机器学习应用实践（实验六）——集成学习之 AdaBoost、GBDT

一、实验目的

1、熟悉 AdaBoost 算法的原理。

2、掌握 scikit-learn 中 AdaBoost 算法的使用。

3、具备 AdaBoost 算法的编程实现能力。

4、理解 GBDT 算法的原理。

5、掌握 scikit-learn 中 GBDT 算法的使用。

二、实验准备

1、回顾 AdaBoost 算法的原理。

2、学习 scikit-learn 中 AdaBoost 算法涉及到的相关知识。

3、学习 GBDT 算法的原理。

4、学习 scikit-learn 中 GBDT 算法涉及到的相关知识。

三、AdaBoost 相关知识介绍

1、Boosting 方法的基本思想

在集成学习的“弱分类器集成”领域，除了通过降低方差来降低整体泛化误差的装袋法 Bagging，还有通过降低整体偏差来降低泛化误差的提升法 Boosting。与 Bagging 相比，Boosting 算法在操作和原理上的难度都更大，但 Boosting 类算法在模型效果方面的表现在整个弱分类器集成领域都非常突出。当代知名的 Boosting 算法当中，XGBoost、LightGBM 与 CatBoost 都是机器学习领域最强大的学习器之一。

在 Boosting 集成算法当中，采用串行建立多个弱评估器（基本是决策树），下一个弱评估器的建立依赖于上一个弱评估器的评估结果（弱评估器之间是相互依赖），最终综合多个弱评估器的结果进行输出。Boosting 的不同算法之间的核心区别就在于上一个弱评估器的评估结果具体如何影响下一个弱评估器的建立过程。

在 sklearn 当中，提供了一些 Boosting 集成算法，包括：AdaBoost(Boosting 入门算法)，梯度提升树 GBDT(Gradient Boosting Decision Tree，性能最稳定、奠定了 Boosting 集成算法效果基础)，直方提升树 HGBT(Hist Gradient Boosting Tree)等。

除了 sklean，研究者们还创造了大量基于 GBDT 进行改造的提升类算法，这些算法大多需要从第三方库进行调用。例如极限提升树 XGBoost(Extreme Gradient Boosting Tree)，轻量梯度提升树 LightGBM(Light Gradiant Boosting Machine)，以及离散提升树 CatBoost(Categorial Boosting Tree)等，如表 1 所示。

表 1 Boosting 算法及库调用类

Boosting 集成算法	库	sklearn.ensemble 中的类

AdaBoost 分类	sklearn	AdaBoostClassifier
AdaBoost 回归	sklearn	AdaBoostRegressor
梯度提升树 GBDT 分类	sklearn	GradientBoostingClassifier
梯度提升树 GBDT 回归	sklearn	GradientBoostingRegressor
直方提升树 HGBT 分类	sklearn	HistGradientBoostingClassifier
直方提升树 HGBT 回归	sklearn	HistGradientBoostingRegressor
极限提升树 XGBoost	第三方库 xgboost	xgboost.train()
轻量梯度提升树 LightGBM	第三方库 lightgbm	lightgbm.train()
离散提升树 CatBoost	第三方库 catboost	catboost.train()

2、AdaBoost 算法概述

AdaBoost(Adaptive Boosting，自适应提升法)是 Boosting 算法中最著名的算法之一。它的主要贡献在于实现了两个变化：

（1）首次实现根据之前弱评估器的结果自适应地影响后续建模过程。

（2）在 Boosting 算法中，首次实现输出结果综合考虑全部弱评估器结果。

AdaBoost 方法是一种迭代算法，它的基本构筑过程如下：

首先，在全部训练样本上建立一颗决策树（选择误差最小的分类器作为本轮弱评估器），每一个训练样本都被赋予一个权重，表明它被某个分类器选入训练集的概率。使用该决策树进行预测，根据预测的结果计算损失函数，自适应地调整样本点的权重，将错分样本点的权重提高，将正确分类样本点的权重降低。

然后，使用加权后的数据集被用于训练下一颗决策树。不断重复上述过程，直到达到某个预定的足够小的错误率或达到评估器的最大数目而停止。这个过程有意地加重“难以被分类正确的样本”的权重，同时降低容易被分类正确的样本”的权重，通过这样的方式，AdaBoost方法能“聚焦于”那些较难分（更富信息）的样本上。每一颗决策树的结果也都通过影响样本权重（即影响数据的分布）来影响下一颗树的建立。

当全部弱评估器都建立完成之后，集成算法的输出就是所有弱评估器的输出值的加权平均（弱评估器的权重也在建树的过程中被自适应地计算出来）。

AdaBoost在完成分类任务时，弱评估器是分类树，当AdaBoost完成回归任务时，弱评估器是回归树。这一点与其他的Boosting算法不同，其他的Boosting算法无论处理分类还是回归任务，弱评估器均为回归树。

3、scikit-learn 中 AdaBoostClassifier 与 AdaBoostRegressor API 介绍

在sklearn中，AdaBoost回归器由类sklearn.ensemble.AdaBoostRegressor实现，AdaBoost分类器则由类sklearn.ensemble.AdaBoostClassifier实现。AdaBoost回归器和分类器的参数也是非常相似的，下面以AdaBoost分类器为例介绍其中参数、属性和方法的含义和使用。程序示例如下：

```
>>> from sklearn.ensemble import AdaBoostClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = AdaBoostClassifier(n_estimators=100, random_state=0)
>>> clf.fit(X, y)
AdaBoostClassifier(n_estimators=100, random_state=0)
>>> clf.predict([[0, 0, 0, 0]])
array([1])
>>> clf.score(X, y)
0.983...
```

同时我们也可以使用 sklearn 中的交叉验证方法来实现 AdaBoostClassifier。程序示例如下：

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import load_iris
>>> from sklearn.ensemble import AdaBoostClassifier

>>> X, y = load_iris(return_X_y=True)
>>> clf = AdaBoostClassifier(n_estimators=100)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.9...
```

（1）重要参数

●base_estimators：object, default = None。base_estimator 是规定 AdaBoost 中使用基评估器的参数。Boosting 算法通过降低偏差来降低整体泛化误差，因此可以使用任意基评估器。在 sklearn 中，AdaBoost 分类器的默认基估计器是最大深度为 1 的“树桩”DecisionTreeClassifier(max_depth=1)，即只能分枝一次。AdaBoost 回归器的默认基评估器是最大深度为 3 的“树苗”，因为 Boosting 算法中回归任务往往更加复杂。也可以自定义某种弱评估器进行输入。

当模型建好之后，我们可以使用属性 base_estimator_来查看基评估器，同时也可以使用 estimators_来查看当前集成模型中所有基评估器的情况。程序示例如下：

```
AdaBoostClassifier = AdaBoostClassifier(n_estimators=4).fit(digits_X,digits_y)
AdaBoostClassifier.base_estimator_
```

```
DecisionTreeClassifier(max_depth=1)
```

```
AdaBoostClassifier.estimators_
```

```
[DecisionTreeClassifier(max_depth=1, random_state=2120684219),
 DecisionTreeClassifier(max_depth=1, random_state=921626904),
 DecisionTreeClassifier(max_depth=1, random_state=1884024666),
 DecisionTreeClassifier(max_depth=1, random_state=1522330772)]
```

●n_estimators：int，default = 50。AdaBoost 中基评估器的数量，默认为决策树的数量。如果完全拟合，学习过程就会提前停止。

●learning_rate：float, default = 1。学习率 learning_rate 可以改变每个分类器的贡献程度。如公式$H_t(x) = H_{t-1}(x) + \eta * \alpha_t h_t(x)$，第 t 轮迭代得到的集成结果为$H_t(x)$，它是在第$(t-1)$轮迭代得到的集成结果$H_{t-1}(x)$的基础上，叠加第 t 轮训练得到的模型的输出$h_t(x)$*模型权重$\alpha_t$，这里在每轮的模型加权输出基础上增加学习率$\eta$，用来防止过拟合。当 learning_rate 较小，需要更多的基评估器达到

更好的评估效果；若 learning_rate 较大时，则可以使用更少的基评估器，因此 learning_rate 和 n_estimators 之间存在权衡关系，往往需要同时调节这两个参数。

●random_state：控制每个 base_estimator 在每次增强迭代中给定的随机种子。仅在 base_estimator 引入 random_state 时使用它。AdaBoost 本身不具有随机性，它的随机性在于使用何种基评估器，若基评估器具有随机性，AdaBoost 才需设置 random_state 参数。

●algorithm：该参数是针对分类器设置的参数，包括"SAMME"（Discrete AdaBoost）与"SAMME.R"（Real AdaBoost）两种算法。两种算法在数学流程上的区别并不大，只不过"SAMME"是基于算法输出的具体分类结果（例如-1，1）进行计算，而"SAMME.R"则是基于弱分配器输出的概率值进行计算，两种算法都支持在 AdaBoost 上完成多分类任务，但"SAMME.R"往往能够得到更好的结果，因此 sklearn 中的默认值是"SAMME.R"。AdaBoost 分类器无论哪种算法，其损失函数都是经典的指数损失：$\ell_{\exp}(H(\boldsymbol{x})|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H(x)}\right]$。

由于 AdaBoost 使用的基评估器是深度非常浅的 CART 树，所以树模型本身的剪枝参数对于降低模型复杂度基本起不到作用，可以增加模型复杂度。当集成模型出现过拟合时，只能通过 learning_rate 和 n_estimators 进行调参。

（2）重要属性

●base_estimateor_：基评估器。

●estimators_：拟合的基评估器(决策树)列表。

●estimator_weights_：每个基评估器的权重集合。

●estimator_errors_：每个基评估器的泛化误差集合。

●feature_importances_：特征重要性。

（3）重要方法

方法	说明
decision_function(self,X)	计算样本 X 的决策函数。
fit(self, X, y[, sample_weight,])	根据训练集（X，y）建立 boosting 分类器。
get_params([deep])	获取此估算器的参数。
predict(X)	预测 X 的类别或回归值。
predict_log_proba(X)	预测输入样本 X 的类对数概率。
predict_proba(X)	预测输入样本 X 的类别概率。
score(X, y[, sample_weight])	返回给定测试数据和标签上的平均准确度。
set_params(**params)	设置此估算器的参数。
staged_decision_function(X)	返回每次迭代建立的决策函数。

方法	说明
staged_predict(X)	返回每次迭代建立的决策树对 X 的阶段预测值。
staged_predict_proba(X)	返回每次迭代建立的决策树对 X 的阶段类概率。
staged_score(X, y[, sample_weight])	返每次迭代建立的决策树上的预测评分。

可以看到，AdaBoostClassifier 中提供了一些不同于之前学到过的机器学习算法的接口，就是 staged_decision_function 、staged_predict、staged_predict_proba、staged_score，通过这些接口可以获得每次 Boosting 迭代后的基评估器的决策函数、预测值、预测类概率值和预测分数。

AdaBoostRegressor 采用的是 AdaBoost.R2 算法，它包括了对数据进行有放回抽样、按损失函数结果调整样本权重、自动计算弱分类器权重、并输出预测结果等 AdaBoost 算法经典的全流程。AdaBoostRegressor 独有的参数是 loss 参数。

●loss：损失函数，包括'linear', 'square', 'exponential' , default = 'linear'。每次增强迭代后，更新权重时，会用到损失函数。'linear'：线性损失函数，'square'：平方损失函数，'exponential'：指数损失函数。

4、AdaBoost 算法流程与实现

按照 AdaBoost 中的算法约定，二分类模型中标签分别用-1 代表负类，用 1 代表正类。

由于 AdaBoost 算法默认的基评估器是决策树，则首先需要实现最大深度为 1 的“决策树”。与决策树和随机森林不同的是，AdaBoost 算法默认的基评估器“决策树”只分枝一次，且特征选择标准不再是节点不纯度。实现最大深度为 1 的“决策树”的流程如下：

1）确定选择哪一个特征作为分类依据；

2）确定以哪个阈值作为二分类划分条件；

3）特征与阈值的判别符号如何确定，即大于阈值还是小于阈值判断为正类；

4）特征选择的标准：加权分类误差率最低，即错分样本的权重之和最小。

循环上述 1）至 4）过程，找到最低的加权分类误差率，及其对应的特征、阈值和判别符号。

在实现决策树桩作为基分类器的基础上，实现 Adaboost 算法仅需按照算法流程逐层训练即可。

输入	训练集：$\{(x_1,y_1),(x_2,y_2),\dots,(x_N,y_N)\}, y_i \in \{+1,-1\}$ 基学习器优化算法 $\mathfrak{L}(h;X,Y,\mathcal{D})$ 训练轮数T
1	样本权重初始化 $\mathcal{D}_1(x) = \left[\omega_i^{(t)} \middle\| i = 1,\dots,N; t = 1\right] = [\frac{1}{N},\dots,\frac{1}{N}]$
2	for $t = 1,2,\dots,T$：
3	$h_t = \mathfrak{L}(h;X,Y,\mathcal{D}_t(X))$
4	$\epsilon_t = P_{x\sim\mathcal{D}_t}(h_t(x_i) \neq y_i) = \sum_{i=1}^{N} \omega_i I(h_t(x_i) \neq y_i)$
5	if $\epsilon_t > 0.5$ then break
6	基分类器权重：$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
7	更新样本权重：$\mathcal{D}_{t+1}(x) = \frac{\mathcal{D}_t(x)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
输出	$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t\, h_t(x))$

四、GBDT 相关知识介绍

1、梯度提升树算法概述

梯度提升树[1]（Gradient Boosted Decision Tree，缩写为 GBDT）最初是 Friedman 在 2001 提出的，它融合了 Bagging 与 Boosting 的思想，可以接受各类弱学习器作为输入，因性能良好应用广泛，也是 XGBoost、LightGBM 等算法的基石。GBDT 在整体建树过程中，与 AdaBoost 相比，做出了以下几个关键的改变：

（1）弱学习器为回归树

对于 AdaBoost 或随机森林算法来说，当集成算法执行回归任务时，弱学习器是回归器；当集成算法执行分类任务时，弱学习器是分类器。而 GBDT 无论执行回归还是分类任务，弱学习器都是回归器(cart 回归树)。GBDT 通过 sigmoid 或 softmax 函数输出分类概率，从而实现分类任务。

（2）损失函数的拓展

在 GBDT 当中，损失函数范围不再局限于固定或单一的某个损失函数，而从数学原理上推广到了任意可微的函数，因此 GBDT 算法中可选的损失函数很多，分类任务中可选择交叉熵损失、指数损失等，回归任务中可选择平方损失、绝对值损失、Huber 损失、quantile 损失等。

（3）建立拟合负梯度的模型

GBDT 延续了 AdaBoost 自适应调整弱学习器的构建思想，但却不像 AdaBoost 一样通过调整数据分布(修改样本权重)来间接影响后续弱学习器。GBDT 通过修改后续弱学习器的拟合目标来直接影响后续弱学习器的结构。在之前学习过的机器学习方法中，全部都使用样本的真实标签作为预测目标进行建模。而在 GBDT 当中，提出以损失函数的负梯度为样本标签建立弱学习器，每个弱学习器会朝着使损失函数下降最快的方向迭代。

具体来讲，梯度下降算法应用于线性回归模型时，需要解决的最优化问题是找到能够使损失函数最小一组模型的参数$\widehat{\boldsymbol{w}}$，是参数的优化。因此是对线性回归模型中的$\widehat{\boldsymbol{w}}$参数进行迭代，迭代公式：

$$\widehat{\boldsymbol{w}}(t) = \widehat{\boldsymbol{w}}(t-1) - \eta \frac{\partial L\big(\widehat{\boldsymbol{w}}(t-1)\big)}{\partial \widehat{\boldsymbol{w}}(t-1)} \qquad (1)$$

其中η为学习率，每次迭代模型都沿着使损失函数下降最快的方向变化一点，直到达到损失函数的局部或全局最小值。

与公式(1)类比，GBDT 将梯度下降算法应用于集成模型时，需要解决的最优化问题是找到能够使损失函数最小的一组弱学习器的集合$H(\boldsymbol{x})$，是函数空间的优化。因此是对集成模型$H(\boldsymbol{x})$进行迭代，迭代公式：

$$H_t(\boldsymbol{x}) = H_{t-1}(\boldsymbol{x}) - \eta \frac{\partial L\big(\boldsymbol{y}, H_{t-1}(\boldsymbol{x})\big)}{\partial H_{t-1}(\boldsymbol{x})} = H_{t-1}(\boldsymbol{x}) + \eta \left(- \frac{\partial L\big(\boldsymbol{y}, H_{t-1}(\boldsymbol{x})\big)}{\partial H_{t-1}(\boldsymbol{x})} \right) \qquad (2)$$

其中$L(y, H(\boldsymbol{x}))$为某种衡量样本真实标签 y 与集成模型的预测输出$H(\boldsymbol{x})$之间的差异的损失函数，可以是平方误差、交叉熵损失等，是全部样本的损失之和$L(y, H(\boldsymbol{x})) = \sum_{i=1}^{N} l(y_i, H(\boldsymbol{x}_i))$。这个过程被称为函数空间上的梯度下降(Gradient Descent in Function Space)。

---

[1] Friedman, J.H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29, 1189-1232.

GBDT 是 boosting 算法，集成模型为$H(\boldsymbol{x})=\sum_{t=1}^{T}\alpha_t h_t(\boldsymbol{x})$，boosting 策略采用前向分步优化，将$H_t(\boldsymbol{x})$进行拆分，在新增弱学习器的训练过程中$H_{t-1}(\boldsymbol{x})$保持不变，每一步优化只新增一个弱学习器$h_t$，公式为：

$$H_t(\boldsymbol{x})=H_{t-1}(\boldsymbol{x})+\alpha_t h_t(\boldsymbol{x}) \tag{3}$$

对比公式(2)和公式(3)，新增的弱学习器h_t可以通过拟合负梯度$-\frac{\partial L(\boldsymbol{y},H_{t-1}(\boldsymbol{x}))}{\partial H_{t-1}(\boldsymbol{x})}$来建立，即令样本为$(\boldsymbol{x},-\frac{\partial L(\boldsymbol{y},H_{t-1}(\boldsymbol{x}))}{\partial H_{t-1}(\boldsymbol{x})})$建立弱学习器$h_t$，权重$\alpha_t$为学习率$\eta$，从而使每次迭代都沿着损失函数下降最快的方向进行。经过 T 轮迭代，每次迭代后的集成模型如下：

$$H_1(\boldsymbol{x})=H_0(\boldsymbol{x})+\eta\left(-\frac{\partial L(\boldsymbol{y},H_0(\boldsymbol{x}))}{\partial H_0(\boldsymbol{x})}\right) \tag{4}$$

$$H_2(\boldsymbol{x})=H_1(\boldsymbol{x})+\eta\left(-\frac{\partial L(\boldsymbol{y},H_1(\boldsymbol{x}))}{\partial H_1(\boldsymbol{x})}\right) \tag{5}$$

…

$$H_T(\boldsymbol{x})=H_{T-1}(\boldsymbol{x})+\eta\left(-\frac{\partial L(\boldsymbol{y},H_{T-1}(\boldsymbol{x}))}{\partial H_{T-1}(\boldsymbol{x})}\right) \tag{6}$$

将上式(4-6)左右两侧相加，得到$H_T(\boldsymbol{x})=H_0(\boldsymbol{x})+\eta\sum_{t=0}^{T-1}\left(-\frac{\partial L(\boldsymbol{y},H_t(\boldsymbol{x}))}{\partial H_t(\boldsymbol{x})}\right)=H_0(\boldsymbol{x})+\eta\sum_{t=1}^{T}h_t(\boldsymbol{x})$

可以看到，最终输出的集成模型是初始学习器$H_0(\boldsymbol{x})$和 T 个弱学习器$h_t(\boldsymbol{x})$(拟合负梯度)的结果叠加而成。

通过让弱学习器拟合负梯度，弱学习器上的结果可以直接影响损失函数，保证损失函数的降低，从而实现 Boosting 算法的根本目标：降低偏差。这一过程避免了许多在其他算法中需要详细讨论的问题：如每个弱学习器的权重α_t如何设置、弱评估器的置信度如何。在 AdaBoost 算法当中，损失函数是“间接”影响弱学习器的建立的，因此有的弱学习器能够降低损失函数，而有的弱学习器不能降低损失函数，因此在对弱学习器集成之前，需要先求弱评估器的置信度(即错误率)，然后再给与置信度高的弱学习器更高的权重，权重α存在的根本意义是为了调节单一弱学习器对集成模型$H(\boldsymbol{x})$的贡献程度。但在 GBDT 当中，由于所有的弱学习器都能够降低损失函数，只不过降低的程度不同，因此就不再需要置信度/贡献度的衡量，因此就不再需要权重。

（5）Bagging 思想的融入

GBDT 加入了 Bagging 随机抽样的思想，在每次建树之前，允许对样本和特征进行抽样来增大弱学习器之间的独立性（因此也可以有袋外数据集），使得 GBDT 在降低偏差的同时，也能够降低一些方差（减轻过拟合）。

2、梯度提升树的建模流程

GBDT 回归的基本流程如下：

（1）初始化迭代起点，确定初始学习器$H_0(\boldsymbol{x})$，$H_0(\boldsymbol{x})=argmin_\rho\sum_{i=1}^{N}l(y_i,\rho)$。

（在 sklearn GBDT 的 API 中由参数 init 确定。）

（2）$for\ t=1,2,\cdots,T$:

1）在现有训练样本(样本量为 N)中，有放回抽取 M 个样本，构成样本子集N_t建立基评估器。（在 sklearn GBDT 的 API 中由参数 subsample 确定抽样比例，M=N*subsample。）

2）$\tilde{y_i} = -\frac{\partial L(y_i, H(x_i))}{\partial H(x_i)}|_{H(x)=H_{t-1}(x)}, i = 1,2,\cdots,M$。如果以全部样本进行训练，则$M = N$。

（在 sklearn GBDT 的 API 中提供了 7 种损失函数，由 loss 参数设置，分类和回归问题使用的损失函数不同。）

3）以$\tilde{y_i}$作为样本子集N_t的新标签，建立 CART 回归树$h_t(\boldsymbol{x_i})$。

4）集成模型为：$H_t(\boldsymbol{x}) = H_{t-1}(\boldsymbol{x}) + \eta h_t(\boldsymbol{x})$，$\eta$为学习率。

（学习率在 sklearn GBDT 的 API 中由参数 learning_rate 确定。）

（3）循环结束，输出：$H_T(\boldsymbol{x}) = H_0(x) + \eta \sum_{t=1}^{T} h_t(x)$

注意：以平方误差为损失函数的 GBDT 建模实例详见“附件：GBDT 简单例子.ipynb”。

3、scikit-learn 中 GradientBoostingClassifier 与 GradientBoostingRegressor API 介绍

在 sklearn 中，GBDT 回归器由类 sklearn.ensemble.GradientBoostingRegressor 实现，GBDT 分类器则由类 sklearn.ensemble.GradientBoostingClassifier 实现。GBDT 回归器和分类器的参数非常相似，参数量非常多，表 2 中分类整理了 GBDT 的参数和属性，其中加粗的参数和属性是其他学过的集成模型中没有的，下面进行重点介绍。

表 2　GBDT 参数/属性表

类型	参数/属性
迭代过程	参数：n_estimators, learning_rate, **loss, alpha, init** 属性：loss_, init_, estimators_, n_estimators_
提前停止	参数：**validation_fraction, n_iter_no_change, tol**
弱评估器结构（决策树参数）	参数：**criterion**, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease
训练数据	参数：**subsample**, max_features, random_state 属性：**oob_improvement, train_score_**
其他	参数：ccp_alpha, warm_start

3.1 迭代过程参数和属性

●init：设置初始学习器$H_0(x)$的评估器对象。可以输入任意评估器, 'zero', 'None'，默认为 None 对象。

（1）当输入任意评估器时，评估器必须要具备 fit 以及 predict_proba 功能，即我们可以使用决策树、逻辑回归等可以输出概率的模型。如果输入一个已经训练过、且精细化调参后的模型，将会给 GBDT 树打下坚实的基础。

（2）输入字符串'zero'，$H_0(x) = 0$。

（3）不填写，或填写为 None 对象，sklearn 会使用类 DummyEstimator 生成预测结果作为$H_0(x)$。对于回归问题，如果损失函数采用'squared_error'平方误差，则以标签均值作为预测值。若损失函数采用'absolute_error'绝对误差，则以标签的中值作为预测值。

与参数 init 相对的属性是 init_，示例程序如下：

```
reg = GradientBoostingRegressor(init_ = None, random_state=1412)
reg.fit(X,y).init_ #返回sklearn中的评估器DummyRegressor
```

```
DummyRegressor()
```

```
rf = RandomForestRegressor(max_depth=22, max_features=14, min_impurity_decrease=0,
                           n_estimators=89, n_jobs=-1, random_state=1412,
                           verbose=False)
reg = GradientBoostingRegressor(init_ = rf, random_state=1412)
reg.fit(X,y).init_
```

```
RandomForestRegressor(max_depth=22, max_features=14, min_impurity_decrease=0,
                      n_estimators=89, n_jobs=-1, random_state=1412,
                      verbose=False)
```

●n_estimators_：实际迭代次数，int 类型。会因为“提前停止”而小于设置的 n_estimators 参数值。

●estimators_：实际建立的弱学习器数量，数组类型。回归任务，返回数组的形状是(n_estimators, 1)。多分类任务，返回数组的形状是(n_estimators, n_classes)。

GBDT 与 AdaBoost 及随机森林的关键区别之一，是 GBDT 中所有的弱学习器都是回归树，因此在实际调用梯度提升树完成分类任务时，需要 softmax 函数或 sigmoid 函数对回归树输出的结果进行处理。

对于二分类任务，$p(\hat{y}=1|\boldsymbol{x}_i)=\sigma(H(\boldsymbol{x}_i))$，其中$\sigma$是 sigmoid 函数，当$p(\hat{y}=1|\boldsymbol{x}_i)>0.5$时，样本$xi$的预测类别为 1，反之则为 0。此时，集成模型中每次迭代都只建立一个弱学习器，n_estimators_= estimators_。

对于 K 分类任务(类别为 1,2,⋯,K)，需要按照y=1, y=2,⋯, y= K 建立 K 个集成模型$H^1(\boldsymbol{x}_i)$，$H^2(\boldsymbol{x}_i)$，…，$H^K(\boldsymbol{x}_i)$，使用 softmax 函数将 K 个集成模型的输出结果转化为 K 个和为 1 的概率，每个概率为$softmax(H^k(\boldsymbol{x}_i))=\frac{e^{H^k(x_i)}}{\sum_{k=1}^{K}e^{H^k(x_i)}}$，$k=1,2,\cdots,$K。此时，每次迭代都建立 K 个弱学习器，estimators_=K*n_estimators_。这一现象只在弱学习器为回归器的各类 boosting 算法中出现，对于弱学习器可以是回归树也可以是分类树的随机森林、Adaboost 来说，多分类时每个类别对应的概率是在叶子节点上自然生成的。因此多分类问题在随机森林上的计算可能会表现得更快。

●loss：损失函数。GBDT 分类器和 GBDT 回归器的损失函数不同。

●alpha：为'huber'损失和'quantile'损失中的系数。

作为基于 AdaBoost 改进的 Boosting 算法，GBDT 的创新之一是将损失函数从有限的指数损失、MSE 等推广到了任意可微函数，因此 GBDT 的损失函数选择很多，这些损失函数可以作为可调的超参数进行寻优。

在 sklearn 中，对于 GBDT 分类器来说，loss 的备选项有：'deviance' , 'exponential', 默认值='deviance'。其中'deviance'直译为偏差，特指交叉熵损失，而'exponential'则特指 AdaBoost 中使用的指数损失函数。

对于 GBDT 回归器来说，loss 的备选有：'squared_error' , 'absolute_error', 'huber', 'quantile'，默认值='squared_error'。其中'squared_error'是平方误差，'absolute_error'是绝对误差，这是一个鲁棒的损失函数。'huber'是以上两者的结合。'quantile'则表示使用分位数回归的弹球损失 pinball_loss。

'squared_error'平方误差的公式为：$L=\sum_{i=1}^{N}\frac{1}{2}(y_i-H(x_i))^2$

'absolute_error'绝对误差的公式为：$L=\sum_{i=1}^{N}|y_i-H(x_i)|$

'huber'损失的公式为：$L=\sum_{i=1}^{N}l(y_i,H(x_i))$

$$\text{其中，}l=\begin{cases}\frac{1}{2}\left(y_i-H(x_i)\right)^2, & |y_i-H(x_i)|\le\alpha\\ \alpha\left(|y_i-H(x_i)|-\frac{\alpha}{2}\right), & |y_i-H(x_i)|>\alpha'\end{cases}\quad\alpha\in(0,1)$$

'quantile'损失的公式为：$L=\sum_{i=1}^{N}l(y_i,H(x_i))$

$$\text{其中，}l=\begin{cases}\alpha\left(y_i-H(x_i)\right), & y_i-H(x_i)>0\\ 0, & y_i-H(x_i)=0\\ (1-\alpha)\left(y_i-H(x_i)\right), & y_i-H(x_i)<0\end{cases}\quad\alpha\in(0,1)$$

其中α是需要设置的超参数，由参数 alpha 控制。在’huber’损失中，alpha 是阈值，在’quantile’损失中，alpha 用于辅助计算损失函数的输出结果，默认为 0.9。

上述四种损失函数因其特点不同，具有不同的应用场合：

（1）当高度关注离群值、并且希望努力将离群值预测正确时，选择平方误差。

在实际进行预测时，离群值往往比较难以预测，因此离群样本的预测值和真实值之间的差异一般会较大。MSE 作为预测值和真实值差值的平方，会放大离群值的影响，会让算法更加向学习离群值的方向进化，这可以帮助算法更好地预测离群值。

（2）努力排除离群值的影响、更关注非离群值的时候，选择绝对误差。

MAE 对一切样本都一视同仁，对所有的差异都只求绝对值，因此会保留样本差异最原始的状态。相比其 MSE，MAE 对离群值完全不敏感，这可以有效地降低 GBDT 在离群值上的注意力。

（3）试图平衡离群值与非离群值、没有偏好时，选择 Huber 或者 Quantileloss。

'huber'损失结合了 MSE 与 MAE，当预测值与真实值的差异大于阈值时，则取绝对值；小于阈值时，则取平方。在真实数据中，部分离群值的差异会大于阈值，部分离群值的差异会小于阈值，因此比起全部取绝对值的 MAE，'huber'损失会将部分离群值的真实预测差异求平方，相当于放大了离群值的影响，但这种影响又比 MSE 小。因此'huber'损失是位于 MSE 和 MAE 之间的、对离群值相对不敏感的损失。

3.2 弱学习器(CART 树)参数和属性

在 GBDT 中，弱学习器(CART 树)参数和属性只有 criterion 与决策树存在差异，其他都与决策树相同。无论是分类器还是回归器，默认的 CART 树的最大深度都为 3。

当随机森林处于过拟合状态时，还可通过降低弱学习器复杂度的手段控制过拟合，但 GBDT 等 Boosting 算法处于过拟合状态时，无法从调整各个弱评估器的复杂度上入手，因为弱评估器的复杂

度已经很低了。也因此，通常认为 Boosting 算法比 Bagging 算法更不容易过拟合，一般在相似的数据上，Boosting 算法表现出的过拟合程度会较轻。

●criterion：特征选择标准。'friedman_mse', 'squared_error', 'mae'，默认为'friedman_mse'。

在 sklearn 当中，GBDT 中的弱学习器是 CART 树，因此每棵树在建立时都依赖于 CART 树分枝的规则进行建立。CART 树为二叉树，每次在分枝时都按照令左右节点的不纯度之和与父节点的不纯度之间的差值(称为不纯度下降量)最大的分枝方式，分为左节点(left)和右节点(right)。在 CART 树中进行分枝时，我们需要找到令左右节点的不纯度之和最小的分枝方式。不纯度的下降量越大，该分枝对于降低不纯度的贡献越大。

此处的'squared_error'表示均方误差，而 loss 参数中'squared_error'表示平方误差，二者计算公式不同。

'friedman_mse'弗里德曼均方误差是由 Friedman 在论文中提出的全新的误差计算方式，但弗里德曼没有给出均方误差公式的具体形式，而只提供了使用弗里德曼均方误差之后推导出的不纯度下降量的公式。该公式如下：

$$\frac{\omega_l \omega_r}{\omega_l + \omega_r} * \left(\frac{\sum_l (r_i - \hat{y}_i)^2}{\omega_l} - \frac{\sum_r (r_i - \hat{y}_i)^2}{\omega_r} \right)^2$$

其中ω是左右叶子节点上的样本量，当调整了样本的权重时，ω则是叶子节点上的样本权重。ri是样本的负梯度标签，$\hat{y}_i$是样本 i 在当前子节点下的预测值（即节点上的样本标签均值）。所以这个公式其实可以理解成：

左右叶子节点上样本量的调和平均 * (左叶子节点上均方误差 - 右叶子节点上的均方误差)的平方

根据论文中的描述，弗里德曼均方误差使用调和平均数($\frac{\omega_l \omega_r}{\omega_l + \omega_r}$)来控制左右叶子节点上的样本数量，相比普通地求均值，调和平均会受短板数据的影响（两个数据中只要有一个值比较小，他们的调和平均值就会较小，两个数据都较大时调和平均值就会越大），弗里德曼均方误差更注重使左右分支节点上的样本量/样本权重尽量接近。使用 CART 树默认的平方误差作为特征选择标准时，会综合考虑左右两个分支的样本不纯度之和最小，而弗里德曼均方误差更注重两个子集之间的 MSE 差距，如果两个子集之间的 MSE 差异很大，则说明其中一个子集的 MSE 一定很小，不纯度较低。这种方式会更快地得到一些纯度较高的节点，同时尽量考虑左右分支节点中的样本量/权重的平衡，可以让整体分支的效率更高。不过很多时候，树模型基于弗里德曼均方误差的分割与基于平方误差的分割会得到相同的结果。

3.3 提前停止过程参数和属性

在机器学习训练流程中，往往是通过给出一个限制参数来控制算法的停止，比如：通过超参数设置允许迭代的最大次数，或者允许建立的弱评估器的个数。因此无论算法是否在很短时间内就锁定了足够接近理论最小值的次小值、或者算法早已陷入了过拟合状态、甚至学习率太低导致算法无法收敛，大多数算法都会持续（且无效地）迭代下去，直到达到给出的限制参数取值。对于复杂度较高、数据量较大的 Boosting 集成算法来说，无效的迭代常常发生，因此作为众多 Boosting 算法的根基算法，梯度提升树自带了提前停止的相关超参数。梯度提升树需要提前停止的情况主要有如下

两种，如图 1 所示，横坐标为迭代次数，纵坐标为损失函数的值。在实际训练过程中，刚开始训练时，测试集和训练集上的损失一般都很高（有时，训练集上的损失甚至比测试集上的损失还高，这说明模型严重欠训练），但随着训练次数的增多，两种损失都会开始快速下降，一般训练集下降得更快，测试集下降得缓慢。

（1）当模型的训练分数和测试分数均已收敛，测试集上的损失不再下降、持续保持平稳时，继续训练会浪费训练资源，迭代下去模型也会停滞不前，算法效果的提升甚微，因此需要停止（如图 1 左图）。

（2）当训练集上的损失还是在稳步下降，测试集上的损失反而从之前的下降趋势变为开始上升，继续迭代下去测试集表现会越来越差，也就是过拟合（如图 1 右侧），需要在过拟合之前及时停止。

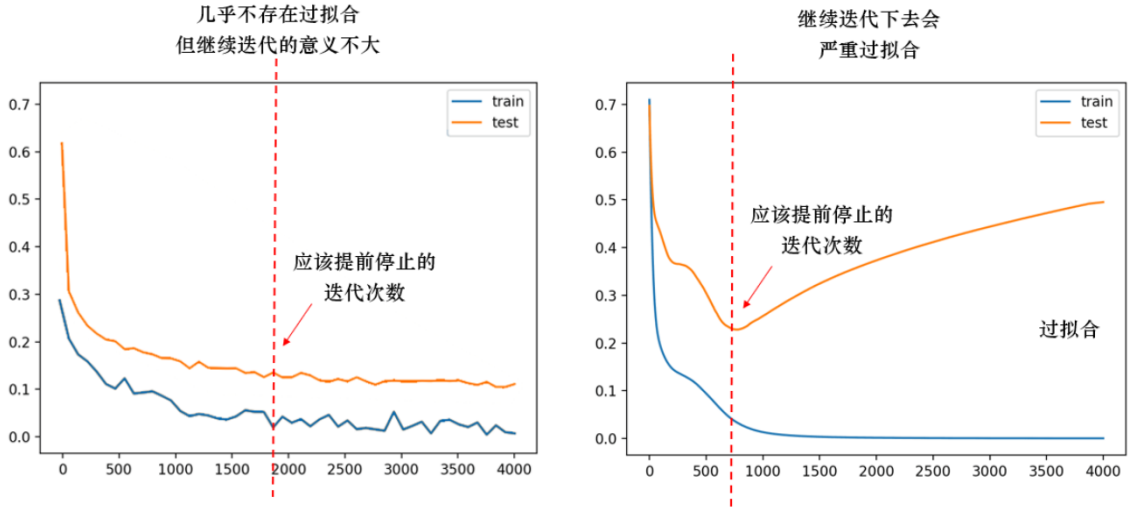


图 1 提前停止

我们可以通过如下方式确定“提前停止”的时刻：当连续 n_iter_no_change 次迭代中，验证集上损失函数的减小值都低于阈值 tol，或者验证集的分数提升值都低于阈值 tol 的时候，就停止迭代。此时，即便我们规定的 n_estimators 或者 max_iter 中的数量还没有到达，也可以停止算法，这种机制就是提前停止机制(Early Stopping)。在 GBDT 当中，这个流程刚好由以下三个参数控制：

●validation_fraction：从训练集中提取出，用于提前停止的验证数据的占比，值域为[0,1]，默认为 0.1。

●n_iter_no_change：当验证集上的损失函数值连续 n_iter_no_change 次没有下降或下降量不达阈值时，则触发提前停止。默认设置为 None，表示不进行提前停止。

●tol：损失函数下降的阈值，默认值为 1e-4(0.0001)。

需要注意的是，当提前停止条件被触发后，梯度提升树会停止训练，即停止建树。因此，当提前停止功能被设置打开时，我们使用属性 n_estimators_调出的结果很可能不足我们设置的 n_estimators，属性 estimators_中的树数量也可能变得更少。

3.4 袋外数据参数和属性

受到随机森林的启发，梯度提升树在每次建树之前，也允许模型对于样本数据进行随机有放回抽样，构建用于训练弱评估器的自助集。在梯度提升树的原理当中，每次建树之前进行随机抽样的梯度提升算法叫做随机梯度提升(Stochastic Gradient Boosting)。相比起传统的梯度提升树，随机提升树输出的结果往往方差更低，但偏差略高。如果发现 GBDT 的结果高度不稳定，则可以尝试使用随机提升树。

●subsample：设置用于训练单一基学习器的样本的比例。

在 GBDT 中，对样本的有放回抽样比例由参数 subsample 确定，当该参数被设置为 1 时，则不进行抽样，直接使用全部数据集进行训练。当该参数被设置为(0,1)之间的数字时，则使用随机梯度提升树，在每轮建树之前对样本进行抽样。对特征的抽样比例由参数 max_features 确定，随机模式则由参数 random_state 确定，这两个参数在 GBDT 当中的使用规则都与随机森林中完全一致。

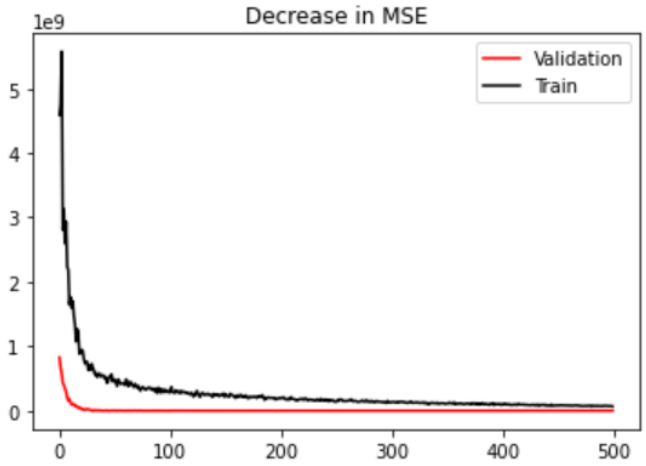
需要注意的是，如果 subsample<1，即存在有放回随机抽样时，当数据量足够大、抽样次数足够多时，大约会有 37%的数据被遗漏在“袋外”，没有参与训练，这部分数据在 GBDT 中被用于对每一个弱评估器的建立结果进行验证。

●oob_improvement_：在袋外样本上损失函数的增量，长度为 n_estimators 的数组。

●train_score_：在袋内样本上损失函数的增量，长度为 n_estimators 的数组。

在 GBDT 当中，袋外分数的变化值被储存在属性 oob_improvement_中，同时，GBDT 还会在每棵树的训练数据上保留袋内分数（in-bag）的变化，且储存在属性 train_score_当中。也就是说，即便在不做交叉验证的情况下，也可以通过属性 oob_improvement 与属性 train_score_来观察 GBDT 迭代的结果，以下图为例。

```
plt.plot(range(500), reg.oob_improvement_, color="red", label="Validation")
plt.plot(range(500), reg.train_score_, color="k", label="Train")
plt.title("Decrease in MSE")
plt.legend()
```



另外需要说明的是，GBDT 中的树是串行建立、且后面建立的树依赖于之前建树的结果，因此 sklearn 并没有提供 n_jobs 并行计算的参数给 Boosting 算法使用。更加先进的 Boosting 算法们已经实现了分枝并行，但 sklearn 还无法实现这个功能，因此 GBDT 的计算速度难以得到加速。

五、实验内容

1、题目一：采用 scikit-learn 中的 AdaBoostClassifier 对葡萄酒数据集进行预测。

具体要求：

（1）导入数据集：葡萄酒数据集在决策树内容中使用过。

（2）模型建立：使用 AdaBoostClassifier 建立分类模型（参数默认即可）。

（3）输出：模型评分（使用交叉验证得到模型的综合评分）。

【讨论一】模型在葡萄酒数据集上的表现如何？是过拟合还是欠拟合？

【讨论二】如何提升模型性能？模型超参数对性能有何影响？

（1）观察不同求解算法（"SAMME"和"SAMME.R"）的性能差异。

参考官网：https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_hastie_10_2.html#sphx-glr-auto-examples-ensemble-plot-adaboost-hastie-10-2-py。

（2）通过学习曲线分析 AdaBoost 算法中的 n_estimators 参数对模型性能的影响（其他参数不变）。

可通过交叉验证结果观察模型性能（如实验七随机森林算法实现的效果），也可使用 staged_predict 或 staged_score 获得每次迭代弱评估器的预测结果或评分，如图 2 所示。

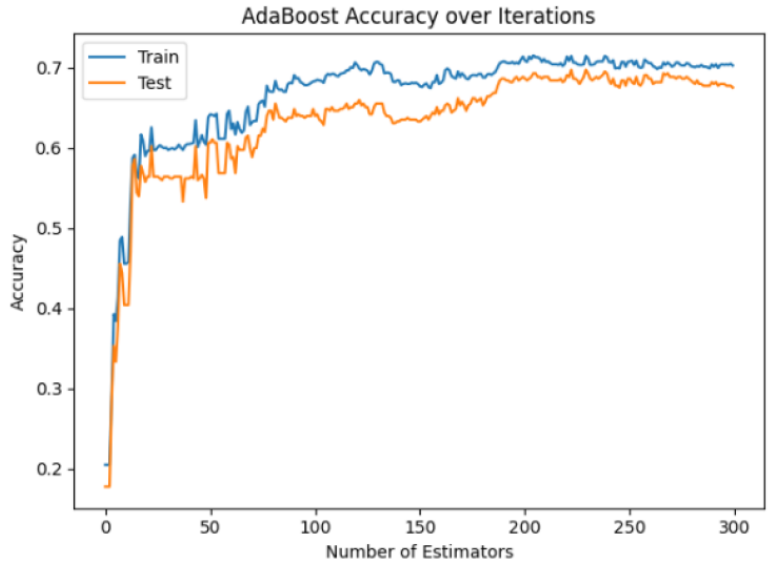


图 2 n_estimators=300 时，基评估器迭代偏差减小(性能提升)的过程

（3）通过学习曲线分析 AdaBoost 算法中的 learning_rate 参数对模型性能的影响。

可同时展现不同学习率下，模型性能随 n_estimators 参数的变化曲线，如图 3 所示。

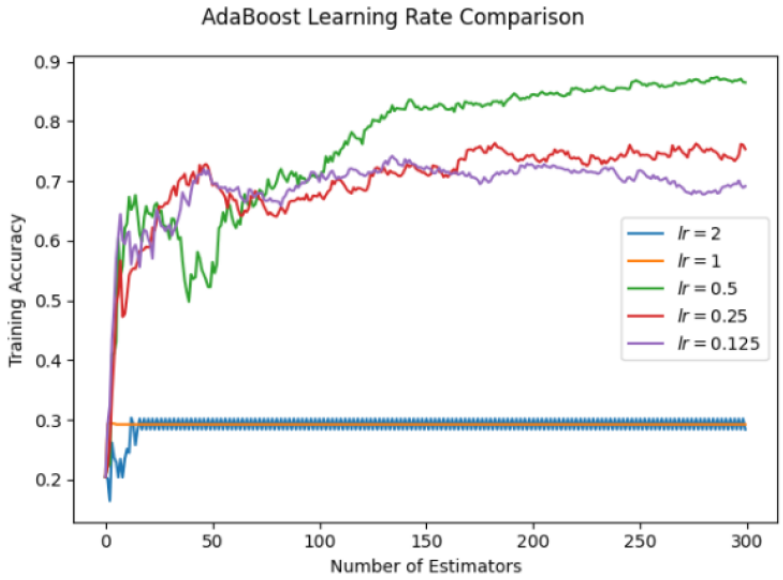


图 3 n_estimators=300 时，不同学习率下，基评估器迭代偏差减小(性能提升)的过程

（3）综合进行超参数选择，找到最好的一组超参数，在交叉验证集上验证模型效果，并与其他算法（决策树、随机森林等）进行对比。

2、题目二：编写 AdaBoost-SAMME 算法，并对乳腺癌数据集进行预测，并展示模型评分，与 sklearn 自带的评估器建模结果进行对比，或与之前学习过的 SVM 算法进行对比。

3、题目三（选做）：采用 scikit-learn 中的 GradientBoostingRegressor 对加州房价数据集进行预测。

具体要求：

（1）导入数据集：california 数据集在随机森林内容中使用过。

（2）模型建立：使用 GradientBoostingRegressor 建立分类模型。

（3）输出：模型评分（使用交叉验证得到模型的综合评分）。

交叉验证集的划分：`cv = KFold(n_splits=5, shuffle=True, random_state=1111)`

【讨论四】自行选择超参数寻优的方法，确定模型最优超参数，并将建模结果与之前学过的调参后的模型(如随机森林、Adaboost)进行比较（可从建模时间，模型预测效果等方面进行比较）。

提示：结合资料中介绍过的超参数寻优的方法，可通过学习曲线、袋外数据、提前停止等手段，帮助判断超参数范围，进而确定寻优的参数空间。