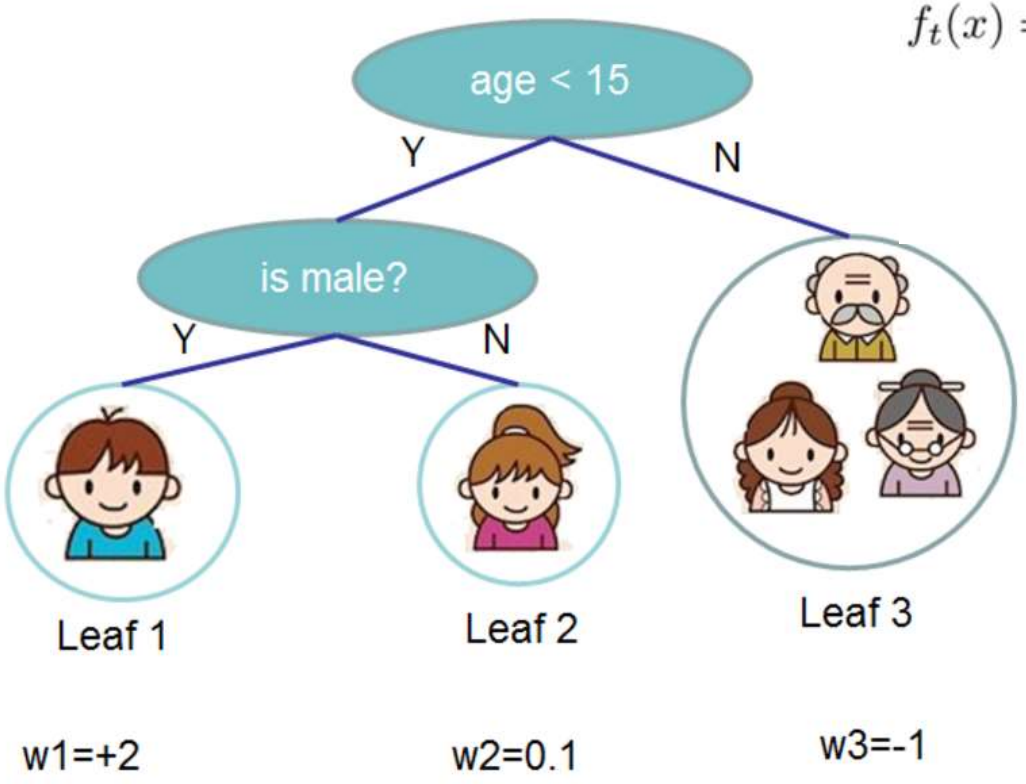
$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

叶子的个数　　w的L2模平方

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

树的结构

叶子的向量



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

集成学习-XGBoost

第j个叶子对应的样本集合用如下式子表示：

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

因为所有的样本都映射到了某个叶子上，所以目标函数可以从样本求和转化为叶子的求和：

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

目标函数转化为一元二次方程的求和。

定义Gj(每个叶子节点里面一阶梯度的和)Hj(每个叶子节点里面二阶梯度的和)：

$$G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$$

集成学习-XGBoost

目标函数改写：

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

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

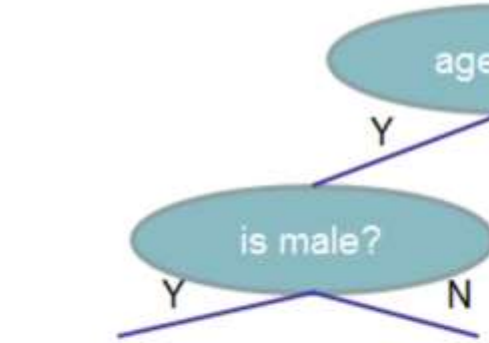
求偏导得出：

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

集成学习-XGBoost

Obj代表了当指定一个树的结构的时候，在目标上面最多减少多少，可叫做结构分数(structure score)，Obj计算示例：

样本号		梯度数据
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



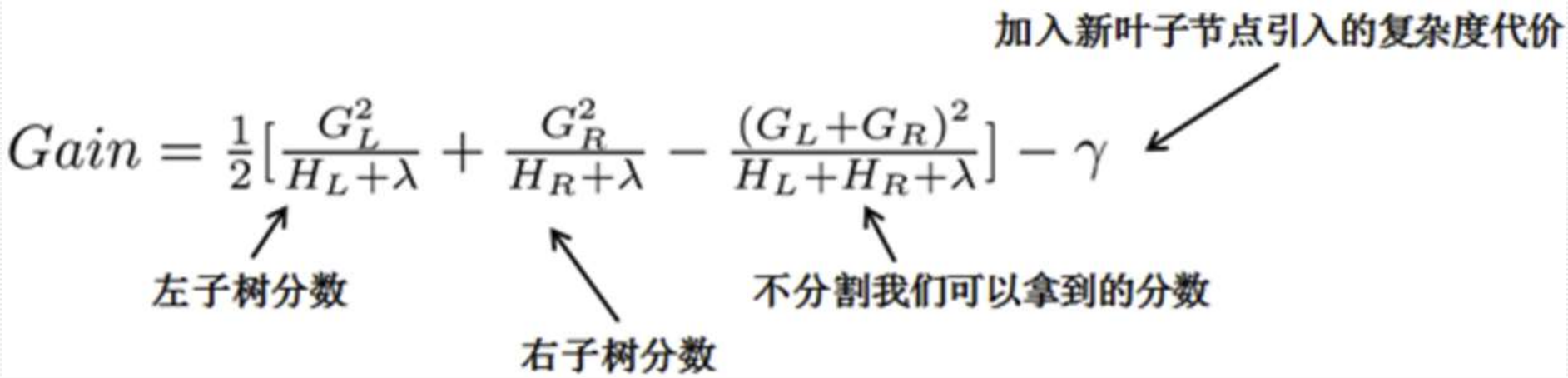
$$I_1 = \{1\} \quad I_2 = \{4\} \quad I_3 = \{2,3,5\}$$
$$G_1 = g_1 \quad G_2 = g_4 \quad G_3 = g_2 + g_3 + g_5$$
$$H_1 = h_1 \quad H_4 = h_4 \quad H_3 = h_2 + h_3 + h_5$$

$$Obj = -\frac{1}{2}\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

这个分数越小，代表这个树的结构越好

集成学习-XGBoost

- 贪心算法获取最优切分点：
- 利用这个打分函数来寻找出一个最优结构的树，加入到模型中，再重复这样的操作。
- 常用的方法是贪心法，每一次尝试去对已有的叶子加入一个分割。对于一个具体的分割方案，计算增益。



谢谢