介绍 Boosted Tree

`BoostedTree` `GBDT`

本文主要是根据 陈天奇 大佬在 2014 年的一篇 paper：Introduction to Boosted Tree 写的，更准确说，是对那篇 paper 的翻译，并加上了一些自己的理解。paper 可以在 github 的 pdf 文件夹找到。

提纲

- 回顾监督学习的重要概念
- 回归树和集成（我们正在学习什么）
- 梯度提升（我们如何学习）
- 总结

1、监督学习的重要元素

- 符号： $x_i \in R^d$ 第 i 个训练实例
- 模型：根据给出的 x_i 如何做出预测 $\hat{y}_i$
  - 线性模型： $\hat{y}_i = \sum_j w_i x_{ij}$ （包括 线性/logistic 回归）
  - 预测分数 $\hat{y}_i$ 根据任务不同可以有不同的解释
    - 线性回归：$\hat{y}_i$ 是预测分数
    - Logistic 回归： $\frac{1}{1+e^{-\hat{y}_i}}$ 预测实例为正例的概率
    - 其他方面...例如，在排名中，$\hat{y}_i$ 可以是排名分数
- 参数：我们需要从数据中学习的东西
  - 线性模型： $\Theta = \{w_j | j = 1, \dots, d\}$
- 目标函数无处不在：

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

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

第一个红框：训练损失/误差函数，衡量我们的模型有多拟合我们的训练数据。
第二个红框：正则化项，衡量模型的复杂度/惩罚复杂模型。

- 训练数据上的损失函数：$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$
  - Square loss（平方损失）：$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Logistic loss（逻辑损失）：$l(y_i, \hat{y}_i) = y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i})$
- 正则化项：模型有多复杂？
  - L2 范式：$\Omega(w) = \lambda \|w\|^2$
  - L1 范式（lasso）：$\Omega(w) = \lambda \|w\|_1$

1.1、将已知知识放到上下文中

- 岭回归： $\sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|^2$

    - 线性模型，平方误差，L2 正则化
- Lasso：$\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda\|w\|_1$
    - 线性模型，平方误差，L1 正则化
- Logistic 回归：

$$\sum_{i=1}^{n}[y_i ln(1+e^{-w^T x_i}) + (1-y_i) ln(1+e^{w^T x_i})] + \lambda\|w\|^2$$

    - 线性模型，logistic 损失，L2 正则化
- 模型，参数，目标之间的概念分离也为您带来了工程上的好处。
    - 考虑一下如何在岭回归和 logistic 回归中实现 SGD 。

1.2、目标和偏差差异权衡

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

Training Loss measures how well model fit on training data

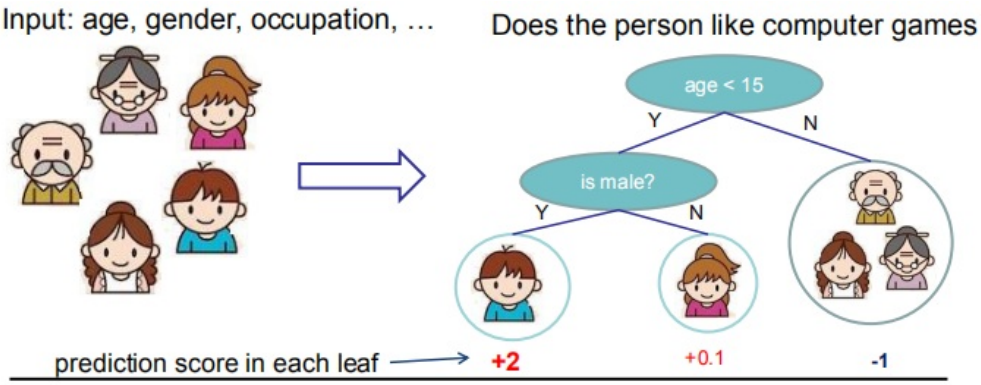
Regularization, measures complexity of model

- 为什么我们要在目标函数中包含两个组成部分？
- 优化训练损失鼓励预测模型
    - 拟合训练数据至少可以让你接近有希望接近底层分布的训练数据
- 优化正则化项鼓励简单的模型
    - 较为简单的模型往往在未来的预测中具有较小的方差，使得预测稳定

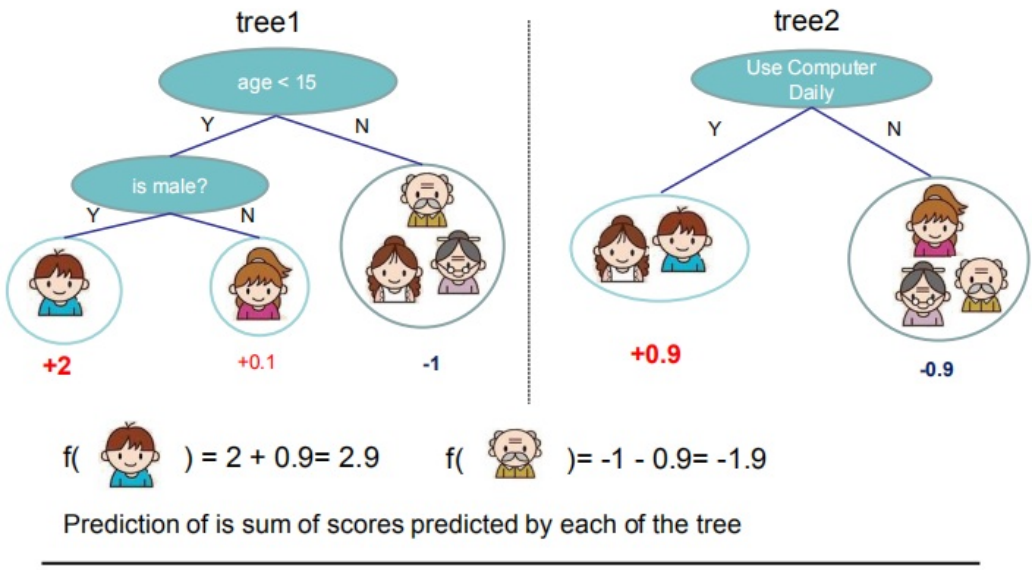
2、回归树和集成（我们正在学习什么？）

2.1、回归树（CART）

- 回归树（也称为分类回归树（classification and regression tree））
    - 决策规则与决策树相同
    - 在每个叶子值中包含一个分数



2.2、回归树集成



预测结果是每个树预测结果的分数之和。

2.3、树集成方法

- 应用非常广泛，可以参考 GBM，随机森林等...
  - 几乎一半的数据挖掘竞赛是通过使用树集成方法的一些变种而获得的。
- 输入缩放不变，因此你不需要很认真的特征归一化。
- 学习特征之间的高阶交互。
- 可扩展，并在工业上使用。

2.4、深入讲解：模型和参数

- 模型：假设我们有 K 棵树

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

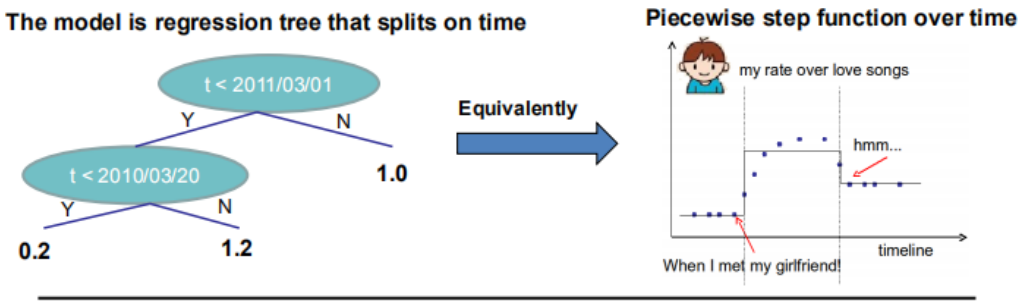
其中 F 表示包含所有回归树的函数的空间。

考虑一下：回归树是一种将属性映射到分数的函数。

- 参数
  - 包括每棵树的结构，以及树叶中的得分
  - 或者简单地使用函数作为参数
  - 我们不是来学习权重，而是学习函数（tree）

2.5、学习单变量树

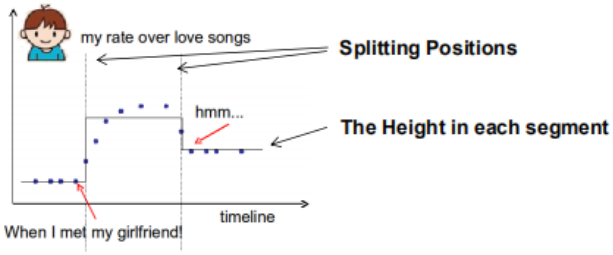
- 我们怎么学习函数？
- 定义目标函数（损失，正则化），并优化它！
- 示例：
  - 考虑单输入 t （时间）上的回归树
  - 我想预测一下我是否喜欢浪漫音乐



上图，左边是根据时间分裂的回归树，右边是一个在时间变量上的阶梯函数

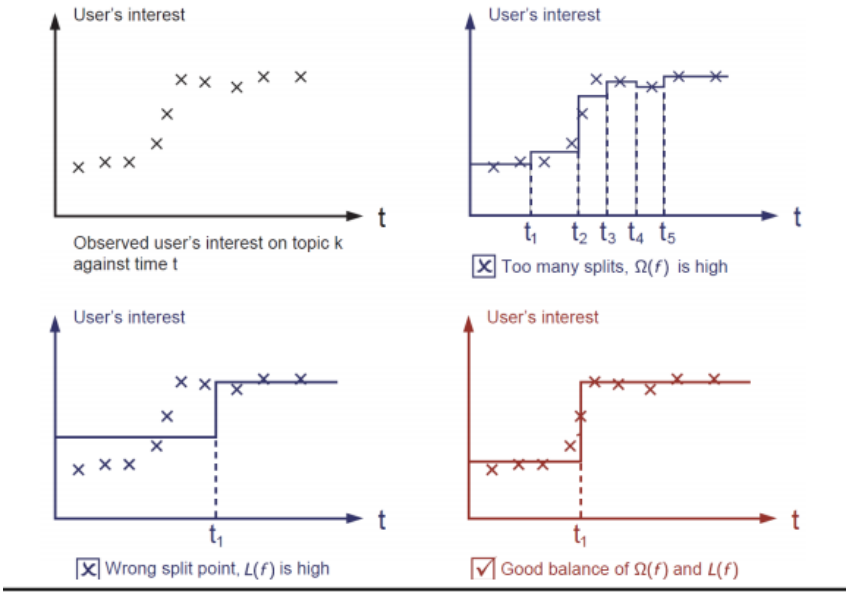
2.6、学习一个阶梯函数

- 我们需要学习的事情



- 单变量回归树的目标（阶梯函数）
  - 训练损失：这个函数如何拟合这些点？
  - 正则化项：我们如何定义函数的复杂性？
    - 分裂点的数量，每个分段的高度的 l2 范数？

2.7、学习阶梯函数（视觉直观上）



上面的 4 个图，每个图的横纵坐标的定义是一样的，横坐标是时间，纵坐标是用户的兴趣。只有右下角的阶梯函数是学习的最好的。

2.8、接下来我们说重点：树集成的目标

- 模型：假设我们有 K 棵树

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

- 目标

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

$l(y_i, \hat{y}_i)$ 是训练损失， $\sum_{k=1}^{K} \Omega(f_k)$ 是树的复杂程度。

- 定义 Ω 的几种可能的方式？
  - 树的节点个数，深度
  - 叶子权重的 L2 范数
  - ...其他的方式稍后详述

2.9、目的与启发式

- 当你说到（决策）树时，通常说的是启发式的。
  - 按信息熵分割数据
  - 对树进行剪枝操作
  - 最大深度
  - 平滑叶子的值
- 大多数启发式方法都能很好地映射到目标上，以正式（客观）的观点让我们知道我们正在学习什么
  - 信息熵 -> 训练损失
  - 剪枝操作 -> 由 #node 定义的正则化
  - 最大深度 -> 对函数空间的约束
  - 平滑叶子的值 -> 叶子权重的 L2 正则化

2.10、回归树并不只是做回归！

- 回归树集成定义了如何来预测分数，可以用于
  - 分类，回归，排名...
  - ....
- 这一切都取决于你如何定义你的目标函数！
- 到目前为止，我们已经知道了：
  - 使用 Square loss（平方损失） $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
    - 将会使用常见的梯度提升算法
  - 使用 Logistic loss（logistic 损失） $l(y_i, \hat{y}_i) = y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{y_i})$
    - 将使用 LogitBoost

3、Gradient Boosting（梯度提升）（我们怎么来学习）

3.1、千万要记住的一节

- 偏差-方差权衡无处不在
- loss（损失） + regularization objective pattern（正则化目标模式）适用于回归树学习（函数学习）
- 我们需要的是可以预测并且尽可能简单的函数
- 这定义了我们想要学习到的东西（目标，模型）。
- 但是我们应该怎么学习？
  - 这是下一节中我们要介绍的东西。

3.2、那么我们怎么学习呢？

- 目标函数：$\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in F$
- 我们不能使用 SGD 等方法来找到 f（因为它们是树，而不是数值向量）
- 解决方案：附加训练（Boosting）
  - 从不断的预测开始，每次增加一个新的函数

$$\hat{y_i}^{(0)} = 0$$
$$\hat{y_i}^{(1)} = f_1(x_i) = \hat{y_i}^{(0)} + f_1(x_i)$$
$$\hat{y_i}^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y_i}^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{(t-1)} + f_t(x_i)$$

注：最后一个表达式中，$\hat{y_i}^{(t)}$ 表示模型处在训练阶段的第 t 阶段，$\hat{y_i}^{(t-1)}$ 表示在上一轮中添加的保留函数，$f_t(x_i)$ 表示新函数。

3.3、附加训练

- 我们怎么决定添加哪个 f ？
  - 优化目标函数！！
- 在第 t 轮迭代中的预测是 $\hat{y_i}^{(t)} = \hat{y_i}^{(t-1)} + f_t(x_i)$ ，这里的 $f_t(x_i)$ 是我们需要在第 t 次迭代中决定的。

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

  上式中，$\sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t)$ 的目标：找到 f_t 来最小化它

- 考虑一下平方损失：

$$Obj^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y_i}^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) + const = \sum_{i=1}^{n} \left[2(\hat{y_i^{(t-1)}} - y_i) f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + c$$

  上式中，$2(\hat{y_i^{(t-1)}} - y_i)$ 通常被称为前一轮的残差。

3.4、泰勒展开损失近似

- 目标函数 $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$
  - 除了平方损失的情况，这看起来似乎还是很复杂
- 对目标函数进行泰勒展开

- 召回 $f(x+\Delta x) \simeq f(x)+f'(x)\Delta x+\frac{1}{2}f''(x)\Delta x^2$
- 定义 $g_i=\partial_{\hat{y}_i^{(t-1)}}l(y_i,\hat{y}_i^{(t-1)}), h_i=\partial_{\hat{y}_i^{(t-1)}}{}^2 l(y_i,\hat{y^{(t-1)}})$

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

- 如果您对这个推导感觉不那么舒服，那么请看一下平方损失

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

- 与我们上一章节讲到的东西做一下对比

3.5、我们的新目标

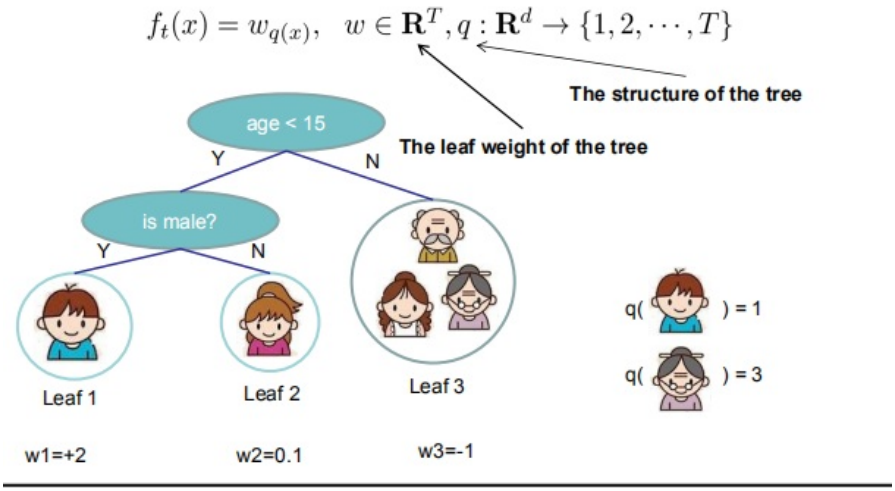
- 移除了常数项的目标函数

$$\sum i=1n\left[g_i f_t(x_i)+frac12h_i f_t^2(x_i)\right]+\Omega(f_t)$$

  - 其中 $g_i=\partial_{\hat{y}_i^{(t-1)}}l(y_i,\hat{y}_i^{(t-1)}), h_i=\partial_{\hat{y}_i^{(t-1)}}{}^2 l(y_i,\hat{y}_i^{(t-1)})$
- 为什么我们花费很大的努力来获得并优化目标，为什么我们不多种几棵树呢？
  - 理论上的好处：知道我们在学习什么，知道我们使什么收敛
  - 工程上的好处：重新了解监督学习的要素
    - g_i 和 h_i 来自损失函数的定义
    - 函数的学习只是取决于 g_i 和 h_i 的目标函数
    - 想想当你被要求实现提升树来处理平方损失和 logistic 损失时，你如何分离你的代码的模块

3.6、细化树的定义

- 我们通过树叶中的分数向量来定义树，以及将实例映射到树叶的树叶索引映射函数



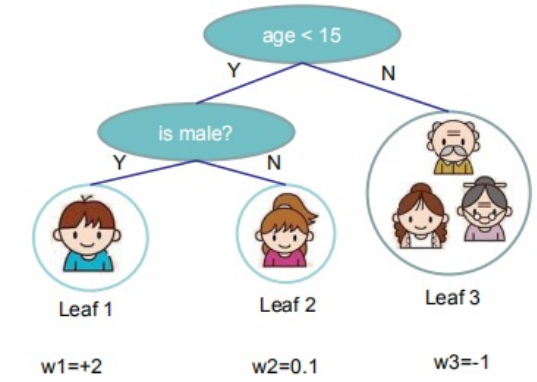
其中 w 是树的叶子的权重，q 是树的结构

3.7、定义树的复杂度（cont'）

定义复杂度如（这不是仅有的可能的定义形式）

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

Number of leaves　　L2 norm of leaf scores



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

3.8、重新审视目标

- 在叶子 j 中定义实例集如 $I_j = i|q(x_i) = j$
- 重新分组每个叶子的目标

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

- 这是 T 独立二次函数的和

3.9、结构分数

- 关于单变量二次函数的两个事实

$$argmin_x\ Gx + \frac{1}{2}Hx^2 = -\frac{G}{H},\ H > 0 \quad \min_x\ Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- 我们定义 $G_j = \sum_{i\in I_j} g_i\ \ H_j = \sum_{i\in I_j} h_i$

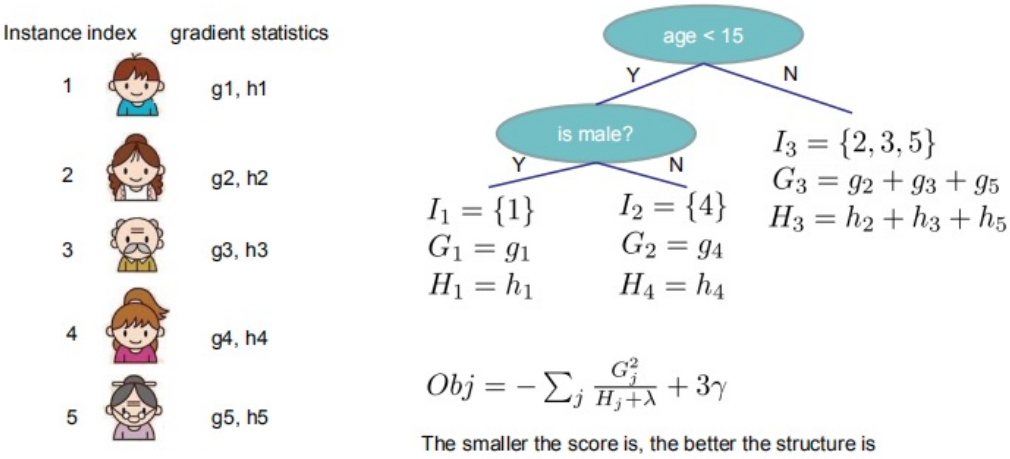
$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T \\ &= \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T \end{aligned}$$

- 假设树的结构 (q(x)) 是固定的，每个叶子的最优权重，以及由此产生的目标值

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

This measures how good a tree structure is!

3.10、结构分数计算



3.11、单个树的搜索算法

- 枚举可能的树的结构 q
- 使用评分等式计算 q 的结构评分

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

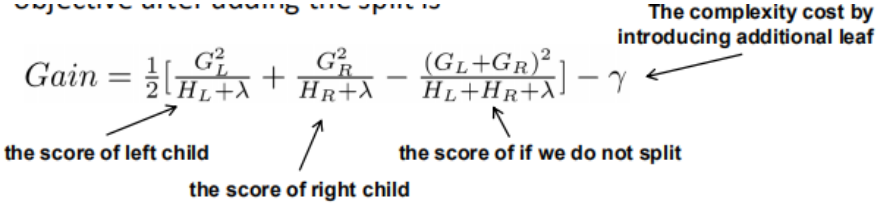
- 找到最佳的树结构，并使用最佳叶权重

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

- 但是...可能有无限种可能的树结构

3.12、树的贪婪学习

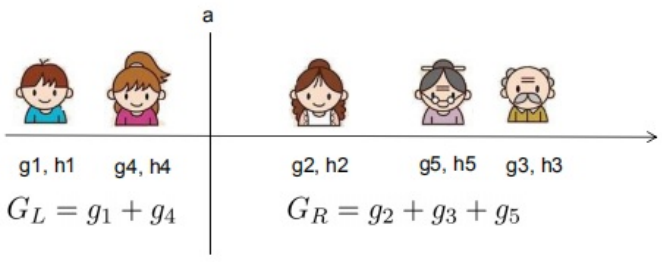
- 在实际实践中，我们贪婪地种树
  - 从深度为 0 的树开始
  - 对于树的每个叶节点，尝试添加一个 split。 添加 split 后的目标变化是



  - 余下的问题：我们如何找到最好的分割？

3.13、找到最佳分割的效率高的方法

- 分割规则 $x_j < a$ 的增益是多少呢？我们假设 x_j 是年龄



- 我们所有需要的是每边的 g 与 h 的和，并且计算

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

- 在排序的实例上从左到右线性扫描足以决定沿着特征的最佳分割

3.14、一种分割算法

- 对于每个节点，列举出所有的特征
  - 对于每个特征，按特征值对实例进行排序
  - 使用线性扫描来确定沿着该特征的最佳分割
  - 沿着所有特征采取最佳的分割解决方案
- 生成一棵深度为 K 的树的时间复杂度
  - 它是 O(n d K log n) ：或者每个级别，需要 O(n log n) 时间进行排序。有 d 个特征，我们需要做 K 级的
  - 这可以进一步优化（优化使用近似或告诉缓存排序的特征）
  - 可以扩展到非常大的数据集上

3.15、分类变量呢？

- 一些树学习算法对分类变量和连续变量分开处理
  - 我们可以很容易地使用我们派生的评分公式来根据分类变量评分
- 实际上我们没有必要单独处理分类
  - 我们可以使用 one-hot 编码来将分类变量编码为数字向量。分配 #categorical 长度向量

$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & otherwise \end{cases}$$

  - 如果有很多类别，向量将是稀疏的，学习算法优先处理稀疏数据

3.16、剪枝和正则化

- 我们回顾一下分割的增益，它可以是负数！

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

  - When **the training loss reduction** is smaller than **regularization**
  - Trade-off between simplicity and predictivness

  上面的两条信息如下：
  - 当训练损失减少小于正则化
  - 在（模型）简单性和预测性之间进行权衡
- 提前停止
  - 如果最好的分割获得的增益是负的，那么就停止分割
  - 但是分割可能对未来的分割有利
- 后剪枝
  - 种植一棵树到达最大深度，递归地修剪所有的叶子分割和负增益

3.17、回顾一下：Boosted Tree 算法

- 在每次迭代中添加一棵新的树

- 在每次迭代开始，计算

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

- 使用统计方法来贪婪地生成一棵树 $f_t(x)$

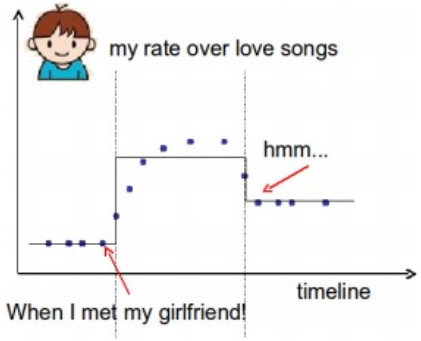
$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

- 将 $f_t(x)$ 添加到模型 $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x)$
  - 通常来说，相反地，我们会这么做 $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ 被称为步长或者收敛，通常设置为 0.1 左右
  - 这意味着我们不会在每个步骤中进行全面的优化，并为未来的迭代过程预留出机会，这有助于防止过拟合

4、总结

4.1、检验您是否真正地了解它的一些问题

- 1、我们如何建立一个提升树分类器来做加权回归问题，使得每个实例具有一个重要的权重？
- 2、回到时间序列问题上，如果我想要学习分段函数。除了自上而下的分割方法之外，还有其他的方法来学习时间分割吗？

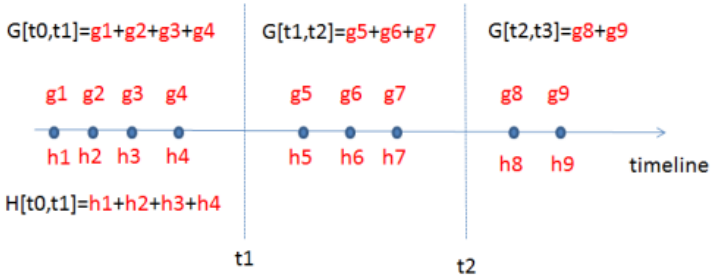


对于上述问题的解答：

- 1、我们如何建立一个提升树分类器来做加权回归问题，使得每个实例具有一个重要的权重？
  - 定义目标，计算 g_i, h_i ，将其反馈到我们用于未加权版本的旧树学习算法

$$l(y_i, \hat{y}_i) = \frac{1}{2}a_i(\hat{y}_i - y_i)^2 \qquad g_i = a_i(\hat{y}_i - y_i) \quad h_i = a_i$$

  - 再次想到模型与目标的分离，理论如何能够更好地组织机器学习工具包
- 2、时间序列问题



- 很重要的一点是分割的结构评分

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

- 自上而下的贪婪算法，与决策树一样
- 自下而上的贪婪，从个别点开始，每个 group 贪婪地合并近邻节点
- 动态规划可以找到这种情况下的最佳解决方案

4.2、小结

- 模型，目标，参数的分离有助于我们理解和自定义我们的模型
- 偏差-方差权衡适用于任何地方，包括函数空间的学习

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

- 我们可以对我们学习的东西和我们如何学习东西惊醒正式的描述。理解清晰可以用来指导更加清晰的实现。

5、引用

- Greedy function approximation a gradient boosting machine. *J.H. Friedman*
  - *First paper about gradient boosting*
- *Stochastic Gradient Boosting. J.H. Friedman*
  - *Introducing bagging trick to gradient boosting*
- *Elements of Statistical Learning. T. Hastie, R. Tibshirani and J.H. Friedman*
  - *Contains a chapter about gradient boosted boosting*
- Additive logistic regression a statistical view of boosting. *J.H. Friedman T. Hastie R. Tibshirani*
  - *Uses second-order statistics for tree splitting, which is closer to the view presented in this slide*
- Learning Nonlinear Functions Using Regularized Greedy Forest. *R. Johnson and T. Zhang*
  - *Proposes to do fully corrective step, as well as regularizing the tree complexity. The regularizing trick is closed related to the view present in this slide*

- Software implementing the model described in this slide: https://github.com/tqchen/xgboost