本节要点

- AdaBoost算法原理。
- 泰勒公式形式与作用。
- GBDT算法原理。

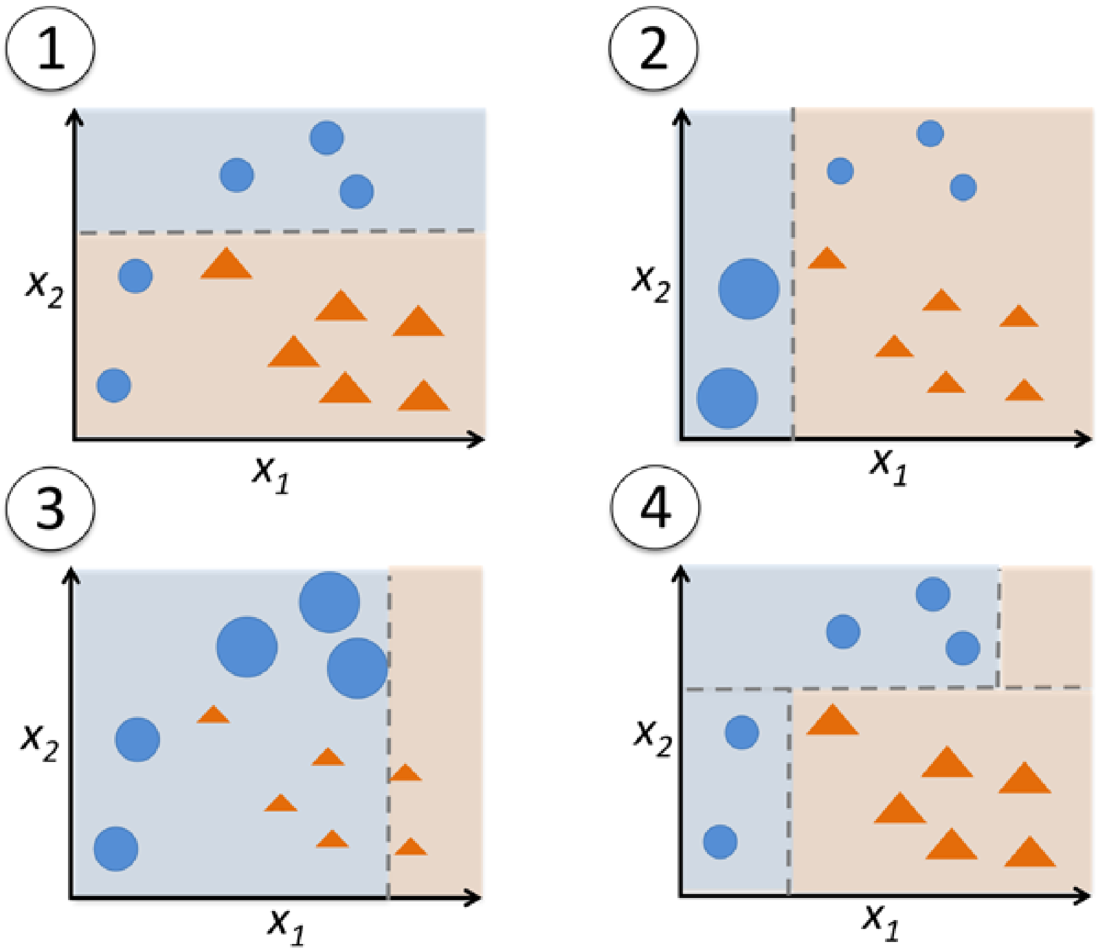
AdaBoost

AdaBoost（Adaptive Boosting——自适应提升），是一种集成方法，应用提升方法的思想。算法将多个弱评估器的输出结合起来创建一个强评估器，从而提高整体预测精度。

工作原理

AdaBoost的工作原理是：

- 在迭代过程中，调整错误分类样本的权重。
    - 如果样本预测正确，则降低权重。
    - 如果样本预测错误，则提高权重。
- 在更新后的权重上训练一个新的弱分类器，以专注于难以分类的样本。
    - 越难区分的样本在训练过程中会变得越重要。



提升预测

Adaboost算法含有两个权重，一个是样本的权重，一个是基本评估器的权重。

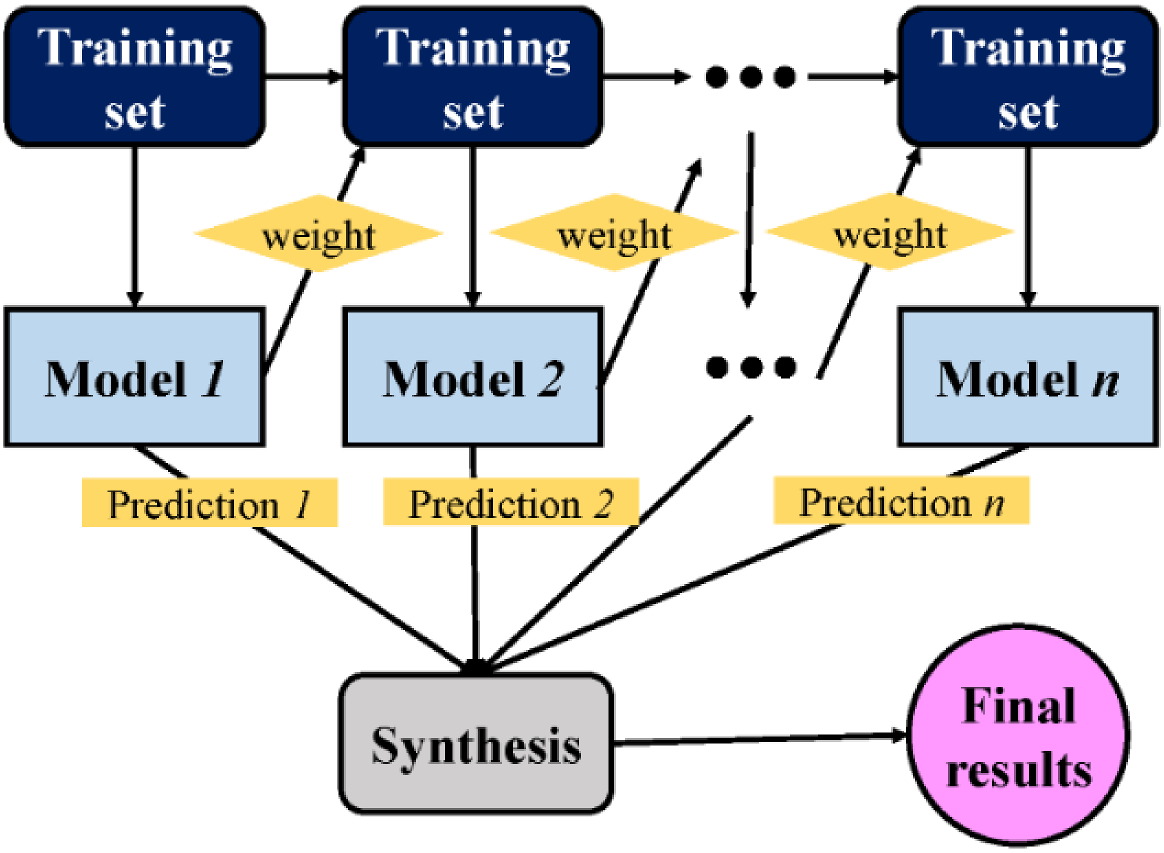
算法给误差率较小的基本评估器以较大的权重，给误差率较大的基本评估器以较小的权重。最终基本评估器的预测结果进行线性加权组合，得到最终的预测。

$$\hat{y} = F_n(x) = \sum_{i=1}^{n} \alpha_i f_i(x)$$

- α_i：每个基本评估器的权重。
- n：基本评估器的数量。

对于分类任务，则在最终的结果上进行sign函数的转换即可：

$$\hat{y} = sign(F_n(x)) = sign(\sum_{i=1}^{n} \alpha_i f_i(x))$$



AdaBoost算法步骤

AdaBoost算法步骤如下（以分类为例）：

1. 初始化每个样本的权重w，使得所有样本的初始权重相同，并且权重之和为1，即：

$$w_1 = (\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m})^T$$

2. 在第k轮迭代中，使用具有权重w_k的样本训练基本评估器$f_k(x)$。
3. 使用基本学习器$f_k(x)$预测样本输出值$\hat{y}$。
4. 计算含有权重的错误率：

$$\epsilon_k = w_k \cdot (y \neq \hat{y})$$

5. 计算第k轮的基本评估器$f_k(x)$的权重系数：

$$\alpha_k = 0.5 * log\frac{1-\epsilon_k}{\epsilon_k}$$

- $\alpha_k > 0$。

6. 更新样本权重：

$$w_k = w_k * e^{-\alpha_k * y * \hat{y}}$$

- 对于预测正确的样本，降低样本权重值，否则提升样本权重值。

7. 对权重w_k进行归一化，使其和为1：

$$w_k = \frac{w_j}{\sum_i w_i}$$

对于更新后的w_k，就会成为下一轮（第$k+1$轮）迭代的权重w_{k+1}

8. 构建基本评估器的线性组合：

$$F_k(x) = \sum_{i=1}^{k} \alpha_i f_i(x)$$

- 该结果为迭代到第k论的预测结果。

9. 重复步骤2 ~ 8，共n次，获得最终的评估器：

$$\hat{y} = sign(F_n(x)) = sign(\sum_{i=1}^{n} \alpha_i f_i(x))$$

AdaBoost示例

假设给定的数据集，如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1

我们假设基本评估器使用决策树桩（深度为1），并且不存度度量标准使用信息熵。

第1轮

在初始状态，所有样本的权值w相同，且和为1。

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

训练评估器

分割点可以为2.5，5.5与8.5。

$$f_1(x) = \begin{cases} 1 & x \le 2.5 \\ -1 & x > 2.5 \end{cases}$$

预测输出值

使用训练好的模型对样本进行预测：

X	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
y	1	1	1	-1	-1	-1	1	1	1	-1
w_1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1

计算错误率

$$\epsilon_1 = w_1 \cdot (y \neq \hat{y}) = 0.3$$

计算权重系数

$$\alpha_1 = 0.5 * log\frac{1-\epsilon_1}{\epsilon_1} \approx 0.424$$

更新权重

$$w_1 = w_1 * e^{-\alpha_1 * y * \hat{y}}$$

如果预测正确，则y与$\hat{y}$符号相同，二者的乘积为正，否则，二者的乘积为负。而α的值大于0，因此，预测正确时，权重降低，预测错误时，权重提高。
因此，降低后的权重为：

$$0.1 * e^{-\alpha_1} \approx 0.065$$

提高后的权重为：

$$0.1 * e^{\alpha_1} \approx 0.153$$

更新后的结果如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_1	0.065	0.065	0.065	0.065	0.065	0.065	0.153	0.153	0.153	0.065
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1

权重归一化

$$w_1 = \frac{w_j}{\sum_j w_j}$$
$$\sum_j w_j = 7 * 0.065 + 3 * 0.153 = 0.914$$

因此，归一化的结果为：
预测正确的样本：$0.065/0.914 \approx 0.071$
预测错误的样本：$0.153/0.914 \approx 0.167$

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_1	0.071	0.071	0.071	0.071	0.071	0.071	0.167	0.167	0.167	0.071
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1

线性组合

经过第1轮后，学习器的线性组合为：

$$F_1(x) = 0.424 * f_1(x)$$

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_1	0.071	0.071	0.071	0.071	0.071	0.071	0.167	0.167	0.167	0.071
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1
$sign(F_1(x))$	1	1	1	-1	-1	-1	-1	-1	-1	-1

第2轮

在第2轮初始时，数据如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_2	0.071	0.071	0.071	0.071	0.071	0.071	0.167	0.167	0.167	0.071

训练评估器

$prop_1(D_p) = 0.071 * 3 + 0.167 * 3 = 0.714$
$prop_{-1}(D_p) = 1 - 0.714 = 0.286$
$I_H(D_p) = -(0.714 * log_2 0.714 + 0.286 * log_2 0.286) = 0.863$
$I_H(D_{x<=2.5}) = 0$
$prop_1(x > 2.5) = (0.167 * 3)/(0.071 * 4 + 0.167 * 3) = 0.638$
$prop_{-1}(x > 2.5) = 1 - 0.638 = 0.362$
$I_H(D_{x>2.5}) = -(0.638 * log_2 0.638 + 0.362 * log_2 0.362) = 0.944$
$prop(x <= 2.5) = 0.071 * 3 = 0.213$
$prop(x > 2.5) = 1 - 0.213 = 0.787$
$IG_H(x = 2.5) = 0.863 - 0.213 * 0 - 0.787 * 0.944 = 0.120$

$I_H(D_{x<=5.5}) = 1$
$prop_1((x > 5.5) = (0.167 * 3)/(0.071 * 1 + 0.167 * 3) = 0.876$
$prop_{-1}(x > 5.5) = 1 - 0.876 = 0.124$
$I_H(D_{x>5.5}) = -(0.876 * log_2 0.876 + 0.124 * log_2 0.124) = 0.541$
$prop(x <= 5.5) = 0.071 * 6 = 0.426$
$prop(x > 5.5) = 1 - 0.213 = 0.574$
$IG_H(x = 5.5) = 0.863 - 0.426 * 1 - 0.574 * 0.541 = 0.126$

$prop_1(x <= 8.5) = (0.071 * 3 + 0.167 * 3)/(0.071 * 6 + 0.167 * 3) = 0.770$
$prop_{-1}(x <= 8.5) = 1 - 0.770 = 0.230$
$I_H(D_{x<=8.5}) = -(0.770 * log_2 0.770 + 0.230 * log_2 0.230) = 0.778$
$I_H(D_{x>8.5}) = 0$
$prop(x <= 8.5) = 0.071 * 6 + 0.167 * 3 = 0.927$
$prop(x > 8.5) = 1 - 0.927 = 0.073$
$IG_H(x = 8.5) = 0.863 - 0.927 * 0.770 - 0.073 * 0 = 0.149$

$$f_2(x) = \begin{cases} 1 & x \leq 8.5 \\ -1 & x > 8.5 \end{cases}$$

预测输出值

使用训练好的学习器对样本进行预测：

X	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
y	1	1	1	-1	-1	-1	1	1	1	-1
w_2	0.071	0.071	0.071	0.071	0.071	0.071	0.167	0.167	0.167	0.071
$f_2(x)$	1	1	1	1	1	1	1	1	1	-1

计算错误率

$$\epsilon_2 = w_2 \cdot (y \neq \hat{y}) = 0.071 * 3 = 0.213$$

计算权重系数

$$\alpha_2 = 0.5 * log\frac{1-\epsilon_2}{\epsilon_2} \approx 0.653$$

更新权重

$$w_2 = w_2 * e^{-\alpha_2 * y * \hat{y}}$$

更新后的结果如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_2	0.037	0.037	0.037	0.136	0.136	0.136	0.087	0.087	0.087	0.037
$f_2(x)$	1	1	1	1	1	1	1	1	1	-1

权重归一化

$$w_2 = \frac{w_j}{\sum_j w_j}$$
$$\sum_j w_j = 0.037 * 4 + 0.136 * 3 + 0.087 * 3 = 0.818$$

因此，归一化的结果为：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_2	0.045	0.045	0.045	0.167	0.167	0.167	0.106	0.106	0.106	0.045
$f_2(x)$	1	1	1	1	1	1	1	1	1	-1

线性组合

经过第2轮后，学习器的线性组合为：

$$f_2(x) = 0.424 * f_1(x) + 0.653 * f_2(x)$$

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_2	0.045	0.045	0.045	0.167	0.167	0.167	0.106	0.106	0.106	0.045
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1
$f_2(x)$	1	1	1	1	1	1	1	1	1	-1
$sign(F_2(x))$	1	1	1	1	1	1	1	1	1	-1

第3轮

在第3轮初始时，数据如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_3	0.045	0.045	0.045	0.167	0.167	0.167	0.106	0.106	0.106	0.045

训练学习器

$prop_1(D_p) = 0.045 * 3 + 0.106 * 3 = 0.453$
$prop_{-1}(D_p) = 1 - 0.453 = 0.547$
$I_H(D_p) = -(0.453 * log_2 0.453 + 0.547 * log_2 0.547) = 0.994$
$I_H(D_{x<=2.5}) = 0$
$prop_1((x > 2.5) = (0.106 * 3)/(0.106 * 3 + 0.167 * 3 + 0.045) = 0.368$
$prop_{-1}(x > 2.5) = 1 - 0.368 = 0.632$
$I_H(D_{x>2.5}) = -(0.368 * log_2 0.368 + 0.632 * log_2 0.632) = 0.949$
$prop(x <= 2.5) = 0.045 * 3 = 0.135$
$prop(x > 2.5) = 1 - 0.135 = 0.865$
$IG_H(x = 2.5) = 0.994 - 0.135 * 0 - 0.865 * 0.949 = 0.173$

$prop_1(x <= 5.5) = (0.045 * 3)/(0.045 * 3 + 0.167 * 3) = 0.212$
$prop_{-1}(x <= 5.5) = 1 - 0.212 = 0.788$
$I_H(D_{x<=5.5}) = -(0.212 * log_2 0.212 + 0.788 * log_2 0.788) = 0.745$
$prop_1((x > 5.5) = (0.106 * 3)/(0.106 * 3 + 0.045 * 1) = 0.876$
$prop_{-1}(x > 5.5) = 1 - 0.876 = 0.124$
$I_H(D_{x>5.5}) = -(0.876 * log_2 0.876 + 0.124 * log_2 0.124) = 0.541$
$prop(x <= 5.5) = 0.045 * 3 + 0.167 * 3 = 0.636$
$prop(x > 5.5) = 1 - 0.636 = 0.364$
$IG_H(x = 5.5) = 0.994 - 0.636 * 0.745 - 0.364 * 0.541 = 0.323$

$prop_1((x <= 8.5) = (0.045 * 3 + 0.106 * 3)/(0.045 * 3 + 0.106 * 3 + 0.167 * 3) = 0.475$
$prop_{-1}(x <= 8.5) = 1 - 0.475 = 0.525$
$I_H(D_{x<=8.5}) = -(0.475 * log_2 0.475 + 0.525 * log_2 0.525) = 0.998$
$I_H(D_{x>8.5}) = 0$
$prop(x <= 8.5) = 0.045 * 3 + 0.106 * 3 + 0.167 * 3 = 0.954$
$prop(x > 8.5) = 1 - 0.954 = 0.046$
$IG_H(x = 8.5) = 0.994 - 0.954 * 0.998 - 0.046 * 0 = 0.042$

$$f_3(x) = \begin{cases} 1 & x > 5.5 \\ -1 & x \leq 5.5 \end{cases}$$

预测输出值

使用训练好的学习器对样本进行预测：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_3	0.045	0.045	0.045	0.167	0.167	0.167	0.106	0.106	0.106	0.045
$f_3(x)$	-1	-1	-1	-1	-1	-1	1	1	1	1

计算错误率

$$\epsilon_3 = w_3 \cdot (y \neq \hat{y}) = 0.045 * 4 = 0.180$$

计算权重系数

$$\alpha_3 = 0.5 * log\frac{1-\epsilon_3}{\epsilon_3} \approx 0.758$$

更新权重

$$w_3 = w_3 * e^{-\alpha_3 * y * \hat{y}}$$

更新后的结果如下：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_3	0.152	0.152	0.152	0.033	0.033	0.033	0.078	0.078	0.078	0.152
$f_3(x)$	-1	-1	-1	-1	-1	-1	1	1	1	1

权重归一化

$$w_3 = \frac{w_j}{\sum_j w_j}$$

$$\sum_j w_j = 0.152 * 4 + 0.033 * 3 + 0.078 * 3 = 0.941$$

因此，归一化的结果为：

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_3	0.152	0.152	0.152	0.033	0.033	0.033	0.078	0.078	0.078	0.152
$f_3(x)$	-1	-1	-1	-1	-1	-1	1	1	1	1

线性组合

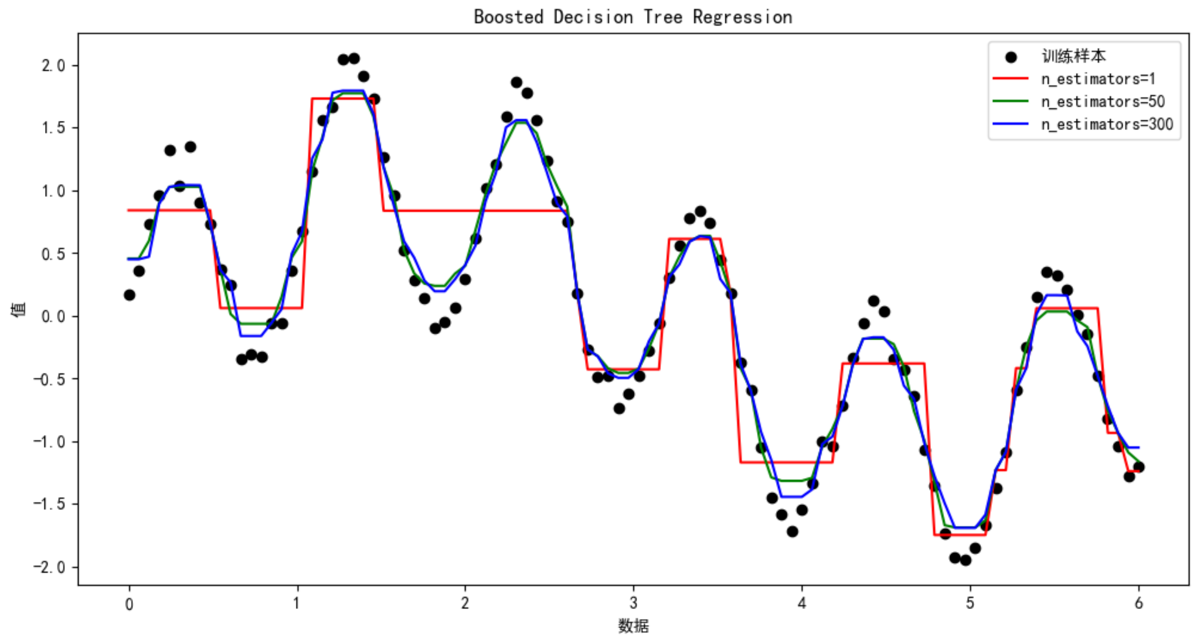
经过第3轮后，学习器的线性组合为：

$$F_3(x) = 0.424 * f_1(x) + 0.653 * f_2(x) + 0.758 * f_3(x)$$

X	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1
w_3	0.152	0.152	0.152	0.033	0.033	0.033	0.078	0.078	0.078	0.152
$f_1(x)$	1	1	1	-1	-1	-1	-1	-1	-1	-1
$f_2(x)$	1	1	1	1	1	1	1	1	1	-1
$f_3(x)$	-1	-1	-1	-1	-1	-1	1	1	1	1
$sign(F_3(x))$	1	1	1	-1	-1	-1	1	1	1	-1

```python
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

from sklearn.tree import DecisionTreeRegressor
# AdaBoostRegressor sklearn中提供的关于AdaBoost回归的模型。
from sklearn.ensemble import AdaBoostRegressor

rng = np.random.RandomState(1)
x = np.linspace(0, 6, 100)
y = np.sin(x) + np.sin(6 * x) + rng.normal(0, 0.1, x.shape[0])
X = x[:, np.newaxis]

regr_1 = DecisionTreeRegressor(max_depth=4)

regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                          n_estimators=50, random_state=rng)
regr_3 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                          n_estimators=300, random_state=rng)

regr_1.fit(X, y)
regr_2.fit(X, y)
regr_3.fit(X, y)
y_1 = regr_1.predict(X)
y_2 = regr_2.predict(X)
y_3 = regr_3.predict(X)

plt.figure(figsize=(12, 6))
plt.scatter(X, y, c="k", label="训练样本")
plt.plot(X, y_1, c="r", label="n_estimators=1")
plt.plot(X, y_2, c="g", label="n_estimators=50")
plt.plot(X, y_3, c="b", label="n_estimators=300")
plt.xlabel("数据")
plt.ylabel("值")
plt.title("Boosted Decision Tree Regression")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2050f8d6f80>
```



泰勒公式

泰勒公式用来将一个函数表示为一个无限级数的形式，这个级数的每一项都是函数在给定点处的导数。通过泰勒公式，可用于求解函数的近似值。泰勒公式的形式如下：

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

- $f^{(n)}(x_0)$：函数$f(x)$在$x = x_0$处的n阶导数。

例如，当函数$f(x)$在点x_0处可导时，我们可以将函数$f(x)$使用一阶泰勒在点x_0展开：

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

二阶泰勒在点x_0展开：

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + 0.5 * f''(x_0)(x - x_0)^2$$

在实际应用中，通常只需要取前几项来近似计算函数值即可。

泰勒展开式示例

我们以正弦函数为例，通过泰勒展开式可以将函数在某个点处进行多项式展开来。设 $f(x) = \sin(x)$，根据：

$$f^{(n)}(x) = \begin{cases} \sin(x), & n\%4 = 0 \\ \cos(x), & n\%4 = 1 \\ -\sin(x), & n\%4 = 2 \\ -\cos(x), & n\%4 = 3 \end{cases}$$

因此，函数$f(x)$在 $x = x_0$ 处展开，有：

$
f(x) = \sin(x) = \sum_{n=0}^\infty \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \
= \sin(x_0) + \cos(x_0)(x-x_0) - \frac{\sin(x_0)}{2!}(x-x_0)^2 \

- \frac{\cos(x_0)}{3!}(x-x_0)^3 + \frac{\sin(x_0)}{4!}(x-x_0)^4 + \cdots
  $

根据之前的总结，我们有：

$$
f^{(n)}(x) = \begin{cases} f(x) & = \sin(x) \\ f'(x) & = \cos(x) \\ f''(x) & = -\sin(x) \\ f'''(x) & = -\cos(x) \\ f^{(4)}(x) & = \sin(x) \\ f^{(5)}(x) & = \cos(x) \\ & \cdots \end{cases}
$$

因此，如果我们在$x_0 = 0$点处，对正弦函数进行泰勒展开，可以得出：

$$
f^{(n)}(x) = \begin{cases} sin(0) = 0 & n = 0, 2, 4, 6, \cdots \\ (-1)^{(n-1)/2} cos(0) = (-1)^{(n-1)/2} & n = 1, 3, 5, \cdots \end{cases}
$$

将以上结果带入，可得：

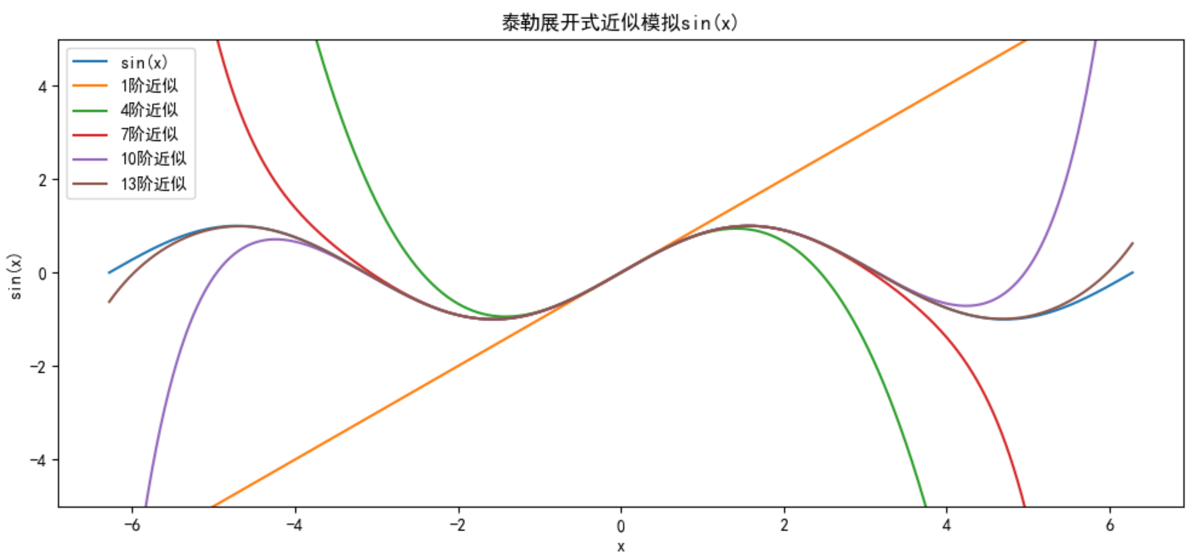
$$
\begin{aligned} \sin(x) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \\ &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \end{aligned}
$$

```
def taylor_sin(x, n):
```

```python
    """通过泰勒公式模拟正弦函数的近似值。

    Parameters
    ----------
    x : float
        自变量。
    n : int
        泰勒展开式的阶数。
    """
    value = 0
    for i in range(n + 1):
        # 正弦函数，偶数
        if i % 2 == 0:
            continue
        v = (-1) ** ((i - 1) / 2) * x ** i / np.math.factorial(i)
        value += v
    return value

plt.figure(figsize=(12, 5))
# 设置定义域，指定-2pi ~ 2pi，两个周期。
x = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
y = np.sin(x)
plt.plot(x, y, label="sin(x)")
# 使用不同阶数的泰勒展开式来近似表示sin(x)。
for n in range(1, 14, 3):
    y_approx = [taylor_sin(k, n) for k in x]
```

```python
    plt.plot(x, y_approx, label=f"{n}阶近似")

plt.xlabel("x")
plt.ylabel("sin(x)")
plt.title("泰勒展开式近似模拟sin(x)")
plt.ylim(-5, 5)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x20510389150>
```



GBDT算法

算法介绍

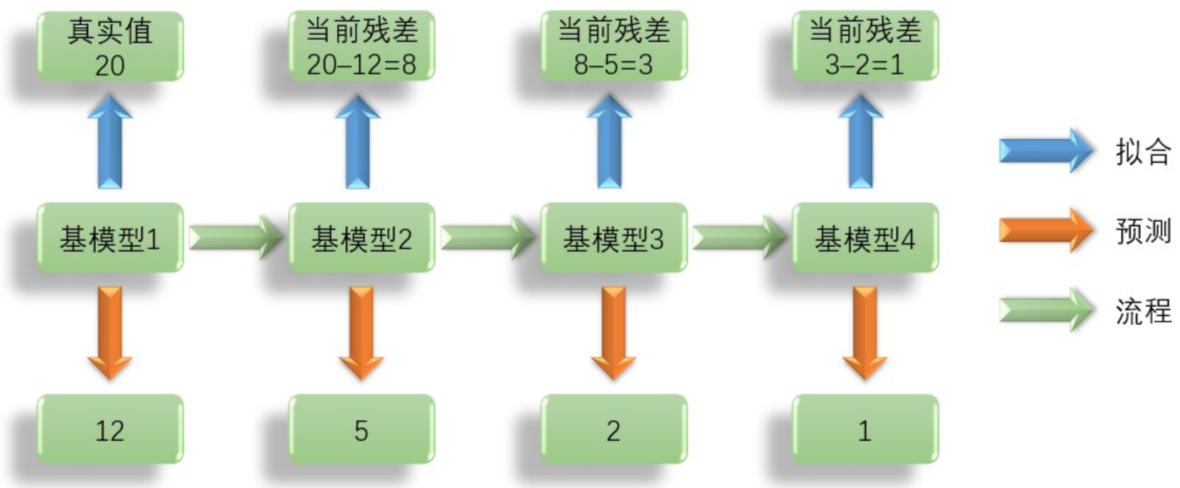
GBDT（Gradient Boosted Decision Trees），是指梯度提升决策树，使用提升方法训练优化，基模型固定使用CART回归树，可以用于分类与回归任务。

模型训练

关于模型的训练，说明如下：

- GBDT(Boosted)使用加法模型，其使用M个基模型（弱模型）的预测结果之和，作为最终的预测结果。
- 在训练时，对于M个基模型，采用从前向后进行迭代，第m个模型去拟合当前阶段的残差。
    - 实际上，拟合的是损失函数的负梯度。
- 经过M轮的迭代，使得预测结果逐渐接近真实值（目标值）。

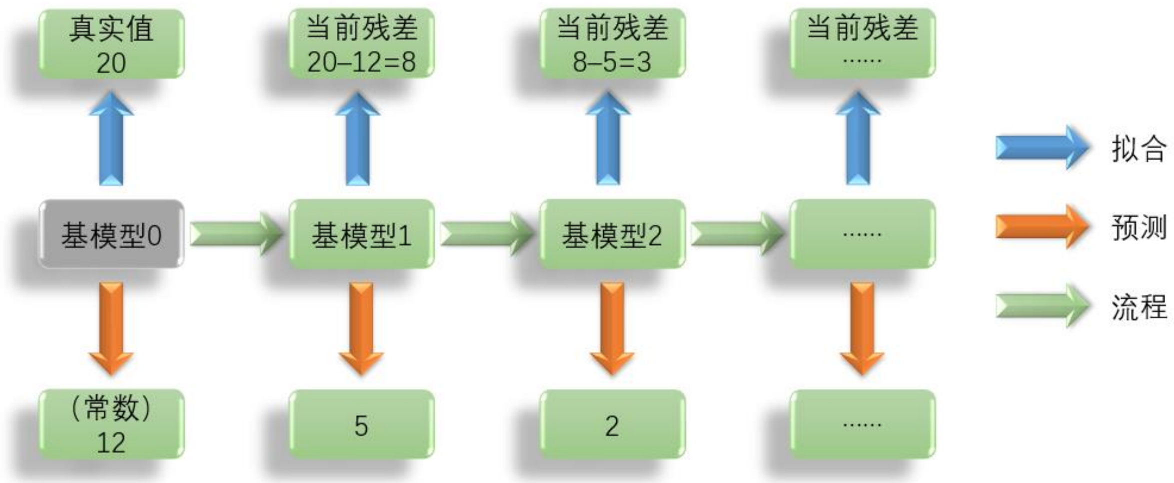
流程图示



问题思考

以上模型是否存在行为不一致之处？是否可以改进？

模型改进



形象理解

当前200克，需要加到500克。残差（300克）	增加200克，还差100克。	增加80克，还差20克。	增加20克。




数学公式推导

回归任务

我们首先以回归任务为例，来讲解GBDT的数学公式推导，对于分类任务，与回归任务是类似的。

模型预测

之前我们提到，GBDT使用加法模型实现预测，因此，对于给定样本x_i，模型的预测结果为：

$$\hat{y}_i = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i)$$

- M：基模型的数量。
- h_m：第m个基模型。
- $h_m(x_i)$：对于样本x_i，第m个基模型的预测结果。
- $F_m(x_i)$：对于样本x_i，前m个基模型的累加预测结果。
  - 该结果即模型迭代到第m轮的预测结果。
  - 如果基模型共有M个评估器，则F_M为最终的预测结果。
- $\hat{y}_i$：第i个样本的最终预测值。

GBDT采用迭代的方式，从前到后实现训练，对于第m次迭代，前m个模型的输出结果，可以表示为：

$$F_m(x) = \sum_{j=1}^{m} h_j(x)$$
$$= \sum_{j=1}^{m-1} h_j(x) + h_m(x)$$
$$= F_{m-1}(x) + h_m(x)$$

例如：

$$F_1(x) = F_0(x) + h_1(x)$$
$$F_2(x) = F_1(x) + h_2(x)$$
$$\cdots$$

损失函数

也就是说，当执行第m次迭代（迭代到第m个模型）时，我们的目的是期望在预测结果中加入$h_m(x)$，使得损失值最小。

$$h_m = \arg\min_h L_m = \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$$

- L_m：第m次迭代时的损失函数，计算所有样本损失。
- l：损失函数，计算单个样本的损失。
- n：样本量（样本个数）。
- $\arg\min_h$：使得函数值最小的h。

损失函数一阶泰勒展开

根据之前的泰勒公式，这里，我们将损失函数：

$$l(y_i, F_{m-1}(x_i) + h_m(x_i))$$

在点$F_{m-1}(x_i)$处进行一阶泰勒展开：

$l(y_i, F{m-1}(x_i) + h_m(x_i)) \approx
l(y_i, F{m-1}(x_i))

- h_m(x_i)
  \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]
  {F=F{m - 1}}$
- $[\ldots]$：损失函数l对第其2个参数求偏导，并在点$F_{m-1}(x_i)$求值的结果。

拟合

令上式中的偏导值为g_i，则:

$$
\begin{aligned}
h_m &\approx \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i)) + h(x_i)g_i \\
&\approx \arg\min_h \sum_{i=1}^{n} h(x_i)g_i
\end{aligned}
$$

对于上式来说，在第m次迭代，基模型$h_m(x_i)$拟合为样本的负梯度，能够使得函数值最小。当然，预测目标未必一定等于负梯度，也可以是负梯度的比例（Shrinkage）。

分类任务

对于分类任务，与回归的训练方式基本是相同的，只是在最后输出$F_M(x_i)$（连续值）的基础上，通过相关函数完成转换，过程与逻辑回归算法相似。

- 对于二分类任务，使用sigmoid函数。
- 对于多分类任务，使用softmax函数。

最终，模型预测类别，为概率最大的那个类别。

说明:

- GBDT使用的基模型，一律是回归决策树。即使应用于分类任务也是如此。

模型参数及优化

在sklearn中，GBDT几个常用的参数如下:

- n_estimators：基模型的数量。
- learning_rate：学习率（收缩）。用来缩减每个模型（树）的拟合程度。
    - $F_m(x) = F_{m-1}(x) + \eta * h_m(x)$

- subsample：在每次迭代训练单个基模型所使用的抽样比例。
  - 取值范围为$(0, 1]$
  - 抽样为无放回抽样。
- loss：损失函数。通常取默认值即可。
  - 对于回归，默认为平方和损失函数。
  - 对于分类，默认为对数损失函数。
- random_state：随机种子。
- warm_start：如果为True，表示在上一次拟合的基础上继续拟合，否则从头开始拟合。默认值为False。

决策树参数：

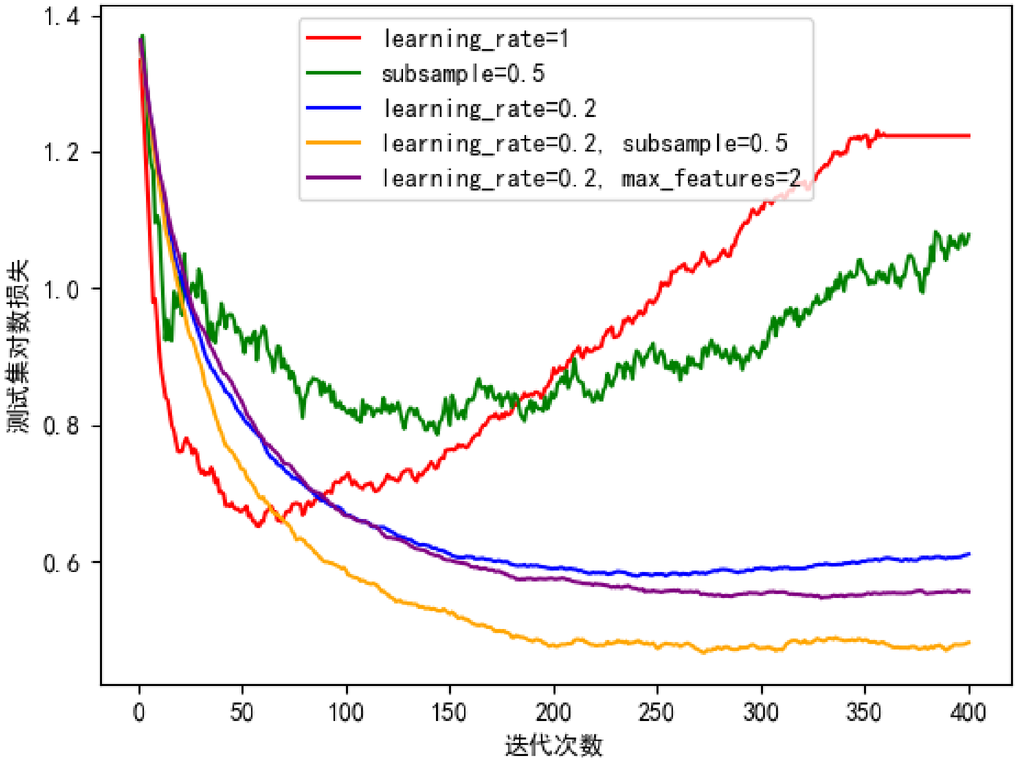
- max_depth：树的最大深度。
- max_features：最大的特征数量。
- min_samples_split：分裂的最小样本数量。
- min_samples_leaf：叶子包含的最小样本数量。
- max_leaf_nodes：最大叶子节点数量。

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_hastie_10_2
from sklearn.metrics import log_loss
from sklearn.model_selection import train_test_split


X, y = make_hastie_10_2(n_samples=4000, random_state=1)
```

```
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.8,
random_state=0)


params = [
    ("learning_rate=1", "r",
{"learning_rate": 1.0, "subsample":
1.0}),
    ("subsample=0.5", "g",
{"learning_rate": 1.0, "subsample":
0.5}),
    ("learning_rate=0.2", "b",
{"learning_rate": 0.2, "subsample":
1.0}),
    ("learning_rate=0.2, subsample=0.5",
"orange", {"learning_rate": 0.2,
"subsample": 0.5}),
    ("learning_rate=0.2, max_features=2",
"purple", {"learning_rate": 0.2,
"max_features": 2})
]
for label, color, p in params:
    gb =
GradientBoostingClassifier(n_estimators=4
00, max_depth=3, random_state=2, **p)
    gb.fit(X_train, y_train)
    test_deviance =
np.zeros(gb.n_estimators_)
```

```
    # 获取每一个阶段（每轮迭代）的预测结果，并对每轮预测结果计算对数损失。
    for i, y_proba in enumerate(gb.staged_predict_proba(X_test)):
        test_deviance[i] = 2 * log_loss(y_test, y_proba[:, 1])
    # 绘制每轮迭代的偏差。

 plt.plot(np.arange(test_deviance.shape[0]) + 1, test_deviance, color=color, label=label)
plt.legend()
plt.xlabel("迭代次数")
plt.ylabel("测试集对数损失")
```

```
Text(0, 0.5, '测试集对数损失')
```



learning_rate=1
subsample=0.5
learning_rate=0.2
learning_rate=0.2, subsample=0.5
learning_rate=0.2, max_features=2
测试集对数损失
迭代次数
1.4
1.2
1.0
0.8
0.6
0
50
100
150
200
250
300
350
400