机器学习总结——决策树及集成学习

1、什么是熵（entropy）？什么是条件熵(conditional entropy)？什么是经验熵？什么是经验条件熵？决策树特征选择的准则有哪些？什么是信息增益（ID3算法）？什么是信息增益比（C4.5算法）？为什么C4.5要选择信息增益比作为特征选择的准则？什么是基尼系数（CART算法）？什么是剪枝？

1）在信息论和概率统计中，熵是表示随机变量不确定性的度量。设X是一个取有限个值的离散随机变量，其概率分布为

$$P(X=x_i)=p_i,\quad i=1,2,\cdots,n$$

则随机变量X的熵定义为

$$H(X)=-\sum_{i=1}^{n}p_i\log p_i$$

熵只依赖于随机变量X的分布，而与X的取值无关，所以可以将X的熵记作***H(p)***,即

$$H(p)=-\sum_{i=1}^{n}p_i\log p_i$$

熵越大，随机变量的不确定性经越大！

2）随机变量***X***给定条件下随机变量***Y***的条件熵***H(Y\|X)***,定义为***X***给定条件下***Y***的条件概率分布的熵对***X***的数学期望

$$H(Y\mid X)=\sum_{i=1}^{n}p_iH(Y\mid X=x_i)$$

这里，

$$p_i=P(X=x_i),\quad i=1,2,\cdots,n.$$

3）当熵和条件熵中的概率由数据估计（特别是极大似然估计）得到时，所对应的熵与条件熵分别称为经验熵（empirical entropy）和经验条件熵（empirical conitional entropy）

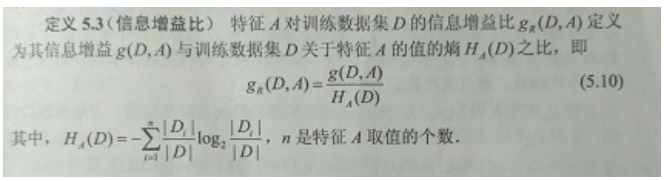
4）决策树特征选择的准则有三种信息增益，信息增益比以及基尼指数，三者分别对应的算法是ID3，C4.5和CART。

- 信息增益（information gain）表示得知特征***X***的信息而使得类***Y***的信息的不确定性减少的程度！样本集合***D***对特征***A***的信息增益（ID3）

$$g(D,A)=H(D)-H(D\mid A)$$
$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$
$$H(D\mid A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)$$

其中，$H(D)$ 是数据集 D 的熵，$H(D_i)$ 是数据集 D_i 的熵，$H(D\mid A)$ 是数据集 D 对特征 A 的条件熵．D_i 是 D 中特征 A 取第 i 个值的样本子集，C_k 是 D 中属于第 k 类的样本子集．n 是特征 A 取值的个数，K 是类的个数．

- 以信息增益作为划分数据集的特征，存在于偏向于选择取值较多的特征的问题，使用信息增益比可以对此问题进行校正。



定义 5.3（信息增益比） 特征 A 对训练数据集 D 的信息增益比 $g_R(D,A)$ 定义为其信息增益 $g(D,A)$ 与训练数据集 D 关于特征 A 的值的熵 $H_A(D)$ 之比，即

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)}\qquad(5.10)$$

其中，$H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$，$n$ 是特征 A 取值的个数．

5）C4.5为什么要用信息增益比？？

根据公式

$$IGain(S,A) = E(S) - E(A)$$ ，

E(S)为初始label列的熵，并未发生变化，则IGain(S,A)的大小取决于E(A)的大小，E(A)越小，IGain(S,A)越大，而根据前文例子，

$$E(A) = \sum_{v \in value(A)} \frac{num(S_v)}{num(S)} E(S_v)$$ ，

若某一列数据没有重复，ID3算法倾向于把每个数据自成一类，此时

$$E(A) = \sum_{i=1} \frac{1}{n} log_2(1) = 0$$ ，

这样E(A)为最小，IGain(S,A)最大，程序会倾向于选择这种划分，这样划分效果极差。

为了解决这个问题，引入了信息增益率（Gain-ratio）的概念，计算方式如下：

$$Info = -\sum_{v \in value(A)} \frac{num(S_v)}{num(S)} log_2 \frac{num(S_v)}{num(S)}$$ ，

这里Info为划分行为带来的信息，信息增益率如下计算：

$$Gain - ratio = \frac{IGain(S,A)}{Info}$$

这样就减轻了划分行为本身的影响。

对于取值多的属性，尤其一些连续型数值，比如两条地理数据的距离属性，这个单独的属性就可以划分所有的样本，使得所有分支下的样本集合都是“纯的”（最极端的情况是每个叶子节点只有一个样本），再举个例子对于多值特征（极端情况下，unique id）这时候按照信息增益切分各部分都是纯的 熵最小 是0，但是这种切分没有意义 。采取信息增益方式，若某属性取值数目远大于类别数量时信息增益可以变得很大，对训练集地分割尽管不错，但泛化能力同时也会变差。这种分割方式导致在每个分割空间内数据较纯净，甚至能使熵为0，但未利用其它实际上可能更加有效的分类属性信息。**一个属性的信息增益越大，表明属性对样本的熵减少的能力更强，这个属性使得数据由不确定性变成确定性的能力越强。**

所以如果是取值更多的属性，更容易使得数据更“纯”（尤其是连续型数值），其信息增益更大，决策树会首先挑选这个属性作为树的顶点。**结果训练出来的形状是一棵庞大且深度很浅的树，这样的划分是极为不合理的。**

C4.5使用了信息增益率，在信息增益的基础上除了一项split information,来惩罚值更多的属性。

6）CART生成决策树就是递归的构建二叉**决策树**的过程，对**回归树用平方误差最小化准则**，对**分类树用基尼指数最小化准则**，进行特征选择，从而生成二叉**树**。

- 回归树生成——最小二乘回归树生成算法

算法 5.5（最小二乘回归树生成算法）

输入：训练数据集 D；

输出：回归树 $f(x)$.

在训练数据集所在的输入空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

（1）选择最优切分变量 j 与切分点 s，求解

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right] \tag{5.21}$$

遍历变量 j，对固定的切分变量 j 扫描切分点 s，选择使式 (5.21) 达到最小值的对 (j,s).

（2）用选定的对 (j,s) 划分区域并决定相应的输出值：

$$R_1(j,s) = \{x \mid x^{(j)} \leqslant s\}, \quad R_2(j,s) = \{x \mid x^{(j)} > s\}$$

$$\hat{c}_m = \frac{1}{N_m}\sum_{x_i \in R_m(j,s)} y_i, \quad x \in R_m, \quad m = 1,2$$

（3）继续对两个子区域调用步骤 (1)，(2)，直至满足停止条件.

（4）将输入空间划分为 M 个区域 $R_1, R_2, \cdots, R_M$，生成决策树：

$$f(x) = \sum_{m=1}^{M} \hat{c}_m I(x \in R_m)$$

- 分类树生成——CART生成算法

算法 5.6（CART 生成算法）

输入：训练数据集 D，停止计算的条件；

输出：CART 决策树.

根据训练数据集，从根结点开始，递归地对每个结点进行以下操作，构建二叉决策树：

（1）设结点的训练数据集为 D，计算现有特征对该数据集的基尼指数. 此时，对每一个特征 A，对其可能取的每个值 a，根据样本点对 $A=a$ 的测试为“是”或“否”将 D 分割成 D_1 和 D_2 两部分，利用式（5.25）计算 $A=a$ 时的基尼指数.

（2）在所有可能的特征 A 以及它们所有可能的切分点 a 中，选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点. 依最优特征与最优切分点，从现结点生成两个子结点，将训练数据集依特征分配到两个子结点中去.

（3）对两个子结点递归地调用（1），（2），直至满足停止条件.

（4）生成 CART 决策树.

算法停止计算的条件是结点中的样本个数小于预定阈值，或样本集的基尼指数小于预定阈值（样本基本属于同一类），或者没有更多特征.

这里的基尼指数公式计算如下：

在特征A的条件下，集合D的基尼指数定义为

$$\text{Gini}(D,A)=\frac{|D_1|}{|D|}\text{Gini}(D_1)+\frac{|D_2|}{|D|}\text{Gini}(D_2) \qquad (5.25)$$

基尼指数Gini(D)表示集合D的不确定性，基尼指数Gini(D,A)表示经A＝a分割后集合D的不确定性。基尼指数值越大，样本集合的不确定性也就越大，这一点与熵很类似。

7）在决策树学习中将已生成的树进行简化的过程称为剪枝（pruning）,具体的，剪枝从已生成的树上裁掉一些子树或叶结点，并将其根结点或父结点作为新的叶结点，从而简化分类树模型，使其具有更好的泛化能力。

决策树学习算法包含**特征选择**、**决策树生成**与**决策树的剪枝**过程。决策树表示在给定特征条件下类的条件概率分布，所以深浅不同的决策树对应着不同复杂度的概率模型。**决策树的生成对应于模型的局部选择**，**决策树的剪枝对应于模型的全局选择**。决策树的生成只考虑局部最优，相对地，决策树的剪枝则考虑全局最优。

2、什么是集成学习？

主要可以分为三大类，Boosting, Bagging, Stacking。Boosting的代表有AdaBoost, gbdt, xgboost。而Bagging的代表则是随机森林(Random Forest)。Stacking 的话，好像还没有著名的代表，可以视其为一种集成的套路。集成学习的关键和核心：如何产生“好而不同”的个体学习器。

ensemble learning=boosting+bagging+stacking

3、什么是Boosting算法？其算法三要素是什么？

Boosting 是一种将弱分离器 $f_i(x)$ 组合起来形成强分类器 $F(x)$ 的算法框架。

一般而言，Boosting算法有三个要素[1]：

1）函数模型：Boosting的函数模型是叠加型的，即 $F(x)=\sum_{i=1}^{k} f_i(x;\theta_i)$ ；

2）目标函数：选定某种损失函数 $E\{F(x)\}=E\{\sum_{i=1}^{k} f_i(x;\theta_i)\}$ 作为优化目标；

3）优化算法：贪婪地逐步优化，即 $\theta_m^*=\arg\min_{\theta_m} E\{\sum_{i=1}^{m-1} f_i(x;\theta_i^*)+f_m(x;\theta_m)\}$。

将上述框架中的 $f_i(x)$ 选为决策树，将 $E\{F(x)\}$ 选为指数损失函数，就可以得到AdaBoost算法。也就是说AdaBoost是Boosting算法框架中的一种实现。当然也有别的实现，比如LogitBoost算法等。

参考文献:[1]Additive Logistic Regression: a Statistical View of Boosting

根据boosting算法的三个要素，可以认为Adaboost算法是模型为加法模型，损失函数为指数函数，学习算法为前向分步算法时的二类分类学习方法。

4、Adaboost算法？前向分布算法？回归问题提升算法？梯度提升算法？

- Adaboost算法推导

算法 8.1（AdaBoost）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中 $x_i\in\mathcal{X}\subseteq\mathbf{R}^n$，$y_i\in\mathcal{Y}=\{-1,+1\}$；弱学习算法；

输出：最终分类器 $G(x)$.

（1）初始化训练数据的权值分布

$$D_1=(w_{11},\cdots,w_{1i},\cdots,w_{1N}),\quad w_{1i}=\frac{1}{N},\quad i=1,2,\cdots,N$$

（2）对 $m=1,2,\cdots,M$

（a）使用具有权值分布 D_m 的训练数据集学习，得到基本分类器

$$G_m(x):\mathcal{X}\to\{-1,+1\}$$

（b）计算 $G_m(x)$ 在训练数据集上的分类误差率

$$e_m=P(G_m(x_i)\neq y_i)=\sum_{i=1}^{N} w_{mi}I(G_m(x_i)\neq y_i) \qquad (8.1)$$

（c）计算 $G_m(x)$ 的系数

$$\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m} \tag{8.2}$$

这里的对数是自然对数.

（d）更新训练数据集的权值分布

$$D_{m+1} = (w_{m+1,1}, \cdots, w_{m+1,i}, \cdots, w_{m+1,N}) \tag{8.3}$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m}\exp(-\alpha_m y_i G_m(x_i)), \quad i = 1,2,\cdots,N \tag{8.4}$$

这里，Z_m 是规范化因子

$$Z_m = \sum_{i=1}^{N} w_{mi}\exp(-\alpha_m y_i G_m(x_i)) \tag{8.5}$$

它使 D_{m+1} 成为一个概率分布.

（3）构建基本分类器的线性组合

$$f(x) = \sum_{m=1}^{M}\alpha_m G_m(x) \tag{8.6}$$

得到最终分类器

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^{M}\alpha_m G_m(x)\right) \tag{8.7}$$

对 AdaBoost 算法作如下说明：

步骤（1） 假设训练数据集具有均匀的权值分布，即每个训练样本在基本分类器的学习中作用相同，这一假设保证第 1 步能够在原始数据上学习基本分类器 $G_1(x)$.

步骤（2） AdaBoost 反复学习基本分类器，在每一轮 $m = 1,2,\cdots,M$ 顺次地执行下列操作：

（a）使用当前分布 D_m 加权的训练数据集，学习基本分类器 $G_m(x)$.

（b）计算基本分类器 $G_m(x)$ 在加权训练数据集上的分类误差率：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{G_m(x_i)\neq y_i} w_{mi} \tag{8.8}$$

这里，w_{mi} 表示第 m 轮中第 i 个实例的权值，$\sum_{i=1}^{N} w_{mi} = 1$. 这表明，$G_m(x)$ 在加权的训练数据集上的分类误差率是被 $G_m(x)$ 误分类样本的权值之和，由此可以看出数据权值分布 D_m 与基本分类器 $G_m(x)$ 的分类误差率的关系.

（c）计算基本分类器 $G_m(x)$ 的系数 α_m. α_m 表示 $G_m(x)$ 在最终分类器中的重要性. 由式 (8.2) 可知，当 $e_m \leqslant \frac{1}{2}$ 时，$\alpha_m \geqslant 0$，并且 α_m 随着 e_m 的减小而增大，所以分类误差率越小的基本分类器在最终分类器中的作用越大.

（d）更新训练数据的权值分布为下一轮作准备．式 (8.4) 可以写成：

$$w_{m+1,i}=\begin{cases}\dfrac{w_{mi}}{Z_m}\mathrm{e}^{-\alpha_m}, & G_m(x_i)=y_i\\ \dfrac{w_{mi}}{Z_m}\mathrm{e}^{\alpha_m}, & G_m(x_i)\neq y_i\end{cases}$$

由此可知，被基本分类器 $G_m(x)$ 误分类样本的权值得以扩大，而被正确分类样本的权值却得以缩小．两相比较，误分类样本的权值被放大 $\mathrm{e}^{2\alpha_m}=\dfrac{e_m}{1-e_m}$ 倍．因此，误分类样本在下一轮学习中起更大的作用．不改变所给的训练数据，而不断改变训练数据权值的分布，使得训练数据在基本分类器的学习中起不同的作用，这是 AdaBoost 的一个特点．

步骤（3） 线性组合 $f(x)$ 实现 M 个基本分类器的加权表决．系数 α_m 表示了基本分类器 $G_m(x)$ 的重要性，这里，所有 α_m 之和并不为 1．$f(x)$ 的符号决定实例 x 的类，$f(x)$ 的绝对值表示分类的确信度．利用基本分类器的线性组合构建最终分类器是 AdaBoost 的另一特点．

AdaBoost 算法还有另一个解释，即可以认为 AdaBoost 算法是模型为加法模型、损失函数为指数函数、学习算法为前向分步算法时的二类分类学习方法．

- 前向分布算法推导

算法 8.2（前向分步算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$；损失函数 $L(y,f(x))$；基函数集 $\{b(x;\gamma)\}$；

输出：加法模型 $f(x)$．

（1）初始化 $f_0(x)=0$

（2）对 $m=1,2,\cdots,M$

（a）极小化损失函数

$$(\beta_m,\gamma_m)=\arg\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,f_{m-1}(x_i)+\beta b(x_i;\gamma)\right) \tag{8.16}$$

得到参数 β_m，γ_m

（b）更新

$$f_m(x)=f_{m-1}(x)+\beta_m b(x;\gamma_m) \tag{8.17}$$

（3）得到加法模型

$$f(x)=f_M(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m) \tag{8.18}$$

这样，前向分步算法将同时求解从 $m=1$ 到 M 所有参数 β_m，γ_m 的优化问题简化为逐次求解各个 β_m，γ_m 的优化问题．

- 回归问题的提升树算法

回归问题提升树使用以下前向分步算法：

$$
\begin{aligned}
&f_0(x)=0\\
&f_m(x)=f_{m-1}(x)+T(x;\Theta_m),\quad m=1,2,\cdots,M\\
&f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)
\end{aligned}
$$

在前向分步算法的第 m 步，给定当前模型 $f_{m-1}(x)$，需求解

$$\hat{\Theta}_m=\arg\min_{\Theta_m}\sum_{i=1}^{N}L(y_i,f_{m-1}(x_i)+T(x_i;\Theta_m))$$

得到 $\hat{\Theta}_m$，即第 m 棵树的参数.

当采用平方误差损失函数时，

$$L(y,f(x))=(y-f(x))^2$$

其损失变为

$$
\begin{aligned}
&L(y,f_{m-1}(x)+T(x;\Theta_m))\\
&\quad=[y-f_{m-1}(x)-T(x;\Theta_m)]^2\\
&\quad=[r-T(x;\Theta_m)]^2
\end{aligned}
$$

这里，

$$r=y-f_{m-1}(x)\tag{8.27}$$

是当前模型拟合数据的残差（residual）. 所以，对回归问题的提升树算法来说，只需简单地拟合当前模型的残差. 这样，算法是相当简单的. 现将回归问题的提升树算法叙述如下.

算法 8.3（回归问题的提升树算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i\in\mathcal{X}\subseteq\mathbf{R}^n$，$y_i\in\mathcal{Y}\subseteq\mathbf{R}$；

输出：提升树 $f_M(x)$.

（1）初始化 $f_0(x)=0$

（2）对 $m=1,2,\cdots,M$

（a）按式 (8.27) 计算残差

$$r_{mi}=y_i-f_{m-1}(x_i),\quad i=1,2,\cdots,N$$

（b）拟合残差 r_{mi} 学习一个回归树，得到 $T(x;\Theta_m)$

（c）更新 $f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$

（3）得到回归问题提升树

$$f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)$$

- 梯度提升算法

提升树利用加法模型与前向分步算法实现学习的优化过程．当损失函数是平方损失和指数损失函数时，每一步优化是很简单的．但对一般损失函数而言，往往每一步优化并不那么容易．针对这一问题，Freidman 提出了梯度提升（gradient boosting）算法．这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值

$$-\left[\frac{\partial L(y,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

作为回归问题提升树算法中的残差的近似值，拟合一个回归树．

算法 8.4（梯度提升算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i\in\mathcal{X}\subseteq\mathbf{R}^n$，$y_i\in\mathcal{Y}\subseteq\mathbf{R}$；损失函数 $L(y,f(x))$；

输出：回归树 $\hat{f}(x)$．

（1）初始化

$$f_0(x)=\arg\min_c\sum_{i=1}^{N}L(y_i,c)$$

（2）对 $m=1,2,\cdots,M$

（a）对 $i=1,2,\cdots,N$，计算

$$r_{mi}=-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

（b）对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶结点区域 R_{mj}，$j=1,2,\cdots,J$

（c）对 $j=1,2,\cdots,J$，计算

$$c_{mj}=\arg\min_c\sum_{x_i\in R_{mj}}L(y_i,f_{m-1}(x_i)+c)$$

（d）更新 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$

（3）得到回归树

$$\hat{f}(x)=f_M(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$$

算法第 1 步初始化，估计使损失函数极小化的常数值，它是只有一个根结点的树．第 2 (a) 步计算损失函数的负梯度在当前模型的值，将它作为残差的估计．对于平方损失函数，它就是通常所说的残差；对于一般损失函数，它就是残差的近似值．第 2 (b) 步估计回归树叶结点区域，以拟合残差的近似值．第 2 (c) 步利用线性搜索估计叶结点区域的值，使损失函数极小化．第 2 (d) 步更新回归树．第 3 步得到输出的最终模型 $\hat{f}(x)$．

5、Boosting 和 Adaboost 的关系和区别是什么?

提升（Boosting）是一种常用的统计学习方法，在分类问题中，它通过改变训练样本的权重，学习多个分类器（一般是弱分类器），并将这些分类器线性组合，最终提高分类器的性能。而针对于这种提升方法而言，需要回答两个问题，*一是在每一轮如何改变训练样本的权值或概率分布；二是如何将弱分类器组合成一个强分类器*。Adaboost属于Boosting一种，它可以很好的解决上述两个问题，针对第一个问题，Adaboost的做法是**提高那些被前一轮弱分类器错误分类样本的权值，而降低那些被正确分类的样本的权重**。从而使得那些被错误分类的样本由于其权值被加大而受到后一轮弱分类器的更多的关照（关注）。而针对第二个问题，Adaboost采取加权多数表决的方法，**加大分类误差率小的弱分类器的权重，使其在最终的分类器表决中起较大作用，减小分类误差率大的弱分类器的权重，使其在表决中起较小的作用**。总结起来，有两个权重的概念，一个是增大错误分类样本

的权重，一个增大分类误差率小的弱分类器的权重。

6、Adaboost与梯度提升的区别？

它们都是常用的提升方法，它们的模型都是加法模型并且优化学习方法采用前向分步算法；但梯度提升算法通**过加法模型组合各个单步模型时，加权系数是1**，即不按照像**Adaboost那样和当前分类错误率相关的权重来组合**，而是简单加和。提升树可以用于回归也可以用于分类，当用于回归时，损失函数为最小二乘（平方损失），用于分类时，可以采用指数损失函数，而Adaboost就是**采用指数损失函数用于二类分类问题**的一种特例。

梯度提升树的算法与原始的提升树（回归树）算法的核心区别主要在于残差计算这里。一般的提升树（回归）是**拟合当前模型的残差**去学习一个回归树，而梯度提升树是针对更一般的损失函数，它是**拟合损失函数的负梯度在当前模型的值（近似残差）**去学习一个回归树。

GBDT的变形和参数建议

GBDT的一个重要的参数就是每个DT（Decision Tree）的深度。类似于Adaboost，如果每次迭代时树都完全长成，那么其实就成为了一个基本的决策树，会导致过拟合，也失去了Boost算法的意义。在一些文章中，很多人推荐把树的深度设为4到8之间，并且认为6是个不错的数值。我觉得，这个也要看变量的个数、样本数、每次迭代的步长有关。这实际上是一个经验活，而且和每次的训练场景之间相关。

GBDT在实际运用时，常常有两种变体：

“Shrinkage”：事实上这是一种正则化（regularization）方法，为了进一步过拟合，在每次对残差估计进行迭代时，不直接加上当前步所拟合的残差，而是乘以一个系数。

即：fm = fm-1 + λ * 当前回归数残差

λ为1时，即为不加Shrinkage的一般GBDT。

有文章指出，10 < λ * 迭代次数（或者说数的数目）< 100，是一个比较合适的学习速率。但是一般这个速率常常被设成了0.1，或者0.05。

“Bagging”：每次迭代单步的树时，随机选一些样本的残差做拟合，而不是把所有样本的残差做拟合（常用的样本残差选取率为0.5-0.6）。这和随机森林的思想有类似之处。

7、什么是随机森林算法？

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。每棵树的按照如下规则生成：

1）如果训练集大小为N，对于每棵树而言，随机且有放回地从训练集中的抽取N个训练样本（这种采样方式称为bootstrap sample方法），作为该树的训练集；（每棵树的训练集都是不同的，而且里面包含重复的训练样本）

- 为什么要随机抽样训练集？

如果不进行随机抽样，每棵树的训练集都一样，那么最终训练出的树分类结果也是完全一样的，这样的话完全没有bagging的必要；

- 为什么要有放回地抽样？

如果不是有放回的抽样，那么每棵树的训练样本都是不同的，都是没有交集的，这样每棵树都是"有偏的"，都是绝对"片面的"（当然这样说可能不对），也就是说每棵树训练出来都是有很大的差异的；而随机森林最后分类取决于多棵树（弱分类器）的投票表决，这种表决应该是"求同"，因此使用完全不同的训练集来训练每棵树这样对最终分类结果是没有帮助的，这样无异于是"盲人摸象"。

2）如果每个样本的特征维度为M，指定一个常数m<<M，随机地从M个特征中选取m个特征子集，每次树进行分裂时，从这m个特征中选择最优的；

3）每棵树都尽最大程度的生长，并且没有剪枝过程。

一开始我们提到的随机森林中的“随机”就是指的这里的两个随机性。两个随机性的引入对随机森林的分类性能至关重要。由于它们的引入，使得随机森林不容易陷入过拟合，并且具有很好得抗噪能力（比如：对缺省值不敏感）。

减小特征选择个数m，树的相关性和分类能力也会相应的降低；增大m，两者也会随之增大。所以关键问题是如何选择最优的m（或者是范围），这也是随机森林唯一的一个参数。

8、Xgboost推导？

模型函数形式

给定数据集$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$，XGBoost进行additive training，学习K棵树，采用以下函数对样本进行预测：

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

这里 $\mathcal{F}$ 是假设空间，f(x)是回归树（CART）：

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

q(x)表示将样本x分到了某个叶子节点上，w是叶子节点的分数（leaf score），所以 $w_{q(x)}$ 表示回归树对样本的预测值

- 参数空间中的目标函数：

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

误差函数：我们的模型有多拟合数据

正则化项：惩罚复杂模型

误差函数可以是square loss，logloss等，正则项可以是L1正则，L2正则等。

Ridge Regression（岭回归）：$\sum_{i=1}^{n}\left(y_i - \theta^T x_i\right)^2 + \lambda\|\theta\|^2$

LASSO：$\sum_{i=1}^{n}\left(y_i - \theta^T x_i\right)^2 + \lambda\|\theta\|_1$

正则项

- XGBoost的目标函数（函数空间）

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

正则项对每棵回归树的复杂度进行了惩罚

- 相比原始的GBDT，XGBoost的目标函数多了正则项，使得学习出来的模型更加不容易过拟合。
- 有哪些指标可以衡量树的复杂度？
  树的深度，内部节点个数，叶子节点个数(T)，叶节点分数(w)...
  XGBoost采用的：

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

对叶子节点个数进行惩罚，相当于在训练过程中做了剪枝

误差函数的二阶泰勒展开

- 第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：

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

- 此时目标函数可写作：

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

公式中 $y_i, \tilde{y}_i^{(t-1)}$ 都已知，模型要学习的只有第t棵树 f_t

- 将误差函数在 $\tilde{y}_i^{(t-1)}$ 处进行二阶泰勒展开：

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

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

误差函数的二阶泰勒展开

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 将公式中的常数项去掉，得到：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 把 $f_t, \Omega(f_t)$ 写成树结构的形式，即把下式代入目标函数中

$$f(\mathbf{x}) = w_{q(\mathbf{x})} \qquad \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

- 得到：

$$\begin{aligned} L^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \lambda\frac{1}{2}\sum_{j=1}^{T} w_j^2 \end{aligned}$$

误差函数的二阶泰勒展开

$$\begin{aligned} L^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \lambda\frac{1}{2}\sum_{j=1}^{T} w_j^2 \end{aligned}$$

对样本累加　　对叶节点累加

- 怎么统一起来？

  定义每个叶节点j上的样本集合为 $I_j = \{i|q(x_i) = j\}$

  则目标函数可以写成按叶节点累加的形式：

$$\begin{aligned} L^{(t)} &= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T \\ &= \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T \end{aligned}$$

误差函数的二阶泰勒展开

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

- 如果确定了树的结构（即q(x)确定），为了使目标函数最小，可以令其导数为0，解得每个叶节点的最优预测分数为：

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

代入目标函数，得到最小损失为：

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

回归树的学习策略

- 当回归树的结构确定时，我们前面已经推导出其最优的叶节点分数以及对应的最小损失值，问题是怎么确定树的结构？
  暴力枚举所有可能的树结构，选择损失值最小的 - NP难问题
  贪心法，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

- **分裂前后的增益怎么计算？**
  ID3算法采用信息增益
  C4.5算法采用信息增益比
  CART采用Gini系数
  XGBoost呢？

XGBoost的打分函数

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

标红部分衡量了每个叶子节点对总体损失的的贡献，我们希望损失越小越好，则标红部分的值越大越好。

因此，对一个叶子节点进行分裂，分裂前后的增益定义为：

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

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最大的进行分割

树节点分裂方法（Split Finding）

- 近似方法
  对于每个特征，只考察分位点，减少计算复杂度
  - Global：学习每棵树前，提出候选切分点
  - Local：每次分裂前，重新提出候选切分点

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk}>s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk}>s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

树节点分裂方法（Split Finding）

- 暴力枚举

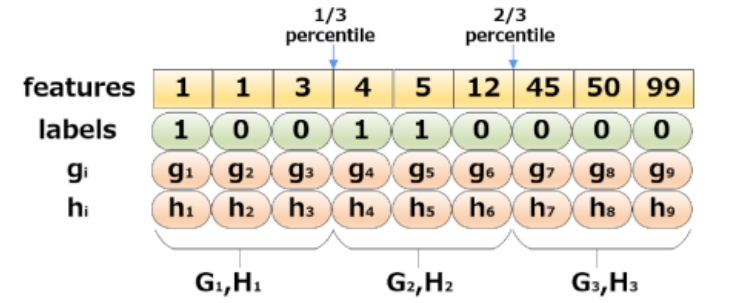
  遍历所有特征的所有可能的分割点，计算gain值，选取值最大的（feature，value）去分割

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

树节点分裂方法（Split Finding）

- 近似方法举例：三分位数



$$Gain = \max\{Gain, \frac{G_1^2}{H_1+\lambda} + \frac{G_{23}^2}{H_{23}+\lambda} - \frac{G_{123}^2}{H_{123}+\lambda} - \gamma, \frac{G_{12}^2}{H_{12}+\lambda} + \frac{G_3^2}{H_3+\lambda} - \frac{G_{123}^2}{H_{123}+\lambda} - \gamma\}$$

树节点分裂方法（Split Finding）

- 实际上XGBoost不是简单地按照样本个数进行分位，而是以二阶导数值作为权重(论文中的Weighted Quantile Sketch)，比如：

features	1	1	3	4	5	12	45	50	99
hi	0.1	0.1	0.1	0.1	0.1	0.1	0.4	0.2	0.6

1/3 percentile　2/3 percentile

- 为什么用hi加权？

  把目标函数整理成以下形式，可以看出hi有对loss加权的作用

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(\mathbf{x}_i) - g_i/h_i)^2 + \Omega(f_t) + constant,$$

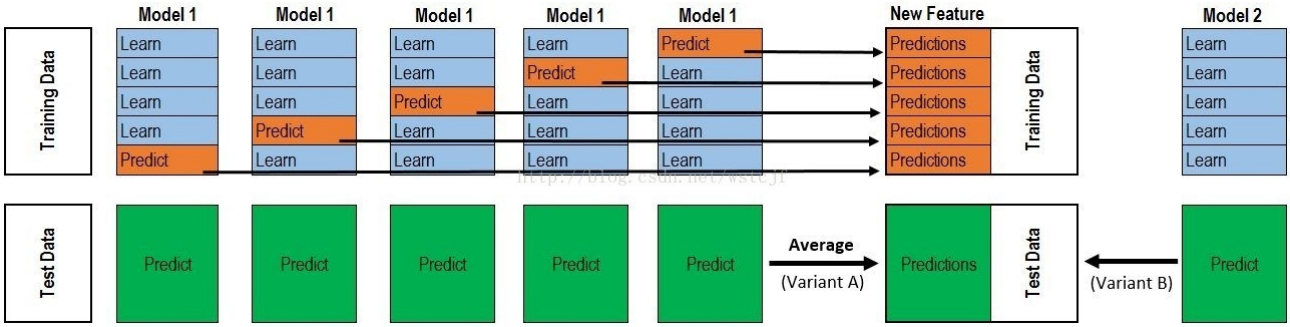
缺失值处理

当特征出现缺失值时，XGBoost可以学习出默认的节点分裂方向

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k$, *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k$, *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

9、stacking的方法具体是？



上半部分是用一个基础模型进行5折交叉验证，如：用XGBoost作为基础模型Model1，5折交叉验证就是先拿出四折作为training data，另外一折作为testing data。注意：在stacking中此部分数据会用到整个traing set。如：假设我们整个training set包含10000行数据，testing set包含2500行数据，那么每一次交叉验证其实就是对training set进行划分，在每一次的交叉验证中

training data将会是8000行，testing data是2000行。

每一次的交叉验证包含两个过程，1. 基于training data训练模型；2. 基于training data训练生成的模型对testing data进行预测。在整个第一次的交叉验证完成之后我们将会得到关于当前testing data的预测值，这将会是一个一维2000行的数据，记为a1。注意！在这部分操作完成后，我们还要对数据集原来的整个testing set进行预测，这个过程会生成2500个预测值，这部分预测值将会作为下一层模型testing data的一部分，记为b1。因为我们进行的是5折交叉验证，所以以上提及的过程将会进行五次，最终会生成针对testing set数据预测的5列2000行的数据a1,a2,a3,a4,a5，对testing set的预测会是5列2500行数据b1,b2,b3,b4,b5。

在完成对Model1的整个步骤之后，我们可以发现a1,a2,a3,a4,a5其实就是对原来整个training set的预测值，将他们拼凑起来，会形成一个10000行一列的矩阵，记为A1。而对于b1,b2,b3,b4,b5这部分数据，我们将各部分相加取平均值，得到一个2500行一列的矩阵，记为B1。

以上就是stacking中一个模型的完整流程，stacking中同一层通常包含多个模型，假设还有Model2: LR，Model3：RF，Model4: GBDT，Model5：SVM，对于这四个模型，我们可以重复以上的步骤，在整个流程结束之后，我们可以得到新的A2,A3,A4,A5,B2,B3,B4,B5矩阵。

在此之后，我们把A1,A2,A3,A4,A5并列合并得到一个10000行五列的矩阵作为training data，B1,B2,B3,B4,B5并列合并得到一个2500行五列的矩阵作为testing data。让下一层的模型，基于他们进一步训练。

以上即为stacking的完整步骤！

10、机器学习算法中GBDT和XGBOOST的区别有哪些？

1）传统GBDT以CART作为基分类器，xgboost还支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

2）传统GBDT在优化时只用到一阶导数信息，xgboost则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。

3）xgboost在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性。

4）Shrinkage（缩减），相当于学习速率（xgboost中的eta）。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（补充：传统GBDT的实现也有学习速率）

5）列抽样（column subsampling）即特征抽样。xgboost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是xgboost异于传统gbdt的一个特性。

6）对缺失值的处理。对于特征的值有缺失的样本，xgboost可以自动学习出它的分裂方向。

7）节点分裂的方式不同，gbdt是用的gini系数，xgboost是经过优化推导后的

8）xgboost工具支持并行。boosting不是一种串行的结构吗?怎么并行的？注意xgboost的并行不是tree粒度的并行，xgboost也是一次迭代完才能进行下一次迭代的（第t次迭代的代价函数里包含了前面t-1次迭代的预测值）。xgboost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），xgboost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

9）可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。

10）多种语言封装支持。

11、xgboost/gbdt在调参时为什么树的深度很少就能达到很高的精度？

用xgboost/gbdt在在调参的时候把树的最大深度调成6就有很高的精度了。但是用DecisionTree/RandomForest的时候需要把树的深度调到15或更高。用RandomForest所需要的树的深度和DecisionTree一样我能理解，因为它是用bagging的方法把DecisionTree组合在一起，相当于做了多次DecisionTree一样。但是xgboost/gbdt仅仅用梯度上升法就能用6个节点的深度达到很高的预测精度，使我惊讶到怀疑它是黑科技了。请问下xgboost/gbdt是怎么做到的？它的节点和一般的DecisionTree不同吗？

这是一个非常好的问题，题主对各算法的学习非常细致透彻，问的问题也关系到这两个算法的本质。这个问题其实并不是一个很简单的问题，我尝试用我浅薄的机器学习知识对这个问题进行回答。

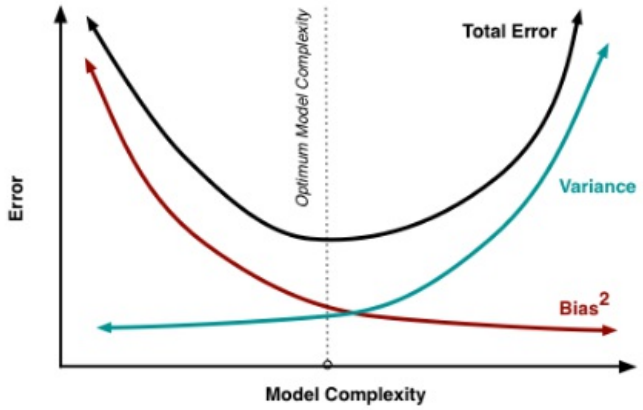
一句话的解释，来自周志华老师的机器学习教科书（机器学习-周志华）：**Boosting主要关注降低偏差**，因此Boosting能基于泛化性能相当弱的学习器构建出很强的集成；**Bagging主要关注降低方差**，因此它在不剪枝的决策树、神经网络等学习器上效用更为明显。

随机森林(random forest)和GBDT都是属于集成学习（ensemble learning)的范畴。集成学习下有两个重要的策略Bagging和Boosting。

Bagging算法是这样做的：每个分类器都随机从原样本中做有放回的采样，然后分别在这些采样后的样本上训练分类器，然后再把这些分类器组合起来。简单的多数投票一般就可以。其代表算法是随机森林。Boosting的意思是这样，他通过迭代地训练一系列的分类器，每个分类器采用的样本分布都和上一轮的学习结果有关。其代表算法是AdaBoost, GBDT。

其实就机器学习算法来说，其泛化误差可以分解为两部分，偏差（bias)和方差(variance)。这个可由下图的式子导出（这里用到了概率论公式D(X)=E(X^2)-[E(X)]^2）。**偏差指的是算法的期望预测与真实预测之间的偏差程度，反应了模型本身的拟合能力；方差度量了同等大小的训练集的变动导致学习性能的变化，刻画了数据扰动所导致的影响**。这个有点儿绕，不过你一定知道过拟合。

如下图所示，当模型越复杂时，拟合的程度就越高，模型的训练偏差就越小。但此时如果换一组数据可能模型的变化就会很大，即模型的方差很大。所以模型过于复杂的时候会导致过拟合。



当模型越简单时，即使我们再换一组数据，最后得出的学习器和之前的学习器的差别就不那么大，模型的方差很小。还是因为模型简单，所以偏差会很大。

也就是说，当我们训练一个模型时，偏差和方差都得照顾到，漏掉一个都不行。

对于Bagging算法来说，由于我们会并行地训练很多不同的分类器的目的就是降低这个方差(variance) ,因为采用了相互独立的基分类器多了以后，h的值自然就会靠近.所以对于每个基分类器来说，目标就是如何降低这个偏差（bias),所以我们会采用深度很深甚至不剪枝的决策树。

对于Boosting来说，每一步我们都会在上一轮的基础上更加拟合原数据，所以可以保证偏差（bias），所以对于每个基分类器来说，问题就在于如何选择variance更小的分类器，即更简单的分类器，所以我们选择了深度很浅的决策树。

12、为什么基于 tree-ensemble 的机器学习方法，在实际的 kaggle 比赛中效果非常好？

通常，解释一个机器学习模型的表现是一件很复杂事情，而这篇文章尽可能用最直观的方式来解释这一问题。我主要从三个方面来回答楼主这个问题。

1. 理论模型 （站在 vc-dimension 的角度）
2. 实际数据
3. 系统的实现 （主要基于 xgboost）

通常决定一个机器学习模型能不能取得好的效果，以上三个方面的因素缺一不可。

（1）站在理论模型的角度统计机器学习里经典的 vc-dimension 理论告诉我们：一个机器学习模型想要取得好的效果，这个模型需要满足以下两个条件：

1. 模型在我们的训练数据上的表现要不错，也就是 trainning error 要足够小。
2. 模型的 vc-dimension 要低。换句话说，就是模型的自由度不能太大，以防overfit.当然，这是我用大白话描述出来的，真正的 vc-dimension 理论需要经过复杂的数学推导，推出 vc-bound. vc-dimension 理论其实是从另一个角度刻画

了一个我们所熟知的概念，那就是 bias variance trade-off.

好，现在开始让我们想象一个机器学习任务。对于这个任务，一定会有一个 “上帝函数” 可以完美的拟合所有数据（包括训练数据，以及未知的测试数据）。很可惜，这个函数我们肯定是不知道的 （不然就不需要机器学习了）。我们只可能选择一个 “假想函数” 来 逼近 这个 “上帝函数”，我们通常把这个 “假想函数” 叫做 hypothesis.

在这些 hypothesis 里，我们可以选择 svm, 也可以选择 logistic regression. 可以选择单棵决策树，也可以选择 tree-ensemble (gbdt, random forest). 现在的问题就是，为什么 tree-ensemble 在实际中的效果很好呢？

区别就在于 “模型的可控性”。

先说结论，tree-ensemble 这样的模型的可控性是好的，而像 LR 这样的模型的可控性是不够好的（或者说，可控性是没有 tree-ensemble 好的）。为什么会这样？别急，听我慢慢道来。

我们之前说，当我们选择一个 hypothsis 后，就需要在训练数据上进行训练，从而逼近我们的 “上帝函数”。我们都知道，对于 LR 这样的模型。如果 underfit，我们可以通过加 feature，或者通过高次的特征转换来使得我们的模型在训练数据上取得足够高的正确率。而对于 tree-enseble 来说，我们解决这一问题的方法是通过训练更多的 “弱弱” 的 tree. 所以，这两类模型都可以把 training error 做的足够低，也就是说模型的表达能力都是足够的。但是这样就完事了吗？没有，我们还需要让我们的模型的 vc-dimension 低一些。而这里，重点来了。在 tree-ensemble 模型中，通过加 tree 的方式，对于模型的 vc-dimension 的改变是比较小的。而在 LR 中，初始的维数设定，或者说特征的高次转换对于 vc-dimension 的影响都是更大的。换句话说，tree-ensemble 总是用一些 “弱弱” 的树联合起来去逼近 “上帝函数”，一次一小步，总能拟合的比较好。而对于 LR 这样的模型，我们很难去猜到这个“上帝函数”到底长什么样子（到底是2次函数还是3次函数？上帝函数如果是介于2次和3次之间怎么办呢？）。所以，一不小心我们设定的多项式维数高了，模型就 “刹不住车了”。俗话说的好，步子大了，总会扯着蛋。这也就是我们之前说的，tree-ensemble 模型的可控性更好，也即更不容易 overfit.

（2）站在数据的角度

除了理论模型之外, 实际的数据也对我们的算法最终能取得好的效果息息相关。kaggle 比赛选择的都是真实世界中的问题。所以数据多多少少都是有噪音的。而基于树的算法通常抗噪能力更强。比如在树模型中，我们很容易对缺失值进行处理。除此之外，基于树的模型对于 categorical feature 也更加友好。

除了数据噪音之外，feature 的多样性也是 tree-ensemble 模型能够取得更好效果的原因之一。通常在一个kaggle任务中，我们可能有年龄特征，收入特征，性别特征等等从不同 channel 获得的特征。而特征的多样性也正是为什么工业界很少去使用 svm 的一个重要原因之一，因为 svm 本质上是属于一个几何模型，这个模型需要去定义 instance 之间的 kernel 或者 similarity（对于linear svm 来说，这个similarity 就是内积）。这其实和我们在之前说过的问题是相似的，我们无法预先设定一个很好的similarity。这样的数学模型使得 svm 更适合去处理 “同性质”的特征，例如图像特征提取中的 lbp 。而从不同 channel 中来的 feature 则更适合 tree-based model, 这些模型对数据的 distributation 通常并不敏感。

（3）站在系统实现的角度

除了有合适的模型和数据，一个良好的机器学习系统实现往往也是算法最终能否取得好的效果的关键。一个好的机器学习系统实现应该具备以下特征：

1. 正确高效的实现某种模型。我真的见过有些机器学习的库实现某种算法是错误的。而高效的实现意味着可以快速验证不同的模型和参数。
2. 系统具有灵活、深度的定制功能。
3. 系统简单易用。
4. 系统具有可扩展性, 可以从容处理更大的数据。

到目前为止，xgboost 是我发现的唯一一个能够很好的满足上述所有要求的 machine learning package. 在此感谢青年才俊 陈天奇。

在效率方面，xgboost 高效的 c++ 实现能够通常能够比其它机器学习库更快的完成训练任务。

在灵活性方面，xgboost 可以深度定制每一个子分类器，并且可以灵活的选择 loss function（logistic，linear，softmax 等等）。除此之外，xgboost还提供了一系列在机器学习比赛中十分有用的功能，例如 early-stop， cv 等等在易用性方面，xgboost 提供了各种语言的封装，使得不同语言的用户都可以使用这个优秀的系统。

最后，在可扩展性方面，xgboost 提供了分布式训练（底层采用 rabit 接口），并且其分布式版本可以跑在各种平台之上，例如 mpi, yarn, spark 等等。

有了这么多优秀的特性，自然这个系统会吸引更多的人去使用它来参加 kaggle 比赛。

综上所述，理论模型，实际的数据，良好的系统实现，都是使得 tree-ensemble 在实际的 kaggle 比赛中“屡战屡胜”的原因。