# 再谈GBDT算法

在写这篇文章的时候，GBDT正在集团内广受关注，因为我们搞了个编码竞赛，以提高GBDT预测模块的性能。现在竞赛才刚刚开始，相信过段时间关注它的人会更多。说起GBDT，还和公司的发展历史有关系。最早我们公司和美国雅虎开始合作的时候，用到的算法包是美国雅虎开发的，其中就包含了GBDT算法，这个算法包在美国雅虎内部也是广泛使用的，后来公司内部开始开发自己的算法包，GBDT仍然是其中一个重要的算法。写到这里，估计会有严谨的同学质疑“真的是自己开发的？是不是就是把雅虎的代码抄了一遍”？诚然，早期肯定是认真参考过雅虎的代码，站在巨人的肩膀上永远没错。但还真的不是抄袭代码，因为随着学术界的研究进展，这个算法本身也在改进，如果不是自己开发，很难把最新的技术用进来，另外公司自己开发也能大大降低公司内部同事对机器学习算法使用的门槛。现在我们公司开发的算法包里的GBDT早就超过了原雅虎算法包里的实现。

在这篇文章里，我会讲解该算法的大致原理，讲GBDT不借助公式很难说明白，因此还是摆了几道公式，但力求简单易懂，不追求数学上的严谨。ATA上还有一篇元涵写的讲解GBDT的文章，侧重在Decision Tree和如何使用那个算法包，我这篇可能更偏原理一些。另外，公达在百家讲坛做过GBDT的报告，也非常值得参考。

## Boosting

GBDT的全称是Gradient Boosting Decision Tree，Gradient Boosting 和 Decision Tree是两个独立的概念。因此我们先说说Boosting。Boosting的概念很好理解，意思是用一些弱分类器的组合来构造一个强分类器。因此，它不是某个具体的算法，它说的是一种理念。和这个理念相对应的是一次性构造一个强分类器。像支持向量机，逻辑回归等都属于后者。通常，我们通过相加来组合分类器，形式如下：

其中x表示输入的特征，表示最终得到的分类器， 分别表示m个弱分类器，是一个常数， 分别表示每个弱分类器对应的权重。

## Gradient Boosting Modeling (GBM)

给定一个问题，我们如何构造这些弱分类器呢？Gradient Boosting Modeling (GBM) 就是构造这些弱分类的一种方法。同样，它指的不是某个具体的算法，仍然只是一个理念。在理解Gradient Boosting Modeling之前，我们先看看一个典型的优化问题：

针对这种优化问题，有一个经典的算法叫Steepest Gradient Descent，也就是最深梯度下降法。这个算法的过程大致如下：

1. 给点一个起始点
2. 对 分别做如下迭代：
   1. , 这里 表示 在点的梯度
3. 直到

以上迭代过程可以这么理解：整个寻优的过程就是个小步快跑的过程，每跑一小步，都往函数当前下降最快的那个方向走一点。

这样寻优得到的结果可以表示成加和形式，即

这个形式和以上Eq. (1) 中的是不是非常相似？Gradient Boosting正是由此启发而来。构造本身也是一个寻优的过程，只不过我们寻找的不是一个最优点，而是一个最优的函数。优化的目标通常都是通过一个损失函数来定义，即

其中表示损失函数在第个样本上的损失值，分别表示第i个样本的特征和目标值。常见的损失函数如平方差函数

类似最深梯度下降法，我们可以通过梯度下降法来构造弱分类器，只不过每次迭代时，令

即对损失函数。

这里有个小问题，一个函数对函数的求导不好理解，而且通常都无法通过上述公式直接求解到梯度函数。为此，采取一个近似的方法，把函数理解成在所有样本上的离散的函数值，即：

不难理解，这是一个N维向量，然后计算

这是一个函数对向量的求导，得到的也是一个梯度向量。注意，这里求导时的变量还是函数，不是样本。

严格来说 只是描述了在某些个别点上的值，并不足以表达，但我们可以通过函数拟合的方法从，这样我们就通过近似的方法得到了函数对函数的梯度求导。

因此GBM的过程可以总结为如下：

1. 选择一个起始常量函数
2. 对 分别做如下迭代：
   1. 计算离散梯度值
   2. 对 做函数拟合得到
   3. 通过line seach得到
3. 直到

常量函数，即

## Gradient Boosting Decision Tree

以上Gradient Boosting Modeling的过程中，还没有说清楚如何通过离散值 构造拟合函数。函数拟合是个比较熟知的概念，有很多现成的方法，不过有一种拟合方法广为应用，那就是决策树Decision Tree，有关决策树的概念，元涵的另外一篇里说的比较详细，这里不再多说。我以为，理解GBDT重点首先是Gradient Boosting，其次才是Decision Tree。GBDT是Gradient Boosting的一种具体实例，只不过这里的弱分类器是决策树。如果你改用其他弱分类器XYZ，你也可以称之为Gradient Boosting XYZ。只不过Decision Tree很好用，GBDT才如此引人注目。

## 损失函数

谈到GBDT，常听到一种简单的描述方式：“先构造一个（决策）树，然后不断在已有模型和实际样本输出的残差上再构造一颗树，依次迭代”。其实这个说法不全面，它只是GBDT的一种特殊情况，为了看清这个问题，需要对损失函数的选择做一些解释。

从对GBM的描述里可以看到Gradient Boosting过程和具体用什么样的弱分类器是完全独立的，可以任意组合，因此这里不再刻意强调用决策树来构造弱分类器，转而我们来仔细看看弱分类器拟合的目标值，即梯度，之前我们已经提到过

因此很重要，拿平方差损失函数为例，可以得到

忽略倍数2，这正好是当前已经构造好的函数 在样本上和目标值之间的差值。

如果我们换一个损失函数，比如绝对差值

这个损失函数的梯度是个符号函数

由此可以看到，只有当损失函数为平方差函数时，才能说GBDT是通过拟合残差来构造弱分类器的。

## GBDT和AdaBoost

Boosting是一类机器学习算法，在这个家族中还有一种非常著名的算法叫AdaBoost，是Adaptive Boosting的简称，AdaBoost在人脸检测问题上尤其出名。既然也是Boosting，可以想象它的构造过程也是通过多个弱分类器来构造一个强分类器。那AdaBoost和GBDT有什么区别呢？

两者最大的区别在于，AdaBoost不属于Gradient Boosting，即它在构造弱分类器时并没有利用到梯度下降法的思想，而是用的Forward Stagewise Additive Modeling (FSAM)。为了理解FSAM，在回过头来看看Eq.(2)中的优化问题。从可以表达成一个迭代的形式，

严格来说Eq. (2)描述的优化问题要求我们同时找出，这个问题很难。为此我们把问题简化为分阶段优化，每个阶段找出一个合适的和。假设我们已经得到前m-1个弱分类器，即，下一步在保证不变的前提下，寻找合适的。按照损失函数的定义，我们可以得到

如果Loss是平方差函数，则我们有

这里就是当前模型在数据上的残差，可以看出，求解合适的就是在这当前的残差上拟合一个弱分类器，且损失函数还是平方差函数。这和GBDT选择平方差损失函数时构造弱分类器的方法恰好一致。

如果Loss是个指数形式，即：

则

其中 =(和要求解的，可以当成样本的权重，因此在这种情况下，构造弱分类器就是在对样本设置权重后的数据上拟合，且损失函数还是指数形式。

这个就是AdaBoost，不过AdaBoost最早并不是按这个思路推出来的，相反，是在AdaBoost提出5年后，人们才开始用Forward Stagewise Additive Modeling来解释AdaBoost背后的原理。

为什么要把平方差和指数形式Loss函数单独拿出来说呢？这是因为对这两个损失函数来说，按照Forward Stagewise Additive Modeling的思路构造弱分类器时比较方便。如果是平方差损失函数，就在残差上做平方差拟合构造弱分类器； 如果是指数形式的损失函数，就在带权重的样本上构造弱分类器。但损失函数如果不是这两种，问题就没那么简单，比如绝对差值函数，虽然构造弱分类器也可以表示成在残差上做绝对差值拟合，但这个子问题本身也不容易解，因为我们是要构造多个弱分类器的，所以我们当然希望构造弱分类器这个子问题比较好解。因此 FSAM思路无法推广到其他一些实用的损失函数上。相比而言，Gradient Boosting Modeling (GBM) 有什么优势呢？GBM每次迭代时，只需要计算当前的梯度，并在平方差损失函数的基础上拟合梯度。虽然梯度的计算依赖原始问题的损失函数形式，但这不是问题，只要损失函数是连续可微的，梯度就可以计算。至于拟合梯度这个子问题，我们总是可以选择平方差函数作为这个子问题的损失函数，因为这个子问题是一个独立的回归问题。

因此 FSAM和GBM得到的模型虽然从形式上是一样的，都是若干弱模型相加，但是他们求解弱分类器的思路和方法有很大的差别。只有当选择平方差函数为损失函数时，这两种方法等同。

## 为何GBDT受人青睐

以上比较了GBM和FSAM，可以看到GBM在损失函数的选择上有更大的灵活性，但这不足以解释GBDT的全部优势。GBDT是拿Decision Tree作为GBM里的弱分类器，GBDT的优势首先得益于Decision Tree本身的一些良好特性，具体可以列举如下：

1. Decision Tree可以很好的处理missing feature，这是他的天然特性，因为决策树的每个节点只依赖一个feature，如果某个feature不存在，这颗树依然可以拿来做决策，只是少一些路径。像逻辑回归，SVM就没这个好处。
2. Decision Tree可以很好的处理各种类型的feature，也是天然特性，很好理解，同样逻辑回归和SVM没这样的天然特性。
3. 对特征空间的outlier有鲁棒性，因为每个节点都是 的形式，至于大多少，小多少没有区别，outlier不会有什么大的影响，同样逻辑回归和SVM没有这样的天然特性。
4. 如果有不相关的feature，没什么干扰，如果数据中有不相关的feature，顶多这个feature不出现在树的节点里。逻辑回归和SVM没有这样的天然特性（但是有相应的补救措施，比如逻辑回归里的L1正则化）。
5. 数据规模影响不大，因为我们对弱分类器的要求不高，作为弱分类器的决策树的深度一般设的比较小，即使是大数据量，也可以方便处理。像SVM这种数据规模大的时候训练会比较麻烦。

当然Decision Tree也不是毫无缺陷，通常在给定的不带噪音的问题上，他能达到的最佳分类效果还是不如SVM，逻辑回归之类的。但是，我们实际面对的问题中，往往有很大的噪音，使得Decision Tree这个弱势就不那么明显了。而且，GBDT通过不断的叠加组合多个小的Decision Tree，他在不带噪音的问题上也能达到很好的分类效果。换句话说，通过GBDT训练组合多个小的Decision Tree往往要比一次性训练一个很大的Decision Tree的效果好很多。因此不能把GBDT理解为一颗大的决策树，几颗小树经过叠加后就不再是颗大树了，它比一颗大树更强。