GBDT由三个概念组成：Regression Decision Tree(DT),Gradient Boosting(GB),Shrinkage(算法的一个重要演进分支，目前大部分源码都按该版本实现）。

GBDT与Xgboost RF的比较在其他笔记里

1.DT:决策树

决策树分为两大类，回归树和分类树，前者用于预测实数值，后者用于分类标签值；前者结果的加减是有意义的，后者结果加减无意义。

GBDT的核心在于**累加所有树的结果作为最终结果**，所以**GBDT中的树都是回归树**。回归树是如何工作的？

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，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featrue>阈值分成的两个分枝的熵最大的feature和阈值，按照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。

回归树总体流程类似，但在每个节点（不一定是叶子节点）都会得到一个预测值，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。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。

2.GB:梯度迭代

Boosting，迭代，即通过迭代多棵树来共同决策。怎么做？

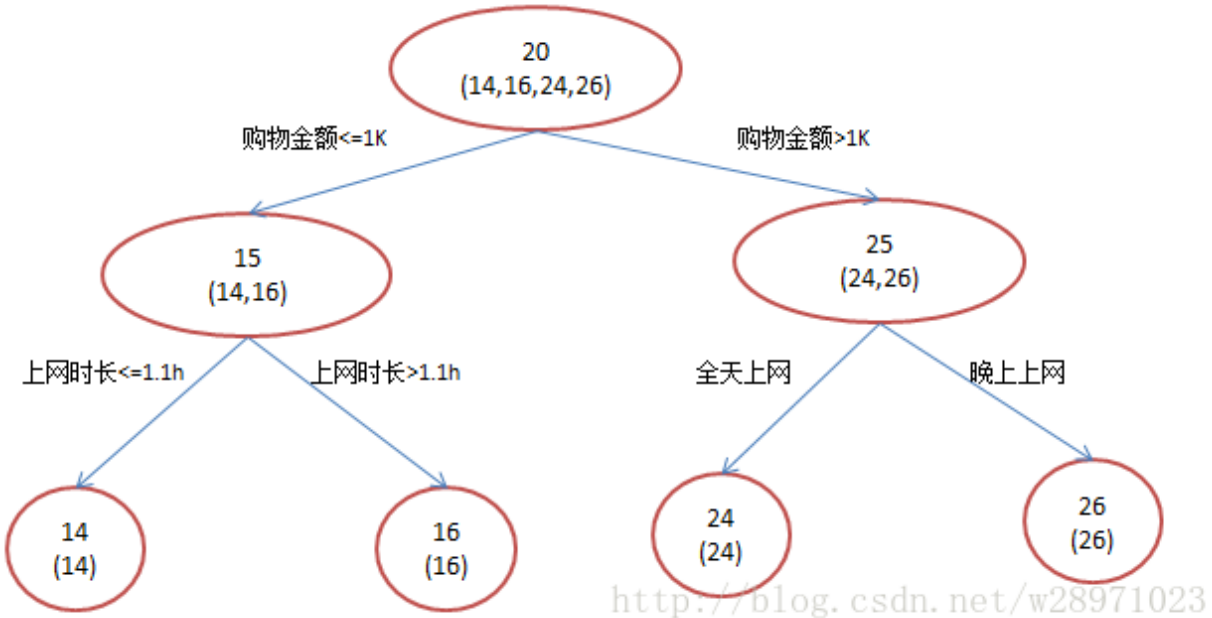
GBDT的核心就在于，**每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得到真实值的累加量。**

> 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（残差为18-（12+6）），那累加两棵树的结论就是A的真实年龄18岁；
> 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。

3.GBDT实例

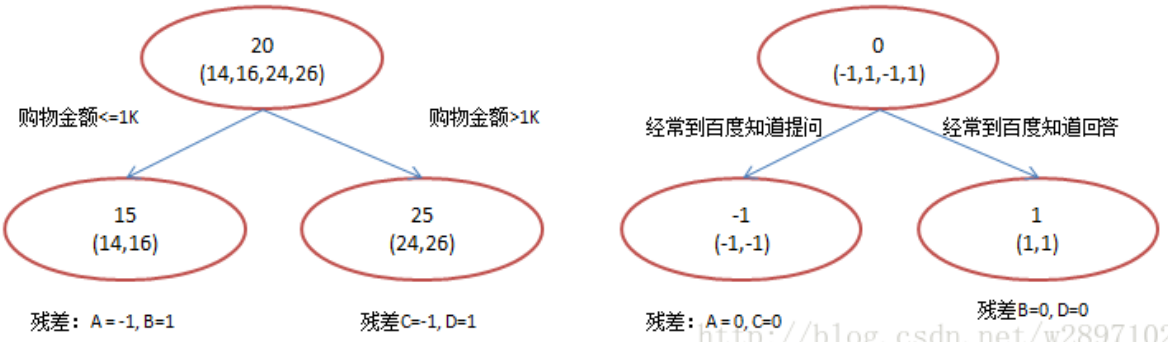
还是年龄预测，简单起见训练集只有4个人，A,B,C,D，他们的年龄分别是14,16,24,26。其中A、B分别是高一和高三学生；C,D分别是应届毕业生和工作两年的员工。

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：



每个节点用平均年龄作为预测值。分别计算ABCD的残差=真实值-预测值。注意，预测值是指前面所有树累加的和，第一棵树直接是15，第二颗树是15+（-1）=14

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。

那么哪里体现了Gradient呢？其实回到第一棵树结束时想一想，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。

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”。详情参见：https://blog.csdn.net/v_july_v/article/details/40718799

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。