1.(b)

XGBoost 算法

XGBoost 算法的原理是在一颗 CART 决策树上不断加树，使得算法的精确了不断提升。xgboost 的模型就是选用一批 CART 决策树做预测，然后简单的将各个树的预测分数相加，其数学模型如下：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

其中 K 是树的棵数，F 表示所有的可能的 CART 树，f 表示其中某一课具体的 CART 树。该模型就由 K 棵 CART 树组成。该模型的目标函数如下：

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

这个目标函数分为两部分，其中第一部分是损失函数，第二部分是正则项，正则项由 K 棵树的正则化项相加而来。

XGBoost 通过加法训练来最小化目标函数以找到最佳的参数组。参数蕴含在每棵 CART 树中。确定一棵 CART 树需要两部分：第一是树的结构，第二是每个叶子结点上的分数。将叶子结点作为参数容易实现：在树的结构确定时，只需要将正则化项设为各个叶子结点的平方和，此时整个目标函数就是 K 棵树的所有叶子结点的函数，可以用梯度下降或者随即下降来优化目标函数。但确定 K 棵树的结构比较困难，需要用特殊的算法，即加法训练。加法训练的思想就是分步骤优化目标函数：首先优化第一棵树，然后在此基础上优化第二颗树，以此类推，直到优化完 K 棵树：

$$\hat{y_i}^{(0)} = 0$$

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

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

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

在第 t 步时，添加一棵最优的 CART 树 ft，这棵树是在现有的 t-1 棵树的基础上，使得目标函数最小的那一棵 CART 树：

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

其中 constant 是前 t-1 棵树的复杂度。l 表示损失函数。如果使用损失函数 MSE，则上述表达式会变成：

$$obj(t) = \sum_{i=1}^{n} (y_i - (\hat{y_i}^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} [2(\hat{y_i}^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant$$

对于一般的损失函数，需要将其作泰勒二阶展开：

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

其中 gi 和 hi 分别是损失函数对$\hat{y_i}^{(t-1)}$的一阶微分和二阶微分。要使得目

标函数最小化，去掉常数项，得到：

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

因此，XGBoost 只需要要求损失函数二次可微。

为了定义后面的$\Omega(f_t)$，对 CART 作另一番定义如下：

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

该定义阐述，一棵树包含 T 个叶子节点，这 T 个叶子节点的值组成了一个 T 维向量 w，q(x)是一个映射，用来将样本映射成 1 到 T 之间的某个值，也就是把它分到某个叶子节点。q(x)代表了 CART 树的结构，而 w_q(x)则代表这棵树对样本 x 的预测值。

由此，使用以下的正则化项：

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

上式中，λ和γ是 XGBoost 自定义的参数，两个参数越大都会导致获得结构越简单的树，因为此时对较多叶子节点的树惩罚较大。

代入之前得到的优化函数，得到：

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

$$\doteq \sum_{j=1}^{n}[(\sum_{i \in I_j} g_i)w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)w_j^2] + \gamma T$$

通过定义：

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

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

将原式简化为：

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

对于第 t 棵 CART 树的某一个确定的结构（可用 q(x)表示），所有的 Gj 与 Hj 都是确定的，且上式中各个叶子节点的值 wj 之间是互相独立的。上式是一个简单的二次式，易得各个叶子节点的最佳值以及此时目标函数的值：

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

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

obj*的值越小，代表树的结构越好。

上面已经求出了最佳的叶子节点的值并得到了评判树结构优劣的标准，剩下的最后一个任务就是确定最优的树的结构。确定了树的结构，由于叶子节点的分数已经知道，因此就完成了算法。

树的结构近乎无限多，一个一个去测算它们的好坏程度然后再取最好的显然是不现实的，因此我们逐步学习出最佳的树结构。这与将 K 棵树的模型分解成一棵一棵树来学习是一个道理，只不过从一棵一棵树变成了一层一层节点而已。最简单的树结构就是一个节点的树。我们可以算出这棵单节点的树的好坏程度 obj*。假设我们现在想将这棵单节点树进行分叉，我们需要知道：1、按照某个

特征分是否有效，也就是是否减少了 obj 的值；2、如果可分，那么以哪个（特征的）值来分。为了回答上面两个问题，可以将需要分叉的点进行排序，从左到右扫描就可以找出所有的切分点。

对每一个确定的切分点，衡量切分好坏的标准如下：

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

Gain 实际上就是单节点的 obj*减去切分后的两个节点的树 obj*，Gain 如果是正的，并且值越大，表示切分后 obj*越小于单节点的 obj*，就越值得切分。同时还可以观察到，Gain 的左半部分如果小于右侧的 γ，则 Gain 就是负的，表明切分后 obj 反而变大了。γ 在这里实际上是一个临界值，它的值越大，表示我们对切分后 obj 下降幅度要求越严。这个值也是 XGBoost 里自定义的参数。

扫描结束后，我们就可以确定是否切分。如果切分，对切分出来的两个节点，递归地调用这个切分过程，我们就能获得一个相对较好的树结构。结合上面求得的最佳叶子节点的值，我们就找到了第 t 棵树。

注意：xgboost 的切分操作和普通的决策树切分过程是不一样的。普通的决策树在切分的时候并不考虑树的复杂度，而依赖后续的剪枝操作来控制。xgboost 在切分的时候就已经考虑了树的复杂度，就是那个 γ 参数。所以，它不需要进行单独的剪枝操作。

2. 比较支持向量机、 AdaBoost 、逻辑斯谛回归模型的学习策略与算法。

（1）支持向量机：策略是极小化正则化合页损失，软间隔最大化；算法是序列最小最优化算法（即 SMO 算法）。

（2）Adaboost：策略是极小化加法模型的指数损失；算法是前向分布加法

（3）逻辑斯谛回归模型：策略是极大对数似然函数，正则化的极大似然估计；算法是改进的迭代尺度算法、梯度下降、拟牛顿法等。