w28971023的专栏

永远不要称自己是程序员！

转 GBDT（MART）迭代决策树入门教程 | 简介

2012-11-29 19:12 30674人阅读 评论(6) 收藏 举报

分类： 算法学习（15）

在网上看到一篇对从代码层面理解gbdt比较好的文章，转载记录一下：

GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree)，是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最终答案。它在被提出之初就和SVM一起被认为是泛化能力（generalization)较强的算法。近些年更因为被用于搜索排序的机器学习模型而引起大家关注。

后记：发现GBDT除了我描述的残差版本外还有另一种GBDT描述，两者大概相同，但求解方法（Gradient应用）不同。其区别和另一版本的介绍链接见这里。由于另一版本介绍博客中亦有不少错误，建议大家还是先看本篇，再跳到另一版本描述，这个顺序当能两版本都看懂。

第**1~4**节：**GBDT**算法内部究竟是如何工作的？
第**5**节：它可以用于解决哪些问题？
第**6**节：它又是怎样应用于搜索排序的呢？

在此先给出我比较推荐的两篇英文文献，喜欢英文原版的同学可直接阅读：
【1】Boosting Decision Tree入门教程 http://www.schonlau.net/publication/05stata_boosting.pdf
【2】LambdaMART用于搜索排序入门教程 http://research.microsoft.com/pubs/132652/MSR-TR-2010-82.pdf

GBDT主要由三个概念组成：Regression Decistion Tree（即DT)，Gradient Boosting（即GB)，Shrinkage (算法的一个重要演进分枝，目前大部分源码都按该版本实现）。搞定这三个概念后就能明白GBDT是如何工作的，要继续理解它如何用于搜索排序则需要额外理解RankNet概念，之后便功德圆满。下文将逐个碎片介绍，最终把整张图拼出来。

一、 DT：回归树 Regression Decision Tree

提起决策树（DT, Decision Tree) 绝大部分人首先想到的就是C4.5分类决策树。但如果一开始就把GBDT中的树想成分类树，那就是一条歪路走到黑，一路各种坑，最终摔得都要咯血了还是一头雾水说的就是LZ自己啊有木有。咳嗯，所以说千万不要以为GBDT是很多棵分类树。决策树分为两大类，回归树和分类树。前者用于预测实数值，如明天的温度、用户的年龄、网页的相关程度；后者用于分类标签值，如晴天/阴天/雾/雨、用户性别、网页是否是垃圾页面。这里要强调的是，前者的结果加减是有意义的，如10岁+5岁-3岁=12岁，后者则无意义，如男+男+女=到底是男是女？ GBDT的核心在于累加所有树的结果作为最终结果，就像前面对年龄的累加（-3是加负3），而分类树的结果显然是没办法累加的，所以**GBDT**中的树都是回归树，不是分类树，这点对理解GBDT相当重要（尽管GBDT调整后也可用于分类但不代表GBDT的树是分类树）。那么回归树是如何工作的呢？

下面我们以对人的性别判别/年龄预测为例来说明，每个instance都是一个我们已知性别/年龄的人，而feature则包括这个人上网的时长、上网的时段、网购所花的金额等。

作为对比，先说分类树，我们知道C4.5分类树在每次分枝时，是穷举每一个feature的每一个阈值，找到使得按照feature<=阈值，和feature>阈值分成的两个分枝的熵最大的feature和阈值（熵最大的概念可理解成尽可能每个分枝的男女比例都远离1:1），按照该标准分枝得到两个新节点，用同样方法继续分枝直到所有人都被分入性别唯一的叶子节点，或达到预设的终止条件，若最终叶子节点中的性别不唯一，则以多数人的性别作为该叶子节点的性别。

回归树总体流程也是类似，不过在每个节点（不一定是叶子节点）都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个feature的每个阈值找最好的分割点，但衡量最好的

个人资料

帛逸TB

+ 加关注 发私信

访问： 103343次
积分： 1279
等级： BLOG 4
排名： 第19520名

原创： 26篇
转载： 29篇
译文： 0篇
评论： 20条

让编辑状态下的UITable\ (6567)
objective-c delegate (5549)
ios中的代理与回调函数 (5439)
Perl SIG信号处理 (4590)
scrollView实现无限快速 (3969)
GBDT理解二三事 (2878)
Gradient Boost 算法流程 (2229)
谈谈分类算法的选择 (2150)

评论排行
GBDT（MART） 迭代决 (6)
GBDT理解二三事 (4)
objective-c delegate (2)
scrollView实现无限快速 (2)
GBDT源码剖析 (2)
String常用用法总结 (1)
谈谈分类算法的选择 (1)
pid match算法思想 (1)
（EM算法）The EM Alg (1)
Cookie、 Session、 Ca (0)

推荐文章
*Spark入门到精通：第九节 Spark SQL运行流程解析
*架构设计：系统间通信（17）——服务治理与Dubbo 中篇（分析）
*你的计划为什么执行不下去？怎么破？
*图解堆算法、链表、栈与队列（多图预警）
*Universal-Image-Loader完全解析(一)之介绍与使用详解
*Android平台Camera实时滤镜实现方法探讨(九)--磨皮算法探讨(一)

最新评论
GBDT（MART） 迭代决策树入门 atomlion: 非常感谢哦！我在实践中发现GBDT的预测效果非常好，在数据质量较差、复杂度高的大样本集中，几乎是效果...
GBDT（MART） 迭代决策树入门 大本_daben: @liufeng_cp:那到底怎么去理解boost呢？它的英文解释就是"推进、提升"的意思，我可以理...
GBDT理解二三事 march_on: 请问"损失函数可以定义为负的log似然",这里损失函数为什么是负的log似然，这里有什么推导吗
GBDT理解二三事 qq_18247987: 你好，我想请问一下，文章末尾，说在节点分裂的时候，使用直方图采样去，优化效率，不用遍历所有特征值，想...
GBDT（MART） 迭代决策树入门 keepreder: logistic regression能用于非线性回归
谈谈分类算法的选择 easonlv: 图片标签都不能显示，麻烦楼主调整一下，谢谢
GBDT理解二三事 帛逸TB: @yangxudong:不行
GBDT理解二三事 yangxudong: GBDT能不能自动组合特征？

标准不再是最大熵，而是最小化均方差--即（每个人的年龄-预测年龄）^2 的总和 / N，或者说是每个人的预测误差平方和 除以 N。这很好理解，被预测出错的人数越多，错的越离谱，均方差就越大，通过最小化均方差能够找到最靠谱的分枝依据。分枝直到每个叶子节点上人的年龄都唯一（这太难了）或者达到预设的终止条件（如叶子个数上限），若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。若还不明白可以Google "Regression Tree"，或阅读本文的第一篇论文中Regression Tree部分。

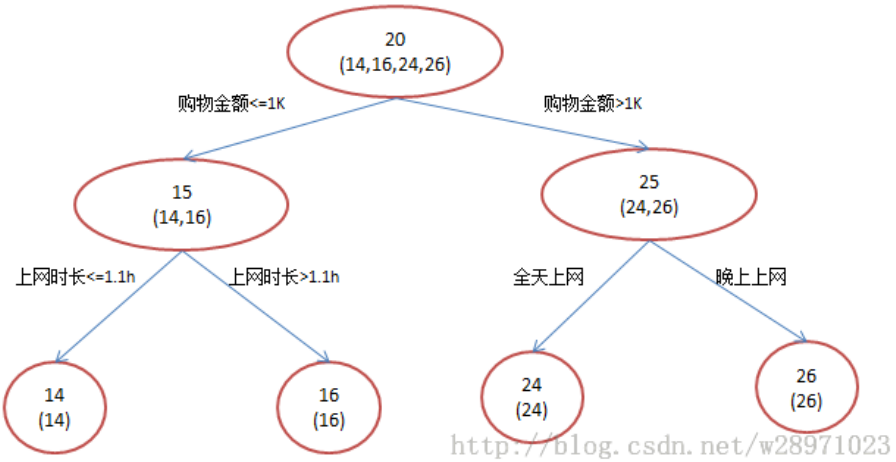
二、 GB：梯度迭代 Gradient Boosting

好吧，我起了一个很大的标题，但事实上我并不想多讲Gradient Boosting的原理，因为不明白原理并无碍于理解GBDT中的Gradient Boosting。喜欢打破砂锅问到底的同学可以阅读这篇英文wikihttp://en.wikipedia.org/wiki/Gradient_boosted_trees#Gradient_tree_boosting

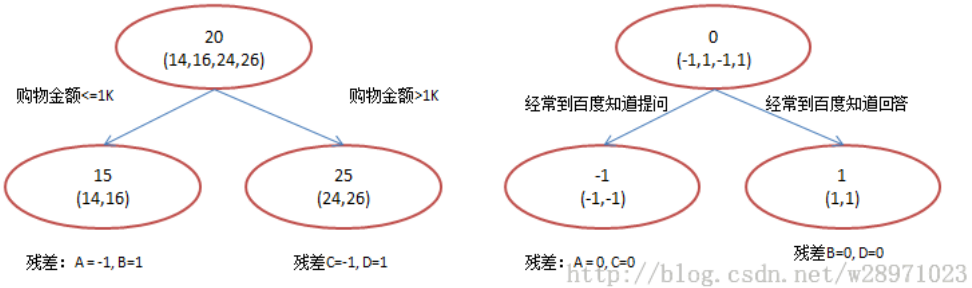
Boosting，迭代，即通过迭代多棵树来共同决策。这怎么实现呢？难道是每棵树独立训练一遍，比如A这个人，第一棵树认为是10岁，第二棵树认为是0岁，第三棵树认为是20岁，我们就取平均值10岁做最终结论？--当然不是！且不说这是投票方法并不是GBDT，只要训练集不变，独立训练三次的三棵树必定完全相同，这样做完全没有意义。之前说过，GBDT是把所有树的结论累加起来做最终结论的，所以可以想到每棵树的结论并不是年龄本身，而是年龄的一个累加量。**GBDT**的核心就在于，每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。比如A的真实年龄是18岁，但第一棵树的预测年龄是12岁，差了6岁，即残差为6岁。那么在第二棵树里我们把A的年龄设为6岁去学习，如果第二棵树真的能把A分到6岁的叶子节点，那累加两棵树的结论就是A的真实年龄；如果第二棵树的结论是5岁，则A仍然存在1岁的残差，第三棵树里A的年龄就变成1岁，继续学。这就是Gradient Boosting在GBDT中的意义，简单吧。

三、 GBDT工作过程实例。

还是年龄预测，简单起见训练集只有4个人，A,B,C,D，他们的年龄分别是14,16,24,26。其中A、B分别是高一和高三学生；C,D分别是应届毕业生和工作两年的员工。如果是用一棵传统的回归决策树来训练，会得到如下图1所示结果：



现在我们使用GBDT来做这件事，由于数据太少，我们限定叶子节点做多有两个，即每棵树都只有一个分枝，并且限定只学两棵树。我们会得到如下图2所示结果：



在第一棵树分枝和图1一样，由于A,B年龄较为相近，C,D年龄较为相近，他们被分为两拨，每拨用平均年龄作为预测值。此时计算残差（残差的意思就是： **A**的预测值 **+ A**的残差 **= A**的实际值），所以A的残差就是16-15=1（注意，A的预测值是指前面所有树累加的和，这里前面只有一棵树所以直接是15，如果还有树则需要都累加起来作为A的预测值）。进而得到A,B,C,D的残差分别为-1,1，-1,1。然后我们拿残差替代A,B,C,D的原值，到第二棵树去学习，如果我们的预测值和它们的残差相等，则只需把第二棵树的结论累加到第一棵树上就能得到真实年龄了。这里的数据显然是我可以做的，第二棵树只有两个值1和-1，直接分成两个节点。此时所有人的残差都是0，即每个人都得到了真实的预测值。

换句话说，现在A,B,C,D的预测值都和真实年龄一致了。Perfect!：
A: 14岁高一学生，购物较少，经常问学长问题；预测年龄A = 15 – 1 = 14
B: 16岁高三学生；购物较少，经常被学弟问问题；预测年龄B = 15 + 1 = 16
C: 24岁应届毕业生；购[illegible]问师兄问题；预测年龄C = 25 – 1 = 24

GBDT（MART） 迭代决策树入门
zhaonvsen: 对c4.5的理解有点出入：c4.5是取信息增益率最大的属性为分类属性，而不是熵。另外，c3.0采用的...

GBDT（MART） 迭代决策树入门
liufeng_cp: 多谢总结，赞，但Boost与迭代不是相同的概念，迭代只是boost的一种具体操作形式

D: 26岁工作两年员工；[illegible]常被师弟问问题；预测年龄D = 25 + 1 = 26

那么哪里体现了Gradien[illegible]棵树结束时想一想，无论此时的cost function是什么，是均方差还是均差，只要它以误差作为衡量标准，残差向量(-1, 1, -1, 1)都是它的全局最优方向，这就是Gradient。

讲到这里我们已经把GBDT最核心的概念、运算过程讲完了！没错就是这么简单。不过讲到这里很容易发现三个问题：

1）既然图**1**和图**2** 最终效果相同，为何还需要**GBDT**呢？
答案是过拟合。过拟合是指为了让训练集精度更高，学到了很多"仅在训练集上成立的规律"，导致换一个数据集当前规律就不适用了。其实只要允许一棵树的叶子节点足够多，训练集总是能训练到100%准确率的（大不了最后一个叶子上只有一个instance)。在训练精度和实际精度（或测试精度）之间，后者才是我们想要真正得到的。我们发现图1为了达到100%精度使用了3个feature（上网时长、时段、网购金额），其中分枝"上网时长>1.1h" 很显然已经过拟合了，这个数据集上A,B也许恰好A每天上网1.09h, B上网1.05小时，但用上网时间是不是>1.1小时来判断所有人的年龄很显然是有悖常识的；
相对来说图2的boosting虽然用了两棵树，但其实只用了2个feature就搞定了，后一个feature是问答比例，显然图2的依据更靠谱。（当然，这里是LZ故意做的数据，所以才能靠谱得如此狗血。实际中靠谱不靠谱总是相对的） Boosting的最大好处在于，每一步的残差计算其实变相地增大了分错instance的权重，而已经分对的instance则都趋向于0。这样后面的树就能越来越专注那些前面被分错的instance。就像我们做互联网，总是先解决60%用户的需求凑合着，再解决35%用户的需求，最后才关注那5%人的需求，这样就能逐渐把产品做好，因为不同类型用户需求可能完全不同，需要分别独立分析。如果反过来做，或者刚上来就一定要做到尽善尽美，往往最终会竹篮打水一场空。

2）**Gradient**呢？不是**"G"BDT**么？
到目前为止，我们的确没有用到求导的Gradient。在当前版本GBDT描述中，的确没有用到Gradient，该版本用残差作为全局最优的绝对方向，并不需要Gradient求解.

3）这不是**boosting**吧？**Adaboost**可不是这么定义的。
这是boosting，但不是Adaboost。**GBDT**不是**Adaboost Decistion Tree**。就像提到决策树大家会想起C4.5，提到boost多数人也会想到Adaboost。Adaboost是另一种boost方法，它按分类对错，分配不同的weight，计算cost function时使用这些weight，从而让"错分的样本权重越来越大，使它们更被重视"。Bootstrap也有类似思想，它在每一步迭代时不改变模型本身，也不计算残差，而是从N个instance训练集中按一定概率重新抽取N个instance出来（单个instance可以被重复sample），对着这N个新的instance再训练一轮。由于数据集变了迭代模型训练结果也不一样，而一个instance被前面分错的越厉害，它的概率就被设的越高，这样就能同样达到逐步关注被分错的instance，逐步完善的效果。Adaboost的方法被实践证明是一种很好的防止过拟合的方法，但至于为什么则至今没从理论上被证明。GBDT也可以在使用残差的同时引入Bootstrap re-sampling，GBDT多数实现版本中也增加的这个选项，但是否一定使用则有不同看法。re-sampling一个缺点是它的随机性，即同样的数据集合训练两遍结果是不一样的，也就是模型不可稳定复现，这对评估是很大挑战，比如很难说一个模型变好是因为你选用了更好的feature，还是由于这次sample的随机因素。

四、**Shrinkage**

Shrinkage（缩减）的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。用方程来看更清晰，即
没用Shrinkage时：（yi表示第i棵树上y的预测值， y(1~i)表示前i棵树y的综合预测值）
y(i+1) = 残差(y1~yi)， 其中： 残差(y1~yi) = y真实值 - y(1 ~ i)
y(1 ~ i) = SUM(y1, ..., yi)
Shrinkage不改变第一个方程，只把第二个方程改为：
y(1 ~ i) = y(1 ~ i-1) + step * yi

即Shrinkage仍然以残差作为学习目标，但对于残差学习出来的结果，只累加一小部分（step*残差）逐步逼近目标，step一般都比较小，如0.01~0.001（注意该step非gradient的step），导致各个树的残差是渐变的而不是陡变的。直觉上这也很好理解，不像直接用残差一步修复误差，而是只修复一点点，其实就是把大步切成了很多小步。本质上，**Shrinkage**为每棵树设置了一个**weight**，累加时要乘以这个**weight**，但和**Gradient**并没有关系。这个weight就是step。就像Adaboost一样，Shrinkage能减少过拟合发生也是经验证明的，目前还没有看到从理论的证明。

五、 **GBDT**的适用范围

该版本GBDT几乎可用于所有回归问题（线性/非线性），相对logistic regression仅能用于线性回归，GBDT的适用面非常广。亦可用于二分类问题（设定阈值，大于阈值为正例，反之为负例）。

六、 搜索引擎排序应用 **RankNet**

搜索排序关注各个doc的顺序而不是绝对值，所以需要一个新的cost function，而**RankNet**基本就是在定义这个**cost function**，它可以兼容不同的算法（GBDT、神经网络...）。

实际的搜索排序使用的是LambdaMART算法，必须指出的是由于这里要使用排序需要的cost function，**LambdaMART**迭代用的并不是残差。Lambda在这里充当替代残差的计算方法，它使用了一种类似Gradient*步长模拟残差的方法。这里的MART在求解方法上和之前说的残差略有不同，其区别描述见这里。

就像所有的机器学习一样，搜索排序的学习也需要训练集，这里一般是用人工标注实现，即对每一个(query,doc) pair给定一个分值（如1,2,3,4）,分值越高表示越相关，越应该排到前面。然而这些绝对的分值本身意义不大，例如你很难说1分和2分文档的相关程度差异是1分和3分文档差距的一半。相关度本身就是一个很主观的评判，标注人员无法做到这种定量标注，这种标准也无法制定。但标注人员很容易做到的是"AB都不错，但文档A比文档B更相关，所以A是4分，B是3分"。RankNet就是基于此制定了一个学习误差衡量方法，即cost function。具体而言，RankNet对任意两个文档A,B，通过它们的人工标注分差，用sigmoid函数估计两者顺序和逆序的概率P1。然后同理用机器学习到的分差计算概率P2（sigmoid的好处在于它允许机器学习得到的分值是任意实数值，只要它们的分差和标准分的分差一致，P2就趋近于P1）。这时利用P1和P2求的两者的交叉熵，该交叉熵就是cost function。它越低说明机器学得的当前排序越趋近于标注排序。为了体现NDCG的作用（NDCG是搜索排序业界最常用的评判标准），RankNet还在cost function中乘以了NDCG。

好，现在我们有了cost function，而且它是和各个文档的当前分值yi相关的，那么虽然我们不知道它的全局最优方向，但可以求导求Gradient，Gradient即每个文档得分的一个下降方向组成的N维向量，N为文档个数（应该说是query-doc pair个数）。这里仅仅是把"求残差"的逻辑替换为"求梯度"，可以这样想：梯度方向为每一步最优方向，累加的步数多了，总能走到局部最优点，若该点恰好为全局最优点，那和用残差的效果是一样的。这时套到之前讲的逻辑，GDBT就已经可以上了。那么最终排序怎么产生呢？很简单，每个样本通过Shrinkage累加都会得到一个最终得分，直接按分数从大到小排序就可以了（因为机器学习产生的是实数域的预测分，极少会出现在人工标注中常见的两文档分数相等的情况，几乎不同考虑同分文档的排序方式）

另外，如果feature个数太多，每一棵回归树都要耗费大量时间，这时每个分支时可以随机抽一部分feature来遍历求最优（ELF源码实现方式）。

上一篇 谈谈分类算法的选择
下一篇 GBDT源码剖析

我的同类文章

算法学习（15）

- 各种排序算法对比
- 各种基本算法实现小结
- pid match算法思想
- Wavelet Multiresolution Analysis
- 海量数据处理常用思路和方法
- （EM算法）The EM Algorithm
- larbin学习网址收集
- 机器学习相关资料
- Gradient Boost 算法流程分析

更多

主题推荐 color 迭代 rgb

猜你在找

- 有趣的算法（数据结构）
- 《C语言/C++学习指南》加密解密篇（安全相关算法）
- 数据结构和算法
- Java经典算法讲解
- C语言在嵌入式开发中的应用
- GBDTMART 迭代决策树入门教程
- 机器学习三GBDTGradient Boosting Decision Tree
- scikit-learn的GBDT工具进行特征选取
- gbdt算法
- LaTeX完整例子_参考文献图表和公式

查看评论

5楼 atomlion 2015-11-28 00:42发表

非常感谢哦！
我在实践中发现GBDT的预测效果非常好，在数据质量较差、复杂度高的大样本集中，几乎是效果最好的算法

4楼 keepreder 2015-08-03 17:19发表

logistic regression能用于非线性回归

3楼 zhaonvsen 2015-05-13 17:57发表

对c4.5的理解有点出入：c4.5是取信息增益率最大的属性为分类属性，而不是熵。另外，c3.0采用的是信息增益。即，Gain(S）=entropy(S)-sum(entropy(sa)*|sa|/|s|）

2楼 liufeng_cp 2015-04-22 14:28发表

多谢总结，赞，但Boost与迭代不是相同的概念，迭代只是boost的一种具体操作形式

Re: 大本_daben 2015-11-27 20:30发表

回复liufeng_cp：那到底怎么去理解boost呢？它的英文解释就是“推进、提升”的意思，我可以理解boost就是通过不断减小误差使最后结果向好的方面发展？

1楼 math1141 2014-05-11 17:14发表

多谢，研究了一天，对GBDT有了很好的了解。

发表评论

用 户 名： zhuqiuhui

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap