

Gradient Boosted Decision Tree

一、Adaptive Boosted Decision Tree

1. 对比 Decision Tree 和 AdaBoost-Tree

我们总结过关于几类aggregation的例子

- Bagging (bootstrap) - uniform
- AdaBoost - non-uniform
- Decision Tree - conditionally

uniformly的Decision Tree就是Random Forest

配合weights的Decision Tree就是AdaBoost-DTree

From Random Forest to AdaBoost-DTree

```
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  $\mathbf{u}^{(t)}$ 
  ② obtain tree  $g_t$  by
    DTree( $\mathcal{D}, \mathbf{u}^{(t)}$ )
  ③ calculate 'vote'  $\alpha_t$  of  $g_t$ 
return  $G = \text{LinearHypo}(\{(g_t, \alpha_t)\})$ 
```

need: **weighted** DTree($\mathcal{D}, \mathbf{u}^{(t)}$)

2. 于是我们试图在Decision-Tree中加入权重，但是我们希望让DT成为一个黑盒子而不是直接对这个模型进行大幅度的改动，因此我们希望抽样的时候就按照权重的比例来抽样，从而我们不需要再对于原来的算法进行修改：

Weighted Decision Tree Algorithm

Weighted Algorithm

minimize (regularized) $E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{N} \sum_{n=1}^N u_n \cdot \text{err}(y_n, h(\mathbf{x}_n))$

if using existing algorithm as **black box** (no modifications),
to get $E_{\text{in}}^{\mathbf{u}}$ approximately optimized.....

'Weighted' Algorithm in Bagging

weights \mathbf{u} expressed by
bootstrap-sampled copies
—request size- N' data $\tilde{\mathcal{D}}_t$
by bootstrapping with \mathcal{D}

A General Randomized Base Algorithm

weights \mathbf{u} expressed by
sampling proportional to u_n
—request size- N' data $\tilde{\mathcal{D}}_t$
by sampling $\propto \mathbf{u}$ on \mathcal{D}

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

3. 我们不希望有一棵树太强，这样整个模型的效果就是去了，于是我们希望通过剪枝来改善，或者仅仅是限制树的高度。

Weak Decision Tree Algorithm

AdaBoost: **votes** $\alpha_t = \ln \diamond_t = \ln \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ with **weighted error rate** ϵ_t

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

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

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

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

4. 极端的剪枝条件下，这个AdaBoost-DT就是一个AdaBoost-Stump

AdaBoost with Extremely-Pruned Tree

what if DTree with **height** ≤ 1 (extremely pruned)?

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? :-)

AdaBoost-Stump
= **special case** of AdaBoost-DTree

二、Optimization View of AdaBoost

1. 回忆一下在AdaBoost中权重的更新方法，我们有如下的推导

Example Weights of AdaBoost

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

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

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

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

AdaBoost当中的权重和 voting score有着一些关系

2. 类比我们在SVM中得到的Margin，我们可以看出实际上voting score是一种有符号且没有被归一化的margin，进而我们需要让voting score是很大的正数就需要 u_n^{T+1} 足够小

结论：AdaBoost 可以减小 $\sum_{n=1}^N u_n^t$

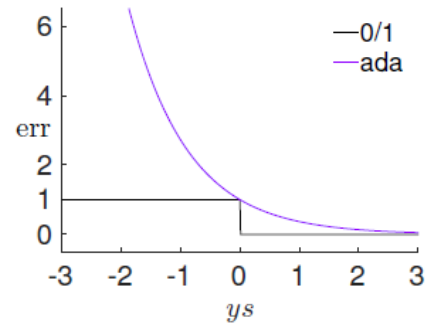
AdaBoost Error Function

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

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

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

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



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

$\widehat{\text{err}}_{\text{ADA}}(s, y) = e^{-ys}$ exponential error measure, 它作为 $\text{err}_{0/1}$ 的一个upper bound可以作为error function的度量

3. 对于 $\widehat{\text{err}}_{\text{ADA}}(s, y) = e^{-ys}$ 我们也希望通过gradient descent的方法来减少这个误差。但是这个时候我们不知道对于权重的梯度到底是多少，我们先照搬一下：

Gradient Descent on AdaBoost Error Function

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

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

at iteration t , to find g_t , solve

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

good h : minimize $\sum_{n=1}^N u_n^{(t)} (-y_n h(\mathbf{x}_n))$

我们加入一个 $\eta h(x_n)$ 相当于走了一小步，就像梯度下降中向梯度方向走的那一步，我们做代数变换，以及泰勒展开得到了是

$$\hat{E}_{ADA} \approx \sum_{n=1}^N u_n^{(t)} - \eta \sum_{n=1}^N u_n^{(t)} y_n h(x_n) \quad (1)$$

于是问题转换为找到 **good h (direction of function)**

$$s.t. \text{ minimize } \sum_{n=1}^N u_n^{(t)} (-y_n h(x_n)) \quad (2)$$

函数的方向就类似于梯度方向。接着我们做代数变形

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

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

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

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

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

那么现在我们就需要让 $E_{\text{in}}^{u^{(t)}}(h)$ 越来越小就行了，于是我们需要做的就是利用AdaBoost中的A降低它即可，我们通过演算法得到的 $g_t = h$ 就是我们希望得到的结果，于是我们以此降低了 \hat{E}_{ADA}

4. 设置blending的权重：

Deciding Blending Weight as Optimization

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 $\min_{\eta} \hat{E}_{ADA} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n))$

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

by solving $\frac{\partial \hat{E}_{ADA}}{\partial \eta} = 0$, **steepest** $\eta_t = \ln \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} = \alpha_t$, **remember? :-)**
—AdaBoost: **steepest** decent with **approximate functional gradient**

我们首先通过最小化 $\hat{E}_{ADA} = \sum_{n=1}^N u_n^{(t)} e^{-y_n \eta h(\mathbf{x}_n)}$ 来找到最好的 $h = g_t$ ，也就是 **function direction** 紧接着我们希望利用这个学到的 g_t 来得到相应的 η （事实上刚才我们并没有引入 η 这个参数，或者说 $\eta = 1$ ），接下来就是需要找到一个 η_t ，它每一次都找到下降最快的位置。这个方法被称为是 **steepest decent for optimization** 也就是所谓的“**最速下降法**”

我们分析一下这个求和式，根据 $y_n, g_t(\mathbf{x}_n)$ 的关系可以分成两种情况，分别计算，其中：

$$y_n = g_t(\mathbf{x}_n) : u_n^{(t)} e^{-\eta} \quad (correct) \quad (3)$$

$$y_n \neq g_t(\mathbf{x}_n) : u_n^{(t)} e^{+\eta} \quad (incorrect) \quad (4)$$

$$\hat{E}_{ADA} = \left(\sum_{n=1}^N u_n^{(t)} \right) * ((1 - \epsilon_t) e^{-\eta} + \epsilon_t e^{+\eta}) \quad (5)$$

通过对 η 求导，我们得到了如下的式子，注意结果正好是我们之前使用的 α_t ：

$$\eta_t = \ln \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} = \alpha_t \quad (6)$$

因此AdaBoost可以看作是近似函数梯度的最速下降法！

三、Gradient Boosting

我们把AdaBoost的思想进行进一步的推广，我们希望得到更加普适的结论，在AdaBoost中我们的 $h(\mathbf{x})$ 是二元分类的假设函数，输出是二值的，现在我们对 $h(\mathbf{x})$ 作任何限制(可以是回归可以是分类)

Gradient Boosting for Arbitrary Error Function

AdaBoost

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

with binary-output hypothesis h

GradientBoost

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

with any hypothesis h (usually real-output hypothesis)

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

Regression: MSE

GradientBoost for Regression

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

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

naïve solution $h(\mathbf{x}_n) = -\infty \cdot (s_n - y_n)$
if no constraint on h

我们首先让 \min_h 项最小，通过泰勒展开配合对于voting score的求导我们得到了
 $h(x_n) = -\infty * 2(s_n - y_n)$

在对于 h 没有任何限制的情况下我们可以得到这个解。显然还不太恰当。我们看看在有条件限制情况下它的表现

事实上 h 的量级不是十分重要，因为我们之后学习的是 η ，接下来我们为了避免naive solution我们采取正则化的操作，就是不希望 h 太大：

Learning Hypothesis as Optimization

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

- **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 $\{(\mathbf{x}_n, \underbrace{y_n - s_n}_{\text{residual}})\}$

GradientBoost for regression:

find $g_t = h$ by regression with **residuals**

通过化简，我们看出实际上我们就是求对于余数 residual 的 MSE 误差。简言之，对于 regression 的 GradientBoost 就是对于 residuals 来说 MSE 的最小的 $g_t = h$

搞定了 g_t 我们考虑怎么学习 η ，这个时候实际上就是一个对于 residual 的 linear regression:

Deciding Blending Weight as Optimization

after finding $g_t = h$,

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

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

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

GradientBoost for regression: $\alpha_t = \text{optimal } \eta$
by $g_t\text{-transformed linear regression}$

因此，GB for regression 就是找到最好的 $\alpha_t = \eta$ ，方式是通过 g_t 转化的 linear regression。

关于它的解：

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

下面对于GDBT做一个总结：

Putting Everything Together

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

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

GBDT: ‘regression sibling’ of AdaBoost-DTree
—**popular in practice**

1. 通过回归算法获 g_t 使得 $MSE(X_n, y_n - s_n)$ 最小
2. 利用单变量线性回归得到 $\alpha_t = LR(g_t(x_n), y_n - s_n)$
3. 更新voting score $s_n = s_n + \alpha_t g_t(x_n)$
4. 重复若干次
5. 返回 $G(x) = \sum_{t=1}^T \alpha_t g_t(x)$

GBDT在回归中用的特别多，和AdaBoost-DTree是“回归兄弟”

四、Summary of Aggregation Models

下面对于各种模型进行一个总结：

首先是Aggregation-learning中的几个基本类型，实际上boosting类型的算法是最为管饭应用的

Map of Aggregation-Learning Models

learning: aggregate **as well as** getting **diverse** g_t

Bagging

diverse g_t by
bootstrapping;
uniform vote
by nothing :-)

AdaBoost

diverse g_t
by reweighting;
linear vote
by steepest search

Decision Tree

diverse g_t
by data splitting;
conditional vote
by branching

GradientBoost

diverse g_t
by residual fitting;
linear vote
by steepest search

boosting-like algorithms most popular

其次是Aggregation of Aggregation的几个模型，在前人的基础上再创新高，三者应用都很广泛。

Map of Aggregation of Aggregation Models

Bagging

AdaBoost

Decision Tree

Random Forest

randomized bagging
+ 'strong' DTree

AdaBoost-DTree

AdaBoost
+ 'weak' DTree

GradientBoost

GBDT

GradientBoost
+ 'weak' DTree

all three frequently used in practice