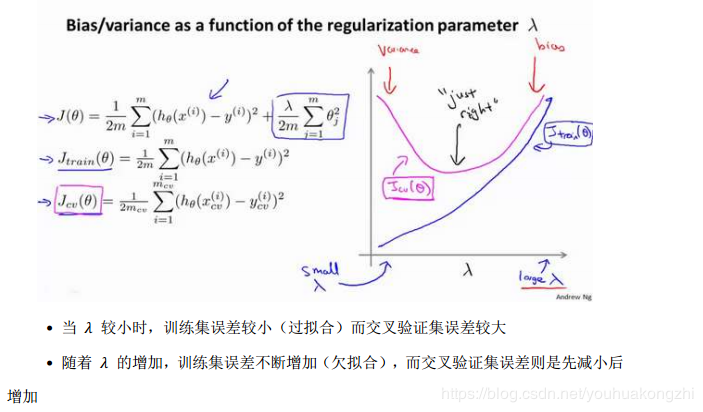
1. **集成学习**： 是一种机器学习范式。在集成学习中，我们会训练多个模型（通常称为「弱学习器」）解决相同的问题，并将它们结合起来以获得更好的结果。最重要的假设是：当弱模型被正确组合时，我们可以得到更精确和/或更鲁棒的模型。集成方法的思想是通过将这些弱学习器的偏置和/或方差结合起来，从而创建一个「强学习器」（或「集成模型」），从而获得更好的性能。为了建立一个集成学习方法，我们首先要选择待聚合的基础模型。在大多数情况下（包括在众所周知的 bagging 和 boosting 方法中），我们会使用单一的基础学习算法，这样一来我们就有了以不同方式训练的同质弱学习器。这样得到的集成模型被称为「同质的」。然而，也有一些方法使用不同种类的基础学习算法：将一些异质的弱学习器组合成「异质集成模型」。很重要的一点是：我们对弱学习器的选择应该和我们聚合这些模型的方式相一致。如果我们选择具有低偏置高方差的基础模型，我们应该使用一种倾向于减小方差的聚合方法；而如果我们选择具有低方差高偏置的基础模型，我们应该使用一种倾向于减小偏置的聚合方法。这就引出了如何组合这些模型的问题。我们可以用三种主要的旨在组合弱学习器的「元算法」：
2. Bagging，该方法通常考虑的是同质弱学习器，相互独立地并行学习这些弱学习器，并按照某种确定性的平均过程将它们组合起来。
3. Boosting，该方法通常考虑的也是同质弱学习器。它以一种高度自适应的方法顺序地学习这些弱学习器（每个基础模型都依赖于前面的模型），并按照某种确定性的策略将它们组合起来。

直观地说，每个模型都把注意力集中在目前最难拟合的观测数据上。这样一来，在这个过程的最后，我们就获得了一个具有较低偏置的强学习器（我们会注意到，boosting 也有减小方差的效果）。和 bagging 一样，Boosting 也可以用于回归和分类问题。由于其重点在于减小偏置，用于 boosting 的基础模型通常是那些低方差高偏置的模型。例如，如果想要使用树作为基础模型，我们将主要选择只有少许几层的较浅决策树。而选择低方差高偏置模型作为 boosting 弱学习器的另一个重要原因是：这些模型拟合的计算开销较低（参数化时自由度较低）。实际上，由于拟合不同模型的计算无法并行处理（与 bagging 不同），顺序地拟合若干复杂模型会导致计算开销变得非常高。

1. Stacking，该方法通常考虑的是异质弱学习器，并行地学习它们，并通过训练一个「元模型」将它们组合起来，根据不同弱模型的预测结果输出一个最终的预测结果。

非常粗略地说，我们可以说 bagging 的重点在于获得一个方差比其组成部分更小的集成模型，而 boosting 和 stacking 则将主要生成偏置比其组成部分更低的强模型（即使方差也可以被减小）

* 当你的训练误差和交叉验证误差或测试误差都很大，且值差不多时，是处于高偏差，低方差，欠拟合状态，需要增加多项式的次数来解决。
* 当你的训练误差和交叉验证误差差距很大，且测试集误差很小，验证误差很大，是处于低偏差，高方差，过拟合状态，需要减少多项式的次数或者利用正则化来解决。



1. **Bagging**

在「并行化的方法」中，我们单独拟合不同的学习器，因此可以同时训练它们。最著名的方法是「bagging」（代表「自助聚合」），它的目标是生成比单个模型更鲁棒的集成模型。这种统计技术先随机抽取出作为替代的 B 个观测值，然后根据一个规模为 N 的初始数据集生成大小为 B 的样本（称为自助样本）。首先初始数据集的大小 N 应该足够大，以捕获底层分布的大部分复杂性。这样，从数据集中抽样就是从真实分布中抽样的良好近似（代表性）。

1. **Boosting**

在「顺序化的方法中」，组合起来的不同弱模型之间不再相互独立地拟合。其思想是「迭代地」拟合模型，使模型在给定步骤上的训练依赖于之前的步骤上拟合的模型。尤其是介绍两个重要的 boosting 算法：自适应提升（adaboost ）和梯度提升（gradient boosting）简而言之，这两种元算法在顺序化的过程中创建和聚合弱学习器的方式存在差异。自适应增强算法会更新附加给每个训练数据集中观测数据的权重，而梯度提升算法则会更新这些观测数据的值。这里产生差异的主要原因是：两种算法解决优化问题（寻找最佳模型——弱学习器的加权和）的方式不同.

* 1. 自适应 boosting（通常被称为「**adaboost**」）

我们将集成模型定义为 L 个弱学习器的加权和

https://pics7.baidu.com/feed/c9fcc3cec3fdfc03a7597069ca455490a5c226b8.jpeg?token=6b949145ca7220f70d439e4360be761f

，其中 c\_l 是系数而 w\_l 是弱学习器

另外，我们将弱学习器逐个添加到当前的集成模型中，在每次迭代中寻找可能的最佳组合（系数、弱学习器）。换句话说，我们循环地将 s\_l 定义如下：

https://pics1.baidu.com/feed/574e9258d109b3defe04830fd3c5bf85810a4c98.jpeg?token=af81414944ec81f090f7565e27e7035c

其中，c\_l 和 w\_l 被挑选出来，使得 s\_l 是最适合训练数据的模型，因此这是对 s\_(l-1) 的最佳可能改进。我们可以进一步将其表示为：

https://pics1.baidu.com/feed/42a98226cffc1e1724f1d82655ea2507728de9de.jpeg?token=10a6d247ed29a6afd37f1dec237afddb

其中，E(.) 是给定模型的拟合误差，e(.,.)是损失/误差函数。因此，我们并没有在求和过程中对所有 L 个模型进行「全局优化」，而是通过「局部」优化来近似最优解并将弱学习器逐个添加到强模型中。

因此，假设我们面对的是一个二分类问题：数据集中有 N 个观测数据，我们想在给定一组弱模型的情况下使用 adaboost 算法。在算法的起始阶段（序列中的第一个模型），所有的观测数据都拥有相同的权重「1/N」。然后，我们将下面的步骤重复 L 次（作用于序列中的 L 个学习器）：

* 用当前观测数据的权重拟合可能的最佳弱模型
* 计算更新系数的值，更新系数是弱学习器的某种标量化评估指标，它表示相对集成模型来说，该弱学习器的分量如何
* 通过添加新的弱学习器与其更新系数的乘积来更新强学习器
* 计算新观测数据的权重，该权重表示我们想在下一轮迭代中关注哪些观测数据（聚和模型预测错误的观测数据的权重增加，而正确预测的观测数据的权重减小）

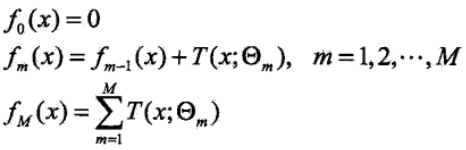
用当前观测数据的权重拟合可能的最佳弱模型计算更新系数的值，更新系数是弱学习器的某种标量化评估指标，它表示相对集成模型来说，该弱学习器的分量如何通过添加新的弱学习器与其更新系数的乘积来更新强学习器计算新观测数据的权重，该权重表示我们想在下一轮迭代中关注哪些观测数据（聚和模型预测错误的观测数据的权重增加，而正确预测的观测数据的权重减小）

* 1. **GBDT算法**

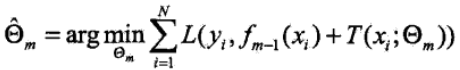
GBDT是梯度提升决策树（Gradient Boosting Decision Tree）的简称，GBDT可以说是最好的机器学习算法之一。GBDT分类和回归时的基学习器都是CART回归树，因为是拟合残差的**。回归树：使用平方误差最小准则.**

GBDT和Adaboost一样可以用前向分布算法来描述，不同之处在于Adaboost算法每次拟合基学习器时，输入的样本数据是不一样的（每一轮迭代时的样本权重不一致），因为Adaboost旨在重点关注上一轮分类错误的样本，GBDT算法在每一步迭代时是输出的值不一样，本轮要拟合的输出值是之前的加法模型的预测值和真实值的差值（模型的残差，也称为损失）。用于一个简单的例子来说明GBDT，假如某人的年龄为30岁，第一次用20岁去拟合，发现损失还有10岁，第二次用6岁去拟合10岁，发现损失还有4岁，第三次用3岁去拟合4岁，依次下去直到损失在我们可接受范围内。

以平方误差损失函数的回归问题为例，来看看以损失来拟合是个什么样子，采用前向分布算法：



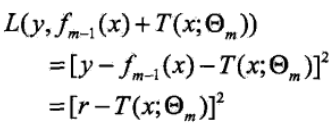
在第m次迭代时，我们要优化的损失函数：



此时我们采用平方误差损失函数为例：

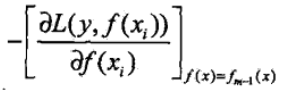
https://images2018.cnblogs.com/blog/1335117/201806/1335117-20180630180244945-1796094186.png

则上面损失函数变为：



问题就成了对残差r的拟合了

然而对于大多数损失函数，却没那么容易直接获得模型的残差，针对该问题，大神Freidman提出了用损失函数的负梯度来拟合本轮损失的近似值，拟合一个回归树

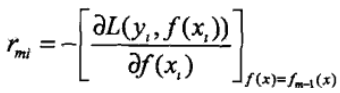


关于GBDT一般损失函数的具体算法流程如下：

1. 初始化f0(x)：



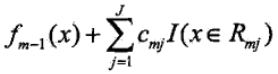
1. 第m次迭代时，计算当前要拟合的残差rmi：



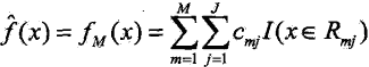
以rmi为输出值，对rmi拟合一个回归树（此时只是确定了树的结构，但是还未确定叶子节点中的输出值），然后通过最小化当前的损失函数，并求得每个叶子节点中的输出值cmj，j表示第j个叶子节点

https://images2018.cnblogs.com/blog/1335117/201806/1335117-20180630182104052-82128240.png

更新当前的模型fm(x)为：



1. 依次迭代到我们设定的基学习器的个数M，得到最终的模型，其中M表示基学习器的个数，J表示叶子节点的个数



GBDT算法提供了众多的可选择的损失函数，通过选择不同的损失函数可以用来处理分类、回归问题，比如用对数似然损失函数就可以处理分类问题。大概的总结下常用的损失函数：

1. 对于分类问题可以选用指数损失函数、对数损失函数。
2. 对于回归问题可以选用均方差损失函数、绝对损失函数。
3. 另外还有huber损失函数和分位数损失函数，也是用于回归问题，可以增加回归问题的健壮性，可以减少异常点对损失函数的影响。】

**GBDT的正则化**

在Adaboost中我们会对每个模型乘上一个弱化系数（正则化系数），减小每个模型对提升的贡献（注意：这个系数和模型的权重不一样，是在权重上又乘以一个0,1之间的小数），在GBDT中我们采用同样的策略，对于每个模型乘以一个系数λ (0 < λ ≤ 1)，降低每个模型对拟合损失的贡献，这种方法也意味着我们需要更多的基学习器。

第二种是每次通过按比例（推荐[0.5, 0.8] 之间）随机抽取部分样本来训练模型，这种方法有点类似Bagging，可以减小方差，但同样会增加模型的偏差，可采用交叉验证选取，这种方式称为子采样。采用子采样的GBDT有时也称为随机梯度提升树（SGBT）。

第三种就是控制基学习器CART树的复杂度，可以采用剪枝正则化。

**GBDT的优缺点**

GBDT的主要优点：

1. 可以灵活的处理各种类型的数据
2. 预测的准确率高
3. 使用了一些健壮的损失函数，如huber，可以很好的处理异常值
4. 预测阶段的计算速度快，树与树之间可并行化计算。
5. 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。
6. 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，并且也不需要对数据进行特殊的预处理如归一化等。

GBDT的缺点：

1. 由于基学习器之间的依赖关系，难以并行化处理，不过可以通过子采样的SGBT来实现部分并行。
2. 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
3. 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。
4. 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。

**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

**实例：**

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；
* 最后在第四课树中用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(yi,)=1/2(yi−)2

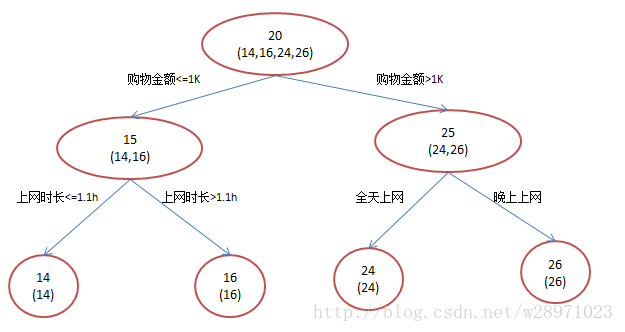
那此时的负梯度是这样计算的

−[∂l(yi, )∂ ]=(yi−)

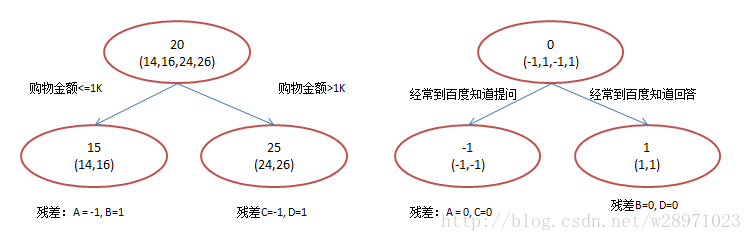
所以，当损失函数选用均方损失函数是时，每一次拟合的值就是（真实值 - 当前模型预测的值），即残差。此时的变量是, 即“当前预测模型的值”，也就是对它求负梯度。

训练过程

简单起见，假定训练集只有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，他们被分为左右两拨，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的实际值 - A的预测值 = A的残差），所以A的残差就是实际值14 - 预测值15 = 残差值-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。

然后拿它们的残差-1、1、-1、1代替A B C D的原值，到第二棵树去学习，第二棵树只有两个值1和-1，直接分成两个节点，即A和C分在左边，B和D分在右边，经过计算（比如A，实际值-1 - 预测值-1 = 残差0，比如C，实际值-1 - 预测值-1 = 0），此时所有人的残差都是0。残差值都为0，相当于第二棵树的预测值和它们的实际值相等，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，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，购物较多，经常问师兄问题，预测年龄C = 25 – 1 = 24

D: 26岁工作两年员工，购物较多，经常被师弟问问题，预测年龄D = 25 + 1 = 26

所以，GBDT需要将多棵树的得分累加得到最终的预测得分，且每一次迭代，都在现有树的基础上，增加一棵树去拟合前面树的预测结果与真实值之间的残差。

* 1. **Xgboost**

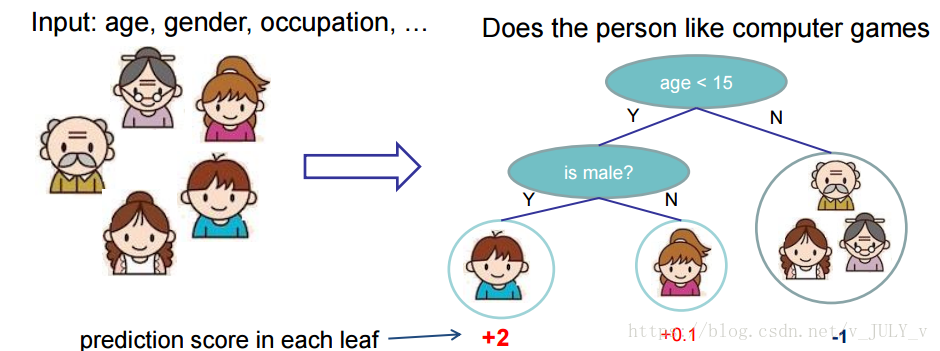
1. 什么是XGBoost

XGBoost是陈天奇等人开发的一个开源机器学习项目，高效地实现了GBDT算法并进行了算法和工程上的许多改进，被广泛应用在Kaggle竞赛及其他许多机器学习竞赛中并取得了不错的成绩。

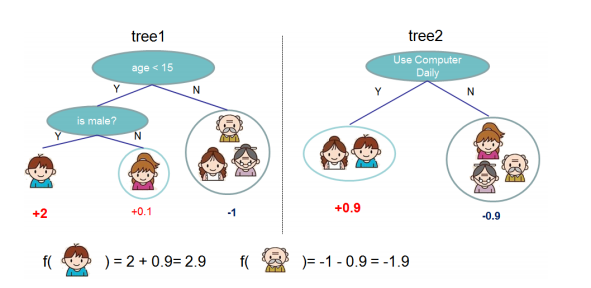
说到XGBoost，不得不提GBDT(Gradient Boosting Decision Tree)。因为XGBoost本质上还是一个GBDT，但是力争把速度和效率发挥到极致，所以叫X (Extreme) GBoosted。包括前面说过，两者都是boosting方法。

1.1 XGBoost树的定义

先来举个例子，我们要预测一家人对电子游戏的喜好程度，考虑到年轻和年老相比，年轻更可能喜欢电子游戏，以及男性和女性相比，男性更喜欢电子游戏，故先根据年龄大小区分小孩和大人，然后再通过性别区分开是男是女，逐一给各人在电子游戏喜好程度上打分，如下图所示。

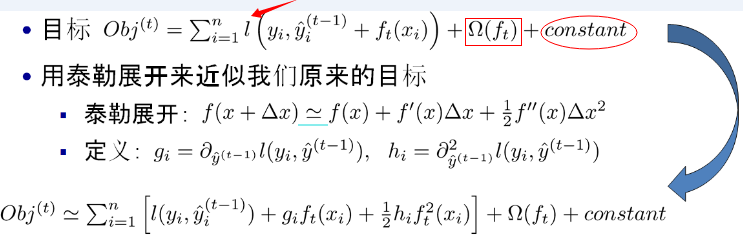


就这样，训练出了2棵树tree1和tree2，类似之前gbdt的原理，两棵树的结论累加起来便是最终的结论，所以小孩的预测分数就是两棵树中小孩所落到的结点的分数相加：2 + 0.9 = 2.9。爷爷的预测分数同理：-1 + （-0.9）= -1.9。具体如下图所示：



恩，你可能要拍案而起了，惊呼，这不是跟上文介绍的GBDT乃异曲同工么？

事实上，如果不考虑工程实现、解决问题上的一些差异，XGBoost与GBDT比较大的不同就是目标函数的定义。XGBoost的目标函数如下图所示：



其中：

红色箭头所指向的L 即为损失函数（比如平方损失函数：l(yi,)=(yi−)2

红色方框所框起来的是正则项（包括L1正则、L2正则）

红色圆圈所圈起来的为常数项

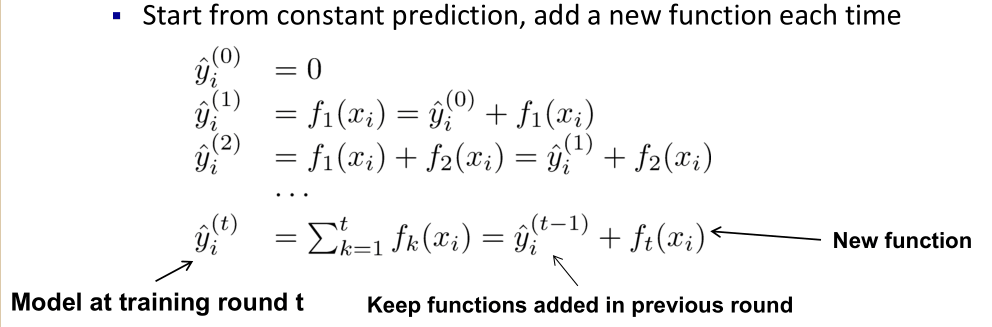
对于f(x)，XGBoost利用泰勒展开三项，做一个近似。f(x)表示的是其中一颗回归树。

看到这里可能有些读者会头晕了，这么多公式，我在这里只做一个简要式的讲解，具体的算法细节和公式求解请查看这篇博文，讲得很仔细：通俗理解kaggle比赛大杀器xgboost - https://blog.csdn.net/v\_JULY\_v/article/details/81410574

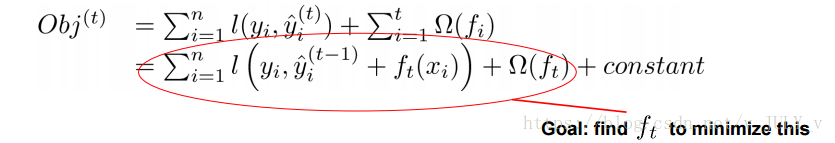
XGBoost的核心算法思想不难，基本就是：

* 不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数f(x)，去拟合上次预测的残差。
* 当我们训练完成得到k棵树，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数
* 最后只需要将每棵树对应的分数加起来就是该样本的预测值。

显然，我们的目标是要使得树群的预测值尽量接近真实值yi, 而且有尽量大的泛化能力。类似之前GBDT的套路，XGBoost也是需要将多棵树的得分累加得到最终的预测得分（每一次迭代，都在现有树的基础上，增加一棵树去拟合前面树的预测结果与真实值之间的残差）。



那接下来，我们如何选择每一轮加入什么 f 呢？答案是非常直接的，选取一个 f 来使得我们的目标函数尽量最大地降低。这里 f 可以使用泰勒展开公式近似。

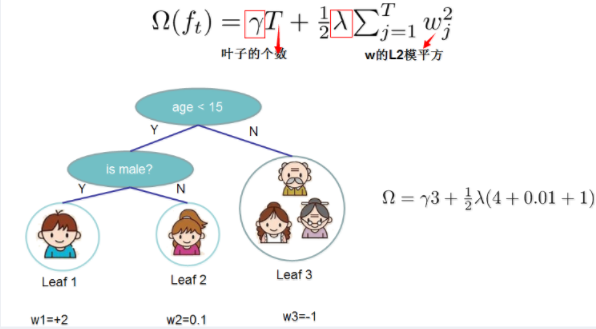


实质是把样本分配到叶子结点会对应一个obj，优化过程就是obj优化。也就是分裂节点到叶子不同的组合，不同的组合对应不同obj，所有的优化围绕这个思想展开。到目前为止我们讨论了目标函数中的第一个部分：训练误差。接下来我们讨论目标函数的第二个部分：正则项，即如何定义树的复杂度。

1.2 正则项：树的复杂度

XGBoost对树的复杂度包含了两个部分：

* 一个是树里面叶子节点的个数T
* 一个是树上叶子节点的得分w的L2模平方（对w进行L2正则化，相当于针对每个叶结点的得分增加L2平滑，目的是为了避免过拟合）



我们再来看一下XGBoost的目标函数（损失函数揭示训练误差 + 正则化定义复杂度）：

L(ϕ)=∑il(−yi)+∑kΩ(ft)

正则化公式也就是目标函数的后半部分，对于上式而言，是整个累加模型的输出，正则化项∑kΩ(ft)是则表示树的复杂度的函数，值越小复杂度越低，泛化能力越强。

1.3 树该怎么长

很有意思的一个事是，我们从头到尾了解了xgboost如何优化、如何计算，但树到底长啥样，我们却一直没看到。很显然，一棵树的生成是由一个节点一分为二，然后不断分裂最终形成为整棵树。那么树怎么分裂的就成为了接下来我们要探讨的关键。对于一个叶子节点如何进行分裂，XGBoost作者在其原始论文中给出了一种分裂节点的方法：枚举所有不同树结构的贪心法

不断地枚举不同树的结构，然后利用打分函数来寻找出一个最优结构的树，接着加入到模型中，不断重复这样的操作。这个寻找的过程使用的就是贪心算法。选择一个feature分裂，计算loss function最小值，然后再选一个feature分裂，又得到一个loss function最小值，你枚举完，找一个效果最好的，把树给分裂，就得到了小树苗。

总而言之，XGBoost使用了和CART回归树一样的想法，利用贪婪算法，遍历所有特征的所有特征划分点，不同的是使用的目标函数不一样。具体做法就是分裂后的目标函数值比单子叶子节点的目标函数的增益，同时为了限制树生长过深，还加了个阈值，只有当增益大于该阈值才进行分裂。从而继续分裂，形成一棵树，再形成一棵树，每次在上一次的预测基础上取最优进一步分裂/建树。

1.4 如何停止树的循环生成

凡是这种循环迭代的方式必定有停止条件，什么时候停止呢？简言之，设置树的最大深度、当样本权重和小于设定阈值时停止生长以防止过拟合。具体而言，则

* 当引入的分裂带来的增益小于设定阀值的时候，我们可以忽略掉这个分裂，所以并不是每一次分裂loss function整体都会增加的，有点预剪枝的意思，阈值参数为（即正则项里叶子节点数T的系数）；
* 当树达到最大深度时则停止建立决策树，设置一个超参数max\_depth，避免树太深导致学习局部样本，从而过拟合；
* 样本权重和小于设定阈值时则停止建树。什么意思呢，即涉及到一个超参数-最小的样本权重和min\_child\_weight，和GBM的 min\_child\_leaf 参数类似，但不完全一样。大意就是一个叶子节点样本太少了，也终止同样是防止过拟合；

2. XGBoost与GBDT有什么不同

除了算法上与传统的GBDT有一些不同外，XGBoost还在工程实现上做了大量的优化。总的来说，两者之间的区别和联系可以总结成以下几个方面。

* GBDT是机器学习算法，XGBoost是该算法的工程实现。
* 在使用CART作为基分类器时，XGBoost显式地加入了正则项来控制模 型的复杂度，有利于防止过拟合，从而提高模型的泛化能力。
* GBDT在模型训练时只使用了代价函数的一阶导数信息，XGBoost对代 价函数进行二阶泰勒展开，可以同时使用一阶和二阶导数。
* 传统的GBDT采用CART作为基分类器，XGBoost支持多种类型的基分类 器，比如线性分类器。
* 传统的GBDT在每轮迭代时使用全部的数据，XGBoost则采用了与随机 森林相似的策略，支持对数据进行采样。
* 传统的GBDT没有设计对缺失值进行处理，XGBoost能够自动学习出缺 失值的处理策略。

3. 为什么XGBoost要用泰勒展开，优势在哪里？

XGBoost使用了一阶和二阶偏导, 二阶导数有利于梯度下降的更快更准. 使用泰勒展开取得函数做自变量的二阶导数形式, 可以在不选定损失函数具体形式的情况下, 仅仅依靠输入数据的值就可以进行叶子分裂优化计算, 本质上也就把损失函数的选取和模型算法优化/参数选择分开了. 这种去耦合增加了XGBoost的适用性, 使得它按需选取损失函数, 可以用于分类, 也可以用于回归。

Github:

https://github.com/NLP-LOVE/ML-NLP/blob/master/Machine%20Learning/3.3%20XGBoost/3.3%20XGBoost.ipynb

1. **Stacking**

Stacking 与 bagging 和 boosting 主要存在两方面的差异。首先，Stacking 通常考虑的是异质弱学习器（不同的学习算法被组合在一起），而bagging 和 boosting 主要考虑的是同质弱学习器。其次，stacking 学习用元模型组合基础模型，而bagging 和 boosting 则根据确定性算法组合弱学习器。

堆叠法（Stacking）

正如上文已经提到的，stacking 的概念是学习几个不同的弱学习器，并通过训练一个元模型来组合它们，然后基于这些弱模型返回的多个预测结果输出最终的预测结果。

因此，为了构建 stacking 模型，我们需要定义两个东西：想要拟合的 L 个学习器以及组合它们的元模型。

例如，对于分类问题来说，我们可以选择 KNN 分类器、logistic 回归和SVM 作为弱学习器，并决定学习神经网络作为元模型。然后，神经网络将会把三个弱学习器的输出作为输入，并返回基于该输入的最终预测。

所以，假设我们想要拟合由 L 个弱学习器组成的 stacking 集成模型。我们必须遵循以下步骤：

* 将训练数据分为两组
* 选择 L 个弱学习器，用它们拟合第一组数据
* 使 L 个学习器中的每个学习器对第二组数据中的观测数据进行预测
* 在第二组数据上拟合元模型，使用弱学习器做出的预测作为输入

将训练数据分为两组选择 L 个弱学习器，用它们拟合第一组数据使 L 个学习器中的每个学习器对第二组数据中的观测数据进行预测在第二组数据上拟合元模型，使用弱学习器做出的预测作为输入在前面的步骤中，我们将数据集一分为二，因为对用于训练弱学习器的数据的预测与元模型的训练不相关。因此，将数据集分成两部分的一个明显缺点是，我们只有一半的数据用于训练基础模型，另一半数据用于训练元模型。

为了克服这种限制，我们可以使用某种「k-折交叉训练」方法（类似于 k-折交叉验证中的做法）。这样所有的观测数据都可以用来训练元模型：对于任意的观测数据，弱学习器的预测都是通过在 k-1 折数据（不包含已考虑的观测数据）上训练这些弱学习器的实例来完成的。

换句话说，它会在 k-1 折数据上进行训练，从而对剩下的一折数据进行预测。迭代地重复这个过程，就可以得到对任何一折观测数据的预测结果。这样一来，我们就可以为数据集中的每个观测数据生成相关的预测，然后使用所有这些预测结果训练元模型。