[Doc]

[Slides]

summary

- 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 $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$

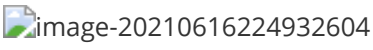
称之为：缩减因子
同样为了避免过拟合

  - 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

原理

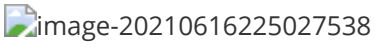
- 每轮生成新的树，该树的生成标准为：损失函数最小化。损失函数代表的意义：保证模型复杂度越低的同时，使预测误差尽可能小。其中模型复杂度包括树的个数以及叶子数值尽可能不极端。 *（这个怎么看，如果某个样本label数值为4，那么第一个回归树预测3，第二个预测为1；另外一组回归树，一个预测2，一个预测2，那么倾向后一种，为什么呢？前一种情况，第一棵树学的太多，太接近4，也就意味着有较大的过拟合的风险）*
- 每一轮产生新的树的时候，其loss是本轮加之前模型与y的差，然后通过泰勒展开做成**t-1轮损失函数对于t-1轮模型即上一轮残差**的一阶导和二阶导与当前树及其平方的乘积

损失函数

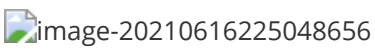


其中l为loss，l为二次则利用二次优化，不是二次则用泰勒展开；Omega为正则项。

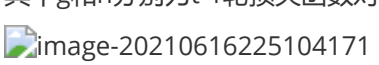
- loss：采用加法策略，第t颗树时：

  所以在添加第t颗树时，需要优化的目标函数为：

  其中g和h分别为t-1轮损失函数对于t-1轮模型的一阶导和二阶导：

  note: 是对谁的导

此处为了简化目标函数，用到了泰勒二阶展开：

- 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 $f(x+\Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$ 保留二次项
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

- *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

https://blog.csdn.net/qq_24852439

loss第一项是真实值与t-1轮预测结果的loss，对t轮不明显，因此删去。所以对于一个模型来说，在每一次优化时，先定义好loss函数，然后计算每一轮loss的一阶导和二阶导即可写出loss，优化即可（每个特征求最佳分裂点，使用最小的特征）。

- 正则项：复杂度：

   image-20210616225141406

  其中w是叶子上的score vector，T是叶子数量

损失函数求解

- 最佳树结构（特征和分裂值）以及叶子节点的预测分数都是通过优化目标函数来得出的，其中叶子节点的预测分数是正则项的一部分；但其实分裂时是通过增益最大化来寻找最佳分裂特征和分裂值的，只有w是通过计算目标函数得出的

  We could further compress the expression by defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$:

$$\text{obj}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T$$

  求出各个叶子节点的最佳值以及此时目标函数的值：

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

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

分裂

- 分裂方式
  - 枚举所有树结构的贪心法（先特征，再分裂点）：
    - 首先，对所有特征都按照特征的数值进行预排序。
    - 其次，在遍历分割点的时候用O(#data)的代价找到一个特征上的最好分割点。
    - 最后，找到一个特征的分割点后，将数据分裂成左右子节点。
  - 优缺点：
    - 优点：精确找到分裂点

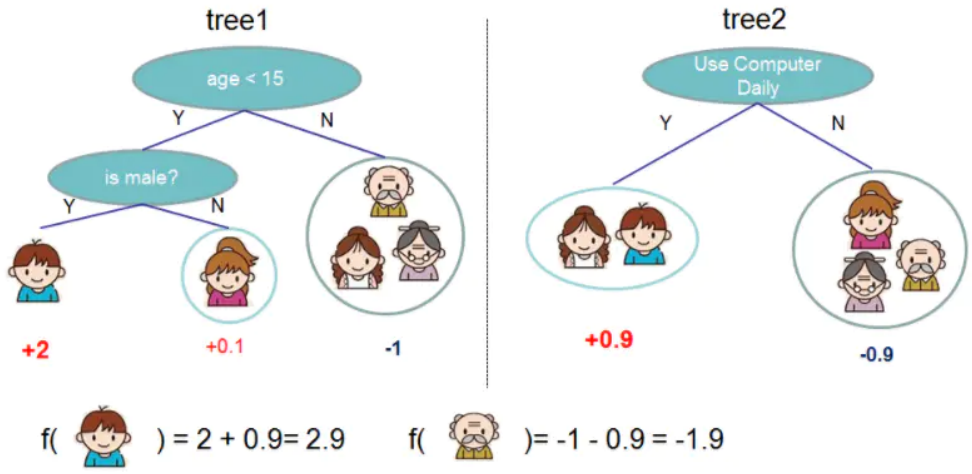
    - 缺点：**空间消耗大**（保存数据的特征值，还保存了特征排序的结果，即2*数据大小）、**时间开销大**（遍历每一个分割点的时候，都需要进行分裂增益的计算）、**cache优化不友好**（预排序后，特征对梯度的访问是一种随机访问，并且不同的特征访问的顺序不一样，无法对 cache 进行优化）
- 分裂的标准：**增益最大化**，每次节点分裂，loss function被影响的只有这个节点的样本，因而每次分裂，计算分裂的**增益**（loss function的降低量）只需要关注打算分裂的那个节点的样本

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

- 终止条件
    - 树的最大数量
    - max_depth
    - 最小增益：当引入的分裂带来的增益小于一个阀值的时候，我们可以剪掉这个分裂，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
    - 当样本权重和小于设定阈值时则停止建树，这个解释一下，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终止

使用

- 每个树的每个节点上的预测值相加



DART Booster

为了解决过拟合，会随机drop trees:

- 训练速度可能慢于gbtree
- 由于随机性，早停可能不稳定

特性

- 算法：二阶导、正则化、自定义loss、缺失数据处理、支持类别特征、早停
- 工程：行采样、列采样、

正则化&行采样&列采样&早停

- 通过最优化求出w，而不是平均值或者多数表决
- 防止过拟合

支持自定义loss

处理缺失值

并行

- **特征间并行**：由于将数据按列存储，可以同时访问所有列，那么可以对所有属性同时执行split finding算法，从而并行化split finding（切分点寻找）
- **特征内并行**：可以用多个block(Multiple blocks)分别存储不同的样本集，多个block可以并行计算-特征内并行

Monotonic Constraints单调性限制

- 一个可选特性:
  会限制模型的结果按照某个特征 单调的进行增减

  也就是说可以降低模型对数据的敏感度，如果明确已知某个特征与预测结果呈单调关系时，那在生成模型的时候就会跟特征数据的单调性有关。

Feature Interaction Constraints单调性限制

- 一个可选特性：
  不用时，在tree生成的时候，一棵树上的节点会无限制地选用多个特征

  设置此特性时，可以规定，哪些特征可以有interaction（一般独立变量之间可以interaction，非独立变量的话可能会引入噪声）
- 好处：
  - 预测时更小的噪声
  - 对模型更好地控制

Instance Weight File

- 规定了模型训练时data中每一条instance的权重
- 有些instance质量较差，或与前一示例相比变化不大，所以可以调节其所占权重

调参

- 通用参数：宏观函数控制。booster/slient/nthread
- Booster参数：控制每一步的booster(tree/regression)。
- 学习目标参数：控制训练目标的表现。

Overfitting

与overfitting有关的参数：

- 学习率
- 直接控制模型复杂度：max_depth, min_child_weight, gamma, lambda, alpha, max_leaf_nodes
- 增加模型随机性以使得模型对噪声有更强的鲁棒性：
  - subsample and colsample_bytree.

过程

- 选择较高的学习速率(learning rate)。XGBoost有一个很有用的函数“cv”，这个函数可以在每一次迭代中使用交叉验证，并返回理想的决策树数量。
- 对于给定的学习速率和决策树数量，进行决策树特定参数调优(max_depth, min_child_weight, gamma, subsample, colsample_bytree)。在确定一棵树的过程中，我们可以选择不同的参数。
- xgboost的正则化参数的调优。(lambda, alpha)。这些参数可以降低模型的复杂度，从而提高模型的表现。
- 降低学习速率，确定理想参数。