第 3 章

Chapter 3

经验风险最小

通过逻辑回归算法，推广泛化为广义线性模型，我们对机器学习中监督学习有了初步的掌握。其实，监督学习，尤其是其中一部分传统的分类问题，能够进一步泛化为一个统一的模型，称为基于分类界限 (Classification Margin) 的结构风险最小 (Structural Risk Minimization) 模型。而要深入解读结构风险最小，就需要理解机器学习中经典分类问题的统计学习基石，就是经验风险最小 (Empirical Risk Minimization)。本质上，结构风险最小模型就是经验风险最小和正则化 (Reguralization) 的组合。有了结构风险最小，我们就能进一步统一逻辑回归、广义线性模型之外分类算法，包括支持向量机、AdaBoost 算法等。

理解基于分类界限的经验风险最小，就要首先理解什么是风险，其次需要懂得为什么要经验风险最小，再次明白什么又是分类界限，最后通过经验风险最小来重新认知逻辑回归和广义线性模型。只有在认知了经验风险最小后，我们才能进一步理解为什么要结构风险最小和什么是正则化。

3.1 经验风险与泛化误差概述

大家已经很直观知道，分类的算法并不唯一，如有逻辑回归和支持向量机算法等。那么就有个很直观的问题，那就是哪个算法更好。在比较算法好坏的时候，有两种模式，一种脱离具体问题，绝对的比较算法。另外一种不脱离具体问题，相对的比较算法。我们在历史上学过一个道理，就是不能脱离历史环境来评价一个人物的好坏。其实，这里也有类似的结论，一个算法的环境或者上下文，就是针对特定的具体问题，即不能脱离具体问题来评价一个算法的好坏。有定理“没有免费的午餐”(No Free Lunch Theorem) 告诉我们，

不依赖特定问题来评价算法无法衡量算法的好坏，或者换句话说，如果均匀概率对待所有可能问题，那么不同算法在所有可能问题域上的期望是相同的。这样的话，一个算法如果在部分问题上表现比另外一个算法好，那么这个算法肯定在有些问题上表现不如别的算法。所以，评价一个分类算法的好坏，离不开特定的具体问题，如图 3.1 所示。

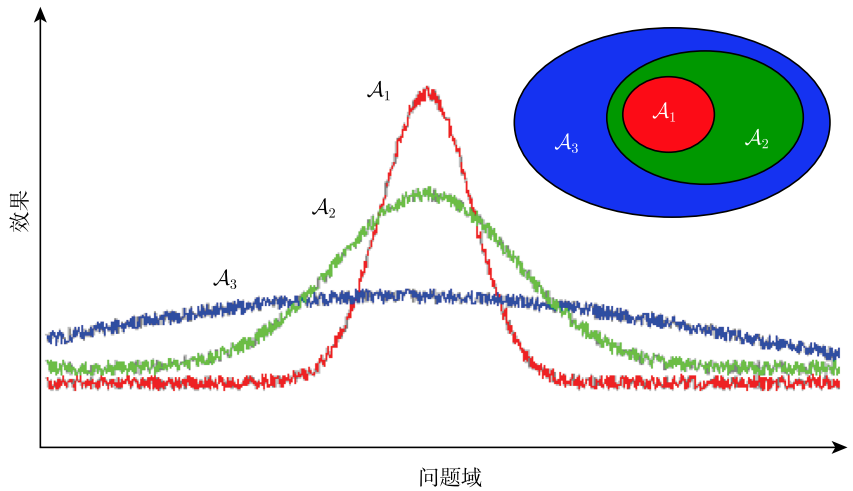

图 3.1 没有免费的午餐定理 (No Free Lunch Theorem) 表明评价一个分类算法的好坏依赖具体问题

如果限定了特定的具体问题，在评价分类算法的时候，还有个限制，就是能不能获得全部数据集合。通常情况下，整个数据集合是无限的，但是只能获得一个有限的样本集合。结合数据集合，又有了新的问题，那就是不同算法，在不同数据集合上评价是否一致。尤其是在一个有限的样本集合上的评价和一个无限的数据集合上的评价是否一致，如果不一致，偏差多少？其实，在不同数据集合上对同一个算法的评价不一定一致。这样，研究偏差多少，变得十分有意义。我们希望知道，一个算法比另外一个算法在有限的样本集合上要好，那么在无限的数据集合上也好可能性有多大。

评价分类算法好坏的上下文是特定的具体问题，并且已知一个有限的样本集合，对应的有一个无限的数据空间。但是，以上的讨论都在比较两个算法的好坏。如果要评价单个算法的好坏呢？这时需要定义一个标杆，对于特定的具体问题和特定数据集合下，表现最优的算法。有了这个标杆，其他算法都和最优算法进行对比。那么，又如何评价这个最优算法呢？就是和理想情况进行对比。由于真实数据世界存在噪声、异常、未知因素等情况，那么假定最优算法未必能达到理想情况是合理的。

举个简单的例子，假设有两个问题，$P_1 : \mathbb{X} \to \mathbb{Y}$，根据水的颜色、气味和固体悬浮物 (集合 $\mathbb{X}$) 来判断水质是否合格 (集合 $\mathbb{Y}$) 和 $P_2 : \mathbb{A} \to \mathbb{B}$，根据风力、温度和湿度 (集合 $\mathbb{A}$)

来判断明天是否有雨 (集合 $\mathbb{B}$)。我们有 3 个算法，即 f_1 逻辑回归、f_2 支持向量机和 f_3 AdaBoost。我们已经知道，当有无穷个问题 $\{P_1, P_2, \cdots, P_\infty\}$ 时，我们说 f_1 比 f_2 在这所有问题上效果要好是不合理的。那么，我们就考察对于 P_1 问题，哪个算法好。这时引入数据集合例子，$\mathbb{X}$ 为全国大型湖泊水的数据空间。因为受实际限制，只能获取到其中部分样本 $\mathbb{S} \in \mathbb{D}$, 例如江苏大型湖泊水的样本空间，其中 $\mathbb{D} = \{\mathbb{X}, \mathbb{Y}\}$ 是全部数据 (Data)，即全国大型湖泊水和对应水质的数据空间。当然，算法也可以是无穷多种类的 $\{f_1, f_2, \cdots, f_\infty\}$。这样，对于问题 P_1，我们直接能得到的是有限算法集合 $\mathcal{F} = \{f_1, f_2, f_3\}$ 在有限样本集合 $\mathbb{S}$ 上最佳算法 $\hat{f}_{\mathcal{F},S}$ (简写为 $\hat{f}_{\mathcal{F}}$)，这样我们就在 3 个算法里面找到合适江苏湖泊水判别水质是否合格的最佳算法。类似假定无限数据空间 $\mathbb{D}$ 上的最佳算法 $f^*_{\mathcal{F},\mathbb{D}}$(简写为 $f^*_{\mathcal{F}}$)，这是在 3 个算法里面找到合适全国湖泊水判别水质是否合格的最佳算法，那么可以明确的是，这个算法在全国湖泊水水质判别的效果上肯定要比前面的效果好。再进一步扩展到无限算法集合 $\{f_1, f_2, \cdots, f_\infty\}$ 上的最佳算法 $f^*_{\mathbb{D}}$(简写为 f^*)，这样在所有算法里面，找到的适合全国湖泊水判别水质是否合格的最佳算法，这个算法在全国湖泊水水质判别的效果上肯定比前面两种都要好。那么，这 3 个最佳算法之间效果的关系，就是要研究的泛化误差的目标。

3.1.1 经验风险

有了上文对问题、数据和算法的关系的分析，我们要进一步理解经验风险的含义。在理解经验风险前需要理解什么是风险 (Risk)。风险是损失函数 (Loss Function) 在数据集上的期望。但是，我们得不到整个数据集，那么损失函数在有限样本集上的均值，就是经验风险。那么，什么是损失函数呢？就是算法对一个样本的估计值和这个样本对应的真实值之间差异的评估函数。例如，我们有江苏 10 个湖泊的水样本，那么太湖作为其中一个样本，算法告诉我们水质合格，但是真实情况是太湖水质不合格。那么算法的估计值和真实值之间就有了差别。把给这种差别打分的函数，称为损失函数。例如，太湖水质估计的不正确打了 1 分，其余 9 个湖泊水质预测成功损失为 0 分。那么 10 个湖泊水质样本平均下来损失是 0.1 分，而这个 0.1 分就是经验风险。

3.1.2 泛化误差

前面提到的 3 种最佳算法 $\hat{f}_{\mathcal{F}}$、$f^*_{\mathcal{F}}$ 和 f^*，在样本数据集 $\mathbb{S}$ 上的损失函数 (Loss Function) 的平均称为各自的经验风险 (Empirical Risk)。而在全部数据集合 $\mathbb{D}$ 上的损失函数的期望，就称为各自的真实风险 (Ture Risk)。其中 f^* 在 $\mathbb{D}$ 上表现最优，那么真实风险最小，它的风险又被称为贝叶斯风险 (Bayes Risk)，贝叶斯风险是一个理论上算法可以达到

的最小的风险。我们知道 $\hat{f}_{\mathcal{F}}$ 就是基于有限样本 $\mathbb{S}$ 根据经验风险最小从有限算法集合 $\mathcal{F}$ 中选出的最佳算法。那么我们很想知道它的真实风险，就是它在 $\mathbb{D}$ 上的表现。并且，也想知道真实风险和贝叶斯风险理论最佳值的差距，这就是我们想探讨的泛化误差。

对应到 $\hat{f}_{\mathcal{F}}$、$f^*_{\mathcal{F}}$ 和 f^* 的真实风险，可分为 3 层理解。

(1) 理论上，我们能预测多好？假设 f^* 的真实风险为 R^*。

(2) 如果限制在有限算法集 $\mathcal{F}$ 条件下，我们能预测多好？假设 $f^*_{\mathcal{F}}$ 的真实风险为 $R^{true}(f^*_{\mathcal{F}})$。

(3) 如果限制在有限算法集 $\mathcal{F}$ 和有限样本 $\mathbb{S}$ 两个条件下，我们能预测多好？假设 $\hat{f}_{\mathcal{F}}$ 的真实风险为 $R^{true}(\hat{f}_{\mathcal{F}})$。

既然是限制条件加强，那么大范围的最优肯定优于子范围的最优，R^* 风险肯定不会大于 $R^{true}(f^*_{\mathcal{F}})$，而 $R^{true}(f^*_{\mathcal{F}})$ 肯定不会大于 $R^{true}(\hat{f}_{\mathcal{F}})$。这样把加了有限算法集 $\mathcal{F}$ 限制后的 $R^{true}(f^*_{\mathcal{F}}) - R^*$ 称为近似误差 (Approximation Error)。而将进一步加样本集合 $\mathbb{D}$ 的限制后的 $R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})$ 称为估算误差 (Estimation Error)，如图 3.2 所示。而把两个限制加上后的误差 $R^{true}(\hat{f}_{\mathcal{F}}) - R^*$ 称为泛化误差 (Generalization Error)。所以泛化误差是近似误差和估算误差之和。这样区分的好处在于，在考虑估算误差的时候，不要考虑算法 $f \in \mathcal{T}$ 的目标空间 $\mathcal{T}$，只考虑 $\mathcal{F} \subset \mathcal{T}$。而在考虑近似误差的时候，不要考虑如何采样得到样本空间 $\mathbb{S}$。

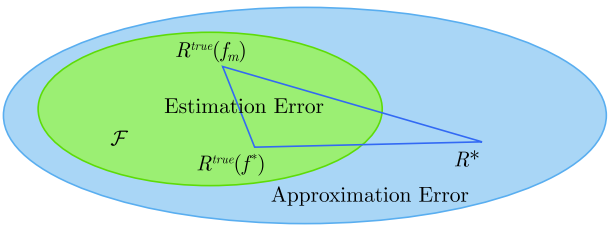

图 3.2 估算误差与近似误差

(1) 近似误差：我们知道新的算法会一直诞生，最优算法本身就是一个理论值，那么只能用算法集来近似。我们希望尽可能发现更有效的算法去逼近理论值，但是永远不会达到。

(2) 估算误差：我们知道要获得全部数据集代价太大，基于有限样本的计算，不管在时间还是计算资源上的代价都是很大的，因此我们来估算全部数据情况下的误差。

(3) 泛化误差：对算法和样本都有限制后的误差是真实风险的误差。泛化就是指从这样有限数据算法情况下到没有任何限制条件下推广会有误差。

要强调的是，上面讲解的是真实风险，真实风险全是在 $\mathbb{D}$ 上的评估。而这样的全数

据集的评估代价过于巨大。相比较而言，经验风险是在 $\mathbb{S}$ 上的评估。那么，泛化误差考察的最优算法 $\hat{f}_{\mathcal{F}}$ 就存在 $\mathbb{D}$ 上的真实风险 (泛化误差) 和在样本集 $\mathbb{S}$ 上的经验风险之间的区别。

① 从有限样本来看 $\hat{f}_{\mathcal{F}}$ 的表现？$\hat{f}_{\mathcal{F}}$ 的经验风险标记为 $R^{emp}(\hat{f}_{\mathcal{F}})$。

② 从有限样本来看 $f^*_{\mathcal{F}}$ 的表现？$f^*_{\mathcal{F}}$ 的经验风险标记为 $R^{emp}(f^*_{\mathcal{F}})$。特别要注意的是，$R^{emp}(f^*_{\mathcal{F}}) \geqslant R^{emp}(\hat{f}_{\mathcal{F}})$，因为 $\hat{f}_{\mathcal{F}}$ 的定义就是 $\mathbb{S}$ 上经验风险最小的算法。

引入经验误差之后，再来看估算误差 $R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})$，我们也引入经验风险进行推理。

$$||R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| = ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}}) + R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \tag{3.1}$$

$$\leqslant ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}})|| + ||R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \tag{3.2}$$

如果 $R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}}) \geqslant 0$，根据 $R^{emp}(f^*_{\mathcal{F}}) \geqslant R^{emp}(\hat{f}_{\mathcal{F}})$，那么

$$0 \leqslant R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}}) \leqslant R^{emp}(f^*_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})$$

如果 $R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}}) \leqslant 0$，根据 $R^{true}(f^*_{\mathcal{F}}) \leqslant R^{true}(\hat{f}_{\mathcal{F}})$，那么

$$0 \leqslant R^{true}(f^*_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}}) \leqslant R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}})$$

根据上面的特征，提取 $R^{true}(f) - R^{emp}(f), f \in \mathcal{F}$ 作为研究对象，如果对于 $\forall f \in \mathcal{F}, ||R^{true}(f) - R^{emp}(f)|| \leqslant \Omega(\mathcal{F}, \mathbb{S}, \delta)$ 以概率 $1-\delta$ 成立。那么，就可以把估算误差设置一个上限了。

根据前面推理，如果 $R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}}) \geqslant 0$

$$||R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \leqslant ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}})|| + ||R^{emp}(f^*_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \tag{3.3}$$

$$\leqslant 2\Omega(\mathcal{F}, \mathbb{S}, \delta) \tag{3.4}$$

如果 $R^{emp}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}}) \leqslant 0$

$$||R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \leqslant ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}})|| + ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{emp}(\hat{f}_{\mathcal{F}})|| \tag{3.5}$$

$$\leqslant 2\Omega(\mathcal{F}, \mathbb{S}, \delta) \tag{3.6}$$

所以，给估算误差找到一个上限 $||R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| \leqslant 2\Omega(\mathcal{F}, \mathbb{S}, \delta)||$，如图 3.3 所示。

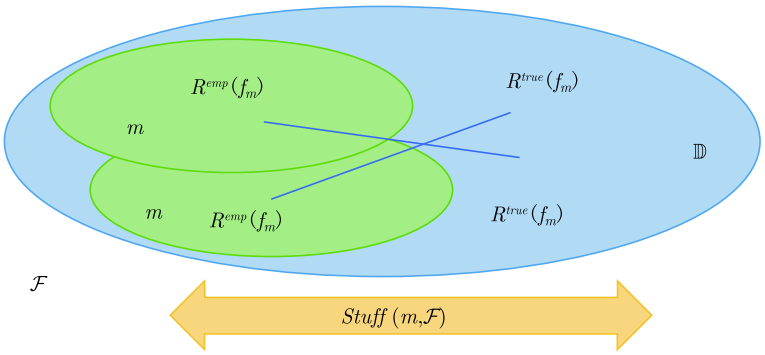

图 3.3 真实风险和经验风险之差的上限

这样把泛化误差分解成了估算误差和近似误差，并成功地给估算误差找了个上界

$$||R^{true}(\hat{f}_{\mathcal{F}}) - R^*|| \leqslant ||R^{true}(\hat{f}_{\mathcal{F}}) - R^{true}(f^*_{\mathcal{F}})|| + ||R^{true}(f^*_{\mathcal{F}}) - R^*|| \tag{3.7}$$

$$\leqslant 2\Omega(\mathcal{F}, \mathbb{S}, \delta) + ||R^{true}(f^*_{\mathcal{F}}) - R^*|| \tag{3.8}$$

很多时候算法的目标空间 $\mathcal{T}$ 是一个未知空间，所以近似误差的分析在近似理论 (Approximation Theory) 上更偏基础理论，离实际应用较远，所以近似误差常常被工程界忽视。但是估算误差对算法的能力有很好的上限描述，对具体算法集合的学习能力有个估算，所以往往得到更多的重视。那么这种算法能力的估算有什么用呢？我们会在后续欠拟合和过拟合中解释。

如果再回到湖水水质判别的问题，我们在江苏的湖水样本上，LR、SVM 和 AdaBoost 3 个分类算法中挑选了最优的算法，如 SVM，SVM 在全国湖水水质检测中的真实风险和采用所有可能算法在全国检测中的真实风险 (Bayes 风险) 的差异为泛化误差。如果采用 0-1 损失，那么真实风险就是错误率 (Error Rate)，也就是 1 减去准确率。如果假定贝叶斯风险对应的错误率为 0，而 SVM 对应的错误率为 0.25，那么泛化误差就是 0.25。但问题是，我们很可能不知道具体贝叶斯风险，因为不知道最好的算法，只有这 3 个算法，我们拿这 3 个算法在全国湖水水质上找到的最好算法，如 AdaBoost，那么 AdaBoost 对应也有错误率，如 0.2。那么这个 0.2 和 Bayes 风险之间的差别就是近似误差，因为我们不知道最好的算法是什么，就先不关心这个近似误差。而 AdaBoost 的 0.2 和 SVM 的 0.25 之间的 0.05 的差别就是估算误差。再进一步考虑可行性，其实这个 0.2 和 0.25 要拿全国的湖水测试，这个可能也拿不到，目前不是只拿到江苏的湖水吗，如 SVM，在江苏湖水的错误率只有 0.1，而 AdaBoost 在江苏湖水上的错误率要稍微差点 0.15，这就是经验风险了。虽然不知道 SVM 和 AdaBoost 的真实风险，但是能够推算 3 个算法中任意一个算法在江苏湖水上的错误率和全国湖水上的错误率之间是有个上限的，如 0.2 (如 SVM 的

真实风险 0.25 和经验风险 0.1 之差为 0.15，而 AdaBoost 真实风险 0.2 和经验风险 0.15 之差为 0.05)。那么就能计算出估算误差的上限为 2 倍的 0.2，也就是 0.4。虽然不是很准确，但毕竟我们还是有了个具体的范围，这个范围有什么用呢？我们可以计算到 SVM 用到全国湖水检验上去的效果比在全国湖水上找的 3 个算法中最好的算法 (如 AdaBoost) 的效果最多差了 0.4。是不是很神奇？所以这里面最奥妙的是如何找到那个上限。我们会在后续 VC 维里面介绍这个奥妙。

3.1.3 欠拟合和过拟合

假设有了对算法在样本集和整个数据集上效果的范围 (全数据对应的真实风险和样本数据对应的经验风险之差)，我们是如何使用的呢？在这之前，先假设算法在样本集和整个数据集上效果的几种具体情况。

(1) 算法在样本集效果不好，在整个数据集效果好。这种情况不太可能存在，因为数据集包括了样本集。这也不符合实际情况，实际情况是第一步要找一个在样本集上效果好的算法。

(2) 算法在样本集效果不好，在整个数据集效果也不好。这也是我们最不希望看到的情况。

(3) 算法在样本集效果好，在整个数据集效果不好。这是我们想避免的情况。

(4) 算法在样本集效果好，在整个数据集效果也好。这是最理想的情况，那么在样本集选到对应算法，在整个数据集表现也是好的。

这样，主要考虑上面后三种情况，首先看第二种情况，如果算法在样本和数据集上的效果都不好，那么会觉得这个算法的能力不行，一般称为欠拟合 (Underfitting)，如图 3.4(a) 所示。第三种情况，如果算法在样本上效果好，但是数据集上效果不好，那么会觉得这个算法能力过强了，一般称为过拟合 (Overfitting)，如图 3.4(c) 所示。最好就是正常拟合，这就是第四种情况，如图 3.4(b) 所示。

在研究拟合问题时候，经常会用到模型的拟合能力，在思考拟合能力的时候，又经常使用多项式曲线的拟合能力作为例子。举个多项式拟合的例子，当直接拿直线拟合的时候，会遇到拟合的不好，用来预测背后的线的趋势也不好。当使用高阶多项式拟合，增加阶数，用 3 阶多项式曲线拟合的时候，会发现拟合得很好，趋势预测也很好。但是如果继续增加阶数到 6 阶多项式拟合的时候，会发现拟合得很好，但是趋势完全不对了。一般认为，多项式的阶越高，拟合能力越高。这个的数学上的解释就是著名的泰勒公式展开多项式。随着展开的阶数越高，那么越精确地逼近原函数，所以能力越高。但是也会发现，并非拟合能力越高，预测效果越好。这个如何解释呢？

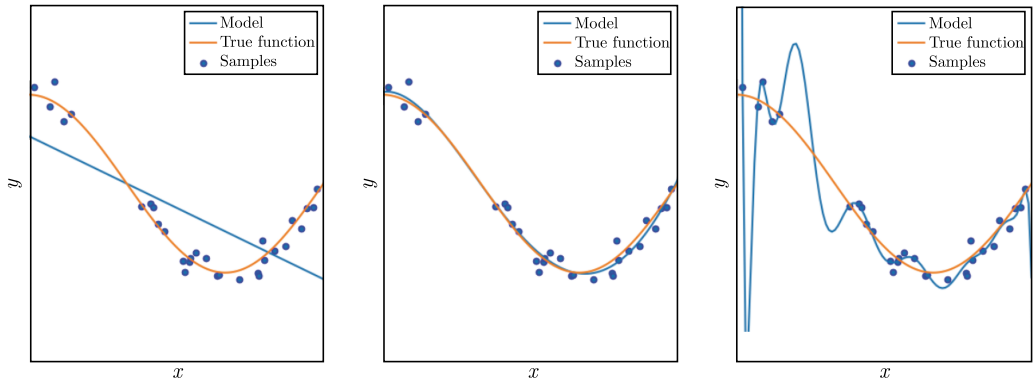

图 3.4 欠拟合 (a)、正常拟合 (b) 和过拟合 (c)

在思考泛化误差的框架下，以上定性分析了存在拟合和过拟合的问题。首先思考如何定性地评价这种现象呢？前面解释不同算法的不同能力的时候谈到 NFL 定理，这里又引入一个算法能力相关的解释，称为奥卡姆剃刀原理 (Principle of Occam’s Razor)。这个原理告诉我们在合理的解释模型里面，最简单的那个模型最佳。用这个原理来解释，3 阶和 6 阶多项式曲线都能较好地解释当前的数据分布了，但是 3 阶要比 6 阶形式上简单很多，所以 3 阶要优于 6 阶。从奥卡姆剃刀原理来说，在选择算法模型的时候，就有以下两种思考。

(1) 增加算法模型复杂度：从简单模型开始尝试，如果效果不好，就增加算法模型的复杂度，直到能够较好地解释当前数据，就停止增加模型的复杂度。

(2) 限制算法模型复杂度：从复杂模型开始尝试，如果效果很好，那么开始限制模型复杂度，直到依然能够较好地解释当前数据，就停止进一步限制模型的复杂度。

第二种思考比第一种思考具有一定优势，即在第一次尝试的时候，就能够明确知道是否将会找到合适的模型。第二种思考，如果这个模型能够解释当前数据，那么接下来只要限制模型就可以找到合适模型。但是第一种思考，却不能有这个直接的判断。所以第二种思考发展出来正则化的方法，作为一种很好地限制模型能力的策略。

有人说奥卡姆剃刀原理和正则化策略的定性思考是挺好的，但是前面的拟合，看上去更像是回归而不是分类问题。如果把这种多项式拟合和分类问题对接起来，当把上述对点的拟合和分类问题联系起来，就能用同样的思考来分析分类问题了 (图 3.5)。首先，多项式拟合看上去是一个回归问题。需要找到合适的线，经过所有的点。但是分类问题，却不是要经过所有的点。用一个两类问题来分析，只考虑两类的边界 (Margin)，假如能拟合出合适的线经过所有边界上的点，那么就能很好地把一个两类问题转换成拟合问题。

这就是基于分类边界的一般性思考。一旦我们能够解决两类问题，就可以合理地解决多类问题，即采用分而治之的思想，先分出一个类别，做二分类。然后再继续分出一个类别，继续做二分类，直到只有两个类别了为止。所以，基于分类边界的思考具有一般性的优点。

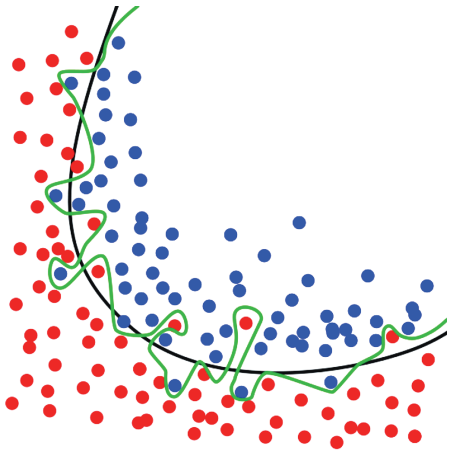

图 3.5　拟合和分类问题

在同一个模型框架下，定性思考的确很好，例如多项式模型，通过增加多项式的阶来增加模型复杂度。但是如何判断不同模型框架的复杂度呢？例如广义线性模型的逻辑回归和非线性的决策树的模型复杂度之间的比较。不同模型框架下，定性思考有一定的局限性，还需要引入定量思考。但是，如何定量思考呢？一种办法就是基于实际数据的分类结果比较，例如基于前面提到的江苏的湖水水质数据来比较逻辑回归和决策树分类。但如果把所有数据作为训练数据，那就无法比较算法的测试效果了，因为没有江苏以外的湖水水质数据了。这时需要数据分组了，一组用来训练，另一组用来测试。可以把江苏水质数据一分为二，一组作为训练集，另一组作为测试集。数据分组要注意尽量随机分，否则选出来的训练分组就没有代表性了。例如把江苏数据分为苏南数据和苏北数据两组，哪一组也不能代表江苏数据。数据分组为训练集和测试集还不够，例如举办个竞赛，测试集是用来看哪个模型最终的效果好。但除了训练集外，需要一个验证集来验证训练好的模型的有效性。这样之前的训练集又分为训练集和验证集。

(1) 训练集 (Train Set)：训练不同的算法模型。

(2) 验证集 (Validation Set)：验证不同模型，选择最合适的模型。

(3) 测试集 (Test Set)：在验证的模型上得到测试准确度。

根据竞赛测试结果，就可以用最好的团队的算法去全国进行湖水水质检验了，进而

进入实际应用。所以，验证集是专门设计用来选择模型的。而这里面就包括验证过拟合的情况。如果一个算法在训练集上的效果好，那么该算法的拟合能力肯定更好。是不是过拟合了，再看验证集效果。但是，有时候大家觉得一个模型和另外一个模型的结果，怎么能够通过一次验证结果来确定呢？这时又提出了交叉验证 (Cross Validation) 的思想。

虽然交叉验证思想解决了如何定性考察算法能力。但是这种定性考察有个很大的局限性，就是受具体数据集的限制。如果想脱离具体数据集，如何定性考察模型的拟合能力呢？又回到分类边界的思想，假设模型能够找到一定的边界的，那么这个边界是否能够正确划分任意的数据分布，就要比较找到的分类边界对任意类别数据的划分能力进行考察了，具体就需要引入 VC 维 (Vapnik and Chervonekis Dimension)。

3.1.4 VC 维

VC 维是 Facebook 人工智能实验室 (Facebook AI Research) 的 Vapnik 提出来的。VC 维里面的 V 就是 Vapnik 的缩写。VC 维伟大的地方在于第一次脱离具体样本数据定量地描述算法模型的拟合能力。同时，VC 维也是机器学习里面计算学习理论中最难点的理论之一。这里仅仅分析 VC 维思想的意义，具体的推导和证明可以参考相关书籍①。

首先 VC 维理论的基石是 Valiant 提出的 PAC 学习。PAC 学习的工作使得 Valiant 获得了 2010 图灵奖。PAC 学习伟大的地方在于给出了一个误差可控的数学模型，在这个数学模型下，算法可以被描述成从已知经验中提取假设，然后根据假设对未知数据做出决策。那么一个误差可控的数学模型的优点在哪里呢？这个数学模型可以很好地按概率收敛的数学理论，尤其是各种概率不等式进行很好的衔接。这种衔接使得后续的推理成为可能，如 VC 维和类似的 Rademacher 复杂度理论都是建立在 PAC 学习的基石和各种概率不等式的基础上的。这些概率不等式包括马尔可夫 (Markov) 不等式、切比雪夫 (Chebyshev) 不等式、霍夫丁 (Hoeffding) 不等式、迈克蒂安米德 (Mcdiarmid) 不等式和均衡定理 (Symmetrization Lemma) 等。类似，在凸优化理论里面，凸函数定义、Lipschitz 连续性、光滑性定义是基础，在这个良好定义的基础上就可以应用后续的 Jensen 不等式、Lyapunov 函数等数学工具，推导出算法收敛。

PAC 学习的是一个概率不等式描述下的误差可控模型，它有以下三方面优点。

(1) 通过限制可控误差、比较概率高低来比较算法的好坏。要达到一个可接受误差，大于 50% 的概率才是一个有意义的算法。而通过高低，就可以划分出强学习器和弱学习器，为以 AdaBoost 算法为代表的 Boosting 思想的讨论打下基石。

① 周志华. 机器学习. 北京：清华大学出版社，2016.

(2) 结合算法计算复杂性 (Computational Complexity) 的思想，在限定多项式时间学习前提下，算法是否能够在可接受误差下达到一定概率，来讨论算法模型的可学习性 (Learnable)。

(3) 结合可打散 (Shattering) 的目标，把算法假设随着数据量的增长不再可打散的上限定义为算法复杂度的度量，从而诞生 VC 维。然后基于算法假设的结果状态空间随着样本量增加而增加定义的增长函数 (Growth Function) 来推理误差上限 (图 3.6)。

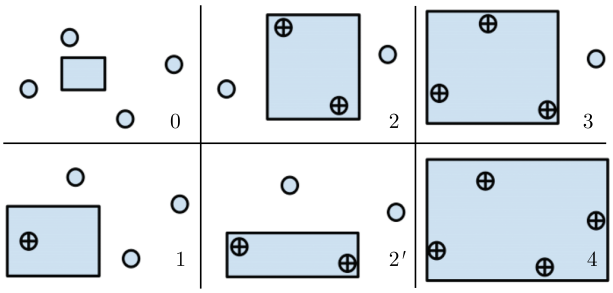

图 3.6　矩形框模型的打散数据量的讨论

对于增长函数，最重要的是它考察的对象是一个集合，当样本数量增加，样本的组合任意变化下会带来结果可能性的变化。大家知道，如果结果的集合对应的可能性越多表示算法的能力越强。但是，受到算法的限制，结果集合组合的可能性不会随着样本的可能性增加而线性地增加。而这种结果集合组合的可能性就是结果状态空间 (图 3.7)。

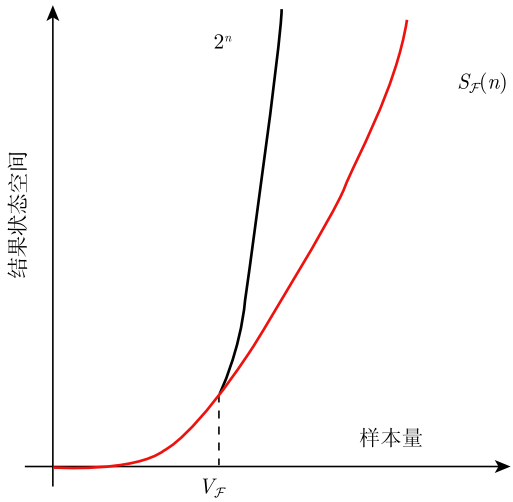

图 3.7　增长函数是用来度量算法的结果状态空间数量是如何随着样本量的增加而增加的

VC 维的考察就是完全看算法的结果集合的状态空间变化，通过增长函数的上限刻

画为 VC 维的表达式。这个过程有以下两个步骤。

(1) VC 维的大小：随着样本数的增加，算法存在从可打散到不可打散的临界样本数，而这个样本数被定义为 VC 维。因为样本数量超过 VC 维时就存在不可被打散的情况，也就是说结果状态空间不再以 2^n 的指数增长 (假设二分类问题，n 个输入样本) (图 3.7)。样本数超过 VC 维之后的继续增长，结果状态数就会存在一个上限。

(2) 泛化误差的大小：误差就是结果和算法输出之间的差异，如果样本量局限于 VC 维，算法输出状态空间能覆盖结果状态空间。但如果存在超过 VC 维的样本，那么必然存在一些结果状态是当前算法难以刻画的情况。所以，把理论上的泛化误差和这种超过 VC 维后结果状态难以刻画的情况建立联系，给出泛化误差上限。基于 VC 维的泛化误差完全不考虑数据的分布情况，仅仅考察算法的能力带来的泛化误差上限。

如果考虑样本集的分布，并且把输出结果和随机结果的相关性上限的期望作为一个复杂度，那么就得到了 Rademacher 复杂度。通过类似的基于不等式的证明，还可以将泛化误差建立在这个 Rademacher 复杂度的基础上。因为考虑了数据分布的复杂度，所以 Rademacher 复杂度可以作为比增长函数更为紧致的一个上限。并且 Rademacher 复杂度和增长函数之间存在恒成立的不等关系，使得基于 Rademacher 复杂度很容易推理出基于 VC 维的泛化误差。所以，这个过程有以下 3 个步骤。

(1) Rademacher 复杂度：通过样本和分布的划分，可以分别计算经验 Rademacher 复杂度和 Rademacher 复杂度 (经验 Rademacher 复杂度在数据空间上的期望)。

(2) 基于 Rademacher 复杂度的泛化误差：误差上限，可以用 Rademacher 复杂度构建表示，根据 McDiarmid 不等式，Rademacher 复杂度可以利用经验 Rademacher 复杂度构建上限，这样误差函数就可以基于经验 Rademacher 复杂度来构建上限，而经验 Rademacher 复杂度可以基于分布计算的。所以直观上，就把分布的影响表达出来了。

(3) 基于 VC 维的泛化误差：利用 Rademacher 复杂度和增长函数的不等式关系，可以将基于 Rademacher 复杂度的泛化误差估算换算到基于 VC 维的泛化误差。

无论是基于 VC 维的还是基于 Rademacher 复杂度的泛化误差计算都是基于集合建模的。VC 维利用了集合的可扩散性，Rademacher 复杂度利用了算法输出集合和随机分类结果集合的相关性。但是，并非所有的泛化误差都是基于集合建模的。如果考虑在线学习 (Online Learning)，那么泛化误差的计算就不是基于集合的，而是要考虑数据的顺序关系。这时考察顺序的情况就是基于树 (Tree) 的分析。基于集合的可打散性，那么集合的度量是集合大小 VC 维。而基于树的分析就是树的高度，称为 Littlestone 维 (Littlestone’s Dimension)。如果不是从打散能力出发，依然从随机集合的相关性出发，这时要考虑顺序 Rademacher(Sequential Rademacher) 复杂度。

3.2 经验风险最小的算法

通过前面的学习了解了经验风险最小的意义。那么，如何通过经验风险最小来选择算法呢？如果把带参数的算法簇作为候选集，那么要求经验风险最小，就相当于寻找最优参数的算法。所以，最优参数可对应于最小经验风险。经验风险 (Empirical Risk, ER) 一般是由一组样本的损失函数 (Loss Function) 之和或者均值来定义的

$$ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) = \frac{1}{n}\sum_{1}^{n} \mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \tag{3.9}$$

这样根据风险最小，可以计算参数

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) \tag{3.10}$$

其中 $\boldsymbol{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n)^\top$, $\boldsymbol{Y} = (\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n)^\top$ 是样本集合。而 $\boldsymbol{\theta}$ 是学习模型的参数。

通常，每个样本的损失函数可以由两种定义角度去定义。

(1) 误差函数 (Error Function, ERF)

$$\mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = ERF(f(\boldsymbol{x}_i; \boldsymbol{\theta}), \boldsymbol{y}_i) \tag{3.11}$$

$$= \begin{cases} (f(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i)^2 & \text{如果误差函数是平方误差} \\ |f(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i| & \text{如果误差函数是绝对值误差} \end{cases} \tag{3.12}$$

常见的 ERF 有平方误差 (Squared Error, SE) 和绝对值误差 (Absolute Error, AE)。这样对应的经验风险就可以是均方差 (MSE) 或均绝对值差 (MAE)。

(2) 负的 log 似然函数 (Negative Log Likelihood, NLL)

$$\mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = -\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = -\ln P(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \tag{3.13}$$

当然如果对于连续情况下，也可以直接利用概率密度函数来计算似然函数。

$$\mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = -\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = -\ln p(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \tag{3.14}$$

那么经验风险就可以看成是样本集合的 NLL。

$$ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) = \frac{1}{n}\sum_{1}^{n} -\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = \frac{1}{n}\sum_{1}^{n} -\ln p(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \tag{3.15}$$

$$= -\frac{1}{n}\ln\prod_{1}^{n} p(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \tag{3.16}$$

$$= -\frac{1}{n}\ln p(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) \tag{3.17}$$

这两种方式内在是有一定的联系的。如果定义残差 (Residual) 为学习模型预测值与真实值之差

$$r(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = f(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i \tag{3.18}$$

那么，最小平方误差等价于残差符合高斯分布 (正态分布) 下的最小 NLL。而最小绝对值误差等价于残差符合拉普拉斯分布下的最小 NNL，具体推理过程就省略了。

$$r(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \sim \mathcal{N}(0, \sigma^2) \Longrightarrow \tag{3.19}$$

$$\arg\min_{\boldsymbol{\theta}} (f(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i)^2 \Leftrightarrow \arg\min_{\boldsymbol{\theta}} -\ln \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{(f(\boldsymbol{x}_i;\boldsymbol{\theta})-\boldsymbol{y}_i)^2}{2\sigma^2}} \tag{3.20}$$

$$r(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) \sim \mathcal{L}aplace(0, b) \Longrightarrow \tag{3.21}$$

$$\arg\min_{\boldsymbol{\theta}} |f(\boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i| \Leftrightarrow \arg\min_{\boldsymbol{\theta}} -\ln \frac{1}{2b} \mathrm{e}^{-\frac{|f(\boldsymbol{x}_i;\boldsymbol{\theta})-\boldsymbol{y}_i|}{b}} \tag{3.22}$$

下面来比较一下损失函数是平方误差和绝对值误差这两种不同情况。

(1) 平方误差。

① 等价于残差符合高斯分布的 NLL。

② 残差较小 (< 1) 的数据分配的权重较小，而残差较大 (> 1) 数据分配权重较大，因此对残差较大项的有抑制效果 (图 3.8)。

③ 连续光滑，容易求梯度 (导数)。

(2) 绝对值误差。

① 等价于残差符合拉普拉斯分布的 NLL。

② 对残差较小 (< 1) 和较大 (> 1) 的数据分配权重平均，因此对不同的残差项同等看待 (图 3.8)。

③ 不光滑，不容易求梯度 (因此有人提出了光滑绝对值误差 (Smoothed Absolute Error)，是一个分段函数，称为 Huber 函数)。

上面简单描述了经验风险的两大类损失函数，即误差函数和负对数似然。并且解释了常见的两种误差，即平方误差和绝对值误差，两种误差都可以利用负对数似然来解释。不过，两种常见的损失函数一般都是用于回归分析。例如，对于最小二乘法的回归，就是设定线性的回归线，然后基于平方误差与经验风险最小，就可以求解到最小二乘法的值。如果用向量表示，假设线性回归线为 $f(\boldsymbol{X}; \boldsymbol{\beta}) = \boldsymbol{X}\boldsymbol{\beta}$，那么经验风险就是。

$$ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\beta}) = (f(\boldsymbol{X}; \boldsymbol{\beta}) - \boldsymbol{Y})^\top (f(\boldsymbol{X}; \boldsymbol{\beta}) - \boldsymbol{Y}) \tag{3.23}$$

$$= (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Y})^\top (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Y}) \tag{3.24}$$

$$= \boldsymbol{Y}^\top \boldsymbol{Y} - 2\boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{Y} + \boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} \tag{3.25}$$

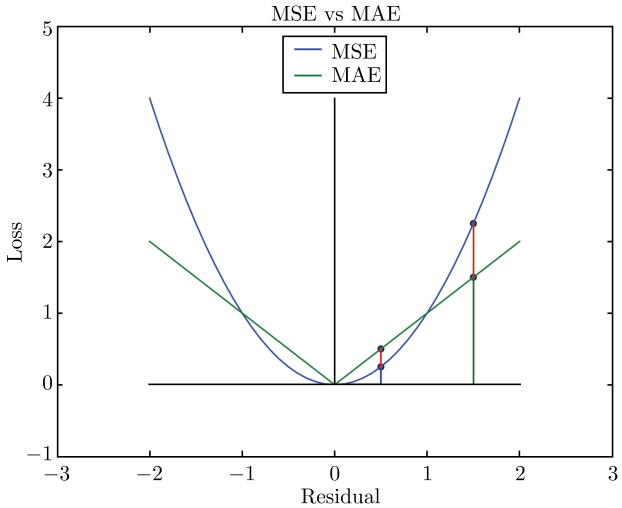

图 3.8　比较平方误差和绝对值误差

根据最小经验风险，对参数 $\boldsymbol{\beta}$ 求导

$$\frac{\partial ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^\top \boldsymbol{Y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} = 0 \tag{3.26}$$

那么，求解结果就是最小二乘法的矩阵表示

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\beta}) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y} = \boldsymbol{A}^+ \boldsymbol{Y} \tag{3.27}$$

3.3　分类边界

分类是一种特殊的回归，因此常用的损失函数不太一样，一般分类中都是基于基本的两类问题，那么结果 $\boldsymbol{y}_i \in \{0, 1\}$，但是 0/1 的标签对称性不好，所以也常利用 $\boldsymbol{y}_i \in \{-1, 1\}$。那么这两种分类标签会带来什么不同呢？我们基于逻辑回归分类算法来解读一下。

3.3.1　分类算法的损失函数

例如，逻辑回归分类算法，一般如何计算损失呢？假设一个两类问题 $y_i \in \{0, 1\}$，最直接的方法就是数一下样本分错的数目，即 0-1 指示函数 (Indicator Function)。

$$Err = \sum_{1}^{n} \mathbb{I}(y_i \neq f(\boldsymbol{x}_i; \boldsymbol{\theta})) \tag{3.28}$$

在最初介绍逻辑回归中，对应的目标的 y_i 取值是 0 或 1，这样在把损失函数定义为负的对数似然 (NLL) 时形式可以很简洁。如果分类问题 y_i 取值换成是 -1 或 1 这样一个对

称的形式，其实更具有优势。这样定义之后，y_i 与 $f(\boldsymbol{x}_i|\boldsymbol{\theta})$ 同号即表示样本分类分对了，定义

$$\mathbb{I}(y_i = f(\boldsymbol{x}_i;\boldsymbol{\theta})) \Leftrightarrow y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta}) = 1 \Leftrightarrow -y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta}) = -1 \tag{3.29}$$

$$\mathbb{I}(y_i \neq f(\boldsymbol{x}_i;\boldsymbol{\theta})) \Leftrightarrow y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta}) = -1 \Leftrightarrow -y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta}) = 1 \tag{3.30}$$

这样损失函数便可以写成单位阶跃函数 (Heaviside Step Function) 的形式

$$\text{Loss} = \mathbb{H}(-y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta})) \Longleftrightarrow \mathbb{H}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geqslant 0 \end{cases} \tag{3.31}$$

一般地，分类问题的损失函数通常表示为 $y_i * f(\boldsymbol{x}_i;\boldsymbol{\theta})$ 形式的函数。

$$\text{Loss}(f(\boldsymbol{x}_i;\boldsymbol{\theta}), y_i) = \phi(y_i f(\boldsymbol{x}_i;\boldsymbol{\theta})) \Longleftrightarrow \phi(x) = \begin{cases} 0, & x > 0 \\ 1, & x \leqslant 0 \end{cases} = \mathbb{I}(x \leqslant 0) \tag{3.32}$$

在前面章节中逻辑回归中的损失函数以负的对数似然来定义，如果把 y_i 的取值由 $\{0,1\}$ 变成了 $\{-1,1\}$，损失函数需要相应地发生变化 (依然以负的对数似然的方式来定义)。对于样本 $(\boldsymbol{x}_i, y_i)$，Logistic 函数同样可以理解为 $y_i = 1$ 时的概率

$$P(\boldsymbol{x}_i) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}} \tag{3.33}$$

此时，逐步代入具体 Logistic 表达式，那么损失函数为

$$\text{Loss}(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}) = \begin{cases} -\ln P(\boldsymbol{x}_i), & y_i = 1 \\ -\ln 1 - P(\boldsymbol{x}_i), & y_i = -1 \end{cases} \tag{3.34}$$

$$= \begin{cases} -\ln \dfrac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}, & y_i = 1 \\ -\ln 1 - \dfrac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}, & y_i = -1 \end{cases} \tag{3.35}$$

$$= \begin{cases} \ln 1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}, & y_i = 1 \\ -\ln \dfrac{\mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}{1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}, & y_i = -1 \end{cases} \tag{3.36}$$

$$= \begin{cases} \ln 1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}, & y_i = 1 \\ \ln \dfrac{1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}{\mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}}, & y_i = -1 \end{cases} \tag{3.37}$$

$$= \begin{cases} \ln 1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}, & y_i = 1 \\ \ln \mathrm{e}^{\boldsymbol{\theta}^\top \boldsymbol{x}_i} + 1, & y_i = -1 \end{cases} \tag{3.38}$$

$$= \begin{cases} \ln 1 + \mathrm{e}^{-\boldsymbol{\theta}^\top \boldsymbol{x}_i}, & y_i = 1 \\ \ln 1 + \mathrm{e}^{\boldsymbol{\theta}^\top \boldsymbol{x}_i}, & y_i = -1 \end{cases} \tag{3.39}$$

$$= \ln 1 + \mathrm{e}^{-y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i)} \tag{3.40}$$

则经验风险为

$$ER(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) = \frac{1}{n} \sum_{1}^{n} \ln 1 + \mathrm{e}^{-y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i)} \tag{3.41}$$

再来对比一下逻辑回归的两种损失函数形式，虽然都是对数似然函数 (NLL)。

第一种，Logistic 函数输出值可以解读成概率值 $P(\boldsymbol{x}_i; \boldsymbol{\theta}) = \dfrac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta} \boldsymbol{x}_i}}$，而目标标签又是 $\boldsymbol{y}_i \in \{0, 1\}$。那么对于概率表示的似然度 (Likelihood) 有

$$L(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = \begin{cases} P(\boldsymbol{x}_i; \boldsymbol{\theta}), & \boldsymbol{y}_i = 1 \\ 1 - P(\boldsymbol{x}_i; \boldsymbol{\theta}), & \boldsymbol{y}_i = 0 \end{cases} \tag{3.42}$$

这时为了表示为统一的表达式，利用了 0/1 指数的良好性质有

$$L(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = P(\boldsymbol{x}_i; \boldsymbol{\theta})^{\boldsymbol{y}_i} (1 - P(\boldsymbol{x}_i; \boldsymbol{\theta}))^{1 - \boldsymbol{y}_i} \tag{3.43}$$

第二种，目标是对称的 $\boldsymbol{y}_i \in \{-1, 1\}$，那么

$$L(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = \begin{cases} P(\boldsymbol{x}_i; \boldsymbol{\theta}) = \dfrac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta} \boldsymbol{x}_i}}, & \boldsymbol{y}_i = 1 \\ 1 - P(\boldsymbol{x}_i; \boldsymbol{\theta}) = \dfrac{1}{1 + \mathrm{e}^{\boldsymbol{\theta} \boldsymbol{x}_i}}, & \boldsymbol{y}_i = -1 \end{cases} \tag{3.44}$$

这时，利用 $-1/1$ 的良好对称性

$$L(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{y}_i) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{y}_i \boldsymbol{\theta} \boldsymbol{x}_i}} \tag{3.45}$$

所以，大家要注意的是逻辑回归采用不同的目标数字化标签，对数的似然度是不一样的，那么对应的损失是负的对数似然度也不一样。如果采用 $\boldsymbol{y}_i \in \{0, 1\}$，那么损失为

$$\mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = -\ln\{P(\boldsymbol{x}_i; \boldsymbol{\theta})^{\boldsymbol{y}_i} (1 - P(\boldsymbol{x}_i; \boldsymbol{\theta}))^{1 - \boldsymbol{y}_i}\} \tag{3.46}$$

$$= -\left[\boldsymbol{y}_i \ln\left\{\frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta} \boldsymbol{x}_i}}\right\} + (1 - \boldsymbol{y}_i) \ln\left\{1 - \frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta} \boldsymbol{x}_i}}\right\}\right] \tag{3.47}$$

$$= \boldsymbol{y}_i \ln\{1 + \mathrm{e}^{-\boldsymbol{\theta} \boldsymbol{x}_i}\} + (1 - \boldsymbol{y}_i) \ln\{1 + e^{\boldsymbol{\theta} \boldsymbol{x}_i}\} \tag{3.48}$$

但如果采用 $\boldsymbol{y}_i \in \{-1, 1\}$，那么损失为

$$\mathrm{Loss}(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = -\ln\left\{\frac{1}{1 + \mathrm{e}^{-\boldsymbol{y}_i \boldsymbol{\theta} \boldsymbol{x}_i}}\right\} \tag{3.49}$$

$$= \ln\{1 + \mathrm{e}^{-\boldsymbol{y}_i \boldsymbol{\theta} \boldsymbol{x}_i}\} \tag{3.50}$$

对应的表达式 $\boldsymbol{y}_i \in \{-1, 1\}$ 要明显简单。

3.3.2 分类算法的边界

如果把 $f(\boldsymbol{x}_i) = \boldsymbol{\theta}\boldsymbol{x}_i$ 看成样本空间上的一条直线。那么，$\boldsymbol{y}_i\boldsymbol{\theta}\boldsymbol{x}_i = \boldsymbol{y}_i f(\boldsymbol{x}_i)$ 被称为边界 (Margin)。事实上，$y_i f(\boldsymbol{x}_i)$(其中 $y_i \in \{-1, 1\}$) 这个定义来自支持向量机。对于一个二分类问题，在线性可分的情况下 (图 3.9)，分属于两个类别的数据 $(\boldsymbol{x}_i, y_i = 1)$ 和 $(\boldsymbol{x}_j, y_j = -1)$，有如下关系成立

$$\begin{cases} f(\boldsymbol{x}_i) = \boldsymbol{\theta}^\top \boldsymbol{x}_i + b \geqslant 1, & y_i = 1 \\ f(\boldsymbol{x}_j) = \boldsymbol{\theta}^\top \boldsymbol{x}_j + b \leqslant -1, & y_j = -1 \end{cases} \Leftrightarrow y f(\boldsymbol{x}) \geqslant 1 \tag{3.51}$$

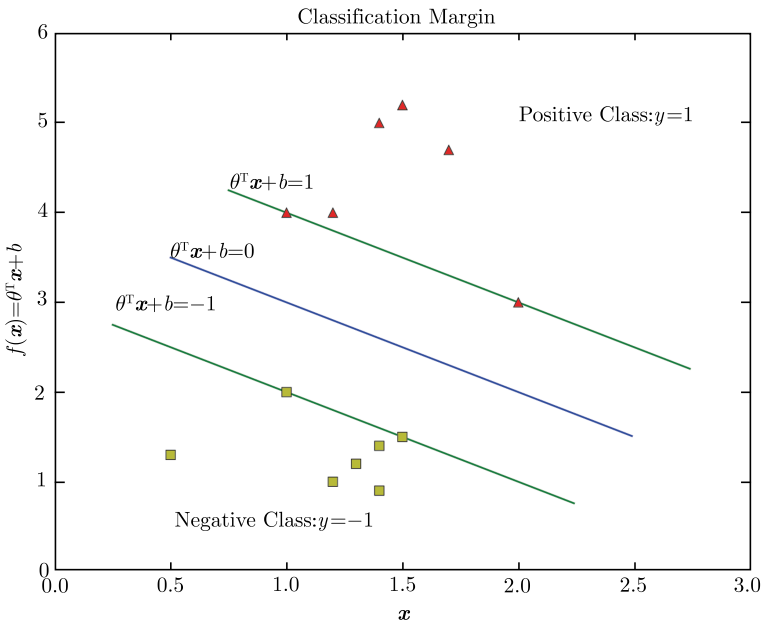

图 3.9 分类界限

这样就把分类边界 (Classification Margin) 定义如下

$$CM(\boldsymbol{x}, y; \boldsymbol{\theta}) = y f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{3.52}$$

从图 3.9 可以看出，两条边界 (支持向量所在直线) 之间 $f(\boldsymbol{x})$ 值相差了 2。而 $2 = 2yf(\boldsymbol{x})$，所以有些教材也把分类界限定义成 $2yf(\boldsymbol{x})$。边界的含义也比较清楚，就是 $f(\boldsymbol{x}_i)$

是样本空间上的一个划分面，如果正确分类，根据目标标签 $\boldsymbol{y}_i \in \{-1,1\}$ 的对称性，那么期望 $f(\boldsymbol{x}_i)$ 与 $\boldsymbol{y}_i$ 的符号相同。当错误分类时，$f(\boldsymbol{x}_i)$ 与 $\boldsymbol{y}_i$ 的符号相异。那么，$\boldsymbol{y}_i f(\boldsymbol{x}_i) < 0$ 就是错误分类的情况，需要计算损失。最简单的是 0-1 计数损失。

$$L(\boldsymbol{y}_i f(\boldsymbol{x}_i)) = \begin{cases} 0, & \boldsymbol{y}_i f(\boldsymbol{x}_i) > 0 \\ 1, & \boldsymbol{y}_i f(\boldsymbol{x}_i) < 0 \end{cases} \tag{3.53}$$

但是逻辑回归的计算表达式是 $\ln\{1+\mathrm{e}^{-\boldsymbol{y}_i f(\boldsymbol{x}_i)}\}$。在 SVM 中，对应的损失称为链损失 (Hinge Loss)，对应的表达式为 $\max(0, 1-\boldsymbol{y}_i f(\boldsymbol{x}_i))$ (参考图 3.10)。更进一步，有时边界对应的不是一条直线，而是组合线，如 AdaBoost 对应的损失函数是指数函数 $\mathrm{e}^{-\boldsymbol{y}_i H(\boldsymbol{x}_i)}$，其中 $H(\boldsymbol{x}_i) = \mathrm{sign}(a_1 h_1(\boldsymbol{x}) + a_2 h_2(\boldsymbol{x}) + a_3 h_3(\boldsymbol{x}))$。

损失函数 $H(\boldsymbol{x})$ 是一个线性组合 (参考图 3.10)。

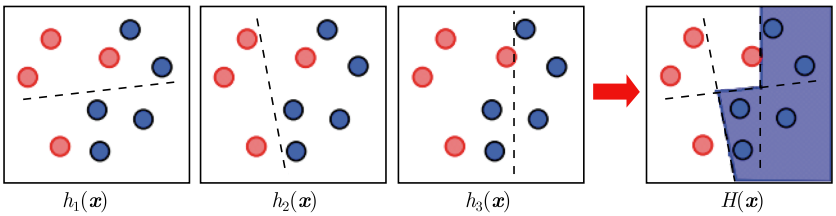

图 3.10　AdaBoost 对应的组合边界 $H(\boldsymbol{x}) = \mathrm{sign}(a_1 h_1(\boldsymbol{x}) + a_2 h_2(\boldsymbol{x}) + a_3 h_3(\boldsymbol{x}))$

SVM 的损失函数，以 CM 来写的话是铰链损失函数

$$\mathrm{Loss}(\boldsymbol{x}, y|\boldsymbol{\theta}) = \max\left(0, 1 - y f(\boldsymbol{x}|\boldsymbol{\theta})\right) \Leftrightarrow \phi(x) = \max(0, 1-x) \tag{3.54}$$

为了让这些不同的损失函数在 $CM=0$ 时有相同的取值，对于 Logistic Loss 通常做一个归一化处理

$$\phi(x) = \frac{1}{\ln 2}\ln\{1+\mathrm{e}^{-x}\} \tag{3.55}$$

于是可以把 Logistic Loss 和 Hinge Loss 可以看成是 0-1 Loss 的一个上限 (参见图 3.11)，而 0-1 Loss 的另外一个常用的上限是指数损失 (Exponential Loss)

$$\mathbb{I}(x \leqslant 0) \leqslant \mathrm{e}^{-x} \Leftrightarrow \phi(x) = \mathrm{e}^{-x} \tag{3.56}$$

而这个对数损失对应的就是 AdaBoost 算法的损失函数 (参见图 3.12)。这个上限在 AdaBoost 误差收敛性证明中也会用到。

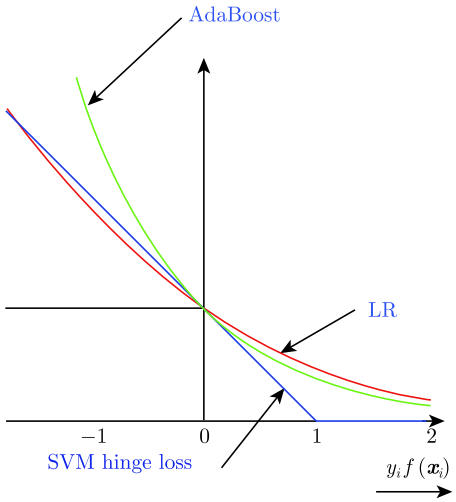

图 3.11 AdaBooost 的指数损失、0-1 计数损失、LR 的 log 损失和 SVM 的 hinge 损失

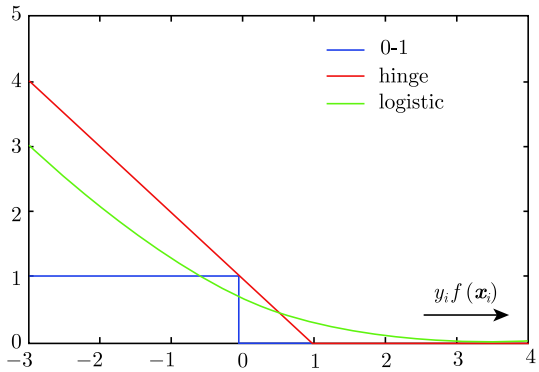

图 3.12 0-1 计数损失、LR 的 log 损失和 SVM 的 hinge 损失

损失函数对应的算法如表 3.1 所示。

表 3.1 损失函数

损失函数	函数形式	对应算法
0-1 损失 (Zero-One Loss)	$\mathbb{I}(x \leqslant 0)$	Linear Binary Classification
铰链损失 (Hinge Loss)	$\phi(x) = \max(0, 1-x)$	SVM
逻辑损失 (Logistic Loss)	$\phi(x) = \dfrac{1}{\ln 2}\ln\{1+\mathrm{e}^{-x}\}$	Logistic Regression
指数损失 (Exponential Loss)	$\phi(x) = \mathrm{e}^{-x}$	AdaBoost

表 3.1 总结了这四种损失函数对应的函数形式及对应的算法。基于经验风险最小和对应的损失函数，那么得到对应的逻辑回归，支持向量机和 AdaBoost 算法的表达式

如下。

$$\hat{\boldsymbol{w}}_{\mathrm{LR}} = \arg\min_{\boldsymbol{w}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \ln\{1 + \exp(-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))\} \right\} \tag{3.57}$$

$$\hat{\boldsymbol{w}}_{\mathrm{SVM}} = \arg\min_{\boldsymbol{w}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))\} + \lambda \boldsymbol{w}^\top \boldsymbol{w} \right\} \tag{3.58}$$

$$\hat{\boldsymbol{w}}_{\mathrm{AdaBoost}} = \arg\min_{w_i, h_i} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp\{-y_i H(\boldsymbol{x}_i)\} \right\}, \quad H(\boldsymbol{x}_i) = \mathrm{sign} \sum_{t=1}^{T} w_t h_t(\boldsymbol{x}_i) \tag{3.59}$$

其中支持向量机部分的形式有点不一样，后面的二次项 $\boldsymbol{w}^\top \boldsymbol{w}$ 是正则化项，将在下一章深入解释。

3.4 小结

通过泛化误差理论引述了经验风险最小的意义，然后通过经验风险最小，描述了通用的损失函数的形式，并区分了回归和分类问题的损失函数。强调了在定义分类问题的损失函数时目标编码的意义。最后通过两类问题的编码，引出分类边界的思想和基于分类边界的常用算法及对应的损失函数。

参 考 文 献

[1] Barron, Andrew R. Approximation and Estimation Bounds for Artificial Neural Networks[J]. Machine Learning, 1994, 115-133.

[2] Kearns, Michael J. and Umesh V. Vazirani. An Introduction to Computational Learning Theory[M]. Cambridge: MIT Press.

[3] Manurangsi, Pasin and Aviad Rubinstein. Inapproximability of VC Dimension and Littlestone's Dimension[J]. CoRR abs/1705.09517. arXiv: 1705. 09517. url: http://arxiv.org/abs/1705.09517.2007.

[4] Rosasco, Lorenzo et al. Are Loss Functions All the Same?[J] Neural Comput. 2004, 1063-1076.

[5] Valiant, Leslie. Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World. Basic Books, 2013.

[6] Vapnik, V. N. and A. Ya. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities[J]. Theory of Probability & Its Applications, 1971.

[7] Wolpert, D. H. and W. G. Macready. No Free Lunch Theorems for Optimization[J]. Trans. Evol. Comp, 1997.

[8] 周志华. 机器学习 [M]. 北京：清华大学出版社，2016.