XGBoost 算法

XGBoost 算法是由陈天奇老师开发的一种经典的集成提升（Boosting）算法，它将许多基学习器集成在一起，形成一个强学习器。同时，该强学习器具有预测效果好、训练效率高、可控参数多等等优点，现 XGBoost 算法已被广泛应用大数据分析领域。

XGBoost 涉及到基学习器和提升（Boosting）的概念。这里的基学习器，XGBoost 算法多采用分类与回归决策树。但相比随机森林中的基学习器，XGBoost 更加多样化，它可以是分类与回归决策树，也可以是线性模型，而随机森林中的基学习器只能是分类与回归决策树。而 Boosting 也与随机森林中的 Bagging 不同。Boosting，即提升，是一个迭代的过程。不像 Bagging，Boosting 给每一个训练样本赋予不同的权值，并且在每一轮 Boosting 的过程结束时便会自动地调整权值，从而使得基分类器可以更好地聚集在那些很难分类的样本上。这样通过聚集每个经过提升的基分类器，XGBoost 算法就可以得到最终的组合分类器，从而建立起一个具有更好的预测性能的模型。

XGBoost 算法可以拆分成四个主要的阶段：目标函数构造、目标函数优化、树信息表示以及确定树结构。

1. 目标函数构造

构造目标函数是任何一个算法必不可少的阶段。算法在进行求解时，都需要根据构建好的目标函数来进行下一步的优化求解。由于 XGBoost 算法是将多个基学习器通过 Boosting 的方式组合成一个强学习器，同时其基学习器多采用分类与回归决策树，故这里我们假设前面已经训练了 K 棵树作为基学习器，则第 i 个样本的预测值为

$$\hat{y}_i = \sum_{k=1}^{K} f_k(X_i) \qquad (1)$$

在公式（1）中 $f_k(X_i)$ 表示第 k 棵树对第 i 个样本的预测值，比如第 1 棵树预测值为$f_1(X_i)$，第 2 棵树预测值为$f_2(X_i)$，依次类推，将这些树的预测值累加到一起，则得到第 i 个样本的最终预测值 $\hat{y}_i$。

同时为了降低过拟合的发生，xgboost 算法添加了正则项来控制模型的复杂

度，进而提高模型的泛化能力。因此 xgboost 算法的目标函数为

$$\mathrm{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1} O(f_k) \quad (2)$$

其中公式（2）中 的$\sum_{k=1} O(f_k)$表示算法的空间复杂度，$\sum_{i=1}^{n} l(y_i, \hat{y}_i)$表示算法的预测误差，Obj 就是 XGBoost 算法的目标损失函数值。由于我们在训练第 K 个树时，第 K 个树的预测值 $\hat{y}_i^k = \hat{y}_i^{k-1} + f_k(X_i)$，又前面第 K-1 个树已知，故$\sum_{j=1}^{k-1} O(f_j)$、$\hat{y}_i^{k-1}$为常数，在目标函数求最值时常数可忽略，则公式（2）可化为以下形式：

$$\mathrm{Obj} = \sum_{i=1}^{n} l[y_i, \hat{y}_i^{k-1} + f_k(X_i)] + O(f_k) \quad (3)$$

此时通过最小化公式（3）中的 Obj，XGBoost 算法就可以同时很好地解决模型欠拟合和过拟合的问题。

2. 目标函数优化

在上述的推导中我们已经得到 xgboost 算法的目标函数为 Obj，此时令 $f(x) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{k-1})$，其中 $x=\hat{y}_i^{k-1}$,令 $f(x+\Delta x)=l[y_i, \hat{y}_i^{k-1} + f_k(X_i)]$，其中 $\Delta x = f_k(X_i)$，现对 f(x+Δx)进行二阶泰勒展开，记其一阶导为：

$$g_i = \partial_{\hat{y}_i^{k-1}} l(y_i, \hat{y}_i^{k-1}) \quad (4)$$

记其二阶导为：

$$h_i = \partial^2_{\hat{y}_i^{k-1}} l(y_i, \hat{y}_i^{k-1}) \quad (5)$$

则目标函数可化为

$$\mathrm{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{k-1}) + g_i \cdot f_k(X_i) + \frac{1}{2} h_i \cdot f_k^2(x_i) + O(f_k) \quad (6)$$

由于前面 K-1 棵树的信息已知，故公式中的第一项加数$l(y_i, \hat{y}_i^{k-1})$已知，其表示为第 K-1 棵树的预测损失值，而g_i、h_i分别表示第 K-1 棵树的一阶、二阶导

数，由于前面 K-1 棵树的信息已知，则g_i、h_i也为已知信息。也就是说，在当前的目标函数中，只有第 K 棵树的信息，包括预测函数$f_k(X_i)$和复杂度函数$O(f_k)$未知，而前面的 K-1 棵树的预测损失信息已经通过一阶导g_i和二阶导h_i传递给第 K 棵决策树中，因此下一步的优化求解工作算法只需要关注第 K 棵树的的信息，包括$f_k(X_i)$、$O(f_k)$，即 XGBoost 算法只需要在每次提升过程中关注当前的决策树即可。

3. 树信息表示

为了同时表示第 K 棵的$f_k(X_i)$、$O(f_k)$，则需要将树结构的信息，即预测值和空间复杂度的表示，引入到上述公式（6）目标函数中。为了解决这个问题，XGBoost 算法选择对第 K 棵树的预测值$f_k(X_i)$和第 K 棵树的复杂度$O(f_k)$进行参数化表示。

对于第 K 棵树的预测值$f_k(X_i)$，XGBoost 算法引入了两个变量：决策树叶子权重 w 和样本所在叶子的位置信息$q(x_i)$，则可将第 K 棵树的预测值$f_k(X_i)$参数化表示为关于叶子权重的函数：

$$f_k(X_i) = w_{q(x_i)} \tag{7}$$

其中公式（7）中的$w_{q(x_i)}$是根据样本落在叶子节点的位置直接遍历计算损失函数，即有 n 个样本就会遍历 n 次位置信息。而从叶子节点遍历的角度出发的话，记 j 表示树的叶子节点，叶子结点总数为 T，I_j表示样本落在第 j 个叶子节点上，则我们可将$w_{q(x_i)}$转化为w_j，即将样本遍历转化为叶子结点遍历，其数学公式推导如下：

$$\sum_{i=1}^{n} g_i \cdot f_k(X_i) = g_1 \cdot w_{q(x_1)} + \cdots\cdots + g_n \cdot w_{q(x_n)} \tag{8}$$

$$w_{q(x_1)} + \cdots\cdots + g_n \cdot w_{q(x_n)} = w_1 \cdot \sum_{i \in I_1} g_i + \cdots\cdots + w_1 \cdot \sum_{i \in I_T} g_T \tag{9}$$

$$\sum_{j=1}^{T} \left(\sum_{i \in I_j} g_i \right) \cdot w_j = w_1 \cdot \sum_{i \in I_1} g_i + \cdots\cdots + w_1 \cdot \sum_{i \in I_T} g_T \tag{10}$$

因此根据公式（7）、（8）、（9）、（10）的等价关系，我们可以得到：

$$\sum_{i=1}^{n} g_i \cdot f_k(X_i) = \sum_{j=1}^{T} (\sum_{i \in I_j} g_i) \cdot w_j \tag{11}$$

故此，针对关乎叶子权重的函数$f_k(X_i)$，我们可以使用叶子权重 w、位置信息I_j和叶子结点数 T 的参数来对其进行表示。

而对于模型复杂度 O(f_k)，XGBooost 算法通过树的深度、叶子节点个数以及叶子节点值来控制。由于 XGBoost 算法中的决策树是分类与回归决策树（classification and regression trees，CART），而 CART 属于二叉树，因此树的深度也间接由叶子节点个数控制了。当叶子节点数越多，决策树的结构也就越复杂。而对于叶子节点值而言，当叶子节点值越小，预测值就会分布在较多的决策树叶子节点上，等同于每棵决策树都参与预测其中的一小部分，降低了模型过拟合的风险。因此，其复杂度函数可被表示为：

$$O(f_k) = \gamma T + \frac{1}{2}\alpha \sum_{j=1}^{T} w_j^2 \tag{12}$$

其中公式（12）中的γ、α分别为叶子节点个数 T 和叶子节点值w_j的超参数。出于最小化损失函数的想法，因此如果我们希望叶子个数 T 尽可能少，则我们可以将超参数γ的值设定得大一些。同理，如果希望叶子权重值w_j尽可能小，那么我们也可以将超参数α的值设定得大一些。将$O(f_k)$、$\sum_{i=1}^{n} g_i \cdot f_k(X_i)$的表达式代入到公式（6）中的目标函数，得到

$$\text{Obj} = \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) \cdot w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \alpha) \cdot w_j^2] + \gamma \cdot T \tag{13}$$

记G_j、H_j分别表示第 K-1 棵树的一阶导g_i、二阶导h_i关于样本 i 在所落叶子节点 j 上的遍历，则

$$G_j = \sum_{i \in I_j} g_i \tag{14}$$

$$H_j = \sum_{i \in I_j} h_i \tag{15}$$

将公式（14）、（15）代入到公式（13）中的目标函数，则 Obj 可被表示为关于w_j的一元二次方程，其表达式如下：

$$\mathrm{Obj} = \sum_{j=1}^{T} [G_j \cdot w_j + \frac{1}{2}(H_j + \alpha) \cdot w_j^2] + \gamma \cdot T \tag{16}$$

此时我们求目标函数的极小值，先对w_j进行求导，从而我们可以得到决策树的最佳叶子节点值如下：

$$w_j^* = -\frac{G_j}{H_j + \alpha} \tag{17}$$

将w_j^*的值代入目标损失函数中，我们可以得到其目标函数损失值如下：

$$Obj^* = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \alpha} + \alpha \cdot T \tag{18}$$

显然，当上述的损失值越小，树的预测准确度和复杂度的平衡越好，更好地提升了算法的预测性能和泛化能力。

4. 树形状确定

在上述树信息表示的工作中，XGBoost 算法引入了位置信息变量，而这需要建立在树形状确定的基础上。只有在此基础上，我们才可得到w_j^*和Obj^*的值，从而进行后续的优化求解。而在 XGBoost 树模型中，其使用贪心算法，在每一次节点分割时选择使用损失函数下降最大的特征，从而最终确定树形状。

5. 算法特点

基于上述 XGBoost 算法底层原理的推导，我们可知道其具有以下的特点：

首先，在算法推导的过程中，算法的目标损失函数使用泰勒二阶展开，相比于一阶泰勒展开，可以更加准确地逼近目标函数，从而提升预测的准确性；

其次，XGBoost 可以处理特征上的缺失值，这是由于其在寻找分割点时，只会遍历非缺失的特征值样本。因此针对缺失值变量，在 XGBoost 的算法框架中不一定非要处理，这有利于处理大规模的数据；

最后，XGBoost 算法支持特征维度的并行化训练，从而可以提升训练速度。