机器学习 **Lecture 5**

集成学习

一、 集成方法

Defination 1 (验证)**.** 从若干模型中选出在验证集上效果最好的模型。

$$f(x) = \arg\min_t E_{\text{val}}(G_t(x))$$

Defination 2 (一致性投票)**.** 对所有模型的结果进行平均

$$f(x) = \frac{1}{m}\sum_{t=1}^{T} G_t(x)$$

$$f(x) = \text{sign}(\sum_{t=1}^{T} G_t(x))$$

Defination 3 (非一致性投票)**.** 对所有模型的结果进行加权平均

$$f(x) = \frac{\sum_{t=1}^{T}\alpha_t G_t(x)}{\sum_{t=1}^{T}\alpha_t}$$

$$f(x) = \text{sign}(\sum_{t=1}^{T}\alpha_t G_t(x))$$

Defination 4 (combine conditionally)**.** 根据一定条件结合的非一致性选票

$$f(x) = \frac{\sum_{t=1}^{T} q_t(x) G_t(x)}{\sum_{t=1}^{T} q_t(x)}$$

$$f(x) = \text{sign}(\sum_{t=1}^{T} q_t(x) G_t(x))$$

二、 集成学习有效的理论分析

f 为集成学习模型，G 为基学习器，f^* 为真实分布

$$\begin{aligned}
\text{avg}\{G_t - f^*\}^2 &= \text{avg}\{G_t^2\} - 2ff^* + f^2 \\
&= \text{avg}\{G_t^2\} - f^2 + (f - f^*)^2 \\
&= \text{avg}\{G_t^2\} - 2f^2 + f^2 + (f - f^*)^2 \\
&= \text{avg}\{(G_t - f)^2\} + (f - f^*)^2
\end{aligned}$$

三、 Adaboost

1. 思路

$$f(x) = \sum_{t=1}^{T} \alpha_t G_t(\mathbf{x})$$

其中，α_i 为弱学习器的权重，G 为弱分类器

Algorithm 1 Adaboost

Input: $D = \{(x_i, y_i)\}_{i=1}^m$; 基学习器算法 $\mathcal{L}$; 训练轮数 T

1: $\mathbf{D}_1(x) = (w_{11}, w_{12}, \cdots, w_{1n}) = 1/m$ ▷ 初始化样本权重

2: **for** $t = 1, 2, \cdots, T$ **do**

3: $G_t = \mathcal{L}(D, \mathcal{D}_t)$ ▷ 训练弱学习器

4: **Compute** ▷ 计算 $\mathcal{D}_t$ 分布下的分类误差率

$$\varepsilon_t = P_{x\sim\mathcal{D}_t}(G_t(x) \neq f(x)) = \sum_{G_t(x_i)\neq f(x_i)} w_{ti}$$

5: **Compute** ▷ 计算弱分类器权重

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$$

6: **Compute**

$$Z_t = \sum_{i=1}^{m} w_{t,i} \exp(-\alpha_t f(x_i) G(x_i))$$

7: **Update** $\mathcal{D}_{t+1}$

$$\mathcal{D}_{t+1,i} = \frac{w_{t,i}}{Z_t} \exp(-\alpha_t f(x_i) G(x_i))$$

8: **end for**

Output: $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \alpha_t G_t(x))$

Defination 5. 总权重值和为 *1*

$$\mathcal{D}_t = \{w_1, w_2, \cdots, w_m\}$$
$$\sum_{i=1}^{m} w_i = 1$$

Defination 6. ε 表示误分率，误分率等于分类错误的权重与总权重之比

$$\varepsilon_t = \sum_{G_t(x_i)\neq f(x_i)} w_{t,i}$$

Defination 7. 保证多样性 $\Leftrightarrow G_t$ 在权重 $\mathcal{D}_t$ 下的性能近似为随机猜

Defination 8. 弱分类器权重为 α_t,α_t 满足：当误分率为 *0.5*（随机猜）时，$\alpha_t = 0$；当误分率为 *0*（分类完全正确时）时,$\alpha_t \to +\infty$；当误分率为 *1*（分类完全错误时）时,$\alpha_t \to -\infty$

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\epsilon_t})$$

Defination 9. 权重 $\mathcal{D}_t$ 的更新满足如下规则：

$$\mathcal{D}_{t+1,i} = \frac{w_{t,i}}{Z_t}\exp(-\alpha_t f(x_i)G(x_i))$$

推导 1. 由 *Defination.7*知：

$$\sum_{G_{t+1}\neq f} w_i = \sum_{G_{t+1}=f} w_i$$

又因为 *Defination.5*, 所以

$$\sum_{G_{t+1}\neq f} w_i = \sum_{G_{t+1}=f} w_i = \frac{1}{2}$$

假设

$$w_{t+1,i} = \begin{cases} w_{t,i}\cdot u_t & G_t(x_i)\neq f(x_i) \\ w_{t,i}\cdot v_t & G_t(x_i) = f(x_i) \end{cases}$$

解得 $u_t = \frac{1}{2\varepsilon}$ $\quad v_t = \frac{1}{2(1-\varepsilon)}$ 进一步将假设简化为

$$\hat{w}_{t+1,i} = \begin{cases} w_{t,i}\cdot s_t & G_t(x_i)\neq f(x_i) \\ w_{t,i}/s_t & G_t(x_i) = f(x_i) \end{cases} \tag{1}$$

解得 $s_t = \sqrt{\frac{1-\varepsilon}{\varepsilon}} = \exp\alpha_t$
带入*1*得

$$\hat{w}_{t+1,i} = w_{t,i}\cdot\exp(\alpha_t\cdot\mathrm{sign}(\mathbb{I}(G_t(x_i)\neq f(x_i)) - 1/2) \tag{2}$$

又因为 *Adaboost* 为二分类问题，所以分类正确时 $G_t(x_i)$ 与 $f(x_i)$ 同号，错误时异号，所以 *(2)* 式可化简为：

$$\hat{w}_{t+1,i} = w_{t,i}\cdot\exp(-\alpha_t f(x_i)G_t(x_i))$$

对 $\hat{w}_t$ 进行归一化操作得

$$Z_t = \sum_{i=1}^{m} w_{t,i}\cdot\exp(-\alpha_t f(x_i)G_t(x_i))$$
$$w_{t+1,i} = \frac{w_{t,i}}{Z_t}\exp(-\alpha_t f(x_i)G(x_i))$$

Theorem 1. *Adaboost*⇔ 用前向分步算法对如下目标函数进行贪婪的学习

$$\min_{\alpha_t,G_t}\sum_{i=1}^{m}\exp\big(-y_i\sum_{t=1}^{T}\alpha_t G_t(x_i)\big)$$

推导 2. 对于每一步 t，由于采用前向分步算法，因此实际上是在对如下目标函数进行优化

$$E_T = \sum_{i=1}^{m} \exp(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)) \cdot \exp(-y_i \alpha_T G_T(x_i)) \tag{3}$$

注意到

$$\begin{aligned}
\hat{w}_T &= w_{T-1} \exp(-\alpha_{T-1} y_i G_{T-1}(x_i)) \\
&= w_1 \exp(\sum_{t=1}^{T-1} -\alpha_t y_i G_t(x_i)) \\
&= w_1 \exp(y_i \sum_{t=1}^{T-1} -\alpha_t G_t(x_i))
\end{aligned}$$

因此 (3) 式可化简为：

$$E_T = \sum_{i=1}^{m} \hat{w}_T \cdot \exp(-y_i \alpha_T G_T(x_i))$$

又因为

$$y_i G_t(x_i) = \begin{cases} 1 & y_i = G(x_i) \\ -1 & yi \neq G(x_i) \end{cases}$$

所以 (3) 式可表示为

$$E_T = \sum_{y_i = G(x_i)} \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(\alpha_T)$$

极小化基学习器 G_T

$$\begin{aligned}
&\arg\min_{G_T} \sum_{y_i = G(x_i)} \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(\alpha_T) \\
=&\arg\min_{G_T} \sum_{y_i = G(x_i)} \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(-\alpha_T) \\
&- \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(\alpha_T) \\
=&\arg\min_{G_T} \sum \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T (\exp(\alpha_T) - \exp(-\alpha_T)) \\
=&\arg\min_{G_T} (\exp(\alpha_T) - \exp(-\alpha_T)) \sum_{y_i \neq G(x_i)} \hat{w}_T \\
=&\arg\min_{G_T} \sum_{y_i \neq G(x_i)} \hat{w}_T
\end{aligned}$$

极小化 α_t

$$\frac{\partial E_T}{\alpha_T} = - \sum_{y_i = G(x_i)} \hat{w}_T \exp(-\alpha_T) + \sum_{y_i \neq G(x_i)} \hat{w}_T \exp(\alpha_T) = 0$$

$$\begin{aligned}
\exp 2\alpha_T &= \frac{\sum_{y_i = G(x_i)} \hat{w}_T}{\sum_{y_i \neq G(x_i)} \hat{w}_T} = \frac{1-\varepsilon}{\varepsilon} \\
\therefore \alpha_T &= \frac{1}{2} \ln(\frac{1-\varepsilon}{\varepsilon})
\end{aligned}$$

四、 梯度 boosting

$$F_T(x) = \sum_{t=1}^{T} \alpha_t G(x)$$

Algorithm 2 梯度 boosting

Input: $D = \{(x_i, y_i)\}_{i=1}^{m}$; 损失函数 $\ell(F(x), y)$; 训练轮数 T

for $t = 1, 2, \cdots, T$ **do**
  $r_{t,i} = -\left[\frac{\partial \ell(F(x_i), y_i)}{\partial F(x_i)}\right]_{F(x_i)=F_{t-1}(x_i)}$
  $f_t = \arg\min_f \sum_{i=1}^{m} (f_t(x_i) - r_{t,i})^2$
  $\alpha_t = \arg\min_\alpha \sum_{i=1}^{m} \ell(F_{t-1}(x_i) + \alpha G_t(x_i), y_i)$
end for

Output: $F_T(x)$

五、 代码实现

本次实验通过训练垂直于某一维度的超平面的弱学习器来作为基学习器。通过 adaboost 的思想进行集成学习。从图1中可以看出，集成学习后的模型通过 9 个超平面很好的区分除了"moon" 数据集，只有一个点预测错误。从图2中可以看出，即是对于更加混乱的数据，集成学习也可以很好的进行分类，至少能够过拟合。

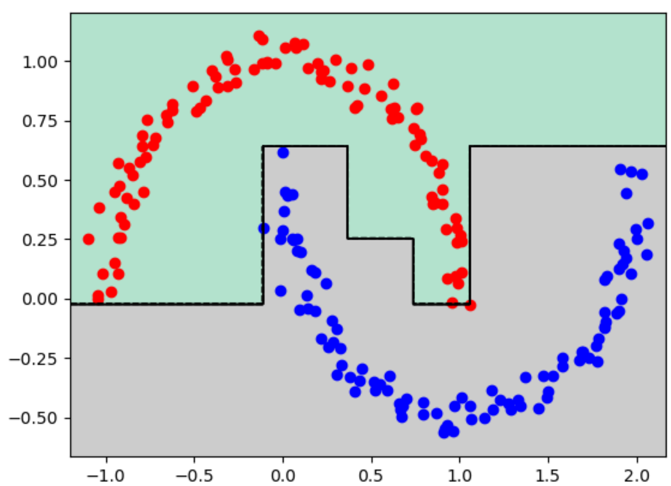

图 1: 利用线性基学习器对非线性数据进行分类

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

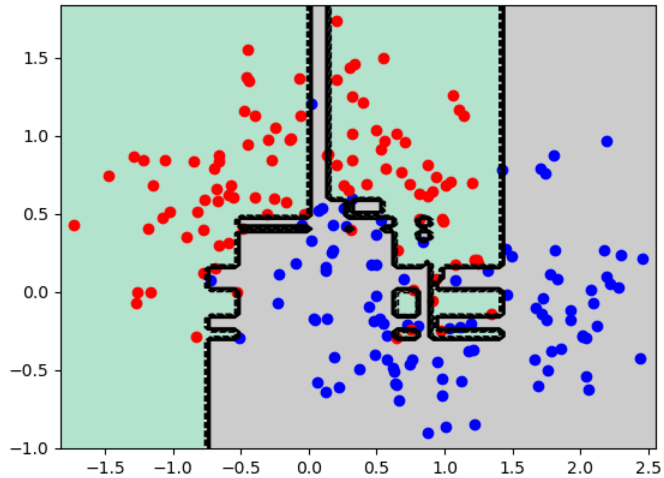

图 2: 利用线性基学习器对非线性数据进行分类

```python


class line_model():
    """
    基学习器，以垂直于某条坐标轴的超平面作为分类面
    对于数据中所有点在该坐标轴的值作为决策节点，对于每个决策节点有两种情况
    >=value的为正类或者 <value的为正类
    基学习器训练的时候接受权重参数，用于训练在D_t分布下的最优基学习器
    基学习器学得3个参数：dim(在哪个维度)，value(用什么值作为决策值)，sign(符号是>=还是<)
    同时会返回最小的误分类误差(0最优，1最差，理论上二分类问题应当小于0.5)
    接受的y应当为1和-1
    """

    def __init__(self, X, y):
        self.X = X
        self.y = y
        self.m = X.shape[0]
        self.n = X.shape[1]

    def train(self, w):
        res = []
        for i in range(self.n):
            for j in set(self.X[:, i]):
                X1 = self.X[:, i].copy()
                X1[self.X[:, i] >= j] = 1
                X1[self.X[:, i] < j] = -1
                X2 = self.X[:, i].copy()
                X2[self.X[:, i] < j] = 1
                X2[self.X[:, i] >= j] = -1

```

```
                eps1 = np.sum(w[np.where(X1 != self.y)])
                eps2 = np.sum(w[np.where(X2 != self.y)])

                if eps1 <= eps2:
                    res.append([eps1, i, j, 0])
                else:
                    res.append([eps2, i, j, 1])
        res = np.array(res)
        # print(res)
        res = res[np.argmin(res[:, 0])]
        self.res = res
        return res[0]

    def predict(self, x):
        """
        接受两种模式，一种是预测单个值，用于绘制决策边界
        一种是批量预测， 更新D_t时比较方便
        """
        dim = int(self.res[1])
        value = self.res[2]
        sign = self.res[3]
        if x.ndim == 1:
            if sign == 0:
                if x[dim] >= value:
                    return 1
                return -1
            if sign == 1:
                if x[dim] < value:
                    return 1
                return -1
        elif x.ndim == 2:
            if sign == 0:
                predict = np.zeros(self.m)
                predict[x[:, dim] >= value] = 1
                predict[x[:, dim] < value] = -1
            if sign == 1:
                predict = np.zeros(self.m)
                predict[x[:, dim] < value] = 1
                predict[x[:, dim] >= value] = -1
            return predict


class adaboost():
    """
    adaboost
    input : X, y, 训练轮数T, 基学习器base_model（应当为一个类，没有实例化）
    """

    def __init__(self, X, y, T, base_model):
        """
```

```
        G -- 基学习器
        w -- 数据集的参数权重，初始时为1/m
        alpha -- alpha_t的列表，表示每个基学习器的权重
        G_list -- 基学习器的列表，存放每个训练好的基学习器
        """
        self.X = X
        self.y = y
        self.T = T
        self.m = X.shape[0]
        self.n = X.shape[1]
        self.G = base_model
        self.w = np.ones(self.m) / self.m
        self.alpha = []
        self.G_list = []

    def train(self):
        for i in range(self.T):
            # 训练
            G = self.G(self.X, self.y)
            self.G_list.append(G)
            epsilon_t = G.train(self.w)
            print(epsilon_t)
            # 计算alpha
            alpha_t = 0.5 * np.log((1 - epsilon_t) / epsilon_t)
            self.alpha.append(alpha_t)
            # 计算Z_t(用于归一化)和更新w
            predict = G.predict(self.X)
            Z_t = np.sum(self.w * np.exp(-alpha_t * self.y * predict))
            self.w = self.w * np.exp(-alpha_t * self.y * predict) / Z_t

    def predict(self, x):
        """
        预测 加权投票，大于等于0为正类，小于0为负类
        """
        res = []
        for G in self.G_list:
            y = G.predict(x)
            res.append(y)
        res = np.array(res) * np.array(self.alpha)
        if np.sum(res) >= 0:
            return 1
        else:
            return -1

    def plot_decision_boundaries(self, resolustion=1000):

        # 取这个用来画网格的
        mins = self.X.min(axis=0) - 0.1
        maxs = self.X.max(axis=0) + 0.1
        xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolustion),
```

```
                          np.linspace(mins[1], maxs[1], resolustion))
        grid = np.c_[xx.ravel(), yy.ravel()]
        predict = []
        for i in grid:
            predict.append(self.predict(i))
        predict = np.array(predict)
        predict = predict.reshape(xx.shape)
        # print(predict[:,0])
        plt.contourf(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                     cmap='Pastel2')

        plt.contour(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                    linewidths=1, colors='k')
        plt.scatter(self.X[self.y == 1, 0], self.X[self.y == 1, 1], c='b')
        plt.scatter(self.X[self.y == -1, 0], self.X[self.y == -1, 1], c='r')
        plt.show()
```