

台湾大学林轩田机器学习技法课程学习笔记11 -- Gradient Boosted Decision Tree

作者：红色石头

微信公众号：AI有道 (ID : redstonewill)

上节课我们主要介绍了Random Forest算法模型。Random Forest就是通过bagging的方式将许多不同的decision tree组合起来。除此之外，在decision tree中加入了各种随机性和多样性，比如不同特征的线性组合等。RF还可以使用OOB样本进行self-validation，而且可以通过permutation test进行feature selection。本节课将使用Adaptive Boosting的方法来研究decision tree的一些算法和模型。

Adaptive Boosted Decision Tree

Random Forest的算法流程我们上节课也详细介绍过，就是先通过bootstrapping“复制”原样本集 D ，得到新的样本集 D' ；然后对每个 D' 进行训练得到不同的decision tree和对应的 g_t ；最后再将所有的 g_t 通过uniform的形式组合起来，即以投票的方式得到 G 。这里采用的Bagging的方式，也就是把每个 g_t 的预测值直接相加。现在，如果将Bagging替换成AdaBoost，处理方式有些不同。首先每轮bootstrap得到的 D' 中每个样本会赋予不同的权重 $u^{(t)}$ ；然后在每个decision tree中，利用这些权重训练得到最好的 g_t ；最后得出每个 g_t 所占的权重，线性组合得到 G 。这种模型称为AdaBoost-D Tree。

```
function RandomForest( $\mathcal{D}$ )
For  $t = 1, 2, \dots, T$ 
  ① request size- $N'$  data  $\tilde{\mathcal{D}}_t$  by bootstrapping with  $\mathcal{D}$ 
  ② obtain tree  $g_t$  by Randomized-DTree( $\tilde{\mathcal{D}}_t$ )
return  $G = \text{Uniform}(\{g_t\})$ 
```

```
function AdaBoost-DTree( $\mathcal{D}$ )
For  $t = 1, 2, \dots, T$ 
  ① reweight data by  $u^{(t)}$ 
  ② obtain tree  $g_t$  by DTree( $\mathcal{D}, u^{(t)}$ )
  ③ calculate 'vote'  $\alpha_t$  of  $g_t$ 
return  $G = \text{LinearHypo}(\{(g_t, \alpha_t)\})$ 
```

但是在AdaBoost-DTree中需要注意的一点是每个样本的权重 $u^{(t)}$ 。我们知道，在Adaptive Boosting中进行了bootstrap操作， $u^{(t)}$ 表示 D 中每个样本在 D' 中出现的次数。但是在决策树模型中，例如C&RT算法中并没有引入 $u^{(t)}$ 。那么，如何在决策树中引入这些权重 $u^{(t)}$ 来得到不同的 g_t 而又不改变原来的决策树算法呢？

在Adaptive Boosting中，我们使用了weighted algorithm，形如：

$$E_{in}^u(h) = \frac{1}{N} \sum_{n=1}^N u_n \cdot err(y_n, h(x_n))$$

每个犯错误的样本点乘以相应的权重，求和再平均，最终得到了 $E_{in}^u(h)$ 。如果在决策树中使用这种方法，将当前分支下犯错误的点赋予权重，每层分支都这样做，会比较复杂，不易求解。为了简化运算，保持决策树算法本身的稳定性和封闭性，我们可以把决策树算法当成一个黑盒子，即不改变其结构，不对算法本身进行修改，而从数据来源 D' 上做一些处理。按照这种思想，我们来看权重 u 实际上表示该样本在bootstrap中出现的次数，反映了它出现的概率。那么可以根据 u 值，对原样本集 D 进行一次重新随机sampling，也就是带权重的随机抽样。sampling之后，会得到一个新的 D' ， D' 中每个样本出现的几率与它权重 u 所占的比例应该是差不多接近的。因此，使用带权重的sampling操作，得到了新的样本数据集 D' ，可以直接代入决策树进行训练，从而无需改变决策树算法结构。sampling可看成是bootstrap的反操作，这种对数据本身进行修改而不更改算法结构的方法非常重要！

'Weighted' Algorithm in Bagging	A General Randomized Base Algorithm
weights u expressed by bootstrap-sampled copies —request size- N' data \tilde{D}_t by bootstrapping with \mathcal{D}	weights u expressed by sampling proportional to u_n —request size- N' data \tilde{D}_t by sampling $\propto u$ on \mathcal{D}

所以，AdaBoost-DTree结合了AdaBoost和DTree，但是做了一点小小的改变，就是使用sampling替代权重 $u^{(t)}$ ，效果是相同的。

AdaBoost-DTree: often via
AdaBoost + **sampling** $\propto u^{(t)}$ + DTree(\tilde{D}_t)
without modifying DTree

上面我们通过使用sampling，将不同的样本集代入决策树中，得到不同的 g_t 。除此之外，我们还要确定每个 g_t 所占的权重 α_t 。之前我们在AdaBoost中已经介绍过，首先算出每个 g_t 的错误率 ϵ_t ，然后计算权重：

$$\alpha_t = \ln \diamond_t = \ln \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}}$$

如果现在有一棵完全长成的树（fully grown tree），由所有的样本 x_n 训练得到。若每个样本都不相同的话，一刀刀切割分支，直到所有的 x_n 都被完全分开。这时候， $E_{in}(g_t) = 0$ ，加权的 $E_{in}^u(g_t) = 0$ 而且 ϵ_t 也为0，从而得到权重 $\alpha_t = \infty$ 。 $\alpha_t = \infty$ 表示该 g_t 所占的权重无限大，相当于它一个就决定了G结构，是一种autocracy，而其

它的 g_t 对G没有影响。

if fully grown tree trained on all \mathbf{x}_n
 $\Rightarrow E_{\text{in}}(g_t) = 0$ if all \mathbf{x}_n different
 $\Rightarrow E_{\text{in}}^u(g_t) = 0$
 $\Rightarrow \epsilon_t = 0$
 $\Rightarrow \alpha_t = \infty$ (autocracy!!)

显然 $\alpha_t = \infty$ 不是我们想看到的，因为autocracy总是不好的，我们希望使用aggregation将不同的 g_t 结合起来，发挥集体智慧来得到最好的模型G。首先，我们来看一下什么原因造成了 $\alpha_t = \infty$ 。有两个原因：一个是使用了所有的样本 \mathbf{x}_n 进行训练；一个是树的分支过多，fully grown。针对这两个原因，我们可以对树做一些修剪（pruned），比如只使用一部分样本，这在sampling的操作中已经起到这类作用，因为必然有些样本没有被采样到。除此之外，我们还可以限制树的高度，让分支不要那么多，从而避免树fully grown。

need: pruned tree trained on some \mathbf{x}_n to be weak

- pruned: usual pruning, or just limiting tree height
- some: sampling $\propto \mathbf{u}^{(t)}$

因此，AdaBoost-DTree使用的是pruned DTree，也就是说将这些预测效果较弱的树结合起来，得到最好的G，避免出现autocracy。

AdaBoost-DTree: often via AdaBoost +
sampling $\propto \mathbf{u}^{(t)}$ + pruned DTree($\tilde{\mathcal{D}}$)

刚才我们说了可以限制树的高度，那索性将树的高度限制到最低，即只有1层高的时候，有什么特性呢？当树高为1的时候，整棵树只有两个分支，切割一次即可。如果impurity是binary classification error的话，那么此时的AdaBoost-DTree就跟AdaBoost-Stump没什么两样。也就是说AdaBoost-Stump是AdaBoost-DTree的一种特殊情况。

DTree (C&RT) with height ≤ 1

learn branching criteria

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{\operatorname{argmin}} \sum_{c=1}^2 |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

—if impurity = binary classification error,
just a decision stump, remember? :-)

值得一提的是，如果树高为1时，通常较难遇到 $\epsilon_t = 0$ 的情况，且一般不采用sampling的操作，而是直接将权重 u 代入到算法中。这是因为此时的AdaBoost-DTree就相当于AdaBoost-Stump，而AdaBoost-Stump就是直接使用 u 来优化模型的。

Optimization View of AdaBoost

接下来，我们继续探讨AdaBoost算法的一些奥妙之处。我们知道AdaBoost中的权重的迭代计算如下所示：

$$\begin{aligned} u_n^{(t+1)} &= \begin{cases} u_n^{(t)} \cdot \diamond_t & \text{if incorrect} \\ u_n^{(t)} / \diamond_t & \text{if correct} \end{cases} \\ &= u_n^{(t)} \cdot \diamond_t^{-y_n g_t(x_n)} = u_n^{(t)} \cdot \exp(-y_n \alpha_t g_t(x_n)) \end{aligned}$$

之前对于incorrect样本和correct样本， $u_n^{(t+1)}$ 的表达式不同。现在，把两种情况结合起来，将 $u_n^{(t+1)}$ 写成一种简化的形式：

$$u_n^{(t+1)} = u_n^{(t)} \cdot \diamond_t^{-y_n g_t(x_n)} = u_n^{(t)} \cdot \exp(-y_n \alpha_t g_t(x_n))$$

其中，对于incorrect样本， $y_n g_t(x_n) < 0$ ，对于correct样本， $y_n g_t(x_n) > 0$ 。从上式可以看出， $u_n^{(t+1)}$ 由 $u_n^{(t)}$ 与某个常数相乘得到。所以，最后一轮更新的 $u_n^{(T+1)}$ 可以写成 $u_n^{(1)}$ 的级联形式，我们之前令 $u_n^{(1)} = \frac{1}{N}$ ，则有如下推导：

$$u_n^{(T+1)} = u_n^{(1)} \cdot \prod_{t=1}^T \exp(-y_n \alpha_t g_t(x_n)) = \frac{1}{N} \cdot \exp(-y_n \sum_{t=1}^T \alpha_t g_t(x_n))$$

上式中 $\sum_{t=1}^T \alpha_t g_t(x_n)$ 被称为voting score，最终的模型

$G = \text{sign}(\sum_{t=1}^T \alpha_t g_t(x_n))$ 。可以看出，在AdaBoost中， $u_n^{(T+1)}$ 与 $\exp(-y_n (\text{voting score on } x_n))$ 成正比。

$$u_n^{(T+1)} = u_n^{(1)} \cdot \prod_{t=1}^T \exp(-y_n \alpha_t g_t(\mathbf{x}_n)) = \frac{1}{N} \cdot \exp\left(-y_n \sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n)\right)$$

- recall: $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t g_t(\mathbf{x})\right)$
- $\sum_{t=1}^T \alpha_t g_t(\mathbf{x})$: **voting score** of $\{g_t\}$ on \mathbf{x}

$$\text{AdaBoost: } u_n^{(T+1)} \propto \exp(-y_n (\text{voting score on } \mathbf{x}_n))$$

接下来我们继续看一下voting score中蕴含了哪些内容。如下图所示，voting score由许多 $g_t(\mathbf{x}_n)$ 乘以各自的系数 α_t 线性组合而成。从另外一个角度来看，我们可以把 $g_t(\mathbf{x}_n)$ 看成是对 \mathbf{x}_n 的特征转换 $\phi_i(\mathbf{x}_n)$ ， α_t 就是线性模型中的权重 w_i 。看到这里，我们回忆起之前SVM中， w 与 $\phi(\mathbf{x}_n)$ 的乘积再除以 w 的长度就是margin，即点到边界的距离。另外，乘积项再与 y_n 相乘，表示点的位置是在正确的那一侧还是错误的那一侧。所以，回过头来，这里的voting score实际上可以看成是没有正规化（没有除以 w 的长度）的距离，即可以看成是该点到分类边界距离的一种衡量。从效果上说，距离越大越好，也就是说voting score要尽可能大一些。

linear blending = LinModel + hypotheses as transform + ~~constraints~~

$$G(\mathbf{x}_n) = \text{sign}\left(\sum_{t=1}^T \underbrace{\alpha_t}_{w_i} \underbrace{g_t(\mathbf{x}_n)}_{\phi_i(\mathbf{x}_n)}\right)$$

voting score

$$\text{and hard-margin SVM margin} = \frac{y_n \cdot (\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}, \text{ remember? :-)}$$

我们再来看，若voting score与 y_n 相乘，则表示一个有对错之分的距离。也就是说，如果二者相乘是负数，则表示该点在错误的一边，分类错误；如果二者相乘是正数，则表示该点在正确的一边，分类正确。所以，我们算法的目的就是让 y_n 与voting score的乘积是正的，而且越大越好。那么在刚刚推导的 $u_n^{(T+1)}$ 中，得到 $\exp(-y_n (\text{voting score}))$ 越小越好，从而得到 $u_n^{(T+1)}$ 越小越好。也就是说，如果voting score表现不错，与 y_n 的乘积越大的话，那么相应的 $u_n^{(T+1)}$ 应该是最小的。

$$y_n(\text{voting score}) = \text{signed \& unnormalized margin}$$

want $y_n(\text{voting score})$ **positive & large**

$$\Leftrightarrow \exp(-y_n(\text{voting score})) \text{ small}$$

$$\Leftrightarrow u_n^{(T+1)} \text{ small}$$

那么在AdaBoost中，随着每轮学习的进行，每个样本的 $u_n^{(t)}$ 是逐渐减小的，直到 $u_n^{(T+1)}$ 最小。以上是从单个样本点来看的。总体来看，所有样本的 $u_n^{(T+1)}$ 之和应该也是最小的。我们的目标就是在最后一轮（T+1）学习后，让所有样本的 $u_n^{(T+1)}$ 之和尽可能地小。 $u_n^{(T+1)}$ 之和表示为如下形式：

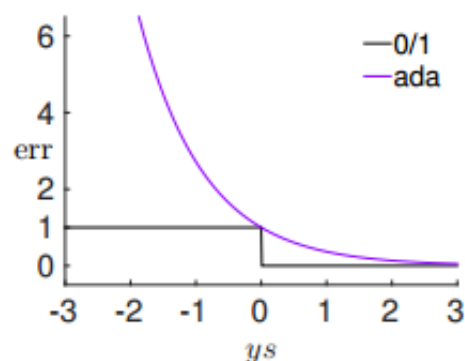
claim: AdaBoost **decreases** $\sum_{n=1}^N u_n^{(t)}$ and thus somewhat **minimizes**

$$\sum_{n=1}^N u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^N \exp \left(-y_n \sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n) \right)$$

上式中， $\sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n)$ 被称为linear score，用s表示。对于0/1 error：若 $ys < 0$ ，则 $err_{0/1} = 1$ ；若 $ys \geq 0$ ，则 $err_{0/1} = 0$ 。如下图右边黑色折线所示。对于上式中提到的指数error，即 $\hat{err}_{ADA}(s, y) = \exp(-ys)$ ，随着ys的增加，error单调下降，且始终落在0/1 error折线的上面。如下图右边蓝色曲线所示。很明显， $\hat{err}_{ADA}(s, y)$ 可以看成是0/1 error的上界。所以，我们可以使用 $\hat{err}_{ADA}(s, y)$ 来替代0/1 error，能达到同样的效果。从这点来说， $\sum_{n=1}^N u_n^{(T+1)}$ 可以看成是一种error measure，而我们的目标就是让其最小化，求出最小值时对应的各个 α_t 和 $g_t(\mathbf{x}_n)$ 。

$$\text{linear score } s = \sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n)$$

- $err_{0/1}(s, y) = \mathbb{I}[ys \leq 0]$
- $\hat{err}_{ADA}(s, y) = \exp(-ys)$:
upper bound of $err_{0/1}$
—called **exponential error measure**



\hat{err}_{ADA} : **algorithmic error measure**
by **convex upper bound** of $err_{0/1}$

下面我们来研究如何让 $\sum_{n=1}^N u_n^{(T+1)}$ 取得最小值，思考是否能用梯度下降（gradient descent）的方法来进行求解。我们之前介绍过gradient descent的核心是在某点处做一阶泰勒展开：

recall: gradient descent (remember? :-)), at iteration t

$$\min_{\|\mathbf{v}\|=1} E_{in}(\mathbf{w}_t + \eta \mathbf{v}) \approx \underbrace{E_{in}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \underbrace{\mathbf{v}^T \nabla E_{in}(\mathbf{w}_t)}_{\text{known}}$$

其中， \mathbf{w}_t 是泰勒展开的位置， \mathbf{v} 是所要求的下降的最好方向，它是梯度 $\nabla E_{in}(\mathbf{w}_t)$ 的反方向，而 η 是每次前进的步长。则每次沿着当前梯度的反方向走一小步，就会不断逼近谷底（最小值）。这就是梯度下降算法所做的事情。

现在，我们对 \tilde{E}_{ADA} 做梯度下降算法处理，区别是这里的方向是一个函数 g_t ，而不是一个向量 \mathbf{w}_t 。其实，函数和向量的唯一区别就是一个下标是连续的，另一个下标是离散的，二者在梯度下降算法应用上并没有大的区别。因此，按照梯度下降算法的展开式，做出如下推导：

at iteration t , to find g_t , solve

$$\begin{aligned} \min_h \hat{E}_{ADA} &= \frac{1}{N} \sum_{n=1}^N \exp \left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_{\tau} g_{\tau}(\mathbf{x}_n) + \eta h(\mathbf{x}_n) \right) \right) \\ &= \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n)) \\ &\stackrel{\text{taylor}}{\approx} \sum_{n=1}^N u_n^{(t)} (1 - y_n \eta h(\mathbf{x}_n)) = \sum_{n=1}^N u_n^{(t)} - \eta \sum_{n=1}^N u_n^{(t)} y_n h(\mathbf{x}_n) \end{aligned}$$

上式中， $h(\mathbf{x}_n)$ 表示当前的方向，它是一个标量， η 是沿着当前方向前进的步长。我们要求出这样的 $h(\mathbf{x}_n)$ 和 η ，使得 \tilde{E}_{ADA} 是在不断减小的。当 \tilde{E}_{ADA} 取得最小值的时候，那么所有的方向即最佳的 $h(\mathbf{x}_n)$ 和 η 就都解出来了。上述推导使用了在 $-y_n \eta h(\mathbf{x}_n) = 0$ 处的一阶泰勒展开近似。这样经过推导之后， \tilde{E}_{ADA} 被分解为两个部分，一个是前 N 个 u 之和 $\sum_{n=1}^N u_n^{(t)}$ ，也就是当前所有的 E_{in} 之和；另外一个包含下一步前进的方向 $h(\mathbf{x}_n)$ 和步进长度 η 的项 $-\eta \sum_{n=1}^N u_n^{(t)} y_n h(\mathbf{x}_n)$ 。 \tilde{E}_{ADA} 的这种形式与gradient descent的形式基本是一致的。

那么接下来，如果要最小化 \tilde{E}_{ADA} 的话，就要让第二项 $-\eta \sum_{n=1}^N u_n^{(t)} y_n h(\mathbf{x}_n)$ 越小越好。则我们的目标就是找到一个好的 $h(\mathbf{x}_n)$ （即好的方向）来最小化 $\sum_{n=1}^N u_n^{(t)} (-y_n h(\mathbf{x}_n))$ ，此时先忽略步进长度 η 。

$$\text{finding good } h \text{ (function direction)} \Leftrightarrow \text{minimize } \sum_{n=1}^N u_n^{(t)} (-y_n h(\mathbf{x}_n))$$

对于binary classification， y_n 和 $h(\mathbf{x}_n)$ 均限定取值-1或+1两种。我们对 $\sum_{n=1}^N u_n^{(t)} (-y_n h(\mathbf{x}_n))$ 做一些推导和平移运算：

for binary classification, where y_n and $h(\mathbf{x}_n)$ both $\in \{-1, +1\}$:

$$\begin{aligned}\sum_{n=1}^N u_n^{(t)} (-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^N u_n^{(t)} \begin{cases} -1 & \text{if } y_n = h(\mathbf{x}_n) \\ +1 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^N u_n^{(t)} + \sum_{n=1}^N u_n^{(t)} \begin{cases} 0 & \text{if } y_n = h(\mathbf{x}_n) \\ 2 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^N u_n^{(t)} + 2E_{in}^{u^{(t)}}(h) \cdot N\end{aligned}$$

—who minimizes $E_{in}^{u^{(t)}}(h)$? **A in AdaBoost! :-)**

最终 $\sum_{n=1}^N u_n^{(t)} (-y_n h(x_n))$ 化简为两项组成，一项是 $-\sum_{n=1}^N u_n^{(t)}$ ；另一项是 $2E_{in}^{u^{(t)}}(h) \cdot N$ 。则最小化 $\sum_{n=1}^N u_n^{(t)} (-y_n h(x_n))$ 就转化为最小化 $E_{in}^{u^{(t)}}(h)$ 。要让 $E_{in}^{u^{(t)}}(h)$ 最小化，正是由AdaBoost中的base algorithm所做的事情。所以说，AdaBoost中的base algorithm正好帮我们找到了梯度下降中下一步最好的函数方向。

A: good $g_t = h$ for 'gradient descent'

以上就是从数学上，从gradient descent角度验证了AdaBoost中使用base algorithm得到的 g_t 就是让 \check{E}_{ADA} 减小的方向，只不过这个方向是一个函数而不是向量。

在解决了方向问题后，我们需要考虑步进长度 η 如何选取。方法是在确定方向 g_t 后，选取合适的 η ，使 \check{E}_{ADA} 取得最小值。也就是说，把 \check{E}_{ADA} 看成是步进长度 η 的函数，目标是找到 \check{E}_{ADA} 最小化时对应的 η 值。

AdaBoost finds g_t by approximately $\min_h \hat{E}_{ADA} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n))$
after finding g_t , how about $\min_{\eta} \hat{E}_{ADA} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n))$

目的是找到在最佳方向上的最大步进长度，也就是steepest decent。我们先把 \check{E}_{ADA} 表达式写下来：

$$\check{E}_{ADA} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta g_t(x_n))$$

上式中，有两种情况需要考虑：

- $y_n = g_t(x_n) : u_n^{(t)} \exp(-\eta)$ correct

- $y_n \neq g_t(x_n) : u_n^{(t)} \exp(+\eta)$ incorrect

经过推导，可得：

$$\tilde{E}_{ADA} = \left(\sum_{n=1}^N u_n^{(t)} \right) \cdot ((1 - \epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta))$$

- optimal η_t somewhat 'greedily faster' than fixed (small) η —called **steepest** descent for optimization
- two cases inside summation:
 - $y_n = g_t(x_n) : u_n^{(t)} \exp(-\eta)$ (correct)
 - $y_n \neq g_t(x_n) : u_n^{(t)} \exp(+\eta)$ (incorrect)
- $\hat{E}_{ADA} = \left(\sum_{n=1}^N u_n^{(t)} \right) \cdot \left((1 - \epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta) \right)$

然后对 η 求导，令 $\frac{\partial \tilde{E}_{ADA}}{\partial \eta} = 0$ ，得：

$$\eta_t = \ln \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} = \alpha_t$$

由此看出，最大的步进长度就是 α_t ，即AdaBoost中计算 g_t 所占的权重。所以，AdaBoost算法所做的其实是在gradient descent上找到下降最快的方向和最大的步进长度。这里的方向就是 g_t ，它是一个函数，而步进长度就是 α_t 。也就是说，在AdaBoost中确定 g_t 和 α_t 的过程就相当于在gradient descent上寻找最快的下降方向和最大的步进长度。

Gradient Boosting

前面我们从gradient descent的角度来重新介绍了AdaBoost的最优化求解方法。整个过程可以概括为：

AdaBoost

$$\min_{\eta} \min_h \frac{1}{N} \sum_{n=1}^N \exp \left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_{\tau} g_{\tau}(x_n) + \eta h(x_n) \right) \right)$$

with binary-output hypothesis h

以上是针对binary classification问题。如果往更一般的情况进行推广，对于不同的error function，比如logistic error function或者regression中的squared error function，那么这种做法是否仍然有效呢？这种情况下的GradientBoost可以写成如下

形式：

GradientBoost

$$\min_{\eta} \min_h \frac{1}{N} \sum_{n=1}^N \text{err} \left(\underbrace{\sum_{\tau=1}^{t-1} \alpha_{\tau} g_{\tau}(\mathbf{x}_n)}_{s_n} + \eta h(\mathbf{x}_n), y_n \right)$$

with any hypothesis h (usually real-output hypothesis)

仍然按照gradient descent的思想，上式中， $h(\mathbf{x}_n)$ 是下一步前进的方向， η 是步进长度。此时的error function不是前面所讲的exp了，而是任意的一种error function。因此，对应的hypothesis也不再是binary classification，最常用的是实数输出的hypothesis，例如regression。最终的目标也是求解最佳的前进方向 $h(\mathbf{x}_n)$ 和最快的步进长度 η 。

GradientBoost: allows **extension to different err** for regression/soft classification/etc.

接下来，我们就来看看如何求解regression的GradientBoost问题。它的表达式如下所示：

$$\min_{\eta} \min_h \frac{1}{N} \sum_{n=1}^N \text{err} \left(\underbrace{\sum_{\tau=1}^{t-1} \alpha_{\tau} g_{\tau}(\mathbf{x}_n)}_{s_n} + \eta h(\mathbf{x}_n), y_n \right) \quad \text{with } \text{err}(s, y) = (s - y)^2$$

利用梯度下降的思想，我们把上式进行一阶泰勒展开，写成梯度的形式：

$$\begin{aligned} \min_h \dots &\stackrel{\text{taylor}}{\approx} \min_h \frac{1}{N} \sum_{n=1}^N \underbrace{\text{err}(s_n, y_n)}_{\text{constant}} + \frac{1}{N} \sum_{n=1}^N \eta h(\mathbf{x}_n) \left. \frac{\partial \text{err}(s, y_n)}{\partial s} \right|_{s=s_n} \\ &= \min_h \text{constants} + \frac{\eta}{N} \sum_{n=1}^N h(\mathbf{x}_n) \cdot 2(s_n - y_n) \end{aligned}$$

上式中，由于regression的error function是squared的，所以，对s的导数就是 $2(s_n - y_n)$ 。其中标注灰色的部分表示常数，对最小化求解并没有影响，所以可以忽略。很明显，要使上式最小化，只要令 $h(\mathbf{x}_n)$ 是梯度 $2(s_n - y_n)$ 的反方向就行了，即 $h(\mathbf{x}_n) = -2(s_n - y_n)$ 。但是直接这样赋值，并没有对 $h(\mathbf{x}_n)$ 的大小进行限制，一般不直接利用这个关系求出 $h(\mathbf{x}_n)$ 。

$$\min_h \text{ constants} + \frac{\eta}{N} \sum_{n=1}^N 2h(\mathbf{x}_n)(s_n - y_n)$$

实际上 $h(\mathbf{x}_n)$ 的大小并不重要，因为有步进长度 η 。那么，我们上面的最小化问题中需要对 $h(\mathbf{x}_n)$ 的大小做些限制。限制 $h(\mathbf{x}_n)$ 的一种简单做法是把 $h(\mathbf{x}_n)$ 的大小当成一个惩罚项（ $h^2(\mathbf{x}_n)$ ）添加到上面的最小化问题中，这种做法与regularization类似。如下图所示，经过推导和整理，忽略常数项，我们得到最关心的式子是：

$$\min \sum_{n=1}^N ((h(\mathbf{x}_n) - (y_n - s_n))^2)$$

上式是一个完全平方项之和， $y_n - s_n$ 表示当前第 n 个样本真实值和预测值的差，称之为余数。余数表示当前预测能够做到的效果与真实值的差值是多少。那么，如果我们想要让上式最小化，求出对应的 $h(\mathbf{x}_n)$ 的话，只要让 $h(\mathbf{x}_n)$ 尽可能地接近余数 $y_n - s_n$ 即可。在平方误差上尽可能接近其实很简单，就是使用regression的方法，对所有 N 个点 $(\mathbf{x}_n, y_n - s_n)$ 做squared-errors的regression，得到的回归方程就是我们要求的 $g_t(\mathbf{x}_n)$ 。

- magnitude of h does not matter: because η will be optimized next
- penalize large magnitude to avoid naïve solution

$$\begin{aligned} \min_h \text{ constants} + \frac{\eta}{N} \sum_{n=1}^N (2h(\mathbf{x}_n)(s_n - y_n) + (h(\mathbf{x}_n))^2) \\ = \text{ constants} + \frac{\eta}{N} \sum_{n=1}^N (\text{constant} + (h(\mathbf{x}_n) - (y_n - s_n))^2) \end{aligned}$$

- solution of penalized approximate functional gradient: squared-error regression on $\{(\mathbf{x}_n, \underbrace{y_n - s_n}_{\text{residual}})\}$

以上就是使用GradientBoost的思想来解决regression问题的方法，其中应用了一个非常重要的概念，就是余数 $y_n - s_n$ 。根据这些余数做regression，得到好的矩 $g_t(\mathbf{x}_n)$ ，方向函数 $g_t(\mathbf{x}_n)$ 也就是由余数决定的。

GradientBoost for regression:

find $g_t = h$ by regression with residuals

在求出最好的方向函数 $g_t(\mathbf{x}_n)$ 之后，就要来求相应的步进长度 η 。表达式如下：

after finding $g_t = h$,

$$\min_{\eta} \min_h \frac{1}{N} \sum_{n=1}^N \text{err} \left(\underbrace{\sum_{\tau=1}^{t-1} \alpha_{\tau} g_{\tau}(\mathbf{x}_n)}_{s_n} + \eta g_t(\mathbf{x}_n), y_n \right) \text{ with } \text{err}(s, y) = (s - y)^2$$

同样，对上式进行推导和化简，得到如下表达式：

$$\min_{\eta} \frac{1}{N} \sum_{n=1}^N (s_n + \eta g_t(\mathbf{x}_n) - y_n)^2 = \frac{1}{N} \sum_{n=1}^N ((y_n - s_n) - \eta g_t(\mathbf{x}_n))^2$$

—one-variable linear regression on $\{(g_t\text{-transformed input, residual})\}$

上式中也包含了余数 $y_n - s_n$ ，其中 $g_t(\mathbf{x}_n)$ 可以看成是 \mathbf{x}_n 的特征转换，是已知量。那么，如果我们想要让上式最小化，求出对应的 η 的话，只要让 $\eta g_t(\mathbf{x}_n)$ 尽可能地接近余数 $y_n - s_n$ 即可。显然，这也是一个 regression 问题，而且是一个很简单的形如 $y = ax$ 的线性回归，只有一个未知数 η 。只要对所有 N 个点 $(\eta g_t(\mathbf{x}_n), y_n - s_n)$ 做 squared-error 的 linear regression，利用梯度下降算法就能得到最佳的 η 。

将上述这些概念合并到一起，我们就得到了一个最终的演算法 Gradient Boosted Decision Tree (GBDT)。可能有人会问，我们刚才一直没有说到 Decision Tree，只是讲到了 Gradient Boost 啊？下面我们来看看 Decision Tree 究竟是在哪出现并使用的。其实刚刚我们在计算方向函数 g_t 的时候，是对所有 N 个点 $(\mathbf{x}_n, y_n - s_n)$ 做 squared-error 的 regression。那么这个回归算法就可以是决策树 C&RT 模型（决策树也可以用来做 regression）。这样，就引入了 Decision Tree，并将 Gradient Boost 和 Decision Tree 结合起来，构成了真正的 GBDT 算法。GBDT 算法的基本流程图如下所示：

Gradient Boosted Decision Tree (GBDT)

$$s_1 = s_2 = \dots = s_N = 0$$

for $t = 1, 2, \dots, T$

- ① obtain g_t by $\mathcal{A}(\{(\mathbf{x}_n, y_n - s_n)\})$ where \mathcal{A} is a (squared-error) regression algorithm

—how about sampled and pruned C&RT?

- ② compute $\alpha_t = \text{OneVarLinearRegression}(\{(g_t(\mathbf{x}_n), y_n - s_n)\})$

- ③ update $s_n \leftarrow s_n + \alpha_t g_t(\mathbf{x}_n)$

$$\text{return } G(\mathbf{x}) = \sum_{t=1}^T \alpha_t g_t(\mathbf{x})$$

值得注意的是， s_n 的初始值一般均设为 0，即 $s_1 = s_2 = \dots = s_N = 0$ 。每轮迭代

中，方向函数 g_t 通过C&RT算法做regression，进行求解；步进长度 η 通过简单的单参数线性回归进行求解；然后每轮更新 s_n 的值，即 $s_n \leftarrow s_n + \alpha_t g_t(x_n)$ 。T轮迭代结束后，最终得到 $G(x) = \sum_{t=1}^T \alpha_t g_t(x)$ 。

值得一提的是，本节课第一部分介绍的AdaBoost-DTree是解决binary classification问题，而此处介绍的GBDT是解决regression问题。二者具有一定的相似性，可以说GBDT就是AdaBoost-DTree的regression版本。

GBDT: 'regression sibling' of AdaBoost-DTree
—popular in practice

Summary of Aggregation Models

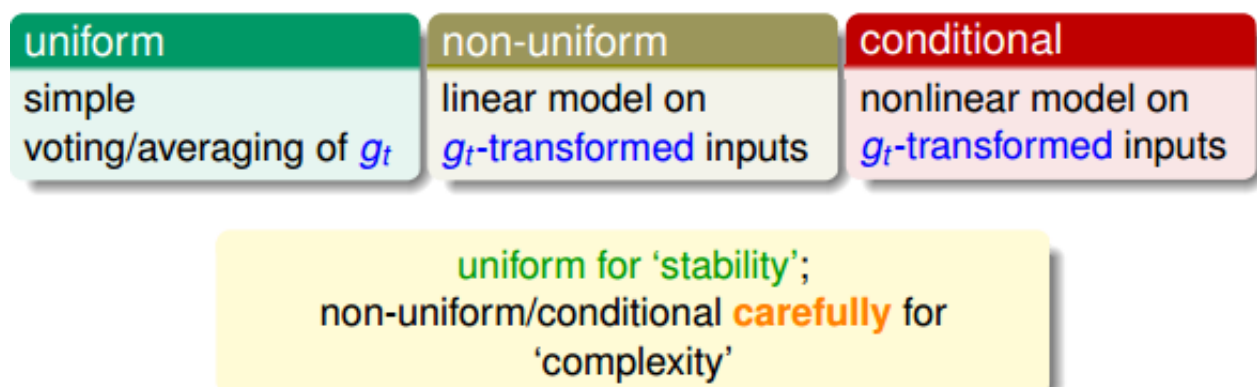
从机器学习技法课程的第7节课笔记到现在的第11节课笔记，我们已经介绍完所有的aggregation模型了。接下来，我们将对这些内容进行一个简单的总结和概括。

首先，我们介绍了blending。blending就是将所有已知的 g_t aggregate结合起来，发挥集体的智慧得到G。值得注意的一点是这里的 g_t 都是已知的。blending通常有三种形式：

- **uniform**：简单地计算所有 g_t 的平均值
- **non-uniform**：所有 g_t 的线性组合
- **conditional**：所有 g_t 的非线性组合

其中，uniform采用投票、求平均的形式更注重稳定性；而non-uniform和conditional追求的更复杂准确的模型，但存在过拟合的危险。

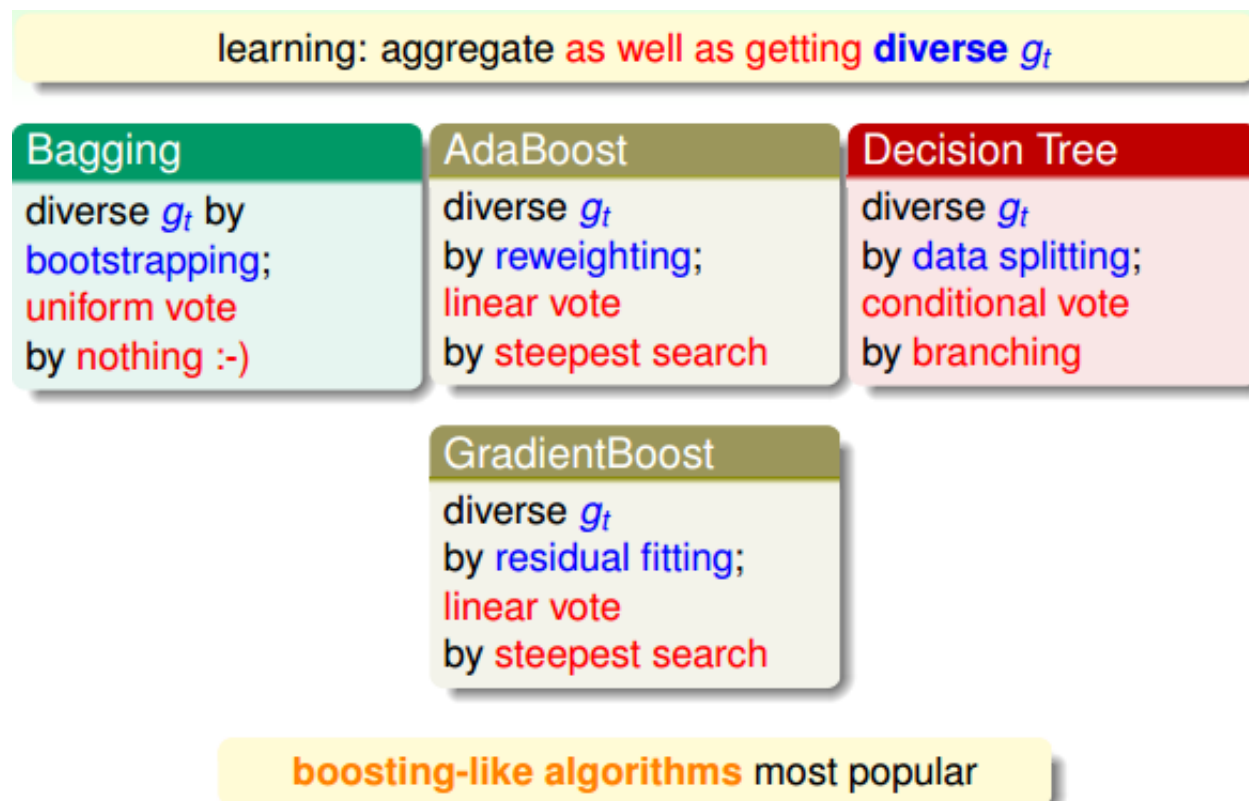
blending: aggregate after getting diverse g_t



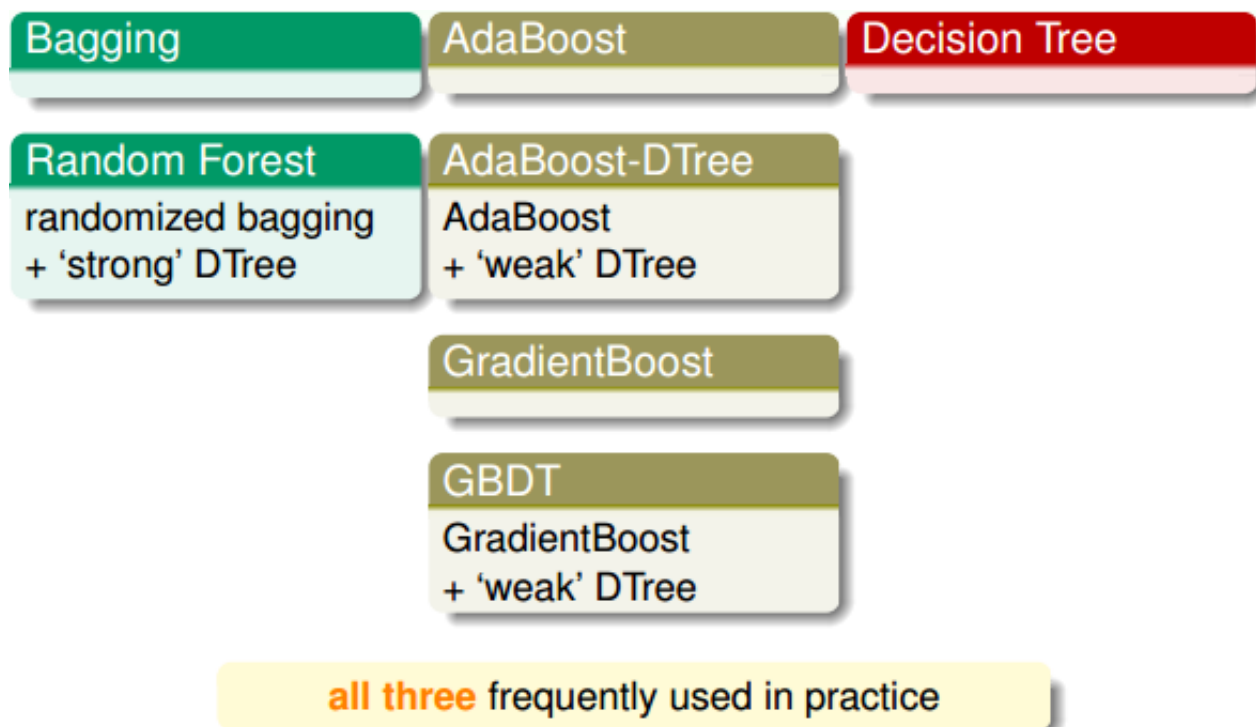
刚才讲的blending是建立在所有 g_t 已知的情况。那如果所有 g_t 未知的情况，对应的就是learning模型，做法就是一边学 g_t ，一边将它们结合起来。learning通常也有三种形式（与blending的三种形式——对应）：

- Bagging : 通过bootstrap方法 , 得到不同 g_t , 计算所有 g_t 的平均值
- AdaBoost : 通过bootstrap方法 , 得到不同 g_t , 所有 g_t 的线性组合
- Decision Tree : 通过数据分割的形式得到不同的 g_t , 所有 g_t 的非线性组合

然后 , 本节课我们将AdaBoost延伸到另一个模型GradientBoost。对于regression问题 , GradientBoost通过residual fitting的方式得到最佳的方向函数 g_t 和步进长度 η 。



除了这些基本的aggregation模型之外 , 我们还可以把某些模型结合起来得到新的aggregation模型。例如 , Bagging与Decision Tree结合起来组成了Random Forest。Random Forest中的Decision Tree是比较“茂盛”的树 , 即每个树的 g_t 都比较强一些。AdaBoost与Decision Tree结合组成了AdaBoost-DTree。AdaBoost-DTree的Decision Tree是比较“矮弱”的树 , 即每个树的 g_t 都比较弱一些 , 由AdaBoost将所有弱弱的树结合起来 , 让综合能力更强。同样 , GradientBoost与Decision Tree结合就构成了经典的算法GBDT。

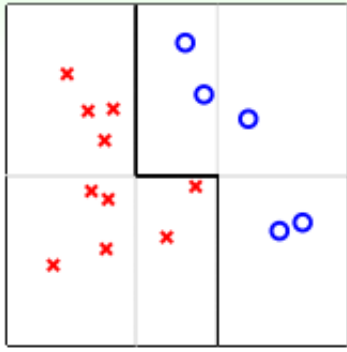


Aggregation的核心是将所有的 g_t 结合起来，融合到一起，即集体智慧的思想。这种做法之所以能得到很好的模型G，是因为aggregation具有两个方面的优点：cure underfitting和cure overfitting。

第一，aggregation models有助于防止欠拟合（underfitting）。它把所有比较弱的 g_t 结合起来，利用集体智慧来获得比较好的模型G。aggregation就相当于feature transform，来获得复杂的学习模型。

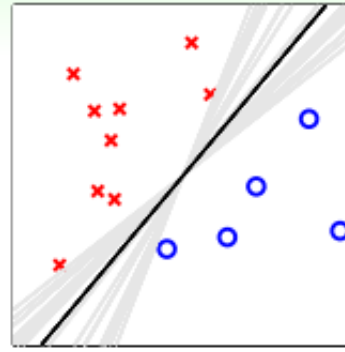
第二，aggregation models有助于防止过拟合（overfitting）。它把所有 g_t 进行组合，容易得到一个比较中庸的模型，类似于SVM的large margin一样的效果，从而避免一些极端情况包括过拟合的发生。从这个角度来说，aggregation起到了regularization的效果。

由于aggregation具有这两个方面的优点，所以在实际应用中aggregation models都有很好的表现。



cure underfitting

- $G(\mathbf{x})$ 'strong'
 - aggregation
- ⇒ **feature transform**



cure overfitting

- $G(\mathbf{x})$ 'moderate'
 - aggregation
- ⇒ **regularization**

proper aggregation (a.k.a. 'ensemble')
⇒ **better performance**

总结

本节课主要介绍了Gradient Boosted Decision Tree。首先讲如何将AdaBoost与Decision Tree结合起来，即通过sampling和pruning的方法得到AdaBoost-D Tree模型。然后，我们从optimization的角度来看AdaBoost，找到好的hypothesis也就是找到一个好的方向，找到权重 α 也就是找到合适的步进长度。接着，我们从binary classification的0/1 error推广到其它的error function，从Gradient Boosting角度推导了regression的squared error形式。Gradient Boosting其实就是不断迭代，做residual fitting。并将其与Decision Tree算法结合，得到了经典的GBDT算法。最后，我们将所有的aggregation models做了总结和概括，这些模型有的能防止欠拟合有的能防止过拟合，应用十分广泛。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程