XGboost 算法学习报告

1. 知识点概述

（1）**XGBoost 算法**：也成**极端梯度提升算法**，是 2014 年提出的基于 CART 回归树的一种 boosting 集成算法,是对梯度提升决策树(GBDT)算法的一种改进。

（2）**XGBoost 的目标**：建立 t 棵回归树使得树群对样本的预测值尽可能接近样本的真实值,并且具有一定的泛化能力。

（3）**GBDT 算法**：全称为 Gradient Boosting Decision Tree，是一种基于决策树(decision tree)实现的分类回归算法,即 GBDT 由两部分组成:梯度提升和决策树。

2. 分类与回归树

（1）**分类与回归树模型(CART)**：典型的二叉决策树，根据其内部节点特征的取值将样本递归的分为两部分，将判断为“是”的样本点划分到左边，右边则是判断为“否”的样本点。这样便将特征空间划分成了有限的单元,并在这些单元上确定预测的概率分布。

（2）**应用**：用于分类或者回归。若样本输出结果是离散值，CART 用于分类；若样本输出结果是连续型数据，CART 则用于回归预测。

2.1. CART 回归树生成算法

1. 假设有样本集 $S=(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)$，一共有 N 个样本，X 为输入变量，Y 为取值连续的输出变量。特征集 $F=f_1,f_2,\ldots,f_M$，每一个样本都对应一组特征。

2. **步骤**：

(1) 现在有原始数据集 S，树的深度 depth=0。

(2) 特征选择,针对集合 S,遍历每一个特征 feature，逐个取出其对应的所有的 value，每个 value 都可看作一个切分点。选择第 f 个特征和它的取值 v，小于该值 v 的样本点划分到左边集合$R_1(f,v)=\{x|x^(f)\le v\}$，将大于该值 v 的样本点划分到右边集合$R_2(f,v)=\{x|x^(f)>v\}$。不同特征 feature 及其对应的不同取值 value,就构成了许多切分方式。接着就需要知道该如何来选择最优的特征 f 以及对应的切分点 v。最小化均方误差即可对固定的特征变量 f 找到最优切分点 v.具体公式如下:

$$\min_{f,v}\left[\min_{c_1}\sum_{x_i\in R_1(f,v)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(f,v)}(y_i-c_2)^2\right]$$

(3) 找到最佳分割 feature 以及最佳分割 value 之后,用该 value 将集合 S 分裂成 2 个集合:左集合 R1、右集合 R2，每一个集合也叫做一个节点。此时树的深度 depth += 1。

(4) 针对集合 R1、R2 分别重复步骤 2、3,直到达到终止条件。

(5) 最后生成的、不再进行分裂的集合就叫做叶子节点。落在该叶子节点内的样本的预测值就是该叶子节点的值。

（**注意**：同一个叶子节点中的样本具有同一个预测值。）

3. **终止条件**:

(1) 节点中的样本个数小于预定阈值。

(2) 节点中的样本基本属于同一类.此时,样本已经被全部划分出来,节点停止分裂。

(3) 节点中已经不存在样本了。

2.2. CART 剪枝算法

1. CART 剪枝算法由两步组成:

(1) 由生成算法产生的决策树T_0底端开始不断剪枝,直到T_0的根节点,形成一个子树序列 $T_0, T_1, \ldots, T_n$。

(2) 通过交叉验证法在独立的验证数据集上对子树序列进行测试,从中选择最优子树。

2. 建模的目标：让损失函数达到最优。用一个变量α来平衡，减少树的大小来防止过拟化,同时防止去掉一些节点后预测的误差会过分增大。

3. 每次剪枝剪的都是某个内部节点的子节点，（将某个内部节点的所有子节点回退到这个内部节点里，并将这个内部节点作为叶子节点）。因此在计算整体的损失函数时,这个内部节点以外的值都没变，只有这个内部节点的局部损失函数改变了，因此我们本需要计算全局的损失函数，但现在只需要计算内部节点剪枝前和剪枝后的损失函数。

3. 基于残差的训练方式

1. Boosting 是用基于残差的训练方式通过迭代多棵树来共同决策。

2. GBDT 的**核心**：每一棵树学的是之前所有树预测结果和的残差,这个残差就是一个加预测值后能得真实值的累加量。

4. Extreme Gradient Boosting (XGBoost)

XGBoost：极端梯度提升(Extreme Gradient Boosting)，是一种加法模型,将模型上次预测(由 t-1 棵树组合而成的模型)值与真实值的残差作为参考进行下一棵树(第 t 棵树)的建立。以此每加入一棵树将其损失函数不断降低。

4.1. 目标函数构建

1. 假设给定数据集 $D = \{(x_i, y_i)\}$，XGBoost 进行加法训练,学习 t 棵树,采用以下函数对样本进行预测:

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

$$\mathcal{F} = \{f(x) = w_q(x)\}(q: \mathbb{R}^m \to T, w \in \mathbb{R}^T)$$

（F 表示所有可能的 CART 树，f(x)是一棵回归树(CART)，表示对样本进行的一次预测。）

加法策略具体描述：

（1）初始化(模型中没有树)时，其预测结果为 0:

$$\hat{y}_i^{(0)} = 0$$

（2）往模型中加入第一棵树:

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

（3）往模型中加入第二棵树:

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

.....

（4）往模型中加入第 t 棵树:

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

加法策略经过 t 次迭代后,模型的预测等于前 t-1 次的模型预测加上第 t 棵树的预测,具体预测函数表达式如下:

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

那么，此时目标函数则可写作:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

4.2. 目标函数近似化处理

1. XGBoost 目标函数的优化使用了一阶和二阶偏导数，二阶导数有利于梯度下降的更快更准。

2. 使用泰勒展开取得函数做自变量的二阶导数形式，可以在没有选定损失函数具体形式的情况下，仅仅依靠输入数据的值就可以进行叶子节点分裂优化计算，本质上也就把损失函数的选取和模型算法优化、参数选择分开了。这种去耦合增加了 XGBoost 的适用性,使得它按需选取损失函数。

3. 最终化简得到：

$$\tilde{L}(t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t)$$

4.3. 目标函数引入树的结构

1. **目标**：寻找一个参数来最小化目标函数 $\tilde{L}(t)$ 。

2. 目标函数在函数空间中的表示方法：

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

参数空间中的目标函数表示如下：

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

把树的结构引入到目标函数中：

$$f(x) = w_{q(x)} \quad (q: \mathbb{R}^n \to \{1,2,\dots,T\}, w \in \mathbb{R}^T)$$

3. 在 XGBoost 算法中，采用叶子节点分数和叶节点个数衡量树的**复杂度**。

其正则项的具体表示如下：

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

代入目标函数中得到：

$$\tilde{L}(t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t)$$

$$= \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

4.4. 贪心算法构建回归树

1. 贪心法思想：每次尝试分裂一个叶节点，计算分裂前后的增益，选择增益最大的。

2. 对一个叶子节点进行分裂,分裂前后的增益定义为:

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

（此处引入了γ,表示加入新叶子节点时增加的复杂度）

3. Gain 的值越大，分裂后 L 减小越多。所以当对一个叶节点分割时,计算所有候选(feature, value)对应的 Gain，选取 gain 最大的(feature, value)进行分裂。