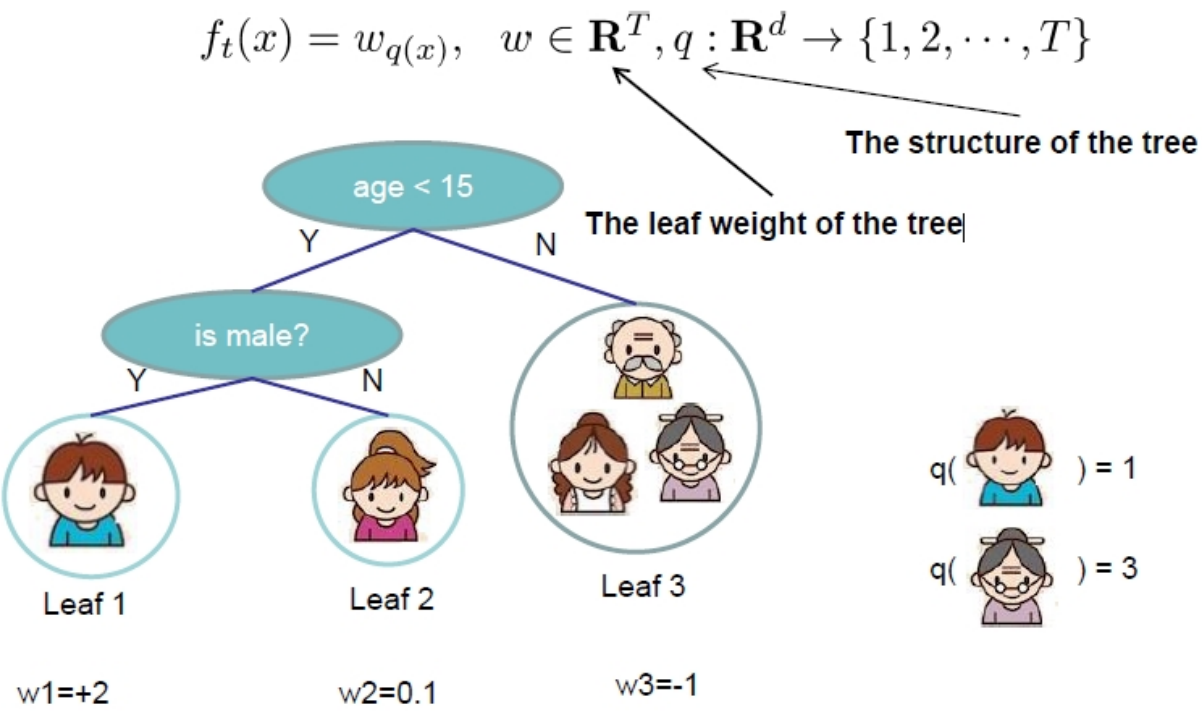
GBDT and XGBoost



参考资料

1. 全面的分析见：GBDT详解，火光摇曳
   - 清晰流畅，十分不错
   - 有一个推导错误

记住我们的目标是求f_k, 它最小化目标函数:

$$
\begin{aligned}
\mathcal{L}_k &= \sum_{i=1}^{N} L(y_i, F_{k-1}(x_i) + \rho f_k(x_i)) + \Omega(f_k) \\
&= \sum_{i=1}^{N} L(y_i, F_{k-1} + \rho f_k) + \Omega(f_k) \\
&\approx \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + \underbrace{\frac{\partial L(y_i, F_{k-1})}{\partial F_{k-1}}}_{:=g_i} f_k + \frac{1}{2} \underbrace{\frac{\partial^2 L(y_i, F_{k-1})}{\partial F_{k-1}^2}}_{:=h_i} f_k^2 \right) + \Omega(f_k) \\
&= \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + g_i f_k + \frac{1}{2} h_i f_k^2 \right) + \Omega(f_k) \\
&= \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2
\end{aligned}
$$

Wrong

fk = \sum_j^J(b_j * I(x_i \in R_j)

跳过这两步，后年的推导是对的

整理出和$\{R_j\}_1^J$, $\{b_j\}_1^J$有关的项:

$$
\begin{aligned}
\mathcal{L}_k(\{b_j\}_1^J, \{R_j\}_1^J) &= \sum_{i=1}^{N} \left(g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2 \\
&= \sum_{x_i \in R_j} \left(g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2
\end{aligned}
$$

2. Tianqi Chen 提供的资料Blog : XGBoost 与 Boosted Tree and PPT: Introduction to Boosted Trees
    - 从有监督学习出发
        > - supervised learnging： model+loss function+learning method
        > - 其中loss function 使用统一的形式 $obj(\Theta) = L(\Theta) + \Omega(\Theta)$，引导出Ridge\Lasso\Logistic regression
    - 到 具体使用的模型 回归树（regresssion tree, i.e., CART） 和 集成方法
    - 再到 求解模型参数的方法 Gradient Boosting
3. GBDT算法原理与系统设计简介
    - 逻辑连贯。涉及大量细节，包括taylor expansion、gradient descent
    - 对比了gradient descend and gradient boosting
        > 同样对比了Newton`s method and Newton boosting, 但是作者理解可能有误。
        >
        > Netwton method使用一阶梯度和二阶海森更新参数，Newton boosting 应该也会用到海森矩阵. And XGBoost computes the second partial derivative of the (element-wise) loss function w.r.t. the predicted label. question on github
    - 提到了内存占用更小、计算速度更快的LightGBM方法

模型+损失函数+学习算法

GBDT=GBRT=MART: Boosted Tree

model

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i) \quad f_t \in \mathcal{F},$$

where

- $f_t(x) = \alpha_t h_t(x; w_t)$, α 是每棵树的权重，h_t 为第t棵树: w_t是回归数的参数(节点怎么分，结构). IF GBDT model
- $f_t(x) = w_{q(x)}, \quad x \in \mathbb{R}^d, q : \mathbb{R}^d \to \{1, 2, \ldots, L\}$ IF XGBoost model
  - q is the structure of the tree
  - w is the leaf weight of the tree
  - L is the number of nodes of the tree
- $\mathcal{F}$ is a space of functions containing all Regression trees

Loss function

$$Obj = \sum_{i}^{n} \underbrace{l(y_i, \hat{y}_i)}_{training\_loss} + \sum_{t=1}^{T} \underbrace{\Omega(f_t)}_{complexity\_of\_the\_tree}$$

Gradient Boosting

Boosting = Additive Training

1. GBDT

   1.1 最早的GBDT模型[1] 没有正则化项，其目标函数为:

   $$F^* = \operatorname{argmin} \sum_{i=1}^{N} l(y_i, F_{t-1} + f_t)$$

   - 其中预测值为:

   $$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i) = F_{t-1} + f_t$$

   1.2 算法过程如下:

   > 求解loss funciton 在 F_{t-1}的导数，找到其值下降最快的地方。然后第t棵树沿着这个方向构建就行了。

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_{w} \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

2. XGBoost

2.1 目标函数为：

$$Obj^{(t)} = \sum_{i}^{n} \underbrace{l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))}_{training\_loss} + \underbrace{\Omega(f_t)}_{complexity\_of\_the\_tree} + \text{constant}$$

- 其中预测值为：

$$\hat{y}_i^t = \sum_{t=1}^{T} f_t(x_i) = \hat{y}_i^{t-1} + f_t(x_i)$$

- 正则化项为：

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

2.2 算法过程

- 对目标函数做二阶泰勒展开近似, 外加一些变换

$$
\begin{aligned}
Obj^{(t)} &= \sum_i^n l\left(y_i,\ \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + \mathrm{c} \\
&\approx \sum_i^n \left\{ l\left(y_i,\ \hat{y}_i^{(t-1)}\right) + \left.\frac{d\, l\left(y_i, \hat{y}_i\right)}{d\, \hat{y}_i}\right|_{\hat{y}_i = \hat{y}_i^{(t-1)}} f_t(x_i) + \frac{1}{2}\left.\frac{d^2\, l\left(y_i, \hat{y}_i\right)}{d\, \hat{y}_i^2}\right|_{\hat{y}_i = \hat{y}_i^{(t-1)}} f_t^2(x_i) \right\} + \Omega(f_t) + \mathrm{c} \\
&= \sum_i^n \left\{ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right\} + \Omega(f_t) + \mathrm{c} \ \leftarrowtail\ l\left(y_i,\ \hat{y}_i^{(t-1)}\right) \textit{ is constant} \\
&= \sum_i^n \left\{ g_i w_{q(x)} + \frac{1}{2} h_i w_{q(x)}^2 \right\} + \Omega(f_t) + \mathrm{c} \ \leftarrowtail\ I_j = \{i | q(x_i) = j\} \text{ 节点}j\text{上的全部样本} \\
&= \sum_i^n \left\{ g_i \sum_{i \in I_j} w_j + \frac{1}{2} h_i \sum_{i \in I_j} w_j^2 \right\} + \Omega(f_t) + \mathrm{c} \ \leftarrowtail\ w_{q(x)} = \sum_{i \in I_j} w_j,\ w_{q(x)}^2 \text{就是把相应节点值的平方加起来} \\
&= \sum_i^n \left\{ g_i \sum_{i \in I_j} w_j + \frac{1}{2} h_i \sum_{i \in I_j} w_j^2 \right\} + \gamma T + \lambda \frac{1}{2} \sum_{i=1}^T w_j^2 + c \ \leftarrowtail\ \sum_i^n \left(g_i \sum_{i \in I_j} w_j \right) = \sum_i^n \left(\left(\sum_{i \in I_j} g_i \right) w_j \right) \\
&= \sum_i^n \left\{ \left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right\} + \gamma T + c \\
&= \sum_i^n \left\{ G_j w_j + \frac{1}{2} H_j w_j^2 \right\} + \gamma T + c
\end{aligned}
$$

然后可以得到$f_t(x) = w_{q(x)}$的显示解

$$\min \quad Obj_j^{(t)}$$

得到：

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

> 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)。你可以认为这个就是类似吉尼系数一样更加一般的对于树结构进行打分的函数。

- 构造树的方法

  不断地枚举（贪心算法）不同树的结构，利用这个打分函数来寻找出一个最优结构的树，加入到我们的模型中，再重复这样的操作 。分割后的得分值：

$$
\begin{aligned}
Gain &= score\_of\_not\_split - score\_of\_split \\
&= -\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma T' - \left[-\frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} \right) + \gamma (T' + 1) \right] \\
&= \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma
\end{aligned}
$$

  - 当Gain小于0，不需要分裂。Trade-off between simplicity and precictive
  - 得到最优解 $f_t(x)$, 更新模型时具体操作用 $\hat{y}_i^t = \hat{y}_i^{t-1} + \epsilon f_t(x_i)$

> This is aim to prevent overfitting via not do full optimization in each step and thus reverve chance for future rounds

算法过程为：

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

常见问题

https://zhuanlan.zhihu.com/p/33700459

1. 随机森林（random forest）和 GBDT 都是属于集成学习（ensemble learning）的范畴，有什么不同?

- 集成学习下有两个重要的策略 Bagging 和 Boosting，Bagging算法是这样，每个分类器都随机从原样本中做有放回的采样，然后分别在这些采样后的样本上训练分类器，然后再把这些分类器组合起来，简单的多数投票一般就可以，其代表算法是随机森林。Boosting 的算法是这样，它通过迭代地训练一系列的

2. 为什么随机森林的树深度往往大于 GBDT 的树深度?
bias-variance tradeoff: random forest uses bagging and GBDT uses boosting.

- 对于 Bagging 算法来说，由于我们会并行地训练很多不同的分类器的目的就是降低这个方差（variance），因为采用了相互独立的基分类器多了以后，h 的值自然就会靠近。所以对于每个基分类器来说，目标就是如何降低这个偏差（bias），所以我们会采用深度很深甚至不剪枝的决策树。
- 对于 Boosting 来说，每一步我们都会在上一轮的基础上更加拟合原数据，所以可以保证偏差（bias），所以对于每个基分类器来说，问题就在于如何选择 variance 更小的分类器，即更简单的分类器，所以我们选择了

深度很浅的决策树。

3. xgboost相比传统gbdt有何不同?

- 传统GBDT以**CART**作为**基分类器**, xgboost还支持**线性分类器**，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。
- 传统GBDT在优化时只用到一阶导数信息，xgboost则对代价函数进行了二阶泰勒展开，同时用到了**一阶和二阶导数**。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。
- xgboost在代价函数里加入了**正则项**，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性。
- **Shrinkage（缩减）**，i.e., $\hat{y}_i^t = \hat{y}_i^{t-1} + \epsilon f_t(x_i)$, 相当于学习速率（xgboost中的eta）。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（补充：传统GBDT的实现也有学习速率）列抽样（column subsampling）。xgboost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是xgboost异于传统gbdt的一个特性。

4. ensemble methods 的优点

- From viewpoint of Tianqi Chen
  - Learn high order interaction between features do not know
  - Invariant to **scaling of inputs**（伸缩不变性）, so you do not need to do careful features normalization
  - can be scalable（扩展性强，意思是说容易并行）, and are used in industry
- From 机器学习-周志华 在集成学习中，如果个体学习器"好而不同"，那么集成学习的泛化能力就比较好。按照这种思路理论分析的结果就是[Krogh and vEdelsby, 1995]给出的 error-ambiguity decomposition(误差-分歧分解，不太出名). 基本的意思就是 $集成性能 = 个体学习器泛化误差加权均值-个体学习器加权分歧值。个体学习器加权分歧值可以用多样性来度量，增加多样性有这几种方法
  - 数据样本扰动: 改变数据集的分布-Bagging, AdaBoost
  - 输入属性扰动：选择输入的子空间
  - 算法参数扰动：dropout

5. benefits of tree model
   - 可解释性强
   - 有特征组合的作用
   - 有特征选择作用
   - 可自然地处理缺失值
   - 对异常点鲁棒

实践例子

1. Greedy Function Approximation: A Gradient Boosting Machine ↩