

# GBDT 梯度提升决策树

## 参考文献：

- python机器学习案例系列教程——集成学习（Bagging、Boosting、随机森林RF、AdaBoost、GBDT、xgboost）  
<https://blog.csdn.net/luanpeng825485697/article/details/79383492>
- python机器学习案例系列教程——GBDT算法、XGBOOST算法：  
<https://blog.csdn.net/luanpeng825485697/article/details/79766455>

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

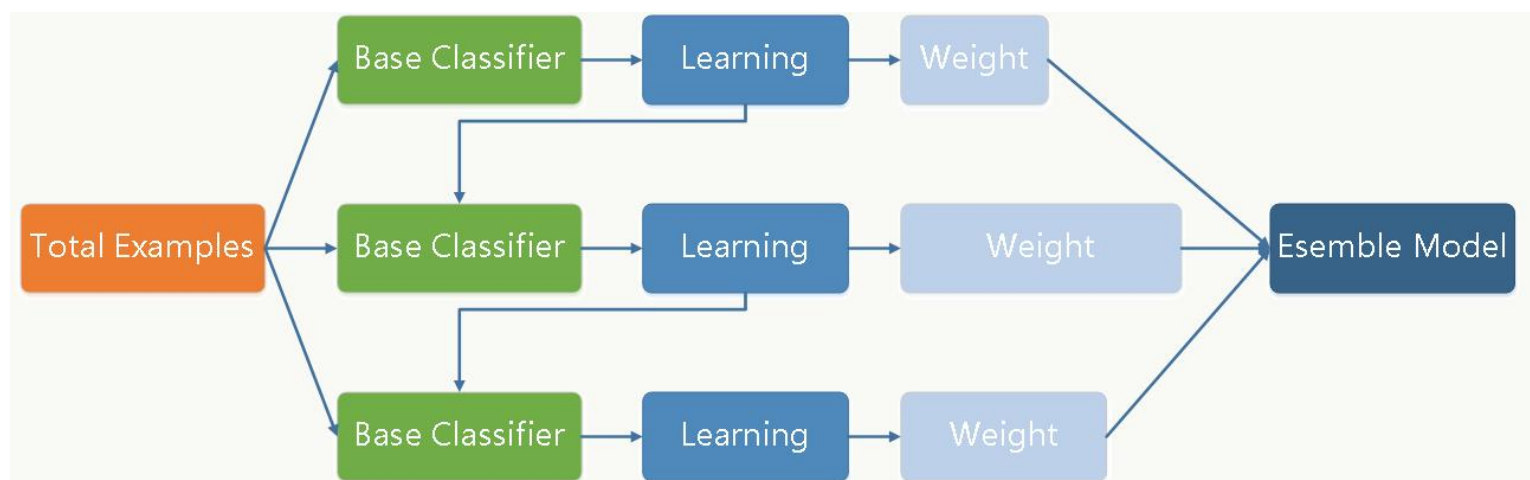
GBDT中的树是回归树（不是分类树），GBDT用来做回归预测，调整后也可以用于分类。

- 回归树可以参考 <http://blog.csdn.net/luanpeng825485697/article/details/78795504>

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。
- 但对于一般的损失函数，往往每一步优化没那么容易，如绝对值损失函数和Huber损失函数。针对这一问题，Freidman提出了梯度提升算法：利用最速下降的近似方法，即利用损失函数的负梯度在当前模型的值，作为回归问题中提升树算法的残差的近似值，拟合一个回归树。

所以说GBDT是通过采用加法模型（即基函数的线性组合），以及不断减小训练过程产生的残差来达到将数据分类或者回归的算法。



## 算法步骤解释：

- 初始化，估计使损失函数极小化的常数值，它是只有一个根节点的树，即 $\gamma$ 是一个常数值。
- 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
  - 估计回归树叶节点区域，以拟合残差的近似值
  - 利用线性搜索估计叶节点区域的值，使损失函数极小化
  - 更新回归树
- 得到输出的最终模型  $f(x)$

GBDT通过多轮迭代，每轮迭代产生一个弱分类器，每个分类器在上一轮分类器的残差基础上进行训练。对弱分类器的要求一般是足够简单，并且是低方差和高偏差的。因为训练的过程是通过降低偏差来不断提高最终分类器的精度。

弱分类器一般会选择为 **CART 树**（也就是分类回归树）。由于上述高偏差和简单的要求 每个分类回归树的深度不会很深。最终的总分类器 是将每轮训练得到的弱分类器加权求和得到的（也就是加法模型）。

## 总结：


在GBDT的迭代中，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CART回归树模型的弱学习器 $h_t(x)$ ，让本轮的损失损失 $L(y, f_t(x)) = L(y, f_{t-1}(x) + h_t(x))$ 最小。也就是说，本轮迭代找到决策树，要让样本的损失尽量变得更小。

GBDT的思想可以用一个通俗的例子解释，假如有个人30岁，我们首先用20岁去拟合，发现损失有10岁，这时我们用6岁去拟合剩下的损失，发现差距还有4岁，第三轮我们用3岁拟合剩下的差距，差距就只有一岁了。如果我们的迭代轮数还没有完，可以继续迭代下面，每一轮迭代，拟合的岁数误差都会减小。

从上面的例子看这个思想还是蛮简单的，但是有个问题是这个损失的拟合不好度量，损失函数各种各样，怎么找到一种通用的拟合方法呢？

## 2. GBDT的负梯度拟合

在上一节中，我们介绍了GBDT的基本思路，但是没有解决损失函数拟合方法的问题。针对这个问题，大牛Freidman提出了用损失函数的负梯度来拟合本轮损失的近似值，进而拟合一个CART回归树。第 $t$ 轮的第 $i$ 个样本的损失函数的负梯度表示为

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


利用 $(x_i, r_{ti}) (i = 1, 2, \dots, m)$ ，我们可以拟合一颗CART回归树，得到了第 $t$ 颗回归树，其对应的叶节点区域 $R_{tj}, j = 1, 2, \dots, J$ 。其中 $J$ 为叶子节点的个数。

针对每一个叶子节点里的样本，我们求出使损失函数最小，也就是拟合叶子节点最好的的输出值 $c_{tj}$ 如下：

$$c_{tj} = \underbrace{\operatorname{argmin}}_c \sum_{x_i \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

这样我们就得到了本轮的决策树拟合函数如下：

$$h_t(x) = \sum_{j=1}^J c_{tj} I(x \in R_{tj})$$

从而本轮最终得到的强学习器的表达式如下：

$$f_t(x) = f_{t-1}(x) + \sum_{j=1}^J c_{tj} I(x \in R_{tj})$$

### 3. GBDT回归算法

好了，有了上面的思路，下面我们总结下GBDT的回归算法。为什么没有加上分类算法一起？那是因为分类算法的输出是不连续的类别值，需要一些处理才能使用负梯度，我们后面讲。

输入是训练集样本  $T = (x_1, y_1), (x_2, y_2), \dots (x_m, y_m)$ ，最大迭代次数  $T$ ，损失函数  $L$ 。

输出是强学习器  $f(x)$

#### 1. 初始化弱学习器

$$f_0(x) = \underbrace{\operatorname{argmin}}_c \sum_{i=1}^m L(y_i, c)$$

#### 2. 对迭代轮数 $t = 1, 2, \dots T$ 有：

a) 对样本  $i = 1, 2, \dots m$ ，计算负梯度

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

b) 利用  $(x_i, r_{ti}) (i = 1, 2, \dots m)$ ，拟合一颗CART回归树，得到第  $t$  颗回归树，其对应的叶子节点区域为  $R_{tj}, j = 1, 2, \dots, J$ 。其中  $J$  为回归树的叶子节点的个数。

c) 对叶子区域  $j = 1, 2, \dots J$ ，计算最佳拟合值

$$c_{tj} = \underbrace{\operatorname{argmin}}_c \sum_{x_i \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

d) 更新强学习器

$$f_t(x) = f_{t-1}(x) + \sum_{j=1}^J c_{tj} I(x \in R_{tj})$$

#### 3. 得到强学习器 $f(x)$ 的表达式

$$f(x) = f_T(x) = f_0(x) + \sum_{t=1}^T \sum_{j=1}^J c_{tj} I(x \in R_{tj})$$



## 4. GBDT分类算法

这里我们再看看GBDT分类算法，GBDT的分类算法从思想上和GBDT的回归算法没有区别，但是由于样本输出不是连续的值，而是离散的类别，导致我们无法直接从输出类别去拟合类别输出的误差。

为了解决这个问题，主要有两个方法：

- 一个是用指数损失函数，此时 GBDT 退化为 Adaboost 算法。
- 另一种方法是用类似于逻辑回归的对数似然损失函数的方法。也就是说，我们用的是类别的预测概率值和真实概率值的差来拟合损失。

本文仅讨论用对数似然损失函数的GBDT分类。而对于对数似然损失函数，我们又有二元分类和多元分类的区别。

### 4.1 二元GBDT分类算法

对于二元GBDT，如果用类似于逻辑回归的对数似然损失函数，则损失函数为：

$$L(y, f(x)) = \log(1 + \exp(-yf(x)))$$

其中 $y \in \{-1, +1\}$ 。则此时的负梯度误差为

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

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为

$$c_{tj} = \underbrace{\operatorname{argmin}}_c \sum_{x_i \in R_{tj}} \log(1 + \exp(-y_i(f_{t-1}(x_i) + c)))$$

由于上式比较难优化，我们一般使用近似值代替

$$c_{tj} = \sum_{x_i \in R_{tj}} r_{ti} / \sum_{x_i \in R_{tj}} |r_{ti}|(1 - |r_{ti}|)$$

除了负梯度计算和叶子节点的最佳残差拟合的线性搜索，二元GBDT分类和GBDT回归算法过程相同。

## 4.2 多元GBDT分类算法

多元GBDT要比二元GBDT复杂一些，对应的是多元逻辑回归和二元逻辑回归的复杂度差别。假设类别数为K，则此时我们的对数似然损失函数为：

$$L(y, f(x)) = - \sum_{k=1}^K y_k \log p_k(x)$$

其中如果样本输出类别为k，则 $y_k = 1$ 。第k类的概率 $p_k(x)$ 的表达式为：

$$p_k(x) = \exp(f_k(x)) / \sum_{l=1}^K \exp(f_l(x))$$

通过上式我们可以看出，我们是将类别进行one-hot编码，每个输出都要建一颗决策树，一个样本通过K个决策树，得到K个输出，在通过softmax函数，获得K个概率。

集合上两式，我们可以计算出第t轮的第i个样本对应类别l的负梯度误差为

$$r_{til} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f_k(x)=f_{l,t-1}(x)} = y_{il} - p_{l,t-1}(x_i)$$

观察上式可以看出，其实这里的误差就是样本i对应类别l的真实概率和t-1轮预测概率的差值。

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为

$$c_{tjl} = \underbrace{\operatorname{argmin}}_{c_{jl}} \sum_{i=0}^m \sum_{k=1}^K L(y_k, f_{t-1,l}(x) + \sum_{j=0}^J c_{jl} I(x_i \in R_{tj}))$$

由于上式比较难优化，我们一般使用近似值代替

$$c_{tjl} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{tjl}} r_{til}}{\sum_{x_i \in R_{tjl}} |r_{til}| (1 - |r_{til}|)}$$

除了负梯度计算和叶子节点的最佳残差拟合的线性搜索，多元GBDT分类和二元GBDT分类以及GBDT回归算法过程相同。

## 5. GBDT常用损失函数

---

这里我们再对常用的GBDT损失函数做一个总结。

**对于分类算法**，其损失函数一般有对数损失函数和指数损失函数两种：

a) 如果是指数损失函数，则损失函数表达式为

$$L(y, f(x)) = \exp(-yf(x))$$

其负梯度计算和叶子节点的最佳残差拟合参见Adaboost原理篇。

b) 如果是对数损失函数，也就是前面说的二元分类和多元分类两种

**对于回归算法**，常用损失函数有如下4种：

a) 均方差，这个是最常见的回归损失函数了

$$L(y, f(x)) = (y - f(x))^2$$

b) 绝对损失，这个损失函数也很常见

$$L(y, f(x)) = |y - f(x)|$$

对应负梯度误差为：

$$\text{sign}(y_i - f(x_i))$$

c) Huber损失，它是均方差和绝对损失的折衷产物，对于远离中心的异常点，采用绝对损失，而中心附近的点采用均方差。这个界限一般用分位数点度量。损失函数如下：

$$L(y, f(x)) = \begin{cases} 0.5(y-f(x))^2, & |y-f(x)| \leq \delta \\ \delta(|y-f(x)|-0.5\delta), & |y-f(x)| > \delta \end{cases}$$

对应的负梯度误差为：

$$r(y_i, f(x_i)) = \begin{cases} y_i - f(x_i), & |y_i - f(x_i)| \leq \delta \\ \delta \text{sign}(y_i - f(x_i)), & |y_i - f(x_i)| > \delta \end{cases}$$

d) 分位数损失。它对应的是分位数回归的损失函数，表达式为

$$L(y, f(x)) = \sum_{y \geq f(x)} \theta |y - f(x)| + \sum_{y < f(x)} (1 - \theta) |y - f(x)|$$

其中 $\theta$ 为分位数，需要我们在回归前指定。对应的负梯度误差为：

$$r(y_i, f(x_i)) = \begin{cases} \theta, & y_i \geq f(x_i) \\ \theta - 1, & y_i < f(x_i) \end{cases}$$

对于Huber损失和分位数损失，主要用于健壮回归，也就是减少异常点对损失函数的影响。



## 6. GBDT的正则化

和Adaboost一样，我们也需要对GBDT进行正则化，防止过拟合。GBDT的正则化主要有三种方式。

第一种是和Adaboost类似的正则化项，即**步长(learning rate)**。定义为 $v$ ，对于前面的弱学习器的迭代

$$f_k(x) = f_{k-1}(x) + h_k(x)$$

如果我们加上了正则化项，则有

$$f_k(x) = f_{k-1}(x) + v h_k(x)$$

$v$ 的取值范围为 $0 < v \leq 1$ 。对于同样的训练集学习效果，较小的 $v$ 意味着我们需要更多的弱学习器的迭代次数。通常我们用步长和迭代最大次数一起来决定算法的拟合效果。

第二种正则化的方式是通过**子采样比例 (subsample)**。取值为 $(0, 1]$ 。注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在 $[0.5, 0.8]$ 之间。

使用了子采样的GBDT有时也称作随机梯度提升树(Stochastic Gradient Boosting Tree, SGBT)。由于使用了子采样，程序可以通过采样分发到不同的任务去做boosting的迭代过程，最后形成新树，从而减少弱学习器难以并行学习的弱点。

第三种是对于弱学习器即CART回归树进行**正则化剪枝**。在决策树原理篇里我们已经讲过，这里就不重复了。



## 7. GBDT小结

---

GBDT终于讲完了，GBDT本身并不复杂，不过要吃透的话需要对集成学习的原理，决策树原理和各种损失函数有一定的了解。由于GBDT的卓越性能，只要是研究 **机器学习** 都应该掌握这个算法，包括背后的原理和应用调参方法。目前GBDT的算法比较好的库是xgboost。当然scikit-learn也可以。

最后总结下GBDT的优缺点。

GBDT主要的优点有：

- 1) 可以灵活处理各种类型的数据，包括连续值和离散值。GBDT使用的是cart树模型，可以处理连续值和离散值特征。对于连续值节点划分时，按照大于小于分割点，对于离散值，按照等于不等于划分分割点。
- 2) 在相对少的调参时间情况下，预测的准备率也可以比较高。这个是相对SVM来说的。
- 3) 使用一些健壮的损失函数，对异常值的鲁棒性非常强。比如 Huber损失函数和Quantile损失函数。

GBDT的主要缺点有：

- 1) 由于弱学习器之间存在依赖关系，难以并行训练数据。不过可以通过自采样的SGBT来达到部分并行。
- 2) 对于稀疏矩阵，容易过拟合，比 lr 和 fm 效果要差，是因为若数据中存在假象数据（例如特征f1为1的样本，输出类别都是1），树模型没法进行过拟合处理。而 lr 或 fm 可以通过正则化，压缩  $w_1$  的值。

## 参考文献:

1. GBDT--原来是这么回事(附代码) <https://www.cnblogs.com/mantch/p/11160496.html>

## GBDT--原来是这么回事(附代码)

### 1. 解释一下GBDT算法的过程

GBDT(Gradient Boosting Decision Tree), 全名叫梯度提升决策树, 使用的是**Boosting**的思想。

#### 1.1 Boosting思想

Boosting方法训练基分类器时采用串行的方式, 各个基分类器之间有依赖。它的基本思路是将基分类器层层叠加, 每一层在训练的时候, 对前一层基分类器分错的样本, 给予更高的权重。测试时, 根据各层分类器的结果的加权得到最终结果。

Bagging与Boosting的串行训练方式不同, Bagging方法在训练过程中, 各基分类器之间无强依赖, 可以进行并行训练。

#### 1.2 GBDT原来是这么回事

GBDT的原理很简单, 就是所有弱分类器的结果相加等于预测值, 然后下一个弱分类器去拟合误差函数对预测值的残差(这个残差就是预测值与真实值之间的误差)。当然了, 它里面的弱分类器的表现形式就是各棵树。

举一个非常简单的例子, 比如我今年30岁了, 但计算机或者模型GBDT并不知道我今年多少岁, 那GBDT咋办呢?

- 它会在第一个弱分类器 (或第一棵树中) 随便用一个年龄比如20岁来拟合, 然后发现误差有10岁;
- 接下来在第二棵树中, 用6岁去拟合剩下的损失, 发现差距还有4岁;
- 接着在第三棵树中用3岁拟合剩下的差距, 发现差距只有1岁了;
- 最后在第四棵树中用1岁拟合剩下的残差, 完美。
- 最终, 四棵树的结论加起来, 就是真实年龄30岁 (实际工程中, gbdt是计算负梯度, 用负梯度近似残差)。

### 为何gbdt可以用负梯度近似残差呢?

回归任务下, GBDT 在每一轮的迭代时对每个样本都会有一个预测值, 此时的损失函数为均方差损失函数,

$$l(y_i, y^i) = \frac{1}{2}(y_i - y^i)^2$$

那此时的负梯度是这样计算的

$$-\left[\frac{\partial l(y_i, y^i)}{\partial y^i}\right] = (y_i - y^i)$$

所以, 当损失函数选用均方损失函数是时, 每一次拟合的值就是 (真实值 - 当前模型预测的值), 即残差。此时的变量是 $y^i$ , 即 “当前预测模型的值”, 也就是对它求负梯度。

### 2. 梯度提升和梯度下降的区别和联系是什么?

下表是梯度提升算法和梯度下降算法的对比情况。可以发现, 两者都是在每一轮迭代中, 利用损失函数相对于模型的负梯度方向的信息来对当前模型进行更新, 只不过在梯度下降中, 模型是以参数化形式表示, 从而模型的更新等价于参数的更新。而在梯度提升中, 模型并不需要进行参数化表示, 而是直接定义在函数空间中, 从而大大扩展了可以使用的模型种类。

梯度提升	函数空间 $F$	$F = F_{t-1} - \rho_t \nabla_F L _{F=F_{t-1}}$	$L = \sum_i l(y_i, F(x_i))$
梯度下降	参数空间 $W$	$w_t = w_{t-1} - \rho_t \nabla_w L _{w=w_{t-1}}$	$L = \sum_i l(y_i, f_w(w_i))$

## 总结:

### 3. GBDT的优点和局限性有哪些?

#### 3.1 优点

1. 预测阶段的计算速度快，树与树之间可并行化计算。~~✗~~
2. 在分布稠密的数据集上，泛化能力和表达能力都很好，这使得GBDT在Kaggle的众多竞赛中，经常名列榜首。
3. 采用决策树作为弱分类器使得GBDT模型具有较好的解释性和鲁棒性，能够自动发现特征间的高阶关系，并且也不需要数据特殊的预处理如归一化等。

#### 3.2 局限性

1. GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络。
2. GBDT在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显。
3. 训练过程需要串行训练，只能在决策树内部采用一些局部并行的手段提高训练速度。

### 4. RF(随机森林)与GBDT之间的区别与联系

相同点:

都是由多棵树组成，最终的结果都是由多棵树一起决定。

不同点:

- 组成随机森林的树可以分类树也可以是回归树，~~而GBDT只由回归树组成~~
- 组成随机森林的树可以并行生成，而GBDT是串行生成
- 随机森林的结果是多数表决表决的，而GBDT则是多棵树累加之和
- 随机森林对异常值不敏感，而GBDT对异常值比较敏感
- 随机森林是减少模型的方差，而GBDT是减少模型的偏差
- 随机森林不需要进行特征归一化。而GBDT则需要进行特征归一化

### 5. 代码实现

GitHub: [https://github.com/NLP-LOVE/ML-NLP/blob/master/Machine%20Learning/3.2%20GBDT/GBDT\\_demo.ipynb](https://github.com/NLP-LOVE/ML-NLP/blob/master/Machine%20Learning/3.2%20GBDT/GBDT_demo.ipynb)