问题

在上一个问题 “11_boosting思想” 中我们已经简单谈了下提升方法 boosting 的基本思路，这个问题让我们深入了解下 boosting 思想中最具代表性的算法 AdaBoost。GBDT和XGBoost《统计学习方法》与《机器学习》这两本书中都没有涉及，但是看别人在牛客网上的面经分享都有提到，其实这两个算法主要在竞赛中经常被用到，因此还是有必要了解一下。

AdaBoost算法

特点：

1. 不改变所给的训练数据，而不断改变训练数据权值的分布，使得训练数据在基本分类器的学习中起不同的作用
2. 利用基本分类器的线性组合构建最终的分类器

假设给定一个二类分类的训练数据集：

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \tag{1}$$

其中每个样本点由实例与标记组成。实例 $x_i \in \chi \subseteq R^n$ ，标记 $y_i \in \Upsilon = \{-1, 1\}$ ，χ 是实例空间，Υ 是标记集合。Adaboost 利用以下算法，从训练数据中学习一系列弱分类器，并将这些弱分类器线性组合成为一个强分类器。

(一) 初始化训练数据的权值分布：

$$D_1 = (w_{11}, \ldots, w_{1i}, \ldots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \ldots, N \tag{2}$$

> 假设训练数据集具有均匀的权值分布，即每个训练样本在基本分类器的学习中作用相同。这一假设保证第 1 步能够在原始数据上学习基本分类器 $G_1(x)$ 。

(二) 一共需要学习 M 个基本分类器，则在每一轮 $m = 1, 2, \ldots, M$ 顺次地执行下列操作：

1. **使用当前分布 D_m 加权的训练数据集，得到基本分类器：**

$$G_m(x) : \chi \to \{-1, +1\} \tag{3}$$

2. **计算 $G_m(x)$ 在训练数据集上的分类误差率：**

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{G_{mi} \neq y_i} w_{mi} \tag{4}$$

   w_{mi} 表示第 m 轮中第 i 个实例的权值，$\sum_{i=1}^{N} w_{mi} = 1$ 。这表明， $G_m(x)$ 在加权的训练数据集上的分类误差率是被 $G_m(x)$ 误分类的样本的权值之和。

3. **计算 $G_m(x)$ 的系数：**

$$\alpha_m = \frac{1}{2} ln \frac{1 - e_m}{e_m} \tag{5}$$

   α_m 表示 $G_m(x)$ 在最终分类器中的重要性。根据式子可知，$e_m \leq \frac{1}{2}$ 时，$\alpha_m \geq 0$ ，并且 α_m 随着 e_m 的减小而增大，所以分类误差率越小的基本分类器在最终分类器中的作用越大。**这里注意所有 α_m 加起来的和并不等于 1 。** （注意 e_m 是有可能比 $\frac{1}{2}$ 大的，也就是说 α_m 有可能小于 0，《统计学习方法》没有讨论这种情况的处理方法，但在西瓜书中的处理方法是抛弃该分类器，且终止学习过程，哪怕学习到的分类器个数远远没有达到预设的 M）

4. **更新训练数据集的权值分布：**

$$
\begin{aligned}
&D_{m+1} = (w_{m+1,1}, \ldots, w_{m+1,i}, \ldots, w_{m+1,N}) \\
&w_{m+1,i} = \frac{w_{mi}}{Z_m} exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \ldots, N \\
&\text{其中} Z_m \text{是规范因子，} Z_m = \sum_{i=1}^{N} w_{mi} exp(-\alpha_m y_i G_m(x_i))
\end{aligned} \tag{6}
$$

$w_{m+1,i}$ 也可以写成分段函数的形式：

$$
w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases} \tag{7}
$$

也就是说被基本分类器 $G_m(x)$ 误分类样本的权值得以增大，而被正确分类的样本的权值却变小。两者相比可知误分类样本的权值被放大 $e^{2\alpha_m} = \frac{1-e_m}{e_m}$ 倍。因此，误分类样本在下一轮学习中起更大的作用。

(三) 经过以上过程后可以得到 M 个基本分类器，构建基本分类器的线性组合：

$$
f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) \tag{8}
$$

得到最终分类器：

$$
G(x) = sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{9}
$$

线性组合 $f(x)$ 实现 M 个基本分类器的加权表决，$f(x)$ 的符号决定了实例 x 的类，$f(x)$ 的绝对值表示分类的确信度。

不得不说，整个算法的设计很巧妙！

引自《机器学习》

> 对无法接受带权样本的基分类器学习方法，则可通过“重采样法”来处理，即在每一轮学习中，根据样本分布对训练集重新采样，再用重采样而得的样本集对基分类器进行训练。上面也说到了，AdaBoost模型学习过程中，当出现一个基分类器的误分类误差大于 0.5 即比随机猜测还要差时，处理方法是将该分类器丢弃，且终止学习过程，此种情形下，初始设置的学习轮次 M 也许还未达到，可能会导致最终集成中只包含很少的基分类器而性能不佳。若采用“重采样法”，则可获得“重启动”机会以避免训练过程过早停止，即在抛弃不满足条件的当前基分类器之后，可根据当前分布重新对训练样本进行采样，再基于新的采样结果重新训练出基分类器，从而使得学习过程可以持续到预设的 M 轮完成。

Adaboost 的另一种解释

> 可以认为 AdaBoost 算法是模型为加法模型、损失函数为指数函数、学习算法为前向分布算法时的二类分类学习方法。

首先需要介绍一下**前向分布算法：**

假设加法模型：

$$
f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{10}
$$

其中，$b(x; \gamma_m)$ 为基函数，γ_m 为基函数的参数，β_m 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即损失函数极小化问题：

$$min_{\beta_m,\gamma_m}\sum_{i=1}^{N}L(y_i,\sum_{m=1}^{M}\beta_m b(x_i;\gamma_m)) \tag{11}$$

这是一个复杂的优化问题，前向分布算法求解这一问题的想法是：因为学习的是加法模型，如果能够从前往后，每一步只学习一个基函数及其系数，逐步逼近以上优化函数式，那么就可以简化优化的复杂度。具体地，每一步只需要优化如下损失函数：

$$min_{\beta,\gamma}\sum_{i=1}^{N}L(y_i,\beta b(x_i;\gamma)) \tag{12}$$

下面使用前向分步算法推导出 AdaBoost

此时的模型是由基本分类器组成的加法模型：

$$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x) \tag{13}$$

其损失函数为**指数损失函数**：$L(y,f(x))=exp(-yf(x))$ (书中还用了一大段来证明前向分步算法的损失函数是指数损失函数)

在第 m 轮迭代中可以得到 α_m，$G_m(x)$ 和 $f_m(x)$。

$$f_m(x)=f_{m-1}(x)+\alpha_m G_m(x) \tag{14}$$

目标是使前向分步算法得到的 α_m 和 $G_m(x)$ 使 $f_m(x)$ 在训练集上的指数损失最小，即：

$$(\alpha_m,G_m(x))=argmin_{\alpha,G}\sum_{i=1}^{N}exp[-y_i(f_{m-1}(x_i)+\alpha G(x_i))] \tag{15}$$

也可以表示为：

$$(\alpha_m,G_m(x))=argmin_{\alpha,G}\sum_{i=1}^{N}\hat{w}_{mi}exp[-y_i\alpha G(x_i)] \tag{16}$$

其中 $\hat{w}_{mi}=exp[-y_i f_{m-1}(x_i)]$。因为 $\hat{w}_{mi}$ 既不依赖于 α 也不依赖于 G，所以与最小化无关。但 $\hat{w}_{mi}$ 依赖于 $f_{m-1}(x)$，随着每一轮迭代而发生改变。

可以证明使上面式子达到最小的 α_m^* 和 $G_m^*(x)$ 就是 AdaBoost 算法所得到的 α_m 和 $G_m(x)$。

GBDT算法

> GBDT (Gradient Boosting Decision Tree) 梯度提升迭代决策树。GBDT 也是 Boosting 算法的一种，我们可以从名字上对这个算法有个初步的理解：GB是 Gradient Boosting，是一种学习策略，而 DT 就是决策树，因此 **GBDT 的含义就是用 Gradient Boosting 策略训练出来的 DT 模型。**

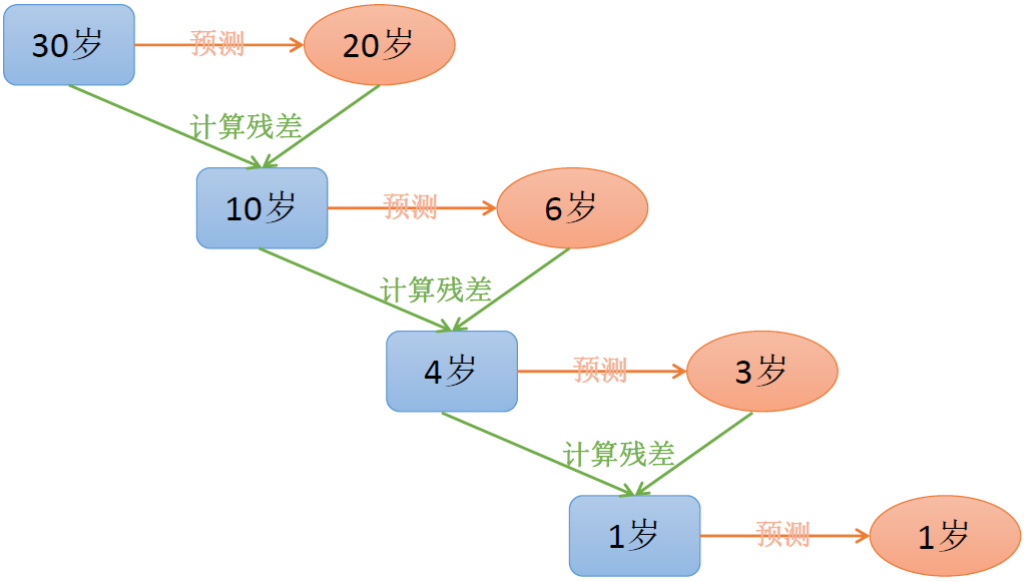
模型的结果是一组回归分类树组合**(基分类器使用的是CART Tree)**(关于这个树的介绍请移步 机器学习文件夹问题34)：$T_1\dots T_k$，其中 T_j 学习的是之前 $j-1$ 棵树预测结果的残差，这种思想就像准备考试前的复习，先做一遍习题册，然后把做错的题目挑出来，再做一次，然后把做错的题目挑出来在做一次，经过反复多轮训练，取得最好的成绩。

GBDT 和 AdaBoost 的模型都可以表示为如下形式：

$$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x) \tag{17}$$

只是 AdaBoost 在训练完一个 $G_m(x)$ 后会重新赋值样本的权重：分类错误的样本的权重会增大而分类正确的样本的权重则会减小。这样在训练时 $G_{m+1}(x)$ 会侧重对错误样本的训练，以达到模型性能的提升，但是AdaBoost模型每个基分类器的损失函数优化目标是相同的且独立的，都是最优化当前样本（样本权重）的指数损失。

GBTD虽然也是一个加性模型，但其是通过不断迭代拟合样本真实值与当前分类器预测的残差来逼近真实值的。可以形象地理解如下：（图来源于博客：GBDT 算法）



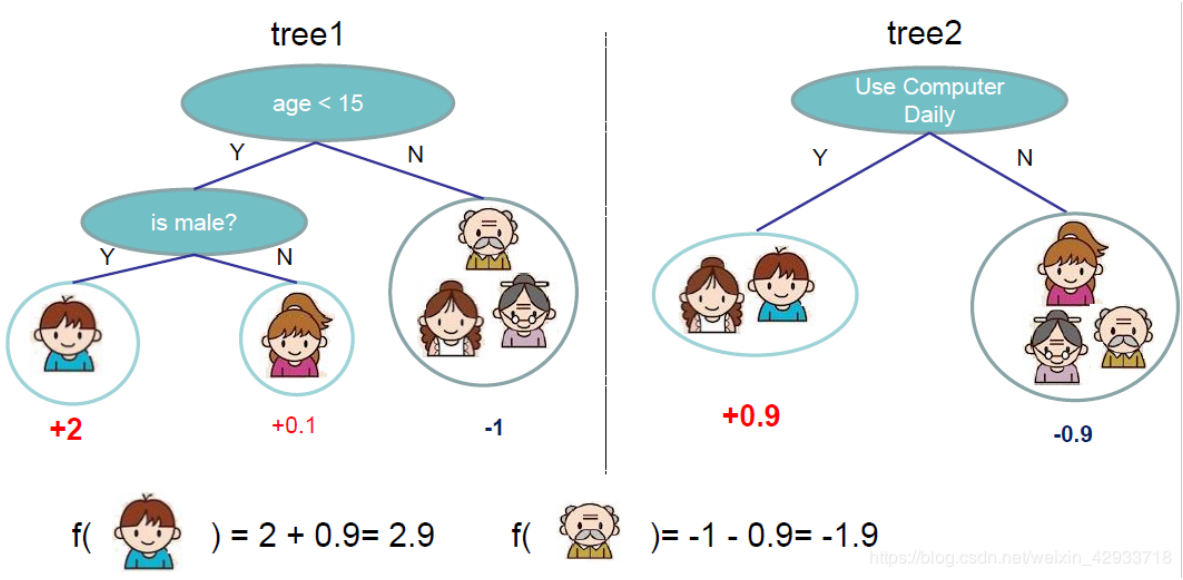
按照这个思路，第 m 个基分类器的预测结果为：

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x) \tag{18}$$

而 $G_m(x)$ 的优化目标就是最小化当前预测结果 $f_{m-1}(x_i) + \alpha G(x_i)$ 与真实值 y_i 之间的差距。

$$(\alpha_m, G_m(x)) = argmin_{\alpha,G} \sum_{i=1}^{N} L(y_i, \quad f_{m-1}(x_i) + \alpha G(x_i)) \tag{19}$$

下面是GDBT的一个简单例子：判断用户是否会喜欢电脑游戏，特征有年龄，性别和职业。需要注意的是，GBDT无论是用于分类和回归，采用的都是回归树，分类问题最终是将拟合值转换为概率来进行分类的。



在上图中,每个用户的最后的拟合值为两棵树的结果相加。

模型求解

Gradient Boosting是Friedman提出的一套框架。其思想类似于数值优化中梯度下降求参方法，参数沿着梯度的负方向以小步长前进，最终逐步逼近参数的局部最优解。在GB中模型每次拟合残差，逐步逼近最终结果。

如上所述，我们每个 stage 的优化目标是：

$$G_m(x) = argmin_G \sum_{i=1}^{N} L(y_i, \quad f_{m-1}(x_i) + \alpha_m G(x_i)) \tag{20}$$

该函数比较难求解，类似于梯度下降方法，给定 $f_{m-1}(x_i)$ 的一个近似解，$\alpha_m G_m(x)$ 可以看作是 $f_{m-1}(x_i)$ 逼近 $f_m(x_i)$ 的步长和方向。所以：

$$f_m(x) = f_{m-1}(x) - \alpha_m [\frac{\partial L(y_i, f_m(x_i))}{\partial f_m(x_i)}]_{f_m(x_i)=f_{m-1}(x_i)} \tag{21}$$
$$\alpha_m = argmin_\alpha \sum_{i=1}^{N} L(y_i, f_{m-1}(x) - \alpha_m [\frac{\partial L(y_i, f_m(x_i))}{\partial f_m(x_i)}]_{f_m(x_i)=f_{m-1}(x_i)})$$

上面的损失函数 L 可以根据问题的不同使用不同的损失函数，而且还可以加上模型复杂度的正则项等等来尽量避免过拟合。

XGBoost

XGBoost 是 GBDT 最广为人知的一个实现。通过使用一定程度的近似，使得求解变得更高效。同时支持分布式和 GPU 优化，有着广泛的使用。

如前文所述，最终分类器可以使用这个公式表示：

$$f(x) = \sum_{m=1}^{M} G_m(x) \tag{22}$$

优化目标如下：

$$Obj = \sum_{i=1}^{N} L(y_i, f(x_i)) + \sum_{m=1}^{M} \Omega(G_m(x)) \tag{23}$$

其中 N 为样本数量，y_i 表示样本真实值，$f(x)$ 是模型输出，所以前半部分代表模型的损失函数。M 表示树的个数，$G_m(x)$ 表示第 m 棵树，Ω 是模型复杂度函数。模型越复杂，越容易出现过拟合，所以采用这样的目标函数，为了使得最终的模型既有很好的准确度也有不错的泛化能力。

追加法训练

核心的思想是，已经训练好的树 $T1...T_{t-1}$ 不再调整。根据目标函数最小原则，新增树 T_t ，表示如下：

$$\begin{aligned}
&f_0(x) = 0 \quad \text{算法初始化} \\
&f_1(x) = G_1(x) = f_0(x) + G_1(x) \quad \text{训练第 1 棵树 } G_1(x) \\
&f_2(x) = G_1(x) + G_2(x) = f_1(x) + G_2(x) \quad \text{训练第 2 棵树 } G_2(x) \\
&f_t(x) = \sum_{m=1}^{t} f_m(x) = f_{t-1}(x) + G_t(x) \quad \text{训练第 } t\text{棵树，前面 } t-1\text{棵不再调整}
\end{aligned} \tag{24}$$

假设此时对第 t 棵树训练，则目标函数表示为：

$$Obj^{(t)} = \sum_{i=1}^{N} L(y_i, f_t(x_i)) + \sum_{m=1}^{t} \Omega(G_m(x)) \tag{25}$$
$$= \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i) + G_t(x_i) + \Omega(G_t(x)) + constant$$

其中 constant 常数代表 $\sum_{m=1}^{t-1} \Omega(G_m(x))$ ，因为前面 t-1 棵树结构不再变化，所以他们的复杂度为常数。

回顾高等数学中的泰勒展开，它使用一个函数的高阶导数，用多项式的形式逼近原始函数。当展开到二阶导数的时候公式如下：

利用泰勒展开公式和上面推导的 $Obj^{(t)}$ ， $Obj^{(t)}$ 式中，$f_{t-1}(x)$ 对应泰勒公式中的 x，而 $G_t(x)$ 是一棵待增加的新树，对应泰勒公式中的 Δx ，$L(y_i, f_{t-1}(x))$ 对应泰勒公式中的 $f(x)$ ，而 $L(y_i, f_{t-1}(x_i) + G_t(x_i)$ 对应泰勒公式中的 $f(x + \Delta x)$ ，则对 $L(y_i, f_{t-1}(x_i) + G_t(x_i)$ 做二阶泰勒展开，得：

$$Obj^{(t)} = \sum_{i=1}^{N} [L(y_i, f_{t-1}(x_i)) + g_i G_t(x_i) + \frac{1}{2} h_i G_t^2(x_i)] + \Omega(G_t(x)) + constant \tag{26}$$

其中：

$$g_i = \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)} \quad \text{一阶导数} \tag{27}$$
$$h_i = \frac{\partial L(y_i, f_{t-1}(x_i))}{\partial^2 f_{t-1}(x_i)} \quad \text{二阶导数}$$

因为我们求解的目标是使得 $Obj^{(t)}$ 最小的 $G_t(x)$ 。当前面 $t-1$ 棵树都已经确定时，$\sum_{i=1}^{N} [L(y_i, f_{t-1}(x_i))$ 是一个常量，可以省略，constant 常量也可以省略，因此简化得到新的目标函数：

$$Obj^{(t)} = \sum_{i=1}^{N} [g_i G_t(x_i) + \frac{1}{2} h_i G_t^2(x_i)] + \Omega(G_t(x)) \tag{28}$$

后面模型求解部分还很复杂，这里就不再展开了，感兴趣的可以阅读参考资料中的第一个，讲解地非常详细了。（其实后面我暂时看不下去了……）

从XGBoost原理部分可以看出，XGBoost的实现中，采用了二阶泰勒展开公式展开来近似损失函数，同时在求解树的过程中，使用了贪心算法，并使用枚举特征，样本按照特征排序后，采用线性扫描找到最优分裂点。这种实现方法，是对GDBT思想的逼近，但是在工程上确非常有效。

参考资料

GBDT的原理和应用
集成树之三：GBDT
《统计学习方法》