第06章：深入浅出ML之Boosting家族

By ZhouYong

发表于 2015-12-12

- author: zhouyongsdzh@foxmail.com
- date: 2015-11-12
- weibo: @周永_52ML

内容列表

文章目录

写在前面

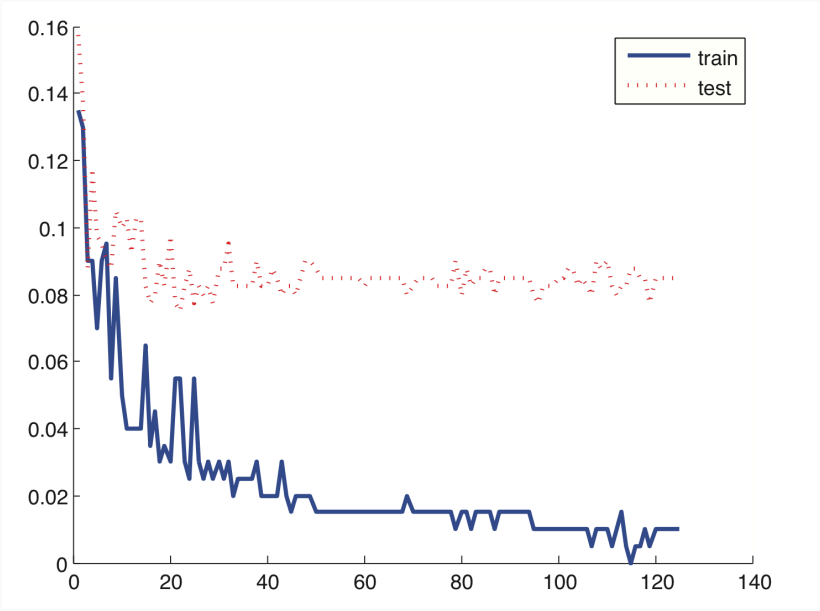
提升（boosting）方法是一类应用广泛且非常有效的统计学习方法。

在2006年，Caruana和Niculescu-Mizil等人完成了一项实验，比较当今世界上现成的分类

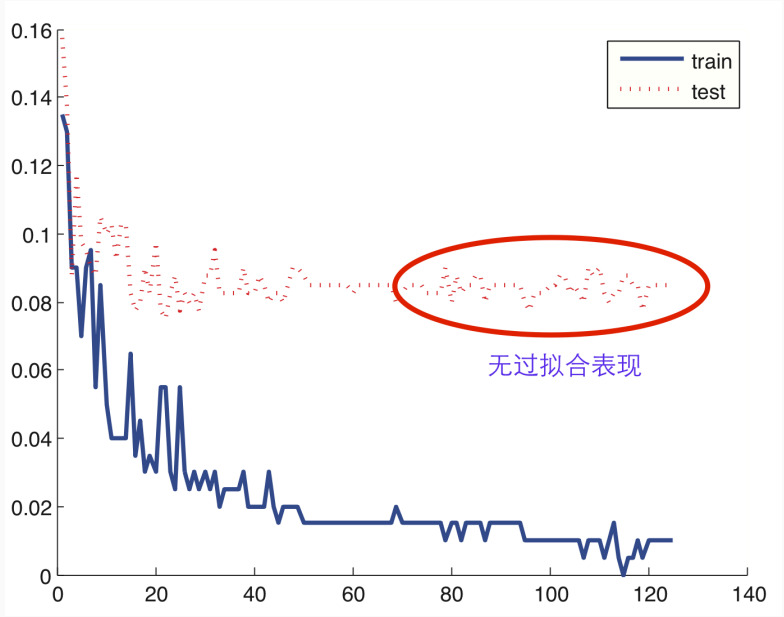
器（off-the-shelf classifiers）中哪个最好？实现结果表明Boosted Decision Tree（提升决策树）不管是在misclassification error还是produce well-calibrated probabilities方面都是最好的分离器，以ROC曲线作为衡量指标。（效果第二好的方法是随机森林）

> 参见paper：《An Empirical Comparison of Supervised Learning Algorithms》ICML2006.

下图给出的是Adaboost算法（Decision Stump as Weak Learner）在处理二类分类问题时，随着弱分类器的个数增加，训练误差与测试误差的曲线图。



损失函数示意图



损失函数示意图

从图中可以看出，Adaboost算法随着模型复杂度的增加，测试误差（红色点线）基本保持稳定，并没有出现过拟合的现象。

其实不仅是Adaboost算法有这种表现，Boosting方法的学习思想和模型结构上可以保证其不容易产生过拟合（除非Weak Learner本身出现过拟合）。

下面我们主要是从损失函数的差异，来介绍Boosting的家族成员；然后我们针对每个具体的家族成员，详细介绍其学习过程和核心公式；最后从算法应用场景和工具方法给出简单的介绍。

Boosting

Boosting介绍

- 基本思想

  Boosting方法基于这样一种思想：

  > 对于一个复杂任务来说，将多个专家的判定进行**适当的综合**得出的判断，要比其中任何一个专家单独的判断好。很容易理解，就是”三个臭皮匠顶个诸葛亮”的意思…😄😄😄。

- 历史由来

  历史上，Kearns和Valiant首先提出了”强可学习（strongly learnable）”和“弱可学习（weakly learnable）”的概念。他们指出：

  > **在概率近似正确（probably approximately correct，PAC）学习框架中：**
  >
  > ①. 一个概念（一个类，label），如果存在一个多项式的学习算法能够学习它，**并且正确率很高**，那么就称这个概念是强可学习的；
  >
  > ②. 一个概念（一个类，label），如果存在一个多项式的学习算法能够学习它，**学习的正确率仅比随机猜测略好**，那么就称这个概念是弱可学习的。
  >
  > Schapire后来证明了: **强可学习和弱可学习是等价的。** 也就是说，**在PAC学习的框架下，一个概念是强可学习的 充分必要条件 是这个概念是弱可学习的。** 表示如下：
  >
  > $$强可学习 \Leftrightarrow 弱可学习$$
  >
  > $$强\ 可\ 学\ 习 \Leftrightarrow 弱\ 可\ 学\ 习$$

  如此一来，问题便成为：在学习中，如果已经发现了”弱学习算法”，那么能否将它提升为”强学习算法”？ 通常的，发现弱学习算法通常要比发现强学习算法容易得多。那么如何具体实施提升，便成为开发提升方法时所要解决的问题。关于提升方法的研究很多，最具代表性的当数AdaBoost算法（是1995年由Freund和Schapire提出的）。

- Boosting学习思路

  对于一个学习问题来说（以分类问题为例），给定训练数据集，求一个弱学习算法要比求一个强学习算法要容易的多。Boosting方法就是从弱学习算法出发，反复学习，

得到一系列弱分类器，然后组合弱分类器，得到一个强分类器。Boosting方法在学习过程中通过改变训练数据的权值分布，针对不同的数据分布调用弱学习算法得到一系列弱分类器。

这里面有两个问题需要回答：

1. 在每一轮学习之前，如何改变训练数据的权值分布?
2. 如何将一组弱分类器组合成一个强分类器?

> 具体不同的boosting实现，主要区别在弱学习算法本身和上面两个问题的回答上。

针对第一个问题，Adaboost算法的做法是：

提高那些被前一轮弱分类器错误分类样本的权值，而降低那些被正确分类样本的权值。

> 如此，那些没有得到正确分类的样本，由于其权值加大而受到后一轮的弱分类器的更大关注。

第二个问题，弱分类器的组合，AdaBoost采取**加权多数表决**的方法。具体地：

加大 分类误差率小 的弱分类器的权值，使其在表决中起较大的作用；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算法的巧妙之处就在于它将这些学习思路自然并且有效地在一个算法里面实现。

前向分步加法模型

英文名称：Forward Stagewise Additive Modeling

- 加法模型（addtive model）

$$f(x)=\sum_{k=1}^{K}\beta_k\cdot b(x;\gamma_k)\qquad(ml.1.6.1)$$

$$f(x)=\sum_{k=1}^{K}\beta_k\cdot b(x;\gamma_k)\qquad(ml.1.6.1)$$

其中，$b(x;\gamma_k)$ $b(x;\gamma_k)$ 为基函数，γ_k γ_k为基函数的参数，β_k β_k为基函数的系数。

- 前向分步算法

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

$$\min_{\beta_k,\gamma_k} \quad \sum_{i=1}^{M} L\left[y^{(i)}, \sum_{k=1}^{K} \beta_k b(x^{(i)};\gamma_k)\right] \qquad (ml.1.6.2)$$

  通常这是一个复杂的优化问题。前向分布算法（forward stagwise algorithm）求解这一优化问题的思路是：**因为学习的是加法模型，如果能够从前向后，每一步只学习一个基函数及其系数，逐步逼近优化目标函数式$(ml.1.6.1)$，那么就可以简化优化的复杂度**。具体地，每步只需优化如下损失函数：

> $$\min_{\beta,\gamma} \quad \sum_{i=1}^{M} L(y^{(i)}, \beta b(x^{(i)};\gamma)) \qquad (n.ml.1.6.1)$$

  给定训练数据集
  $D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\}, x^{(i)}\in\mathcal{X}\subseteq R^n, y^{(i)}\in\mathcal{Y}=\{-1,$
  损失函数$L(y, f(x))$和基函数的集合$\{b(x;\gamma)\}$，学习加法模型$f(x)$的前向分步算法如下：

> {
>
> 输入：训练数据集$D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\}$; 损失函数$L(y$
>
> 基函数集$\{b(x,\gamma)\}$；
>
> 输出：加法模型$f(x)$。
>
> 计算过程：
>
> (1). 初始化$f_0(x)=0$
>
> (2). 对于$k=1,2,\cdots,K$
>
> (a). 极小化损失函数
>
> $$(\beta_k,\gamma_k)=\arg\min_{\beta,\gamma}\sum_{i=1}^{M} L(y^{(i)}, f_{k-1}(x^{(i)})+\beta b(x;\gamma)) \quad (n.m$$
>
> 得到参数β_k,γ_k.
>
> (b). 更新
>
> $$f_k(x)=f_{k-1}(x)+\beta_k b(x;\gamma_k) \qquad (n.ml.1.6.3)$$
>
> (3). 得到加法模型
>
> $$f(x)=f_K(x)=\sum_{k=1}^{K}\beta_k b(x;\gamma_k) \qquad (n.ml.1.6.4)$$
>
> }

这样前向分步算法将**同时求解**从$k=1$到K的所有参数β_k,γ_k的优化问题简化为**逐次求解**各个β_k,γ_k的优化问题。

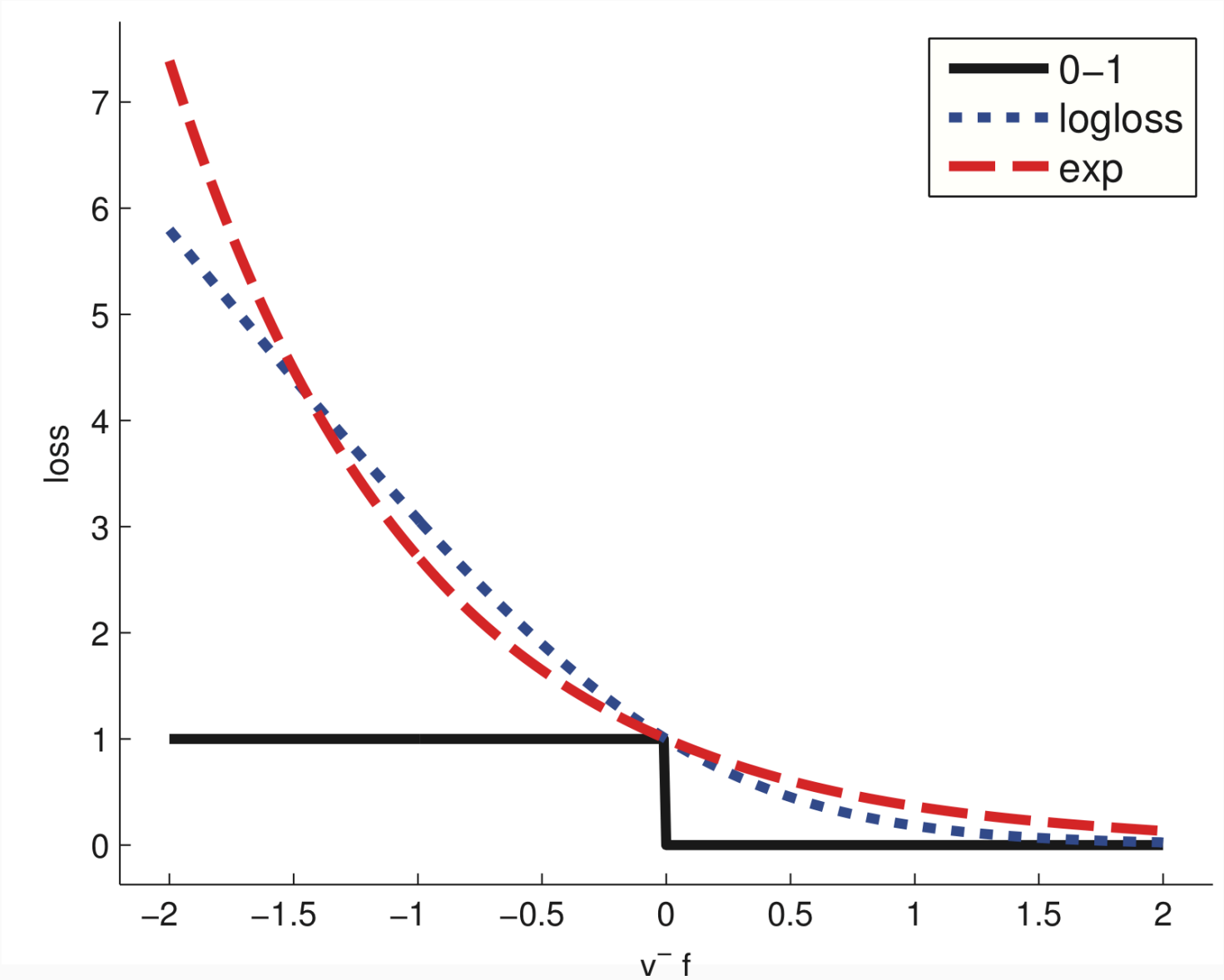
Boosting四大家族

Boosting并非是一个方法，而是一类方法。这里按照损失函数的不同，将其细分为若干类算法，下表给出了4种不同损失函数对应的Boosting方法：

名称(Name)	损失函数(Loss)	导数(Derivative)	目标函数f^*	算法
平方损失(Squared Error)	$\frac{1}{2}(y^{(i)}-f(x^{(i)}))^2$	$y^{(i)}-f(x^{(i)})$	$E[y\|x^{(i)}]$	L2Boos
绝对损失(Absolute Error)	$\|y^{(i)}-f(x^{(i)})\|$	$sign(y^{(i)}-f(x^{(i)})$	$median(y\|x^{(i)})$	Gradie Boosti
指数损失(Exponentail Loss)	$\exp(-\tilde{y^{(i)}}f(x^{(i)}))$	$-\tilde{y^{(i)}}exp(-\tilde{y^{(i)}}f(x^{(i)}))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBo
对数损失(LogLoss)	$\log(1+e^{-\tilde{y^{(i)}}f_i})$	$y^{(i)}-\pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBo

> 说明：
>
> 该表来自于Machine Learning: A Probabilistic PerspectiveP587页
>
> **L2Boosting全称：Least Squares Boosting；该算法由Buhlmann和Yu在2003年提出。**

二分类问题时损失函数示意图：



损失函数示意图

下面主要以AdaBoost算法作为示例，给出以下3个问题的解释：

- AdaBoost为什么能够提升学习精度?
- 如何解释AdaBoost算法?
- Boosting方法更具体的实例－Boosting Tree。

下面首先介绍Adaboost算法。

Adaboost

算法学习过程

Adaboost算法在分类问题中的主要特点：**通过改变训练样本的权重，学习多个分类器，并将这些分类器进行线性组合，提高分类性能。** AdaBoost－算法描述（伪代码）如下：

{

输入：训练数据集$D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\}$，$x^{(i)}\in\mathcal{X}\subseteq R^N$

$y^{(i)}\in\mathcal{Y}=\{-1,+1\}$，弱分类器；

输出：最终分类器$G(x)$.

过程：

(1). 初始化训练数据的权值分布

$$D_1=(w_{11},w_{12},\cdots,w_{1M}),\quad w_{1i}=\frac{1}{M},\ i=1,2,\cdots,M$$

(2). 训练K个弱分类器$k=1,2,\cdots,K$

(a). 使用具有权值分布D_k的训练数据集学习，得到基本分类器

$$G_k(x):\mathcal{X}\to\{-1,+1\}\qquad(ml.1.6.3)$$

(b). 计算$G_k(x)$在训练数据集上的分类误差率

$$e_k=P(G_k(x^{(i)})\neq y^{(i)})=\sum_{i=1}^{M}w_{ki}I(G_k(x^{(i)})\neq y^{(i)})\qquad(ml.1.$$

(c). 计算$G_k(x)$的系数

$$\alpha_k=\frac{1}{2}\log\frac{1-e_k}{e_k}\quad(e\text{是自然对数})\qquad(ml.1.6.5)$$

(d). 更新训练数据集的权值分布

$$D_{k+1}=(w_{k+1,1},w_{k+1,2},\cdots,w_{k+1,M})$$

$$w_{k+1,i}=\frac{w_{k,i}}{Z_k}\exp(-\alpha_k y^{(i)}G_k(x^{(i)})),\quad i=1,2,\cdots,M\quad(ml.1.$$

Z_k是规范化因子

$$Z_k=\sum_{i=1}^{M}w_{k,i}\cdot\exp(-\alpha_k y^{(i)}G_k(x^{(i)}))\qquad(ml.1.6.7)$$

使D_{k+1}成为一个概率分布。

(3). 构建基本分类器的线性组合

$$f(x)=\sum_{k=1}^{K}\alpha_k G_k(x)\qquad(ml.1.6.8)$$

得到最终的分类器

$$G(x)=sign(f(x))=sign\left(\sum_{k=1}^{K}\alpha_k G_k(x)\right)\qquad(ml.1.6.9)$$

}

- **AdaBoost算法描述说明**

- 步骤（1）假设训练数据集具有均匀（相同）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，第一步能在原始数据上学习基本分类器$G_1(x)$。

- 步骤（2）AdaBoost反复学习基本分类器，在每一轮$k=1,2,\cdots,K$顺序地执行下列操作：

  - （a）学习基本分类器：使用当前分布D_k加权的训练数据集，学习基本分类器$G_k(x)$；

  - （b）误差率：计算基本分类器$G_k(x)$在加权训练数据集上的分类误差率

    $$e_k = P(G_k(x^{(i)}) \neq y^{(i)}) = \sum_{G_k(x^{(i)}) \neq y^{(i)}} w_{ki} \qquad (n.ml.1.6.5)$$

    这里，w_{ki}表示第k轮中第i个样本的权值，$\sum_{i=1}^{M} w_{ki} = 1$。

    > 这表明，$G_k(x)$在加权的训练数据集上的分类误差率 是 被$G_k(x)$误分类样本的权值之和。由此可以看出数据权值分布D_k与基本分类器$G_k(x)$的分类误差率的关系。

  - （c）分类器权重：计算基本分类器$G_k(x)$的系数α_k，α_k表示$G_k(x)$在最终分类器中的重要性

    > 根据(ml.1.6.3)中公式可知，当$e_k \leq 0.5$时，$\alpha_k \geq 0$，并且α_k随着e_k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。

  - （d）更新训练数据的权值分布，为下一轮做准备，公式$(ml.1.6.6)$可以写成：

    $$w_{k+1,i} = \begin{cases} \frac{w_{ki}}{Z_k} e^{-\alpha_k}, & G_k(x^{(i)}) = y^{(i)} \\ \frac{w_{ki}}{Z_k} e^{\alpha_k}, & G_k(x^{(i)}) \neq y^{(i)} \end{cases} \qquad (n.ml.1.6.6)$$

    > 由此可知，被基本分类器$G_k(x)$误分类样本的权值得以扩大，而被正确分类样本的权值却得以缩小。相比较来说，误分类样本的权值被放大$e^{2\alpha_k} = \frac{e_k}{1-e_k}$倍。因此，误分类样本在下一轮学习中起更大的作用。

    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中起不同的作用，这也是AdaBoost的一个特点。

- 步骤（3）线性组合$f(x)$实现K个基本分类器的加权表决。系数α_k表示了极本分类器$G_k(x)$的重要性。

> 注意：在这里所有α_k之和并不为1。$f(x)$的符号决定实例x的类别，$f(x)$的绝对值表示分类的精确度。

利用基本分类器的线性组合构建最终分类器是AdaBoost的另一个特点。

示例：AdaBoost算法

此示例参考李航老师的《统计学习方法》.

> 给定下表所示训练数据。
>
> | 序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
> |---|---|---|---|---|---|---|---|---|---|---|
> | x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
> | y | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 | -1 |
>
> 假设弱分类器由$x \leq v$或$x > v$产生，其阈值v使该分类器在训练数据集上分类误差率最低。试用AdaBoost算法学习一个强分类器。
>
> 解： 首先初始化数据权值分布（均匀分布）：
>
> $$D_1 = (w_{1,1}, w_{1,2}, \cdots, w_{1,10}), \quad w_{1,i} = 0.1,\ i = 1, 2, \cdots, 10$$
>
> 对$k = 1$,
>
> (a). 在权值分布为D_1的训练数据上，阈值v取2.5时，分类误差率最低，故基本分类器为：
>
> $$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$
>
> (b). $G_1(x)$在训练数据集上的误差率$e_1 = P(G_1(x^{(i)}) \neq y^{(i)}) = 0.3$;
>
> (c). 计算$G_1(x)$的系数：$\alpha_1 = \frac{1}{2}\log\frac{1-e_1}{e_1} = 0.4236$;
>
> (d). 更新训练数据的权值分布:

$$
\begin{aligned}
&D_2 = (w_{2,1}, w_{2,2}, \cdots, w_{2,10}) \\
&w_{2,i} = \frac{w_{1,i}}{Z_1}\exp(-\alpha_1 y^{(i)} G_1(x^{(i)})), \quad i = 1, 2, \cdots, 10 \\
&D_2 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, \\
&\qquad 0.1666, 0.1666, 0.1666, 0.0715) \\
&f_1(x) = 0.4236 G_1(x)
\end{aligned}
$$

分类器$sign[f_1(x)]$在训练数据上有3个误分类点。

对$k = 2$,

(a). 在权值分布为D_2的训练数据上，阈值v取8.5时，分类误差率最低，基本分类器为：

$$
G_2(x) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}
$$

(b). $G_2(x)$在训练数据集上的误差率$e_2 = 0.2143$;

(c). 计算$G_2(x)$的系数：$\alpha_2 = 0.6496$;

(d). 更新训练数据的权值分布:

$$
\begin{aligned}
&D_3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.1667, \\
&\qquad 0.1060, 0.1060, 0.1060, 0.0455) \\
&f_2(x) = 0.4236 \cdot G_1(x) + 0.6496 \cdot G_2(x)
\end{aligned}
$$

分类器$sign[f_2(x)]$在训练数据上有3个误分类点。

对$k = 3$,

(a). 在权值分布为D_3的训练数据上，阈值v取5.5时，分类误差率最低，基本分类器为：

$$
G_3(x) = \begin{cases} 1, & x < 5.5 \\ -1, & x > 5.5 \end{cases}
$$

(b). $G_3(x)$在训练数据集上的误差率$e_3 = 0.1820$;

(c). 计算$G_3(x)$的系数：$\alpha_2 = 0.7514$;

(d). 更新训练数据的权值分布:

$$
D_4 = (0.125, 0.125, 0.125, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, 0.125)
$$

于是得到模型线性组合

$$f_3(x) = 0.4236 \cdot G_1(x) + 0.6496 \cdot G_2(x) + 0.7514 \cdot G_3(x)$$

分类器$sign[f_3(x)]$在训练数据上误分类点个数为0。

于是最终分类器为：

$$\begin{aligned} G(x) &= sign[f_3(x)] \\ &= sign[0.4236 \cdot G_1(x) + 0.6496 \cdot G_2(x) + 0.7514 \cdot G_3(x)] \end{aligned}$$

训练误差分析

AdaBoost算法最基本的性质是它能在学习过程中不断减少训练误差，即在训练数据集上的分类误差率。 对于这个问题，有个定理可以保证分类误差率在减少－AdaBoost的训练误差界。

- **定理：AdaBoost训练误差界**

［定理］AdaBoost训练误差界

$$\frac{1}{M}\sum_{i=1}^{M} I(G(x^{(i)}) \neq y^{(i)}) \leq \frac{1}{M}\sum_{i} \exp(-y^{(i)} f(x^{(i)})) = \prod_{k=1}^{K} Z_k \qquad (ml.1.6.10)$$

其中，$G(x), f(x)$和Z_k分别由公式$(ml.1.6.9), (ml.1.6.8), (ml.1.6.7)$给出

证明如下：

当$G(x^{(i)}) \neq y^{(i)}$时，$y^{(i)} f(x^{(i)}) < 0$，因而$\exp(-y^{(i)} f(x^{(i)})) \geq 1$。由此，可以直接推导出前半部分。

后半部分的推导要用到Z_k的定义式$(ml.1.6.7)$和$(ml.1.6.6)$的变形:

$$w_{k,i} \exp(-\alpha_k y^{(i)} G_k(x^{(i)})) = Z_k w_{k+1,i} \qquad (n.ml.1.6.7)$$

推导如下：

$$
\begin{aligned}
\frac{1}{M}\sum_{i=1}^{M}\exp(-y^{(i)}f(x^{(i)})) &= \frac{1}{M}\sum_{i=1}^{M}\exp\left(-\sum_{k=1}^{K}\alpha_k y^{(i)}G_k(x^{(i)})\right) \\
&= \sum_{i=1}^{M}w_{1,i}\prod_{k=1}^{K}\exp(-\alpha_k y^{(i)}G_k(x^{(i)})) \\
&= \sum_{i=1}^{M}w_{1,i}\exp(-\alpha_1 y^{(i)})G_k(x^{(i)})\prod_{k=2}^{K}\exp(-\alpha_k y^{(i)}G_k(x^{(i)})) \\
&= Z_1\sum_{i=1}^{M}w_{2,i}\prod_{k=2}^{K}\exp(-\alpha_k y^{(i)}G_k(x^{(i)})) \\
&= Z_1Z_2\sum_{i=1}^{M}w_{3,i}\prod_{k=3}^{K}\exp(-\alpha_k y^{(i)}G_k(x^{(i)})) \\
&= \cdots\cdots \\
&= Z_1Z_2\cdots Z_{K-1}\sum_{i=1}^{M}w_{K,i}\exp(-\alpha_K y^{(i)}G_K(x^{(i)})) \\
&= \prod_{k=1}^{K}Z_k
\end{aligned}
$$

注意：$w_{1,i}=\frac{1}{M}$

这一定理说明：**可以在每一轮选取适当的G_k使得Z_k最小，从而使训练误差下降最快。** 对于二类分类问题，有如下定理。

- 定理：二类分类问题AdaBoost训练误差界

［定理］二类分类问题AdaBoost训练误差界

$$
\prod_{k=1}^{K}Z_k=\prod_{k=1}^{K}\left[2\sqrt{e_k(1-e_k)}\right]=\prod_{k=1}^{K}\sqrt{(1-4\gamma_k^2)}\leq\exp\left(-2\sum_{k=1}^{K}\gamma_k^2\right)\qquad(ml.1.6
$$

这里，$\gamma_k=0.5-e_k$

证明：由公式$(ml.1.6.7)$和$(n.ml.1.6.5)$可得：

$$
\begin{aligned}
Z_k &= \sum_{i=1}^{M} w_{k,i} \exp(-\alpha_k y^{(i)} G_k(x^{(i)})) \\
&= \sum_{y^{(i)}=G_k(x^{(i)})} w_{k,i} \cdot e^{-\alpha_k} + \sum_{y^{(i)} \neq G_k(x^{(i)})} w_{k,i} \cdot e^{\alpha_k} \qquad (n.ml.1.6.9) \\
&= (1-e_k) \cdot e^{-\alpha_k} + e_k \cdot e^{\alpha_k} \\
&= 2\sqrt{e_k(1-e_k)} = \sqrt{1-4\gamma_m^2}
\end{aligned}
$$

注：$\alpha_k = \frac{1}{2}\log\frac{1-e_k}{e_k}, e^{\alpha_k} = \sqrt{\frac{1-e_k}{e_k}}$

对于不等式部分

$$
\prod_{k=1}^{K} \sqrt{1-4\gamma_m^2} \leq \exp\left(-2\sum_{k=1}^{K}\gamma_k^2\right) \qquad (n.ml.1.6.10)
$$

则可根据e^x和$\sqrt{1-x}$在点$x=0$的泰勒展开式推出不等式
$\sqrt{1-4\gamma_m^2} \leq \exp(-2\gamma_m^2)$。

[推论] AdaBoost训练误差指数速率下降

如果存在$\gamma > 0$，对所有的m有$\gamma_k \geq \gamma$，则有

$$
\frac{1}{M}\sum_{i=1}^{M} I(G(x^{(i)}) \neq y^{(i)}) \leq \exp(-2K\gamma^2) \qquad (ml.1.6.12)
$$

|

推论表明，在此条件下，**AdaBoost的训练误差是以指数速率下降的**。这一性质对于AdaBoost计算（迭代）效率是利好消息。

> 注意：AdaBoost算法不需要知道下界γ，这正是Freund和Schapire设计AdaBoost时所考虑的。与一些早期的提升方法不同，AdaBoost具有适应性，即它能适应弱分类器各自的训练误差率。这也是其算法名称的由来（适应的提升）。Ada是Adaptive的简写。

前向分步加法模型与Adaboost

AdaBoost算法还有另一个解释，即可以认为**AdaBoost算法是模型为加法模型、损失函数为指数函数、学习算法为前向分步算法**时的学习方法。

根据前向分步算法可以推导出AdaBoost，用一句话叙述这一关系.

AdaBoost算法是前向分步加法算法的特例

此时，模型是由基本分类器组成的加法模型，损失函数是指数函数。

> 证明：**前向分步算法学习的是加法模型，当基函数为基本分类器时，该加法模型等价于AdaBoost的最终分类器：**
>
> $$f(x)=\sum_{k=1}^{K}\alpha_k G_k(x) \qquad (n.ml.1.6.11)$$
>
> 由基本分类器$G_k(x)$及其系数α_k组成，$k=1,2,\cdots,K$。前向分步算法逐一学习基函数，这一过程与AdaBoost算法逐一学习基本分类器的过程一致。
>
> 下面证明：
>
> **前向分步算法的损失函数是指数损失函数（Exponential）**
> $L(y,f(x))=\exp[-yf(x)]$ **时，其学习的具体操作等价于AdaBoost算法学习的具体操作。**
>
> 假设经过$k-1$轮迭代，前向分步算法已经得到$f_{k-1}(x)$:
>
> $$\begin{aligned} f_{k-1}(x) &= f_{k-2}(x)+\alpha_{k-1}G_{k-1}(x) \\ &= \alpha_1 G_1(x)+\cdots+\alpha_{m-1}G_{m-1}(x) \end{aligned} \qquad (n.ml.1.6.12)$$
>
> 在第k轮迭代得到$\alpha_k, G_k(x)$和$f_k(x)$。
>
> $$f_k(x)=f_{k-1}(x)+\alpha_k G_k(x) \qquad (n.ml.1.6.13)$$
>
> 目标是使前向分步算法得到的α_k和$G_k(x)$使$f_k(x)$在训练数据集D上的指数损失最小，即
>
> $$(\alpha_k,G_k(x))=\arg\min_{\alpha,G}\underbrace{\sum_{i=1}^{M}\exp[-y^{(i)}(f_{k-1}(x)+\alpha G(x^{(i)}))]}_{\text{指数损失表达式}} \qquad (n.ml.1.6.14)$$
>
> 进一步可表示为：
>
> $$(\alpha_k,G_k(x))=\arg\min_{\alpha,G}\sum_{i=1}^{M}\overline{w}_{k,i}\cdot\exp[-y^{(i)}\alpha G(x^{(i)})] \qquad (n.ml.1.6.15)$$

其中，$\overline{w}_{k,i}=\exp[-y^{(i)}f_{k-1}(x^{(i)})]$表示第$i$样本在之前模型上的指数损失。因为$\overline{w}_{k,i}$既不依赖$\alpha$也不依赖$G$，所以与最小化无关。但$\overline{w}_{k,i}$依赖于$f_{k-1}(x)$，随着每一轮迭代而发生变化。

现在使公式$(n.ml.1.6.15)$达到最小的α_k^*和G_k^*就是AdaBoost算法所得到的α_k和$G_k(x)$。求解公式$(n.ml.1.6.15)$可分为两步：

第一步：求G_k^*. 对于任意$\alpha>0$，使公式$(n.ml.1.6.15)$最小的$G(x)$由下式得到：

$$G_k^*(x)=\arg\min_G\sum_{i=1}^M\overline{w}_{k,i}\cdot I(y^{(i)}\neq G(x^{(i)}))\qquad(n.ml.1.6.16)$$

此分类器$G_k^*(x)$即为AdaBoost算法的基本分类器$G_k(x)$，因为它是使第k轮加权训练数据分类误差率最小的基本分类器。

之后，求α_k^*。参考公式$(n.ml.1.6.5)$，公式$(n.ml.1.6.15)$中的：

$$\begin{aligned}\sum_{i=1}^M\overline{w}_{k,i}\cdot\exp[-y^{(i)}\alpha G(x^{(i)})]&=\sum_{y^{(i)}=G_k(x^{(i)})}\overline{w}_{k,i}\cdot e^{-\alpha}+\sum_{y^{(i)}\neq G_k(x^{(i)})}\overline{w}_{k,i}\cdot e^{\alpha}\\&=(e^{\alpha}-e^{-\alpha})\sum_{i=1}^M\overline{w}_{k,i}I(y^{(i)}\neq G(x^{(i)}))+e^{-\alpha}\sum_{i=1}^M\overline{w}\end{aligned}$$

把已得到的G_k^*带入公式$(n.ml.1.6.17)$，并对α求导（导数为0），即得到使公式$(n.ml.1.6.15)$最小的α。

$$\alpha_k^*=\frac{1}{2}\log\frac{1-e_k}{e_k}$$

e_k为分类误差率

$$e_k=\frac{\sum_{i=1}^M\overline{w}_{k,i}I(y^{(i)}\neq G_k(x^{(i)}))}{\sum_{i=1}^M\overline{w}_{k,i}}=\sum_{i=1}^M w_{k,i}I(y^{(i)}\neq G_k(x^{(i)}))$$

可以看出，这里求得的α_k^*与AdaBoost算法$(2)-(c)$步的α_k完全一致。

最后看一下每一轮样本权值的更新。由：$f_k(x)=f_{k-1}(x)+\alpha_kG_k(x)$以及$\overline{w}_{k,i}=\exp[-y^{(i)}f_{k-1}(x^{(i)})]$，可得：

$$\overline{w}_{k+1,i}=\overline{w}_{k,i}\exp[-y^{(i)}\alpha_kG_k(x)]$$

这与Adaboost算法的第$(2)-(d)$步的样本权值的更新，可看出二者是等价的（只相差

规范化因子）。

- AdaBoost算法缺点

  - 对异常点敏感

    指数损失存在的一个问题是不断增加误分类样本的权重（指数上升）。如果数据样本是异常点（outlier），会极大的干扰后面基本分类器学习效果；

  - 模型无法用于概率估计

    > MLAPP中的原话："$e^{-\tilde{y}f}$ is not the logarithm of any pmf for binary variables $\tilde{y} \in \{-1, +1\}$; consequently we cannot recover probability estimate from $f(x)$."
    >
    > 意思就是说对于取值为$\tilde{y} \in \{-1, +1\}$的随机变量来说，$e^{-\tilde{y}f}$不是任何概率密度函数的对数形式，模型$f(x)$的结果无法用概率解释。

Boosted Decision Tree

提升决策树是指以**分类与回归树（CART）**为基本分类器的提升方法，被认为是统计学习中性能最好的方法之一。

> 提升决策树简称提升树，Boosting Tree.

提升树模型

提升树模型实际采用加法模型（即基函数的线性组合）与前向分步算法，以决策树为基函数的提升方法称为提升树（Boosting Tree）。

> 对分类问题决策树是二叉分类树，对回归问题决策树是二叉回归树。在6.1.3节AdaBoost例子中，基本分类器是\(xv\)，可以看作是由一个跟结点直接连接两个叶结点的简单决策树，即所谓的决策树桩（Decision Stump）。

提升树模型可以表示为CART决策树的加法模型：

$$f_K(x) = \sum_{k=1}^{K} T(x; \Theta_k) \qquad (ml.1.6.13)$$

其中，$T(x;\Theta_k)$表示二叉决策树，Θ_k为决策树的参数，K为树的个数。

> 基本学习器－CART决策树，请参考第03章：深入浅出ML之Tree-Based家族

提升树算法

提升树算法采用前向分步算法。首先确定初始提升树$f_0(x)=0$，第k步的模型为：

$$f_k(x)=f_{k-1}(x)+T(x;\Theta_k) \qquad (ml.1.6.14)$$

其中，$f_{k-1}(x)$为当前模型，通过**经验风险极小化**确定下一颗决策树的参数Θ_k，

$$\hat{\Theta}_k=\arg\min_{\Theta_k}\sum_{i=1}^{M}L(y^{(i)},\ f_{k-1}(x)+T(x^{(i)};\Theta_k)) \qquad (ml.1.6.15)$$

> 由于树的线性组合可以很好的拟合训练数据，即使数据中的输入和输出之间的关系很复杂也是如此，所以提升树是一个高功能的学习算法。

提升树家族

不同问题的提升树学习算法，其主要区别在于**损失函数**不同。平方损失函数常用于回归问题，用指数损失函数用于分类问题，以及绝对损失函数用于决策问题。

- **二叉分类树**

  对于二类分类问题，提升树算法只需要将AdaBoost算法例子中的基本分类器限制为**二叉分类树**即可，可以说此时的决策树算法时AdaBoost算法的特殊情况。

  > 损失函数仍为指数损失，提升树模型仍为前向加法模型。

- **二叉回归树**

  已知训练数据集
  $D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\},x^{(i)}\in\mathcal{X}\subseteq R^n,y^{(i)}\in\mathcal{Y}\subseteq R,\mathcal{Y}$为输出空间。如果将输入空间$\mathcal{X}$划分为$J$个互不相交的区域$R_1,R_2,\cdots,R_J$，并且在每个区域上确定输出的常量$c_j$，那么树可以表示为：

  $$T(x;\Theta)=\sum_{j=1}^{J}c_jI(x\in R_j) \qquad (ml.1.6.16)$$

  其中，参数$\Theta=\{(R_1,c_1),(R_2,c_2),\cdots,(R_J,c_J)\}$表示树的区域划分和各区域上

的常数。J是回归树的复杂度即叶结点的个数。

- 回归问题提升树－前向分步算法

> 回归问题提升树使用以下前向分步算法：
>
> $$\begin{aligned} f_0(x) &= 0 \\ f_k(x) &= f_{k-1}(x) + T(x;\Theta_k) \quad k=1,2,\cdots,K \\ f_K(x) &= \sum_{k=1}^{K} T(x;\Theta_k) \end{aligned} \tag{n.ml1.6.17}$$
>
> 在前向分布算法的第k步，给定当前模型$f_{k-1}(x)$，需求解：
>
> $$\hat{\Theta}_k = \arg\min_{\Theta_k} \sum_{i=1}^{M} \underbrace{L(y^{(i)},\ f_{k-1}(x) + T(x^{(i)};\Theta_k))}_{\text{损失函数}} \tag{n.ml.1.6.18}$$
>
> 得到$\hat{\Theta}_k$，即第k颗树的参数。
>
> 当采用**平方误差损失函数**时，
>
> $$L(y, f(x)) = (y - f(x))^2 \tag{n.ml.1.6.19}$$
>
> 将平方误差损失函数展开为：
>
> $$\begin{aligned} &L(y, f_{k-1}(x) + T(x;\Theta_k)) \\ &\quad = [y - f_{k-1}(x) - T(x;\Theta_k)]^2 \\ &\quad = [r - T(x;\Theta_k)]^2 \end{aligned} \tag{n.ml.1.6.20}$$
>
> 这里$r = y - f_{k-1}(x)$，表示当前模型的拟合数据的残差（residual）。所以，对回归问题的提升树算法来说，只需要简单地拟合当前模型的残差。
>
> > 由于损失函数是平方损失，因此该方法属于**L2Boosting**的一种实现。

- **回归问题提升树－算法描述**

{

输入：训练数据集$D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\},x^{(i)}\in\mathcal{X}\subseteq R$

输出：提升树$f_K(x)$.

过程：

(1). 初始化模型$f_0(x)=0$；

(2). 循环训练K个模型$k=1,2,\cdots,K$

(a). 计算残差：$r_{ki}=y^{(i)}-f_{k-1}(x^{(i)}),\quad i=1,2,\cdots,M$

(b). 拟合残差r_{ki}学习一个回归树，得到$T(x;\Theta_k)$

(c). 更新$f_k(x)=f_{k-1}(x)+T(x;\Theta_k)$

(3). 得到回归提升树

$f_K(x)=\sum_{k=1}^{K}T(x;\Theta_k)$

}

Gradient Boosting

提升树方法是利用加法模型与前向分布算法实现整个优化学习过程。Adaboost的指数损失和回归提升树的平方损失，在前向分布中的每一步都比较简单。但对于一般损失函数而言（比如绝对损失），每一个优化并不容易。

针对这一问题。Freidman提出了梯度提升（gradient boosting）算法。该算法思想：

利用损失函数的负梯度在当前模型的值作为回归问题提升树算法中残差的近似值，拟合一个回归树。

损失函数的负梯度为：

$$-\left[\frac{\partial L(y^{(i)},f(x^{(i)}))}{\partial f(x^{(i)})}\right]_{f(x)=f_{k-1}(x)}\approx r_{m,i}$$

- Gradient Boosting－算法描述

{

输入：训练数据集$D=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(M)},y^{(M)})\},x^{(i)}\in\mathcal{X}\subseteq R$

损失函数$L(y,f(x))$;

输出：提升树$\hat{f}\,(x)$.

过程：

(1).初始化模型

$$f_0(x)=\arg\min_c\sum_{i=1}^{M}L(y^{(i)},c)\;;$$

(2).循环训练K个模型$k=1,2,\cdots,K$

(a).计算残差：对于$i=1,2,\cdots,M$

$$r_{ki}=-\left[\frac{\partial L(y^{(i)},\,f(x^{(i)}))}{\partial f(x^{(i)})}\right]_{f(x)=f_{k-1}(x)}$$

(b).拟合残差r_{ki}学习一个回归树，得到第k颗树的叶结点区域R_{kj}，　$j=1,2$

(c).对$j=1,2,\cdots,J$,计算：

$$c_{kj}=\arg\min_c\sum_{x^{(i)}\in R_{kj}}L(y^{(i)},\;f_{k-1}(x^{(i)})+c)$$

(d).更新模型：

$$f_k(x)=f_{k-1}(x)+\sum_{j=1}^{J}c_{kj}I(x\in R_{kj})$$

(3).得到回归提升树

$$\hat{f}\,(x)=f_K(x)=\sum_{k=1}^{K}\sum_{j=1}^{J}c_{kj}I(x\in R_{kj})$$

}

> 算法解释：
>
> 1. 第（1）步初始化，估计使损失函数极小化的常数值（是一个只有根结点的树）；
> 2. 第(2)(a)步计算损失函数的负梯度在当前模型的值，将它作为残差的估计。(对于平方损失函数，他就是残差；对于一般损失函数，它就是残差的近似值)
> 3. 第(2)(b)步估计回归树的结点区域，以拟合残差的近似值；
> 4. 第(2)(c)步利用线性搜索估计叶结点区域的值，使损失函数极小化；
> 5. 第(2)(d)步更新回归树。

Boosting利器

Boosting类方法在不仅在二分类、多分类上有着出色的表现，在预估问题上依然出类拔萃。

> 2012年KDD cup竞赛和Kaggle上的许多数据挖掘竞赛，Boosting类方法帮助参赛者取得好成绩提供了强有力的支持。

“工欲善其事，必先利其器”。Github上和机器学习工具包（如sklearn）中有很多优秀的开源boosting实现。在这里重点介绍两个Boosting开源工具。

- XGBoost

  说到Boosting开源工具，首推@陈天奇怪同学的XGBoost (eXtreme Gradient Boosting)。上面说的各种竞赛很多优秀的战果都是用@陈天奇同学的神器。

  从名称可以看出，该版本侧重于Gradient Boosting方面，提供了Graident Boosting算法的框架，给出了GBDT，GBRT，GBM具体实现。提供了多语言接口（C++, Python, Java, R等），供大家方便使用。

  更令人振奋的一件事情是，最新版本的xgboost是基于分布式通信协议rabit开发的，可部署在分布式资源调度系统上（如yarn，s3等）。我们完全可以利用最新版的xgboost在分布式环境下解决分类、预估等场景问题。

  > 注：
  >
  > 1. XGBoost是DMLC（即分布式机器学习社区）下面的一个子项目，由@陈天奇怪，@李沐等机器学习大神发起。
  > 2. Rabit是一个为分布式机器学习提供Allreduce和Broadcast编程范式和容错功能的开源库（也是@陈天奇同学的又一神器）。它主要是解决MPI系统机器之间无容错功能的问题，并且主要针对Allreduce和Broadcast接口提供可容错功能。
  >
  > 题外话：
  >
  > 2014年第一次用XGBoost时，用于Kaggle移动CTR预估竞赛。印象比较深刻的是，同样的训练数据（特征工程后），分别用XGBoost中的GBDT和MLLib中的LR模型（LBFGS优化），在验证集上的表现前者比后者好很多（logloss和auc都是如此）。线上提交结果时，名次直接杀进前100名，当时给我留下了非常好的印象。后来，因为项目原因，没有过多的使用xgboost，但一直关注着。
  >
  > 目前，我个人更加关注的是：**基于Rabit开发/封装一些在工业界真正能发挥重要价值的（分布式）机器学习工具**，用于解决超大规模任务的学习问题。这里面会涉及到分布式环境下的编程范式，可以高效地在分布式环境下工作的优化算法（`admm`等）和模型(`loss + regularization term`)等。
  >
  > 关于大数据下的机器学习发展，个人更看好将**计算引擎模块**与**资源调度模块**独立开来，专注做各自的事情。计算引擎可以在任意的分布式资源调度系统上工作，实现真正的可插拔，是一个不错的方向。

> 与之观念对应的是，spark上集成的graphx和mllib中的许多计算模块，虽然使用起来很简便（几十行核心代码就能搭建一个学习任务的pipeline）。但可以想象的是，随着spark的进一步发展，该分布式计算平台会变的非常重，功能也会越来越多。离专注、专一和极致的解决某类问题越来越远，对每一类问题给出的解决方案并不会特别好。

- MultiBoost

  MultiBoost工具的侧重点不同于XGBoost，是Adaboost算法的多分类版本实现，更偏向于解决**multi-class / multi-label / multi-task的分类问题**。

  我们曾经基于该工具训练了用于**用户兴趣画像**的多标签（multi-label）分类模型，其分类效果（Precision / Recall作为指标）要比Naive Bayes好。

  MultiBoost是用C++实现的。值得一提的是，由我们组的算法大神和男神@BaiGang实现了MulitBoost的spark版本（Scala语言），详见Github: Spark_MultiBoost

---

- 更多信息请关注：计算广告与机器学习－CAML 技术共享平台

深入浅出机器学习

AdaBoost BGDT Boosting Gradient Boosting XGBoost

上一篇：

第09章：深入浅出ML之Factorization家族

下一篇：

第05章：深入浅出ML之Bayes-Based家族

分类

深入浅出机器学习 9

计算广告学 2

标签

计算广告学 2 Agent 2 参数服务器 2 Dynamic Programming 1 DP 1

Policy Evalation 1 Policy Improvement 1 Policy Iteration 1 Value Iteration 1

Greedy Policy 1 连续随机变量 1 MDP 1 Markov Decision Process 1 RL 1

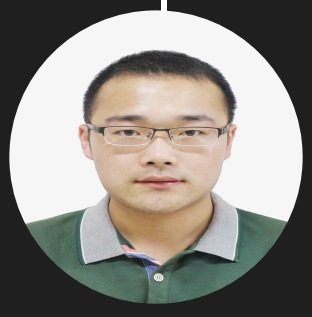
Environments 1 GD 1 FTRL 1 AdaGrad 1 AdaDelta 1 Adam 1

友情链接

码农圈

Jark's Blog

RSS 订阅



Hello, Welcome to CAML technology sharing platform.

I'm Zhou Yong, engaged in algorithms work on computational advertising and machine learning.