Decision Tree and Random Forest and XGBoost

Chunyan Li

Feb 2021-Sept. 5th 2022

1 Introduction

In this article, we will talk about decision tree (DT), Gradient Boosting decision tree (GBDT), random forest (RF) and Extreme Gradient Boosting (XGBoost). The main problem of Decision tree is how to decide the root. As long as one can find a criteria to determine the root, then one can repeat the procedure to decide the nodes of the whole decision tree until meet the stop criteria.

2 Decision Tree

建立决策树/分裂特征的标准是: 尽可能使得各个分裂子集尽可能地纯, 即尽可能让一个分裂子集中待分类地项属于同一个类别. 数学中用于刻画数据'纯度'有三种方法:

- Gini impurity $\leftarrow$ classification tree (CART for classification)
- Information gain $\leftarrow$ classification tree (ID3 and C4.5)
- variance reduction $\leftarrow$ regression tree (CART)

Decision tree models could be classified into two types based on the type of target variable.

- classification trees if the target variable can take a discrete set of values
- regression trees if the target variable can take continuous values

Decision trees have several nice advantages over nearest neighbor algorithms:

1. Once the tree is constructed, the training data doesn't need to be stored. Instead, we can simply store how many points of each label ended up in each leaf-typically these are pure so we just have to store the label of all points
2. decision trees are very fast during test time, as test inputs simply need to traverse down the tree to a leaf-the prediction is the majority label of the leaf
3. decision trees require no metric because the splits are based on feature thresholds and not distances.

2.1 Gini impurity

Gini impurity is used by the CART (Calssification and Regression Tree) algorithm for classification trees which is first introduced by Breiman[?] in 1894. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability p_k of an item with label k being chosen times the probability $\sum_{i\neq k} p_k = 1 - p_k$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

Let $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}, y_i \in \{1, ..., c\}$ be a set of data pairs, c is the number of classes. To compute Gini impurity for a set of items with c classes, suppose $k \in \{1, 2, ..., c\}$ and let $S_k \subseteq S$ where $S_k = \{(\mathbf{x}, y) \in S : y = k\}$ be the set of all inputs with labels k, $S = S_1 \cup ... \cup S_c$, then, $p_k = \frac{|S_k|}{|S|}$ is the fraction of items in S labeled with class k in the set, then

$$G(S) = \sum_{k=1}^{c} \left(p_k \sum_{i\neq k} p_i \right) = \sum_{k=1}^{c} p_k(1 - p_k) = \sum_{k=1}^{c} p_k - \sum_{k=1}^{c} p_k^2 = 1 - \sum_{k=1}^{c} p_k^2 \tag{1}$$

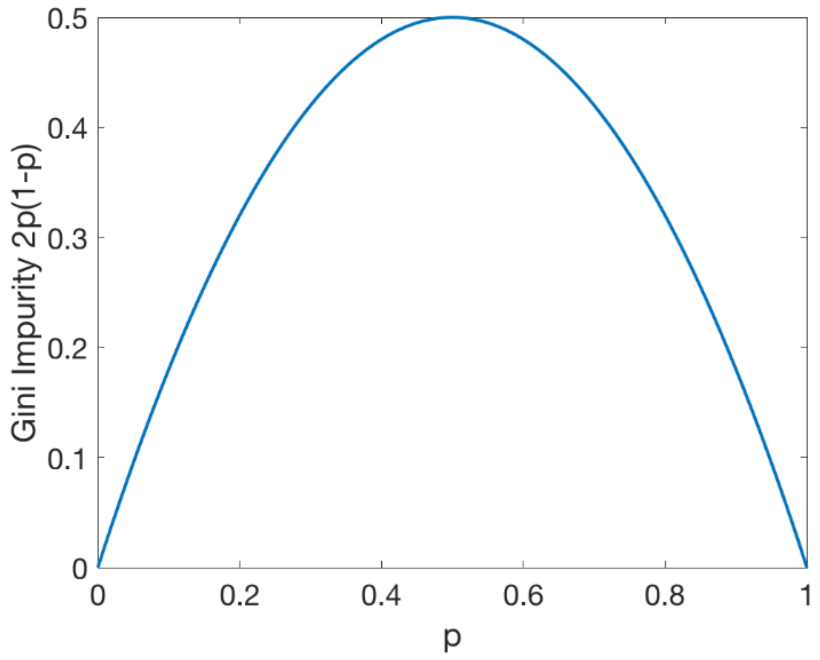

Figure 1: The Gini Impurity Function in the binary case reaches its maximum at $p = 0.5$. A node that has all classes of the same type (perfect class purity) will have $G = 0$, where a node that has a half and half split of classes for a binary classification problem (worst purity) will have a $G = 0.5$.

Gini impurity of a feature with value a_i is

$$G(S, A = a_i) = G(S^i) = 1 - \sum_{k=1}^{c} Pr(k|A = a_i)^2 \tag{2}$$

Gini impurity (Gini index) of a feature A is defined by

$$G(S, A) = \frac{|S^1|}{|S|}G(S^1) + \frac{|S^2|}{|S|}G(S^2) + ... + \frac{|S^N|}{|S|}G(S^N) \tag{3}$$

where

- $S = S^1 \cup ... \cup S^N$
- $S^i \cap S^j = \emptyset$ for $i \neq j$
- $N \leftarrow$ total number of all possible values a_i of feature A
- $S^1 \leftarrow$ the collection of instances that has $A = a_1$
- $\frac{|S^i|}{|S|} \leftarrow$ fraction of inputs/instances that has $A = a_1$

2.1.1 How to build a decision tree using gini impurity

Creating a decision tree involves two steps:

- select best input variables (features) to split
- select best split points on these features until a suitable tree is constructed.

The selection of which input variable to use and the specific split or cut-point is chosen using a greedy algorithm to minimize a cost function (Gini impurity function). Tree construction ends using a predefined stopping criterion, such as a minimum number of training points assigned to each leaf node of the tree, the maximum depth of a tree.

2.1.2 Classification Tree （CART binary tree 二叉树）

The cost function is Gini impurity function. The smaller the impurity, the purer the data. A greedy approach is used to divide the space called recursive binary splitting. This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected. All input variables (features) and all possible split points are evaluated and chosen in a greedy manner.

Algorithm:

1. Let S be the dataset in current node, then, one compute $G(S, A)$ for all features. 对于每一个特征A, 对其可能的每一个取值a_i, 根据样本点对$A = a_j$的测试'Yes' or 'No'将S分割成S^L, S^R, 然后计算$G(S, A = a_j)$
2. 在所有可能的特征A和他们所有可能的切分点a_i中,选择Gini index最小的特征A^*及其对应的切分点a^*作为最优特征和最优切分点, 然后根据二者生成2个子节点,将数据集按照特征和切分点分配到两个子节点中.

2.1.3 Variance Reduction and Regression Tree

In regression case, the key idea is to approximate the regression function between input $\mathbf{x}$ and target label y by piecewise const function.

In regression case, we assume labels are continuous $y_i \in R$, then, the cost function (Impurity) is

$$L(S) = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} (\hat{y} - \bar{y}_S)^2 \leftarrow \text{average squared difference from average label} \tag{4}$$

where $\bar{y}_S = \frac{1}{|S|}\sum_{(\mathbf{x},y)\in S} y \leftarrow$ average label, $\hat{y}$ is prediction. Variance reduction的概念使得我们可以依据如下损失函数来构造二叉树：在保证切分点左右两侧区域的均方差都尽量小的同时，保证两个区域的最小均方差之和也是最小的。

$$\min_{j,s} g(j,s) := \left[\min_{c_1} \sum_{x_i\in\mathbb{R}_L(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i\in\mathbb{R}_R(j,s)} (y_i - c_2)^2\right] \tag{5}$$

计算回归树的步骤:

1. determine the best split feature j and best split point s by solving eq.5. 遍历（traverse）特征j, 对固定j扫描切点s，选择使得上式最小的(j^*, s^*) pairs,
2. 用选定的(j^*, s^*) pairs 对区域进行划分，并且决定相应的输出值: $\mathbb{R}_R(j,s) = \{x|x^{(j)} > s\}$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i\in\mathbb{R}_m(j,s)} y_i \tag{6}$$

   where $x \in \mathbb{R}_m, m = L, R$, $\mathbb{R}_L(j,s) = \{x|x^{(j)} \le s\}$,

决策树可以表示为:

$$f(x) = \sum_{m=1}^{M} \hat{c}_m \cdot \chi(x \in \mathbb{R}_m) \tag{7}$$

决策空间最终被决策树划分为M个区域$\mathbb{R}_1, ..., \mathbb{R}_M$.

求解$\min_{j,s} g(j,s)$的方法:

1. determine (c_1^*, c_2^*) for given (j,s) by taking derivative of eq.8 w.r.t c_m

$$\min_{c_m} \sum_{x_i\in\mathbb{R}_m(j,s)} (y_i - c_m)^2 \tag{8}$$

   so that one can get c_m^* by assigning 0 to derivative,

$$\sum_{x_i\in\mathbb{R}_m(j,s)} -2(y_i - c_m^*) = 0 \tag{9}$$

$$\Rightarrow c_m^* = \frac{1}{N} \sum_{x_i\in\mathbb{R}_m(j,s)} y_i \tag{10}$$

   where $N_m = |\mathbb{R}_m(j,s)| \leftarrow$ 区域$\mathbb{R}_m$中instances的数目, (j,s) is fixed.

2. determine (j^*, s^*) where j is index of feature (finite many) and s is index of possible value of a feature (could be finite many/discrete, could be infinitely many/continuous). 对于给定的一对(j, s)，计算c_1, c_2 by eq.9，然后带入$g(j, s) = c_1 + c_2$，遍历所有(j, s), 选择$\min_{j,s} g(j, s) = \hat{c}_1 + \hat{c}_2$.
   - For discrete feature, one can determine (j^*, s^*) by traversing all possible cases.
   - For continuous feature, one discretizes the continuous feature according to the following rule: (used in both CART and C4.5) m个样本的连续特征A 有m个取值$a_1, ..., a_m$, 则CART取相邻两样本值的平均数作为划分点，一共有$m-1$个，其中第i个splitting point is $s_i = \frac{a_i + a_i + 1}{2}, i = 1, ..., m-1$

Summary of CART:

- CART generates binary tree
- CART needs to determine optimal splitting feature and optimal splitting point
- It is very time consuming for CART to handle discrete variables with too many possible values. For example, if a discrete variable has m possible values, then, there will be $2^m - 2$ possible splitting results (empty set and whole set are useless), one has to compute the gini index for all these cases, to determine the node
- How to deal with continuous features in classification tree? Same discretization as used in regression.
- A feature could be reused in CART

2.2 Information Gain

Information gain is used by ID3, C4.5 and C5.0 tree-generation algorithms. Information gain is based on the concept of entropy and information content from information theory.

2.2.1 Entropy and Information Gain

Entropy: From the above Figure 1, we know $p_1 = p_2 = ... = p_c = \frac{1}{c}$ is the worst case (enemy) since each leaf is equally likely. Prediction is random guessing. Define the impurity as how close we are to uniform distribution. We can use KL divergence to compute the 'closeness'. We expect that p is far away from uniform distribution q. In the discrete case,

$$D_{KL}(p||q) = \sum_{k=1}^{c} p_k log \frac{p_k}{q_k} \tag{11}$$

if $q_k = \frac{1}{c},\ for\ k = 1, ..., c$, then

$$\begin{aligned} D_{KL}(p||\frac{1}{c}) &= \sum_{k=1}^{c} p_k log \frac{p_k}{1/c} \qquad (12) \\ &= \sum_{k=1}^{c} p_k log p_k + \sum_{k=1}^{c} p_k log c \leftarrow log(c) \text{ is a const } \sum_k p_k = 1 \qquad (13) \\ &= \sum_{k=1}^{c} p_k log p_k + log c \qquad (14) \end{aligned}$$

Hence, To maximize the KL divergence is equivalent to minimize entropy:

$$\max_p \sum_{k=1}^{c} p_k log p_k = \min_p - \sum_{k=1}^{c} p_k log p_k \leftarrow \text{Entropy} \tag{15}$$

where p, q are distributions in dataset S.

Information Gain: Information Gain (used in ID 3 algorithm and C4.5 algorithm) is defined based on the entropy and condition entropy.
The entropy of a set S with c classes is defined by

$$H(S) = -\sum_{i=1}^{c} p_i log_2(p_i) \tag{16}$$

Information gain on feature A with value a is defined by

$$IG(S,a) = H(S) - H(S|a) = -\sum_{i=1}^{c} p_i log_2 p_i - \sum_{i=1}^{c} -Pr(i|a) log_2 Pr(i|a) \tag{17}$$

where

- $IG(S,a) \leftarrow$ the information gain of feature A with value a.
- $H(S) \leftarrow$ parent entropy which is a const as long as the parent node is determined
- $H(S|a) \leftarrow$ sum of children entropies (sum over all classes)
- $p_i \leftarrow$ fraction/probability of items in set S labeled with class i
- $Pr(i|a) \leftarrow$ The probability of an instance being labeled class i conditioned on $A = a$

Information gain on feature A is defined by averaing over all the possible values of A:

$$IG(S,A)) := \mathbf{E}_A(IG(S,a)) = H(S) - H(S|A) \tag{18}$$

$$= -\sum_{i=1}^{c} p_i log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{c} -Pr(i|a) log_2 Pr(i|a) \tag{19}$$

where $p(a) \leftarrow$ probability of feature A has value a.
Note that $H(S)$ is a const when the parent node is determined, hence, maximize the information gain of feature A, $IG(S,A)$, is equivalent to minimize the entropy of feature A, $H(S|A)$.

2.2.2 ID3 algorithm for polytree for classification（多叉树）

Information gain is used to decide which feature to split on at each step in building the tree in ID3 algorithm. The ID3 algorithm begins with the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy $H(S)$ or the information gain $IG(S)$ of that attribute/feature. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split or partitioned by the selected attribute A (based on all its values a_i) to produce subsets of the data. (For example, a node can be split into child nodes based upon the subsets of the population whose ages are less than 50, between 50 and 100, and greater than 100.) The algorithm continues to recurse on each subset, considering only attributes never selected before.

Algorithm of ID3:

1. For each feature, compute the Info Gain $IG(S,A)$ induvially, and then then choose the biggest one as the split node A^*.
2. Split the node into N branches based on best split feature A^* which has N possible values.

The stop criteria is:

- All instances belong to the same class in this node, then, this node becomes leaf, and the class is the corresponding label of this leaf.
- There is no more features to be split. This node becomes leaf and the corresponding label is determined by majority voting.

Summary of ID3 algorithm:

- It is hard for ID3 to deal with continuous features since it is very time consuming. So people usually one use ID3 for discrete features.
- ID3 generates classification tree not regression tree.
- every feature/variable only be used once in ID3.

2.3 Gain ratio and C4.5增益率算法

由于信息增益选择分裂属性的方式倾向于选择具有大量取值的属性/特征,比如,一个特征是’student ID’, 即按照该变量划分,每个划分都是纯的, ’student ID’的信息增益是最大值1. 但是按照这个自变量的每个值进行分类的方式没有意义. 所以为了避免这种问题,人们使用信息增益比率(gain ratio)来作为选择最佳分裂属性的标准,从而发展出ID3的进阶版C4.5算法.

2.3.1 Gain ratio

$$\text{Gain ratio}(S, A) = \frac{IG(S, A)}{SI(A)} \tag{20}$$

where

- $SI(A) := -\sum_{v=1}^{N} \frac{|S^v|}{|S|} log_2 \frac{|S^v|}{|S|} \leftarrow A$ 的固有值, splitting information
- $\frac{|S^v|}{|S|} = p(A = v)$, hence, $SI(A) = -\sum_{v=1}^{N} p(A = v) log_2 p(A = v)$
- $N \leftarrow$ number of all possible values of feature A

2.3.2 C4.5 algorithm

1. Compute IG of all features to find the features that has IG larger than average IG as the candidates
2. compute the Gain ratio of candidates, choose the feature which has largest Gain ratio as the best feature to split.
3. split the node into N branches based on the best split feature which has N possible values.
   - For continuous feature, one discretizes the continuous feature according to the following rule: (used in both CART and C4.5) m个样本的连续特征A 有m个取值$a_1, ..., a_m$, 则CART取相邻两样本值的平均数作为划分点，一共有$m - 1$个，其中第i个splitting point is $s_i = \frac{a_i + a_i + 1}{2}, i = 1, ..., m - 1$
   - 连续特征时，C4.5对二分叉，离散特征对应多分叉

Summary of C4.5

- every feature/variable only be used once in C4.5
- C4.5 can eliminate the shortcomings of info Gain
- C4.5 can handle continuous features automatically
- 连续特征：二分叉，离散特征：多分叉。

Remark: 所有算法对应的剪枝策略和stop criteria 都没有介绍。

3 Random Forest

Tree ensemble methods includes 2 techniques:

- Bagging $\leftarrow$ RF
- Boosting $\leftarrow$ AdaBoost and XGBoost

RF is a successful case for tree ensemble method. The basic idea of generate a random forest is to sample data and features with replacement many times (N) and then, build N trees using these sampled subsets in a parallel fashion. The final prediction is made by majority voting.

3.1 Bagging/Bootstrap Aggregating

3.2 Advantage of Bagging and OOB (out of bag error)

3.3 RF

4 Introduction of XGBoost

XGBoost 表示"Extreme Gradient Boosting"，而"Gradient Boosting"起源于Friedman的论文"Greedy function Approximation: A Gradient Boosting Machine". Gradient Boosting = Gradient Descent + Boosting, 而XGBoost 是Gradient Boosting的快速实现，希望极尽计算资源来进行训练。（The gradient boosting algorithm is the top technique on a wide range of predictive modeling problems, and XGBoost is the fastest implementation.）The goal of XGBoost libaray is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate library. XGBoost是一个可扩展，分布式的gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library of regression, classification, and ranking problem.

Figure 2: Caption

4.1 监督学习的元素

XGBoost是用于监督学习问题的所谓监督学习是指，用包含多个特征的训练数据x_i预测对应的标签y_i. 我们先回顾一下监督学习的基本元素，然后再回答什么是树。

4.1.1 模型和参数

监督学习中的模型指的是：假设预测值是输入值（特征）的函数，数学表达式为

$$y_i = f(x_i)$$

一个最常见的例子就是线性模型，假设预测值是各个特征的加权求和，表达式为$\hat{y}_i = \sum_j \theta_j x_{ij}$ 而这个预测值基于任务可以有不同的解释，比如回归，或者分类任务。比如说，在logistic regression中，这个预测值可以通过logistic transform 来得到positive class的概率. 它也可以被解释为排序分值，如果我们对输出进行排序的话。

参数是指我们需要从数据中学到/决定的部分。在线性回归中，参数就是各个特征的系数θ。下文中，我们就用θ来表述模型中的所有需要学习的参数。

4.1.2 目标函数: 训练损失+正则项

通过选择恰当的y_i, 我们可以表示不同的任务，比如回归，分类和排序。训练模型的目标就是找到能使得训练数据x_i和y_i达到最佳拟合效果的最好的参数θ. 因此，为了训练模型，我们需要定义目标函数来衡量how well the model fit the training data.

目标函数的显著特征是它包含两个部分：训练损失和正则项：

$$obj(\theta) = L(\theta) + \Omega(\theta) \tag{21}$$

其中L表示训练损失函数，Ω表示正则项。训练损失函数可以衡量模型拟合训练数据的好坏程度，而正则项可以衡量模型的复杂程度。

一个常见的训练损失函数是mean square error

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2 \tag{22}$$

另外一个常见的训练损失函数是用于logistic regression的训练损失函数:

$$L(\theta) = \sum_i \left[y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i} \right] \tag{23}$$

正则项是通过控制模型复杂度，来避免过拟合。以下是一个例子来解释正则项的作用。

机器学习中的基本原则是希望我们的模型可以有好的预测能力的同时又是简单模型。故而目标函数中包含两项：训练损失函数和正则项。

4.2 Decision Trees Ensembles

XGBoost使用的模型是decision tree ensembles. 而tree ensemble 模型包含回归树和分类树，更具体一些是CART。一般的决策树的叶子节点上只包含决策值（预测值）。而CART的每个叶子节点上包含的是一个得分（score), 这就使得我们可以采用一个原则性的，统一性的方法进行优化。通常单个决策树是不太具有实用性的，而实践中常用的策略是ensemble model, 也就是训练多棵树，然后用所有树的和作为最终的预测值。

假设K表示树的个数，$\mathcal{F}$表示所有可能的CART树的集合，f_k表示函数空间$\mathcal{F}$中的一个函数, 则ensemble model （预测值）可以表示为：$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$，而需要优化的目标函数可以表示为:

$$obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \omega(f_k) \tag{24}$$

其中$\omega(f_k)$表示树f_k的复杂度，后面会详细定义。

Remark：Random Forest and XGBoost的共同点：都是基于Tree ensemble的模型，而二者的区别是如何训练。故，一个predictive service for tree ensemble可以同时为random forest and XGBoost二者调用。RF是用随机选取的subsamples and subsets of features来分别训练K棵树(bagging)，然后对K棵树求和（Or majority voting）得到最终结果。而XGBoost是按顺序一个一个训练K棵树，每棵树都用与修正前一棵树的误差(boosting)。因此可以发现RF和XGBoost中的树的作用是不一样的。

4.3 Tree Boosting 提升树

我们前面以已经介绍了模型。现在我们开始回答如下问题：我们应该怎么训练/学习树呢？对于所有的监督学习问题，我们总是可以通过定义一个目标函数来优化这个函数得到最终模型。

Boosting method主要是想回答以下问题：”Can a set of weak learners create a single strong learner?”

Boosting method解决上述问题的想法是（类似于res-net）我们都构造一个序列模型，每一个模型都试图修正前一个模型来得到更好的效果。也就是说每一个模型都尝试拟合前一个模型和目标的residual。

我们用$\hat{y}_i^{(t)}$表示第t步的预测值，用T个函数(basic learner/weaker learner)的和来逼近函数y，用数学语言可以表达为：

$$\hat{y} = \sum_{t=1}^{T} f_t(x) + const \tag{25}$$

其中$f_t(x) \in \mathcal{H}$属于某个给定的集合。

故而我们可以得到如下目标函数

$$obj = \sum_{i=1}^{n} l(y_i, \sum_{t=1}^{T} f_t(x) + const) + \sum_{t=1}^{T} \omega(f_t) \tag{26}$$

mathematically, 我们可以表示最终模型为：

$$\hat{y} = arg \min_{f_t \in \mathcal{H}, t=1,...,T} obj = arg \min_{f_t \in \mathcal{H}, t=1,...,T} \sum_{i=1}^{n} l(y_i, \sum_{t=1}^{T} f_t(x) + const) + \sum_{t=1}^{T} \omega(f_t) \tag{27}$$

而尝试直接一次性学习所有的树是不现实的，所以我们采用退而求其次的方式additive training. 也就是固定已经学好的模型，然后每次只学一个新的模型。如此一来，每一步，我们就只需要优化如下目标函数at step t:

$$f_t^* = arg \min_{f_t \in \mathcal{H}} obj^{(t)} \tag{28}$$

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \omega(f_t) + const \tag{29}$$

此处复杂度只有$\omega(f_t)$,因为前面每一步的模型都固定了，所以其复杂度被视为const.

那现在问题就转化为如何优化上述目标函数呢？

我们对上述目标函数的训练损失函数在$\hat{y}_i^{(t-1)}$处进行Taylor expansion到第二阶，

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) + \omega(f_t) + const \tag{30}$$

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

因为第t步之前的模型都固定，所以$l(y_i, \hat{y}_i^{(t-1)})$可以被视为一个常数，那么，忽略所有常数，上述优化问题就等价于如下优化问题

$$arg \min_{f_t \in \mathcal{H}} obj^{(t)} = arg \min_{f_t \in \mathcal{H}} \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t) \tag{31}$$

所以每一步我们就可以通过优化上述目标函数来找到当前步的模型（tree）.

这个目标函数表达式的一个重要优势就是：目标函数只依赖于g_i, h_i, 所以我们用一个只需要g_i, h_i作为输入的求解器来优化任意给定的损失函数/目标函数（只要存在二阶导。This is how XGBoost supports custom loss functions. We can optimize every loss function using exactly the same solver that takes g_i, h_i as input!

4.3.1 模型复杂度

我们现在来给出模型复杂度的具体表达式。XGBoost中选则的weaker learner is CART, 因此首先定义树$f(x)$，然后定义树的复杂度。

树可以定义为：

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1, 2, ..., T\}. \tag{32}$$

其中w是所有叶子节点上的得分向量(vector of scores on leaves)，q是将每一个数据点映射到相应叶子上的函数，T是指叶子的总个数。

在XGBoost中，复杂度定义如下：

$$\omega(f) = \gamma T + \frac{1}{2}\sum_{j=1}^{T} w_j^2 \tag{33}$$

叶子的总个数和所有叶子节点上得分的平方和。(L2正则项)

4.3.2 The Structure Score

通过重新改写树模型，我们可以得到第t个树的目标函数：

$$obj^{(t)} \approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \tag{34}$$

这个表达式是按照训练集中的数据点来进行计数

$$obj^{(t)} \approx \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2)] + \gamma T \tag{35}$$

这个表达式是通过叶子节点来计数，先数清楚每个叶子节点中的数据点，因为每个叶子节点中的数据具有同样的得分，然后再对所有叶子节点求和。

其中$I_j = \{i|q(x_i) = j\}$表示被分配到第j个叶子的数据点下标的集合。

我们可以进一步简化表达式，通过引入2个新变量，$G_j = \sum_{i\in I_j} g_i, H_j = \sum_{i\in I_j} h_i$:

$$obj^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \tag{36}$$

其中得分w_j是互相独立的，而$G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2$是二次型，而对于给定的树结构$q(x)$, 最优$w_j$ 可以直接求解

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{37}$$

对于给定的树结构，每个叶子节点的最佳得分

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{38}$$

可以用来衡量给定的这个树结构有多好。i.e. measure how good a tree structure $q(x)$ is.

至此，我们已经找到了给定树结构所对应的最佳得分（每个叶子结点的权重w），接下来我们就只需要关注如何找到树的最佳结构。

4.3.3 学习树的结构Learn the tree structure

现在我们已经找到了一种衡量树结构表现有多好的方式，理想的想法是，我们可以直接遍历所有可能的树，然后选择最佳的树结构。然而这个计算量是不现实的，所以，我们实际操作中是，每次只尝试优化树的一层结构(level-wise)，也就是说，我们尝试计算把一个叶子节点分类成2个叶子节点时，得分增益是（the score this split gains is）：

$$Gain = [(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}) - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - 2\gamma \tag{39}$$

或者

$$Gain = \frac{1}{2}[(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}) - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma \tag{40}$$

得分增益的表达是可以分为四个部分：

1. 新的左叶子节点的得分，
2. 新的有叶子节点的得分，
3. 原来也叶子节点（被分裂之前的也叶子节点）的得分，
4. 对于2个额外叶子节点的正则项。

我们现在已经知道如何衡量对一个叶子节点进行一次分裂时得到的的得分增益，那么如何快速找到最佳的分裂节点（optimal split, 这里指best feature to split and the optimal value at this feature to split）呢？

我们可以把所有例子（数据/样本）按照顺序排好，然后从左到右扫描，从而快速计算出所有可能的分裂的结构得分，从而快速找到最佳分裂。

Remark: 我们从得分增益中可以观察到：如果增益比γ小，那么我们就不应该进行分裂，而这就是树模型中使用的剪枝策略，而且得分增益还告诉了我们为什么这个剪枝策略works！

Two main questions:
a. What is the difference between the gradient descent in GBDT and gradient descent in deep learning?
learning rate/step size is fixed in GBDT, usually, lr=0.01,
learning rate in
b.

Reference

1. https://daimajiaoliu.com/daima/4833c5a83900410
2. https://blog.csdn.net/zhangbaoanhadoop/article/details/79904175
3. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote17.html DT
4. https://www.zhihu.com/question/299719792
5. https://zhuanlan.zhihu.com/p/48274203
6. https://www.cnblogs.com/wxquare/p/5379970.html
7. https://blog.csdn.net/weixin_34029680/article/details/94642649?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog A very good one!
8. http://leijun00.github.io/2014/09/decision-tree/ A very good one!
9. http://leijun00.github.io/2014/10/decision-tree-2/ A very good one!
10. https://www.cnblogs.com/wxquare/p/5372996.html Not read yet! GBDT
11. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote18.html bagging and RF
12. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote19.html Boosting and AdaBoost
13. https://xgboost.readthedocs.io/en/stable/tutorials/model.html XGBoost