第5章

AdaBoost

5.1 AdaBoost 算法原理

对于分类问题，如果任意一个分类器解决不了的问题，那就多个分类器都试试。分类器一多，怎么整理结果？走类似参数拟合的路线，用权重来平衡不同分类器，用误差函数来帮助回归，简单明了——AdaBoost。

5.1.1 算法引入

在进行数据分析时，如何探索出性能优良的分类器一直是科学家执着的追求。通常情况下想要找到一个强分类器总是比较困难的，但想要获得一个弱分类器却相对容易。因此，我们提出这样一个问题，能否通过这些容易实现的弱分类器，经过一定组合后构建一个强分类器？AdaBoost 算法可解决该问题。AdaBoost 属于 Boosting 的一种算法，具有“自适应”功能。该算法通过不断迭代并根据每次迭代的误差率赋予当前基础分类器权重，同时，自动调整样本权重，最后把所有的分类器通过线性整合，形成一个性能优良的强分类器。AdaBoost 提供一种框架，该框架可以将各种弱分类器组合起来，并且该算法相对简单，不需要进行特征的筛选，难出现过

拟合。AdaBoost 框架如图 5-1 所示。

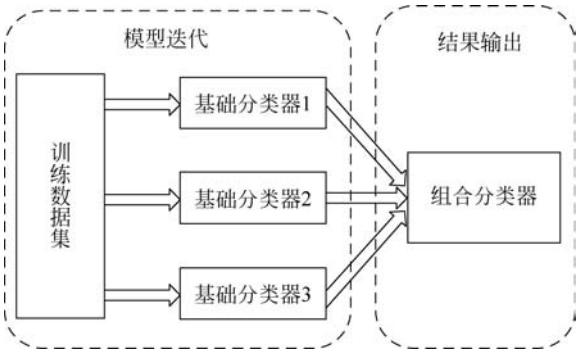

图 5-1　AdaBoost 框架图

5.1.2　科学问题

1. 相关定义

我们首先定义以下几个基本概念。

定义 1：基础分类器 $H(x)$

AdaBoost 只是提供了一种用于将弱分类器通过线性整合，以获得强分类器的框架。具体的基础分类器可以是任意分类器，在分类时其误差率应满足 $0<\varepsilon_t<\frac{1}{2}$。

定义 2：误差率 ε_t

误差率主要是用来衡量基础分类器在此次分类中的加权错误率，每次选择误差率最低的分类器加入到最终线性分类器中。

定义 3：分类器权重 α_t

AdaBoost 通过线性加和的方式得到最终分类器，α_t 表示每一个基础分类器在最终分类器中的权值，数字 t 表示第 t 轮迭代次数。

定义 4：样本权重 $u_t=(u_{t,1},u_{t,2},\cdots,u_{t,N})$，$N$ 表示样本总数

AdaBoost 每一次迭代都会更新样本集合 D 的权重，u_t 代表的是第 $t-1$ 迭代后所计算出的样本权重。

2. 问题定义

在给定数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$以及基础分类器 $H(x)$的情况下，AdaBoost 算法拟解决以下问题：

(1) 如何衡量分类器在 $H_t(x)$每一轮迭代中的权重 α_t？其中 $t=\{1,2,\cdots,T\}$，表示迭代次数。AdaBoost 算法通过计算误差率 ε_t，并通过公式 $\alpha_t=\frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$计算每一轮的权重。

(2) 如何在迭代过程中更新样本集的权重？为了使得每次迭代后的弱分类器更精确，并且使之前误分类样本在下轮迭代中所占比重更大，因此 AdaBoost 算法根据分类器分类结果对样本的权重进行更新，权重更新公式为

$$u_{t+1,i}=\frac{u_{t,i}}{Z_t}\exp(-\alpha_t y_i H_t(x_i))$$

5.1.3 算法流程

1. 算法步骤

该部分给出 AdaBoost 算法的步骤：

(1) 对给定的二分类数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}(x\in X,y\in Y)$，首先初始化每个样本的权重 $u_{1,i}=\left[\frac{1}{N},\frac{1}{N},\cdots,\frac{1}{N}\right](i=1,2,\cdots,N)$，$N$ 是样本数量，所有样本在初始阶段同等重要。

(2) AdaBoost 算法通过多次迭代生成多个基础分类器，在每次迭代过程中以最小误差率 ε_t 选择当前最优的基础分类器，误差率越小表示基础分类器对数据样本预测结果越准确，计算误差率 ε_t 的公式为：

$$\varepsilon_t=\sum_{i=1}^{N}u_{t,i}I(H_t(x_i)\neq y_i) \tag{5-1}$$

(3) AdaBoost 算法采用“加权多数表决”方法组合多个基础分类器。算法根据误差率 ε_t 为基础分类器 $H_t(x)$赋予权重 α_t，误差率越小，则分类器权重越大，即该基础

分类器在最终分类器中作用越大，计算分类器的权重 α_t 的公式为：

$$\alpha_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t} \tag{5-2}$$

（4）AdaBoost 算法在生成下一个基础分类器之前将放大误分类样本权重并缩小正确分类样本权重，这使得下一轮迭代更关注误分类样本，以期待在下次生成基础分类器过程中正确分类误分类样本，样本权重更新公式为：

$$u_{t+1,i} = \frac{u_{t,i}}{Z_t}\exp(-\alpha_t y_i H_t(x_i)) \quad (i=1,2,\cdots,N) \tag{5-3}$$

式(5-3)中 $t=1,2,\cdots,T$，Z_t 是归一化因子，Z_t 计算公式如下：

$$Z_t = \sum_{i=1}^{N} u_{t,i}\exp(-\alpha_t y_i H_t(x_i)) \tag{5-4}$$

（5）若多个基础分类器经线性组合后能全部准确预测数据样本类别或者迭代次数超过预定义次数，则迭代终止并形成最终线性分类器：

$$f(x) = \sum_{t=1}^{T}\alpha_t H_t(x) \tag{5-5}$$

把该分类器二值化，得到最终的分类器。

（6）最开始提出的 AdaBoost 算法用于解决二分类问题，$f(x)$经二值化后得到最终分类器，形式化如下：

$$H(x) = \mathrm{sign}(f(x)) = \mathrm{sign}\Big(\sum_{t=1}^{T}\alpha_t H_t(x)\Big) \tag{5-6}$$

2. AdaBoost 理论推导

AdaBoost 是一个线性整合模型，假设前 $t-1$ 轮迭代所产生的分类器已知，即：

$$f_{t-1}(x) = f_{t-2}(x) + \alpha_{t-1}H_{t-1}(x) = \alpha_1 H_1(x) + \cdots + \alpha_{t-1}H_{t-1}(x) \tag{5-7}$$

下一轮迭代目标是生成在当前数据样本权重下损失函数最小的基础分类器，AdaBoost 损失函数为指数函数，即：

$$(\alpha_t, H_t(x)) = \underset{\alpha,H}{\mathrm{argmin}}\sum_{i=1}^{N}\exp[-y_i(f_{t-1}(x_i) + \alpha H_t(x_i))] \tag{5-8}$$

式(5-8)又可以表示为：

$$(\alpha_t, H_t(x)) = \arg\min_{\alpha, H} \sum_{i=1}^{N} \bar{u}_{t,i} \exp[-y_i \alpha H_t(x_i)] \tag{5-9}$$

其中，通过式(5-3)可得到 $\bar{u}_{ti} = \exp[-y_i f_{t-1}(x_i)]$，此时可以看出 $\bar{u}_{ti}$ 与所要求的损失函数没有依赖，与上式的最小化过程没有关系。因此上式求解分为两步：

(1) 求解最优的分类器 $H_t^*(x_i)$，对任意的 $\alpha > 0$，对式(5-9)求解最小值，得到：

$$H_t^*(x_i) = \arg\min_{H} \sum_{i=1}^{N} \bar{u}_{t,i} I(y_i \neq H(x_i)) \tag{5-10}$$

损失函数(5-9)可表示为：

$$\begin{aligned} \sum_{i=1}^{N} \bar{u}_{t,i} \exp[-y_i \alpha H_t(x_i)] &= \sum_{y_i = H_t(x_i)}^{N} \bar{u}_{t,i} \exp[-\alpha] + \sum_{y_i \neq H_t(x_i)}^{N} \bar{u}_{t,i} \exp[\alpha] \\ &= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} \bar{u}_{t,i} I(y_i \neq H_t(x_i)) + e^{-\alpha} \sum_{i=1}^{N} \bar{u}_{t,i} \end{aligned} \tag{5-11}$$

将之前所求的 $H_t^*(x_i)$ 代入式(5-11)中，对 α 求偏导并使偏导为 0，可得到此时 α_t^* 的求解公式(5-2)。其中，ε_t 是分类器的误差率，此时 ε_t 的表示形式如下：

$$\varepsilon_t = \frac{\sum_{i=1}^{N} \bar{u}_{t,i} I(y_i \neq H_t(x_i))}{\sum_{i=1}^{N} \bar{u}_{t,i}} = \sum_{i=1}^{N} \bar{u}_{t,i} I(y_i \neq H_t(x_i)) \tag{5-12}$$

(2) 对于每一轮的权值更新，由 $f_t(x) = f_{t-1}(x) + \alpha_t H_t(x)$ 且 $\bar{u}_{ti} = \exp[-y_i f_{t-1}(x_i)]$ 可得权值更新公式：

$$u_{t+1,i} = \bar{u}_{t,i} \exp(-\alpha_t y_i H_t(x_i)) \quad (i = 1, 2, \cdots, N) \tag{5-13}$$

5.1.4 算法描述

下面给出 AdaBoost 算法的伪代码：

算法 5-1 AdaBoost 算法

输入：训练数据集 $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$，$(x \in X, y \in Y)$，以及基础分类器 $H_t(x_i)$，最低错误率 ξ；

初始化样本数据集权值 $u_{1,i}=\left[\frac{1}{N},\frac{1}{N},\cdots,\frac{1}{N}\right]$ $(i=1,2,\cdots,N)$；
for $t=1,2,\cdots,T$ **do**
　寻找当前权重下误差率最小的分类器，误差率 ε_t 通过式(5-1)计算得出；
　if $\varepsilon_t>0.5$
　　continue；
　else
　　通过误差率 ε_t 计算出该分类器的权重 α_t，权重通过式(5-2)计算得出；
　　通过式(5-3)和上一轮样本的权重 u_{t-1n} 更新当前样本集权重；
　　计算当前分类器 $f(x)=\sum_{t=1}^{T}\alpha_t H_t(x)$ 的错误率 ε；
　　if $\varepsilon\leqslant\xi$ **then**
　　　返回当前分类器 $f(x)$；
　　break；
　end if
end if
end for
输出最终的分类 $f(x)=\sum_{t=1}^{T}\alpha_t H_t(x)$；

5.1.5 补充说明

1. 基础分类器的误差率应该小于 0.5

AdaBoost 在选择基础分类器时，需要注意到基础分类器的误差率应小于 0.5。这一点通过权重迭代式(5-2)可以看出，当分类器的误差率为 0.5 时，分类器的权重为 0。这与我们的日常感知也是趋于一致的，因为当一个分类器的误差率为 0.5 时，相当于随机猜测的效果，也就是该分类器在此次分类过程中不起作用。因此在最终得到的线性分类器 $f_t(x)$里，该分类器权重为 0。同样，当基础分类器的误差率接近零时，该分

类器经过 $f_t(x)$计算权重变得很大,即该分类器在全部分类器中所占比重大。

2. AdaBoost 算法误差分析

本节将对 AdaBoost 算法误差进行分析,首先给出该算法误差分析的结论:

$$\frac{1}{N}\sum_{i}^{N}\{H_t(x)\neq y_i\}\leqslant\frac{1}{N}\sum_{i=1}^{N}\{\exp(-y_i*f(x_i))\}=\prod_{t}Z_t$$

权值更新的式(5-3)可以转换为如下形式:

$$u_{t,i}\exp(-\alpha_t y_i H_t(x_i))=Z_t u_{t+1,i} \tag{5-14}$$

又因为当 $H_t(x)\neq y_i$ 时,$y_i*f(x_i)<0$,因此得出 $\exp(-y_i*f(x_i))\geqslant 1$,此时 $\frac{1}{N}\sum_{i=1}^{N}\{H_t(x)\neq y_i\}\leqslant\frac{1}{N}\sum_{i=1}^{N}\{\exp(-y_i*f(x_i))\}$ 成立。

另外,对于 $\frac{1}{N}\sum_{i=1}^{N}\{\exp(-y_i*f(x_i))\}=\prod_{t}Z_t$,由式(5-4)可得出:

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}\{\exp(-y_i*f(x_i))\}&=\frac{1}{N}\sum_{i=1}^{N}\exp\Big(-\sum_{t=1}^{T}\alpha_t y_i H_t(x_i)\Big)\\&=\sum_{i=1}^{N}u_{1,i}\prod_{t=1}^{T}\exp(-\alpha_t y_i H_t(x_i))\\&=Z_1\sum_{i=1}^{N}u_{2,i}\prod_{t=2}^{T}\exp(-\alpha_t y_i H_t(x_i))\\&=Z_1Z_2\sum_{i=1}^{N}u_{3,i}\prod_{t=3}^{T}\exp(-\alpha_t y_i H_t(x_i))\\&\cdots\\&=\prod_{t=1}^{T}Z_t\end{aligned} \tag{5-15}$$

由此,通过式(5-15)可以看出在每轮迭代过程中,只需要满足训练得到的分类器使得归一化因子 Z_t 最小,便可以进一步降低训练误差。由式(5-3)可得到:

$$\begin{aligned}Z_t&=\sum_{i=1}^{N}u_{t,i}\exp(-\alpha_t y_i H_t(x_i))\\&=\sum_{y_i=H_t(x_i)}^{N}u_{t,i}\exp(-\alpha_m)+\sum_{y_i\neq H_t(x_i)}^{N}u_{t,i}\exp(\alpha_t)\end{aligned}$$

$$=(1-\varepsilon_t)\exp(-\alpha_m)+\varepsilon_t\exp(\alpha_m)$$

把式(5-2)$\alpha_t=\frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$代入上式中可以得到$2\sqrt{\varepsilon_t(1-\varepsilon_t)}=\sqrt{1-4r_m^2}$。在这里我们令$r_m=\frac{1}{2}-\varepsilon_t$，通过泰勒公式分别对$e^x$和$\sqrt{1-x}$展开，所展开的泰勒公式如下所示：

$$f(x)=\frac{f(x_0)}{0!}+\frac{f'(x_0)}{1!}(x-x_0)+\frac{f''(x_0)}{1!}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+R_n(x) \tag{5-16}$$

这里只展示三阶的泰勒展开式，分别得到e^x和$\sqrt{1-x}$：

$$e^x=1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\cdots+R_n(x) \tag{5-17}$$

$$\sqrt{1-x}=1-\frac{1}{2}x-\frac{1}{4}x^2-\frac{3}{8}x^3+\cdots+R_n(x) \tag{5-18}$$

可以看出当$\sqrt{1-x}\leqslant e^x$，同理可得$\sqrt{1-4r_t^2}\leqslant\exp(-2r_t^2)$，如果存在$r>0$，对于所有的$t$有$r_t\geqslant r$成立，因此可以得到：

$$\frac{1}{N}\sum_{i=1}^{N}I(G_{x_i}\neq y_i)\leqslant\exp(-2Tr^2) \tag{5-19}$$

综上，AdaBoost算法在迭代的过程中，其误差率以指数速率下降。

5.2 AdaBoost算法实现

本章展示了算法实现的流程图和核心类。算法实现的流程如图5-2所示，包含了算法实现的类和函数。

5.2.1 简介

AdaBoost算法实现采用Python语言，主要类如表5-1所示。

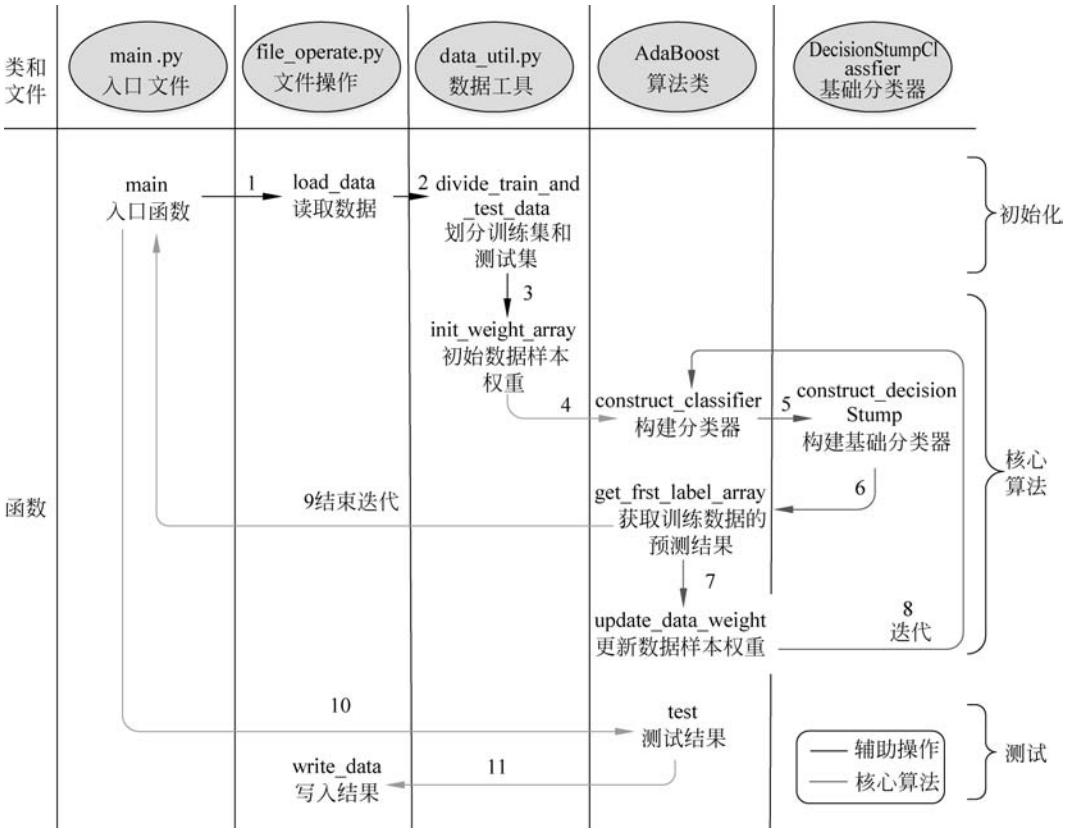

图 5-2 算法设计流程图

表 5-1 类和文件的描述

类和文件	描 述

Iris	(具体数据 Model 类 Iris)
	成员变量：
	`sepal_length # 花萼长度`
	`sepal_width # 花萼宽度`
	`petal_length # 花瓣长度`
	`petal_width # 花瓣宽度`
	`label # 标签`
	`//特征键集`
	`features = ['sepalLength', 'sepalWidth', 'petalLength', 'petalWidth']`

续表

类和文件	描　　述

SimpleDataSet	（具体数据 Model 类 SimpleDataSet）
	成员变量：
	x_ais # x 轴
	y_ais # y 轴
	label # 标签
	//特征键集
	features = ['xAis', 'yAis']

AdaBoost	（AdaBoost 算法核心类，实现分类器构建、误差率计算、数据样本更新以及预测结果获取）
	成员变量：
	train_data_array　　# 训练数据集
	test_data_array　　# 测试数据集
	train_label_array　　# 训练标签集
	weight_array　　# 样本权重集
	函数：
	def construct_classifier(self):
	　　"""
	　　构建多个基础分类器的组合
	　　:return: DecisionStump 列表
	　　"""
	
	def get_error(self, temp_array, threshold, lt_label, gt_label):
	　　"""
	　　计算当前所选特征下的误差率
	　　:param temp_array: 当前特征数组
	　　:param threshold: 阈值
	　　:param lt_label: 小于阈值的标签
	　　:param gt_label: 大于阈值的标签
	　　:return: 误差率
	　　"""

续表

类和文件	描　　述

AdaBoost	def update_data_weight(self, ds: DecisionStump):
	"""
	更新每个数据样本的权重
	:param ds: 当前构造的分类器
	:return: weightArray 更新后的数据样本权重
	"""
	
	def get_frst_lable_array(self, ds_list, data_flag: int):
	"""
	综合每次所得到的分类器,代入原始数据计算最终预测结果
	:param ds_list: 分类器集
	:param data_flag: 分类器集
	:return: 预测标签集
	"""
	
	def test(self, ds_list, test_label_array):
	"""
	测试分类器
	:param ds_list: 分类器集
	:param test_label_array: 测试标签集
	:return:
	"""

DecisionStump-Classifier	(实现基础分类器之一——决策树桩的构建,打印全部的决策树桩) 函数: def construct_decision_stump(ada_boost, train_data_array): """ 构建当次决策树桩 :param ada_boost: 算法对象 :param train_data_array: 训练数据 :return: DesicionStumpList 当次的决策树桩(为空时表示当前最低错误率大于 0.5,使得无法构建决策树桩)

续表

类和文件	描　　述
DecisionStump-Classifier	""" def print_decision_stump_classifier(self, ds_list, cur_node_model): """ 打印决策树桩分类器 :param ds_list: 决策树集 :param cur_node_model: 当前数据实体 :return: """
DecisionStump	（决策树桩的 Model 实体类） 成员变量： feature_index　　# 所选特征 threshold　　# 阈值 lt_label　　# 小于阈值的分类 gt_label　　# 大于阈值的分类 alpha_weight　　# 当前分类器权重
data_util. py	（有关数据的工具类，在这里我们实现了：①从全部数据中取出不含类别标签的数据集和从类别标签集；②根据随机数从原始数据中划分训练数据、测试数据、训练标签和测试标签；③初始化数据样本权重） 函数： def divide_train_and_test_data(datas, percent, begin, end): """ 把原始数据划分为训练数据、测试数据、训练标签和测试标签 :param datas: 原始数据 :param begin: 随机数的开始数 :param end: 随机数的结束数 :param percent: 随机数所占比例 :return: 训练数据、测试数据、训练标签和测试标签 """

续表

类和文件	描　　述

data_util. py	def get_data_array(datas, feature_count):
	"""
	从全部数据样本中取出不含标签的数据集
	:param datas: 原始数据
	:param feature_count: 标签总数
	:return: 不含标签的数据集
	"""
	
	def get_label_array(datas: list, label_col_index: int):
	"""
	从全部数据样本中取出标签集
	:param datas:
	:param label_col_index:
	:return:
	"""
	
	def init_weight_array(train_data_count):
	"""
	初始化训练数据样本权重
	:param train_data_count: 训练数据样本数量
	:return:
	"""

Configuration	(配置类,包括读写数据路径和算法所需参数的配置)
	属性:
	class Configuration:
	# 读数据的路径
	DATA_PATH = "../data/wine.txt"
	# 写数据的路径
	RESULT_PATH = "../data/wine_result.txt"
	# 数据样本类别标签所在的列号
	LABEL_INDEX = 0

续表

类和文件	描　　述

Configuration	# 所选数据样本开始的编号
	BEGIN = 1
	# 所选数据样本结束的编号
	END = 100
	# 算法最大迭代次数
	MAX_ITER = 500
	# 训练数据样本占全部样本的比例,范围:0.5<= percent<= 0.9
	PERCENT = 0.8
	# 实例化数据实体类
	NODE_MODEL = model.Wine

main.py	（AdaBoost 算法入口文件） 函数： def main():

5.2.2 核心代码

根据算法的思想与流程给出其核心代码，包括 Main 类、AdaBoost 类和 DecisionStumpClassifier 类的详细代码和解释说明，在这里我们选的基本分类器为决策树桩。

1. main() 方法

（1）首先获取原始数据并划分训练集和测试集。

```
start_time = time.clock()

# step1 获取原始数据并划分训练集和测试集
train_data_array, train_label_array, test_data_array, test_label_array = \
    divide_train_and_test_data(load_data(Config.DATA_PATH, delim = None),
                               Config.PERCENT, Config.BEGIN, Config.END)
```

（2）初始化训练数据样本权重。

```
# step2 初始化训练数据样本权重
weight_array = init_weight_array(len(train_label_array)
```

（3）核心步骤，构建多个分类器。

```
# step3 核心步骤,构建多个基础分类器的组合
ada_boost = AdaBoost(train_data_array,
                     test_data_array,
                     train_label_array,
                     weight_array)
ds_list = ada_boost.construct_classifier()
```

（4）把测试数据代入最终分类器，得到的预测标签集并统计正确与错误数。

```
# step4 测试
right_count, error_count = ada_boost.test(ds_list, test_label_array)

end_time = time.clock()
```

（5）把结果输出到文件。

```
# step5 打印输出最终分类器和测试结果到文件
write_data(Config.RESULT_PATH, ds_list, Config.NODE_MODEL, right_count,
               error_count, start_time, end_time)
```

2. AdaBoost 类

（1）构建多个基础分类器的组合。

```
 def construct_classifier(self) ->list:
     """
     构建多个基础分类器的组合
     :return: DecisionStump 列表
```

```
    """
    ds_list = []
    dsc = DecisionStumpClassifier()
    for i in range(1, Config.MAX_ITER):
        ds = dsc.construct_decision_stump(self, self.__train_data_array)
        if ds is None:
            continue
        else:
            ds_list.append(ds)
            # 获取多个分类器 boost 后得到的预测标签集,1 表示采用训练数据集
            frst_lable_array = self.get_frst_lable_array(ds_list, 1)
            # 比较预测标签集和训练数据真实标签集
            # 如果都预测正确,结束迭代
            if operator.eq(frst_lable_array, self._train_label_array):
                break
            # 否则更新数据样本权重并继续获取下一个分类器
            else:
                self.__weight_array = self.update_data_weight(ds)

    return ds_list
```

（2）在构建决策树桩分类器时计算该分类器的误差率。

```
def get_error(self, temp_array: list, threshold: float,
                  lt_label: int, gt_label: int) -> float:
    """
    计算当前所选特征下的误差率
    :param temp_array: 当前特征数组
    :param threshold: 阈值
    :param lt_label: 小于阈值的标签
    :param gt_label: 大于阈值的标签
    :return: 误差率
    """
    error = .0
    for i in range(len(temp_array)):
        if temp_array[i] < threshold:
            if self._train_label_array[i] != lt_label:
                error += self._weight_array[i]
```

```
            else:
                  if self._train_label_array[i] != gt_label:
                      error += self._weight_array[i]
        return error
```

（3）如果现有分类器不能完全准确分类训练数据集，则本次需要更新每个数据样本的权重。

```
def update_data_weight(self, ds: DecisionStump):
    """
    更新每个数据样本的权重
    :param ds: 当前构造的分类器
    :return: weightArray 更新后的数据样本权重
    """
    feature_index = ds.feature_index  # 所选特征的索引号
    threshold = ds.threshold          # 阈值
    lt_label = ds.lt_label            # 小于阈值的类别
    gt_label = ds.gt_label            # 大于阈值的类别
    alpha_weight = ds.alpha_weight    # 当前分类器的权重

    temp_array = self.__train_data_array[feature_index]

    # 计算规范化因子
    def z_cal(tmp_ele, train_label_ele, weight_ele):
        # 小于阈值
        if tmp_ele<threshold:
            # 预测标签不等于真实标签
            if train_label_ele != lt_label:
                return weight_ele * math.pow(math.e, alpha_weight)
            else:
                return weight_ele * math.pow(math.e, -alpha_weight)
        else:
            if train_label_ele != gt_label:
                return weight_ele * math.pow(math.e, alpha_weight)
            else:
                return weight_ele * math.pow(math.e, -alpha_weight)

```

```
        self._weight_array = [z_cal(temp_array[i],
                                    self._train_label_array[i],
                                    self._weight_array[i])
                              for i in range(len(temp_array))]
        z = sum(self._weight_array)
        self._weight_array = [w / z for w in self._weight_array]
        return self._weight_array
```

（4）综合每次所得到的分类器，代入数据计算每个数据样本最终预测结果。

```
def get_frst_lable_array(self, ds_list, data_flag: int):
    """
    综合每次所得到的分类器,代入原始数据计算最终预测结果
    :param ds_list: 分类器集
    :param data_flag: 分类器集
    :return: 预测标签集
    """
    tmp_labels = [.0] * len(self._train_data_array[0])
    cur_data_array = self._train_data_array \
        if 1 == data_flag else self._test_data_array

    for ds in ds_list:
        feature_index = ds.feature_index      # 所选特征的索引号
        threshold = ds.threshold              # 阈值
        lt_label = ds.lt_label                # 小于阈值的类别
        gt_label = ds.gt_label                # 大于阈值的类别
        alpha_weight = ds.alpha_weight        # 当前分类器的权重

        tmp_array = cur_data_array[feature_index]

        tmp_labels = [alpha_weight * lt_label + tmp_labels[j]
                      if tmp_array[j] < threshold
                      else alpha_weight * gt_label + tmp_labels[j]
                      for j in range(len(tmp_array))]

    return [math.copysign(1.0, tmp_ele) for tmp_ele in tmp_labels]
```

（5）测试分类器。

```
def test(self, ds_list, test_label_array):
    """
    测试分类器
    :param ds_list: 分类器集
    :param test_label_array: 测试标签集
    :return:
    """
    right_count = 0
    error_count = 0
    if Config.PERCENT<= 0.9:
        frst_label_array = self.get_frst_lable_array(ds_list, 2)
        for i in range(len(test_label_array)):
              if frst_label_array[i] == test_label_array[i]:
                    right_count += 1
              else:
                    error_count += 1
    return right_count, error_count
```

3. DecisionStumpClassifier 类

（1）构建当次的决策树桩。

```
def construct_decision_stump(ada_boost, train_data_array):
    """
    构建当次决策树桩
    :param ada_boost: 算法对象
    :param train_data_array: 训练数据
    :return: DesicionStumpList 当次的决策树桩
    (为空表示当前最低错误率大于 0.5,使得无法构建决策树桩)
    """
    feature_index = 1          # 所选特征的索引号
    threshold = .0             # 阈值
    lt_label = 0               # 小于阈值的类别
    gt_label = 0               # 大于阈值的类别
    min_error = float('inf')   # 误差率
```

```
14
15      # 选取误差率最小的特征和阈值
16      feature_count = len(train_data_array)
17      data_count = len(train_data_array[0])
18
19      i = 1
20      while i<= feature_count:
21            tmp_array = train_data_array[i-1]   # 数据样本的当前所选特征集合
22            max_value = max(tmp_array)
23            min_value = min(tmp_array)
24            step = (max_value - min_value) / data_count
25            tmp_threshold = min_value
26
27            # 从 min 到 max 不断增加步长,计算当前阈值下的失误率
28            while tmp_threshold<= max_value:
29                  tmp_array = train_data_array[i-1]
30                  tmp_error1 = ada_boost\
31                        .get_error(tmp_array, tmp_threshold, -1, 1)
32                  tmp_error2 = ada_boost\
33                        .get_error(tmp_array, tmp_threshold, 1, -1)
34
35                  if tmp_error1 < tmp_error2:
36                     if tmp_error1 < min_error:
37                          feature_index, threshold, = i-1, tmp_threshold
38                          lt_label, gt_label, min_error = -1, 1, tmp_error1
39                  else:
40                          if tmp_error2 < min_error:
41                          feature_index, threshold = i-1, tmp_threshold
42                          lt_label, gt_label, min_error = 1, -1, tmp_error2
43                     tmp_threshold += step
44          i += 1
45
46      # 最低错误率应满足 0<min_error<0.5
47      if 0 < min_error < 0.5:
48            alpha_weight = 0.5 * math.log((1-min_error) / min_error)
49            return DecisionStump(feature_index, threshold, lt_label,
50                                       gt_label, alpha_weight)
51
```

（2）打印决策树桩分类器。

```
def print_decision_stump_classifier(self, ds_list, cur_node_model):
    """
    打印决策树桩分类器
    :param ds_list: 决策树集
    :param cur_node_model: 当前数据实体
    :return:
    """
    print('共{:d}个决策树桩'.format(len(ds_list)))
    for index, ds in enumerate(ds_list):
        print(self
                .info
                .format(ds_num = index + 1,
                        threshold = ds.threshold,
                        feature = cur_node_model.features[ds.feature_index],
                        lt_label = ds.lt_label,
                        gt_label = ds.gt_label,
                        alpha_weight = ds.alpha_weight))
```

5.3 实验数据

本书选择在公开数据集 UCI 上的 Iris 数据集(http://archive.ics.uci.edu/ml/datasets/Iris)以及一份简单的数据集。

Iris 数据集中每个数据样本均有四个属性和一个标签，表 5-2 展示了部分数据样本。

表 5-2 Iris 数据集部分数据样本

sepal_length	sepal_width	petal_length	petal_width	class
5.1	3.5	1.4	0.2	1
4.9	3.0	1.4	0.2	1
4.7	3.2	1.3	0.2	1
4.6	3.1	1.5	0.2	1

续表

sepal_length	sepal_width	petal_length	petal_width	class
7.0	3.2	4.7	1.4	2
6.4	3.2	4.5	1.5	2
6.9	3.1	4.9	1.5	2
5.5	2.3	4.0	1.3	2

另一份数据集来自《机器学习实战》第 5 章，该书作者把它用于 Logistic 回归。本书把该数据集取名为 SimpleDataSet，用于 AdaBoost。表 5-3 是该数据集的部分数据样本。

表 5-3　SimpleDataSet 数据集部分数据样本

x_axis	y_axis	class
−0.017612	14.053064	−1
−1.395634	4.662541	1
−0.752157	6.53862	−1
−1.322371	7.152853	−1
0.423363	11.054677	−1
0.406704	7.067335	1
0.667394	12.741452	−1
−2.46015	6.866805	1

5.4　实验结果

5.4.1　结果展示

对于 Iris 数据集，由于其中有四个属性不便于可视化展示，因此我们给出经 AdaBoost 后所生成的分类器集。对于 SimpleDataSet，我们则绘制实验效果图以展示结果。

首先，我们把 AdaBoost 用于 Iris 数据集，表 5-4 展示了所生成的分类器集，同时展示了把测试数据用于最终分类器所预测得到的正确分类和错误分类数。

表 5-4 AdaBoost 用于 Iris 数据集后生成的分类器集与测试结果

总共用时：414ms 共 14 个分类器	
第 1 个分类器为： 取的特征为：petalWidth，阈值为：1.60 1，petalWidth＜1.60 －1，petalWidth≥1.60 权重：1.354025100551105	第 2 个分类器为： 取的特征为：petalLength，阈值为：4.92 1，petalLength＜4.92 －1，petalLength≥4.92 权重：0.9076449833191255
第 3 个分类器为： 取的特征为：petalLength，阈值为：5.14 1，petalLength＜5.14 －1，petalLength≥5.14 权重：0.703945231704386	第 4 个分类器为： 取的特征为：sepalLength，阈值为：6.51 －1，sepalLength＜6.51 1，sepalLength≥6.51 权重：0.6839372392899267
第 5 个分类器为： 取的特征为：petalLength，阈值为：4.83 1，petalLength＜4.83 －1，petalLength≥4.83 权重：0.429936514537254	第 6 个分类器为： 取的特征为：petalWidth，阈值为：1.71 1，petalWidth＜1.71 －1，petalWidth≥1.71 权重：0.4899552768715438
第 7 个分类器为： 取的特征为：sepalWidth，阈值为：2.81 －1，sepalWidth＜2.81 1，sepalWidth≥2.81 权重：0.39721503595826485	第 8 个分类器为： 取的特征为：petalWidth，阈值为：1.30 1，petalWidth＜1.30 －1，petalWidth≥1.30 权重：0.3975433595017072
第 9 个分类器为： 取的特征为：petalLength，阈值为：5.14 1，petalLength＜5.14 －1，petalLength≥5.14 权重：0.6062702960358095	第 10 个分类器为： 取的特征为：sepalWidth，阈值为：3.10 －1，sepalWidth＜3.10 1，sepalWidth≥3.10 权重：0.4433451016170988

续表

第 11 个分类器为： 取的特征为：petalWidth，阈值为：1.71 1，petalWidth＜1.71 －1，petalWidth≥1.71 权重：0.3668771900466325	第 12 个分类器为： 取的特征为：sepalWidth，阈值为：2.61 －1，sepalWidth＜2.61 1，sepalWidth≥2.61 权重：0.4294725865428733
第 13 个分类器为： 取的特征为：petalLength，阈值为：4.42 1，petalLength＜4.42 －1，petalLength≥4.42 权重：0.4364675253599789	第 14 个分类器为： 取的特征为：petalLength，阈值为：5.14 1，petalLength＜5.14 －1，petalLength≥5.14 权重：0.43145215977489
测试数据共 20 个，正确 19 个，错误 1 个	

其次，我们把 AdaBoost 用于 SimpleDataSet 数据集，因为该数据集只有两个属性和一个标签，便于可视化，因此我们绘制该数据集及所生成的分类器，实验效果如图 5-3～图 5-6 所示。

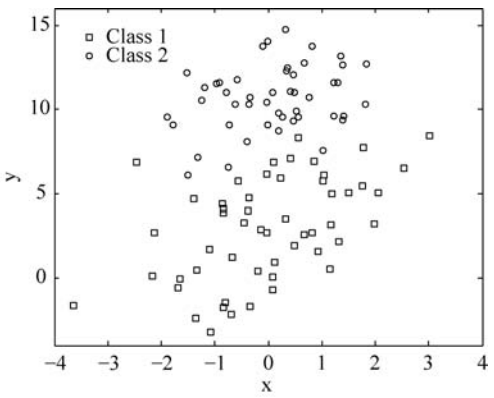

图 5-3　原始数据集

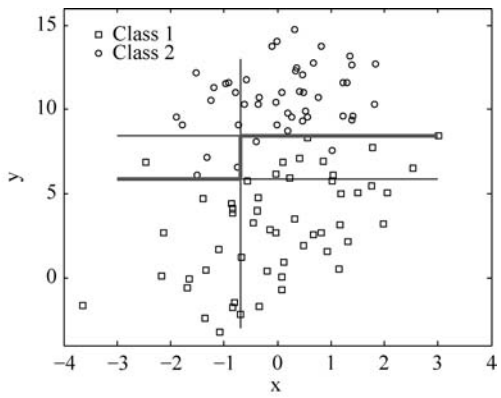

图 5-4　3 个决策树桩分类器

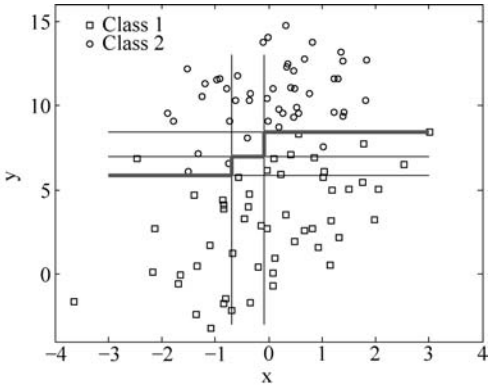

图 5-5 5 个决策树桩分类器

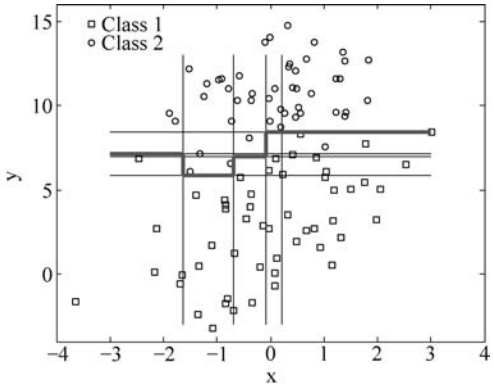

图 5-6 10 个决策树桩分类器

5.4.2 结果分析

通过所展示的效果图，我们可以看到随着基础分类器的逐渐增加，所生成的综合分类器在划分数据集中的 Class 1 和 Class 2 时逐渐准确，这也进一步证实 AdaBoost 算法综合多个基础分类器后所取得的效果远优于一个基础分类器。

AdaBoost 算法最开始仅用于解决二分类问题，后扩展到解决多分类问题。当数据样本具有多个类别时，可采用两种方式将其转换为二分类问题。一种方式是把训练样本某一类当成一类，剩下的几类归为同一类；另一种方式是选择训练样本某一类当成一类，再选择另外的某一类自成一类。不管哪种方式，AdaBoost 均会根据类别两两组合情况生成多个最终分类器，并基于“多数表决”的规则把全部分类器所预测的最多类别作为当前数据样本类别。