# 简介

【参考】

- csdn - GBDT（MART） 迭代决策树入门教程 | 简介
- 简书 - GBDT：梯度提升决策树

GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree)或者 GBRT (Gradient Boosted Regression Trees)，是一种**迭代**的决策树算法，该算法由**多棵决策树组成，所有树的结论累加起来做最终答案**。它在被提出之初就和SVM一起被认为是泛化能力（generalization)较强的算法。近些年更因为被用于搜索排序的机器学习模型而引起大家关注，也可以用于生态学（ecology）。

GBDT中的树是**回归树**（不是分类树），GBDT用来做**回归预测**，调整后也可以用于分类。

GBDT的思想使其具有天然优势可以发现多种有区分性的特征以及特征组合。业界中，Facebook使用其来自动发现有效的特征、特征组合，来作为LR模型中的特征，以提高 CTR预估（Click-Through Rate Prediction）的准确性。GBDT在淘宝的搜索及预测业务上也发挥了重要作用。

GBDT主要由三个概念组成：Regression Decistion Tree（即DT)，Gradient Boosting（即GB)，Shrinkage (算法的一个重要演进分枝，目前大部分源码都按该版本实现）。

# DT 回归树

提起决策树（DT, Decision Tree) 绝大部分人首先想到的就是 $C4.5$ 分类决策树。但如果一开始就把GBDT中的树想成分类树，那学习的过程中就会非常的迷茫，因此不要认为 GBDT 就是多棵分类树。要理解 Regression Decision Tree 就需要了解下决策树的分类。以下面的数据为例：

| 编号 | 网购金额 | 上网 | 教育 | 年龄 |
|---|---|---|---|---|
| A | 300 | 0.5h | 高一 | 14 |
| B | 800 | 1.5h | 高三 | 16 |
| C | 1200 | 全天上网 | 应届毕业 | 24 |
| D | 3000 | 晚上上网 | 工作两年 | 26 |

## 分类与回归决策树

【参考】

- csdn - 分类树和回归树的区别

决策树分为两大类，回归树和分类树。前者用于预测实数值，如明天的温度、用户的年龄、网页的相关程度；后者用于分类标签值，如晴天/阴天/雾/雨、用户性别、网页是否是垃圾页面。

- 回归树：总体流程也是类似，区别在于，回归树的每个节点（不一定是叶子节点）都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值 $\tilde{a}$（在根节点时为 20 ）。划分时穷举每一个 特征 的每个阈值找最好的分割点，但衡量最好的标准不再是最大熵（搬运系列：决策树算法 中进行的是分类，因此使用的是信息增益和增益率），而是最小化均方差，即(每个人的年龄-预测年龄)^2 的总和 / N（ $\frac{1}{N}\sum_{i=1}^{N}(x_i-\tilde{a})^2$ ）。也就是被预测出错的人数越多，错的越离

谱，均方差就越大（方差越大越分散，越小越聚集），通过最小化均方差能够找到最可靠的划分依据。划分直到每个叶子节点上人的年龄都唯一或者达到预设的终止条件(如叶子个数上限)，若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。

也就是说，**回归树使用最小均方差划分节点；每个节点样本的均值作为测试样本的回归预测值。**

- 分类树：以C4.5分类树为例，C4.5分类树在每次分枝时，是穷举每一个特征的每一个阈值（连续属性，如西瓜数据集中的密度与含糖量，上面数据集中的年龄和购物金额，通过 C4.5 采用二分法），找到使得按照特征<=阈值（密度<=0.381，年龄<=20》），和特征>阈值分成的两个分枝的熵最大的阈值（熵的计算和定义参见搬运系列：决策树算法），按照该标准划分得到两个新节点，用同样方法继续分枝直到所有人都被分入性别唯一的叶子节点，或达到预设的终止条件，若最终叶子节点中的类别不唯一，则以多数人的性别作为该叶子节点的类别。

也就是说， **分类树使用信息增益(ID3)或增益率(C4.5)来划分节点；每个节点样本的类别情况投票决定测试样本的类别。**

这里要强调的是，前者的结果加减是有意义的，如10岁+5岁-3岁=12岁，后者则无意义，如男+男+女=到底是男是女？ GBDT的核心在于累加所有树的结果作为最终结果，就像前面对年龄的累加（-3是加负3），而分类树的结果显然是没办法累加的，所以**GBDT中的树都是回归树，不是分类树**，这点对理解GBDT相当重要（尽管GBDT调整后也可用于分类但不代表GBDT的树是分类树）。

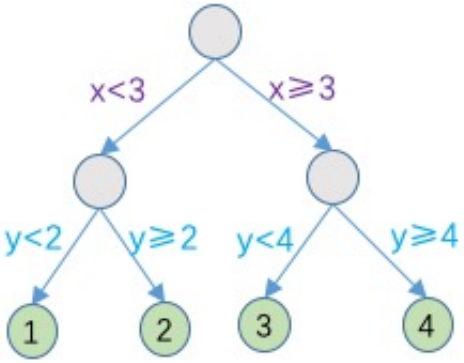
# 回归树（Regression Tree）

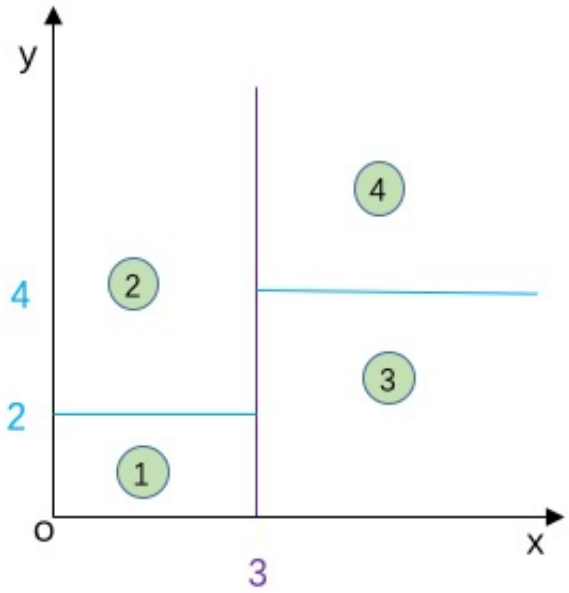
【参考】

- csdn - Regression Tree 回归树
- 简书 - CART 分类与回归树

树形算法包含了随机森林、GBDT、XGBoost等，树形算法的基础就是决策树。决策树因其易理解、易构建、速度快的特性，被广泛应用于统计学、数据挖掘、机器学习领域。

决策树实际上是将空间用超平面进行划分的一种方法，每次分割的时候，都将当前的空间一分为二， 这样使得每一个叶子节点都是在空间中的一个不相交的区域，在进行决策的时候，会根据输入样本每一维特征的值，一步一步往下，最后使得样本落入N个区域中的一个（假设有N个叶子节点），如下图所示：



三种比较常见的分类决策树分支划分方式包括：ID3, C4.5, CART，可参见搬运系列：决策树算法

分类与回归树（classificationandregressiontree, CART)模型由Breiman等人在1984年提出，是应用广泛的决策树学习方法。CART同样由特征选择、树的生成及剪枝组成，既可以用于分类也可以用于回归，只是选择评价的指标不同：

- **分类问题：选择 Gini 指数来评价**
- **回归问题：选择最小二乘法（LSD）或者最小绝对偏差（LAD）**

如下面的算法使用的就是最小二乘法。

## 原理

既然是决策树，那么必然会存在以下两个核心问题：**如何选择划分点？如何决定叶节点的输出值?**

一个回归树对应着输入空间（即特征空间）的一个划分以及在划分单元上的输出值。分类树中，我们采用信息论中的方法，通过计算选择最佳划分点。而在回归树中，采用的是**启发式**的方法。假如我们有 n 个特征，每个特征有$s_i(i \in (1,n))$个取值，那我们遍历所有特征，尝试该特征所有取值，对空间进行划分，直到取到特征 j 的取值 s，使得损失函数最小，这样就得到了一个划分点。描述该过程的公式如下：

$$\min_{j,s}\left[\min_{c_1} Loss\left(y_i, c_1\right)+\min_{c_2} Loss\left(y_i, c_2\right)\right]$$

假设将输入空间划分为 M 个单元：$R_1, R_2, \ldots, R_m$，如上图输入空间被划分为了 4 个单元， 那么每个区域的输出值就是：$c_m = average(y_i|x_i \in R_m)$也就是该区域内所有点 y 值的平均数。

s 值的选择可以参考 《搬运系列：决策树算法 连续值的处理》一节提到的连续值离散化技术。

## 算法过程

最小二乘法回归树生成算法
**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 $(c_1, c_2)$，并决定每个子区域上的输出值，构建二叉决策树：
(1) 选择最优切分特征 $j$ 与划分点 $s$(特征值)，求解

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

遍历 $j$（特征），对固定切分特征 $j$ 遍历切分点 $s$（特征值），取使得上式达到最小值的 $(j, s)$ 对。

(2)对选定的 $(j, s)$ 对，划分区域并决定相应的输出值：

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

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

(3) 继续对两个子区域调用步骤(1)(2)，直到满足停止条件
(4) 将输入空间划分为 M 个区域，$R_1, R_2, \ldots, R_m$，生成决策树：

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

## 示例

有如下数据集：

| x | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 5.56 | 5.7 | 5.91 | 6.4 | 6.8 | 7.05 | 8.9 | 8.7 | 9 | 9.05 |

1. 选择最优切分变量j与最优切分点s

在本数据集中，只有一个特征，因此最优划分特征自然是x。

接下来我们考虑9个切分点 s = $[1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]$。损失函数定义为平方损失函数 $Loss(y, f(x)) = (f(x) - y)^2$ ，将上述9个切分点一依此代入下面的公式，其中 $c_m = average(y_i \mid x_i \in R_m)$：

$$\min_{j,s} \left[ \min_{c_1} Loss(y_i, c_1) + \min_{c_2} Loss(y_i, c_2) \right]$$

即：

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

例如，取 s=1.5 。此时 $R_1 = \{1\}, R_2 = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$，这两个区域的输出值分别为：$c_1 = 5.56, c_2 = \frac{1}{9}(5.7 + 5.91 + 6.4 + 6.8 + 7.05 + 8.9 + 8.7 + 9 + 9.05) = 7.50$。之后按照此法依次计算 s 剩下的八个值，得到下表：

| s | 1.5 | 2.5 | 3.5 | 4.5 | 5.5 | 6.5 | 7.5 | 8.5 | 9.5 |
|---|---|---|---|---|---|---|---|---|---|
| $c_1$ | 5.56 | 5.63 | 5.72 | 5.89 | 6.07 | 6.24 | 6.62 | 6.88 | 7.11 |
| $c_2$ | 7.5 | 7.73 | 7.99 | 8.25 | 8.54 | 8.91 | 8.92 | 9.03 | 9.05 |

把在每个分割点求得的 $c_1, c_2$ 代入到上面的公式，最小的划分点，如以分割点 s=1.5为例，此时的 $c_1 = 5.56, c_2 = 7.5$，那么因为$R_1 = \{1\}$ 只含有一个值，因此损失为$l_1 = (5.56 - 5.56)^2 = 0$；$R_2 = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ 含有九个值，因此：

$$\begin{aligned} l_2 = &(5.7 - 7.5)^2 + (5.91 - 7.5)^2 + (6.4 - 7.5)^2 + (6.8 - 7.5)^2 \\ &+ (7.05 - 7.5)^2 + (8.9 - 7.5)^2 + (8.7 - 7.5)^2 + (9 - 7.5)^2 + (9.05 - 7.5)^2 = 15.72 \end{aligned}$$

因此在 s=1.5 的切分点的损失为 $m(s=1.5)=l_1+l_2=0+15.72=15.72$。同理计算 m(s=2.5)=12.07、m(s=3.5)=8.36等，最终得到如下的划分点损失表格：

| s | 1.5 | 2.5 | 3.5 | 4.5 | 5.5 | 6.5 | 7.5 | 8.5 | 9.5 |
|---|---|---|---|---|---|---|---|---|---|
| m(s) | 15.72 | 12.07 | 8.36 | 5.78 | 3.91 | 1.93 | 8.01 | 11.73 | 15.74 |

可以看到在 s=6.5 的时候，m(s) 取得最小值，因此第一个划分点为(j,s) = (x,6.5)。之后，进行算法的第二步，确定划分区域与输出值：

- 划分属性为 x，划分值为6.5，即将 x <= 6.5 的划分一个节点，x > 6.5 的划分一个节点，因此有两个区域为：$R_1=\{1,2,3,4,5,6\}, R_2=\{7,8,9,10\}$
- 每个区域的输出值为 $c_m=average(y_i|x_i\in R_m)$ 对 y 值求平均
  $c_1=\frac{1}{6}(5.56+5.7+5.91+6.4+6.8+7.05)=6.24, c_2=8.91$

接下来对$R_1, R_2$ 重复上面的过程，以对$R_1$继续划分为例，$R_1=\{1,2,3,4,5,6\}$：

| x | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| y | 5.56 | 5.7 | 5.91 | 6.4 | 6.8 | 7.05 |

取切分点是$s=[1.5,2.5,3.5,4.5,5.5]$，则各区域的输出值c如下表:

| s | 1.5 | 2.5 | 3.5 | 4.5 | 5.5 |
|---|---|---|---|---|---|
| $c_1$ | 5.56 | 5.63 | 5.72 | 5.89 | 6.07 |
| $c_2$ | 6.37 | 6.54 | 6.75 | 6.93 | 7.05 |

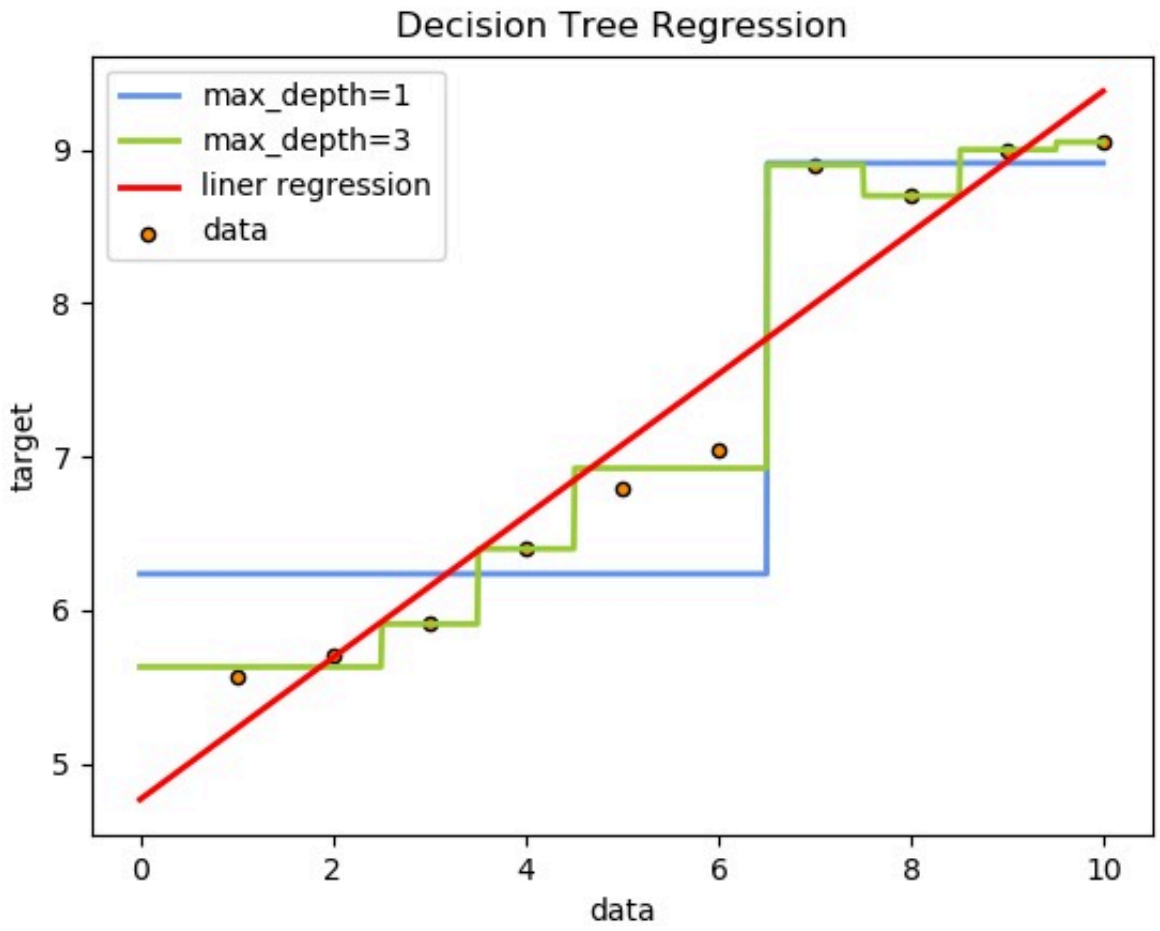
计算m(s)：

| s | 1.5 | 2.5 | 3.5 | 4.5 | 5.5 |
|---|---|---|---|---|---|
| m(s) | 1.3087 | 0.754 | 0.2771 | 0.4368 | 1.0644 |

在 s=3.5 时 m(s) 最小，以此为划分点。之后不断重复这个过程，知道达到停止条件为止。

# 总结

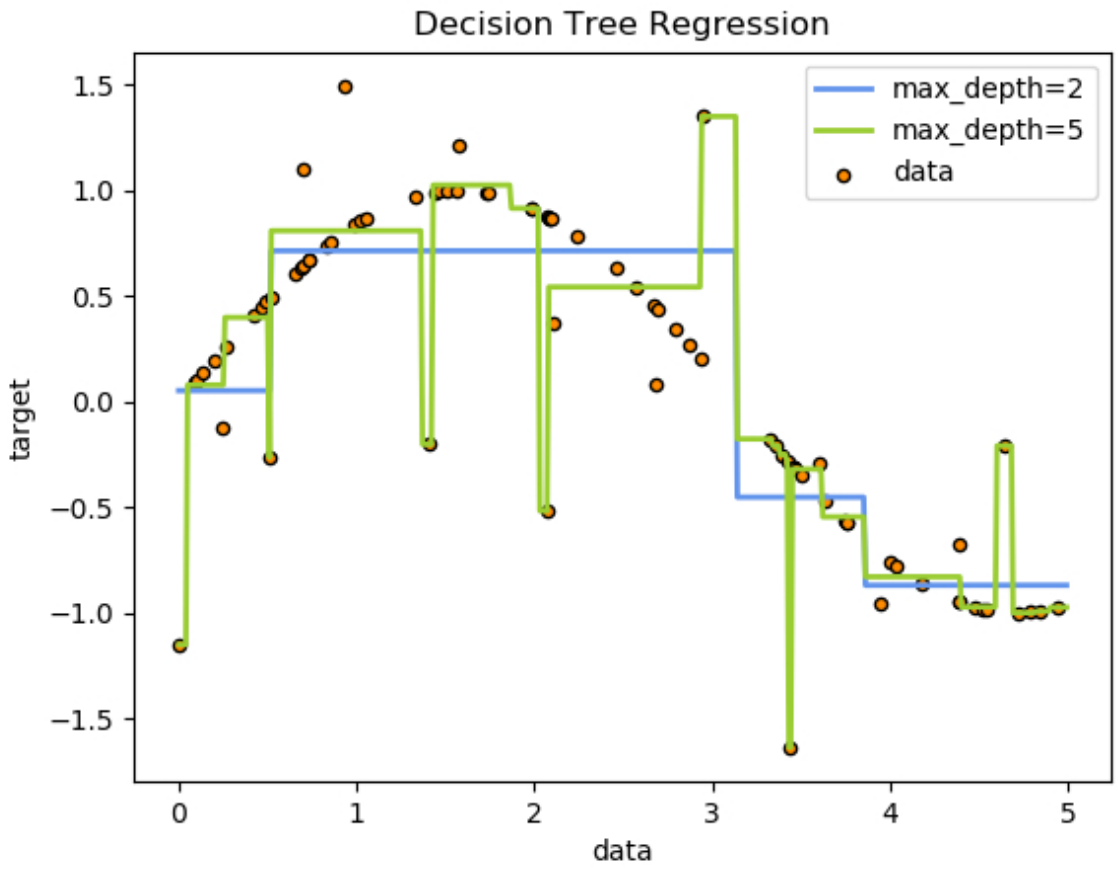
回归树与线性回归的区别：




实际上，回归树总体流程类似于分类树，分枝时穷举每一个特征的每一个阈值，来寻找最优切分特征j和最优切分点s，衡量的方法是平方误差最小化。分枝直到达到预设的终止条件(如叶子个数上限)就停止。

当然，处理具体问题时，单一的回归树肯定是不够用的。可以利用集成学习中的boosting框架，对回归树进行改良升级，得到的新模型就是提升树（Boosting Decision Tree），在进一步，可以得到梯度提升树（Gradient Boosting Decision Tree，GBDT），再进一步可以升级到XGBoost。

下面是 sklearn 官网的示例：Decision Tree Regression




# GBDT

【参考】

- medium - Gradient Boosting from scratch
- 个站 - A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning

## 梯度迭代(GB)

梯度迭代即 Gradient Boosting，Boosting。

迭代，即通过迭代多棵树来共同决策。这怎么实现呢？难道是每棵树独立训练一遍，比如A这个人，第一棵树认为是10岁，第二棵树认为是0岁，第三棵树认为是20岁，我们就取平均值10岁做最终结论？当然不是！且不说投票方法并不是GBDT，只要训练集不变，独立训练三次的三棵树必定完全相同，这样做完全没有意义。之前说过，GBDT是把所有树的**结论累加**起来做最终结论的，所以可以想到每棵树的结论并不是年龄本身，而是**年龄的一个累加量。GBDT的核心就在于，每一棵树学的是之前所有树结论和的残差**（想想 ResNet），这个**残差就是一个加预测值后能得真实值的累加量**。比如A的真实年龄是18岁，但第一棵树的预测年龄是12岁，差了6岁，即残差为6岁。那么在第二棵树里我们把A的年龄设为6岁去学习，如果第二棵树真的能把A分到6岁的叶子节点，那累加两棵树的结论就是A的真实年龄；如果第二棵树的结论是5岁，则A仍然存在1岁的残差，第三棵树里A的年龄就变成1岁，继续学。这就是Gradient Boosting在GBDT中的意义。

## 残差学习

【参考】

- 机器学习-一文理解GBDT的原理-20171001

假设现在你有样本集 $(x_1,y_1),(x_2,y_2),...(x_n,y_n)$，然后你用一个模型，如 $F(x)$ 去拟合这些数据，使得这批样本的平方损失函数（即 $\frac{1}{2}\sum_0^n(y_i-F(x_i))^2$ ）最小。但是你发现虽然模型的拟合效果很好，但仍然有一些差距，比如预测值 $F(x_1)=0.8$，而真实值$y_1=0.9$，$F(x_2)=1.4$，$y_2=1.3$等等。另外你不允许更改原来模型 $F(x)$ 的参数，那么你有什么办法进一步来提高模型的拟合能力呢。

既然不能更改原来模型的参数，那么意味着必须在原来模型的基础之上做改善，那么直观的做法就是建立一个新的模型 $f(x)$ 来拟合 $F(x)$ 未完全拟合真实样本的残差，即 $y-F(X)$ 。所以对于每个样本来说，拟合的样本集就变成了： $(x_1,y_1-F(x_1)),(x_2,y_2-F(x_2)),...(x_n,y_n-F(x_n))$

## 基于残差的 GBDT

在第一部分， $y_i-F(x_i)$ 被称为残差，这一部分也就是前一模型 $F(x_i)$ 未能完全拟合的部分，所以交给新的模型来完成。

我们知道gbdt的全称是Gradient Boosting Decision Tree，其中gradient被称为梯度，更一般的理解，可以认为是一阶导，那么这里的残差与梯度是什么关系呢。在第一部分，我们提到了一个叫做平方损失函数的东西，具体形式可以写成 $\frac{1}{2}\sum_0^n(y_i-F(x_i))^2$ ，熟悉其他算法的原理应该知道，这个**损失函数主要针对回归类型的问题，分类则是用熵值类的损失函数**。具体到平方损失函数的式子，你可能已经发现它的一阶导其实就是残差的形式，所以基于残差的gbdt是一种特殊的gbdt模型，它的损失函数是**平方损失函数，只能处理回归类的问题**。具体形式可以如下表示：

$$\text{损失函数：}\mathbf{L}(y,F(x))=\frac{1}{2}(y-F(X))^2$$

$$\text{最小化：}J=\frac{1}{2}\sum_{i=0}^{n}(y_i-F(x_i))^2$$

而损失函数的一阶导数为：

$$\frac{\partial J}{\partial F(x_i)}=\frac{\sum_i L(y_i,\mathbf{F}(x_i))}{\partial F(x_i)}=\frac{L(y_i,\mathbf{F}(x_i))}{\partial F(x_i)}=F(x_i)-y_i$$

正好残差等于负梯度：

$$y_i-F(x_i)=-\frac{\partial J}{\partial F(x_i)}$$

## 基于残差的 GBDT 缺点

首先基于残差的gbdt只能处理回归类的问题，不能处理分类问题，这是损失函数所限制的，所以更一般化的 GBDT 是基于梯度的算法，这也就意味着只要我给出的损失函数是可导的，那么我就能用gbdt的思想去解决问题。具体解决的问题就不会仅仅限于回归了。

另外，基于残差的gbdt在解决回归问题上也不算是一个好的选择，一个比较明显的缺点就是对异常值过于敏感。我们来看一个例子：

| $y_i$ | 0.5 | 1.2 | 2 | 5* |
|---|---|---|---|---|
| $F(x_i)$ | 0.6 | 1.4 | 1.5 | 1.7 |
| $L=(y-F)^2/2$ | 0.005 | 0.02 | 0.125 | 5.445 |

很明显后续的模型会对第4个值关注过多，这不是一种好的现象，所以一般回归类的损失函数会用绝对损失或者huber损失函数来代替平方损失函数：

$$\text{绝对损失}: L(y,F)=|y-F|$$

$$Huber\text{损失}: L(y,F)=\begin{cases}\frac{1}{2}(y-F)^2, & |y-F|\le\delta\\ \delta(|y-F|-\frac{\delta}{2}), & y-F|>\delta\end{cases}$$

| $y_i$ | 0.5 | 1.2 | 2 | 5* |
|---|---|---|---|---|
| $F(x_i)$ | 0.6 | 1.4 | 1.5 | 1.7 |
| Square loss | 0.005 | 0.02 | 0.125 | 5.445 |
| Absolute loss | 0.1 | 0.2 | 0.5 | 3.3 |
| Huber loss($\delta=0.5$) | 0.005 | 0.02 | 0.125 | 1.525 |

## Boosting 加法模型

如前面所述，gbdt模型可以认为是是由 K 个基模型(可以是决策树，也可以是回归模型)组成的一个加法运算式：

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

其中 F 是指所有基模型组成的`函数空间`。

那么一般化的损失函数是预测值 $\hat{y}$ 与 真实值y 之间的关系，如我们前面的**平方损失函数（均方误差）**，那么对于n个样本来说，则可以写成：

$$L=\sum_{i=1}^{n}l(y_i,\hat{y}_i)$$

更一般的，我们知道一个好的模型，在偏差和方差上有一个较好的平衡，而算法的损失函数正是代表了模型的偏差面，最小化损失函数，就相当于最小化模型的偏差，但同时我们也需要兼顾模型的方差，所以目标函数还包括抑制模型复杂度的正则项，因此目标函数可以写成：

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

其中 $\Omega$ 代表了基模型的复杂度，若基模型是树模型，则树的深度、叶子节点数等指标可以反应树的复杂程度。

对于Boosting来说，它采用的是前向优化算法，即从前往后，逐渐建立基模型来优化逼近目标函数，具体过程如下：

$$
\begin{aligned}
\hat{y}_i^0 &= 0 \\
\hat{y}_i^1 &= f_1(x_i) = \hat{y}_i^0 + f_1(x_i) \\
\hat{y}_i^2 &= f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i) \\
&\cdots \\
\hat{y}_i^t &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i)
\end{aligned}
$$

那么，在每一步，如何学习一个新的模型呢，答案的关键还是在于 GBDT 的目标函数上，即新模型的加入总是以优化目标函数为目的的。

我们以第t步的模型拟合为例，在这一步，模型对第 i 个样本 $x_i$ 的预测为：

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

其中 $f_t(x_i)$ 就是我们这次需要加入的新模型，即需要拟合的模型，此时，目标函数就可以写成：

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{i=i}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{t-1} + f_t(x_i)\right) + \Omega(f_t) + constant
\end{aligned}
\tag{2}
$$

即此时最优化目标函数，就相当于求得了 $f_t(x_i)$ 。

# GBDT 的目标函数

我们知道泰勒公式中，若 $\Delta x$ 很小时，我们只保留二阶导是合理的（GBDT 是一阶导，XGboost是二阶导，我们以二阶导为例，一阶导可以自己去推，因为更简单），即：

$$
f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2 \tag{3}
$$

那么在等式（2）中，我们把 $\hat{y}_i^{t-1}$ 看成是等式（3）中的x， $f_t(x_i)$ 看成是 $\Delta x$ ，即：

$$
f(\hat{y}_i^{t-1} + f_t(x_i)) \approx f(\hat{y}_i^{t-1}) + f'(\hat{y}_i^{t-1}) f_t(x_i) + \frac{1}{2} f''(\hat{y}_i^{t-1}) f_t(x_i)^2
$$

因此等式（2）可以写成：

$$Obj^{(t)} = \sum_{i=1}^{n}\left[l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) + constant \quad (4)$$

其中 $g_i$ 为损失函数的一阶导， $h_i$ 为损失函数的二阶导，注意这里的导是对 $\hat{y}_i^{t-1}$ 求导。我们以**平方损失函数**为例，则

$$\begin{aligned} l &= \sum_{i=1}^{n}\left(y_i - \left(\hat{y}_i^{t-1} + f_t(x_i)\right)\right)^2 \\ g_i &= \partial_{\hat{y}^{t-1}}\left(\hat{y}^{t-1} - y_i\right)^2 = 2(\hat{y}^{t-1} - y_i) \\ h_i &= \partial^2_{\hat{y}^{t-1}}\left(\hat{y}^{t-1} - y_i\right)^2 = 2 \end{aligned}$$

由于在第t步 $\hat{y}_i^{t-1}$ 其实是一个已知的值，所以 $l(y_i, \hat{y}_i^{t-1})$ 是一个常数，其对函数优化不会产生影响，因此，等式（4）可以写成：

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) \quad (5)$$

所以我们只要求出每一步损失函数的一阶和二阶导的值（由于前一步的 $\hat{y}^{t-1}$ 是已知的，所以这两个值就是常数）代入等式 5，然后最优化目标函数，就可以得到每一步的 f(x)，最后根据加法模型得到一个整体模型。

# GBDT 算法流程

【参考】

- 个站 - GBDT算法原理与系统设计简介
- stochastic gradient boosting

符号约定：

- $N$：表示样本的个数
- $M$：表示弱分类器的个数
- $L$：表示损失函数
- $f$：表示弱分类器
- $c$：表示步长

1.初始化 $f_0$，即初始化一个弱分类器，寻找一个使得损失函数最小化的常数值，此时树只有一个根节点：

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

2.**for m=1 to M:**

（a）对 $i = 1, 2, .., N$ 计算损失函数的负梯度值在当前模型的值，将其作为残差的估计，即：

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

对于平方损失函数，他就是通常说的残差；对于一般损失函数，他就是残差的近似值
（b）对 $r_{mi}$ 进行拟合一个回归树，得到第 m 棵树的叶节点区域 $R_{mj}, j = 1, 2, \ldots, J$

（c）对 $j = 1, 2, \ldots, J$计算

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

即利用线性搜索估计叶结点区域的值，使损失函数极小化
（d）更新回归树

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

3.得到最终的输出模型：

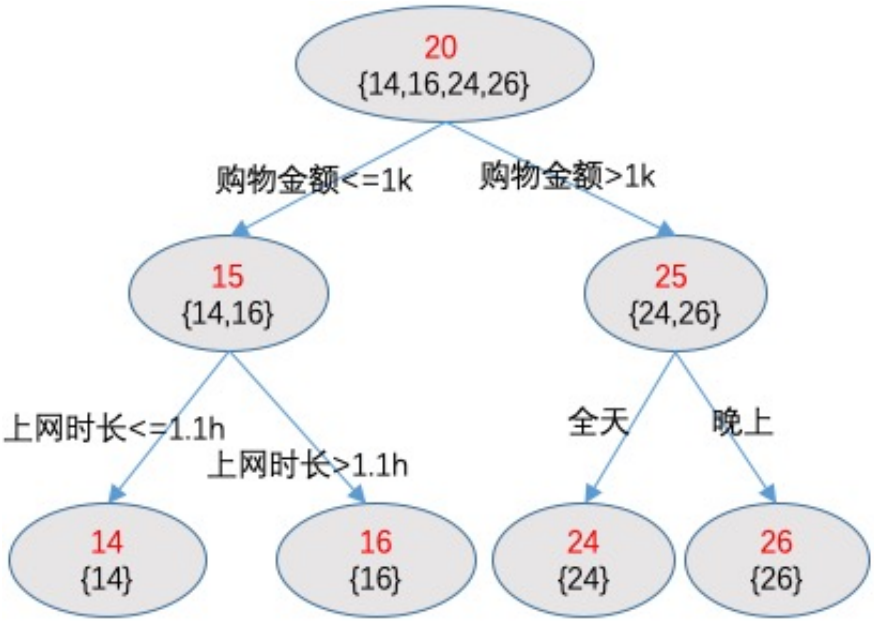
$$\hat{f}(\boldsymbol{x}) = f_M(\boldsymbol{x}) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} \mathbf{I}(\boldsymbol{x} \in R_{mj})$$

# GBDT 示例

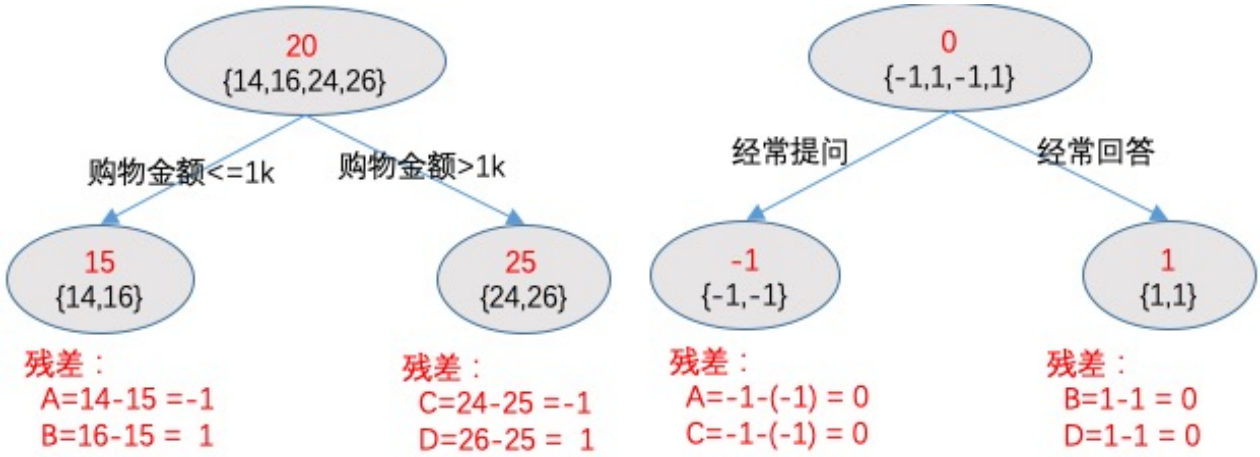
还是以预测年龄为例，数据集如下面所示：

| 编号 | 网购金额 | 上网 | 百度知道 | 教育 | 年龄 |
|---|---|---|---|---|---|
| A | 300 | 0.5h | 经常提问 | 高一 | 14 |
| B | 800 | 1.5h | 经常回答 | 高三 | 16 |
| C | 1200 | 全天上网 | 经常提问 | 应届毕业 | 24 |
| D | 3000 | 晚上上网 | 经常回答 | 工作两年 | 26 |

如果使用传统的训练方式，将得到如下的决策树：




使用GBDT来做这件事，由于数据太少，我们限定叶子节点最多有两个，即每棵树都只有一个分枝，并且限定只学两棵树。我们会得到如下图所示结果：




在第一棵树划分和传统训练方式一样，由于A,B年龄较为相近，C,D年龄较为相近，他们被分为两拨，每拨用平均年龄作为预测值。此时计算残差（残差的意思就是： A的预测值 + A的残差 = A的实际值），所以A的残差就是14-15=-1（需要注意的是，A的预测值是指前面所有树累加的和，这里前面只有一棵树所以直接是15，如果还有树则需要都累加起来作为A的预测值）。进而得到A,B,C,D的残差分别为-1,1，-1,1。然后我们拿残差替代A,B,C,D的原值，到第二棵树去学习，如果我们的预测值和它们的残差相等，则只需把第二棵树的结论累加到第一棵树上就能得到真实年龄了。第二棵树只有两个值1和-1，直接分成两个节点。此时所有人的残差都是0，即每个人都得到了真实的预测值。

那么我们将两棵树的预测值相加就可以得到其真实值。以 B 为例，在第一棵树预测值为 15，在第二棵树预测值为 1，因此 B 的实际值就是这两个值的相加，即 15 + 1 = 16。同理可以得到其他三个人的真是值： A: 15 – 1 = 14 C: 25 – 1 = 24 D: 25 + 1 = 26

那么哪里体现了Gradient呢？其实回到第一棵树结束时想一想，无论此时的**代价函数**是什么，是均方差还是均差，只要它以**误差**作为衡量标准，残差向量(-1, 1, -1, 1)都是它的全局最优方向，这就是Gradient。

对于 GBDT 的特征选择，一般可以通过配置文件设置。一般要么对所有特征拟合，要么随机sample一定比例的特征拟合；树的深度与停止条件相关参数有关，比如叶子的最大个数、树的最大深度、叶子节点最小实例数。如果完全不限制，树会分裂到不能分裂为止。树的个数是可配的，一般要通过validation集合确定，经验可以给与一定指导，例如shinkage * 树个数至少要>1，一般要>10才比较好.

# 正则化

- sklearn - ensemble Regularization
- Generalized Boosted Models: A guide to the gbm package

## shrinkage

Shrinkage（缩减）也可以叫做学习率，是一种正则化方法，他的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。

没用Shrinkage时：

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

其中：

$$F_{m-1}(x) = \sum_{i=1}^{m-1} \gamma_i h_i(x)$$

Shrinkage加入时：

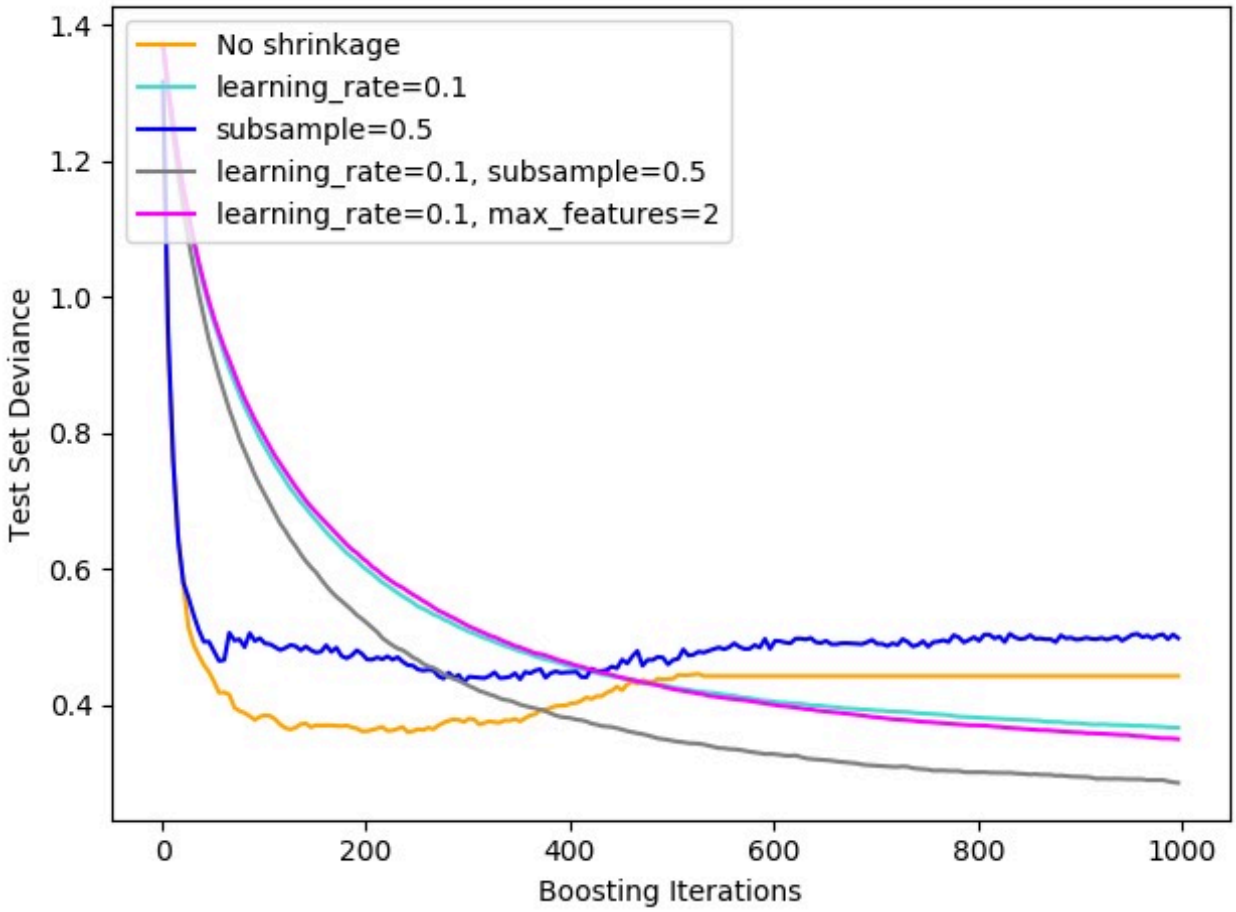
$$F_m(x) = F_{m-1}(x) + \nu\gamma_m h_m(x)$$

即Shrinkage仍然以残差作为学习目标，但对于残差学习出来的结果，只累加一小部分逐步逼近目标，step 一般都比较小，如0.01~0.001，导致各个树的残差是渐变的而不是陡变的。直觉上这也很好理解，不像直接用残差一步修复误差，而是只修复一点点，其实就是把大步切成了很多小步。**本质上，Shrinkage为每棵树设置了一个weight，累加时要乘以这个weight，但和Gradient并没有关系**。这个weight就是step。就像Adaboost一样，Shrinkage能减少过拟合发生也是经验证明的，目前还没有看到从理论的证明。

## subsampling

使用随机采样的 GBDT 叫做 stochastic gradient boosting，他结合了 Gradient Boosting 和 bootstrap averaging (bagging)， 该采样是无放回的采样。每次迭代训练分类器时，在训练数据集上采集一部分数据进行训练，采集的量通过`subsample`来控制，通常设置为 0.5 。

下图是shrinkage 与 subsampleing 在最优模型上的效果：可以看到没有 shrinkage 的 subsample 表现最差（蓝线）；shrinkage 与 subsample 结合表现的最好（灰色线）。




# GBDT 总结

## GBDT 优缺点

优点：

- 可以处理混合型数据类型
- 预测能力强
- 在输出空间对异常值鲁棒性很好

缺点：

- 扩展性不好，由于序列本性，导致其很难并行化处理

## GBDT 与 OOB

【参考】

- sklearn - Gradient Boosting Out-of-Bag estimates
- Random Forests OOB vs. Test Partition Performance
- Bias of the Random Forest Out-of-Bag (OOB) Error for Certain Input Parameters
- reddit - Full understanding of Out-Off-Bag-Error in Random Forests

OOB(out-of-bag) 可以用于评估 boosting 迭代的最优次数，OOB 与 交叉验证（cross-validation，CV）评估几乎一样，但 OOB 可以在运行时就可以计算，而不是像 CV 需要重复的进行模型拟合。但 OOB 只能适用于 Stochastic Gradient Boosting，即`subsample < 1.0`，他基于不在自助采样（boostrap sample）中的样本（即包外样本）来改善损失。

对于真实的损失来说，OOB 评估器是一个悲观的评估器，也就是说如果分类器真实的损失是 0.05，那么 OOB 评估器会悲观的认为其实分类器没有这么好，他会认为损失是 0.15。详细的对比数据可以查看Bias of the Random Forest Out-of-Bag (OOB) Error for Certain Input Parameters 给出的对比数据。

那么，为什么 OOB 评估器为什么会这么悲观内，这还要 OOB 自身说起。在 GBDT 的训练过程中，数据通常被分为训练集和测试集，在使用训练集训练时，可以采用 OOB 方法或者 CV 方法。OOB 通常是通过设置的采样率来决定从训练集中随机采样多少数据进行训练（与自助采样 bootstrap sample 不同），比如设置采样率为 0.5，即有一半数据用于训练，一半数据用于评估损失。这样从整体上来看，OOB 评估器在评估时只用了森林中的一小部分树，即只有一部分学习器参与了评估。

以 10 个样本的训练集为例，我们设置 10 个评估器，采样率为 0.5，如下图，其中 1 表示此次迭代时用于训练样本，0 表示用于评估的样本，X 表示样本，H 表示弱学习器：

| | H0 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 |
|---|---|---|---|---|---|---|---|---|---|---|
| X0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| X1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| X2 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| X3 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| X4 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| X5 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| X6 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| X7 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| X8 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| X9 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

从上图可以看到，总体上来看，在进行评估时并没有使用到全部的若分类器。以样本 X0 为例，在用于评估时只有四个分类器（H1、H3、H6、H8）对他进行了评估。显然只是用一部分分类器进行评估是不准确的，要比真是的损失要大很多，因此说 OOB 评估器是悲观的。可以看到使用最多评估器的是样本 X7，只有两个分类器没有对他评估，对于 X7 的损失就相对准确，但也只是相对准确，仍然没有使用到全部的分类器进行评估。

而使用 CV 方法就会使用到全部分的分类器，得到的结果就比较准确，可以知道分类器的实际表现。如下图：7 个样本进行训练，3 个用于评估：

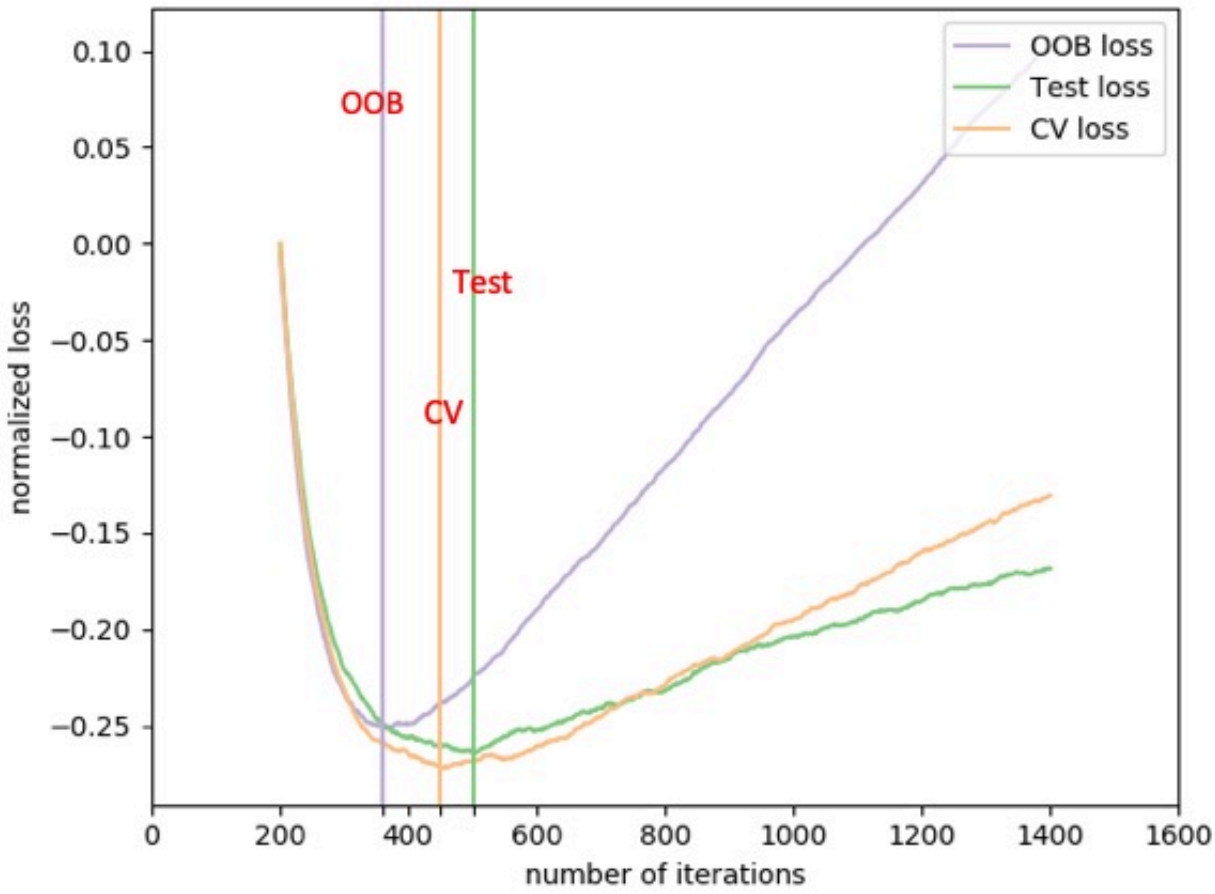
训练

| | H0 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 |
|---|---|---|---|---|---|---|---|---|---|---|
| X0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

验证

| | H0 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 |
|---|---|---|---|---|---|---|---|---|---|---|
| X7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| X8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| X9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

OOB、CV 与 Test 损失评估对比图：




可以看到 OOB 评估方法收敛的是最快的，但随后就朝着更悲观的方向。也可以看到使用 CV 评估方法会得到更好的结果，但需要的计算量会更大，这是因为在每一次使用验证集评估时都需要训练一次数据，而 OOB 在训练时就可以进行评估。

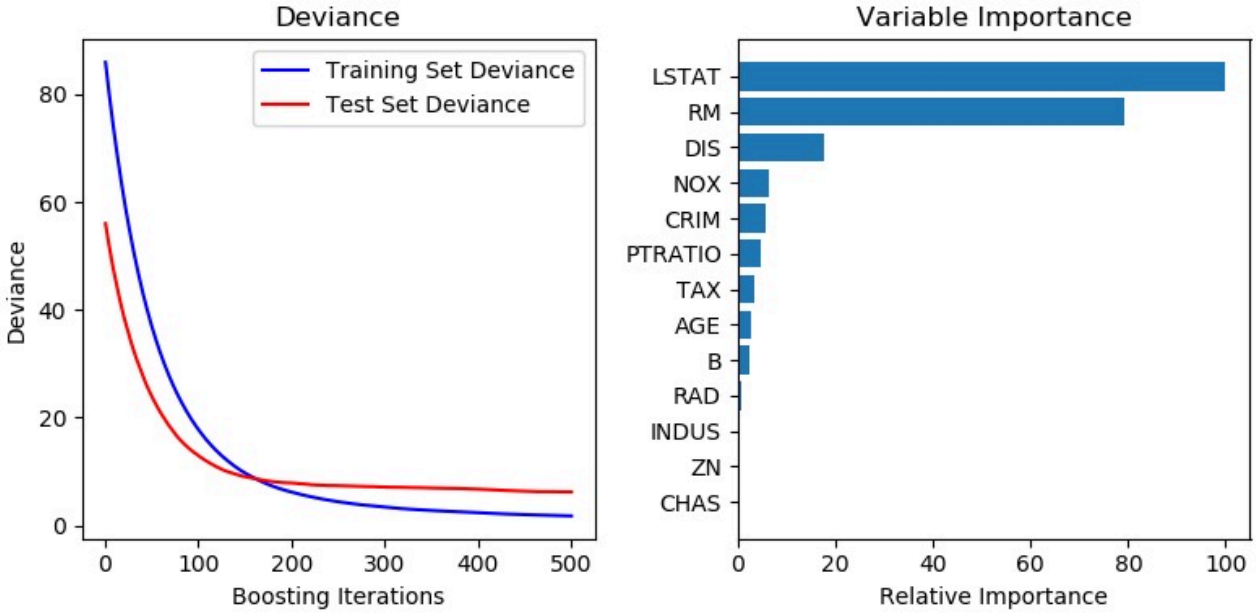
因此推荐使用 CV 的方法，只有在 CV 计算量特别大的时候使用 OOB。

# 可解释性

## 评估特征的重要性

- sklearn - Feature importance

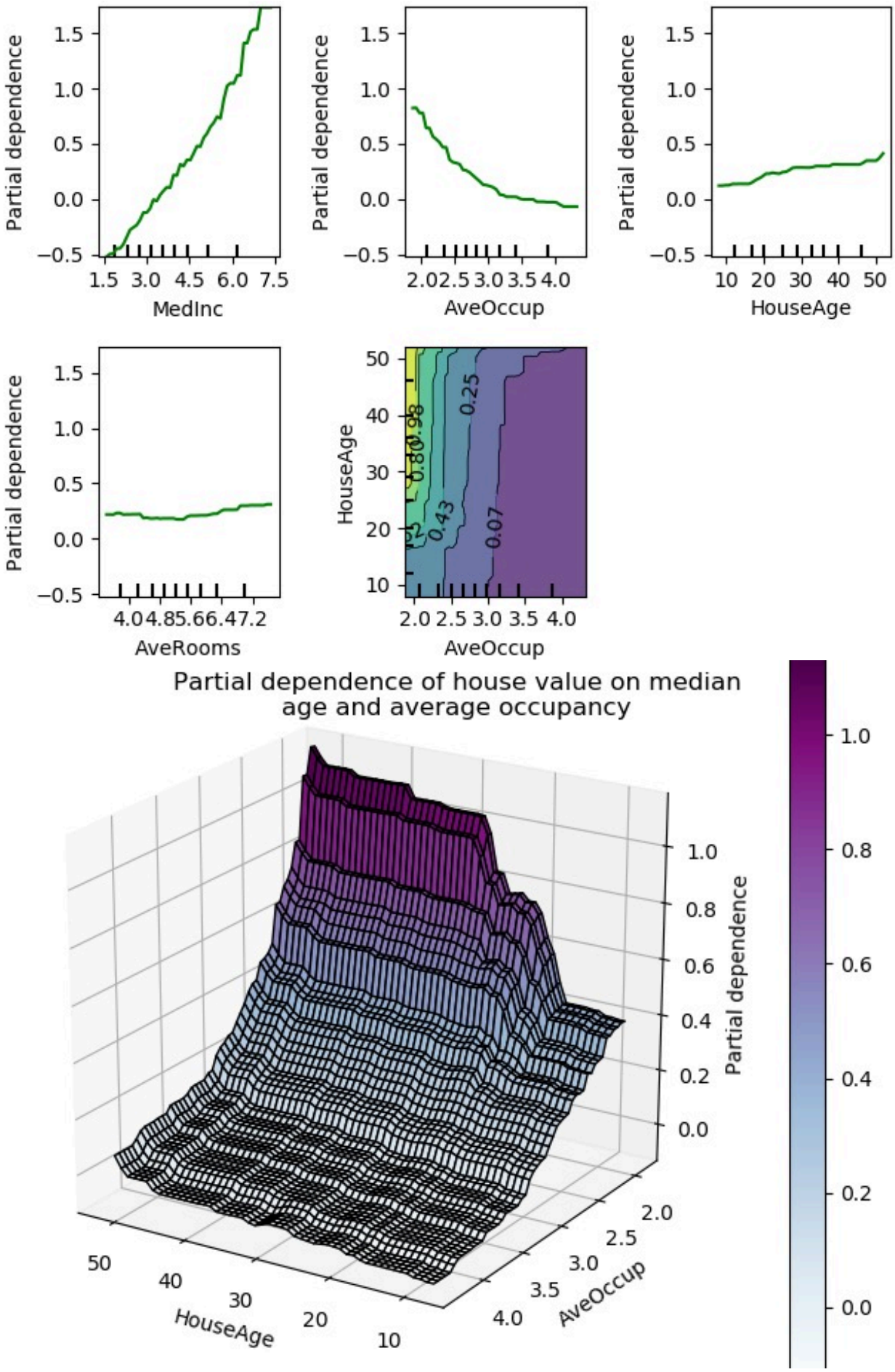
特征的重要习是通过判断特征在树分裂节点时使用的频率判断，使用的越频繁则特征越重要：




# PDP分析

【参考】

- 知乎 - 随机森林的直观理解
- kaggle - Partial Dependence Plots
- arxiv - Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation
- Introducing PDPbox


Partial dependence
MedInc
AveOccup
HouseAge
AveRooms
HouseAge
AveOccup
Partial dependence of house value on median age and average occupancy
HouseAge
AveOccup
Partial dependence

## 常见问题

# 为什么要使用 GBDT

既然使用的前后效果是一样的，为什么还需要使用 GBDT 呢？答案是为了防止过拟合。

我们发现原始的决策树为了达到100%精度使用了3个feature（上网时长、时段、网购金额），其中分枝“上网时长>1.1h” 很显然已经过拟合了，这个数据集上A,B也许恰好A每天上网1.09h, B上网1.05小时，但用上网时间是不是>1.1小时来判断所有人的年龄很显然是有悖常识的；

相对来说 GBDT 虽然用了两棵树，但其实只用了2个特征就搞定了，后一个特征是问答比例，显然 GBDT 的依据更靠谱。Boosting的最大好处在于，每一步的残差计算其实变相地增大了分错样本的权重，而已经分对的样本则都趋向于0。这样后面的树就能越来越专注那些前面被分错的样本。就像我们做互联网，总是先解决60%用户的需求凑合着，再解决35%用户的需求，最后才关注那5%人的需求，这样就能逐渐把产品做好，因为不同类型用户需求可能完全不同，需要分别独立分析。如果反过来做，或者刚上来就一定要做到尽善尽美，往往最终会竹篮打水一场空。

# GBDT 的 G 在哪

到目前为止，我们的确没有用到求导的Gradient。在当前版本GBDT描述中，的确没有用到Gradient，该版本用残差作为全局最优的绝对方向，并不需要Gradient求解.

# Adaboost不是这么定义的

这是 boosting，但不是 `Adaboost`。`GBDT` 不是 `Adaboost Decistion Tree`。就像提到决策树大家会想起 `C4.5`，提到 boost 多数人也会想到 `Adaboost`。`Adaboost`是另一种 `boost` 方法，它按分类对错，分配不同的weight，计算代价函数时使用这些weight，从而让“错分的样本权重越来越大，使它们更被重视”。Bootstrap 也有类似思想，它在每一步迭代时不改变模型本身，也不计算残差，而是从N个样本训练集中按一定概率重新抽取N个样本出来（单个样本可以被重复采样），对着这N个新的样本再训练一轮。由于数据集变了迭代模型训练结果也不一样，而一个样本被前面分错的越厉害，它的概率就被设的越高，这样就能同样达到逐步关注被分错的样本，逐步完善的效果。

Adaboost 的方法被实践证明是一种很好的防止过拟合的方法，但至于为什么则至今没从理论上被证明。GBDT也可以在使用残差的同时引入Bootstrap re-sampling，GBDT多数实现版本中也增加的这个选项，但是否一定使用则有不同看法。re-sampling 一个缺点是它的随机性，即同样的数据集合训练两遍结果是不一样的，也就是模型不可稳定复现，这对评估是很大挑战，比如很难说一个模型变好是因为你选用了更好的特征，还是由于这次采样的随机因素。