决策树算法

陈 鑫

2023 年 4 月 30 日

目录

目录

决策是进行各项活动时普遍存在的一种择优手段，决策树是一种树形结构，其中每个内部节点表示一个属性（特征）上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。它是统计，数据挖掘和机器学习中使用的预测建模方法之一。目标变量采用一组离散值的树模型称为分类树；在这些树结构中，叶子代表类标签，分支代表划分标签的依据。目标变量采用连续值（通常是实数）的决策树称为回归树。

当标签和特征之间呈现线性关系时，使用线性回归算法来拟合标签与特征的关系是最合适的算法，即使有时标签与特征之间的关系是非线性的，也可以对数据进行特征变换后使用线性回归算法求解，例如多项式回归。但是，当标签分布是非连续型分布时，线性回归就不合适了。例如导言中的贷款问题和下面的这个例子。

案例 8.1 样本空间 $X = [-1, 1] \in \mathbb{R}$，对均匀分布产生的任一样本特征 x，相应的标签 y 服从正态分布 $D_x = N(f(x), 0.05)$。其中，正态（高斯）分布的期望是以下的分段函数：

$$f(x) = \begin{cases} -1, & x < -0.5 \\ 1, & -0.5 \leq x < 0 \\ -0.5, & 0 \leq x < 0.5 \\ 0.5, & x \geq 0.5 \end{cases}$$

对上述的分布进行线性回归，观察其效果。

参考 linear_regression_bad.py

决策树是一个类似于人们决策过程的树结构，从根节点开始，每个分枝代表一个新的决策事件，会生成两个或多个分枝，每个叶子代表一个最终判定所属的类别。

例如，如下是一个决策树，代表薪水大于 30W 的男性会买车。

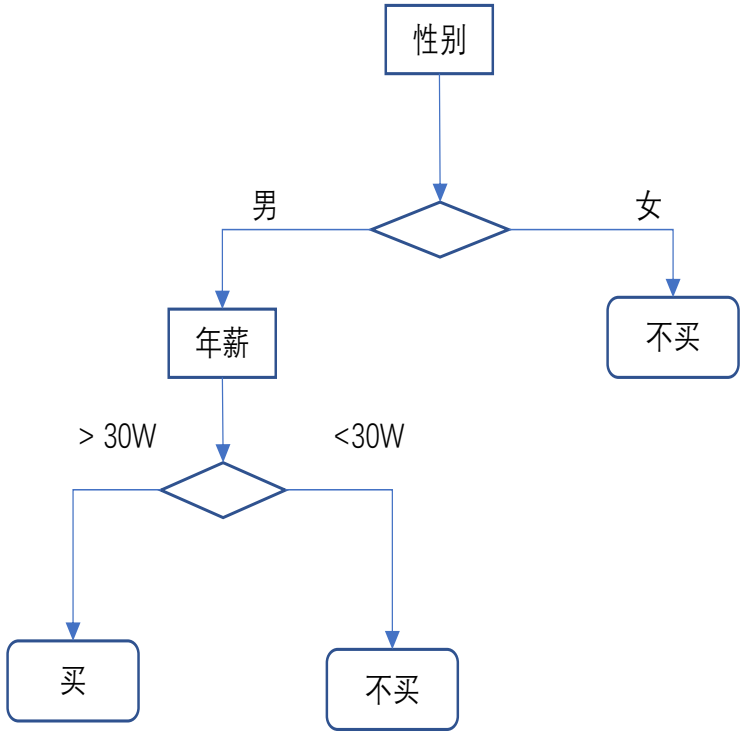


图 1: 决策树的决策过程

决策树模型按照样本特征将样本空间划分成一些局部区域，并对每一个区域制定一个统一的标签预测。任给一个数据样本，决策树将根据样本的特征判断它所属的区域，并将该区域的指定标签预测值作为对这个数据样本的标签预测。

Algorithm 1: 决策树模型

决策树节点数据结构 node:
j: 特征下标 θ: 阈值 p：标签预测值 left：左孩子节点 right：右孩子节点
决策树模型的递归算法

```
def T(node, x):
    if node.left == NULL and node.right == NULL: then
        return node.p
    else
        j = node.j
        if x_j ≤ node.θ then
            return T(node.left, x)
        else
            return T(node.right, x)
```

决策树模型 h($\boldsymbol{x}$) = T(root,$\boldsymbol{x}$)

目录

决策树回归算法

因为决策树算法无法像线性回归一样采取正则化，为了防止出现过拟合，必须对决策树模型的深度做出限制。

Algorithm 2: 限制深度的决策树回归算法

决策树回归算法（带深度限制）

Input: m 条训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$

模型假设：$H^d = \{h(\boldsymbol{x}) | h \text{ 为深度不超过 } d \text{ 的决策树}\}$

$\min_{h \in H^d} \frac{1}{m} \sum_{i=1}^{m} (h(\boldsymbol{x}^{(i)}) - y^{(i)})^2$

案例 8.2 假设样本空间 $X = [0, 3\pi)$。对于任意特征 $x \in X$，标签 y 服从正态分布 $D_x = N(cos\ (x), 0.1)$，随机生成 50 个样本，对其使用 sklearn 提供的决策树模型进行拟合，得到如图2所示的图像，原始数据是均值为余弦函数，标准差为 0.1 的正态分布。参考 cos_decision_tree_sklearn.py

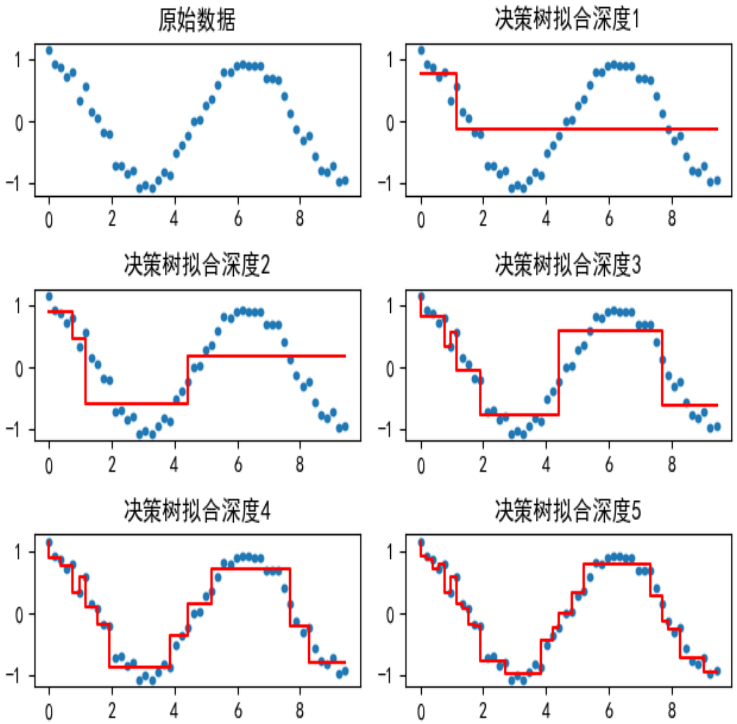


图 2: 原始数据及决策树模型拟合

目录

决策树模型除了可以应用于回归问题，还可以应用于分类问题。当正采样或负采样的分布不是一个整体的连续区域，而是由多个局部区域构成时，决策树模型比第 5 章的 Logistic 回归更适合，只需将算法1中的概率值 p 设为概率向量即可。在一个分类问题中，决策树分类算法是一个以决策树模型为模型假设，以交叉熵为目标函数的经验损失最小化算法。该算法采用 $0-1$ 向量的标签形式，决策树模型输出一个 k 维概率向量。向量的第 i 个分量表示样本属于类别 i 的概率。

Algorithm 3: 限制深度的决策树分类算法

决策树分类算法（带深度限制）

Input: m 条训练数据 $S=\{(\boldsymbol{x}^{(1)},y^{(1)}),(\boldsymbol{x}^{(2)},y^{(2)}),\cdots,(\boldsymbol{x}^{(m)},y^{(m)})\}$

模型假设：$H^d=\{h(\boldsymbol{x})|h \text{ 为深度不超过 } d \text{ 的决策树}\}$

$\min_{h\in H^d}-\frac{1}{m}\sum_{i=1}^{m}y^{(i)T}\log h(\boldsymbol{x}^{(i)})$

针对类别预测任务形式，决策树先完成概率预测任务，然后通过最大概率分类函数做出对类别的预测，当训练数据的分类边界是非线性的，且能够用矩形来划分特征空间时，决策树是求解分类问题的一个合适的算法。

案例实战 8.3 由月亮形函数 make_moon 和环形函数 make_circles 生成样本数据，并用决策树分类算法，对这两种数据分类。moon_circles_decision_tree_classifier.py

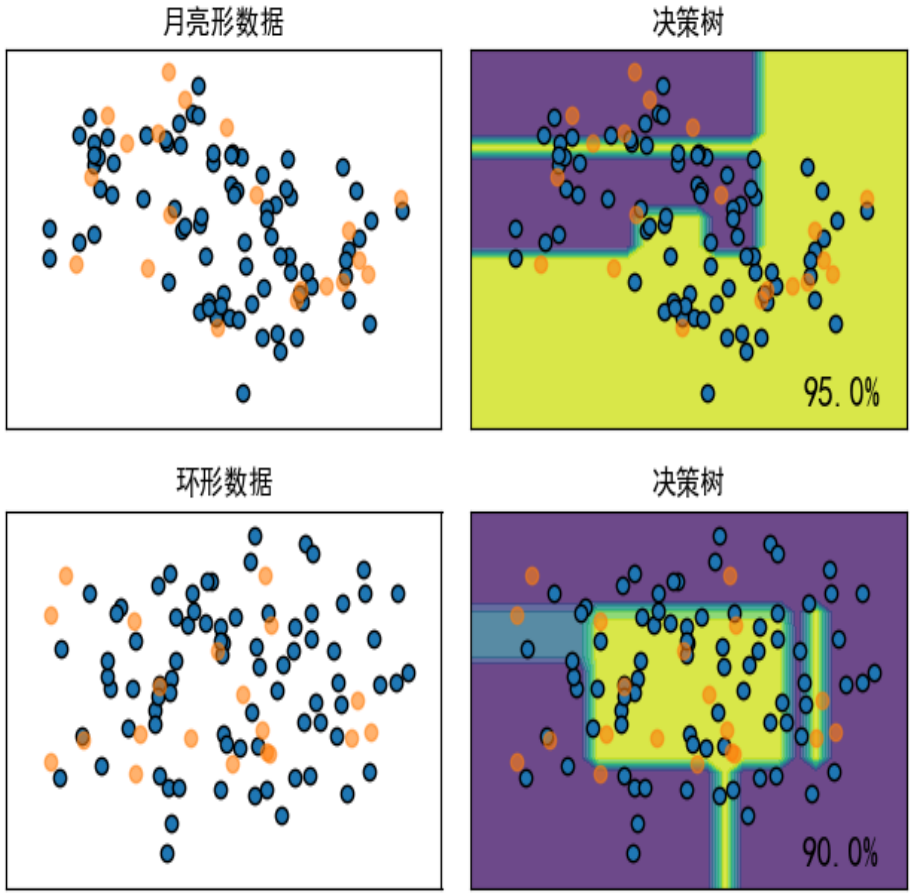


图 3: 原始数据及决策树模型拟合

目录

决策树优化算法

常见的决策树优化算法：

1. ID3（Iterative Dichotomiser 3）由 Ross Quinlan 在 1986 年提出。该算法创建一棵二叉树。
2. C4.5 算法对 ID3 算法进行了改进，在生成决策树的过程中，用信息增益比来选择特征。
3. C5.0 是 Quinlan 根据专有许可证发布的最新版本。它使用更少的内存，并建立比 C4.5 更小的规则集，同时更准确。
4. CART（Classification and Regression Trees 分类和回归树）与 C4.5 非常相似，但它不同之处在于它支持数值目标变量（回归），并且不计算规则集。CART 使用在每个节点产生最大信息增益的特征和阈值来构造二叉树。

sklearn 使用 CART 算法的优化版本。决策树算法不是一个凸优化算法，目前没有任何已知的高效算法能够精确地计算出最优决策树，只能寻求在有限资源下的高效近似算法。CART 就是这样一种近似算法。

当决策树深度为 $d=0$ 时，考虑模型假设 H^0 的均方误差优化问题，这时决策树只有一个节点，它既是根节点又是叶子结点。其目标函数为最小均方误差 $\min_{h\in H^0}\frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x}^{(i)})-y^{(i)})^2$，其中 $h(\boldsymbol{x}^{(i)})=p$，通过简单的求驻点和极值，可以求得 $p=\frac{1}{m}\sum_{i=1}^{m}y^{(i)}$。也就是当模型假设为 H^0 时，算法2的优化问题的最优解是标签的均值 p，记为 $Avg(S)$，其标签的方差记为 $Var(S)=\frac{1}{m}\sum_{i=1}^{m}(Avg(S)-y)^2$ 。

当决策树深度为 $d=1$ 时，考虑模型假设 H^1 的优化问题，这时决策树包含一个根节点和两个叶子节点。由算法1的决策树模型的数据结构，每一棵这样的决策树由 4 个参数决定：根节点中的特征 j，阈值 θ，两个叶子节点的预测值 p_L 和 p_R。

用 $mse(j,\theta,p_L,p_R)=\frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x}^{(i)})-y^{(i)})^2$ 表示模型在 S 上的均方误差，则算法2的优化问题的目标函数为 $h=\min_{j,\theta,p_L,p_R}mse(j,\theta,p_L,p_R)$。对给定的特征 j 和阈值 θ，定义 $mse(j,\theta)=\min_{p_L,p_R}mse(j,\theta,p_L,p_R)$，则有 $h=\min_{j,\theta}mse(j,\theta)$。

取定特征 j 和阈值 θ，可将数据集 S 分成两部分 $S_L = \{(\boldsymbol{x}, y) \in S | x_j \leq \theta\}$ 和 $S_R = \{(\boldsymbol{x}, y) \in S | x_j > \theta\}$，因此可以将 h 分为两部分：$h = \frac{1}{m}(\sum_{(\boldsymbol{x},y)\in S_L}(p_L - y)^2 + \sum_{(\boldsymbol{x},y)\in S_R}(p_R - y)^2)$，类似 $d = 0$ 的推导，可以求得

$$p_L = \frac{1}{|S_L|} \sum_{(\boldsymbol{x},y)\in S_L} y \tag{1}$$

$$p_R = \frac{1}{|S_R|} \sum_{(\boldsymbol{x},y)\in S_R} y \tag{2}$$

$$mse(j, \theta) = \frac{|S_L|}{m} Var(S_L) + \frac{|S_R|}{m} Var(S_R) \tag{3}$$

因此，遍历所有可能的特征 j 和阈值 θ，并对每一组 j 和 θ，用（3）式计算 $mse(j, \theta)$。以使得 $mse(j, \theta)$ 最小的一组 j 和 θ 作为参数生成根节点，以式（1）和（2）计算出来的 p_L 和 p_R 作为两个叶子节点的预测值。

CART 算法是上述两个模型假设下的决策树算法在深度 $d > 1$ 时的推广。

Algorithm 4: 决策树回归问题的 CART 算法

Input: m 条训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$
Output: *root*
def *generate_tree(S, d)*:

```
root = new node
if d == 0 or |S| < 2 then
    root.p = Avg(S)
else
    for j = 1, 2, ··· , n do
        for θ ∈ {x_j | x ∈ S} do
            S_L(j, θ) = {(x, y) ∈ S | x_j ≤ θ}
            S_R(j, θ) = {(x, y) ∈ S | x_j > θ}
            mse(j, θ) = (|S_L|/m) Var(S_L(j, θ)) + (|S_R|/m) Var(S_R(j, θ))
    j*, θ* = arg min_{j,θ} mse(j, θ)
    root.j = j*
    root.θ = θ*
    root.left = generate_tree(S_L(j*, θ*), d − 1)
    root.right = generate_tree(S_R(j*, θ*), d − 1)
return root
```

算法4在每一次划分数据时，都只考虑将当前的数据分为两部分的局部最优分法，而并不考虑当前虽然是次优的，但在未来的递归划分中可能是全局最优的划分，因此，CART 是一个贪心算法。

决策树分类问题的 CART 算法

CART 算法同样可以应用到决策树分类问题中，只是进行数据划分的准则不同。给定训练数据 $S = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$，其中 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^T$，$\boldsymbol{y} \in \{0, 1\}^k$ 是一个 k 维的 $0-1$ 向量。定义 $Avg(S) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{y}^{(i)}$，即标签的平均值。这里，$Avg(S)$ 是一个 k 维向量，每一个分量 $Avg(S)_j = \frac{1}{m}\sum_{i=1}^{m} y_j^{(i)}, j = 1, 2, \cdots, k$，其取值都是在 $[0, 1]$ 之间，表示 S 中第 j 类出现的频率，即 $Avg(S) = (Avg(S)_1, Avg(S)_2, \cdots, Avg(S)_k)$。定义 S 的熵为 $Entropy(S) = -Avg(S)^T \log Avg(S)$。

当决策树深度为 $d = 0$ 时，考虑模型假设 H^0 的分类优化问题，设决策树的模型预测值为 $\boldsymbol{p} \in [0, 1]^k$。

则目标函数是交叉熵

$$Ce(p) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \log p_j \tag{4}$$

$$= -\sum_{j=1}^{k}(\log p_j)\frac{\sum_{i=1}^{m} y_j^{(i)}}{m} \tag{5}$$

$$= -\sum_{j=1}^{k}(\log p_j) Avg(S)_j \tag{6}$$

求解上述交叉熵的最优解，得到 $p_j = Avg(S)_j$，即 $\boldsymbol{p} = Avg(S)$ 是 $Ce(p)$ 达到最小值的概率预测向量。将其重新代回 $Ce(p)$，正好等于 $Entropy(S)$，即 $Ce(Avg(S)) = Entropy(S)$。因此，当深度为 0 的时候，算法3中算法优化问题的最优解就是交叉熵 $Entropy(S)$。

当决策树深度为 $d=1$ 时，考虑模型假设 H^1 的分类优化问题。深度为 1 的决策树包含一个根节点和两个叶子节点。由算法1的决策树模型的数据结构，每一棵这样的决策树由 4 个参数决定：根节点中的特征 j，阈值 θ，两个叶子节点的预测值 $\boldsymbol{p}_L$ 和 $\boldsymbol{p}_R$，其中 $\boldsymbol{p}_L$ 和 $\boldsymbol{p}_R$ 都是 k 维向量。

用 $Ce(j,\theta,\boldsymbol{p}_L,\boldsymbol{p}_R)=-\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{y}^{(i)T}\log h(\boldsymbol{x}^{(i)})$ 表示模型在 S 上的交叉熵，则算法3的优化问题的目标函数为 $h=\min_{j,\theta,\boldsymbol{p}_L,\boldsymbol{p}_R} Ce(j,\theta,\boldsymbol{p}_L,\boldsymbol{p}_R)$。对给定的特征 j 和阈值 θ，定义 $Ce(j,\theta)=\min_{\boldsymbol{p}_L,\boldsymbol{p}_R} Ce(j,\theta,\boldsymbol{p}_L,\boldsymbol{p}_R)$，则有 $h=\min_{j,\theta} Ce(j,\theta)$。取定特征 j 和阈值 θ，可将数据集 S 分成两部分 $S_L=\{(\boldsymbol{x},\boldsymbol{y})\in S|x_j\le\theta\}$ 和 $S_R=\{(\boldsymbol{x},\boldsymbol{y})\in S|x_j>\theta\}$，因此可以将 h 分为两部分：$h=-\frac{1}{m}(\sum_{(\boldsymbol{x},\boldsymbol{y})\in S_L}\boldsymbol{y}^T\log\boldsymbol{p}_L+\sum_{(\boldsymbol{x},\boldsymbol{y})\in S_R}\boldsymbol{y}^T\log\boldsymbol{p}_R)$，类似 $d=0$ 的推导，可以求得

$$\boldsymbol{p}_L=Avg(S_L) \tag{7}$$

$$\boldsymbol{p}_R=Avg(S_R) \tag{8}$$

$$Ce(j,\theta)=\frac{1}{m}|S_L|Entropy(S_L)+\frac{1}{m}|S_R|Entropy(S_R) \tag{9}$$

遍历所有的特征 j 和阈值 θ，计算出式（9）最小的 $Ce(j,\theta)$，取出使得 $Ce(j,\theta)$ 最小的 j 和 θ 生成根节点，再利用（7）和（8）计算出 p_L 和 p_R 作为两个叶子节点的预测值。从而生成了一个深度为 1 的决策树。

Algorithm 5: 决策树分类问题的 CART 算法

Input: m 条训练数据 $S = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
Output: *root*
def *generate_tree(S, d)*:
    $root = new\ node$
    if $d == 0$ *or* $|S| < 2$ **then**
        $root.p = Avg(S)$
    else
        for $j = 1, 2, \cdots, n$ **do**
            for $\theta \in \{x_j | \boldsymbol{x} \in S\}$ **do**
                $S_L(j,\theta) = \{(\boldsymbol{x}, \boldsymbol{y}) \in S | x_j \leq \theta\}$
                $S_R(j,\theta) = \{(\boldsymbol{x}, \boldsymbol{y}) \in S | x_j > \theta\}$
                $Ce(j,\theta) = \frac{|S_L|}{m} Entropy(S_L) + \frac{|S_R|}{m} Entropy(S_R)$
        $j^*, \theta^* = \arg\min_{j,\theta} Ce(j,\theta)$
        $root.j = j^*$
        $root.\theta = \theta^*$
        $root.left = generate\_tree(S_L(j^*, \theta^*), d-1)$
        $root.right = generate\_tree(S_R(j^*, \theta^*), d-1)$
    return *root*

目录

决策树的回归算法和分类算法的描述，可以看到，它们的结构完全相同，只是在数据划分准则上有所区别，因此可以设计一个划分准则函数，使其在决策树回归问题中表示方差，在分类问题中表示熵，这样两个算法的实现方式就完全相同。基于此，可以设计一个统一的 CART 算法的基类，然后再由基类的构造函数传入指定的划分准则函数，就可以实现两种不同算法，达到代码复用的目的。decision_tree_base.py

回归问题

有了 CART 基类之后，对于回归问题，只需要编写一个派生类继承自 CART 基类，并实现方差函数，将其传递给基类的构造函数即可。参考 decision_tree_regressor.py

案例 8.3 对案例 8.1 的数据进行决策树回归。创建 decision_tree_depth3_vs_decision_treedepth5.py。

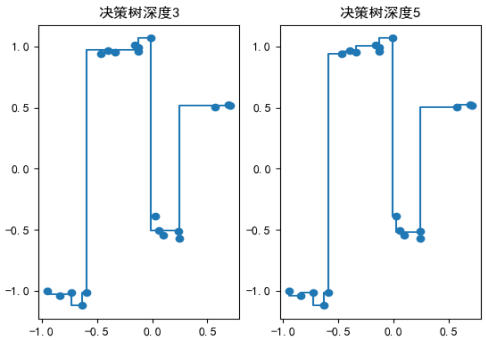


图 4: 原始数据及决策树模型拟合

案例 8.4 对案例 8.2 的余弦数据集进行决策树回归拟合。cos_decision_tree_sklearn.py 参考图2

分类问题

对于分类问题，只需要编写一个派生类继承自 CART 基类，并实现熵函数，然后将其传递给基类的构造函数，就可以得到决策树分类算法的实现。参考 decision_tree_classfier.py

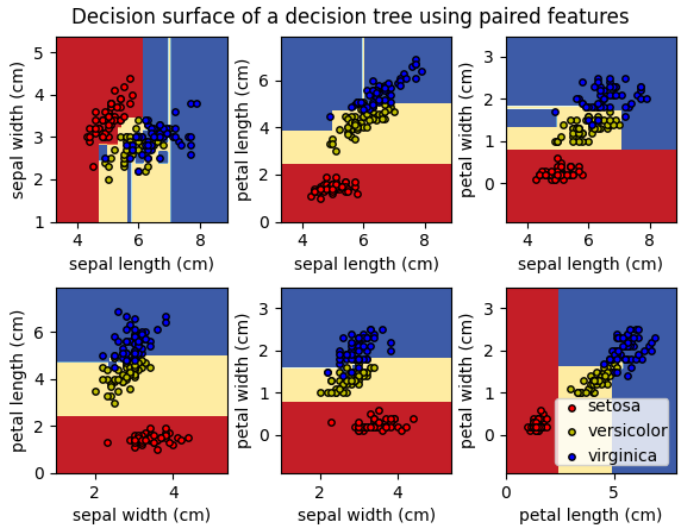
案例实战 8.5 使用决策树回归算法对加州房价进行评估。california_housing_decision_tree_regressor.py

案例实战 8.6 使用决策树分类算法对月亮形数据和环形数据进行分类。moon_circles_decison_tree_classifier.py 参考图3

案例实战 8.7 使用 sklearn 提供的决策树回归算法对加州房价进行评估。california_housing_decision_tree_regressor_sklearn.py

案例实战 8.8 使用 Sklearn 决策树分类算法对月亮形数据和环形数据进行分类。moon_circles_decision_tree_classifier_sklearn.py

案例 8.9 用 sklearn 决策树分类器对鸢尾花进行分类。
iris_decision_tree_classifier_sklearn.py



(a) 鸢尾花的决策边界

(b) 决策树的可视化划分标准

图 5: 用 sklearn 决策树分类器对鸢尾花进行分类

目录

集成学习（Ensemble Learning）是非常流行的机器学习算法，它本身不是一个单独的机器学习算法，而是在数据上构建多个模型，集成所有模型的建模结果。集成算法会考虑多个评估器的建模结果，汇总之后得到一个综合的结果，以此来获取比单个模型更好的回归或分类表现。多个模型集成而成的模型叫做集成评估器（Ensemble Estimator），组成集成评估器的每个模型都叫做基评估器（Base Estimator）。

集成方法通常分为两种：一种是平均方法，该方法的原理是构建多个独立的评估器，然后取它们的预测结果的平均。一般来说组合之后的评估器比单个评估器要好的，因为它的方差减小了。装袋法的核心思想是构建多个相互独立的评估器，然后对其预测进行平均或多数表决原则来决定集成评估器的结果。装袋法的代表模型就是随机森林。另一种是提升法，其基评估器是相关的，基评估器按顺序依次构建的，并且每一个基估计器都尝试去减少组合评估器的偏差。其核心思想是结合弱评估器的力量对难以评估的样本进行预测，逐渐构成一个强评估器。提升法的代表模型有 AdaBoost 和梯度提升树。

在集成算法中，装袋方法在原始训练集的随机子集上构建一类黑盒评估器的多个实例，然后把这些评估器的预测结果结合起来形成最终的预测结果。该方法通过在构建模型的过程中引入随机性，来减少基评估器的方差。在多数情况下，装袋方法提供了一种非常简单的方式来对单一模型进行改进，而无需修改背后的算法。因为装袋方法可以减小过拟合，所以通常在强分类器和复杂模型上使用时表现很好（例如完全生长的决策树），相比之下，提升方法则在弱模型上表现更好（例如浅层决策树）。

案例实战 8.10 比较决策树回归与装袋回归。dtr_vs_br_sklearn.py

随机森林算法

随机森林是非常具有代表性的装袋集成算法，它的所有基评估器都是决策树，分类树组成的森林称为随机森林分类器，回归树所集成的森林称为随机森林回归器。单个决策树的准确率越高，随机森林的准确率也会越高，因为装袋法是依赖于平均值或者少数服从多数原则来决定集成的结果的。

案例实战 8.11 使用红酒数据集对比决策树与随机森林的分类效果。

wine_decision_tree_vs_random_forest_sklearn.py

AdaBoost 提升

AdaBoost，是英文”Adaptive Boosting”（自适应提升）的缩写，是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。其算法本身是通过改变数据分布来实现的，它根据每次训练集中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值。将修改过权值的新数据集送给下层分类器进行训练，最后将每次训练得到的分类器最后融合起来，作为最后的决策分类器。Adaboost 的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。参考实例：
https://blog.csdn.net/cxx654/article/details/106299351

Adaboost 的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。具体说来，整个 Adaboost 迭代算法就 3 步：

1. 初始化训练数据的权值分布。如果有 N 个样本，则每一个训练样本最开始时都被赋予相同的权重：1/N。
2. 训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权重就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。然后，权重更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。
3. 将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

案例实战 8.12 Adaboost 提升决策树分类。

使用高斯分位数生成分类数据集，并绘制决策边界和决策分数。这个例子里面拟合了一个 AdaBoosted 的决策树，使用的数据集是由两个 “高斯分位数”(参看 sklearn.datasets.make_gaussian_quantiles) 组成的非线性可分的分类数据集，并绘制决策边界和决策分数。对于 A 类和 B 类样本，分别给出了决策分数的分布。每个样本的预测类别标签是由决策分数的符号决定的。决策分数大于 0 的样本被分类为 B，否则被分类为 A。决策评分的大小决定了与预测的类标签相似的程度。

two_adaboosted_decision_tree_sklearn.py

案例实战 8.13 AdaBoost 决策树回归。

使用 AdaBoost 提升决策树。首先随机生成带有高斯噪声的一维正弦数据集，在该数据集上使用 AdaBoost 提升决策树算法拟合正弦函数曲线。该实例对比了由一棵决策树（即一个弱回归学习器、评估器）和相当于总共有 10 棵决策树、10 个弱回归学习器构成的强评估器的拟合效果，然后继续加大弱回归学习器的数量，将其增加至 300 个弱回归学习器，再对比其拟合效果，结果如下图所示，可以看出，随着提升次数的增加，回归器可以拟合出更多的细节。sin_sin6_adaboost_decision_tree_regressor_sklearn.py

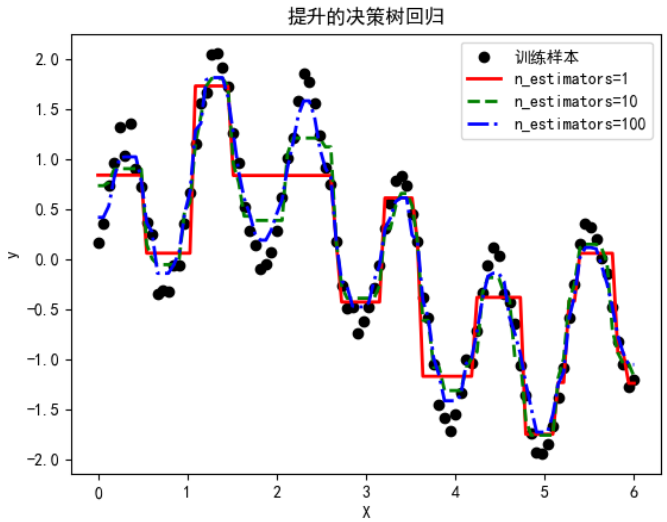


图 6: 决策树及其 adaboost 提升对比

梯度提升决策树

梯度提升回归树（GBRT）是对于任意的可微损失函数的提升算法的泛化。类似于 Adaboost，我们把重要参数分为两类，第一类是 Boosting 框架的重要参数，第二类是弱学习器即 CART 回归树的重要参数。

案例实战 8.14 手写数字分类。digits_gradient_boost_classifier_sklearn.py

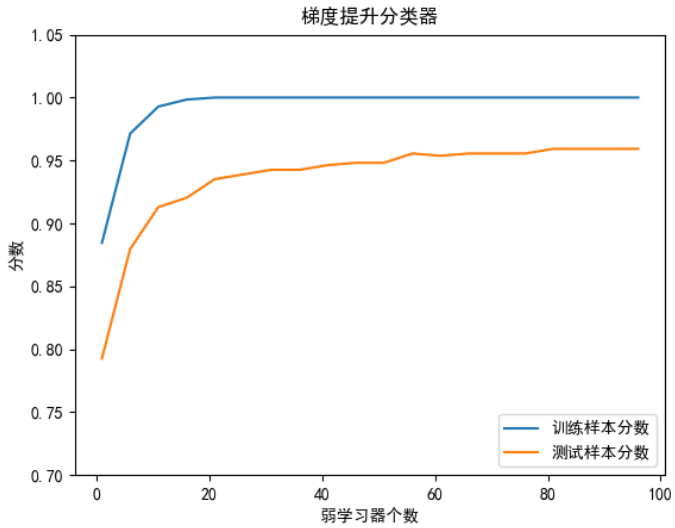


图 7: 手写数字得分随弱学习器数目的变化

案例实战 8.14 糖尿病数据集梯度提升回归。
diabetes_gradient_boost_regressor_sklearn.py

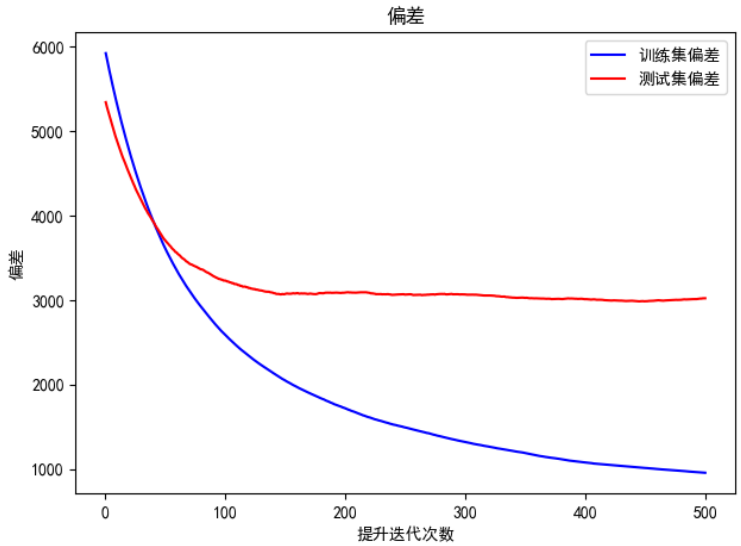


图 8: 糖尿病数据集的偏差随迭代次数的变化