组合方法

本文将介绍树模型，及其相关的组合方法，并用 Python 实现原型。

****

2014-12-29

作者：钟超

目录

2 Boosting

2.1 Boosting 过程

提升（boosting）一词表示可以将一系列的弱学习算法转化为强学习算法。从直觉上来说，弱学习算法是指比随机猜测稍强的学习算法，强学习算法是指性能非常完美的学习算法。Boosting 算法起初是为了回答 Kearns 和 Valiant 于 1989 年提出的弱可学习和强可学习是否是等价的问题。弱学习和强学习之间的关系问题是非常重要的基础问题，因为如果弱学习和强学习是等价的，那么弱学习就可以转化为强学习，在实际应用中得到弱学习器是非常容易的而得到强学习器却比较难。Schapire 于 1990 年证明弱学习可以转化为强学习。

Boosting 过程非常，以二分类为例，预测样本将被分为正样本和负样本。训练样本空间为$\mathcal{X}$，其中的样本是依分布$\mathcal{D}$独立同分布产生的，分类函数为f。假设空间$\mathcal{X}$由$\mathcal{X}_1$，$\mathcal{X}_2$，$\mathcal{X}_3$组成，每个部分占 1/3，用随机猜测的分类器对样本进行分类，错误率为 50%。我们想获得准确的分类结果（分类误差率为 0）但是很不幸，手头只有弱分类器，它只能在$\mathcal{X}_1$，$\mathcal{X}_2$上有准确的分类结果，在$\mathcal{X}_3$上的分类结果是错的，这样分类误差率就为 1/3。将这个弱分类器定义为h_1。显然h_1并不能满足需求。

Boosting 的思想是通过h_1校准分类误差。可以从$\mathcal{D}$中产生新的分布$\mathcal{D}'$，在新的分布上h_1错误分类的误差率变大（**此处的分类误差率是带权重的**），因为此时的$\mathcal{X}_3$上的样本权重变大了。接着可以在$\mathcal{D}'$上训练分类器h_2，假设此时的h_2依然是弱分类器，它只能在$\mathcal{X}_1$，$\mathcal{X}_3$上正确分类，在$\mathcal{X}_2$上分类错误。以一种合适的方式将h_1和h_2组合起来，组合分类器在$\mathcal{X}_1$上能正确分类，在$\mathcal{X}_2$和$\mathcal{X}_3$上可能有错分情况。根据h_1和h_2组合效果，可以从$\mathcal{D}$中产生新的分布$\mathcal{D}''$，在$\mathcal{D}''$分布上h_1和h_2组合起来的分类误差率亦会变大，继续训练一个分类器h_3，其在$\mathcal{X}_2$和$\mathcal{X}_3$上分类效果很好。这时将h_1，h_2和h_3组合起来就得到了非常完美的分类器，因为在$\mathcal{X}_1$，$\mathcal{X}_2$，$\mathcal{X}_3$上组合分类器能够完全正确分类。

简单的说，Boosting 方法就是训练一系列分类器，并将它们组合起来进行预测，后一个分类器更关注前一个分类器分错了的样本。图 2.1 概述了一般的 Boosting 过程。

输入： 样本分布：$\mathcal{D}$
基本的学习算法：$\mathfrak{L}$
学习分类器的个数：T

训练过程：

```
1 D_1 = D
2 for t = 1,···,T:
3     h_t = L(D_t);
4     ε_t = P_{x~D_t}(h_t(x) ≠ f(x));
5     D_{t+1} = Adjust_Distribution(D_t, ε_t);
6 end
```

输出： $H(x) = Combine\_Outputs(\{h_1(x), \cdots, h_t(x)\})$

图 2.1 一般的 Boosting 过程

2.2 AdaBoost 算法

图2.1所介绍的一般化的Boosting过程并不是真正的算法，因为它没有详细介绍实现过程，如$Adjust_Distribution$和$Combine_Outputs$等。AdaBoost算法[Freund and Schapire,1997]是最有影响力的算法，可以将其作为一个示例进行讲解，算法过程见图2.2。

输入： 数据集$\mathrm{D}=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$
基本的学习算法$\mathfrak{L}$
学习分类器的个数T

训练过程：

1 $\mathcal{D}_1(x)=\frac{1}{m}$。%初始化目标值的权重分布
2 **for** $\mathrm{t}=1,\cdots,\mathrm{T}$:
3 $h_t=\mathfrak{L}(D,\mathcal{D}_t)$; %训练集$D$在分布$\mathcal{D}_t$下训练分类器$h_t$
4 $\epsilon_t=P_{x\sim D_t}\big(h_t(x)\neq f(x)\big)$; %计算$h_t$的分类误差率
5 if $\epsilon_t>0.5$ then break
6 $\alpha_t=\frac{1}{2}ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; %计算$h_t$的权重
7 $\mathcal{D}_{t+1}(x)=\frac{\mathcal{D}_t(x)}{Z_t}\times\begin{cases}\exp(-\alpha_t)\ if\ h_t(x)=f(x)\\ \exp(\alpha_t)\ if\ h_t(x)\neq f(x)\end{cases}=\frac{\mathcal{D}_t(x)\exp(-\alpha_t f(x)h_t(x))}{Z_t}$
; %更新分布，Z_t是归一化因子，其使$\mathcal{D}_{t+1}$满足概率分布的性质
8 **end**

输出： $H(x)=\mathrm{sign}(\sum_{t=1}^{T}\alpha_t h_t(x))$

图 2.2 AdaBoost 算法

考虑二分类问题，类别集合为$\{+1,\ -1\}$。AdaBoost[Friedman et al., 2000]通过指数损失函数最小化达到训练模型的目的

$$l_{exp}(h|\mathcal{D})=\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)h(x)}\right] \tag{2.1}$$

使用加权组合将弱分类器组合起来

$$H(x)=\sum_{t=1}^{T}\alpha_t h_t(x) \tag{2.2}$$

指数损失函数简单简洁的计算特征是这里采取它为损失函数的原因之一，同时它还和最小化分类误差的目标向一致，并且通过它和对数似然函数之间的关系证明此处采用指数损失函数是合理的。当通过H最小化指数损失函数时，指数损失函数关于每个x的偏导数为零

$$\frac{\partial e^{-f(x)H(x)}}{\partial H(x)}=-f(x)e^{-f(x)H(x)}=-e^{-H(x)}P(f(x)=1|x)+e^{H(x)}P(f(x)=-1|x)=0 \tag{2.3}$$

求解式（2.3）得到

$$H(x)=\frac{1}{2}\ln\frac{P(f(x)=1|x)}{P(f(x)=-1|x)} \tag{2.4}$$

因此有

$$\begin{aligned}\mathrm{sign}\big(H(x)\big)&=\mathrm{sign}\left(\frac{1}{2}\ln\frac{P(f(x)=1|x)}{P(f(x)=-1|x)}\right)=\begin{cases}1, & P(f(x)=1|x)>P(f(x)=-1|x)\\ -1, & P(f(x)=1|x)<P(f(x)=-1|x)\end{cases}\\&=\mathrm{argmax}P(f(x)=y|x),y\in\{+1,-1\}\end{aligned} \tag{2.5}$$

从上式可以看出$\mathrm{sign}\big(H(x)\big)$等价于贝叶斯误差率（**贝叶斯错误率**是应用贝叶斯分类规则的分类器的错误率。贝叶斯分类规则的一个性质是:在最小化分类错误率上是最优的，所以在分类问题中，**贝叶斯错误率**是一个分类器对某个类别所能达到的最低的分类错误率。）。注意这里忽略了$P(f(x)=1|x)=P(f(x)=-1|x)$的情况。以上的公式推导过程表明，当指数损失函数最小时，分类误差率也是最小的，所以指数损失函数可以代替不可微分分类误差率，作为目标函数。

α_t和$h_t(x)$的组合起来组成了H。刚开始时，在原始分布上训练弱分类器得到h_1。在$\mathcal{D}_t$上训练$h_t(x)$后，权重α_t可以通过最小化$\alpha_t h_t(x)$的指数损失确定

$$l_{exp}(\alpha_t h_t|\mathcal{D}_t) = \mathbb{E}_{x\sim\mathcal{D}_t}\left[e^{-f(x)\alpha_t h_t}\right] = \mathbb{E}_{x\sim\mathcal{D}_t}\left[e^{-\alpha_t}\mathbb{I}\left(f(x) = h_t(x)\right) + e^{\alpha_t}\mathbb{I}\left(f(x) \neq h_t(x)\right)\right]$$
$$= e^{-\alpha_t}P_{x\sim\mathcal{D}_t}\left(f(x) = h_t(x)\right) + e^{\alpha_t}P_{x\sim\mathcal{D}_t}\left(f(x) \neq h_t(x)\right)$$
$$= e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t \tag{2.6}$$

其中，$\epsilon_t = P_{x\sim\mathcal{D}_t}\left(h_t(x) \neq f(x)\right)$。为了获得最优化的$\alpha_t$，将$l_{exp}(\alpha_t h_t|\mathcal{D})$对$\alpha_t$求导，并使其等于 0

$$\frac{\partial l_{exp}(\alpha_t h_t|\mathcal{D})}{\alpha_t} = e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t = 0 \tag{2.7}$$

解得到

$$\alpha_t = \frac{1}{2}\ln\left(\frac{(1-\epsilon_t)}{\epsilon_t}\right) \tag{2.8}$$

这就是图 2.2 中第六行的公式。

一旦这些分类器训练完成，并且得到了每个分类器相应的权重，将这些训练好的分类器组合起来作为H_{t-1}。接着 AdaBoost 算法调节样本分布，准备下一次训练，基本学习算法将输出一个弱分类器h_t，它将对被H_{t-1}误分的样本进行最大可能的正确分类。再次考虑指数损失函数，理想的分类器h_t对被H_{t-1}误分的样本进行正确分类时，必须最小化如下指数损失函数

$$l_{\exp}(H_{t-1} + h_t|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)\left(H_{t-1}(x)+h_t(x)\right)}\right] = \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}e^{-f(x)h_t(x)}\right] \tag{2.9}$$

将$e^{-f(x)h_t(x)}$泰勒展开，式（2.9）可以近似为

$$l_{\exp}(H_{t-1} + h_t|\mathcal{D}) \approx \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\left(1 - f(x)h_t(x) + \frac{f(x)^2h_t(x)^2}{2}\right)\right]$$
$$= \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\left(1 - f(x)h_t(x) + \frac{1}{2}\right)\right] \tag{2.10}$$

其中$f(x)^2 = 1$，$h_t(x)^2 = 1$。

因此理想的分类器h_t为

$$h_t(x) = \arg_h\min l_{\exp}(H_{t-1} + h_t|\mathcal{D}) = \arg_h\min\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\left(1 - f(x)h_t(x) + \frac{1}{2}\right)\right]$$
$$= \arg_h\max\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}f(x)h_t(x)\right]$$
$$= \arg_h\max\mathbb{E}_{x\sim\mathcal{D}}\left[\frac{e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\right]}f(x)h_t(x)\right] \tag{2.11}$$

其中$\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\right]$为常数。

将$\mathcal{D}_t$表示为

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x)e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\right]} \tag{2.12}$$

根据数学期望的定义，式（2.11）等价于

$$h_t(x) = \arg_h\max\mathbb{E}_{x\sim\mathcal{D}}\left[\frac{e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\right]}f(x)h_t(x)\right] = \arg_h\max\mathbb{E}_{x\sim\mathcal{D}_t}[f(x)h_t(x)] \tag{2.13}$$

由于$f(x)h_t(x) = 1 - 2\mathbb{I}\left(f(x) \neq h_t(x)\right)$，所以分类器$h_t(x)$为

$$h_t(x) = \arg_h\min\mathbb{E}_{x\sim\mathcal{D}_t}[f(x) \neq h_t(x)] \tag{2.14}$$

可以看到理想分类器$h_t(x)$在分布$\mathcal{D}_t$下使分类误差最小。因此在$\mathcal{D}_t$分布下学习弱分类器，而且在$\mathcal{D}_t$分布下弱分类器$h_t(x)$的分类误差率要小于 0.5。

$\mathcal{D}_t$和$\mathcal{D}_{t+1}$有如下关系

$$\mathcal{D}_{t+1}(x) = \frac{\mathcal{D}(x)e^{-f(x)H_t(x)}}{\mathbb{E}_{x\sim\mathcal{D}}[e^{-f(x)H_t(x)}]} = \frac{\mathcal{D}(x)e^{-f(x)H_{t-1}(x)}e^{-f(x)\alpha_t h_t(x)}}{\mathbb{E}_{x\sim\mathcal{D}}[e^{-f(x)H_t(x)}]}$$
$$= \mathcal{D}_t(x)e^{-f(x)\alpha_t h_t(x)}\frac{\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\right]}{\mathbb{E}_{x\sim\mathcal{D}}[e^{-f(x)H_t(x)}]} \quad (2.15)$$

式（2.15）就是图 2 的第 7 行更新分布的公式。

需要注意的是，图 2.2 描述的 AdaBoost 算法要求基本的学习算法能够学习不同的样本分布问题。在每次迭代训练中根据之前组合分类器的分类误差率对训练数据**重加权**，以达到调整其分布的目的。如果基本的学习算法无法学习加权后的训练样本，则可以采用**重采样**技术对原始样本进行处理，即在每次训练中根据期望的分布对样本进行采样。

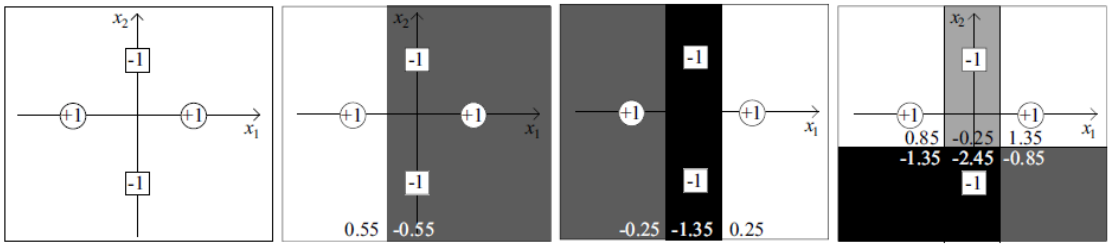
如果学习算法既可以学习重加权数据也可以学习重采样数据，通常来说这两种处理训练样本的方式不会对训练出的分类器性能有什么影响。但是重采样技术可以重新计算 Boosting[Kohavi andWolpert, 1996]。在 AdaBoost 每次计算过程中都进行一次有效校验（图 2.2 中的第 5 行），这种校验能够确保当前学习得到的基本分类器比随机猜测的效果要好。当只有少量的弱分类器时，这种校验可能带来一些事与愿违的结果，AdaBoost 的迭代过程可能提前结束，所以也就不会进行T次迭代计算。在多分类问题时这种情况经常发生。当采用重采样方法时，无法通过有效验证的分类器可以被丢弃，同时产生一个新的数据样本，在新的数据样本上重新训练基本分类器，此时 AdaBoost 可以避免过早迭代结束。

2.3 例子

通过观察 AdaBoost 的行为，有利于对其直观认识。以二维空间上人工构造的数据集为例，图形见图 2.3（a）。该数据集仅有四个样本，如下

$$\begin{Bmatrix} z_1 = (+1,0), y_1 = +1 \\ z_2 = (-1,0), y_1 = +1 \\ z_3 = (0,+1), y_1 = -1 \\ z_4 = (0,-1), y_1 = -1 \end{Bmatrix}$$

$y_i = f(z_i)$是样本的标签。这是一个 XOR 问题。从图中可以看出，无法用一条直线将正负样本分开，换句话说线性分类器无法将这两类样本分开



(a) The XOR data　(b) 1st round　(c) 2nd round　(d) 3rd round

图 2.3 XOR 问题上的 AdaBoost

假设有一个基准的学习算法，它对八个基函数h_1到h_8进行求值，如图 2.4 所示，这个基准学习算法的输出结果是分类误差最小的那个基函数。如果有多于一个基函数的分

类误差一样且最小，则可从中随机选取一个。可以看出这八个基函数没有一个能够将正负样本分开。

下面将会看到 AdaBoost 算法是如何运算的：

1 首先在原始的数据集上训练基准学习算法。因为h_2，h_3，h_5，h_8都具有最小的分类误差率为 0.25，假设基准学习算法的输出分类器h_2。在h_2上只有样本z_1会被误分，所以分类误差率是$\frac{1}{4}=0.25$，h_2的权重$\alpha_2=0.5\times\ln 3\approx 0.55$。图 2.3（b）展示出了分类关系，阴影区域被分为负类（-1），同时在图中显示出了分类权重 0.55 和-0.55。

$$h_1(x)=\begin{cases}+1, if(x_1>-0.5)\\ -1, otherwise\end{cases} \quad h_2(x)=\begin{cases}-1, if(x_1>-0.5)\\ +1, otherwise\end{cases}$$
$$h_3(x)=\begin{cases}+1, if(x_1>+0.5)\\ -1, otherwise\end{cases} \quad h_4(x)=\begin{cases}-1, if(x_1>+0.5)\\ +1, otherwise\end{cases}$$
$$h_5(x)=\begin{cases}+1, if(x_2>-0.5)\\ -1, otherwise\end{cases} \quad h_6(x)=\begin{cases}-1, if(x_2>-0.5)\\ +1, otherwise\end{cases}$$
$$h_7(x)=\begin{cases}+1, if(x_2>+0.5)\\ -1, otherwise\end{cases} \quad h_8(x)=\begin{cases}-1, if(x_2>+0.5)\\ +1, otherwise\end{cases}$$

x_1，x_2分别为数据第一维和额第二维特征

图 2.4 基准学习器中用到的八个基函数

2 z_1的权重被加大，然后再次执行基准学习算法，这一次h_3，h_5，h_8有最小的学习误差为 0.80，假设此时选择h_3作为此次学习生成的分类器，图 2.3（c）展示了组合分类器h_2，h_3，以及它们的权重，根据它们的组合权重，用较深的颜色区分出负类区域。

3 z_2当前权重被加大，再次训练，只有h_5，h_8的分类误差最小为 1.10，从中选出h_5作为此次训练产生的分类器。图 2.3（d）显示了h_2，h_3，h_5组合情况。

结果三次计算之后，观察图 2.3 中分类权重标记标记的区域，从中可以看出z_1，z_2的分类权重标记为“+”，z_3，z_4的标记为“-”，这就意味着所有的样本都能够准确分类。所以通过将并不完美的线性分类器组合起来，AdaBoost 产生了一个准确的（分类误差率为 0）非线性分类器。

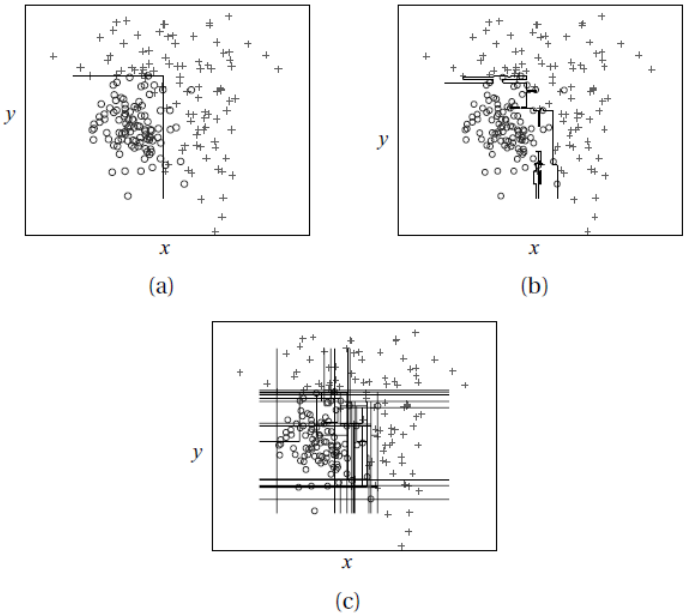


图 2.5 （a）单决策树的决策边界，（b）AdaBoost 的决策边界，（c）使用 AdaBoost 组合的十棵决策树的分类边界

为了进一步理解 AdaBoost 算法，在三个相同的高斯数据集上使用单决策树，AdaBoost，以及用 AdaBoost 组合起来的十个决策树，它们的决策边界如图 2.5。可以看到与单决策树相比，AdaBoost 的决策边界更加多变，这使得分类误差率从单决策树的 9.4% 降低到提升组合方法的 8.3%。

在 UCI 机器学习库的 40 个数据集上对 AdaBoost 算法进行评价，通过重加权方式在 50 个弱分类器上进行训练，Weka 实现了 AdaBoost.M1 算法。基本上所有的学习算法都可以当作基准学习算法，如决策树，神经网络等。此处采用三种基准的学习算法：决策树桩，经过剪枝的 J48 决策树和未剪枝的 J48 决策树（Weka 实现了 C4.5 决策树）。在图 2.6 中画出了对比结果，从中可以看出，AdaBoost 通常都能超越基准学习算法，只有极少数的 AdaBoost 算法的性能比基准算法查。

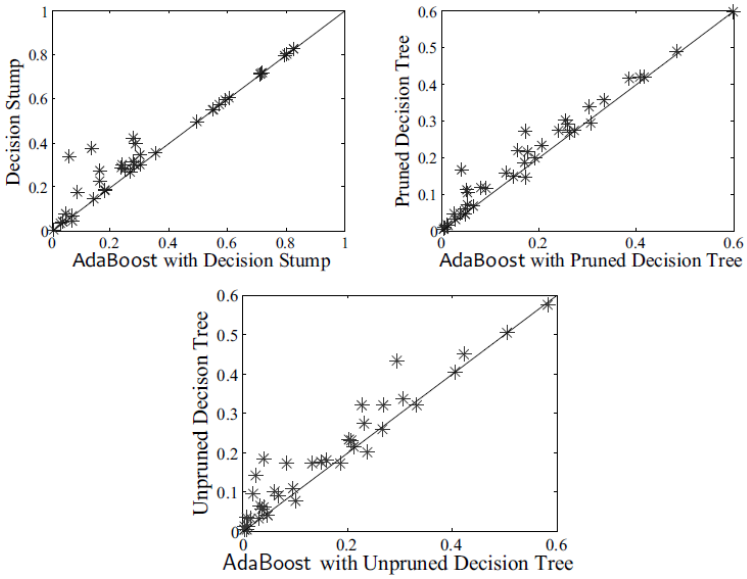


图 2.6 在 40 个 UCI 数据集上 AdaBoost 算法与各个基准算法的对比图，每个点代表一个数据集上两种算法的预测误差，对角线表两种预测算法有相同的预测误差。

2.4 相关理论分析

2.4.1 初始状态分析

Freund 和 Schapire [1997]证明，如过 AdaBoost 的基本学习算法的分类误差为$\epsilon_1, \epsilon_2, \cdots, \epsilon_T$，则最终的组合分类器上界$\epsilon$为

$$\epsilon = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{I}[H(x) \neq f(x)] \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \leq e^{-2\Sigma_{t=1}^{T}\gamma_T^2} \tag{2.16}$$

$\gamma_t = 0.5 - \epsilon_t$，可以看出 AdaBoost 以指数两级减少分类误差。这就意味着为使分类误差小于ϵ，迭代计算的步数T的上界为

$$T \leq \left\lceil \frac{1}{2\gamma^2} ln \frac{1}{\epsilon} \right\rceil \tag{2.17}$$

其中假设$\gamma \geq \gamma_1 \geq \gamma_2 \cdots \geq \gamma_T$。

但是在实际计算中所有的估计都是基于训练数据D的，即$\epsilon_D = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{I}[H(x) \neq f(x)]$，所以$\epsilon_D$表示的是训练误差，实际上更应该关注泛化误差$\epsilon_{\mathcal{D}}$。Freund和Schapire[1997]证明AdaBoost泛化误差的上界为

$$\epsilon_{\mathcal{D}} \leq \epsilon_D + \tilde{O}\left(\sqrt{\frac{dT}{m}}\right) \tag{2.18}$$

d是基准学习方法的 VC 维，m是训练样本的个数，T是训练迭代次数，为了隐藏对数项和常数项用$\tilde{O}(\cdot)$代替$O(\cdot)$。

2.4.2 对间隔的解释

式（2.18）表明，为了达到理想的泛化能力，需要控制基本学习器的复杂度和迭代计算的次数。如果不对基准学习器的复杂度和迭代计算的次数加以控制 AdaBoost 可能导致过拟合。

然而，实际研究发现 AdaBoost 通常来说不会过拟合，也就是说结果多次迭代计算（例如 1000 次），即使训练误差达到零后测试误差还会继续减小。如图 2.7 所示 Schapire et al. [1998]画出了典型的 AdaBoost 算法的性能，从中可以看到在结果 10 次迭代计算后 AdaBoost 的训练误差达到了 0，而泛化误差还在继续减小，这种现象看似与奥卡姆剃刀原理相背，奥卡姆剃刀原理认为，如果有一个复杂模型和一个简单模型，它们都能很好的解决我们的问题，则选择简单的模型。所以研究 AdaBoost 不容易过拟合问题就变成了重要的理论问题，引起了很多人的关注。

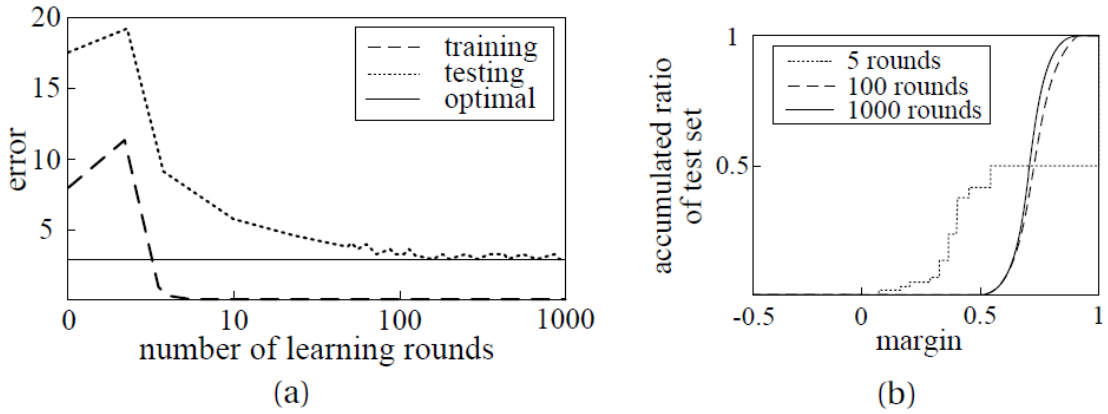


图 2.7 （a）训练误差和测试误差，（b）AdaBoost 算法在 UCI 的 letter 数据集上的泛化误差的间隔分布

Schapire et al. [1998]用基于分类间隔的方式解释了 AdaBoost。通常来说在二分类中，即$f(x) \in \{-1,+1\}$，样本x在分类器h上的边界，即样本x距离分类器h的间隔定义为$f(x)h(x)$，同理组合分类器$H(x)=\sum_{t=1}^{T}\alpha_t h_t(x)$的边界是$f(x)H(x)=\sum_{t=1}^{T}\alpha_t f(x)h_t(x)$，组合方法的归一化边界是

$$f(x)H(x)=\frac{\sum_{t=1}^{T}\alpha_t f(x)h_t(x)}{\sum_{t=1}^{T}\alpha_t} \tag{2.19}$$

α_t是基本学习算法的权重。

在边界概念的基础上，Schapire et al. [1998]证明，在训练数据集D上，以至少$1-\delta$的概率给定边界阈值$\theta>0$，则组合方法的泛化误差$\epsilon_{\mathcal{D}}=P_{x\sim\mathcal{D}}(H(x)\neq f(x))$可以写成

$$\begin{aligned}\epsilon_{\mathcal{D}} &\leq P_{x\sim\mathcal{D}}(H(x)f(x)\leq\theta)+\tilde{O}\left(\sqrt{\frac{d}{m\theta^2}+\ln\frac{1}{\delta}}\right)\\ &\leq 2^T\prod_{t=1}^{T}\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}}++\tilde{O}\left(\sqrt{\frac{d}{m\theta^2}+\ln\frac{1}{\delta}}\right)\end{aligned} \tag{2.20}$$

d，m，T，$\tilde{O}(\cdot)$和式（2.18）中的意义相同，ϵ_t是h_t的训练误差。式（2.20）说明当其它变量都固定时，在训练集上的边界越大，泛化误差越小。这样 Schapire et al. [1998]就得出，，AdaBoost 不容易过拟合原因，是由于即使在训练误差达到 0 后，组合分类器的边界还是可以增大，图 2.7（b）展示了在不同学习次数时 AdaBoost 的边界分布。

注意式（2.20）中的上界重要依赖于最小边界，因为当最小边界变大时概率$P_{x\sim\mathcal{D}}(H(x)f(x)\leq\theta)$将会变小。在这些理论基础上 Breiman [1999]开发了 arc-gv 算法，其是 AdaBoost 算法的变体，它直接最大化最小间隔（minimummargin）

$$\rho=\min_{x\in D}f(x)H(x) \tag{2.21}$$

在每次迭代计算过程中，arc-gv 根据下式更新α_t

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1+\gamma_t}{1-\gamma_t}\right)-\frac{1}{2}\ln\left(\frac{1+\rho_t}{1-\rho_t}\right) \tag{2.22}$$

γ_t是h_t的边界，ρ_t是到当前迭代计算时组合分类器的最小间隔。

在最小间隔的基础上，Breiman[1999]证明了更为严密的泛化误差上界。因为 arc-bv 的最小间隔收敛于最大可能的最小间隔，可以认为 arc-gv 的性能应该好于 AdaBoost 算法。然而，Breiman [1999]在实验的时候发现，与 AdaBoost 相比虽然 avc-gv 能够产生更大的最小间隔，但是 arc-gv 的测试误差却比 AdaBoost 要大的多。因此 Breiman[1999]确信用基于间隔的理论解释 AdaBoost 有待商榷，需要继续寻找新的理论解释 AdaBoost。这基本上否定了间隔理论。

几年后 Reyzin and Schapire [2006]公布了个有趣的发现。式（2.20）所示的泛华误差的上界与间隔，学习器迭代运行的次数以及基本学习器的复杂度有关。为了研究间隔对学习器的影响，需要把其它影响因素都固定下来。当将 arc-gv 和 AdaBoost 进行对比研究时，Breiman [1999]使用固定叶子数量的决策树来控制基本学习器的复杂度。然而，Reyzin and Schapire [2006]发现这些树有诸多不同的形状。Arc-gv 产生的树有更深的深度，AdaBoost 产生的树有更大的宽度。图 2.8（a）展示了两种算法所产生的不同深度的树。虽然树的叶子节点相同，深度较深的树具有更多的预测属性，因此这些树不可能具有相同的复杂度。所以 Reyzin and Schapire [2006]使用只有两个叶子节点的决策树桩重复了 Breiman 的实验，这样就固定了基本学习器的复杂度，发现 AdaBoost 的间隔分布优于 arc-gv，如图 2.8（b）所示。

这样就证明了间隔分布对 AdaBoost 的泛华性能有重要影响。Reyzin and Schapire [2006]建议使用平均间隔或者间隔中位数作为衡量两种算法间隔的分布。

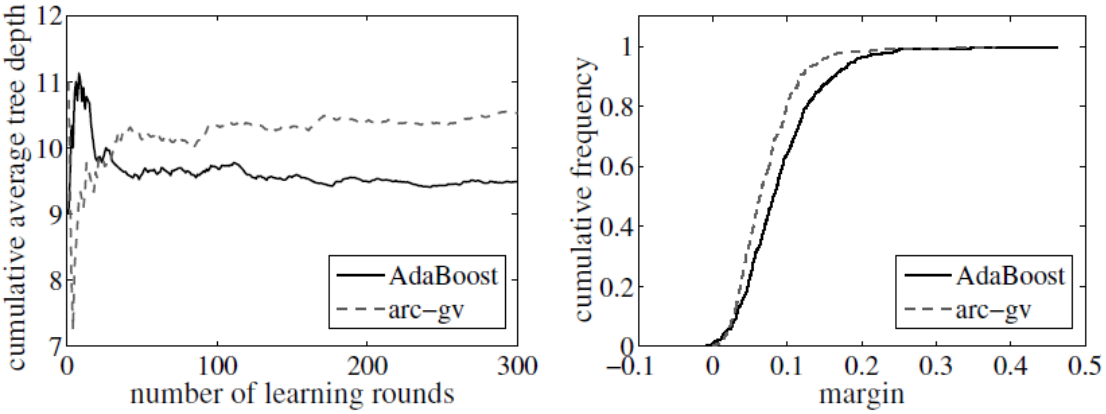


图 2.8 在 UCI 的 clearn1 数据集上 AdaBoost 和 arc-gv（a）训练的树的深度，（b）间隔分布

2.4.3 统计观点

用间隔的概念解释 AdaBoost，有很好的几何直观性，机器学习社区的人大多关注这一概念，但对统计学家却没有多大吸引力，统计学家更喜欢从统计方法的角度研究 AdaBoost。Friedman et al. [2000]在这方面取得了突破，他证明 AdaBoost 可以看成是用分段估计持续拟合加性的逻辑回归模型，这也正是式（2.2）表达的。

注意到式（2.2）是加性模型形式。式（2.1）所示的指数损失函数是可微分的，0-1 损失函数的上界用于衡量误分类的误差。如果采用逻辑函数，则

$$P(f(x)=1|x)=\frac{e^{H(x)}}{e^{H(x)}+e^{-H(x)}} \tag{2.23}$$

可以发现指数损失函数和对数损失函数

$$l_{log}(h|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[\ln\left(1+e^{-2f(x)h(x)}\right)\right] \quad (2.24)$$

都可以通过式（2.4）进行最小化。所以 Friedman et al. [2000]建议不使用 AdaBoost 中的类似牛顿法的优化方法，而使用梯度下降法优化对数损失函数对加性逻辑回归模型进行拟合，这就是 LogitBoost 算法，如图 2.9。

输入： 数据集$\mathrm{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$
最小二乘学习算法$\mathfrak{L}$
学习分类器的个数T

训练过程：

1 $y_0(x) = f(x)$。%初始化目标
2 $H_0(x) = 0$。%初始化函数
3 **for** $\mathrm{t} = 1, \cdots, \mathrm{T}$:
4 $p_t(x) = \frac{1}{1+e^{-2H_{t-1}(x)}})$； %计算概率
5 $y_t(x) = \frac{y_{t-1}(x)-p_t(x)}{p_t(x)(1-p_t(x))})$； %更新目标
6 $\mathcal{D}_t(x) = p_t(x)\left(1-p_t(x)\right)$；%更新权重
7 $h_t = \mathfrak{L}(D, y_t, \mathcal{D}_t)$； %训练分类器$h_t$用于拟合$\mathcal{D}_t$分布下的$y_t$
8 $H_t(x) = H_{t-1}(x) + \frac{1}{2}h_t(x)$； %更新组合分类器
9 **end**

输出： $H(x) = \mathrm{sign}(\sum_{t=1}^{T} h_t(x))$

图 2.9 LogitBoost 算法

根据 Friedman et al. [2000]的解释，AdaBoost 算法是对代理损失函数进行优化达到拟合加性模型的目的。理想状况下代理损失函数应该是连续的，即优化代理损失函数产生的最终的优化函数与真实的损失函数拥有相同的贝叶斯误差率，与直接优化真实的损失函数相比，优化代理损失函数的计算更高效。由于代理损失函数不同衍生出多种不同的 AdaBoost 方法，如 LogitBoost 采用的是对数损失，L2Boost 采用的是l_2损失[B¨uhlmann and Yu, 2003]。

另一方面，如果将 boosting 过程看成是优化过程，则可以使用数学规划[Demiriz et al., 2002, Warmuth et al., 2008]求解加权的弱分类器。一个弱分类器池$\mathcal{H}$中的加性模型$\sum_{h\epsilon\mathcal{H}} \alpha_h h$，$\xi_i$表示样本$x_i$在模型上的损失，Demiriz et al. [2002]证明如果有下式存在

$$\sum_{h\epsilon\mathcal{H}} \alpha_h + C\sum_{\mathrm{i}=1}^{m} \xi_i \le B \quad (2.25)$$

上式界定了模型的复杂度，因此泛化误差的上界为

$$\epsilon_{\mathcal{D}} \le \tilde{\mathrm{O}}\left(\frac{\ln m}{m}B^2\ln(Bm) + \frac{1}{m}\ln\frac{1}{\delta}\right) \quad (2.26)$$

其中$C \ge 0$，$\alpha_h \ge 0$。显然当B变小时上边界也变小。假设有T个弱分类器，$y_i = f(x_i)$是样本x_i的标签，$H_{i,j} = h_j(x_i)$是x_i在弱分类器h_j上的输出结果，所以有如下的优化问题

$$\begin{aligned} &\min_{\alpha_j,\xi_i} \sum_{j=1}^{T}\alpha_j + C\sum_{j=1}^{T}\xi_i \\ &\text{s.t.} \quad y_i\sum_{j=1}^{T} H_{i,j}\,\alpha_j + \xi_i \ge 1 (\forall i = 1, \cdots, m) \\ &\qquad \xi_i \ge 0 (\forall i = 1, \cdots, m) \\ &\qquad \alpha_j \ge 0 (\forall j = 1, \cdots, T) \end{aligned} \quad (2.27)$$

或者优化如下的等价问题

$$\max_{\alpha_j,\xi_i,\rho} \rho - C'\sum_{i=1}^{m}\xi_i$$

$$\text{s.t.}\quad y_i\sum_{j=1}^{T}H_{i,j}\,\alpha_j+\xi_i\geq\rho(\forall i=1,\cdots,m)$$
$$\sum_{j=1}^{T}\alpha_j=1$$
$$\xi_i\geq 0(\forall i=1,\cdots,m)$$
$$\alpha_j\geq 0(\forall j=1,\cdots,T)$$

（2.28）

式（2.28）的对偶形式为

$$\min_{\omega_i,\beta}\beta$$
$$\text{s.t.}\quad\sum_{i=1}^{m}\omega_i y_i H_{i,j}\leq\beta(\forall j=1,\cdots,T)$$
$$\sum_{i=1}^{m}\omega_i=1$$
$$\omega_i\in[0,C'](\forall i=1,\cdots,m)$$

（2.29）

当T变的很大时优化问题求解起来会非常困难。

2.5 多分类扩展

前面主要介绍了 AdaBoost 的二分类问题，即$\mathcal{Y}=\{+1,\ -1\}$。但是在许多实际问题中，样本的类别是多类的。例如手写识别的类别就是十类，即$\mathcal{Y}=\{0,\cdots,9\}$。有许多方式可以将 AdaBoost 扩展成多分类问题。

AdaBoost.M1 [Freund and Schapire, 1997]的扩展方式比较直接，算法过程与图 2.2 一致，只是用基本学习器从二分类变为多分类。这种处理方式无法使用二分类器，因此其有很强的局限性，every base learner has less than 1/2multiclass 0/1-loss.

SAMME [Zhu et al., 2006]在 AdaBoostg.M1 上做了些许改善，将图 2.2 中的第六行替换为如下表达式

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)+ln(|\mathcal{Y}|-1)\qquad(2.31)$$

这种改进是对多分类指数损失的最小化，可以证明这种改进类似于二分类问题，优化多分类指数损失函数能够取得贝叶斯误差率的效果，即

$$\text{sign}\big(h^*(x)\big)=\text{argmax}_{y\in\mathcal{Y}}\text{P}(y|x)\qquad(2.32)$$

其中h^*是多分类指数损失的最优解。

一般来说可以将多分类问题分解为多个二分类问题。比较流行的分解方法为 **one-versus-rest** 和 **one-versus-one**。one-versus-rest 分解法将一个具有$|\mathcal{Y}|$个类的多分类器分解为$|\mathcal{Y}|$个二分类器，其中第i个分类器对样本的分类结果为，属于第i类或者不属于第i类。**one-versus-one** 将含有$|\mathcal{Y}|$个类的多类任务分解为$\frac{|\mathcal{Y}||\mathcal{Y}|-1|}{2}$个分类器，每个分类器将样本分类为第$i$类或者第$j$类。

AdaBoost.MH [Schapire and Singer, 1999]采用 **one-versus-rest** 分类方法。需要训练$|\mathcal{Y}|$个二分类器，用$H(x)=\sum_{t=1}^{T}\alpha_t h_t(x)$而不是每个 AdaBoost 的二分类器对样本进行分类，即

$$H(x)=\text{argmax}_{y\in\mathcal{Y}}H_y(x)\qquad(2.33)$$

其中，H_y是二分类的 AdaBoost 分类器，它能够将样本分给第y个类别。

AdaBoost.M2 [Freund and Schapire, 1997]采用 one-versus-one 策略，它通过最小化 pseudo-loss。AdaBoost.M2 又被改进为 AdaBoost.MR [Schapire and Singer, 1999]，该方法通过最小化排序损失进行分类，因为排在前面的类更可能成为正确的类。通过投票法，配对耦合，有向无环图可以将 one-versus-one 法获得的二分类器组合起来[Hsu and Lin, 2002, Hastie and Tibshirani, 1998]。其中投票法和配对耦合法较为著名，图 2.11 展示了有向无环图的例子。

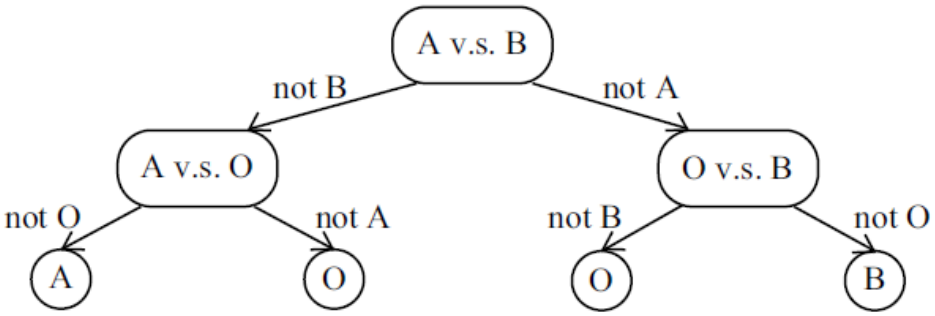


图 2.11 有向无环图组合 one-versus-one 的分类器，能够对 A，B，O 进行分类

2.6 噪声误差

实际数据中常常包括噪声。初始的 AdaBoost 算法只是为不含噪声的数据设计的，这种算法对噪声数据非常敏感。AdaBoost 算法采用的指数损失函数导致它对噪声非常敏感，因为如果有一个样本被误分，那么这个样本的权重就会变得很大。所以当一个训练样本被错误的打上了错误的标记（如一个样本本来是 A 类的，而被认为成是 B 类的），AdaBoost 仍然会将它预测为已经标记的类别，这样就会降低分类器的性能。

MadaBoost [Domingo and Watanabe, 2000]通过压缩较大权重的样本改进了 AdaBoost 算法。除了在权重更新准则上的改进外，其它方面和原生的 AdaBoost 方法一致。原生的 AdaBoost 权重更新方式如下

$$\mathcal{D}_{t+1}(x)=\frac{\mathcal{D}_t(x)}{Z_t}\times\begin{cases}\exp(-\alpha_t)\ if\ h_t(x)=f(x)\\ \exp(\alpha_t)\ if\ h_t(x)\neq f(x)\end{cases}=\frac{\mathcal{D}_t(x)\exp\left(-\alpha_t f(x)h_t(x)\right)}{Z_t}$$
$$=\frac{\mathcal{D}_1(x)}{Z_t'}\prod_{i=1}^{t}\exp\left(-\alpha_i f(x)h_i(x)\right)$$

（2.34）

Z_t和Z_t'为归一化因子。从上式可以看出，如果对一个样本的预测结果错了$\prod_{i=1}^{t}\exp\left(-\alpha_i f(x)h_i(x)\right)$将会变的很大，在下一轮训练的分类器上就对该样本做正确分类。由噪声引起的误分类问题会导致被误分类的样本权重加大，MadaBoost 算法给每个样本的权重设置了一个阈值上限

$$\mathcal{D}_{t+1}(x)=\frac{\mathcal{D}_1(x)}{Z_t'}\times min\left\{1,\prod_{i=1}^{t}\exp\left(-\alpha_i f(x)h_i(x)\right)\right\}\qquad(2.35)$$

Z_t'为归一化系数。用这种更新权重的方式，样本的权重不会无限制的增长。

FilterBoost [Bradley and Schapire, 2008]没有使用 AdaBoost 中采用的指数损失函数，而是采用了式（2.24）的对数损失函数。和 2.2 节导出的 AdaBoost 算法一致，通过拟合加性模型最小化对数损失函数。在第t轮学习过程中，组合学习器为H_{t-1}，训练产生的分类器为h_t。将损失函数进行泰勒展开，如下

$$l_{log}(H_{t-1}+h_t|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[-\ln\frac{1}{1+e^{-f(x)\left(H_{t-1}(x)+h_t(x)\right)}}\right]$$
$$\approx \mathbb{E}_{x\sim\mathcal{D}}\left[\ln\left(1+e^{-f(x)H_{t-1}(x)}\right)-\frac{f(x)h_t(x)}{1+e^{f(x)H_{t-1}(x)}}+\frac{e^{f(x)H_{t-1}(x)}}{2(1+e^{f(x)H_{t-1}(x)})^2}\right]$$
$$\approx \mathbb{E}_{x\sim\mathcal{D}}\left[-\frac{f(x)h_t(x)}{1+e^{f(x)H_{t-1}(x)}}\right] \tag{2.36}$$

其中$f(x)^2=1$，$h_t(x)^2=1$。为了最小化损失函数，h_t必须满足下式

$$h_t = \arg\min_h l_{log}(H_{t-1}+h_t|\mathcal{D}) = \arg\max_h \mathbb{E}_{x\sim\mathcal{D}}\left[\frac{f(x)h_t(x)}{1+e^{f(x)H_{t-1}(x)}}\right]$$
$$= \arg\max_h \mathbb{E}_{x\sim\mathcal{D}}\left[\frac{f(x)h_t(x)}{Z_t(1+e^{f(x)H_{t-1}(x)})}\right] = \arg\max_h \mathbb{E}_{x\sim\mathcal{D}_t}[f(x)h(x)] \tag{2.37}$$

其中$Z_t = \mathbb{E}_{x\sim\mathcal{D}}\left[\frac{1}{1+e^{f(x)H_{t-1}(x)}}\right]$是归一化因子，权重更新规则如下

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x)}{Z_t}\frac{1}{1+e^{f(x)H_{t-1}(x)}} \tag{2.38}$$

显然式 2.38 的更新准则更新样本的权重时，最大权重为 1，类似于 MadaBoost 中的方法，但比 MadaBoost 中的方法更平滑。

BBM (Boosting-By-Majority) [Freund, 1995]是第一个迭代的Boost方法。虽然它对噪声的容错性较好[Aslam and Decatur,1993]，但它要求用户事先设置一些难以理解的参数。BrownBoost [Freund, 2001]是BBM的自适应版本，它继承了BBM的噪声容错能力。源于BBM的累积二项分布的损失函数的思想，BrownBoost对应于布朗运动过程（*Brownian motion process*[Gardiner, 2004], i.e.）

$$l_{bmp}(H_{t-1}+h_t|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[1-erf\left(\frac{f(x)H_{t-1}(x)+f(x)h_t(x)+c-t}{\sqrt{c}}\right)\right] \tag{2.39}$$

参数c定义了整个 boosting 过程需要的时间，t是从 0 迭代到当前步骤的时间，$erf(\cdot)$是误差函数

$$rf(\alpha) = \frac{1}{\pi}\int_{-\infty}^{\alpha} e^{-x^2}dx \tag{2.40}$$

损失函数（2.39）可以扩展为

$$l_{bmp}(H_{t-1}+h_t|\mathcal{D}) \approx \mathbb{E}_{x\sim\mathcal{D}}\begin{bmatrix} 1-erf\left(\frac{f(x)H_{t-1}(x)+c-t}{\sqrt{c}}\right) \\ -\frac{2}{c\pi}e^{-(f(x)H_{t-1}(x)+c-t)^2/c}f(x)h_t(x) \\ -\frac{4}{c^2\pi}e^{-(f(x)H_{t-1}(x)+c-t)^2/c}f(x)^2h_t(x)^2 \end{bmatrix}$$
$$\approx -\mathbb{E}_{x\sim\mathcal{D}}\left[e^{-(f(x)H_{t-1}(x)+c-t)^2/c}f(x)h_t(x)\right] \tag{2.41}$$

则学习器需要最小化的损失函数为

$$h_t = \arg\min_h l_{bmp}(H_{t-1}+h_t|\mathcal{D}) = \arg\max_h \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-(f(x)H_{t-1}(x)+c-t)^2/c}f(x)h_t(x)\right]$$
$$= \arg\max_h \mathbb{E}_{x\sim\mathcal{D}_t}[f(x)h(x)] \tag{2.42}$$

权重更新规则为

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x)}{Z_t}e^{-(f(x)H_{t-1}(x)+c-t)^2/c} \tag{2.43}$$

其中Z_t为归一化项。这里的权重函数$e^{-(f(x)H_{t-1}(x)+c-t)^2/c}$和上面介绍的 Boosting 方法中的不同。当分类间隔$f(x)H_{t-1}(x)$等于$-(c-t)$，从式（2.43）看出此时权重会变得最大，当间隔$f(x)H_{t-1}(x)$变大或者变小时权重$\mathcal{D}_t(x)$都会减小。这表明 BrownBoost/BBM 不会对一些不好训练的数据进行训练。随着迭代训练步骤的增加$-(c-t)$趋向于零。这表明 BrownBoost/BBM 使大多数样本的边界都是正的，让那些不好训练的样本点当作噪声数据。

RobustBoost [Freund, 2009]是对 BrownBoost 的改进，主要是通过提升归一化的分类间隔（这和泛化误差相关见 2.4.2 节）达到改善噪声容错的能力。换句话说，RobustBoost 不是对分类误差进行最小化，而是最小化下式

$$\mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{I}\left(\frac{\sum_{t=1}^{T}\alpha_t f(x)h_t(x)}{\sum_{t=1}^{T}\alpha_t}\leq\theta\right)\right] \tag{2.44}$$

其中θ是目标间隔。此时 BrownBoost 中的 Brownian 的想法变成了 Ornstein-Uhlenbeck process[Gardiner, 2004]中的 mean-reverting，损失函数如下

$$l_{oup}(H_{t-1}+h_t|\mathcal{D})=\mathbb{E}_{x\sim\mathcal{D}}\left[1-erf\left(\frac{\tilde{m}(H_{t-1}(x)+h_t(x))-\mu\left(\frac{t}{c}\right)}{\sigma\left(\frac{t}{c}\right)}\right)\right] \tag{2.45}$$

$\tilde{m}(H)$是归一化的间隔，$0\leq\frac{t}{c}\leq 1$，$\mu\left(\frac{t}{c}\right)=(\theta-2\rho)e^{1-\frac{t}{c}}+2\rho$和$\sigma\left(\frac{t}{c}\right)^2=\left(\sigma_f^2+1\right)e^{2\left(1-\frac{t}{c}\right)}-1$分别是过程的均值和方差，$\rho$，$\sigma_f$和$\theta$都是算法的参数。和 BrownBoost 中的想法类似，RobustBoost 的权重更新公式为

$$\mathcal{D}_t(x)=\frac{\mathcal{D}(x)}{Z_t}e^{-\left(f(x)H_{t-1}(x)-\mu\left(\frac{t}{c}\right)\right)^2/2\sigma\left(\frac{t}{c}\right)^2} \tag{2.46}$$

式（2.46）和式（2.43）的主要区别在于式 2.46 中$\mu\left(\frac{t}{c}\right)$无限逼近$\theta$，而式（2.43）中是$t$无限逼近时间$c$。这样 RobustBoost 归一化的分类间隔就会变得比目标间隔θ大，而 BrownBoost 仅仅是让其分类间隔大于零。

2.7 进一步阅读

见原书

3 Bagging

3.1 两种集成模式

根据基本学习器的产生方式，总的来说，有两种集成方式，即**顺序集成方式**（sequential ensemblemethods）和**并行集成方式**（parallel ensemble methods），顺序集成方式中的基本学习器是按照一定顺序产生的，当前基本学习器和之前集成起来的基本学习器有关系，AdaBoost 是顺序集成模型的代表方法，并行集成方法中，基本学习器的产生是并行产生的，基本学习器之间没有关系，Bagging [Breiman, 1996d]是并行学习器的代表方法。

顺序集成方式的基本思想是利用基本学习器之间的依赖关系，因为学习器的整体性能可以通过减小残差的方式提升，详见第 2 章的介绍。并行集成方式的基本思想是利用基本分类器之间的相互独立性，因为可以通过将相互独立的分类器组合起来达到减小误差的目的。

以二分类为例说明**并行集成方式**，类别为$\{-1,1\}$。假设真实函数为f，每个基本学习器都**有独立的泛化误差ϵ**，即对于每个基本学习器h_i

$$P\big(h_i(x) \neq f(x)\big) = \epsilon \tag{3.1}$$

根据下式将T个基本学习器组合起来

$$H(x) = sign\big(\sum_{i=1}^{T} h_i(x)\big) \tag{3.2}$$

当有一半以上的基本学习器h_i发生错误分类时，集成起来的H才会发生错误分类。因此按照 **Hoeffding** 不等式，集成起来的分类器的泛化误差为

$$P\big(H(x) \neq f(x)\big) = \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon) \leq \exp\left(-\frac{1}{2}T(2\epsilon-1)^2\right) \tag{3.3}$$

式（3.3）表明泛化误差随着T的增加以指数形式减小，随着T的增加泛化误差趋近与 0。实际上无法得到真正相互独立的基本学习器，因为这些学习器都是由相同的训练数据训练的，通过随机采样技术可以让学习器之间的依赖性降低，这样得到的集成学习器有较好的泛化能力。

并行集成模式的另一个有点是它与生俱来的并行计算能力，利用多核处理器或者并行计算机可以很方便的对并行集成模式进行加速计算。在当今的计算技术下这很容易实现。

3.2 Bagging 算法

Bagging 是 *Bootstrap AGGregatING*[Breiman, 1996d]的缩写。从名字中可以看出 Bagging 有两个要素：自助法（bootstrap）和集成。

我们知道将相互独立的分类器组合起来可以显著的减少学习误差，所以应该尽可能的获得独立的基本分类器。给定一个训练数据集，一种可能的做法是从这个数据集中采样出一些不重叠的数据子集，在每个子集上训练分类器。但是由于我们的数据集都是有限的所以采样出的数据集可能非常小，这样就不具有代表性，用这些数据集训练处的基本分类器的性能也就会很差。

Bagging 方法用 bootstrap 分布训练不同的基本学习器。换句话说 Bagging 利用 bootstrap sampling [Efron and Tibshirani,1993]技术采样子数据集，并用这些子数据集训练基本的分类器。具体来说，给定一个训练数据集，包括m个训练样本，通过可放回采样方式进行采样。在采样过程中有些原始样本可以出现多次，有的可能一次都不出现。通过T次采样得到T个样本子集。然后在每个样本子集上训练基本分类器。

Bagging 方法采样非常流行的策略处理每个基本学习器的输出结果，即对于分类问题用每个基本分类器的结果进行投票，投票的结果即为最终结果；回归问题采用平均的方式处理每个基本分类器的输出结果，将平均值作为最终的集成分类器的输出结果。以分类问题为例，对一个样本进行预测时，Bagging 将该样本输入到所有的已经训练好的基本分类器中，记录所有基本分类器的输出结果，然后用投票的方式得出最终的分类结果，即在分类结果中看哪个类别的个数最多，最终结果就是哪个类。Bagging 算法可以处理二分类问题也可以处理多分类问题。Bagging 算法见图 3.1。

Breiman [1996d]指出，在m样本的训练集中，第 i 个样本的被选中$0,1,2,\cdots$次，这符合参数$\lambda = 1$的泊松分布，所以第i个样本至少被选中一次的概率为$1 - \frac{1}{e} \approx 0.632$。换句话说，Bagging 中的每个学习器在学习过程中，约有 36.8%的原始样本不予使用。基本学习器的好坏可以用这 36.8%的样本（out-of-bag）进行评估。因此可以估计出集成学习器的泛化误差[Breiman, 1996c, Tibshirani, 1996b, Wolpert and Macready,1999]。

输入： 数据集$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$
最小二乘学习算法$\mathfrak{L}$
学习分类器的个数T

训练过程：

1 **for** $t = 1, \cdots, T$:
2 $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$； %$\mathcal{D}_{bs}$是 bootstrap 分布
3 **end**

输出： $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y)$

图 3.1 Bagging 算法

为了得到 **out-of-bag** 的估计，需要将用于训练每个基本学习器的样本记录下来。$H^{oob}(x)$表示x的 **out-of-bag** 预测，它只包括没有用x进行训练的学习器，即如下

$$H^{oob}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y) \cdot \mathbb{I}(x \notin D_t) \quad (3.4)$$

则用 out-of-bag 数据估计 Bagging 的泛化误差公式为

$$err^{oob} = \frac{1}{|D|} \sum_{(x,y) \in D} \mathbb{I}\left(H^{oob}(x) \neq y\right) \quad (3.5)$$

out-of-bag 还可以有其它用处，例如，当选用决策树为基本学习器时，每棵树的每个节点的后验概率可以用 out-of-bag 数据进行估计，如果一个节点不包括 out-of-bag 样本，将被标记为“uncounted”。对于一个测试样本，它的后验概率可以用它所在节点的后验概率平均值表示。

bootstrap 算法

3.3 示例

为了对 Bagging 有个直观的认识，在三个高斯分布的数据集上分别画出单棵分类树的决策边界，Bagging 算法的决策边界，由十棵树组成的 Bagging 算法的决策边界，详见图 3.2,。可以发现 Bagging 的决策边界比单棵树的决策边界更多变，这使单棵树的误差从 9.4%降低到 Bagging 方法的 8.3%。

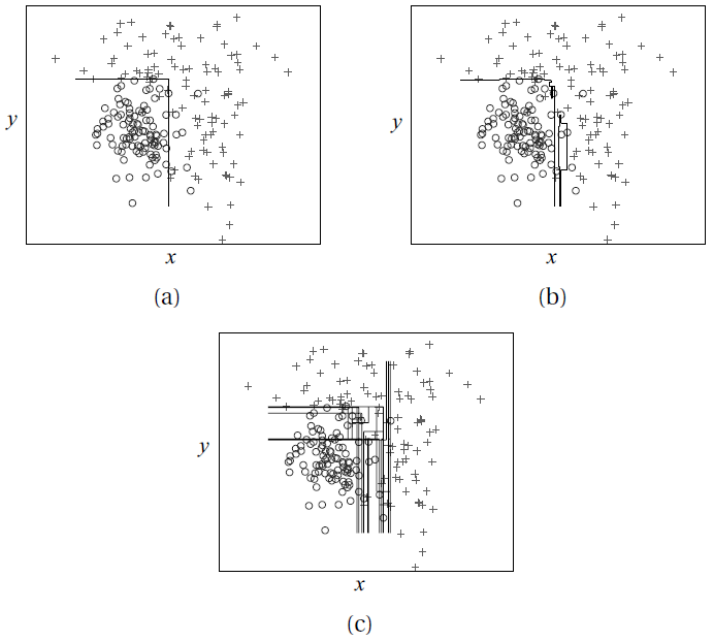


图 3.2 （a）单棵树的决策边界，（b）Bagging 的决策边界，（c）十棵树组成的 Bagging 的决策边界

在 40 天的 UCI 机器学习数据集上对 Bagging 算法进行评价。使用 Weka 实现的由 20 个基本分类器的 Bagging 算法，基本学习器包括：决策树桩，剪枝和未剪枝的 J48 决策树。图 3.3 是对比结果，从中可以看出 Bagging 算法的分类性能要好于其使用的基本分类器的性能，基本上没有比它所使用的基本分类器性能差的情况。

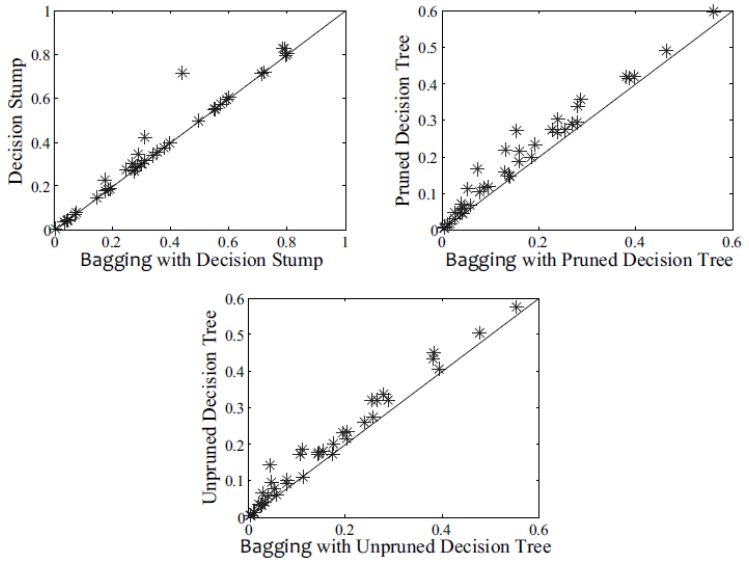


图 3.3 在 40 天的 UCI 数据集上 Bagging 算和其使用的基本分类器性能对比。每个点表示一个数据集，数据点的位置表示两种方法的预测误差，对角线表示两种方法的预测误差相等

进一步研究发现使用决策树桩作为基本学习器的 Bagging 没有使用决策树作为基本学习器的 Bagging 算法效果好。这种现象在图 3.4 中更容易发现。

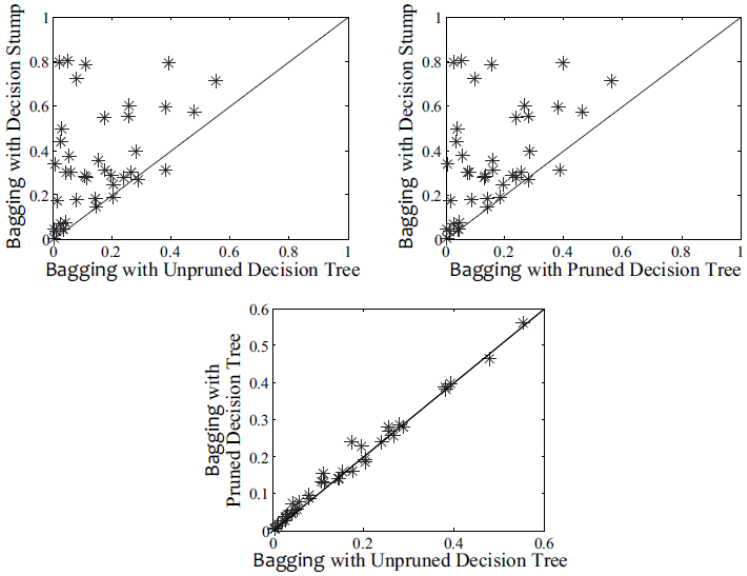


图 3.4 使用决策树桩，剪枝过的决策树，未剪枝的决策树作为基本学习器的 Bagging 方法预测误差对比。每个点代表一个数据集，数据点的位置表示两种方法的预测误差，对角线表示两种预测误差相等

Bagging 方法采样 bootstrap 采样技术产生不同的数据样本，采出来的所有样本集合与原始数据集的重复度为 63.2%。如果一个基本学习器对训练数据集之间的差异不是很敏感，则用这些训练样本集合训练出的不同学习器之间差异也就会很小，因此它们的组合分类器的泛化性能也不会增强。这种类型的学习器称之为稳定学习器（**stable learners**）。决策树是非稳定学习器，而决策树桩更趋向于稳定学习器。在十分稳定的学习器（如 k 近邻学习器）上 Bagging 算法将会失效。图 3.5 中是最近邻分类器的决策边界和由最近邻分类器组成的 Bagging 方法的决策边界。从图中很难发现这两者的决策边界有何差异，而且两者的预测误差都为 9.1%。

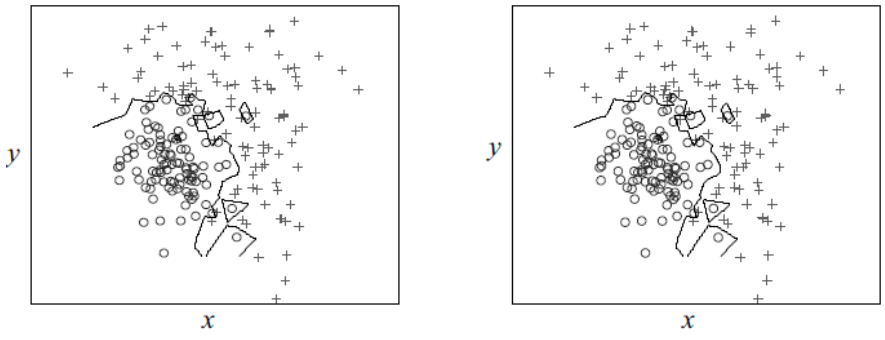
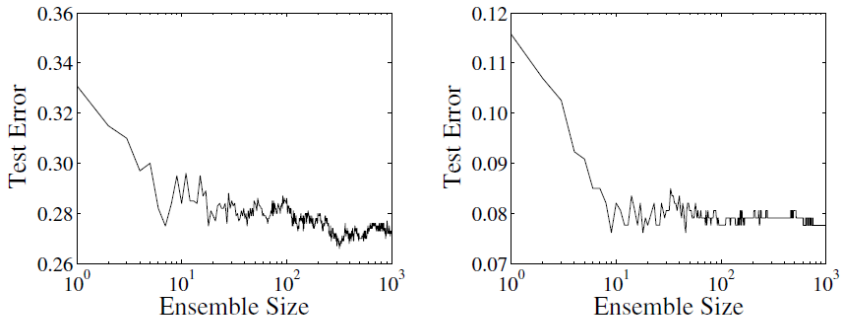


图 3.5 （a）最近邻分类器的决策边界，（b）用最近邻分类器作为基本学习器的 Bagging 方法的决策边界

众所周知，Bagging 方法通常用于非稳定的基本学习器，总的来说，基本学习器越不稳定则 Bagging 的分类器性能越高。从图 3.4 中可以看出这种现象的存在，未剪枝的决策树 Bagging 方法的分类性能要高于剪枝后的决策树 Bagging 方法。因为未剪枝的决策树比剪枝后的决策树更不稳定。这表明当使用决策树作为 Bagging 方法的基本分类器时不需要对其进行剪枝，剪了反而影响 Bagging 的分类效果。

式（3.3）表明由独立的基本分类器组成的 Bagging 方法的泛化误差和基本分类器的总个数T（也是从原始数据集中 bootstrap 采样出的子数据集的个数）呈指数衰减关系，随着T趋向于无穷，泛化误差最终趋向于 0。而在实际中无法提供无穷多个用于训练基本分类器的子集，所以T是有限的，同时 Bagging 中的每个基本分类器也不可能相互独立，因为用于训练它们的子数据集是用 bootstrap 方法从原始的同一份数据中采样得到的。虽然 Bagging 方法的泛化误差不可能为零，但是 Bagging 方法的泛化误差还是会随着T的增加而收敛，即随着基本学习器的个数增加 Bagging 方法的泛化误差会最终收敛。详见图 3.6。



(a) credit-g　　(b) soybean

图 3.6 在两个 UCI 数据集上 Bagging 的参数T与测试误差的关系

3.4 理论问题

Bagging 方法能够明显的降低方差，同时在不稳定的学习器上非常有效。理解 Bagging 方法的这些性质是研究 Bagging 理论的基础。

Breiman [1996d]在提出 Bagging 方法的时候，给出了理论分析。首先考虑回归问题。f表示真值函数，$h(x)$表示由 bootstrap 采样中得到的$\mathcal{D}_{bs}$数据集训练出的学习器，通过 Bagging 方法的组合学习器为

$$H(x) = \mathbb{E}_{\mathcal{D}_{bs}}[h(x)] \tag{3.6}$$

使用简单的代数运算和不等式关系$(\mathbb{E}[X])^2 \leq \mathbb{E}[X^2]$，得到如下不等式

$$\big(f(x) - H(x)\big)^2 \leq \mathbb{E}_{\mathcal{D}_{bs}}\Big[\big(f(x) - H(x)\big)^2\Big] \tag{3.7}$$

对两边进行积分，可以得到$H(x)$的均方误差小于$h(x)$在 bootstrap 采用分布上的平均值，得到的不等式如下

$$\big(\mathbb{E}_{\mathcal{D}_{bs}}[h(x)]\big)^2 \leq \mathbb{E}_{\mathcal{D}_{bs}}[h(x)^2] \tag{3.8}$$

这表明基本分类器的不稳定性是非常重要的。也就是说如果$h(x)$在不同的 bootstrap 采样上性能没有多大差异，则组合起来的学习器性能也不会有提升。这解释了为什么不稳定的学习器组成的 Bagging 算法如此高效，同时它能通过平滑作用减少方差。

在分类问题中，假设$h(x)$的预测类别是$y \in \{y_1, y_2, \cdots, y_c\}$。$P(y|x)$表示$x$被分为类别$y$的概率，则$h(x)$正确分类的概率为

$$\sum_y P(h(x) = y)P(y|x) \tag{3.9}$$

整体分类正确率为

$$\int \sum_y P(h(x) = y)P(y|x)P(x)\, dx \tag{3.10}$$

$P(x)$是输入样本的概率分布。

如果输入样本x被分为类别y的概率大于被分为其它类别的概率，则h也会把x预测为类别y，即

$$\arg\max_y P(h(x)=y) = \arg\max_y P(y|x) \tag{3.11}$$

h被称为x的序正确（*order-correct*）。

Bagging 组合分类器$H(x)=\arg\max_y P(h(x)=y)$的正确分类的概率为

$$\sum_y \mathbb{I}(\arg\max_z P(h(x)=z)=y)P(y|x) \tag{3.12}$$

如果h在输入x上是序正确（*order-correct*），则以上概率等于$\max_y P(y|x)$。则组合分类器H正确分类的概率是

$$\int_{x\in C}\max_y P(y|x)P(x)dx + \int_{x\in C'}\left[\sum_y \mathbb{I}(H(x)=y)P(y|x)\right]P(x)dx \tag{3.13}$$

C是样本x的集合，h是序正确的，C'是C的补集。h在C'不是序正确的。所以有下式成立

$$\sum_y P(h(x)=y)P(y|x) \le \max_y P(y|x) \tag{3.14}$$

则 Bagging 的最优准确率为

$$\int \max_y P(y|x)P(x)dx \tag{3.15}$$

上式等于贝叶斯误差率。

对比式（3.10）和式（3.13）发现如果预测函数是序正确的，则 Bagging 可以使序正确集的预测函数的正确率达到最优，单独的预测函数则无法达到这一点。如果基本的学习器是不稳定的，则从样本子集训练出的各个h之间将会有很大的差异，对x的预测结果也将不同，则$P(h(x)=y)$的概率也会变小。由式（3.9）可知对 x 的正确分类概率会变小。但是如果h在x上是序正确的，Bagging 对x的正确分类概率将会变大。这说明当基本学习器是不稳定的且是序正确时，Bagging 将会使组合学习器的性能有巨大的提升。

Friedman and Hall [2007]从统计的观点对预测函数进行分解，研究 Bagging 算法。他们假设学习器$h(x;\gamma)$是参数化的，参数为γ，γ可以求解下式获得

$$\sum_{i=1}^{n} g\big((x_i,y_i),\gamma\big)=0 \tag{3.16}$$

g是平滑的多变量函数，(x_i,y_i)是第i个训练样本，n是训练样本集合的大小，一旦求出γ，则h就可以确定。

假设γ^*是$\mathbb{E}_{x,y}\big[g\big((x_i,y_i),\gamma\big)\big]=0$的解。在$\gamma^*$附近将$g\big((x_i,y_i),\gamma\big)$展开，则式（3.16）可以重写为

$$\sum_{i=1}^{n}\Bigg[g\big((x_i,y_i),\gamma^*\big)+\sum_k g_k\big((x_i,y_i),\gamma^*\big)(\gamma-\gamma^*)_k + \sum_{k_1}\sum_{k_2} g_{k_1,k_2}\big((x_i,y_i),\gamma^*\big)(\gamma-\gamma^*)_{k_1}(\gamma-\gamma^*)_{k_2}+\cdots\Bigg]=0 \tag{3.17}$$

γ_k是γ的k个元素，g_k是g关于γ_k的偏导数。假设$\hat{\gamma}$是式（3.16）的解，再结合式（3.17），$\hat{\gamma}$可以表示为

$$\hat{\gamma}=\Gamma+\sum_{k_1}\sum_{k_2}\alpha_{k_1,k_2}(\bar{\Phi}-\phi)_{k_1}(\bar{\Phi}-\phi)_{k_2} + \sum_{k_1}\sum_{k_2}\sum_{k_3}\alpha_{k_1,k_2,k_3}(\bar{\Phi}-\phi)_{k_1}(\bar{\Phi}-\phi)_{k_2}(\bar{\Phi}-\phi)_{k_3}+\cdots \tag{3.18}$$

α_{k_1,k_2}，α_{k_1,k_2,k_3}为系数。

$$\Gamma=\gamma^*+M^{-1}\frac{1}{n}\sum_{i=1}^{n} g\big((x_i,y_i),\gamma^*\big) \tag{3.19}$$

$$\bar{\Phi} = \frac{1}{n}\bar{\Phi}_i \quad \phi = \mathbb{E}[\bar{\Phi}_i] \tag{3.20}$$

M是矩阵，它的k列是$\mathbb{E}_{x,y}\left[g\left((x_i,y_i),\gamma^*\right)\right]$，$\bar{\Phi}_i$是向量其元素为$g\left((x_i,y_i),\gamma^*\right)$，$g_{k_1,k_2}\left((x_i,y_i),\gamma^*\right)$，…。显然$\hat{\gamma}$可以分解为线性项和高阶项。

假设由 bootstrap 样本训练的学习器由参数$\hat{\gamma}'$控制，样本个数为$m(m<n)$。由式（3.18）可得

$$\hat{\gamma}' = \Gamma' + \sum_{k_1}\sum_{k_2}\alpha_{k_1,k_2}\left(\bar{\Phi}' - \phi\right)_{k_1}\left(\bar{\Phi}' - \phi\right)_{k_2} + \sum_{k_1}\sum_{k_2}\sum_{k_3}\alpha_{k_1,k_2,k_3}\left(\bar{\Phi}' - \phi\right)_{k_1}\left(\bar{\Phi}' - \phi\right)_{k_2}\left(\bar{\Phi}' - \phi\right)_{k_3} + \cdots \tag{3.21}$$

Bagging 的组合学习器的参数为

$$\hat{\gamma}_{bag} = \mathbb{E}[\hat{\gamma}'|D] \tag{3.22}$$

如果$\hat{\gamma}$是线性的，则有$\mathbb{E}\left[\Gamma'|D\right] = \Gamma$，因此$\hat{\gamma}_{bag} = \hat{\gamma}$。这说明 Bagging 没有改善$\hat{\gamma}$中的线性项。

下面来看高阶项

$$\rho_m = \frac{n}{m} \tag{3.23}$$

$$\hat{\sigma}_{k_1,k_2} = \frac{1}{n}\sum_{i=1}^{n}(\bar{\Phi}_i - \bar{\Phi})_{k_1}(\bar{\Phi}_i - \bar{\Phi})_{k_2} \tag{3.24}$$

$$\mathrm{S} = \sum_{k_1}\sum_{k_2}\alpha_{k_1,k_2}\hat{\sigma}_{k_1,k_2} \tag{3.25}$$

Friedman and Hall [2007]证明，如果$\rho_m \to \rho(1 \le \rho \le \infty)$，$n \to \infty$，$\hat{\gamma}_{bag}$为

$$\hat{\gamma}_{bag} = \Gamma + \frac{1}{n}\rho_m S + \delta_{bag} \tag{3.26}$$

δ_{bag}表示高于二阶的高阶项。从式（3.24）可知，随着样本个数n的增加，$\hat{\sigma}_{k_1,k_2}$的方差将变小，and the dependence of the variance on the sample size is in the orderO(n−1)。因为S是$\hat{\sigma}_{k_1,k_2}$的线性组合，the dependence of the variance of S onthe sample size is also in theorderof O(n−1)。ρ_m渐近于一个常数，由方差的属性$var(aX) = a^2 var(X)$得，the dependence of the variance of 1n ρ mS on the sample size is in the order of O(n−3)。如果重写式（3.18）$\hat{\gamma} = \Gamma + \Delta$，可以得到类似的结果，$\Delta$

Friedman and Hall [2007]证明 Bagging 可以降低高阶项的方差，但对线性项没有影响。这表明 Bagging 更适用于高度的非线性学习器，因为高度非线性学习器更不稳定，即非线性学习器的性能随着训练数据集的不同变化很大。

很容易理解当T增大时 Bagging 收敛加快。给定一个训练数据集，Bagging 使用 bootstrap 采样技术获得用于训练基本学习器的数据子集。这一过程等价于按照 bootstrap 采样分布，从由基本学习器组成的集合中随机抽取一些学习器样本。当对样本进行预测时基本学习器的输出可以看成是一个与训练数据有相同分布的随机变量Y。以二分类问题为例$Y \in \{-1, +1\}$。Bagging 用投票的方式将基本学习器的结果组合起来，首先对结果进行平均$\bar{Y}_T = \frac{1}{T}\sum_{i=1}^{T} Y_i$，$\mathbb{E}[Y]$表示期望。根据大数定理得到

$$\lim_{T\to\infty} P(|\bar{Y}_T - \mathbb{E}[Y]| < \epsilon) = 1 \tag{3.27}$$

当$\mathbb{E}[Y] \neq 0$时将上式转化为投票的方式

$$\lim_{T\to\infty} P(\mathrm{sign}(\bar{Y}_T) = \mathrm{sign}(\mathbb{E}[Y]) < \epsilon) = 1 \tag{3.28}$$

所以随着T的增加，Bagging将收敛到稳定的误差率（除了Bagging算法等价于随机猜测的情况）。实际上所有并行的集成方法都具有这样的性质，即随着T的增加，并行的集成算法收敛到稳定的误差率。

3.5 随机集成树

3.5.1 随机森林

随机森林（RF [Breiman, 2001]）是非常有代表性的集成方法。它是Bagging算法的扩展。随机森林与Bagging的主要差别在于，随机森林中使用了随机特征选择算法。在构建决策树的过程中，在每次选择拆分特征时，随机森林从这个特征空间中随机选择出特征子集，然后在特征子集上寻找用于拆分树的特征。K控制随机性的大小。当K值等于特征个数时构建的随机决策树和传统方法构建的决策树是完全一样的，当$K = 1$时，每个特征的选取将是完全随机的。K的建议取值为特征个数的对数值[Breiman, 2001]。注意随机选择只针对特征，而不是在特征上对剖分点进行随机处理。

图3.8对比了RF，Bagging和它们的基本学习器的决策边界。可以看出RF和它的基本学习器的决策边界更加多变，所以其泛化能力也较强。在*three-Gaussians*数据集上RF的测试误差为7.85%，而Bagging算法的测试误差为8.3%。图3.9对比了RF和Bagging算法在40个UCI数据集上的测试误差。可以看出，无论使用的基本决策树剪枝与否，RF的性能明显好于Bagging算法。

RF的收敛特性和Bagging类似，收敛特性见图3.10，从图中看出当树的个数较少时RF的性能要比Bagging差，极端情况，当RF中树的个数只有一个时，RF退化成一棵对特征随机采样产生的随机树，所以测试误差变的很大。但是对着随机树的个数增加测试误差会迅速降低。值得一提的是RF的训练比Bagging方法更高效。这是因为在构建树的过程中，Bagging使用确定的决策树，这就需要计算所有特征寻找最佳剖分点，而RF使用随机采样选取特征，所以只需要从部分特征中选取出最佳剖分点。

输入：数据集$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$
特征子集的个数K
训练过程：
1 N ←在数据集D上创建一个树节点。
2 如果所有样本都属于同一个类别则返回N 。
3 $\mathcal{F}$ ←可以被进一步分割的特征集合。
4 如果$\mathcal{F}$为空则返回N。
5 $\tilde{\mathcal{F}}$ ←从$\mathcal{F}$中随机选择K个特征。
6 $N.f$ ←在$\tilde{\mathcal{F}}$中选出最优的剖分特征。
7 $N.p$ ←在$N.f$中的最优剖分点。
8 D_l ←特征$N.f$上取值小于$N.p$的那些样本集合。
9 D_r ←特征$N.f$上取值大于等于$N.p$的那些样本集合。
10 N_l ←参数为(D_l, K)的调用过程。
11 N_r ←参数为(D_r, K)的调用过程。
12 返回N。
输出：随机决策树

图3.7为RF的随机决策树算法。

注意用于训练每棵树的数据集是从原始数据集合中有放回的抽样出来的

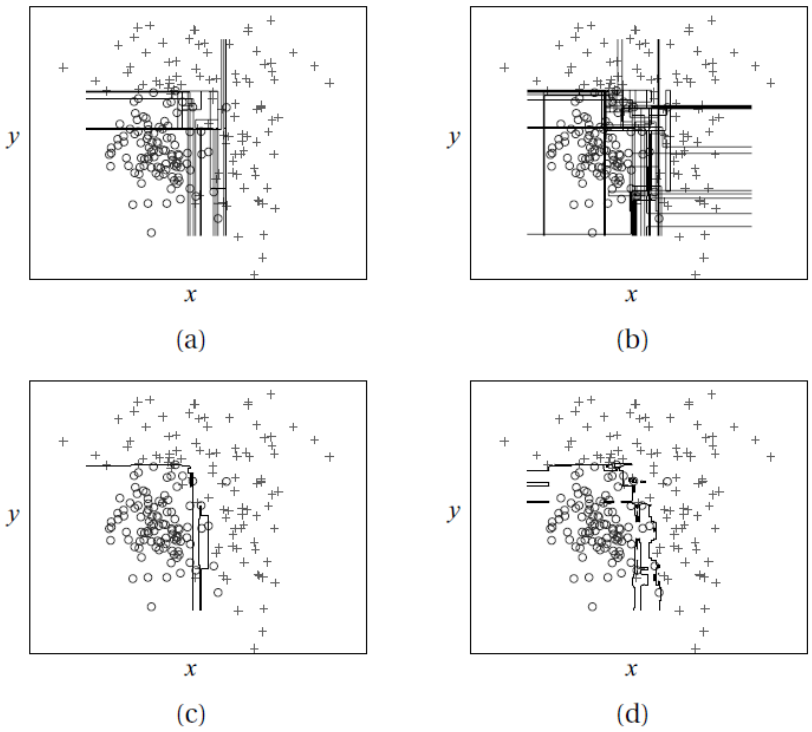


图 3.8 （a）Bagging 的十个基本分类器，（b）RF 的十个基本分类器，（c）Bagging 算法，（d）RF 算法在 *three-Gaussians* 数据集上的决策边界

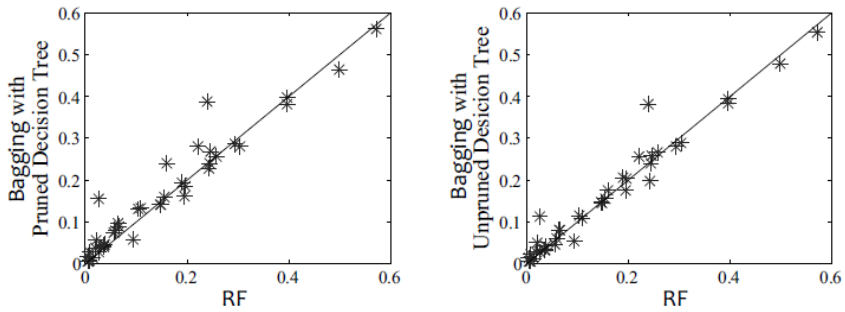


图 3.9 在 40 个 UCI 数据集上 RF 个 Bagging 的预测误差对比，每个点代表一个数据集，点的位置代表两种算法的测试误差，对角线表示测试误差相等

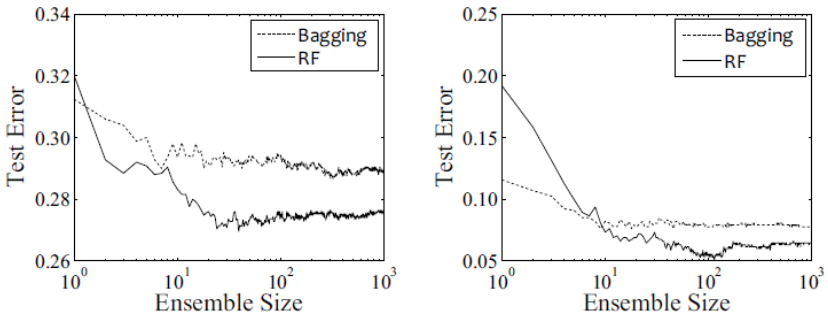


图 3.10 在 UCI 数据集上 RF 和 Bagging 方法中基本学习器的个数对测试误差的影响

3.5.2 随机谱

从每个节点数据中选择特征子集，在特征子集上按照传统方法选择最优剖分点，以这种方式创建随机树。Liu et al. [2008a]发表了 VR-Tree 组合方法，该方法在特征选择和最优剖分点选择上都采用了随机化技术。

VR-Tree 是 VR-Tree 组合方法的基本学习器。在每个节点处，通过抛硬币（也就是贝努利实验）的方式决定最佳剖分点的计算方式，硬币正面朝上的概率为α。如果硬币正面朝上则采用传统的计算最佳剖分点的方式计算最佳剖分点，即在该节点处所有的特征中计算最佳剖分点（不进行特征子集的采样抽取），如果硬币的反面朝上则在整个特征中随机采样获得特征子集，在特征子集上随机抽取一个点最为最佳剖分点。图 3.11 描述了 VR-Tree 算法。

输入： 数据集$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$
参数α用于控制是否用传统方法选择节点上的最佳剖分点
训练过程：
1 N ←在数据集D上创建一个树节点。
2 如果所有样本都属于同一个类别则返回N 。
3 $\mathcal{F}$ ←可以被进一步分割的特征集合。
4 如果$\mathcal{F}$为空则返回N。
5 r ←从[0,1]均匀分布中采样一个随机数。
6 if $r < \alpha$
7 then $N.f$ ←将$\tilde{\mathcal{F}}$中所有特征都选取出来。
8 $N.p$ ←在$N.f$中的最优剖分点。
9 else $N.f$ ←在$\tilde{\mathcal{F}}$中随机选择特征子集。
10 $N.p$ ←在$N.f$中最忌选择剖分点。

11 D_l ←特征$N.f$上取值小于$N.p$的那些样本集合。
12 D_r ←特征$N.f$上取值大于等于$N.p$的那些样本集合。
13 N_l ←参数为(D_l, α)的调用过程。
14 N_r ←参数为(D_r, α)的调用过程。
15 返回N。
输出： VR-Tree

图 3.11 VR-Tree 算法

参数α控制着随机程度，当$\alpha = 1$时 VR-Tree 就是一颗传统的决策树，当$\alpha = 0$时 VR-Tree 完全是随机的。通过调节参数α的值可以得到一个随机谱图[Liu et al.,2008a]见图 3.12。从谱图中可以看出随机性对组合方法性能的影响。谱图中有两个极端情况，完全随机即$\alpha = 0$，完全确定，即$\alpha = 1$。当$\alpha = 0$时树有很强的多样性，而且生成的树很大，当$\alpha = 1$时生成的树精度很高，树也较小。虽然两种极端情况下有完全不同的特性，但是可以通过将参数α向谱图的中间部位移动改善组合方法的性能。在实际应用中是很难知道谱图中的哪个点是最优的。Liu et al. [2008a]发表的 Coalescence 方法中将α设为随机变量，α服从[0,0.5]的均匀分布，这样构建 VR-Tree 时α的选择就是随机的。实验证明 Coalescence 方法中采样的随机的α能够使其性能比采用固定α的 RF 和 VR-Tree 组合学习器的性能要高。

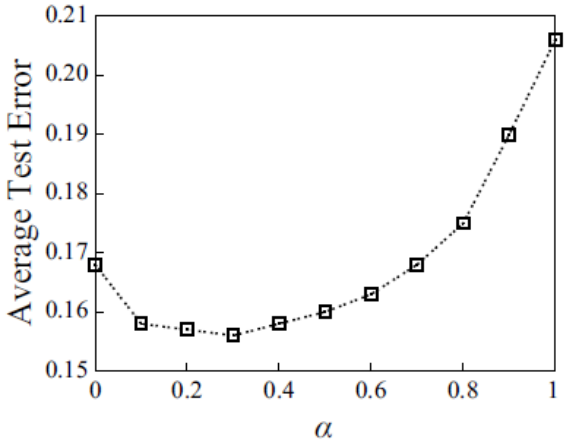
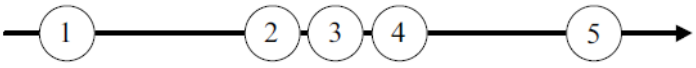


图 3.12 随机谱图，横轴为α，纵轴为在 45 个 UCI 数据集上的 VR-Tree 组合学习器的平均测试误差

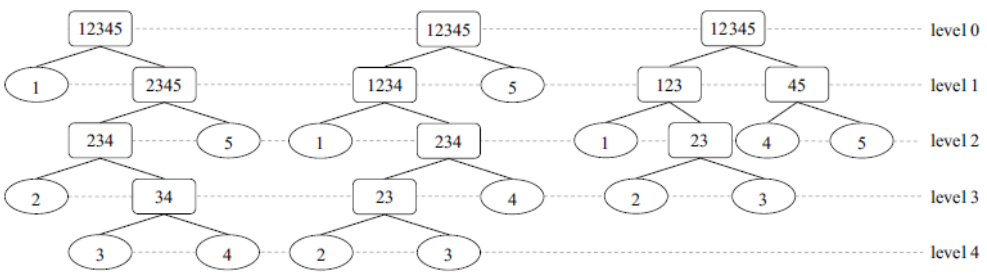
3.5.3 随机树集成算法估计概率密度

可以用随机树集成算法估计概率密度[Fan et al.,2003, Fan, 2004, Liu et al., 2005]。因为概率密度估计是无监督学习，训练数据上没有类别标签，因此可以采用完全随机树。完全随机树不考虑样本所属于的类别，生成的完全随机树每个叶子节点都包括一个样本。将算法 3.11 中的第二步换成“只有一个样本时”，并去掉第五步和第八步，就可以得到构建完全随机树的算法了。

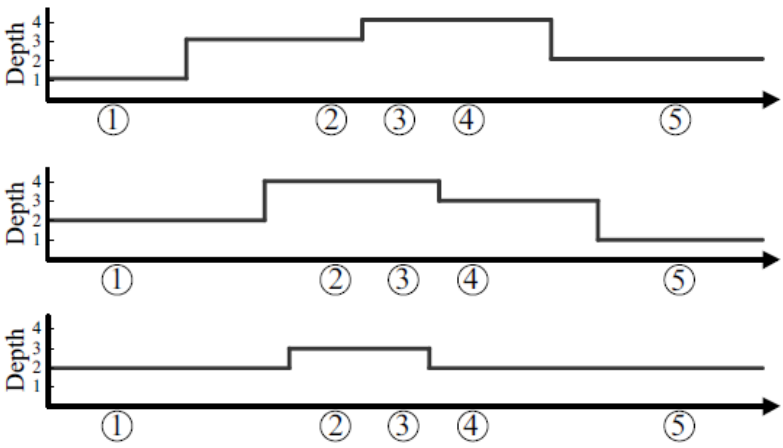
图 3.13 用完全随机树的集成算法估计概率密度。图 3.13（a）画出了五个一维数据点，分别标记为 1,2,3,4,5。当每个叶子节点上只有一个样本点时完全随机树停止生长。首先从 1 至 5 的点中随机选取一个，用这个点把 1 至 5 的数据拆分成两组。从（a）的分布中可以看出，拆分点可以再 1 和 2 之间或者在 4 和 5 之间，因为 1 和 2，4 和 5 之间距离较长。假设剖分点选在 1 和 2 之间，这样 1 就变成了叶子节点，则下一次的的剖分点就很有可能在 4 和 5 之间，如果剖分点选择 5，则 5 就变成了叶子节点，这表明 1 和 5 很容易成为深度较小的叶子节点，而 2 和 4 更容易成为深度较大的叶子节点。图 3.13（b）中画出了生成的三棵完全随机树，从中可以得到每个点的平均深度为：1 和 5 的平均深度为 1.67，2 的平均深度为 3.33，3 的平均深度为 3.67，4 的平均深度为 3。即使只有三棵随机树，也可以看出 1 和 5 位于相对稀疏的区域，2 和 4 位于相对密集的区域，图 3.13（d）画出了概率密度估计的结果，以 1 为例，其概率密度值的计算方法为 $1.67/1.67 \times 2 + 3.33 + 3.67 + 3$。



(a) Five one-dimensional data points



(b) Three completely random trees



(c) The depth of leaves on the one-dimensional data, each corresponding to a random tree in the sub-figure (b)

(d) The density estimation result

对一维数据的这种处理方式也适用于高维数据和更复杂的数据分布。通过逐步的组合学习到的随机树，也可以将此种方法扩展到在线学习或者流数据上。值得一提的是构造完全随机树的过程是非常高效的，因为只要随机采样就可以构造出随机树了。总的来说只要构建一系列的完全随机树，然后计算每个样本点的平均深度，就可以计算出概率密度分布了，在实际应用中这将是一个非常高效的工具。

3.5.4 随机集成树异态检测

异常点是指那些与大部分数据点的行为不同的数据点，异常点检测的任务就是要从整体的数据集中找到这些异常点[Chandola et al., 2009]。通常来说异常点和极端值点可以互换使用的。有很多方法可以用于异常检测[Hodge and Austin, 2004, Chandola et al., 2009]。其中一种方法是通过估计数据的概率密度分布，然后将概率密度很低的点作为异常点。然而 Liu et al. [2008b]指出概率密度方法无法很好的检测异常点，因为一小族异常点数据的概率密度可能很高，而异常点周围的正常数据的概率密度可能很低。因为异常点的基本特性不好描述，当异常点一般都是孤立的，这一特性能够更好的描述异常点[Liu et al., 2008b]。因为可以采用随机树进行异常点检测。图 3.14 是用随机树进行异常点检测过程。从中可以看出正常的数据点需要多次剖分才能成为独立的数据点，对于一个异常点只要经过很少的几次剖分就可以变成一个独立的数据点。

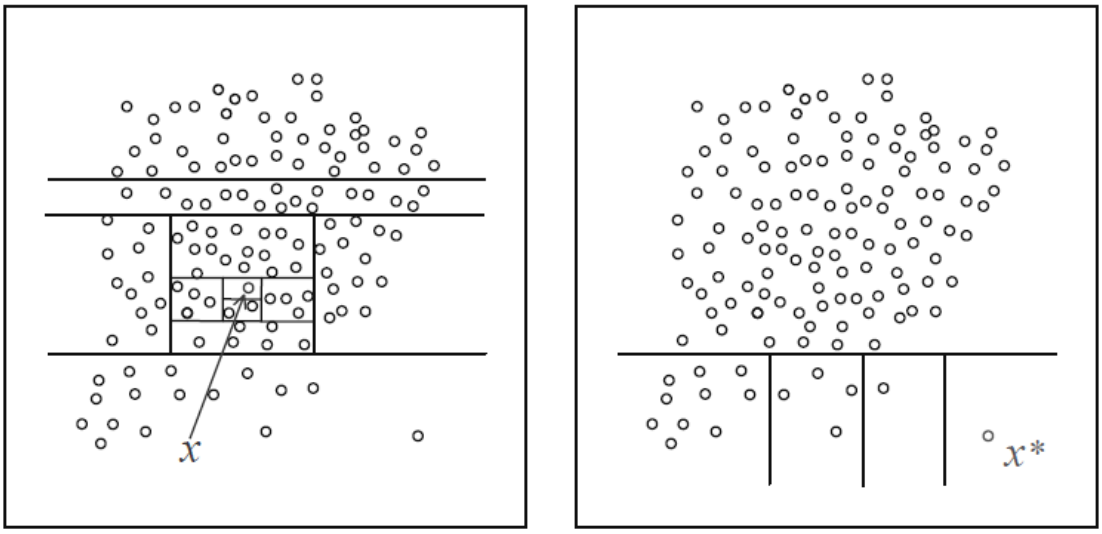


图 3.14 通过随机树算法的异常点检测。（a）正常数据点需要进行 11 次剖分才能变成独立的点，（b）异常点只需要四次剖分就可以变成独立的点

Liu et al. [2008b]用 iForest 方法进行异常点检测。对于每棵随机树，每个独立的数据点被剖分的次数等于该独立数据点到根节点的深度。被剖分的次数越少的数据点越有可能成为异常点。显然异常检测只关心那些有最短路径的点。因此为了减少不必要的计算，iForest 中用到的随机树有高度限制。可以将图 3.11 中的算法第二步改为“只有一个样本或者树的高度达到了特定的阈值”并去掉第五步和第八步，这样得到的算法就是 iForest 中的随机树。考虑到随机树在大数据集上的性能和可扩展性，应该从原始数据集中采样出一部分数据，在采样的子数据集上构建随机树。假设数据集中样本个数为ψ，则 iForest 的高度被限制为$\lceil \log_2(\psi) \rceil$，$\lceil \log_2(\psi) \rceil$是$\psi$个叶子节点的平均高度[Knuth, 1997]。

为了计算异常点x_i的异常得分$s(x_i)$，将x_i输入 iForest 集成方法中的每棵随机树，然后计算在这些树上x_i的路径均值$\mathbb{E}[h(x_i)]$。从每棵随机树上获得的路径长度都要用$c(n)$调节。对于一棵有n个节点的树，$\mathrm{c}(n)$为路径的平均长度[Preiss, 1999]

$$\mathrm{c}(n) = \begin{cases} 2H(n-1) - (2(n-1)/n), n > 1 \\ 0, n = 1 \end{cases} \quad (3.29)$$

$H(\alpha)$是一个调和数，可以用下式计算

$$H(\alpha) \approx \ln(\alpha) + 0.5772156649 (\text{Euler's constant})$$

$s(x_i)$ 的计算方式为[Liu et al., 2008b]

$$s(x_i) = 2^{-\frac{\mathbb{E}[h(x_i)]}{\mathrm{c}(\psi)}} \quad (3.30)$$

$\mathrm{c}(\psi)$为归一化因子。如果$s(x_i)$非常接近于 1，说明x_i为一个异常点，如果$s(x_i)$远小于 0.5，则x_i为正常数据点，如果对所有的点$s(x_i) \approx 0.5$，则数据集中没有异常点[Liu et al., 2008b]。

Liu et al. [2010]提出了 SCiForest，这是 iForest 的变体。与 iForest 相比，SCiForest 只对与坐标轴平行的特征进行剖分，通过对原始特征的组合得到分割超平面，这样做是为了得到更平滑的边界这种做法与 oblique decision trees 类似。Furthermore, since the hypothesis space is more complicated by considering hyper-planes, rather than using a completely **random manner, a split selection criterion is defined in SCiForest to facilitate the selection of appropriate hyper-planes at each node to reduce the risk of falling into poor sub-optimal solutions.**

3.6 进一步阅读

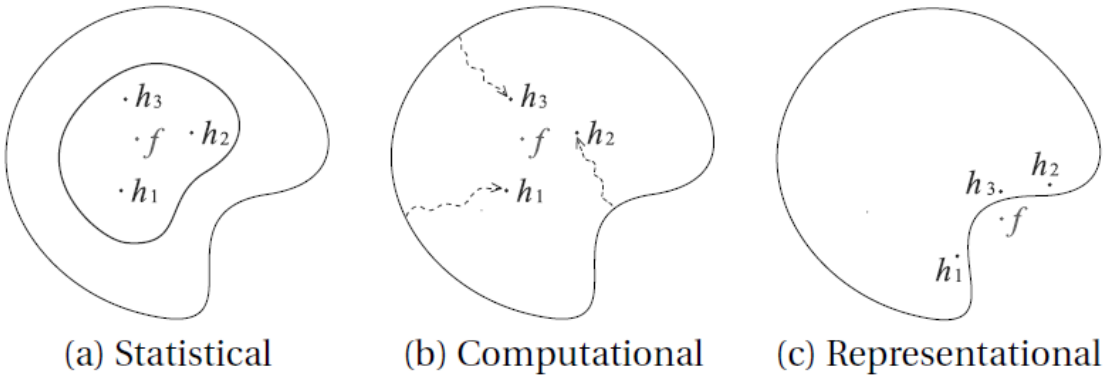
见原文

4 集成方式

4.1 集成的益处

在训练得到一些列的基本学习器后，集成方法不会从这些基本学习器中选取最优的一个学习器最为最终的学习器，而是将这些学习器组合记起来，这样才能达到很强的泛化能力，因此各个基本学习器之间的集成方式变得异常重要。Dietterich [2000a]阐述了集成方法三个优势，如下

- 统计学方面：通常来说假设空间都比较大，训练数据无法描述整个假设空间特性，所以在相同的训练数据上可以对应不同的假设空间，而且这些假设空间在训练数据上的有相同的精度。如果只有一个学习得到一个假设空间，并用得到的假设空间预测未知数据，那么预测错误的风险就很大。如图 4.1（a）所示，如果将这些假设空间集成起来，那么就可以减少对未知数据预测的错误率，即泛化能力变强。
- 计算方面：很多学习器学习时，时常陷入局部最优。即使有充足的训练数据也会发生这种情况，这就导致无法找到最优的假设空间（最优学习器）。从不同的初始点点开始找到局部最优解，将这些局部最优解组合起来能够更好的近似未知的假设空间。如图 4.1 所示，将这些假设空间集成起来，能够降低选择单个错误局部最优解的风险。
- 在假设空间的表示方面：在许多机器学习过程中，实际上无法用任何假设去表示未知的实际假设空间。如图 4.1（c）所示，将各个假设组合起来可以扩展可表示函数的空间，因此学习算法就可以构造更精确的解去近似未知的假设。



(a) Statistical (b) Computational (c) Representational

图 4.1 采用集成方法的三种原因。（a）统计学方面，（b）计算方面，（c） 在假设空间的表示方面。（a）中的外面曲线表示假设空间，里面的曲线表示在训练数据上具有相同精度的假设空间，f表示真实的假设空间，h_i表示单个训练得到的假设空间。

其它机器学习方法在以上介绍的三个方面常常失效。如果一个学习器有统计学方面的问题，通常来说它对应高方差问题（high “variance”），如果一个学习器有计算方面的问题，则它对应为高计算方差问题（high “computational variance”），如果一个学习器有特征表示方面的问题，则其对应于高偏差问题（high “bias”）。因此通过对基本学习器的组合可以同时降低偏差（“bias”）和方差（“variance”）。这已经被很多实际研究证明了。

4.2 平均方法

取平均是最常用也是最基本的对输出的数值结果进行组合的方法。本节将以回归为例介绍如何对数据取平均。假设有T个基本学习器$\{h_1, \cdots, h_T\}$，h_i在样本x上的输出为$h_i(x) \in \mathbb{R}$。下面将要做的就是将所有h_i的预测结果集成起来用于预测变量的真实值。

4.2.1 简单平均

简单平均只需要对每个h_i的输出直接取平均。简单平均公式如下

$$H(x) = \frac{1}{T}\sum_{i=1}^{T} h_i(x) \tag{4.1}$$

假设真实函数为$f(x)$，x的分布为$p(x)$。则每个基本学习器的输出为真实加上一个误差项

$$h_i(x) = f(x) + \epsilon_i(x), i = 1, \cdots, T \tag{4.2}$$

h_i的均方误差为

$$\int \big(h_i(x) - f(x)\big)^2 p(x)dx = \int \epsilon_i(x)^2 p(x)dx \tag{4.3}$$

基本分类器的基本误差为

$$\overline{err}(h) = \frac{1}{T}\sum_{i=1}^{T} \int \epsilon_i(x)^2 p(x)dx \tag{4.4}$$

同理可以得到集成学习器的期望误差

$$err(H) = \int \left(\frac{1}{T}\sum_{i=1}^{T} h_i(x) - f(x)\right)^2 p(x)dx = \int \left(\frac{1}{T}\sum_{i=1}^{T} \int \epsilon_i(x)\right)^2 p(x)dx \tag{4.5}$$

显然有

$$err(H) \le \overline{err}(h) \tag{4.6}$$

也就是说集成学习器的期望误差小于等于基本学习器的平均误差。

如果假设误差项ϵ_i的均值为零且相互独立，则有

$$\int \epsilon_i(x)p(x)dx = 0 \quad \int \epsilon_i(x)\epsilon_j(x)p(x)dx = 0 \tag{4.7}$$

所以不难得到

$$err(H) = \frac{1}{T}\overline{err}(h) \tag{4.8}$$

这说明集成学习器的误差小于单个基本学习器的平均误差是由T引起的。因为集成学习器的误差等于基本学习器平均误差除以T。

鉴于直接平均方法非常简单，有效，因此它的使用范围十分广泛。但是，值得注意的是式（4.8）是基于误差是相互独立的假设的，而在实际的集成学习中误差是高度相关的，因为基本学习器所学习的是同一个问题。因此式 4.8 只能看成是理想情况。

4.2.2 加权平均

通过对每个基本学习器的输出给予不同的权重，然后将它们累加起来，作平均，称为加权平均。加权平均能够改善不同基本学习器的重要性。$H(x)$加权平均的表达式如下

$$H(x) = \sum_{i=1}^{T} \omega_i h_i(x) \tag{4.9}$$

ω_i是h_i的权重，ω_i满足下式

$$\omega_i \ge 0, \quad \sum_{i=1}^{T} \omega_i = 1 \tag{4.10}$$

和 4.2.1 节中类似，假设真值函数为$f(x)$，x的分布为$p(x)$。每个学习器也可以写成式（4.2）的形式。很容易写出集成学习器的训练误差为[Perrone and Cooper, 1993]

$$\begin{aligned} err(H) &= \int \left(\frac{1}{T}\sum_{i=1}^{T} \omega_i h_i(x) - f(x)\right)^2 p(x)dx \\ &= \int \left(\frac{1}{T}\sum_{i=1}^{T} \omega_i h_i(x) - f(x)\right)\left(\frac{1}{T}\sum_{j=1}^{T} \omega_j h_j(x) - f(x)\right) p(x)dx \\ &= \sum_{i=1}^{T}\sum_{j=1}^{T} \omega_i \omega_j C_{ij} \end{aligned}$$

（4.11）

其中

$$C_{ij} = \int \big(h_i(x) - f(x)\big)\Big(h_j(x) - f(x)\Big)p(x)dx \tag{4.12}$$

可以得到最优的权重为

$$\omega = \arg\min_{\omega} \sum_{i=1}^{T}\sum_{j=1}^{T} \omega_i \omega_j C_{ij} \tag{4.13}$$

通过拉格朗日乘子法解得ω_i[Perrone and Cooper, 1993]

$$\omega_i = \frac{\sum_{j=1}^{T} C_{ij}^{-1}}{\sum_{k=1}^{T}\sum_{j=1}^{T} C_{kj}^{-1}} \tag{4.14}$$

式（4.14）是求解权重ω_i的闭形式解。用闭形式求解ω_i时要求相关矩阵C是可逆的，然而在集成学习中矩阵C通常是奇异的或者病态的。因为每个基本学习器的误差之间是高度相关的，基本学习器也可能是相似的，因为基本学习器都是在同一个问题上训练的。因此式（4.14）的解通常是不可行的（在可行域外），即求出的解不满足$\omega_i \geq 0$。

简单平均可以看成是加权平均的特殊形式，所有权重都相等。其它方法如投票法也是加权平均的变体。实际上集成方法的思想来源于加权平均，因为任何集成方法都可以认为是计算基本学习器权重的方法，不同的集成方法可以看成是不同的加权平均方法。从这种角度看，很容易知道没有哪一种集成方法一定是最优的，因为计算权重并不是件容易的事情。

虽然简单平均是加权平均的特例，但是不能说简单平均的效果就比加权平均差。在一些文献中实验结果也没有显示出加权平均要优于简单平均[Xu et al., 1992, Ho et al., 1994, Kittler et al., 1998]。因为在实际问题上，训练数据常常不足而且数据中含有噪声，所以在这样的数据上估计出的权重也是不可靠的。当集成算法中的基本学习器很多时，就有需要学习很多的权重，也就会导致过拟合。简单平均不需要学习任何权重，所以也出现过拟合的可能性也就小的多。普遍认为加权平均的基本学习器由相近的性能，但是如果基本学习器各自都有自己的长处，则使用加权平均可以达到更好的性能。

4.3 投票法

投票法是一种常用的且基本的对输出结果进行组合的方法。本节将以分类问题为例解释投票法的组合方式。假设给定T个基本分类器$\{h_1, \cdots, h_T\}$，投票的任务是组合每个h_i的l个输出$\{c_1, \cdots, c_l\}$，预测样本的最终类别。通常来说，h_i对样本x的预测类别有l维，用向量表示为$\Big(h_i^1(x), \cdots, h_i^l(x)\Big)^T$，$h_i^j(x)$是$h_i$对类别$c_j$的预测结果。$h_i^j(x)$可以采用不同类型值。

- 标签（离散）类型：$h_i^j(x) \in \{0,1\}$，样本被h_i预测为c_j时取 1 否则取 0。
- 概率（连续）类型：$h_i^j(x) \in [0,1]$，可以将其看成$P\big(c_j|x\big)$的后验概率。

对于有些特殊的分类器，如 SVMs 会产生非归一化的是间隔，calibration methods such as Platt scaling [Platt, 2000] or Isotonic Regression[Zadrozny and Elkan, 2001b]可以将这些分类器的输出转化为概率的形式，就可以使用投票组合方法了。值得注意的是大多数分类器估计出的概率都是不准确的，但是基于概率的组合方法比基于离散类别的组合方法性能要好的多，如果能做一些标准化的处理，基于概率的组合效果将会更好。

4.3.1 多数表决

多数表决时最常用的方法。对一个样本，每个分类器都输出一个分类类别，最终的类别由类别集合中超过半数的那个类别决定。如果没有一个类的类别数半数，组合分类

器则不会给出分类结果，即无法明显给出此样本的分类结果。用公式表示的组合分类器的类别输出是

$$H(x)=\begin{cases} c_j \ \ if \ \sum_{i=1}^{T} h_i^j(x) > \frac{1}{2}\sum_{k=1}^{l}\sum_{i=1}^{T} h_i^k(x) \\ \text{rejection otherwise} \end{cases} \tag{4.15}$$

对于二分类问题至少有$\left\lfloor \frac{T}{2}+1 \right\rfloor$个基本分类器分类正确，组合结果才是正确的。假设分类器的输出是独立的，且每个分类器都有个精度p，即每个基本分类器分类正确的概率为p。至少有$\left\lfloor \frac{T}{2}+1 \right\rfloor$个基本分类器分类正确的情况下，使用二项分布可以计算出集成分类器的正确分类的概率为

$$P_{mv}=\sum_{k=\left\lfloor \frac{T}{2}+1 \right\rfloor}^{T}\binom{T}{k}p^k(1-p)^{T-k} \tag{4.16}$$

不同的p和T对应的集成分类器的精度如图 4.2 所示

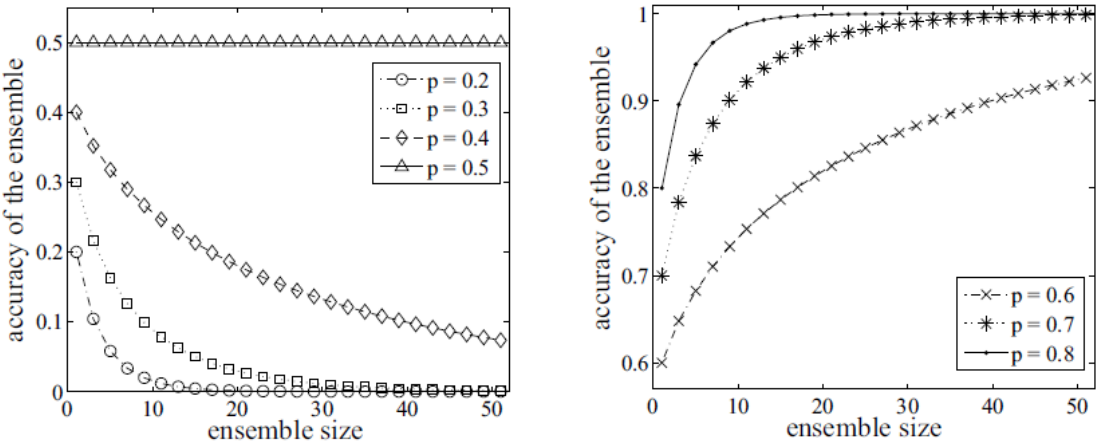


图 4.2 二分类问题的多数表决精度，T个独立的分类器，每个分类器的精度为p

Lamand Suen [1997]证明

- 如果$p>0.5$，则P_{mv}在T中是单调递增的，同时有
$$\lim_{T\to+\infty}P_{mv}=1$$
- 如果$p<0.5$，则P_{mv}在T中是单调递减的，同时有
$$\lim_{T\to+\infty}P_{mv}=0$$
- 如果$p=0.5$，对于任何T，$P_{mv}=0.5$

需要注意的是这些结论都是基于基本分类器是统计独立的假设的，然而在实际中分类器是高度相关的，因为它们都是为了解决同一个问题而训练的，所以不要期望多数表决会随着基本分类器的数量增加而收敛到 1。

4.3.2 多元表决

在多数表决中要求最终的获胜者至少要达到总票数的一半，多远表决则是将投票结果中类别最多的那个类作为最终的类（不要求获胜的类的数量要达到整个投票数的一半），即组合分类器的最终分类结果为

$$H(x)=c_{\arg\max_j \sum_{i=1}^{T} h_i^j(x)} \tag{4.17}$$

显然多元表决不会出现样本无法分类的现象，因为它总是能找到最大的类别数量。在二分类问题中多元表决等价于多数表决。

4.3.3 加权表决

从直观上感觉，不是每个学习器的性能都一样，所以在投票时应该给性能强的学习器以更大的权重，这就是加权表决。组集成分类的输出为

$$H(x)=c_{\arg\max_j \sum_{i=1}^{T}\omega_i h_i^j(x)} \tag{4.18}$$

ω_i是h_i的权重。在实际应用中权重满足$\omega_i\geq 0$，且$\sum_{i=1}^{T}\omega_i=1$，这与加权平均中的条件一致。

用一个例子来比较加权表决和多数表决的区别。假设有 5 个基本分类器它们的精度分别为{0.7,0.7,0.7,0.9,0.9}。则多数表决的精度为

$$P_{mv}=0.7^3+2\times 3\times 0.7^2\times 0.3\times 0.9\times 0.1+3\times 0.7\times 0.3\times 0.9^2\approx 0.933$$

这个精度比最好的基本分类器的精度要好。对于加权表决问题，基本学习器的权重为$\left\{\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{3},\frac{1}{3}\right\}$，加权表决的精度为

$$P_{mv}=0.9^2+2\times 3\times 0.9\times 0.1\times 0.7^2\times 0.3+2\times 0.9\times 0.1\times 0.7^3\approx 0.951$$

这表明通过调节基本分类器的权重分配，加权表决既优于最好的基本分类器也优于多数表决方法的精度。和加权平均类似，关键在于如何设置权重。

设$\boldsymbol{l}=(l_1,\cdots,l_T)^T$表示一个基本分类器的输出，$l_i$表示$h_i$对样本$x$的分类类别，$p_i$为$h_i$的精度。集成分类器在类别$c_j$上的贝叶斯最优判别函数为

$$H^j(x)=\log\left(P(c_j)P(\boldsymbol{l}|c_j)\right) \tag{4.19}$$

假设基本分类器是条件独立的，即$P(\boldsymbol{l}|c_j)=\prod_{i=1}^{T}P(l_i|c_j)$，所以可以得到下式

$$\begin{aligned}H^j(x)&=\log P(c_j)+\sum_{i=1}^{T}\log P(l_i|c_j)=\log P(c_j)+\log\left(\prod_{i=1,l_i=c_j}^{T}P(l_i|c_j)\prod_{i=1,l_i\neq c_j}^{T}P(l_i|c_j)\right)\\&=\log P(c_j)+\log\left(\prod_{i=1,l_i=c_j}^{T}p_i\prod_{i=1,l_i\neq c_j}^{T}(1-p_i)\right)\\&=\log P(c_j)+\sum_{i=1,l_i=c_j}^{T}\log\frac{p_i}{1-p_i}+\sum_{i=1}^{T}\log(1-p_i)\end{aligned} \tag{4.20}$$

因为$\sum_{i=1}^{T}\log(1-p_i)$与$c_j$无关，$l_i=c_j$可以由$h_i^j(x)$表示，则判别函数可以简化为

$$H^j(x)=\log P(c_j)+\sum_{i=1}^{T}h_i^j(x)\log\frac{p_i}{1-p_i} \tag{4.21}$$

式（4.21）中的第一项与基本学习器无关，第二项显示加权表决的最优化权重满足

$$\omega_i\propto\log\frac{p_i}{1-p_i} \tag{4.22}$$

上式表明权重和基本分类器的性能成正比。

注意式（4.22）是基于基本分类器相互独立的假设的，然而所有的基本分类器都是针对同一个问题训练的，所以这些基本分类器之间常常高度相关。以上的推导还需要估计基本分类器的精度，同时它没有考虑类别的先验分布。因此实际情况中用式（4.22）的权重加权结果并不是一定比多数表决的效果好。

4.3.4 Soft 表决

多数表决，多元表决，加权表决可用于离散的类别标签，如果基本分类器的输出是概率值，则可以使用 soft 表决。h_i对样本x的预测类别有l维，用向量表示为$\left(h_i^1(x),\cdots,h_i^l(x)\right)^T$，$h_i^j(x)\in[0,1]$，$h_i^j(x)$可以认为是后验概率$P(c_j|x)$。

如果同等对待所有基本分类器，则可以将所有基本分类器的输出简单平均起来，类别c_j的最终输出为

$$H^j(x)=\frac{1}{T}\sum_{i=1}^{T}h_i^j(x) \tag{4.23}$$

如果每个基本分类器的权重不同，则可以使用加权的 soft 表决方法，有如下四种形式

- 每个分类器都有一个权重，则加权组合的分类器对类别c_j的预测为

$$H^j(x)=\frac{1}{T}\sum_{i=1}^{T}\omega_i h_i^j(x) \tag{4.24}$$

ω_i是$h_i^j(x)$的权重。

- 每个分类器预测出的每类个有不同的权重，则加权组合的分类器对类别c_j的预测为

$$H^j(x)=\frac{1}{T}\sum_{i=1}^{T}\omega_i^j h_i^j(x) \tag{4.25}$$

ω_i^j是$h_i(x)$对类别c_j预测的权重。

- 每个类别中的每个样本有不同的权重，则加权组合的分类器对类别c_j的预测为

$$H^j(x)=\sum_{i=1}^{T}\sum_{k=1}^{m}\omega_{ik}^j h_i^j(x) \tag{4.26}$$

ω_{ik}^j表示分类器$h_i(x)$预测x_k为c_j时x_k的权重。

实际应用中第三种情况用的较少，因为它需要确定很多权重系数。第一种情况类似于加权平均或者加权表决。所以下面集中精力研究第二中情况。因为$h_i^j(x)$可以认为是后验概率$P(c_j|x)$的估计值，因此有

$$h_i^j(x)=P(c_j|x)+\epsilon_i^j(x) \tag{4.27}$$

$\epsilon_i^j(x)$是误差项。在分类问题中，目标输出是一个类别标签。如果基本分类器输出的估计是无偏的，则集成分类器的输出$H^j(x)=\frac{1}{T}\sum_{i=1}^{T}\omega_i^j h_i^j(x)$也是无偏的。通过设置权重系数可以得到用于估计$P(c_j|x)$的无偏的方差最小的$H^j(x)$。在约束条件$\omega_i^j\geq 0$，$\sum_{i=1}^{T}\omega_i^j=1$下，可以通过求解式（4.28）的优化问题，使组合误差的方差$\sum_{i=1}^{T}\omega_i^j\epsilon_i^j(x)$最小

$$\omega^j=\arg\min_{\omega^j}\sum_{k=1}^{m}\left(\sum_{i=1}^{T}\omega_i^j h_i^j(x_k)-\mathbb{I}(f(x_k)=c_j)\right)^2, j=1,\cdots,l \tag{4.28}$$

soft 表决只适用于同类型的基本学习器的输出结果组合（要么事逻辑回归，要么是 SVM，不能将两个混合）。如果基本学习器的类型不一样（可以将逻辑回归和 SVM 混合起来）则无法使用 soft 表决。如果基本学习器的类型不同，如果不对它们的输出结果进行标准化处理，则它们的输出结果之间是没有可比性（因为输出的量纲可能不一样），所以通常是将每个基本的分类器的输出概率转化为相应的类别，即如果$h_i^j(x)=\max_j\{h_i^j(x)\}$，则$h_i^j(x)=1$否则为 0。然后这就转化为离散输出问题了。对于离散输出问题上面已经介绍的相关的处理方法。

4.3.5 相关理论

4.3.5.1 多数表决的理论边界

Narasimhamurthy [2003, 2005]分析了多数表决的理论边界。本节将详细介绍这种分析方法。

以二分类问题为例，有T个基本分类器$h_1,\cdots,h_T$，它们的精度分别为$p_1,\cdots,p_T$，为了简单考虑，假设T为奇数（偶数问题见[Narasimhamurthy, 2005]）。可以使用韦恩图统表示集成分类器的统计量，图 4.3 为三个分类器的情形。每个基本分类器用一个数字表示 1 表示分类正确，0 表示分类错误。韦恩图中区域表示点的组合。例如标为x_3的区域与组合“110”对应，“110”表示h_1，h_2分类正确，h_3分类错误，x_3表示这种情况发生的概率。$x=[x_0,\cdots,x_{2^T-1}]^T$表示向量的联合概率，$bit(i,T)$表示整数$i$的$T$个二分表示，$f_{mv}$表示长度为$2^T$的向量，其中第$i$个元素是

$$f_{mv}(i)=\begin{cases}1, \text{如果} bit(i,T)\text{中 1 的个数} > \frac{T}{2}\\ 0, \text{其它}\end{cases} \tag{4.29}$$

则多数表决正确分类的概率为$f_{mv}^T x$，通过最优化方法求解$f_{mv}^T x$最大/最小值[Narasimhamurthy, 2003, 2005]。

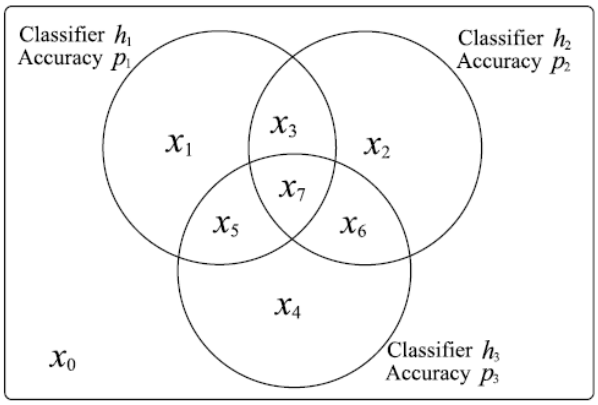


图 4.3 三个分类器的联合统计韦恩图。用x_i标记的不同区域对应于不同的 0 和 1 的组合
(Plot based on a similar figure in [Narasimhamurthy, 2003].)

因为h_i的精度为p_i，也就是说h_i的正确分类概率为p_i，p_i可以表示如下

$$b_i^T x = p_i (1<i<T) \tag{4.30}$$

很容易得到

$$b_1=(0,1,\cdots,0,1)^T$$
$$b_2=(0,0,1,1,\cdots,0,0,1,1)^T$$
$$\vdots$$
$$b_T=\left(\overbrace{0,0,0,0}^{2^{T-1}},\cdots,\overbrace{1,1,1,1}^{2^{T-1}}\right)^T$$

让$B=(b_1,\cdots,b_T)^T$，$p=(p_1,\cdots,p_T)^T$。约束为$\sum_{i=0}^{2^T-1}x_i=1$，$0\le x_i\le 1$，多数表决的上下界可以通过以下线性规划求解[Narasimhamurthy, 2003, 2005]

$$\begin{gathered}\min/\max_x \boldsymbol{f_{mv}^T x}\\ B\boldsymbol{x}=\boldsymbol{p}\\ \boldsymbol{1}^T\boldsymbol{x}=\boldsymbol{1}\\ 0\le x_i\le 1\ (\forall i=0,1,\cdots,2^T-1)\end{gathered} \tag{4.31}$$

图 4.4 分别是三个和五个基本分类器的多数表决理论上的上下界。为了方便演示，假设所有分类器的精度一致。精度的取值范围为 0 到 1，每个精度值都对应于多数表决分类器的理论上下界，并画出每个基本分类器的精度，从图中可以看出基本分类器的精度位于多数表决曲线的上下界之间。

从式（4.31）中可以看出，在T范围内约束的个数是线性变化的，而向量x的维数在范围 T 内是指数变化的。换句话说自由度随着与T是成指数变化的。因此对于特定精度的分类器，增加分类器的个数将会使理论下界变小，理论上界变大，这种趋势可以从图 4.4（a）和（b）中看出来。但这只是理想情况，因为实际中基本分类器并不是相互独立的。

4.3.5.2 决策边界分析

Tumer and Ghosh [1996]发表了决策边界分析框架。Later, Fumera and Roli [2005]用决策边界分析了加权的 soft 表决方法，下面主要主要介绍这个框架和相关结果[Tumer and Ghosh, 1996, Fumera and Roli, 2005]。

为了简单起见只考虑一维特征空间[Tumer and Ghosh, 1996, Fumera and Roli, 2005]，从一维特征空间得到的结果同样适用于多维空间[Tumer, 1996]。根据贝叶斯决策理论，当后验概率$P(c_i|x)$最大时x的类别为c_i。如图 4.5 所示在c_i和c_j之间理想的决策边界为x^*，x^*满足下式

$$P(c_i|x^*) = P(c_j|x^*) > \max_{k\neq i,j} P(c_k|x^*) \tag{4.32}$$

在实际应用中分类器用$h^j(x)$估计真实的后验概率$P(c_i|x)$

$$h^j(x) = P(c_i|x) + \epsilon_j(x) \tag{4.33}$$

$\epsilon_j(x)$是误差项，它是一个均值为β_j，方差为σ_j^2的随机变量。因此使用贝叶斯决策理论，误分情况用下式表示

$$\arg\max_i h^j(x) \neq \arg\max_i P(c_i|x) \tag{4.34}$$

从后验概率估计中得到决策边界x_b

$$h^i(x_b) = h^j(x_b) \tag{4.35}$$

它与理想的决策边界之间有个差值

$$b = x_b - x^* \tag{4.36}$$

在图 4.5 中可以看到，式（4.36）将会导致在贝叶斯误差上叠加一个额外的误分误差，称这个额外误差为附加误差（added error）[Tumer and Ghosh, 1996]。

下面主要研究无偏和不相关误差，详见[Tumer and Ghosh, 1996, Fumera and Roli, 2005]。对于任意给定的x，误差项$\epsilon_j(x)$的均值等于0，即$\beta_j = 0$，在不同的类别上误差项是不相关的，即$i \neq j$时$\epsilon_i(x)$和$\epsilon_j(x)$不相关。

不失一般性假设$b > 0$。从图 4.5 可以看出额外误差（added error）依赖b，并有如下关系

$$\int_{x^*}^{x^*+b} \left[P(c_j|x) - P(c_i|x)\right] p(x)dx \tag{4.37}$$

$p(x)$是x的概率分布，假设$b = x_b - x^*$很小，在x^*领域内的线性展开为

$$P(c_k|x^* + b) = P(c_k|x^*) + bP'(c_k|x^*) \tag{4.38}$$

$p(x)$可以用$p(x^*)$近似，因此额外误差变成$\frac{p(x^*)t}{2}b^2$，$t = P'(c_j|x^*) - P'(c_i|x^*)$，通过式（4.32）和式（4.35）很容易得到

$$b = \frac{\epsilon_i(x_b) - \epsilon_j(x_b)}{t} \tag{4.39}$$

因为$\epsilon_i(x)$和$\epsilon_j(x)$是无偏的且是不相关的，b的偏差和方差为$\beta_0 = 0$和$\sigma_b^2 = \frac{\sigma_i^2 + \sigma_j^2}{t^2}$。则关于b的加性误差的期望$err_{add}$为[Tumer and Ghosh, 1996]

$$err_{add}(h) = \frac{p(x^*)t}{2}\left(\beta_b^2 + \sigma_b^2\right) = \frac{p(x^*)}{2t}\left(\sigma_i^2 + \sigma_j^2\right) \tag{4.40}$$

现在考虑最简单的加权表决，每个基本学习器的权重为ω_i。在式（4.33）基础上，第j个类的输出结果为

$$h_{wsv}^{j}=\sum_{i=1}^{T}\omega_i h_i^j(x)=P\left(c_j|x\right)+\epsilon_j^{wsv}(x) \tag{4.41}$$

其中，$\epsilon_j^{wsv}(x)=\sum_{i=1}^{T}\omega_i\epsilon_i^j(x)$，$wsv$表示加权 soft 表决。类比图 4.5 决策边界$x_{b_{wsv}}$满足约束$h_{wsv}^{i}\left(x_{b_{wsv}}\right)=h_{wsv}^{j}\left(x_{b_{wsv}}\right)$，$x_{b_{wsv}}$与真实的决策边界$x^*$差一个边界$b_{wsv}$。按照上面相同的步骤，加权 soft 表决的加性误差期望可以写为

$$err_{add}^{wsv}(H)=\frac{p(x^*)}{2t}\sum_{k=1}^{T}\omega_k^2\left[\left(\sigma_i^k\right)^2+\left(\sigma_j^k\right)^2\right]=\sum_{k=1}^{T}\omega_k^2 err_{add}(h_k) \tag{4.42}$$

其中

$$err_{add}(h_k)=\frac{p(x^*)}{2t}\left[\left(\sigma_i^k\right)^2+\left(\sigma_j^k\right)^2\right] \tag{4.43}$$

是基本分类器h_k的加性误差的期望。这样 soft 表决方法的性能就可以用基本分类器的加性误差期望进行分析，而不是使用偏差和方差进行分析。

在约束条件$\omega_k\geq 0$，$\sum_{k=1}^{T}\omega_k=1$下的最优化权重为

$$\omega_k=\left(\sum_{i=1}^{T}\frac{1}{err_{add}(h_i)}\right)^{-1}\frac{1}{err_{add}(h_k)} \tag{4.44}$$

这表明最优化权重与分类器的加性误差的期望成反比。这也同时说明不同特点的分类器应该使用不同的权重[Tumer and Ghosh, 1996]。

将式（4.44）代入式（4.42）最优化权重$err_{add}^{wsv}(H)$可以表示成

$$err_{add}^{wsv}(H)=\frac{1}{\frac{1}{err_{add}(h_1)}+\cdots+\frac{1}{err_{add}(h_T)}} \tag{4.45}$$

另一方面如果使用简单的 soft 表决，将式（4.42）中权重替换为$\omega_i=\frac{1}{T}$，则加性误差的期望为

$$err_{add}^{ssv}(H)=\frac{1}{T^2}\sum_{k=1}^{T}err_{add}(h_k) \tag{4.46}$$

ssv表示简单 soft 表决（simple soft voting）。

当基本分类器的加性误差的期望都是一样的时候，即$err_{add}(h_i),\forall i,j$，有$err_{add}^{wsv}(H)=err_{add}^{ssv}(H)$，同时加性误差的期望随着$T$的增大而减小。

当基本分类器的加性误差不相等时，有$err_{add}^{wsv}(H)\leq err_{add}^{ssv}(H)$。不失一般性，最小加性误差和最大加性误差为

$$err_{add}^{best}(H)=\min_i\{err_{add}(h_i)\},\ err_{add}^{worst}(H)=\max_i\{err_{add}(h_i)\} \tag{4.47}$$

相应的分类器分别对应为最佳分类器和最差分类器。在式（4.46）两边同时除以$err_{add}(h_k)$，如下

$$\frac{err_{add}^{ssv}(H)}{err_{add}(h_k)}=\frac{1}{T^2}\left(1+\sum_{i\neq k}\frac{err_{add}(h_k)}{err_{add}(h_k)}\right) \tag{4.48}$$

称式（4.48）为减缩因子，对应的最优和最差的分类器的减缩因子分别为

$$\frac{err_{add}^{ssv}(H)}{err_{add}^{best}(H)}\in\left(\frac{1}{T},+\infty\right),\ \frac{err_{add}^{ssv}(H)}{err_{add}^{\text{worst}}(H)}\in\left(\frac{1}{T^2},\frac{1}{T}\right) \tag{4.49}$$

从式（4.45）可得到加权的 soft 表决的减缩因子为

$$\frac{err_{add}^{wsv}(H)}{err_{add}(h_k)}=\left(1+\sum_{i\neq k}\frac{err_{add}(h_k)}{err_{add}(h_k)}\right)^{-1} \tag{4.50}$$

因此有

$$\frac{err_{add}^{wsv}(H)}{err_{add}^{best}(H)}\in\left(\frac{1}{T},1\right),\ \frac{err_{add}^{wsv}(H)}{err_{add}^{worst}(H)}\in\left(0,\frac{1}{T}\right) \tag{4.51}$$

From (4.49) and (4.51) it can be seen that, if the individual classifiers have nonidentical added errors, both the simple and weighted soft voting achieve an error reduction smaller than a factor of T

over the best classifier, and larger than a factor of T over the worst classifier. Moreover, weighted soft voting always performs better than the best individual classifier, while when the performances of individual classifiers are quite poor, the added error of simple soft voting may become arbitrarily larger than that of the best individual classifier. Furthermore, the reduction achieved by weighted soft voting over the worst individual classifier can be arbitrarily large, while the maximum reduction achievable by simple soft voting over the worst individual classifier is 1/T 2. It is worth noting that all these conclusions are obtained based on the assumptions that the individual classifiers are uncorrelated and that the optimal weights for weighted soft voting can be solved, yet real situations are more complicated.

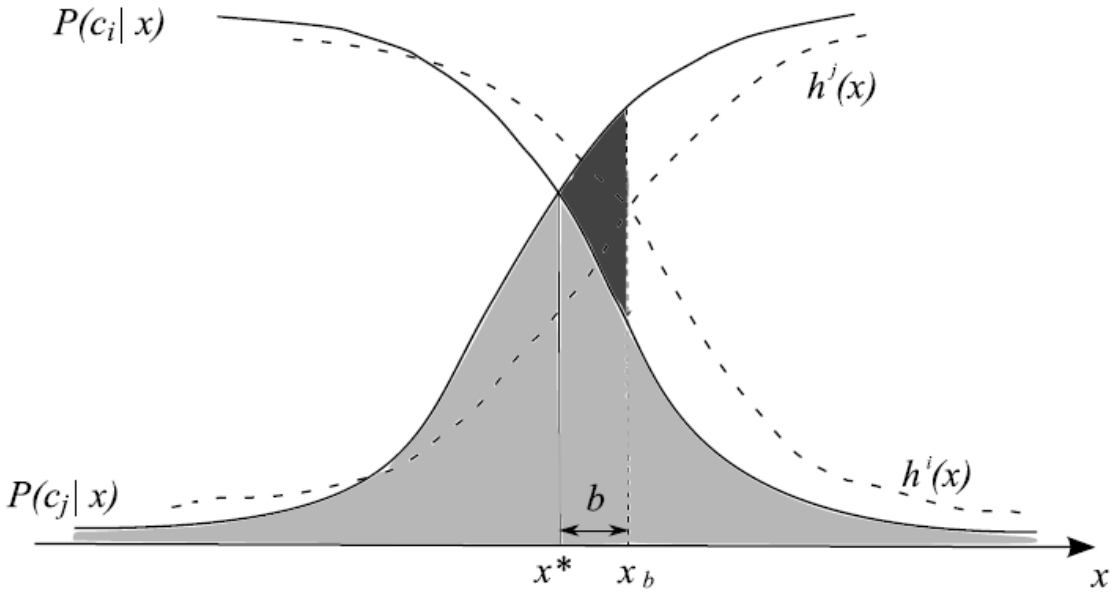


图4.5 实线表示真实的厚颜分布$P(c_i|x)$和$P(c_j|x)$，x^*为真实的决策边界。估计得到的后验概率$h^i(x)$和$h^j(x)$用虚线表示，$h^i(x)$和$h^j(x)$的决策边界为x_b。浅色和深色的阴影区分别为贝叶斯误差和额外误差（added error）。（[Fumera and Roli, 2005]中有类似的图形）

4.4 通过学习的组合方法

4.4.1 Stacking 方法

Stacking [Wolpert, 1992, Breiman, 1996b, Smyth and Wolpert, 1998]是一种通用的学习过程。它通过训练将基本学习器组合起来。其中的基本学习器称为第一层学习器，组合学习器称为第二层学习器（*second-level learner*）或者元学习器（meta-learner）。

Stacking 的基本思想是先用原始数据训练一阶学习器，用第一层学习器的输出作为第二层学习器的输入特征，原始数据的标签作为新特征的标签，然后用这个新的数据集训练第二层学习器。第一层学习器中可以包括不同的学习算法，Stacking 集成方法中（第二层学习器）通常也包括不同的学习器，当然它的基本学习器也可以是同类型的。图 4.6 中是一般化的 Stacking 算法的伪码

输入： 数据集$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$
一阶学习算法$\mathfrak{L}_1, \cdots, \mathfrak{L}_T$
二阶学习算法$\mathfrak{L}$
训练过程：
1 for $t = 1, \cdots, T$
2 $h_t = \mathfrak{L}_t(D)$
3 end
4 $D' = \emptyset$

5 for $t = 1, \cdots, m$
6 for $t = 1, \cdots, T$
7 $z_{it} = h_t(x_i)$
8 end
9 $D' = D' \cup \left((z_{i1}, \cdots, z_{iT}), y_i \right)$
10 end
11 $h' = \mathfrak{L}(D')$。
输出：$H(x) = h'\left(h_1(x), \cdots, h_T(x)\right)$

图 4.6 一般化的 Stacking 计算过程

可以从两方面看待 Stacking 方法，第一它是许多集成方法的推广。第二它是通过学习得到的集成方法。

在 Stacking 的训练阶段，从第一层学习器中得到新的数据集。如果用同一份完全相同的数据训练第一层分类器，并用该份数据在第一层分类器上的输出作为第二次分类器的训练数据，这会有过拟合风险。因此训练第一层分类器的数据，不能作为构造第二次分类器的数据，所以在构造第二次分类器的训练数据时需要用交叉验证或留一法选取第二层分类器的训练数据。

以k折交叉验证为例。将数据集D随机拆分成k个基本上相等的子集$D_1, \cdots, D_k$，在第j折中定义D_j和$D_{(-j)} = D \backslash D_{-j}$为测试数据和训练数据。有 T 个基本学习算法，用$D_{(-j)}$训练第$t$个学习算法得到第一层的基本学习器$h_t^{(-j)}$。第$j$折中的测试数据集$D_j$中的样本$x_i$在$h_t^{(-j)}$上的输出为$z_{it}$，其中$1 \le t \le T$。每个交叉验证都进行这样的计算，从第一层的$T$个基本学习器中得到的新的训练数据为

$$D' = \{(z_{i1}, \cdots, z_{iT}, y_i)\}_{i=1}^{m} \tag{4.52}$$

第二层学习器h'是$(z_1, \cdots, z_T)$关于y的函数。在为第二层生成新的数据集后要用全量数据再次训练第一层的全部学习器。

Breiman [1996b]在 1996 年发表了 Stacking 回归算法。在一次层基本学习算方法中他采用了不同大小的回归树或者不同变量个数的线性回归模型。第二层采用最小二乘的回归模型，模型系数都要非负。这个非负约束的条件很重要，它能够保证 Stacking 集成的结果比单个学习器的性能要好。

对于 Stacking 算法的分类问题，Wolpert [1992]指出第一层学习器产生的新数据中的特征类型和第二层学习器所采用的学习算法对分类性能的影响很大。

Ting and Witten [1999]推荐使用类别概率而不是类别标签作为新生成数据集的特征，因为采用类别概率作为特征既可以使用预测结果信息，也可以将基本分类器的可信度考虑进去，分类器h_k在样本x上的输出为$\left(h_{k1}(x), \cdots, h_{kl}(x)\right)$，它表示在所有可能类别$\{c_1, \cdots, c_l\}$上的概率分布，$h_{kj}(x)$表示 x被h_k预测成c_j的概率。虽然h_k以最大概率$h_{kj}(x)$将x预测成c_j，其它类别的概率也是有用的。可以把所有的第一层分类器对x的预测概率和该样本的真实类别一起构造第二次分类器的训练数据。

Ting and Witten [1999]提出可以使用多响应线性回归作为第二层学习算法（**multi-response linear regression** (MLR),），它其实是最小二乘回归的变体。任何具有实数值特征的分类问题都可以转化为多响应的回归问题。对于一个分类问题具有l个类别$\{c_1, \cdots, c_l\}$，可以构造l个回归问题，如下：对于每个类别c_j，构造一个回归问题预测二分类变量，如果c_j是正确类别则回归结果为 1 否咋为 0。系数用最小二乘计算。在预测阶段，所有线性回归模型都会对样本x给出预测结果，将预测结果的类别值最大的输出作为结果。在回归问题中对系数进行非负的约束与分类器的性能改善之间没有相关关系

[Breiman, 1996b]。Seewald [2002]建议使用不同的特征集合训练l个线性回归问题。也就是说不同的分类器只对c_j进行预测，即$h_{kj}(x)(k=1,\cdots,T)$仅仅是与c_j相关的线性回归问题。因此每个线性回归问题有T个而不是有$l\times T$个特征。Weka 实现了标准的 Stacking 算法和 Seewald [2002]算法中的 StackingC 算法。

Clarke [2003]对比了 Stacking 方法和贝叶斯模型平均（Bayesian Model Averaging (BMA)）方法之间的差异，BMA 方法给每个后验概率的模型赋上不同的权重。理论上看如果模型集合中存在正确的生成模型，且噪声水平比较低，则 BMA 的结果不会很差，通常来说会超越 Stacking 算法。当通常来说 BMA 的结果不会比 Stacking 的结果好，因为准确的生成模型是很难计算的。Clarke [2003]的经验法则表明 Stacking 方法一般好于 BMA 方法，而且 BMA 方法对近似误差非常敏感。

4.4.2 无限集成

大部分的集成方法仅仅使用有限的假设空间中的子集，Lin and Li [2008]提出了无限集成框架，利用无限个假设子空间加以集成。这种学习框架需要为所有可能的假设空间都学习出权重。它利用支持向量机，把无限个假设空间嵌入到核函数中。这种框架的学习问题可以归结为特征核函数的 SVM 训练问题。

$\mathcal{H}=\{h_\alpha:\alpha\in\mathcal{C}\}$表示假设空间，$\mathcal{C}$是测度空间，嵌入$\mathcal{H}$中的核函数定义为

$$K_{\mathcal{H},r}(x_i,x_j)=\int_{\mathcal{C}}\Phi_{x_i}(\alpha)\Phi_{x_j}(\alpha)\,d\alpha \tag{4.53}$$

$\Phi_x(\alpha)=r(\alpha)h_\alpha(x)$，选择合适的$r:\mathcal{C}\mapsto\mathbb{R}^+$，让对所有的$x_i,x_j$积分都存在。$\alpha$是$h_\alpha$的参数，$Z(\alpha)$表示 Z依赖于α。当不考虑r时用$K_{\mathcal{H}}$代替$K_{\mathcal{H},r}$。很容易证明式（4.53）定义的核是有效的[Sch ölkopf and Smola, 2002]。

根据 SVM 理论，无限集成框架的优化问题如下

$$\begin{gathered}\min_{\omega\in\mathcal{L}_2(\mathcal{C}),b\in\mathbb{R},\xi\in\mathbb{R}^m}\frac{1}{2}\int_{\mathcal{C}}\omega^2(\alpha)d\alpha+C\sum_{i=1}^{m}\xi_i\\ \text{s.t.}\,y_i\left(\int_{\mathcal{C}}\omega(\alpha)r(\alpha)h_\alpha(x)d\alpha+b\right)\geq 1-\xi_i\\ \xi_i\geq 0(\forall i=1,\cdots,m)\end{gathered} \tag{4.54}$$

最终的分类器如下

$$g(x)=sign\left(\int_{\mathcal{C}}\omega(\alpha)r(\alpha)h_\alpha(x)d\alpha+b\right) \tag{4.55}$$

显然如果$\mathcal{C}$是不可数的，则在集成方法中h_α的权重$\omega(\alpha)r(\alpha)d\alpha$是非常小的。因此无限集成方法最终得到的分类器将和其它方法得到的分类器完全不同。假设将$\mathcal{H}$全部变为负值，即$h\in\mathcal{H}\Leftrightarrow -h\in\mathcal{H}$，则$\mathcal{H}$上的每个线性组合等价于非负权重的线性组合。将 b 看成常数假设式（4.55）可以看成是无限假设的集成方法。

通过使用拉格朗日乘子法和核技巧，可以得到式（4.54）的对偶问题，最终的分类问题可以写成

$$g(x)=sign(\textstyle\sum_{i=1}^{m}y_i\lambda_iK_{\mathcal{H}}(x_i,x)+b) \tag{4.56}$$

$K_{\mathcal{H}}$为假设空间$\mathcal{H}$上的核，λ_i为拉格朗日乘子。式（4.56）等价于式（4.55），因此式（4.56）也是在$\mathcal{H}$上的无限集成方法。在实际应用中，如果可以采用合适的嵌入函数r（树桩核函数感知器核函数），就可以通过式（4.53）构造出$K_{\mathcal{H}}$[Lin and Li, 2008]。则学习问题就可以转化为求解核函数为$K_{\mathcal{H}}$的 SVM 问题，因此最终的优化问题就可以采用 SVM 的优化方法求解。

4.5 其它组合方法

除了平均方法，投票表决方法，学习集成方法外。本节将简单介绍一些代数方法，BKS 方法和决策模板方法。

4.5.1 代数方法

因为基本分类器输出每个类别的概率可以看成是后验概率的估计值，可以在概率论框架下使用组合方法[Kittler et al.,1998]。

$h_i^j(x)$是h_i对类别c_j预测的概率。根据贝叶斯决策理论，给定T个基本分类器，样本x被分为c_j对应于后验概率$P\left(c_j\middle|h_1^j,\cdots,h_T^j\right)$最大化，按照贝叶斯理论有如下式子

$$P\left(c_j\middle|h_1^j,\cdots,h_T^j\right)=\frac{P(c_j)P\left(h_1^j,\cdots,h_T^j\middle|c_j\right)}{\sum_{i=1}^{l}P(c_i)P\left(h_1^j,\cdots,h_T^j\middle|c_i\right)} \tag{4.57}$$

$P\left(h_1^j,\cdots,h_T^j\middle|c_j\right)$为分类器输出的联合概率密度。

假设输出之间是条件独立的，则

$$P\left(h_1^j,\cdots,h_T^j\middle|c_j\right)=\prod_{i=1}^{T}P\left(h_i^j\middle|c_j\right) \tag{4.58}$$

式（4.57）可变为

$$P\left(c_j\middle|h_1^j,\cdots,h_T^j\right)=\frac{P(c_j)\prod_{i=1}^{T}P\left(h_i^j\middle|c_j\right)}{\sum_{i=1}^{l}P(c_i)\prod_{i=1}^{T}P\left(h_i^j\middle|c_j\right)}\propto P\left(c_j\right)^{-(T-1)}\prod_{i=1}^{T}P\left(c_j\middle|h_i^j\right) \tag{4.59}$$

h_i^j是概率输出，则有$P\left(c_j\middle|h_i^j\right)=h_i^j$。如果所有类别的先验概率都相等，则可得到组合起来的乘法法则[Kittler et al., 1998]为

$$H^j(x)=\prod_{i=1}^{T}h_i^j(x) \tag{4.60}$$

Kittler et al. [1998]用最大/最小/中值准则提出了 soft 表决方法。简单的说就是选择基本分类器输出的最大/最小/中值组合起来作为最终输出值。例如中值准则产生的组合输出为

$$H^j(x)=\mathrm{med}_i\left(h_i^j(x)\right) \tag{4.61}$$

med(·)表示统计中值。

4.5.2 行为知识空间法

Huang and Suen [1995].提出了行为知识空间方法（BKS）。$\boldsymbol{l}=(l_1,\cdots,l_T)^T$为基本分类器$h_1,\cdots,h_T$对$x$的预测类别，$l_i=h_i(x)$。如果将$\boldsymbol{l}$看成$T$维的随机变量，则主要工作就变成了估计$P\left(c_j\middle|\boldsymbol{l}\right)$。因此每个类别的可能组合可以认为是 BSK 表中的一个元素[Huang and Suen, 1995]。BSK 的表用数据集 D 填充，(x_i,y_i)在表中的位置的索引为$\left(h_1(x_i),\cdots,h_T(x_i)\right)$。对表中的每个单元中的样本进行统计，选择最有代表性的类别作为该单元对应的类别，如果单元为空则赋一个随机类别。在预测阶段，BSK 方法将 x 预测为$\left(h_1(x_i),\cdots,h_T(x_i)\right)$索引的单元所属的标签。

在大数据集上 BSK 法有不错的效果。在小数据集上它过拟合比较严重。为了解决这个问题 Raudys and Roli [2003]分析了 BSK 的泛化误差，他们得到了一个泛化误差和样本数量之间的解析模型关系。在这个模型的基础上，他们建议在 BKS 表的类别不明确的单元上使用线性分类器，这种方法极大的改善了 BKS 的性能。

4.5.3 决策模板法

Kuncheva et al. [2001].提出了决策模板方法。在这种方法中样本x在分类器上的输出用矩阵表示为一个决策问题。

$$DP(x)=\begin{pmatrix} h_1^1(x) & \cdots & h_1^j(x) & \cdots & h_1^l(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_i^1(x) & \cdots & h_i^j(x) & \cdots & h_i^l(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_T^1(x) & \cdots & h_T^j(x) & \cdots & h_T^l(x) \end{pmatrix} \tag{4.62}$$

在训练数据集$\mathrm{D}=\{(\mathrm{x}_1,\mathrm{y}_1),\cdots,(\mathrm{x}_m,\mathrm{y}_m)\}$上，决策模板是期望$DP(x)$的估计值

$$DK_k=\frac{1}{m_k}\sum_{i:y_i=c_k}DP(x_i),k=1,\cdots,l \tag{4.63}$$

m_k是类别c_k中的训练样本。

在预测阶段，该算法类似近邻算法。即首先计算出样本x的$DP(x)$之间的相似度，DK_k[Kuncheva et al., 2001]。将最相似决策模板所属的类赋给x。

4.6 相关方法

4.6.1 误差校正输出代码

4.6.2 动态分类器选择

4.6.3 混合专家

4.7 进一步阅读

见原文

5 多样性

5.1 集成多样性

集成方法的多样性是指组成集成方法的基本学习器之间差异性，这是集成方法需要研究的基本问题。

从直觉上很容易理解只有基本学习器之间存在多样性，集成起来的学习器才能提高性能，如果所有基本学习器都是一样的，则组合起来的学习器和基本学习器没有什么差别。通过引入参数θ用于描述基本分类器之间的相关关系，Tumer and Ghosh [1995]使用4.3.5.2节介绍的决策边界理论分析了简单soft表决方法的性能。他们给出了集成方法的加性误差的期望为

$$err_{add}^{ssv}\frac{1+\theta(T-1)}{T}\overline{err}_{add}(h) \qquad (5.1)$$

$\overline{err}_{add}(h)$为基本学习器的加性误差期望（为了简单化，假设所有基本学习器的误差是一样的）T是基本学习器的个数。式（5.1）表明如果基本学习器是独立的，即$\theta = 0$，集成学习器的误差将变为基本学习器误差的$\frac{1}{T}$。如果基本学习器是完全相关的，即$\theta = 1$，则集成学习器的性能和基本学习器的性能一致，即集成学习器的性能没有提示。从以上分析中可得基本学习器的多样性对集成方法至关重要。其它的集成学习方法也有类似的性质。

然而产生多样性的基本学习器却不那么简单。因为训练的基本学习器都是基于相同的训练数据，所有训练得到的基本学习器之间是高度相关的。许多理论上貌似可行的方法在实际中并不那么好使，如式（4.14）的最优化加权平均就是这样的理论方法，原因很简单，理论上的性能都是基于基本学习器独立性或者不相关的假设的。二实际的问题并不是独立那么简单。例如在式（5.1）中假设的是基本学习器的后验概率是相互独立的，实际上这是不可能的。

如果想让基本学习器的性能不是很差，而且又要使基本学习器具有多样性，这还是具有相当挑战性。如果基本学习器的性能较差，有没有多样性则集成的结果也不会好。例如从式（4.52）可以看出，当基本学习器的性能较差时，简单soft表决的加性误差将非变的很大。其它的集成方法也有这样相似的结论。

所以理想的基本学习器既要**准确又要多样**。假设有两种集成学习器类型：用精度都很高的基本学习器组合起来写成集成学习器A；用部强学习器和部分弱学习器组成的集成学习器为B，B的性能会比A的高，因为基本学习器间的相互补充比单纯的精度更重要。最后，集成学习成功的关键在于，要再学习器的性能和多样性方面做好平衡。

虽然多样性如此重要，目前还是无法很好的认识多样性，例如目前还没有很好定义多样性。**毫无疑问对多样性的研究是集成学习中的重要方面。**

5.2 误差分解

集成学习器的泛化误差和多样性之间联系甚密。因此本节介绍两种误差分解的方法：**歧义误差分解**和**偏差、方差、协方差分解**

5.2.1 歧义误差分解

5.2.2 偏差、方差、协方差分解

5.3 多样性度量方法

5.3.1 Pairwise 度量法

5.3.2 非 Pairwise 度量法

5.5.3 归纳和可视化

5.3.4 多样性度量方法的局限性

5.4 信息理论的多样性

5.4.1 信息论和集成方法

5.4.2 互信息的多样性

5.4.3 多信息的多样性

5.4.4 估计方法

5.5 多样性的产生