通俗理解kaggle比赛大杀器xgboost_v_JULY_v的博客-CSDN博客_通俗理解kaggle比赛大杀器xgboost

通俗理解kaggle比赛大杀器xgboost

说明：若出现部分图片无法正常显示而影响阅读，请以此处的文章为准：xgboost 题库版。
时间：二零一九年三月二十五日。

0 前言

xgboost一直在竞赛江湖里被传为神器，比如时不时某个kaggle/天池比赛中，某人用xgboost于千军万马中斩获冠军。

而我们的机器学习课程里也必讲xgboost，如寒所说：“RF和GBDT是工业界大爱的模型，Xgboost 是大杀器包裹，Kaggle各种Top排行榜曾一度呈现Xgboost一统江湖的局面，另外某次滴滴比赛第一名的改进也少不了Xgboost的功劳”。

此外，公司七月在线从2016年上半年起，就开始组织学员参加各种比赛，以在实际竞赛项目中成长（毕竟，搞AI不可能没实战，而参加比赛历经数据处理、特征选择、模型调优、代码调参，是一个极好的真刀真枪的实战机会，对能力的提升和找/换工作的帮助都非常大）。

AI大潮之下，今年特别多从传统IT转行转岗转型AI的朋友，很多朋友都咨询如何转行AI，我一般都会着重强调学习AI或找/换AI的四大金刚：课程 + 题库 + OJ + kaggle/天池。包括集训营的毕业考核更会融合kaggle或天池比赛。

考虑到kaggle/天池比赛对搞数学科学的重要性，特写此文介绍xgboost，助力大家快速入门xgboost以及在比赛中获得优异成绩。

最后，xgboost不是我July发明的，但我会确保本文对它的介绍是最通俗易懂的（且本文得到七月在线AI lab负责人陈博士审校）。另，感谢文末所列的全部参考文献，有何问题，欢迎随时留言评论，thanks。

1 决策树

举个例子，集训营某一期有100多名学员，假定给你一个任务，要你统计男生女生各多少人，当一个一个学员依次上台站到你面前时，你会怎么区分谁是男谁是女呢？

很快，你考虑到男生的头发一般很短，女生的头发一般比较长，所以你通过头发的长短将这个班的所有学员分为两拨，长发的为“女”，短发为“男”。

相当于你依靠一个指标“头发长短”将整个班的人进行了划分，于是形成了一个简单的决策树，而划分的依据是头发长短。
这时，有的人可能有不同意见了：为什么要用“头发长短”划分呀，我可不可以用“穿的鞋子是否是高跟鞋”，“有没有喉结”等等这些来划分呢，答案当然是可以的。

但究竟根据哪个指标划分更好呢？很直接的判断是哪个分类效果更好则优先用哪个。所以，这时就需要一个评价标准来量化分类效果了。

怎么判断“头发长短”或者“是否有喉结”是最好的划分方式，效果怎么量化呢？直观上来说，如果根据某个标准分类人群后，纯度越高效果越好，比如说你分为两群，“女”那一群都是女的，“男”那一群全是男的，那这个效果是最好的。但有时实际的分类情况不是那么理想，所以只能说越接近这种情况，我们则认为效果越好。

量化分类效果的方式有很多，比如信息增益（ID3）、信息增益率（C4.5）、基尼系数（CART）等等。

信息增益的度量标准：熵

ID3算法的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。

什么是信息增益呢？为了精确地定义信息增益，我们先定义信息论中广泛使用的一个度量标准，称为**熵**（entropy），它刻画了任意样例集的纯度（purity）。给定包含关于某个目标概念的正反样例的样例集S，那么S相对这个布尔型分类的熵为：

$$Entropy(S) \equiv -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus}$$

上述公式中，p+代表正样例，比如在本文开头第二个例子中p+则意味着去打羽毛球，而p-则代表反样例，不去打球(在有关熵的所有计算中我们定义ologo为0)。

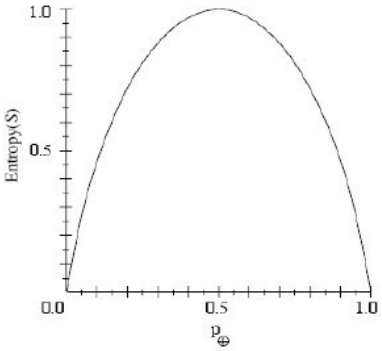
举例来说，假设S是一个关于布尔概念的有14个样例的集合，它包括9个正例和5个反例（我们采用记号[9+，5-]来概括这样的数据样例），那么S相对于这个布尔样例的熵为：

Entropy（[9+，5-]）=-（9/14）log2（9/14）-（5/14）log2（5/14）=0.940。

So，根据上述这个公式，我们可以得到：

- 如果S的所有成员属于同一类，则Entropy(S)=0；
- 如果S的正反样例数量相等，则Entropy(S)=1；
- 如果S的正反样例数量不等，则熵介于0，1之间

如下图所示：



看到没，通过Entropy的值，你就能评估当前分类树的分类效果好坏了。

更多细节如剪枝、过拟合、优缺点、可以参考此文《决策树学习》。

所以，现在决策树的灵魂已经有了，即依靠某种指标进行树的分裂达到分类/回归的目的，总是希望纯度越高越好。

2.回归树与集成学习

如果用一句话定义xgboost，很简单：Xgboost就是由很多CART树集成。但，什么是CART树？

数据挖掘或机器学习中使用的决策树有两种主要类型：

1. 分类树分析是指预测结果是数据所属的类（比如某个电影去看还是不看）
2. 回归树分析是指预测结果可以被认为是实数（例如房屋的价格，或患者在医院中的逗留时间）

而术语分类回归树（CART，Classification And Regression Tree）分析是用于指代上述两种树的总称，由Breiman等人首先提出。

2.1 回归树

事实上，分类与回归是两个很接近的问题，分类的目标是根据已知样本的某些特征，判断一个新的样本属于哪种已知的样本类，它的结果是离散值。而回归的结果是连续的值。当然，本质是一样的，都是特征（feature）到结果/标签（label）之间的映射。

理清了什么是分类和回归之后，理解分类树和回归树就不难了。

分类树的样本输出（即响应值）是类的形式，比如判断这个救命药是真的还是假的，周末去看电影《风语咒》还是不去。而回归树的样本输出是数值的形式，比如给某人发放房屋贷款的数额就是具体的数值，可以是0到300万元之间的任意值。

所以，对于回归树，你没法再用分类树那套信息增益、信息增益率、基尼系数来判定树的节点分裂了，你需要采取新的方式评估效果，包括预测误差（常用的有均方误差、对数误差等）。而且节点不再是类别，是数值（预测值），那么怎么确定呢？有的是节点内样本均值，有的是最优化算出来的比如Xgboost。

CART回归树是假设树为二叉树，通过不断将特征进行分裂。比如当前树结点是基于第j个特征值进行分裂的，设该特征值小于s的样本划分为左子树，大于s的样本划分为右子树。

$$R_1(j,s) = \{x|x^{(j)} \leq s\} \; and \; R_2(j,s) = \{x|x^{(j)} > s\}$$

而CART回归树实质上就是在该特征维度对样本空间进行划分，而这种空间划分的优化是一种NP难问题，因此，在决策树模型中是使用启发式方法解决。典型CART回归树产生的目标函数为：

$$\sum_{x_i \in R_m} (y_i - f(x_i))^2$$

因此，当我们为了求解最优的切分特征j和最优的切分点s，就转化为求解这么一个目标函数：

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

所以我们只要遍历所有特征的的所有切分点，就能找到最优的切分特征和切分点。最终得到一棵回归树。

2.2 boosting集成学习

所谓集成学习，是指构建多个分类器（弱分类器）对数据集进行预测，然后用某种策略将多个分类器预测的结果集成起来，作为最终预测结果。通俗比喻就是“三个臭皮匠赛过诸葛亮”，或一个公司董事会上的各董事投票决策，它要求每个弱分类器具备一定的“准确性”，分类器之间具备“差异性”。

集成学习根据各个弱分类器之间有无依赖关系，分为Boosting和Bagging两大流派：

1. Boosting流派，各分类器之间有依赖关系，必须串行，比如Adaboost、GBDT(Gradient Boosting Decision Tree)、Xgboost
2. Bagging流派，各分类器之间没有依赖关系，可各自并行，比如随机森林（Random Forest）

而著名的Adaboost作为boosting流派中最具代表性的一种方法，本博客曾详细介绍它。

AdaBoost，是英文"Adaptive Boosting"（自适应增强）的缩写，由Yoav Freund和Robert Schapire在1995年提出。它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

具体说来，整个Adaboost 迭代算法就3步：

1. 初始化训练数据的权值分布。如果有N个样本，则每一个训练样本最开始时都被赋予相同的权值：1/N。
2. 训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权值就被降低；相反，如果某个样本点没有被准确地分类，那么它的权值就得到提高。然后，权值更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。
3. 将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

而另一种boosting方法GBDT（Gradient Boost Decision Tree），则与AdaBoost不同，GBDT每一次的计算是都为了减少上一次的残差，进而在残差减少（负梯度）的方向上建立一个新的模型。

boosting集成学习由多个相关联的决策树联合决策，什么叫相关联？举个例子

1. 有一个样本[数据->标签]是：[(2，4，5)-> 4]
2. 第一棵决策树用这个样本训练的预测为3.3
3. 那么第二棵决策树训练时的输入，这个样本就变成了：[(2，4，5)-> 0.7]
4. 也就是说，下一棵决策树输入样本会与前面决策树的训练和预测相关

很快你会意识到，Xgboost为何也是一个boosting的集成学习了。

而一个回归树形成的关键点在于：

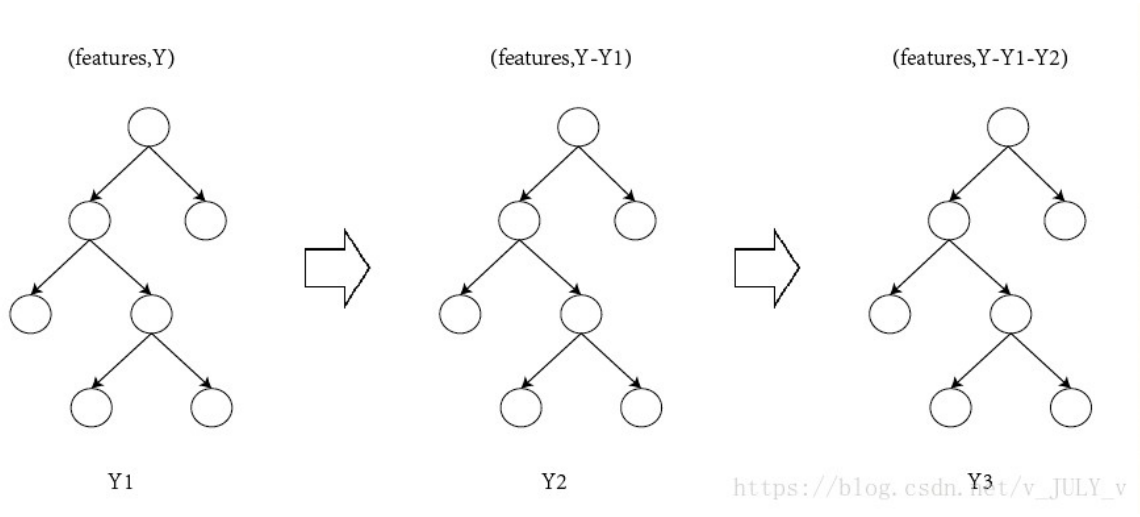
- 分裂点依据什么来划分（如前面说的均方误差最小，loss）；
- 分类后的节点预测值是多少（如前面说，有一种是将叶子节点下各样本实际值得均值作为叶子节点预测误差，或者计算所得）

至于另一类集成学习方法，比如Random Forest（随机森林）算法，各个决策树是独立的、每个决策树在样本堆里随机选一批样本，随机选一批特征进行独立训练，各个决策树之间没有啥关系。本文暂不展开介绍。

3.GBDT

说到Xgboost，不得不先从GBDT(Gradient Boosting Decision Tree)说起。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。

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梯度/残差(这个梯度/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。如图所示：Y = Y1 + Y2 + Y3。



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？

1. 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；
2. 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；
3. 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；
4. 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。

注意，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，

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

那此时的负梯度是这样计算的

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

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。

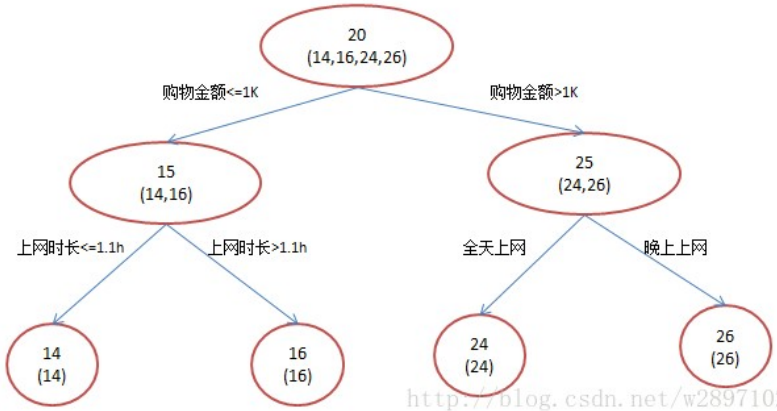
> 另外，这里还得再啰嗦一下，上面预测年龄的第一个步骤中的“随便”二字看似随便，其实深入思考一下一点都不随便，你会发现大部分做预测的模型，基本都是这么个常规套路，先随便用一个值去预测，然后对比预测值与真实值的差距，最后不断调整 缩小差距。所以会出来一系列目标函数：确定目标，和损失函数：缩小误差。
>
> 再进一步思考，你会发现这完全符合人类做预测的普遍常识、普遍做法，当对一个事物不太了解时，一开始也是根据经验尝试、初探，直到逼近某种意义上的接近或者完全吻合。

还是年龄预测的例子。

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。

所以，现在的问题就是我们要预测这4个人的年龄，咋下手？很简单，先随便用一个年龄比如20岁去拟合他们，然后根据实际情况不断调整。

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。

> 残差在数理统计中是指实际观察值与估计值（拟合值）之间的差。“残差”蕴含了有关模型基本假设的重要信息。如果回归模型正确的话， 我们可以将残差看作误差的观测值。

进而得到A,B,C,D的残差分别为-1,1，-1,1。

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。

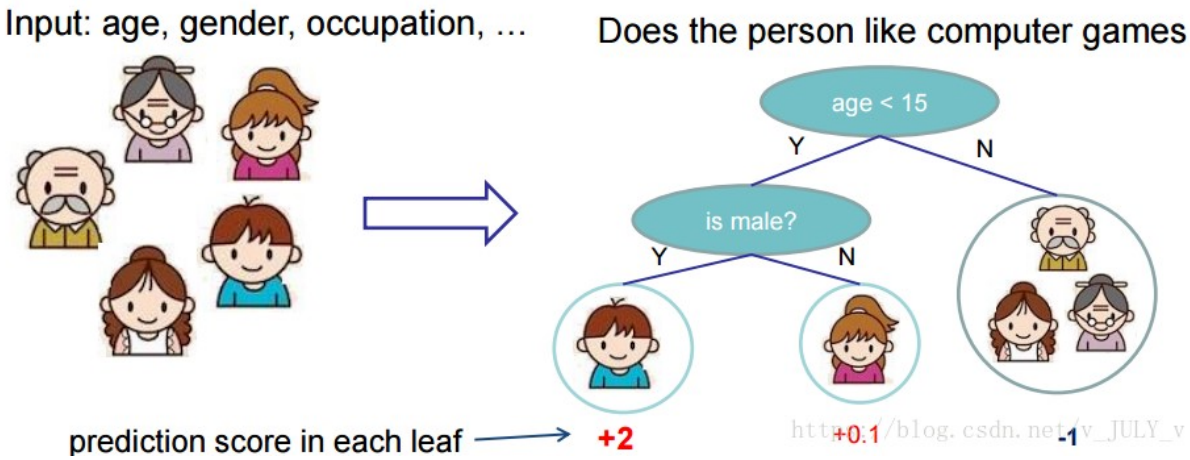
4.Xgboost

4.1 xgboost树的定义

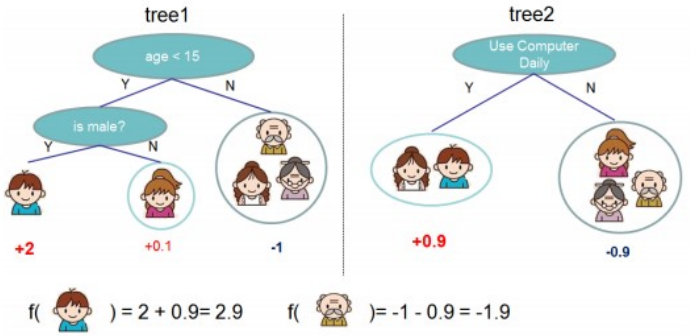
本节的示意图基本引用自xgboost原作者陈天奇的讲义PPT中。

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。

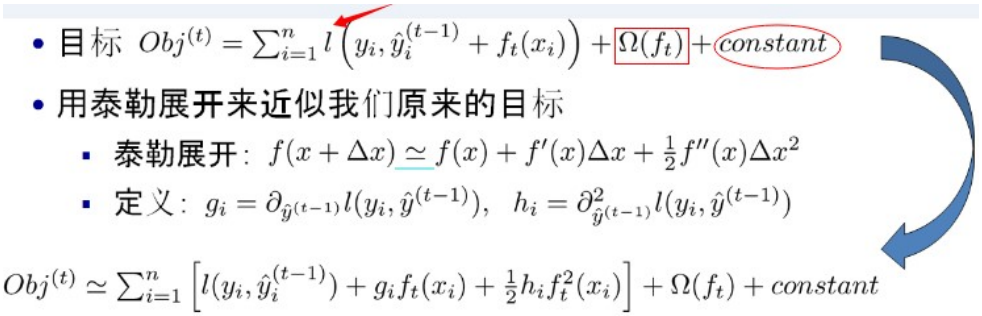


就这样，训练出了2棵树tree1和tree2，类似之前gbdt的原理，两棵树的结论累加起来便是最终的结论，所以小孩的预测分数就是两棵树中小孩所落到的结点的分数相加：2 + 0.9 = 2.9。爷爷的预测分数同理：-1 + （-0.9）= -1.9。具体如下图所示



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

我们可以很清晰地看到，最终的目标函数只依赖于每个数据点在误差函数上的一阶导数和二阶导数。

额，峰回路转，突然丢这么大一个公式，不少人可能瞬间就懵了。没事，下面咱们来拆解下这个目标函数，并一一剖析每个公式、每个符号、每个下标的含义。

4.2 xgboost目标函数

xgboost的核心算法思想不难，基本就是

1. 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，去拟合上次预测的残差**。

$$\hat{y} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$$
$$where\ F = \{f(x) = w_{q(x)}\}(q: R^m \to T, w \in R^T)$$

   注：w_q(x)为叶子节点q的分数，$\mathcal{F}$对应了所有K棵回归树（regression tree）的集合，而f(x)为其中一棵回归树。

2. 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
3. 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$\hat{y}_i$尽量接近真实值y_i，而且有尽量大的泛化能力。

所以，从数学角度看这是一个泛函最优化问题，故把目标函数简化如下：

$$L(\phi) = \sum_i l(\hat{y}_i - y_i) + \sum_k \Omega(f_k)$$

如你所见，这个目标函数分为两部分：损失函数和正则化项。且**损失函数揭示训练误差**（即预测分数和真实分数的差距），**正则化定义复杂度。**

对于上式而言，$\hat{y}_i$是整个累加模型的输出，正则化项$\sum_k \Omega(f_k)$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，其表达式为

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

T表示叶子节点的个数，w表示叶子节点的分数。直观上看，目标要求预测误差尽量小，且叶子节点T尽量少（γ控制叶子结点的个数），节点数值w尽量不极端（λ控制叶子节点的分数不会过大），防止过拟合。

插一句，一般的目标函数都包含下面两项

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

误差函数：我们的模型有多拟合数据

正则化项：惩罚复杂模型

其中，误差/损失函数鼓励我们的模型尽量去拟合训练数据，使得最后的模型会有比较少的 bias。而正则化项则鼓励更加简单的模型。因为当模型简单之后，有限数据拟合出来结果的随机性比较小，不容易过拟合，使得最后模型的预测更加稳定。

4.2.1 模型学习与训练误差

具体来说，目标函数第一部分中的i表示第i个样本，$l(\hat{y}_i - y_i)$表示第i个样本的预测误差，我们的目标当然是误差越小越好。

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）。

- Start from constant prediction, add a new function each time

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

New function

Model at training round t

Keep functions added in previous round

但，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。

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant
\end{aligned}$$

Goal: find f_t to minimize this

再强调一下，考虑到**第t轮的模型预测值**$\hat{y}_i^{(t)}$ **= 前t-1轮的模型预测**$\hat{y}_i^{(t-1)} + f_t(x_i)$**，因此误差函数记为：**$l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$，后面一项为正则化项。

对于这个误差函数的式子而言，在第t步，y_i是真实值，即已知，$\hat{y}_i^{(t-1)}$可由上一步第t-1步中的y_i^{t-2}加上$f_{t-1}(x_i)$计算所得，某种意义上也算已知值，故模型学习的是f。

上面那个Obj的公式表达的可能有些过于抽象，我们可以考虑当l是平方误差的情况（相当于$l(y_i, y^i) = (y_i - y^i)^2$），这个时候我们的目标可以被写成下面这样的二次函数（图中画圈的部分表示的就是预测值和真实值之间的残差）：

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\
&= \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const
\end{aligned}
$$

更加一般的，损失函数不是二次函数咋办？利用泰勒展开，不是二次的想办法近似为二次（如你所见，定义了一阶导g和二阶导h）。

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

保留二次项

$$
Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) + constant
$$

- *If you are not comfortable with this, think of square loss*

$$
g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2
$$

- Compare what we get to previous slide

恩恩，注意了！不少人可能就会在这里卡壳，网上也很少有文章解释清楚，在和七月在线AI lab陈博士讨论之后，发现这里面最关键的其实就是把泰勒二阶展开各项和xgboost 目标函数的对应关系搞清楚，相当于我们可以利用泰勒二阶展开去做目标函数的近似。

首先，这是泰勒二阶展开$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$

对应到xgboost的目标函数里头

$$
Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant
$$

忽略损失函数l中的y_i（别忘了上面说的“ 在第t步，y_i是真实值，即已知 ”，不影响后续目标函数对$\hat{y}_i^{(t-1)}$的偏导计算），做下一一对应：

得到：

$$
Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) + constant
$$

其中，$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

呜呼，透了！不过，这个转化过程中的关键泰勒二次展开到底是哪来的呢？

> 在数学中，泰勒公式（英语：Taylor's Formula）是一个用函数在某点的信息描述其附近取值的公式。这个公式来自于微积分的泰勒定理（Taylor's theorem），泰勒定理描述了一个可微函数，如果函数足够光滑的话，在已知函数在某一点的各阶导数值的情况之下，泰勒公式可以用这些导数值做系数构建一个多项式来近似函数在这一点的邻域中的值，这个多项式称为泰勒多项式（Taylor polynomial）。
>
> 相当于告诉我们可由利用泰勒多项式的某些次项做原函数的近似。
>
> 泰勒定理：
> 设 n 是一个正整数。如果定义在一个包含 a 的区间上的函数 f 在 a 点处 n+1 次可导，那么对于这个区间上的任意 x，都有：
>
> $$
> f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(x)
> $$
>
> 其中的多项式称为函数在a 处的泰勒展开式，剩余的$R_n(x)$是泰勒公式的余项，是$(x-a)^n$的高阶无穷小。

接下来，考虑到我们的第t 颗回归树是根据前面的t-1颗回归树的残差得来的，相当于t-1颗树的值$\hat{y}_i^{(t-1)}$是已知的。换句话说，$l(y_i, \hat{y}_i^{(t-1)})$对目标函数的优化不影响，可以直接去掉，且常数项也可以移除，从而得到如下一个比较统一的目标函数。

- Objective, with constants removed

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

  - where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

这时，目标函数只依赖于每个数据点在误差函数上的一阶导数g和二阶导数h（相信你已经看出xgboost和gbdt的不同了，目标函数保留了泰勒展开的二次项）。

总的指导原则如就职Google的读者crab6789所说：

> 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。

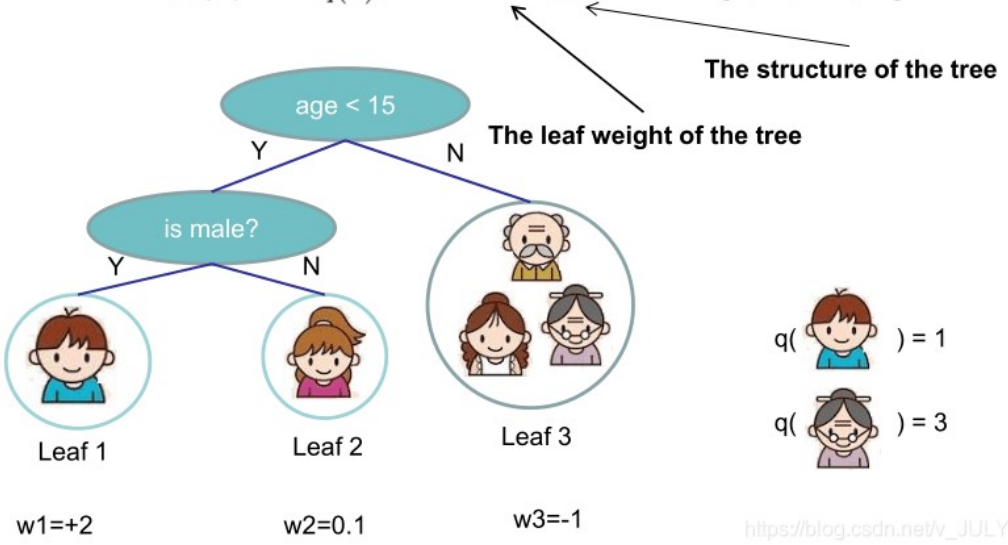
4.2.2 正则项：树的复杂度

首先，梳理下几个规则

- 用叶子节点集合以及叶子节点得分表示
- 每个样本都落在一个叶子节点上
- q(x)表示样本x在某个叶子节点上，wq(x)是该节点的打分，即该样本的模型预测值

所以当我们把树成**结构部分q**和**叶子权重部分w后，**结构函数q把输入映射到叶子的索引号上面去，而w给定了每个索引号对应的叶子分数是什么。

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

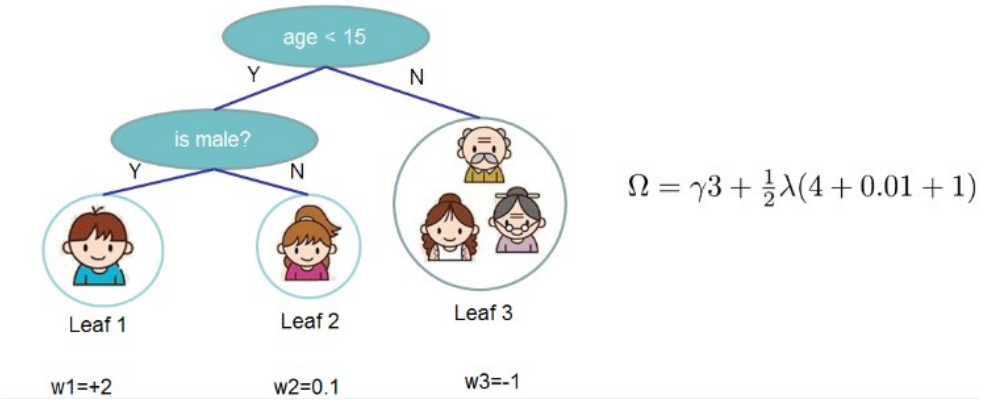


另外，如下图所示，xgboost对树的复杂度包含了两个部分：

1. 一个是树里面叶子节点的个数T
2. 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）

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

叶子的个数　　w的L2模平方



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

在这种新的定义下，我们可以把之前的目标函数进行如下变形（另，别忘了：$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$）

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

其中 I_j 被定义为每个叶节点 j 上面样本下标的集合 $I_j = \{i | q(x_i) = j\}$，g是一阶导数，h是二阶导数。这一步是由于xgboost目标函数第二部分加了两个正则项，一个是叶子节点个数(T),一个是叶子节点的分数(w)。

从而，加了正则项的目标函数里就出现了两种累加

这一个目标包含了T个相互独立的单变量二次函数。

理解这个推导的关键在哪呢？在和AI lab陈博士讨论之后，其实就在于理解这个定义：I_j 被定义为每个叶节点 j 上面样本下标的集合 $I_j = \{i | q(x_i) = j\}$，这个定义里的q(xi)要表达的是：每个样本值xi 都能通过函数q(xi)映射到树上的某个叶子节点，从而通过这个定义把两种累加统一到了一起。

接着，我们可以定义

$$G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$$

最终公式可以化简为

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \\ &= \sum_{j=1}^{T} [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T \end{aligned}$$

通过对 w_j 求导等于0，可以得到

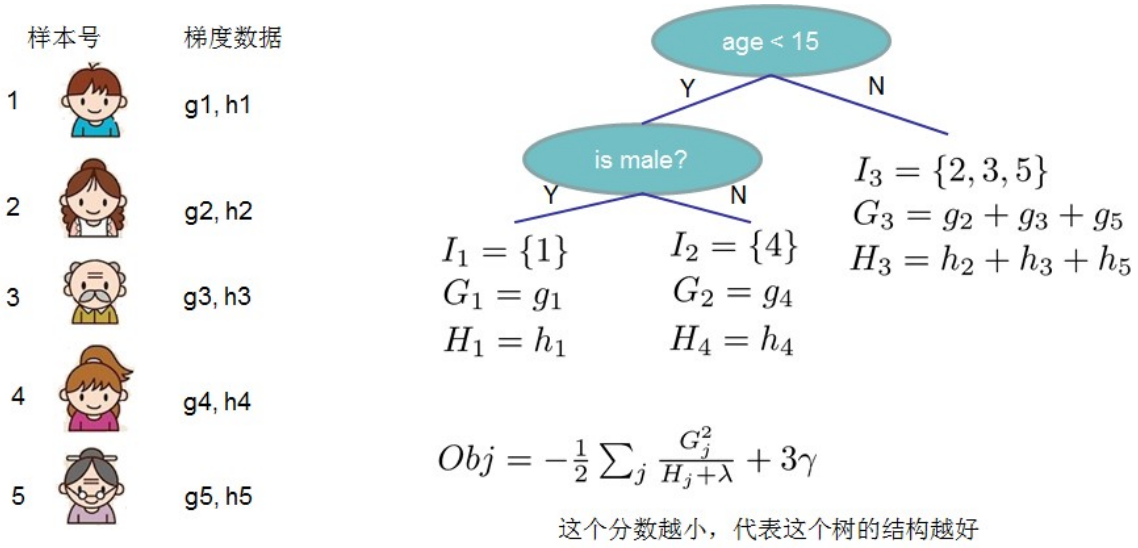
$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

然后把 w_j 最优解代入得到：

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

4.3 打分函数计算

Obj代表了当我们指定一个树的结构的时候，我们在目标上面最多减少多少。我们可以把它叫做结构分数(structure score)



4.3.1 分裂节点

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两种分裂节点的方法

（1）枚举所有不同树结构的贪心法

现在的情况是只要知道树的结构，就能得到一个该结构下的最好分数，那如何确定树的结构呢？

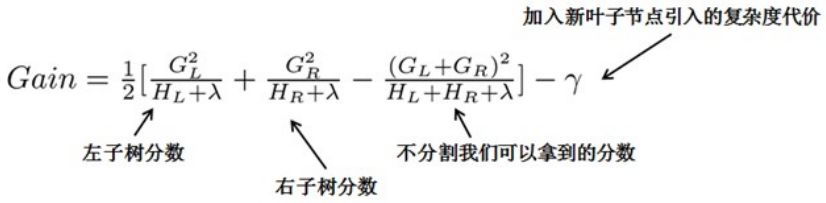
一个想当然的方法是：不断地枚举不同树的结构，然后利用打分函数来寻找出一个最优结构的树，接着加入到模型中，不断重复这样的操作。而再一想，你会意识到要枚举的状态太多了，基本属于无穷种，那咋办呢？

我们试下贪心法，从树深度0开始，每一节点都遍历所有的特征，比如年龄、性别等等，然后对于某个特征，**先按照该特征里的值进行排序，然后线性扫描该特征进而确定最好的分割点**，最后对所有特征进行分割后，我们选择所谓的增益Gain最高的那个特征，而Gain如何计算呢？

还记得4.2节最后，我们得到的计算式子吧？

$$Obj = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j+\lambda} + \gamma T$$

换句话说，目标函数中的G/(H+λ)部分，表示着每一个叶子节点对当前模型损失的贡献程度，融合一下，得到Gain的计算表达式，如下所示：



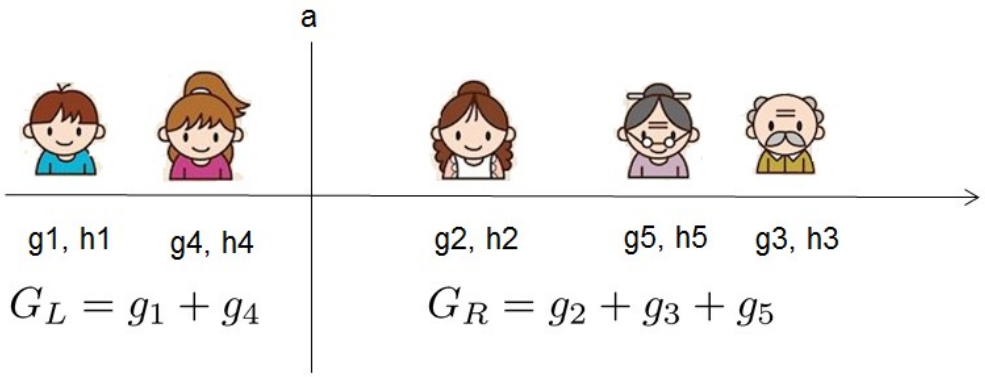
第一个值得注意的事情是“对于某个特征，先按照该特征里的值进行排序”，这里举个例子。

比如设置一个值a，然后枚举所有x < a、a ＜x这样的条件（**x代表某个特征比如年龄age，把age从小到大排序：假定从左至右依次增大，则比a小的放在左边，比a大的放在右边**），对于某个特定的分割a，我们要计算a左边和右边的导数和。

比如总共五个人，按年龄排好序后，一开始我们总共有如下4种划分方法：

1. 把第一个人和后面四个人划分开
2. 把前两个人和后面三个人划分开
3. 把前三个人和后面两个人划分开
4. 把前面四个人和后面一个人划分开

接下来，把上面4种划分方法全都各自计算一下Gain，看哪种划分方法得到的Gain值最大则选取哪种划分方法，经过计算，发现把第2种划分方法“前面两个人和后面三个人划分开”得到的Gain值最大，意味着在一分为二这个第一层层面上这种划分方法是最合适的。



换句话说，对于所有的特征x，我们只要做一遍从左到右的扫描就可以枚举出所有分割的梯度和GL和GR。然后用计算Gain的公式计算每个分割方案的分数就可以了。

然后后续则依然按照这种划分方法继续第二层、第三层、第四层、第N层的分裂。

第二个值得注意的事情就是引入分割不一定会使得情况变好，所以我们有一个引入新叶子的惩罚项。优化这个目标对应了树的剪枝，当引入的分割带来的增益小于一个阀值γ 的时候，则忽略这个分割。

换句话说，当引入某项分割，结果分割之后得到的分数 - 不分割得到的分数得到的值太小（比如小于我们的最低期望阀值γ），但却因此得到的复杂度过高，则相当于得不偿失，不如不分割。即做某个动作带来的好处比因此带来的坏处大不了太多，则为避免复杂 多一事不如少一事的态度，不如不做。

相当于在我们发现“分”还不如“不分”的情况下后（得到的增益太小，小到小于阈值γ），会有2个叶子节点存在同一棵子树上的情况。

下面是论文中的算法

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

（2）近似算法

主要针对数据太大，不能直接进行计算

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ *to* m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ *to* m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

就职于Google的读者crab6789点评：

> 把样本从根分配到叶子结点，就是个排列组合。不同的组合对应的cost不同。求最好的组合你就要try，一味穷举是不可能的，所以才出来贪婪法。不看从头到尾 就看当下节点怎么分配最好。这才有了那个exact greddy方法，后来还想加速才有了histogram的做法。

4.4 小结：Boosted Tree Algorithm

总结一下，如图所示

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

一阶　　二阶

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

贪心算法寻找切分点，生产每一轮新的树

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$

称之为：缩减因子
同样为了避免过拟合

  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

咱们来再次回顾整个过程。

如果某个样本label数值为4，那么第一个回归树预测3，第二个预测为1； 另外一组回归树，一个预测2，一个预测2，那么倾向后一种，为什么呢？前一种情况，第一棵树学的太多，太接近4，也就意味着有较大的过拟合的风险。

OK，听起来很美好，可是怎么实现呢，上面这个目标函数跟实际的参数怎么联系起来，记得我们说过，回归树的参数：

1. 选取哪个feature分裂节点呢
2. 节点的预测值（总不能靠取平均值这么粗暴不讲道理的方式吧，好歹高级一点）

最终的策略就是：贪心 + 最优化（对的，二次最优化）。

通俗解释贪心策略：就是决策时刻按照当前目标最优化决定，说白了就是眼前利益最大化决定，“目光短浅”策略。

这里是怎么用贪心策略的呢，刚开始你有一群样本，放在第一个节点，这时候T=1，w多少呢，不知道，是求出来的，这时候所有样本的预测值都是w,带入样本的label数值，此时loss function变为

$$L(\phi) = \sum_i l(w - y_i) + \gamma + \frac{1}{2}\lambda||w||^2$$

- 如果这里的l(w−yi)误差表示用的是平方误差，那么上述函数就是一个关于w的二次函数求最小值，取最小值的点就是这个节点的预测值，最小的函数值为最小损失函数。
- 本质上来讲，这就是一个二次函数最优化问题！但要是损失函数不是二次函数咋办？泰勒展开，不是二次的想办法近似为二次。

接着来，接下来要选个feature分裂成两个节点，变成一棵弱小的树苗，那么需要：

1. 确定分裂用的feature，how？最简单的是粗暴的枚举/穷举（嗯，够粗暴），然后选择loss function效果最好的那个；
2. 如何确立节点的w以及最小的loss function，大声告诉我怎么做？对，二次函数的求最值（计算二次的最值一般都有固定套路，即导数等于0的点）。所以，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。

在分裂的时候，你可以注意到，每次节点分裂，loss function被影响的只有这个节点的样本，因而每次分裂，计算分裂的增益（loss function的降低量）只需要关注打算分裂的那个节点的样本。

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。

以下便为设定的阈值

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

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，是不是贪心策略？

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

1. 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γ（即正则项里叶子节点数T的系数）；
2. 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；
3. 当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；
4. 貌似看到过有树的最大数量的...

6 参考文献与推荐阅读

1. xgboost原始论文：https://arxiv.org/pdf/1603.02754v1.pdf

2. xgboost作者讲义PPT：https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
3. XGBoost 与 Boosted Tree
4. xgboost原理：https://blog.csdn.net/a819825294/article/details/51206410
5. xgboost入门与实战（原理篇）：https://blog.csdn.net/sb19931201/article/details/52557382
6. CART的Wikipedia
7. 集成学习（Ensemble Learning)
8. 浅谈集成学习：Boosting与随机森林
9. 从决策树学习谈到贝叶斯分类算法
10. 决策树（三）--完整总结（ID3，C4.5，CART,剪枝，替代）兼源码剖析
11. 一文读懂机器学习大杀器XGBoost原理
12. 为什么在实际的 kaggle 比赛中 gbdt 和 random forest 效果非常好？
13. 通俗、有逻辑的写一篇说下Xgboost的原理，供讨论参考
14. 七月在线机器学习第8期第4课 决策树、随机森林、GBDT、xgboost
15. xgboost的一些核心参数
16. 怎样通俗的理解泰勒级数？：https://www.zhihu.com/question/21149770
17. xgboost 为何需要泰勒二阶展开：https://www.julyedu.com/question/big/kp_id/23/ques_id/990
18. 最通俗的梯度提升树(GBDT)原理
19. ID3、C4.5、CART、随机森林、bagging、boosting、Adaboost、GBDT、xgboost算法总结
20. XGBoost二阶泰勒展开公式推导
21. 泰勒公司Wikipedia
22. 七月在线集6学员超级玛丽写的xgboost笔记：一文带你读懂xgboost原理
23. 在线编辑LaTeX公式：http://private.codecogs.com/latex/eqneditor.php?lang=zh-cn

后记

终于大致搞懂了这个经常刷屏的xgboost，再次印证我之前说过的一句话：当你学习某个知识点感觉学不懂时，十有八九不是你不够聪明，十有八九是你所看的资料不够通俗、不够易懂（如果还是不行，问人）。

希望阅读此文的你，也有同样的感受。

以下的本文的改进过程，供理解上参考：

1. 8.4上午第一版，通过一个通俗易懂的年龄预测例子介绍gbdt，因为gbdt是理解xgboost的基础；
2. 8.4下午第二版，xgboost的推导里公式很多，初学者很容易陷进去，后通过抓住xgboost的核心：目标函数，梳理清晰xgboost的脉络框架；
3. 8.5上午第三版，优化了决策树的介绍部分，比如增加对信息增益的介绍；
4. 8.5下午第四版，优化大部分公式的显示，比如之前是纯文本显示，现改成LaTeX图片显示；
5. 8.6上午第五版，优化对booting集成学习的介绍，已让全文更循序渐进；
6. 8.6晚上第六版，规范xgboost目标函数的公式表示，并梳理全文多处细节、公式；
7. 9.1上午第七版，完善4.3.1节中xgboost树的分裂划分方式，以更清晰；
8. 19年1.9第八版，完善4.3.1节中关于分裂节点的描述，以让逻辑更清晰、行文更通俗；
9. 19年1.10第九版，第3部分增加一个预测年龄的例子，以更通俗化解释GBDT；
10. 19年1.14第十版，把泰勒二阶展开和xgboost 目标函数的对应关系写清楚，从而理解更顺畅。

看完全文后，你会发现理解xgboost有三个关键点：①4.2.1节中，理清楚xgboost目标函数各项和泰勒展开二项的一一对应关系，②4.2.2节中对计算数的得分w时使用的一个数学技巧，③4.3.1节中所示的树的分裂算法。理清了这三点，则理解xgboost不再有障碍。

July、二零一八年八月六日晚上~二零一九年一月十四日晚上。