

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.1 Plots of $\text{sigm}(w_1x_1 + w_2x_2)$. Here $w = (w_1, w_2)$ defines the normal to the decision boundary. Points to the right of this have $\text{sigm}(w^T x) > 0.5$, and points to the left have $\text{sigm}(w^T x) < 0.5$. Based on Figure 39.3 of (MacKay 2003).

8 Logistic regression

8.1 Introduction

One way to build a probabilistic classifier is to create a joint model of the form $p(y, x)$ and then to condition on x , thereby deriving $p(y|x)$. This is called the generative approach. An alternative approach is to fit a model of the form $p(y|x)$ directly. This is called the discriminative approach, and is the approach we adopt in this chapter. In particular, we will assume discriminative models which are linear in the parameters. This will turn out to significantly simplify model fitting, as we will see. In Section 8.6, we compare the generative and discriminative approaches, and in later chapters, we will consider non-linear and non-parametric discriminative models.

8.2 Model specification

As we discussed in Section 1.4.6, logistic regression corresponds to the following binary classification model:
 $p(y|x,w) = \text{Ber}(y|\text{sigm}(w^T x))$ (8.1)

A 1d example is shown in Figure 1.19(b). Logistic regression can easily be extended to higher-dimensional inputs. For example, Figure 8.1 shows plots of $p(y=1|x,w) = \text{sigm}(w^T x)$ for 2d input and different weight vectors w . If

we threshold these probabilities at 0.5, we induce a linear decision boundary, whose normal (perpendicular) is given by w .

已思考 58s

译文

图 8.1 画出了函数 $\text{sigm}(w_1x_1 + w_2x_2)$ 的曲面图。这里 $w = (w_1, w_2)$ 给出了决策边界 (decision boundary) 的法向量。该边界右侧的点满足 $\text{sigm}(w^\top x) > 0.5$, 左侧的点满足 $\text{sigm}(w^\top x) < 0.5$ 。本图基于 (MacKay, 2003) 的图 39.3。

8 逻辑回归 (Logistic regression)

8.1 引言

构建概率分类器的一种方法，是先建立形如 $p(y, x)$ 的联合模型，然后在给定 x 的条件下求 $p(y | x)$ 。这称为生成式 (generative) 方法。另一种做法是直接拟合形如 $p(y | x)$ 的模型。这称为判别式 (discriminative) 方法，也是本章采用的方法。特别地，我们将假设模型在参数上是线性的。正如我们将看到的，这会使模型拟合大大简化。在第 8.6 节，我们将比较生成式与判别式方法；在后续章节，我们还会考虑非线性与非参数的判别式模型。

8.2 模型说明

正如在第 1.4.6 节讨论的，逻辑回归对应如下的二分类模型：

$$p(y | x, w) = \text{Ber}(y | \text{sigm}(w^\top x)) \quad (8.1)$$

(一维示例见图 1.19(b)。) 逻辑回归很容易推广到更高维的输入。例如，图 8.1 展示了在二维输入、不同权重向量 w 下

$$p(y = 1 | x, w) = \text{sigm}(w^\top x)$$

的图像。如果我们把这些概率按 0.5 进行阈值化，就得到一条线性决策边界，其法向量 (垂直方向) 由 w 给出。

解释 (通俗且详尽)

1) 符号都表示什么？

- $x = (x_1, x_2, \dots)$: 样本的特征 (输入)。图中是二维，只画了 x_1, x_2 。
- $y \in \{0, 1\}$: 标签 (输出)，1 表示“正类”，0 表示“负类”。
- $w = (w_1, w_2, \dots)$: 要学习的参数 (权重)。

- $w^\top x = \sum_i w_i x_i$: 点积/线性组合, 把多维特征压成一个数。
- $\text{sigm}(z) = \frac{1}{1 + e^{-z}}$: S 形 (Sigmoid/Logistic) 函数, 把任何实数 z 压到区间 $(0,1)$, 可当作“概率”。

2) 模型在做什么?

逻辑回归用两步把“特征 \rightarrow 概率”:

1. 先算线性打分: $z = w^\top x$ 。
2. 再过 Sigmoid: $p = \text{sigm}(z) = \frac{1}{1 + e^{-z}}$, 得到“属于 1 类”的概率 $p(y = 1 | x, w)$ 。

于是:

- $p(y = 1 | x, w) = \text{sigm}(w^\top x)$
- $p(y = 0 | x, w) = 1 - \text{sigm}(w^\top x)$

Ber(\cdot) 表示伯努利分布: 对 $y \in \{0, 1\}$,

$$\text{Ber}(y | p) = p^y(1 - p)^{1-y}.$$

所以式 (8.1) 只是把“概率等于 $\text{sigm}(w^\top x)$ ”用分布的方式写出来。

3) 为什么 0.5 是分界?

Sigmoid 的中点在 $z = 0$:

- 当 $w^\top x = 0$ 时, $\text{sigm}(0) = 0.5$ 。
- 当 $w^\top x > 0$ 时, $\text{sigm}(z) > 0.5$, 更偏向标签 1;
- 当 $w^\top x < 0$ 时, $\text{sigm}(z) < 0.5$, 更偏向标签 0。

把“概率 ≥ 0.5 判为 1, 否则判为 0”, 就得到决策边界:

$$\{x : w^\top x = 0\}.$$

在二维里, 这是一条直线; 在三维是一个平面; 更高维是一个超平面。

4) w 决定边界“朝向”和“陡峭度”

- **朝向 (法向量)** : 边界的“垂直方向”就是 w 。直线的斜率由 $-w_1/w_2$ 决定 (当 $w_2 \neq 0$)。
- **哪一侧是类 1?** 往 w 指向的一侧 (使 $w^\top x$ 变大的方向) 是“概率 > 0.5 ”的那一侧。图注里说的“右侧/左侧”, 就是指相对于 w 的方向。
- **陡峭度:** $\|w\|$ 越大, Sigmoid 从 0 到 1 过渡越“陡” (对分界更“自信”) ; $\|w\|$ 小则过渡更“缓”。

5) 图 8.1 怎么读?

图中每个小三维曲面对应一个不同的 $w = (w_1, w_2)$ 。横轴是 x_1, x_2 , 竖轴是 $\text{sigm}(w_1x_1 + w_2x_2)$, 即预测为 1 的概率。

- 例如 $w = (2, 2)$ 时, 沿着 x_1 或 x_2 增大, $w^\top x$ 变大, 曲面迅速升到接近 1;
- 若 $w = (2, -2)$, 说明 x_1 增大有利于类 1, 但 x_2 增大反而对类 1 不利 (因为权重为负) ;
- 不同的 w 让直线 $w_1x_1 + w_2x_2 = 0$ 的方向不同, 这就是不同的决策边界。

6) 生成式 vs 判别式 (易懂版)

- **生成式:** 先学“数据是怎么来的”, 即学 $p(y, x)$ (比如学每一类的分布长啥样), 再用公式算 $p(y | x)$ 。
- **判别式:** 不管数据怎么来, 直接学“给定 x 时 y 的概率” $p(y | x)$ 。
本章选判别式, 并且假设“对参数线性” (像 $w^\top x$ 这样), 这样训练更简单。

7) 一个常见小补充 (偏置项)

很多教材把偏置/截距 b 写出来: $\text{sigm}(w^\top x + b)$ 。也可把它并入向量里: 在 x 前面加一维常数 1 (记作 $x_0 = 1$), 再把 b 合并到 w_0 。

- **没有偏置时**, 边界 $w^\top x = 0$ 必过原点;
- **有偏置时**, 边界可以任意平移, 更灵活。

8) 用它做分类的三步

1. 计算 $z = w^\top x$ (或 $w^\top x + b$) 。
2. 概率 $p = \text{sigm}(z)$ 。
3. 若 $p \geq 0.5$, 判为 1, 否则判为 0。 (实际应用中阈值也可按需求调整。)

这样, 你既能得到“硬分类” (0/1), 也能得到“置信度” (概率)。

翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

8.3 Model fitting

In this section, we discuss algorithms for estimating the parameters of a logistic regression model.

8.3.1 MLE

The negative log-likelihood for logistic regression is given by

$$\text{NLL}(w) = -\sum_{i=1}^N \log[\mu_i l(y_i=1) + (1 - \mu_i)l(y_i=0)] \quad (8.2)$$

$$= -N \sum_{i=1}^N [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)] \quad (8.3)$$

This is also called the cross-entropy error function (see Section 2.8.2). Another way of writing this is as follows.

Suppose $\tilde{y}_i \in \{-1, +1\}$ instead of $y_i \in \{0, 1\}$. We have $p(y = 1) = 1 / (1 + \exp(-w^T x))$ and $p(y = -1) = \exp(-w^T x) / (1 + \exp(-w^T x))$. Hence

$$NLL(w) = -N \sum_{i=1}^N \tilde{y}_i \log(1 + \exp(-\tilde{y}_i w^T x)) \quad (8.4)$$

Unlike linear regression, we can no longer write down the MLE in closed form. Instead, we need to use an optimization algorithm to compute it. For this, we need to derive the gradient and Hessian. In the case of logistic regression, one can show (Exercise 8.3) that the gradient and Hessian of this are given by the following

$$\mathbf{g} = \frac{\partial}{\partial w} f(w) = \sum_{i=1}^N (\mu_i - y_i) x_i = X^T(\mu - y) \quad (8.5)$$

$$\mathbf{H} = \frac{\partial^2}{\partial w^2} g(w)^T = \sum_{i=1}^N (\nabla w \mu_i) x_i x_i^T = \sum_{i=1}^N \mu_i (1 - \mu_i) x_i x_i^T = X^T S X \quad (8.6)$$

where $S = \text{diag}(\mu_i(1 - \mu_i))$. One can also show (Exercise 8.3) that H is positive definite. Hence the NLL is convex and has a unique global minimum. Below we discuss some methods for finding this minimum.

已思考 38s

下面先给逐字完整翻译，再做通俗解释（带上关键式子的来龙去脉）。我会尽量把每个符号都说清楚。

译文

8.3 模型拟合 (Model fitting)

本节我们讨论用于估计逻辑回归模型参数的算法。

8.3.1 极大似然估计 (MLE)

逻辑回归的负对数似然 (negative log-likelihood) 为

$$NLL(w) = -\sum_{i=1}^N \log[\mu_i^{y_i=1} \times (1 - \mu_i)^{y_i=0}] \quad (8.2)$$

$$= -\sum_{i=1}^N [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)] \quad (8.3)$$

这也被称为交叉熵误差函数（见第 2.8.2 节）。还可以用另一种方式来写。假设把标签写成 $\tilde{y}_i \in \{-1, +1\}$ ，而不是 $y_i \in \{0, 1\}$ 。此时有

$$p(y = +1) = \frac{1}{1 + \exp(-w^\top x)}, \quad p(y = -1) = \frac{1}{1 + \exp(+w^\top x)}.$$

因此

$$\text{NLL}(w) = \sum_{i=1}^N \log(1 + \exp(-\tilde{y}_i w^\top x_i)). \quad (8.4)$$

与线性回归不同，这里我们不能再写出闭式 (closed form) 的 MLE 解。我们需要用**优化算法**来数值计算它。为此，我们要推导梯度 (gradient) 和海森矩阵 (Hessian)。对逻辑回归，可以证明 (练习 8.3) 它们为

$$g = \frac{d}{dw} f(w) = \sum_i (\mu_i - y_i) x_i = X^\top (\mu - y), \quad (8.5)$$

$$H = \frac{d}{dw} g(w)^\top = \sum_i (\nabla_w \mu_i) x_i^\top = \sum_i \mu_i (1 - \mu_i) x_i x_i^\top \quad (8.6)$$

$$= X^\top S X, \quad (8.7)$$

其中 $S = \text{diag}(\mu_i(1 - \mu_i))$ 。还可以证明 (练习 8.3) H 是**正定的**。因此 NLL 是**凸函数**，并且只有唯一的全局最小值。下面我们讨论一些寻找该最小值的方法。

注：上面“ $p(y = 1) = \frac{1}{1+\exp(+w^\top x)}$ ”处，按上下文应为 $p(y = -1) = \frac{1}{1+\exp(+w^\top x)}$ 。我在译文里已按常规定义写成 $y = -1$ 。

解释 (基础且详尽)

0) 背景与符号

- 共有 N 个样本。第 i 个样本的特征是 $x_i \in \mathbb{R}^D$ ，标签是 $y_i \in \{0, 1\}$ (或 $\tilde{y}_i \in \{-1, +1\}$)。
- 参数 $w \in \mathbb{R}^D$ 是我们要学习的“权重”。
- 记 $z_i = w^\top x_i$, $\sigma(z) = \frac{1}{1 + e^{-z}}$ 为 Sigmoid。定义

$$\mu_i = \sigma(z_i) = \sigma(w^\top x_i) = p(y_i = 1 \mid x_i, w).$$

- 记数据矩阵 $X \in \mathbb{R}^{N \times D}$ (第 i 行是 x_i^\top)，向量 $\mu = [\mu_1, \dots, \mu_N]^\top$, $y = [y_1, \dots, y_N]^\top$ 。

1) 为什么式 (8.2) 与 (8.3) 长这样？

- 对于单个样本，若 $y_i = 1$ ，模型给出的概率是 μ_i ；若 $y_i = 0$ ，概率是 $1 - \mu_i$ 。

- $\mathbf{I}(\cdot)$ 是指示函数：条件成立取 1，否则取 0。于是

$$p(y_i | x_i, w) = \mu_i^{\mathbf{I}(y_i=1)} (1 - \mu_i)^{\mathbf{I}(y_i=0)}.$$

- 独立样本的似然是所有样本概率的乘积；取对数变成和；再取负号得到 NLL（越小越好）。这就得到 (8.2)。
- 把指示函数分别展开，就得到更常见的交叉熵写法 (8.3)：

$$-[y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)].$$

它会对“很自信但错了”的预测给出很大的惩罚。

2) 用 $\{-1, +1\}$ 标签时 (式 8.4)

把标签换成 $\tilde{y}_i \in \{-1, +1\}$ 后，有一个紧凑的统一写法：

$$\text{NLL}(w) = \sum_{i=1}^N \log(1 + \exp(-\tilde{y}_i w^\top x_i)).$$

直觉：

- 若样本是正类 ($\tilde{y}_i = +1$)，我们希望 $w^\top x_i$ 大（使 $-\tilde{y}_i w^\top x_i$ 变小），从而损失 $\log(1 + e^{-})$ 变小；
- 若是负类 ($\tilde{y}_i = -1$)，希望 $w^\top x_i$ 小（负且绝对值大），同理让损失变小。
这一式子把两种情况合并成了一个统一的表达。

3) 为什么没有“闭式解”？

线性回归的目标是二次函数，推导一阶导=0 能直接解出 $w = (X^\top X)^{-1} X^\top y$ 。

而逻辑回归的目标里有 log 与 Sigmoid（指数函数），使得“设导数=0”后得到的是非线性方程，解不出简单的代数式，所以要靠数值优化算法（比如梯度下降、牛顿法、拟牛顿 L-BFGS、共轭梯度、IRLS 等）迭代求解。

4) 梯度 (8.5) 从哪来？

先对单个样本的交叉熵

$$\ell_i(w) = -[y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$$

求导。因为 $\mu_i = \sigma(z_i)$, $z_i = w^\top x_i$, 利用链式法则与 $\sigma'(z) = \sigma(z)(1 - \sigma(z))$:

$$\frac{\partial \mu_i}{\partial w} = \mu_i(1 - \mu_i)x_i.$$

带入可得

$$\nabla_w \ell_i = \left(-\frac{y_i}{\mu_i} + \frac{1 - y_i}{1 - \mu_i} \right) \cdot \mu_i(1 - \mu_i)x_i = (\mu_i - y_i)x_i.$$

把所有样本相加，就得到

$$g = \nabla_w \text{NLL}(w) = \sum_i (\mu_i - y_i)x_i = X^\top(\mu - y).$$

直觉：预测 μ_i 大于真实 y_i （偏高）时，梯度沿 $+x_i$ 方向推动参数往回调；反之亦然。

5) 海森矩阵 (8.6)–(8.7) 从哪来？

再对梯度求导：

$$H = \nabla_w^2 \text{NLL}(w) = \sum_i \nabla_w [(\mu_i - y_i)x_i] = \sum_i (\nabla_w \mu_i) x_i^\top = \sum_i \mu_i(1 - \mu_i) x_i x_i^\top.$$

把所有项堆在一起，得到矩阵式

$$H = X^\top S X, \quad S = \text{diag}(\mu_i(1 - \mu_i)).$$

这里 S 是 $N \times N$ 的对角矩阵；每个对角元素都在 0 与 0.25 之间（因为 $0 < \mu_i < 1$ 且 $\mu_i(1 - \mu_i) \leq 1/4$ ）。

6) 为什么 H 正定、NLL 凸、极小值唯一？

- 因为 S 的对角元都**严格为正** ($\mu_i \in (0, 1) \Rightarrow \mu_i(1 - \mu_i) > 0$)，于是对任意非零向量 v ：

$$v^\top H v = v^\top X^\top S X v = (Xv)^\top S (Xv) > 0,$$

(在常见的“特征不退化”(列满秩)假设下) 说明 H 正定。

- H 正定 \Rightarrow 目标函数是**严格凸的**；凸函数只有一个全局最小点，所以解是**唯一的**。
直观图像：NLL 像一个“碗”，从任何地方出发往碗底走，都只会落到同一个最低点。

7) 这给优化带来什么好处？

- 收敛性好**：因为目标是凸的，没有局部极小值的陷阱。
- 二阶信息可用**：有了 $H = X^\top S X$ ，像牛顿法或 IRLS (迭代加权最小二乘) 这样利用二阶信息的算法会非常高效。
- 一阶法也行**：用梯度下降/动量/Adam 等一阶优化器也能稳健下降（但二阶法往往步数更少）。

8) 一个微型数值感受 (单样本)

若某个样本 $y = 1$ ，模型给出 $\mu = 0.99$ ，则损失 $-\log 0.99 \approx 0.010$ (很好)；
若却给出 $\mu = 0.01$ ，损失 $-\log 0.01 \approx 4.605$ (很大惩罚)。

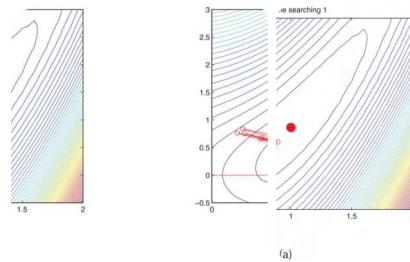
这就是交叉熵“错得越自信，罚得越重”的特性，能推动模型尽快把“极错的自信预测”纠正回来。

9) 矩阵形状 (便于对照)

- $X: N \times D; w: D \times 1; \mu, y: N \times 1.$
- $g = X^\top(\mu - y): D \times 1.$
- $S: N \times N$ 对角; $H = X^\top S X: D \times D.$

小结

- 逻辑回归用交叉熵作为训练目标，其 NLL 是凸的；
- 梯度 $X^\top(\mu - y)$ 与海森 $X^\top S X$ 都有简洁形式；
- 没有闭式解，需数值优化，但因为凸性与二阶结构，求解通常稳定且高效。



(a)

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.2 Gradient descent on a simple function, starting from $(0, 0)$, for 20 steps, using a fixed learning rate (step size) η . The global minimum is at $(1, 1)$. (a) $\eta = 0.1$. (b) $\eta = 0.6$. Figure generated by `steepestDescentDemo`.

Figure 8.3 (a) Steepest descent on the same function as Figure 8.2, starting from $(0, 0)$, using line search. Figure generated by `steepestDescentDemo`. (b) Illustration of the fact that at the end of a line search (top of picture), the local gradient of the function will be perpendicular to the search direction. Based on Figure 10.6.1 of (Press et al. 1988).

8.3.2 Steepest descent

Perhaps the simplest algorithm for unconstrained optimization is gradient descent, also known as steepest descent. This can be written as follows:

$$\theta_{k+1} = \theta_k - \eta g_k \quad (8.8)$$

where η is the step size or learning rate. The main issue in gradient descent is: how should we set the step size? This turns out to be quite tricky. If we use a constant learning rate, but make it too small, convergence will be very slow, but if we make it too large, the method can fail to converge at all. This is illustrated in Figure 8.2. where we plot the following (convex) function

$$f(\theta) = 0.5(\theta_2 - 1)^2 + 0.5(\theta_1 - 1)^2, \quad (8.9)$$

We arbitrarily decide to start from $(0, 0)$. In Figure 8.2(a), we use a fixed step size of $\eta = 0.1$; we see that it moves slowly along the valley. In Figure 8.2(b), we use a fixed step size of $\eta = 0.6$; we see that the algorithm starts oscillating up and down the sides of the valley and never converges to the optimum.

Let us develop a more stable method for picking the step size, so that the method is guaranteed to converge to a local optimum no matter where we start. (This property is called global convergence, which should not be confused with convergence to the global optimum!) By Taylor's theorem, we have

$$f(\theta + \eta d) \approx f(\theta) + \eta g^T d \quad (8.10)$$

where d is our descent direction. So if η is chosen small enough, then $f(\theta + \eta d) < f(\theta)$, since the gradient will be negative. But we don't want to choose the step size η too small, or we will move very slowly and may not reach the minimum. So let us pick η to minimize

$$\varphi(\eta) = f(\theta_k + \eta d_k) \quad (8.11)$$

This is called line minimization or line search. There are various methods for solving this 1d optimization problem; see (Nocedal and Wright 2006) for details. Figure 8.3(a) demonstrates that line search does indeed work for our simple problem. However, we see that the steepest descent path with exact line searches exhibits a characteristic zig-zag behavior. To see why, note that an exact line search satisfies $\eta_k = \arg\min_{\eta > 0} \varphi(\eta)$. A necessary condition for the optimum is $\varphi'(\eta) = 0$. By the chain rule, $\varphi'(\eta) = d^T g$, where $g = f'(\theta_k + \eta d_k)$ is the gradient at the end of the step. So we either have $g = 0$, which means we have found a stationary point, or $g \perp d$, which

means that exact search stops at a point where the local gradient is perpendicular to the search direction. Hence consecutive directions will be orthogonal (see Figure 8.3(b)). This explains the zig-zag behavior. One simple heuristic to reduce the effect of zig-zagging is to add a momentum term, $(\theta_k - \theta_{k-1})$, as follows:

$$\theta_{k+1} = \theta_k - \eta_k g_k + \mu_k (\theta_k - \theta_{k-1}) \quad (8.12)$$

where $0 \leq \mu_k \leq 1$ controls the importance of the momentum term. In the optimization community, this is known as the heavy ball method (see e.g., (Bertsekas 1999)). An alternative way to minimize "zig-zagging" is to use the method of conjugate gradients (see e.g., (Nocedal and Wright 2006, ch 5) or (Golub and van Loan 1996, Sec 10.2)). This is the method of choice for quadratic objectives of the form $f(\theta) = \frac{1}{2} \theta^T A \theta + b^T \theta$, which arise when solving linear systems. However, non-linear CG is less popular.

已思考 37s

下面先给**完整逐字翻译**，再做**通俗解释**（把每个概念都说清楚）。我会把式子排版得便于阅读。

译文

图 8.2 对一个简单函数做梯度下降，从 $(0, 0)$ 出发，固定学习率（步长） η ，共 20 步。全局最小值在 $(1, 1)$ 处。**(a)** $\eta = 0.1$ ；**(b)** $\eta = 0.6$ 。本图由 `steepestDescentDemo` 生成。

图 8.3 (a) 在与图 8.2 相同的函数上做最速下降，从 $(0, 0)$ 出发，使用**线搜索**。本图由 `steepestDescentDemo` 生成。**(b)** 说明了这样一个事实：在一次线搜索结束时（图中上方），函数在该点的**局部梯度与搜索方向是垂直的**。基于 (Press 等, 1988) 的图 10.6.1。

8.3.2 最速下降 (Steepest descent)

或许最简单的无约束优化算法就是**梯度下降**，又称**最速下降**。它可以写成：

$$\theta_{k+1} = \theta_k - \eta_k g_k \quad (8.8)$$

其中 η_k 是步长或学习率。梯度下降的关键问题在于：**步长该怎么取？** 这其实很棘手：若用常数学习率但设得太小，收敛会非常慢；若设得太大，方法可能根本不收敛。这在图 8.2 中有所展示，我们绘制的是下面这个（凸）函数：

$$f(\theta) = \frac{1}{2} (\theta_1^2 - \theta_2)^2 + \frac{1}{2} (\theta_1 - 1)^2 \quad (8.9)$$

我们随意选择从 $(0, 0)$ 出发。在图 8.2(a) 中，我们使用固定步长 $\eta = 0.1$ ；可以看到它沿着“山谷”缓慢移动。在图 8.2(b) 中，我们使用固定步长 $\eta = 0.6$ ；可以看到算法开始在山谷两侧上下来回振荡，从未收敛到最优点。

下面我们来设计一种**更稳定**的取步长方法，使得**无论从哪里开始该方法都保证收敛到某个局部最优**。（这叫做**全局收敛性**，不要与“收敛到全局最优解”混淆！）由泰勒定理，

$$f(\theta + \eta d) \approx f(\theta) + \eta g^\top d \quad (8.10)$$

其中 d 是我们的下降方向。因此如果 η 取得足够小，就有 $f(\theta + \eta d) < f(\theta)$ ，因为沿下降方向时 $g^\top d < 0$ 。但是我们又不希望 η 太小，否则移动很慢，可能到不了极小处。所以让我们把 η 选为下面的一维最小化问题的解：

$$\phi(\eta) = f(\theta_k + \eta d_k) \quad (8.11)$$

这叫**线最小化或线搜索**。求解这个一维优化问题的方法有很多；细节见 Nocedal & Wright (2006)。图 8.3(a) 说明，在我们的简单问题上，线搜索确实有效。

不过我们也看到：带**精确线搜索**的最速下降，其路径会出现**典型的“之”字形**。原因如下：
精确线搜索满足

$$\eta_k = \arg \min_{\eta > 0} \phi(\eta).$$

最优的必要条件是 $\phi'(\eta) = 0$ 。由链式法则，

$$\phi'(\eta) = d^\top g,$$

其中 $g = \nabla f(\theta + \eta d)$ 是**走完这一步后的梯度**。于是要么 $g = 0$ （表示已经到达驻点），要么 $g \perp d$ （表示精确线搜索会停在一个**局部梯度与搜索方向互相垂直的点**）。因此**连续两步的方向往往互相正交**（见图 8.3(b)），这就解释了“之”字形。

一个降低“之”字效应的简单启发式做法，是加入**动量项** $(\theta_k - \theta_{k-1})$ ，如下：

$$\theta_{k+1} = \theta_k - \eta_k g_k + \mu_k (\theta_k - \theta_{k-1}) \quad (8.12)$$

其中 $0 \leq \mu_k \leq 1$ 控制动量的重要程度。在优化界，这叫**重球法** (heavy ball method)（见如 Bertsekas, 1999）。另一种减少“之”字的方法是用**共轭梯度法**（参见 Nocedal & Wright 2006 第 5 章，或 Golub & Van Loan 1996 第 10.2 节）。它是求解**二次型目标** $f(\theta) = \frac{1}{2} \theta^\top A \theta - b^\top \theta + c$ （等价于解线性方程组）时的首选。不过，**非线性**共轭梯度在实践中较少使用。

通俗解释（一步步讲清楚）

1) 基本符号

- θ : 要优化的“位置/参数”（二维例子里是 (θ_1, θ_2) ）。
- $g_k = \nabla f(\theta_k)$: 当前位置的梯度（指向“上坡”最陡方向）。
- η_k : 步长/学习率，控制这一步走多远。
- d_k : 搜索方向。最速下降里 $d_k = -g_k$ （向最陡下坡处走）。

2) 梯度下降更新式 (8.8)

每一步都做：

当前位置 θ_k 计算梯度 g_k ；

朝着**负梯度**方向走一小步 η_k : $\theta_{k+1} = \theta_k - \eta_k g_k$ 。

3) 为什么步长很关键？（看图 8.2）

- 这个函数的等高线像一条“细长的山谷”。
- **小步长**（如 $\eta = 0.1$ ）：每次只走一点点，沿谷底缓慢爬行→慢。
- **大步长**（如 $\eta = 0.6$ ）：一下子跨过谷底冲到另一侧，再被梯度拉回去，又冲过去……→左右摇摆、发散/不收敛。

直观比喻：你在陡峭的峡谷里走路，步子太小走得慢；步子太大就会在两壁之间来回撞。

4) 线搜索为何更稳？（式 8.10–8.11）

- 泰勒近似：沿方向 d 走 η 的代价近似是 $f(\theta) + \eta g^\top d$ 。若 $g^\top d < 0$ ，取**足够小**的 η 就能下降。
- 但“足够小”很慢，于是我们**不预先固定** η ，而是对 $\phi(\eta) = f(\theta + \eta d)$ 做一维最小化（线搜索），每次都选**能让该条直线上最小的** η （或满足某些下降准则的 η ）。这通常能兼顾“能降下去”和“别太慢”。

5) 为什么会“之”字形？（精确线搜索的几何）

- 精确线搜索的最优条件： $\phi'(\eta) = 0 \Rightarrow d^\top g(\theta + \eta d) = 0$ 。
- 也就是：**走到这一步的终点时，新的梯度与刚才的搜索方向相互垂直**。
- 下一步的方向又取负梯度，于是与上一步方向**正交**，在细长山谷里就会**左右折返**，画出“之”字。

6) 如何降低“之”字？

- **动量（重球法）**：在更新里加上“上一段位移”($\theta_k - \theta_{k-1}$)，相当于给“球”一点惯性，减少来回折返， μ 越大惯性越强（但太大也会不稳）。

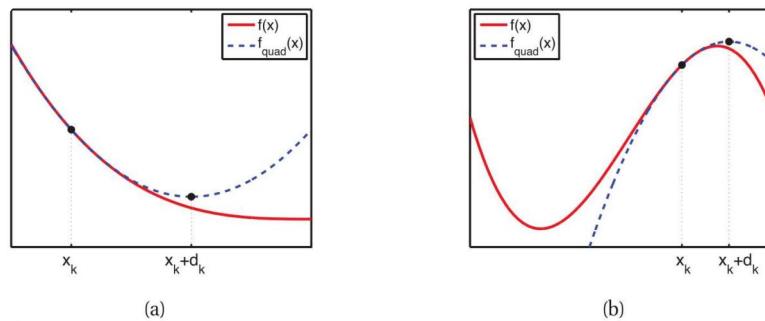
- **共轭梯度 (CG)** : 专治二次函数的“之”字问题, 对 $f(\theta) = \frac{1}{2}\theta^\top A\theta - b^\top \theta + c$ 之类目标特别高效 (求解线性方程组的经典利器)。对一般非线性目标也有“非线性 CG”, 但使用得没那么广。

7) “全局收敛”别混淆

- 这里“全局收敛”指：从任意初始点出发，算法都能保证收敛到某个局部极小值（比如满足某些充分下降条件的线搜索法）。
 - 它不意味着一定能找到全局最小值（整个函数的最低点）。

小結

- 梯度下降=沿负梯度方向更新；**步长**过小慢、过大可能不收敛。
 - **线搜索**：每步都在当前直线方向上一维最小化，稳定下降，但会出现**之字**路径，因为“终点梯度 \perp 搜索方向”。
 - **动量与共轭梯度**可缓解之字：动量给惯性，CG 对二次目标尤其高效。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.4 Illustration of Newton's method for minimizing a 1d function. (a) The solid curve is the function $f(x)$. The dotted line $f_{\text{quad}}(x)$ is its second order approximation at x_k . The Newton step d_k is what must be added to x_k to get to the minimum of $f_{\text{quad}}(x)$. Based on Figure 13.4 of (Vandenberghe 2006). Figure generated by newtonsMethodMinQuad. (b) Illustration of Newton's method applied to a nonconvex function. We fit a quadratic around the current point x_k and move to its stationary point, $x_{k+1} = x_k + d_k$. Unfortunately, this is a local maximum, not minimum. This means we need to be

careful about the extent of our quadratic approximation.

Based on Figure 13.11 of (Vandenberghe 2006). Figure generated by newtonsMethodNonConvex.

8.3.3 Newton's method

One can derive faster optimization methods by taking the curvature of the space (i.e., the Hessian) into account.

These are called second order optimization methods. The primary example is Newton's algorithm. This is an iterative algorithm which consists of updates of the form

$$\theta_{k+1} = \theta_k - \eta_k H^{-1} k g_k \quad (8.13)$$

The full pseudo-code is given in Algorithm 2. This algorithm can be derived as follows. Consider making a second-order Taylor series approximation of $f(\theta)$ around θ_k :

$$f_{\text{quad}}(\theta) = f_k + g_k^T (\theta - \theta_k) + \frac{1}{2} (\theta - \theta_k)^T H_k (\theta - \theta_k) \quad (8.14)$$

Let us rewrite this as

$$f_{\text{quad}}(\theta) = \theta^T A \theta + b^T \theta + c \quad (8.15)$$

where

$$A = \frac{1}{2} H_k, b = g_k - H_k \theta_k, c = f_k - g_k^T \theta_k + \frac{1}{2} \theta_k^T H_k \theta_k \quad (8.16)$$

The minimum of f_{quad} is at

$$\theta = -\frac{1}{2} A^{-1} b = \theta_k - H_k^{-1} k g_k \quad (8.17)$$

Thus the Newton step $d_k = -H_k^{-1} k g_k$ is what should be added to θ_k to minimize the second order approximation of f around θ_k . See Figure 8.4(a) for an illustration.

In its simplest form (as listed), Newton's method requires that H_k be positive definite, which will hold if the function is strictly convex. If not, the objective function is not convex, then H_k may not be positive definite, so $d_k = -H_k^{-1} k g_k$ may not be a descent direction (see Figure 8.4(b) for an example). In this case, one simple strategy is to revert to steepest descent, $d_k = -g_k$. The Levenberg Marquardt algorithm is an adaptive way to blend between Newton steps and steepest descent steps. This method is widely used when solving nonlinear least squares problems. An alternative approach is this: Rather than computing $d_k = -H_k^{-1} k g_k$ directly, we can solve the linear system of equations $H_k d_k = -g_k$ for d_k using conjugate gradient (CG). If H_k is not positive definite, we can simply truncate the CG iterations as soon as negative curvature is detected; this is called truncated Newton.

已思考 19s

下面先给**完整逐字翻译**, 再做**通俗解释** (把每个式子、名词都讲清楚)。

译文

图 8.4 用来说明牛顿法在最小化一维函数时的做法。(a) 实线是函数 $f(x)$ 。虚线 $f_{\text{quad}}(x)$ 是它在 x_k 处的二阶近似。牛顿步 d_k 是必须加到 x_k 上、才能到达 $f_{\text{quad}}(x)$ 的最小值的位置。基于 (Vandenberghe 2006) 的图 13.4。图由 `newtonsMethodMinQuad` 生成。(b) 说明把牛顿法用于**非凸**函数会发生什么。我们在当前点 x_k 周围拟合一个二次函数，并移动到它的驻点 $x_{k+1} = x_k + d_k$ 。不幸的是，这里是一个**局部最大**而不是最小。这意味着我们需要谨慎看待二次近似的适用范围。基于 (Vandenberghe 2006) 的图 13.11。图由 `newtonsMethodNonConvex` 生成。

8.3.3 牛顿法 (Newton's method)

通过把“空间的曲率”（即海森矩阵）考虑进去，可以得到更快的优化方法。这类方法称为**二阶优化方法**。最主要的例子就是**牛顿算法**。它是一个迭代算法，其更新形式为

$$\theta_{k+1} = \theta_k - \eta_k H_k^{-1} g_k \quad (8.13)$$

完整的伪代码见算法 2。这个算法可以这样推导。考虑把 $f(\theta)$ 在 θ_k 附近做二阶泰勒展开的近似：

$$f_{\text{quad}}(\theta) = f_k + g_k^\top (\theta - \theta_k) + \frac{1}{2} (\theta - \theta_k)^\top H_k (\theta - \theta_k) \quad (8.14)$$

把它改写为

$$f_{\text{quad}}(\theta) = \theta^\top A \theta + b^\top \theta + c \quad (8.15)$$

其中

$$A = \frac{1}{2} H_k, \quad b = g_k - H_k \theta_k, \quad c = f_k - g_k^\top \theta_k + \frac{1}{2} \theta_k^\top H_k \theta_k. \quad (8.16)$$

f_{quad} 的最小值在

$$\theta = -\frac{1}{2} A^{-1} b = \theta_k - H_k^{-1} g_k \quad (8.17)$$

因此，牛顿步 $d_k = -H_k^{-1} g_k$ 就是应该加到 θ_k 上、以便最小化 f 在 θ_k 处的二阶近似的那个位移。见图 8.4(a) 的示意。

在它最简单（如上所列）的形式中，牛顿法要求 H_k 为**正定**，这在函数严格凸时成立。若目标函数**不是凸的**，那么 H_k 可能不是正定的，于是 $d_k = -H_k^{-1}g_k$ 可能并不是下降方向（一个例子见图 8.4(b)）。在这种情况下，一个简单策略是退回到最速下降： $d_k = -g_k$ 。**Levenberg–Marquardt 算法**是一种自适应地在牛顿步和最速下降步之间**混合**的方法；它在求解**非线性最小二乘**问题时被广泛使用。另一种做法是：与其直接计算 $d_k = -H_k^{-1}g_k$ ，不如用**共轭梯度法 (CG)** 求解线性方程组 $H_k d_k = -g_k$ 来得到 d_k 。如果 H_k 不是正定的，我们可以**在一旦检测到负曲率时就截断 CG 迭代**；这称为**截断牛顿 (truncated Newton)**。

通俗解释（一步步讲清楚）

1) 一堆符号到底是谁？

- θ : 要优化的参数向量（位置）。一维时就写成 x 。
- $g_k = \nabla f(\theta_k)$: 在 θ_k 处的**梯度**（指向最陡上升）。
- $H_k = \nabla^2 f(\theta_k)$: 在 θ_k 处的**海森矩阵**（二阶导，描述“曲率/弯曲程度”）。
- η_k : 步长/阻尼系数（常取 1，再配合线搜索或信任域使其安全）。
- d_k : 这一步要走的方向与幅度，牛顿法里 $d_k = -H_k^{-1}g_k$ 。

2) 牛顿法的核心直觉：用“局部二次模型”替代原函数

在当前点 θ_k 附近，用泰勒展开把 f 近似成一个**二次函数** f_{quad} （就像把真实地形在脚下用抛物面近似）。

- 这个二次模型在 θ_k 处与原函数有同样的**函数值、梯度、曲率**。
- 对二次函数，**最小值可以一次性算出来**：求解 $\nabla f_{\text{quad}}(\theta) = 0$ 就是式 (8.17)，得到新的位置。
- 于是“新位置-旧位置”的位移就是**牛顿步** $d_k = -H_k^{-1}g_k$ 。

一维的更直观：

$$d_k = -\frac{f'(x_k)}{f''(x_k)}, \text{ 所以}$$

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

斜率 $f'(x_k)$ 越大、曲率 $f''(x_k)$ 越大，步子就相应调整。

3) 为什么牛顿法常常很快？

- 如果目标函数本身就是**二次的**（或“在极小点附近近似二次”），那么牛顿法一步就能找到极小点（或**二次收敛**：误差平方级地缩小）。

- 相比只看一阶信息的梯度下降，牛顿法利用了**二阶曲率**，能自动把不同方向的伸缩比例调好（类似“自动调学习率”），减少“之字形”。

4) 什么时候会出问题？（非凸、H 不正定）

- 正定**含义：对任意非零向量 v , $v^\top H v > 0$ 。这等价于“此处像一个碗口向上”的形状→**局部凸**。
- 若 H_k **正定**，则 $g_k^\top d_k = -g_k^\top H_k^{-1} g_k < 0$ ，说明 d_k 一定是**下降方向**（走这步 f 会变小）。
- 但在**非凸区域**， H_k 可能**不正定**（有方向像“马鞍/向下开口”），这时按 $-H_k^{-1} g$ 走，可能把你带到**局部最大或鞍点**（图 8.4(b) 就是：拟合的二次模型在这些驻点是**最大点**）。

5) 如何保安全、提稳健？

常见做法有三类（文本里都提到了）：

1. 退回最速下降

一旦发现 H_k 不是正定（或线搜索/信任域检测到不安全），就用 $d_k = -g_k$ 这一步，保证是下降方向。

2. Levenberg–Marquardt (LM)

它把牛顿与梯度下降**融合**：用 $(H_k + \lambda I)^{-1} g_k$ 之类的形式 ($\lambda > 0$)，等价于给二次模型“加厚”一个正的曲率，确保“像个碗”，从而方向安全。 λ 会自适应：下降得好就减小，下降不好就增大。LM 在**非线性最小二乘**里最常用。

3. (截断) 牛顿–CG

不显式求 H^{-1} ，而是解线性方程 $H_k d_k = -g_k$ 。

- 用**共轭梯度** (CG) 来解，避免存/求逆 H 。
- 一旦在 CG 过程中检测到**负曲率**（说明不是正定），就**截断**迭代，返回当前方向（或切换策略）。这叫**截断牛顿**，既利用二阶信息，又更稳健、可伸缩到高维。

现实里，牛顿步通常还会配合**线搜索或信任域**：

- 线搜索：保持方向 d_k 不变，沿这条线找合适的步长 $\eta_k \leq 1$ ，确保函数值下降；
- 信任域：只在“我们信得过的小邻域”内信任二次模型，超出就缩小域或调整模型。

6) 图 8.4 怎么读？

- (a) 红色实线是真实 $f(x)$ ，蓝色虚线是 x_k 处的二次近似 $f_{\text{quad}}(x)$ 。把 x_k 往虚线的**最小点**移动一个距离 d_k ，这一步就是牛顿步。
- (b) 若 f 非凸，某些位置附近的二次近似“像个山顶”，它的驻点是**最大点**。照着牛顿步走过去会“上山顶”，不是我们要的“下山谷”。所以要用上面那些稳健化手段。

小结

- 牛顿法: $\theta_{k+1} = \theta_k - \eta_k H_k^{-1} g_k$, 相当于在当前点用**二次函数**近似原函数, 并直接**最小化**这个近似。
- 若 H_k 正定 (函数局部凸), 牛顿步是下降方向, 通常**收敛很快**。
- 非凸时要小心: 可退回梯度下降、用 Levenberg–Marquardt 进行**阻尼/正则化**, 或用****截断牛顿** (牛顿–CG) **等**稳健方案**。

翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

8.3.4 Iteratively reweighted least squares (IRLS)

Let us now apply Newton's algorithm to find the MLE for binary logistic regression. The Newton update at iteration $k + 1$ for this model is as follows (using $\eta_k = 1$, since the Hessian is exact):

$$w_{k+1} = w_k - H^{-1}g_k \quad (8.18)$$

$$= w_k + (XTS_kX)^{-1}XT(y - \mu_k) \quad (8.19)$$

$$= (XTS_kX)^{-1} (XTS_kX)w_k + XT(y - \mu_k) \quad (8.20)$$

$$= (XTS_kX)^{-1}XT [S_kXw_k + y - \mu_k] \quad (8.21)$$

$$= (XTS_kX)^{-1}XTS_kz_k \quad (8.22)$$

where we have defined the working response as

$$z_k = Xw_k + S_k^{-1} (y - \mu_k) \quad (8.23)$$

Equation 8.22 is an example of a weighted least squares problem, which is a minimizer of

$$\sum_{i=1}^N z_{ki}^2 (z_{ki} - w^T x_i)^2 \quad (8.24)$$

Since S_k is a diagonal matrix, we can rewrite the targets in component form (for each case $i = 1: N$) as

$$z_{ki} = w^T x_i + y_i - \mu_{ki} \mu_{ki} (1 - \mu_{ki}) \quad (8.25)$$

This algorithm is known as iteratively reweighted least squares or IRLS for short, since at each iteration, we solve a weighted least squares problem, where the weight matrix S_k changes at each iteration.

已思考 42s

下面先给**完整逐字翻译**, 再做**通俗解释**。我会把每个等式怎么来的都讲清楚。

译文

8.3.4 迭代加权最小二乘 (IRLS)

现在让我们把牛顿算法用于二元逻辑回归的 MLE。对该模型，第 $k + 1$ 次迭代的牛顿更新如下（因为海森矩阵是精确的，这里取 $\eta_k = 1$ ）：

$$w_{k+1} = w_k - H^{-1}g_k \quad (8.18)$$

$$= w_k + (X^\top S_k X)^{-1} X^\top (y - \mu_k) \quad (8.19)$$

$$= (X^\top S_k X)^{-1} [(X^\top S_k X)w_k + X^\top (y - \mu_k)] \quad (8.20)$$

$$= (X^\top S_k X)^{-1} X^\top [S_k X w_k + y - \mu_k] \quad (8.21)$$

$$= (X^\top S_k X)^{-1} X^\top S_k z_k \quad (8.22)$$

其中我们把“工作响应”(working response) 定义为

$$z_k = X w_k + S_k^{-1} (y - \mu_k) \quad (8.23)$$

式 (8.22) 是一个加权最小二乘问题的解，它是下面目标的极小化解：

$$\sum_{i=1}^N S_{ki} (z_{ki} - w^\top x_i)^2 \quad (8.24)$$

由于 S_k 是对角矩阵，我们可以把“目标值”按分量写开（对每个样本 $i = 1, \dots, N$ ）为

$$z_{ki} = w_k^\top x_i + \frac{y_i - \mu_{ki}}{\mu_{ki}(1 - \mu_{ki})}. \quad (8.25)$$

该算法称为**迭代加权最小二乘 (IRLS)**，因为在每次迭代里，我们都要解一个加权最小二乘问题，而权重矩阵 S_k 在每次迭代都会改变。

通俗解释 (基础且详尽)

0) 先把符号对齐

- 有 N 个样本，特征 $x_i \in \mathbb{R}^D$ ，标签 $y_i \in \{0, 1\}$ 。
- $X \in \mathbb{R}^{N \times D}$ ：第 i 行是 x_i^\top 。
- 参数（要学的权重） $w \in \mathbb{R}^D$ 。
- $\mu_i = \sigma(w^\top x_i)$, $\sigma(z) = 1/(1 + e^{-z})$ 。向量 $\mu = [\mu_1, \dots, \mu_N]^\top$ 。
- 梯度与海森（见前一节推导）：

$$g = X^\top(\mu - y), \quad H = X^\top S X, \quad S = \text{diag}(\mu_i(1 - \mu_i)).$$

在第 k 次迭代，用当前 w_k 计算得到 μ_k, S_k 。

1) 从牛顿步到式 (8.22) 的“连环等价”

- 牛顿更新: $w_{k+1} = w_k - H^{-1}g_k$ 。
- 代入 $H_k = X^\top S_k X$, $g_k = X^\top(\mu_k - y)$:

$$w_{k+1} = w_k + (X^\top S_k X)^{-1} X^\top (y - \mu_k),$$

得到 (8.19)。

- 把 w_k 写成 $(X^\top S_k X)^{-1}(X^\top S_k X)w_k$, 合并进同一个括号里, 得 (8.20)。
- 把括号里 $X^\top S_k X w_k$ 与 $X^\top(y - \mu_k)$ 的公共因子 X^\top 提出来, 得 (8.21)。
- 定义

$$z_k = Xw_k + S_k^{-1}(y - \mu_k)$$

(注意 $S_k z_k = S_k X w_k + y - \mu_k$) , 代回 (8.21) 就是 (8.22):

$$w_{k+1} = (X^\top S_k X)^{-1} X^\top S_k z_k.$$

2) 这和“加权最小二乘 (WLS) ”的关系

- 解 加权最小二乘问题

$$\min_w \sum_{i=1}^N S_{ki} (z_{ki} - w^\top x_i)^2$$

的一阶条件 (“法方程”) 是

$$X^\top S_k (Xw - z_k) = 0 \implies (X^\top S_k X)w = X^\top S_k z_k.$$

于是

$$w = (X^\top S_k X)^{-1} X^\top S_k z_k,$$

这恰好就是 (8.22)。

- 所以: 一次牛顿更新 = 解一个 WLS, 其“目标值/伪响应”是 z_k , 其“样本权重”是 $S_{ki} = \mu_{ki}(1 - \mu_{ki})$ 。

3) “工作响应” z_k 是什么直觉?

把逻辑回归在当前 w_k 附近线性化、再做一小步, 就得到一个“临时的回归目标”:

$$z_{ki} = w_k^\top x_i + \frac{y_i - \mu_{ki}}{\mu_{ki}(1 - \mu_{ki})}.$$

- 若某点预测偏差 $y_i - \mu_{ki}$ 大，就会把 z_{ki} 往修正方向“推”得更多。
- 分母 $\mu_{ki}(1 - \mu_{ki})$ 最大是 0.25，在边界 ($\mu \approx 0.5$) 附近权重大；当 μ 接近 0 或 1 时，这个值变小，对应该点对本轮更新的可信度/权重变低（因为模型对它已经很“自信”了）。

4) 为什么这叫“迭代加权最小二乘”？

- “迭代”：每一轮都用当前 w_k 重新计算 μ_k, S_k, z_k ，再解一次 WLS 得到 w_{k+1} 。
- “加权”：WLS 里每个样本的权重是 $S_{ki} = \mu_{ki}(1 - \mu_{ki})$ ，**每一轮都变**。
- 重复直到参数（或目标函数）变化很小就停。

5) 一个可操作的小“算法清单”

1. 初始化 w_0 （全零或小随机）。

2. 循环：

- 计算 $\mu_k = \sigma(Xw_k)$ 。
- 计算对角权重 $S_k = \text{diag}(\mu_k \odot (1 - \mu_k))$ 。
- 计算工作响应 $z_k = Xw_k + S_k^{-1}(y - \mu_k)$ 。
- 解线性方程 $(X^\top S_k X)w_{k+1} = X^\top S_k z_k$ （用 Cholesky/QR，不要显式求逆）。
- 若 $\|w_{k+1} - w_k\|$ 很小或 NLL 改善很小，则停止；否则设 $k \leftarrow k + 1$ 继续。

备注：很多实现把 IRLS 视为 Fisher scoring；在逻辑回归（GLM 的典型例子）里，两者等价。

6) 数值与工程提示（简明）

- 直接构造并求解 $(X^\top S_k X)w = X^\top S_k z_k$ 。规模大时可用共轭梯度等迭代解法。
- 若需要 L_2 正则（岭回归），只要把方程改为 $(X^\top S_k X + \lambda I)w = X^\top S_k z_k$ 。
- 虽然上面取 $\eta_k = 1$ ，实际中常配线搜索/阻尼来更稳（但逻辑回归是凸的，通常很稳）。

小结

- 将牛顿法用于逻辑回归，可把每一步更新改写成一次**加权最小二乘**：

$$w_{k+1} = (X^\top S_k X)^{-1} X^\top S_k z_k.$$

- 这里的“权重” $S_{ki} = \mu_{ki}(1 - \mu_{ki})$ 、“工作响应” z_{ki} 如上式。
- 因为每一步都要“重新加权”，所以叫 IRLS。它利用二阶信息，收敛通常比单纯的梯度下降快而稳。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.3.5 Quasi-Newton (variable metric) methods

The mother of all second-order optimization algorithm is Newton's algorithm, which we discussed in Section 8.3.3. Unfortunately, it may be too expensive to compute H explicitly. QuasiNewton methods iteratively build up an approximation to the Hessian using information gleaned from the gradient vector at each step. The most common method is called BFGS (named after its inventors, Broyden, Fletcher, Goldfarb and Shanno), which updates the approximation to the Hessian $B_k \approx H_k$ as follows:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T k y_k} - \frac{(B_k s_k)(B_k s_k)^T}{s_k^T k s_k} \quad (8.26)$$

$$s_k = \theta_k - \theta_{k-1} \quad (8.27)$$

$$y_k = g_k - g_{k-1} \quad (8.28)$$

This is a rank-two update to the matrix, and ensures that the matrix remains positive definite (under certain restrictions on the step size). We typically start with a diagonal approximation, $B_0 = I$. Thus BFGS can be thought of as a "diagonal plus low-rank" approximation to the Hessian.

Alternatively, BFGS can iteratively update an approximation to the inverse Hessian, $C_k \approx H^{-1}_k$, as follows:

$$C_{k+1} = I - \frac{s_k y_k^T}{y_k^T k y_k} C_k \frac{I - y_k s_k^T}{y_k^T k y_k} + \frac{s_k s_k^T}{y_k^T k y_k} \quad (8.29)$$

Since storing the Hessian takes $O(D^2)$ space, for very large problems, one can use limited memory BFGS, or L-BFGS, where H_k or H^{-1}_k is approximated by a diagonal plus low rank matrix. In particular, the product $H^{-1}_k g_k$ can be obtained by performing a sequence of inner products with s_k and y_k , using only the m most recent (s_k, y_k) pairs, and ignoring older information. The storage requirements are therefore $O(mD)$. Typically $m \sim 20$ suffices for good performance. See (Nocedal and Wright 2006, p177) for more information. L-BFGS is often the

method of choice for most unconstrained smooth optimization problems that arise in machine learning.

下面先给**完整逐字翻译**, 然后做**通俗解释**, 把每个符号的含义与直觉都讲清楚。

译文

8.3.5 拟牛顿 (变度量) 方法

所有二阶优化算法的“老祖宗”是第 8.3.3 节讨论过的牛顿算法。不幸的是, 显式计算 H (海森矩阵) 可能代价太高。**拟牛顿方法**通过每一步的**梯度向量**所提供的信息, **迭代地构造**对海森矩阵的近似。最常见的方法叫 **BFGS** (以其发明者 Broyden、Fletcher、Goldfarb 和 Shanno 的姓氏命名), 它对海森的近似 $B_k \approx H_k$ 的更新为

$$B_{k+1} = B_k + \frac{y_k y_k^\top}{y_k^\top s_k} - \frac{(B_k s_k)(B_k s_k)^\top}{s_k^\top B_k s_k} \quad (8.26)$$

$$s_k = \theta_k - \theta_{k-1} \quad (8.27)$$

$$y_k = g_k - g_{k-1} \quad (8.28)$$

这对矩阵来说是一次秩 2 (rank-two) 的更新, 并且 (在对步长作出某些限制时) 能保证矩阵保持正定。我们通常从一个对角近似开始, $B_0 = I$ 。因此可以把 BFGS 看作对海森的“对角 + 低秩”近似。

或者, BFGS 也可以**迭代更新海森的逆矩阵的近似**, $C_k \approx H_k^{-1}$, 如下:

$$C_{k+1} = \left(I - \frac{s_k y_k^\top}{y_k^\top s_k} \right) C_k \left(I - \frac{y_k s_k^\top}{y_k^\top s_k} \right) + \frac{s_k s_k^\top}{y_k^\top s_k} \quad (8.29)$$

由于存储海森需要 $O(D^2)$ 的空间, 对于非常大的问题, 可以使用**有限内存 BFGS (L-BFGS)**: 用“对角 + 低秩矩阵”来近似 H_k 或 H_k^{-1} 。特别地, 乘积 $H_k^{-1} g_k$ 可以通过与 s_k 和 y_k 做一系列内积来得到, 只使用最近的 m 对 (s_k, y_k) , 忽略更早的信息。这样存储需求为 $O(mD)$ 。通常 $m \approx 20$ 就能有很好的效果。更多信息见 Nocedal 与 Wright (2006, p.177)。在机器学习中, 大多数无约束的光滑优化问题, L-BFGS往往是首选方法。

通俗解释 (基础且详尽)

1) 为什么要拟牛顿?

- 牛顿法需要方向 $d_k = -H_k^{-1}g_k$ 。但 H_k 是 $D \times D$ 的矩阵:

- 计算/存储 H_k 要 $O(D^2)$;
- 求解线性方程或求逆也很贵。
- 拟牛顿的想法：不显式计算 H ，而是用我们已经有的量——梯度的变化——逐步“学出”一个近似的曲率。

2) 两对关键量： s_k 与 y_k

- $s_k = \theta_k - \theta_{k-1}$: 本轮从旧点到新点走了多远（步子）。
- $y_k = g_k - g_{k-1}$: 对应地，梯度改变了多少。
- 直觉：在“真正的”二阶信息里，有割线条件（secant condition）：

$$H_k s_k \approx y_k$$

意思是：沿着 s_k 方向，曲率把步子映射成梯度变化。拟牛顿更新就是要找一个新矩阵 B_{k+1} 既接近日的 B_k ，又尽量满足 $B_{k+1}s_k = y_k$ ，同时保持对称正定。

3) BFGS 的更新式 (8.26) 在做什么？

- 它对 B_k 做秩 2 的修正（两个外积项），把刚刚观测到的 (s_k, y_k) 信息“注入”到曲率近似里。
- 在满足曲率条件 $y_k^\top s_k > 0$ 时（实际中用 Wolfe 线搜索来保证），BFGS 能保持 B_{k+1} 对称正定，于是搜索方向

$$d_k = -B_k^{-1}g_k$$

一定是下降方向（安全）。

小提示：很多实现直接维护逆近似 $C_k \approx H_k^{-1}$ （式 8.29），因为我们真正需要的是 $H^{-1}g$ ；这样每步方向就是 $d_k = -C_k g_k$ ，避免每次解线性方程。

4) 为什么“对角 + 低秩”？

- 初始化 $B_0 = I$ （对角），之后每次加上低秩的外积修正。
- 这让我们逐步在最近走过的方向上补充曲率信息，效果像“自适应地学习一个度量（metric）”。

5) L-BFGS：如何把内存降到线性？

- 不存矩阵 B_k 或 C_k 。只保存最近 m 轮的 (s_i, y_i) 。
- 计算方向 $p_k = -H_k^{-1}g_k$ 用一个著名的两遍（two-loop）递推：
 - 先从当前梯度出发，依次用最近的 (s_i, y_i) 做内积做修正；
 - 再正向把修正“还原”。

- 存储与计算量是 $O(mD)$, 通常 $m \approx 10 \sim 20$ 就很好, 因此非常适合**高维优化** (如深度学习早期、线性/逻辑回归的大规模训练、CRF 等)。

6) 和牛顿/一阶法的对比

- 与牛顿:** 拟牛顿**不需要显式 H** , 但能近似“二阶”的好处 (自动调各方向的步幅), 因此通常**远快于**纯梯度下降。
- 与一阶自适应法** (如动量/Adam) : BFGS 类方法有严格的数值优化理论做支撑 (尤其是在光滑凸问题), 且常能以更少步数收敛; 但在非凸、噪声大、批量很小的情形下, 一阶自适应法更稳健一些。

7) 几点工程注意

- 为了保证 $y_k^\top s_k > 0$ (从而保持正定), 实际实现通常配合**线搜索** (如 Wolfe 条件) 或**信任域**。
- 若问题非常大, 也可把 L-BFGS 与**稀疏特征、正则化**结合; L2 正则会在目标二阶里加一个 λI , 对 BFGS/L-BFGS 来说很自然。
- 对**非凸问题**, BFGS 仍广泛好用, 但理论保证较弱; 实践中依赖于良好的初始化与线搜索。

小结

- 拟牛顿**用梯度变化 (s_k, y_k) 逐步逼近曲率, 无需显式海森。
- BFGS更新 (8.26) /逆更新 (8.29) 保持 (在适当条件下) **正定**, 方向安全、收敛快。
- L-BFGS只保留最近 m 对 (s, y), 内存 $O(mD)$ 、计算高效, 是机器学习里**无约束光滑优化**的常用首选。

翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

8.3.6 ℓ_2 regularization

Just as we prefer ridge regression to linear regression, so we should prefer MAP estimation for logistic regression to computing the MLE. In fact, regularization is important in the classification setting even if we have lots of data. To see why, suppose the data is linearly separable. In this case, the MLE is obtained when $\|w\| \rightarrow \infty$, corresponding

to an infinitely steep sigmoid function, $I(w^T x > w_0)$, also known as a linear threshold unit. This assigns the maximal amount of probability mass to the training data. However, such a solution is very brittle and will not generalize well. To prevent this, we can use ℓ_2 regularization, just as we did with ridge regression. We note that the new objective, gradient and Hessian have the following forms:

$$f(w) = \text{NLL}(w) + \lambda w^T w \quad (8.30)$$

$$g(w) = g(w) + \lambda w \quad (8.31)$$

$$H(w) = H(w) + \lambda I \quad (8.32)$$

It is a simple matter to pass these modified equations into any gradient-based optimizer.

已思考 28s

下面先给**完整逐字翻译**，再做**通俗解释**。我会把每个符号都说清楚，并给直觉与实操要点。

译文

8.3.6 ℓ_2 正则化

就像我们更偏好**岭回归**而不是普通线性回归一样，在逻辑回归里也应当更偏好**MAP 估计**而不是只求 MLE。事实上，即使我们有大量数据，在分类问题中**正则化**依然很重要。为什么？假设数据是**线性可分的**。在这种情况下，当 $\|w\| \rightarrow \infty$ 时可以得到 MLE；这对应于一个**无限陡峭**的 sigmoid 函数，极限是指示函数 $I(w^T x > w_0)$ ，也叫做**线性阈值单元**。这样的模型把尽可能多的概率质量分配给训练数据。然而，这样的解非常**脆弱**，**泛化能力**很差。为防止这种情况，我们可以像做岭回归那样使用 ℓ_2 正则化。新的目标函数、梯度和海森矩阵的形式如下：

$$f_\lambda(w) = \text{NLL}(w) + \lambda w^T w \quad (8.30)$$

$$g_\lambda(w) = g(w) + \lambda w \quad (8.31)$$

$$H_\lambda(w) = H(w) + \lambda I \quad (8.32)$$

把这些改过的公式交给任何基于梯度的优化器都很容易。

解释（基础且详尽）

1) 这里在说什么？

- MLE (最大似然) 只关注“把训练集的似然做大”。
- MAP (最大后验) = 最大化 $\log p(\text{数据} | w) + \log p(w)$ 。令 $p(w)$ 为零均值高斯 $N(0, \sigma^2 I)$ 时,

$$-\log p(w) = \frac{1}{2\sigma^2} \|w\|^2$$

就得到式 (8.30) 的 $\lambda \|w\|^2$ (其中 $\lambda = 1/(2\sigma^2)$)。

所以** ℓ_2 正则化 \Leftrightarrow 给参数加高斯先验**; “偏好权重更小的模型”。

2) 为什么线性可分会导致 $\|w\| \rightarrow \infty$?

- 对一个完全可分的数据集, 总能找到某个方向 w 使得 $w^\top x_i$ 对正类全是正的、对负类全是负的。
- 让 $w \leftarrow cw$, 当 $c \rightarrow \infty$ 时:
 $\sigma(c w^\top x_i)$ 会对正类趋近 1、对负类趋近 0,
于是 **负对数似然 NLL $\downarrow 0$** , 可以被任意逼近得更小——因此**没有有限的最优解**。
- 这等价于把 S 形函数压成一个**硬阈值** $I(w^\top x > w_0)$ 的分类器。这样的解对数据里一点点噪声/新样本非常敏感, **过拟合严重**。

3) ℓ_2 正则化起到什么作用?

- 在目标里加 $\lambda \|w\|^2$ 像给每个权重系上“弹簧”, 把它们往 0 拉:
 - 防止权重无限变大;
 - 让决策边界不过分“自信陡峭”, 从而**更能泛化**;
 - 数值上让问题更稳定, **海森** + λI 通常**正定**, 优化更顺滑。

4) 式 (8.30)–(8.32) 如何用?

- 记住未正则化时

$$g(w) = X^\top(\mu - y),$$

$$H(w) = X^\top S X \quad (S = \text{diag}(\mu_i(1 - \mu_i))) .$$
- 加上 ℓ_2 后:
 - **目标:** $f_\lambda = \text{NLL} + \lambda \|w\|^2$
 - **梯度:** $g_\lambda = X^\top(\mu - y) + \lambda w$
 - **海森:** $H_\lambda = X^\top S X + \lambda I$
- 这意味着:
 - **梯度下降:** $w \leftarrow w - \eta [X^\top(\mu - y) + \lambda w]$ 。
 - **牛顿/IRLS:** 把法方程改为

$$(X^\top S X + \lambda I)w = X^\top S z$$

 (只是在原来的矩阵上加了 λI)。

- 对任何“基于梯度/海森”的优化器，只需把 λ 项加进去即可。

5) 一些实用的小细节

- 偏置/截距通常不罚：**做法是把 $w = [b; \tilde{w}]$ 分开，只对 \tilde{w} 施加 ℓ_2 ；或在 λI 的相应位置放 0。
- λ 越大，惩罚越强，模型越“平滑/保守”； λ 太大则欠拟合。**用交叉验证选 λ** 是常规做法。
- 特征最好标准化**（零均值、单位方差），这样 ℓ_2 约束对各个方向的效果更均衡。

6) 直觉小图景

- 没有正则时：目标像个“长而平的谷底”，可沿着把 $\|w\|$ 放大的方向一直下降，解跑到无穷远；
- 加 $\lambda\|w\|^2$ 后：谷底被抬起并向原点“拉紧”，出现**唯一的有限最小点**；优化好解、泛化也更好。

小结

- 逻辑回归在可分数据上，MLE 会把 $\|w\|$ 推向无穷，得到像硬阈值的、但很脆弱的解。
- 加 ℓ_2 正则（等价于高斯先验的 MAP）能**阻止权重爆炸、提升泛化**，且只需把 λ 项加入**目标/梯度/海森**：

$$f_\lambda = \text{NLL} + \lambda\|w\|^2, \quad g_\lambda = g + \lambda w, \quad H_\lambda = H + \lambda I.$$

- 实作时常不惩罚偏置，用交叉验证挑 λ ，并配合特征标准化。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.3.7 Multi-class logistic regression

Now we consider multinomial logistic regression, sometimes called a maximum entropy classifier. This is a model of the form

$$p(y = c|x, W) = \frac{\exp(w^T c^T x)}{\sum_{c=1}^C \exp(w^T c^T x)} \quad (8.33)$$

A slight variant, known as a conditional logit model, normalizes over a different set of classes for each data case; this can be useful for modeling choices that users

make between different sets of items that are offered to them. Let us now introduce some notation. Let $\mu_{ic} = p(y_i = c | x_i, W) = S(\eta_i)c$, where $\eta_i = Wx_i$ is a $C \times 1$ vector. Also, let $y_{ic} = I(y_i = c)$ be the one-of- C encoding of y_i ; thus y_i is a bit vector, in which the c 'th bit turns on iff $y_i = c$.

Following (Krishnapuram et al. 2005), let us set $w_C = 0$, to ensure identifiability, and define $w = \text{vec}(W(:, 1:C-1))$ to be a $D \times (C-1)$ column vector. With this, the log-likelihood can be written as

$$I(W) = \log \sum_{i=1}^N \sum_{c=1}^C \mu_{ic}^{y_{ic}} = \sum_{i=1}^N \sum_{c=1}^C y_{ic} \log \mu_{ic} \quad (8.34)$$

$$= \sum_{i=1}^N \sum_{c=1}^C y_{ic} w^T c x_i - \log \sum_{c=1}^C \exp(w^T c x_i) \quad (8.35)$$

Define the NLL as

$$f(w) = -I(W) \quad (8.36)$$

We now proceed to compute the gradient and Hessian of this expression. Since w is blockstructured, the notation gets a bit heavy, but the ideas are simple. It helps to define $A \otimes B$ be the kronecker product of matrices A and B . If A is an $m \times n$ matrix and B is a $p \times q$ matrix, then $A \times B$ is the $mp \times nq$ block matrix

$$A \otimes B = [\dots [a_{11}B \dots a_{1n}B \dots \dots \dots a_{m1}B \dots a_{mn}B] \dots] \quad (8.37)$$

Returning to the task at hand, one can show (Exercise 8.4) that the gradient is given by

$$g(W) = \nabla f(w) = \sum_{i=1}^N (\mu_i - y_i) \otimes x_i \quad (8.38)$$

where $y_i = (I(y_i = 1), \dots, I(y_i = C-1))$ and $\mu_i(W) = [p(y_i = 1 | x_i, W), \dots, p(y_i = C-1 | x_i, W)]$ are column vectors of length $C-1$. For example, if we have $D=3$ feature dimensions and $C=3$ classes, this becomes

$$g(W) = \begin{bmatrix} \dots & \dots & \dots \\ \mu_1 - y_1 & \mu_1 - y_1 & \mu_1 - y_1 \\ \mu_2 - y_2 & \mu_2 - y_2 & \mu_2 - y_2 \\ \mu_3 - y_3 & \mu_3 - y_3 & \mu_3 - y_3 \end{bmatrix} \quad (8.39)$$

In other words, for each class c , the derivative for the weights in the c 'th column is

$$\nabla_w f(W) = \begin{bmatrix} \dots & \dots & \dots \\ \mu_{1c} - y_{1c} & \mu_{1c} - y_{1c} & \mu_{1c} - y_{1c} \\ \mu_{2c} - y_{2c} & \mu_{2c} - y_{2c} & \mu_{2c} - y_{2c} \\ \mu_{3c} - y_{3c} & \mu_{3c} - y_{3c} & \mu_{3c} - y_{3c} \end{bmatrix} \quad (8.40)$$

This has the same form as in the binary logistic regression case, namely an error term times x_i . (This turns out to be a general property of distributions in the exponential family, as we will see in Section 9.3.2.)

One can also show (Exercise 8.4) that the Hessian is the following block structured $D(C-1) \times D(C-1)$ matrix:

$$H(W) = \nabla^2 f(w) = \sum_{i=1}^N (\text{diag}(\mu_i) - \mu_i \mu_i^T) \otimes (x_i x_i^T) \quad (8.41)$$

For example, if we have 3 features and 3 classes, this becomes

$$H(W) = \sum_i (\mu_{i1} - \mu_{i2})^2 + \sum_{i=1}^3 \sum_{j=1}^3 \mu_{ij} \mu_{ij}^T - \sum_{i=1}^3 \sum_{j=1}^3 \mu_{ij} \mu_{ij}^T \quad (8.42)$$

$$= \sum_i (\mu_{i1} - \mu_{i2}) X_i - \sum_{i=1}^3 \sum_{j=1}^3 \mu_{ij} X_i \quad (8.43)$$

where $X_i = x_i x_i^T$. In other words, the block c, c submatrix is given by

$$H_{c,c}(W) = \sum_i \mu_{ic} (\delta_{cc} - \mu_{ic}) x_i x_i^T \quad (8.44)$$

This is also a positive definite matrix, so there is a unique MLE. Now consider minimizing

$$f(W) = -\log p(D|w) - \log p(W) \quad (8.45)$$

where $p(W) = \det V^{-1/2}$. The new objective, its gradient and Hessian are given by

$$f(W) = f(W) + \frac{1}{2} w^T V^{-1} w \quad (8.46)$$

$$g(W) = g(W) + V^{-1} w \quad (8.47)$$

$$H(W) = H(W) + I_C \otimes V^{-1} \quad (8.48)$$

where $p(W) = \det V^{-1/2}$. The new objective, its gradient and Hessian are given by $f(W) = f(W) + \frac{1}{2} w^T V^{-1} w$ $g(W) = g(W) + V^{-1} w$ $H(W) = H(W) + I_C \otimes V^{-1}$

已思考 48s

下面先给**完整逐字翻译**（不遗漏），然后做**通俗解释**（把每个符号和结论都讲清楚；有一处原文疑似排版/笔误，我会在解释里注明更常用的正确形式）。

译文

8.3.7 多分类逻辑回归

现在我们来考虑**多项式** (multinomial) 逻辑回归，有时也称为**最大熵分类器**。其模型形式为

$$p(y=c | x, W) = \frac{\exp(w_c^T x)}{\sum_{c'=1}^C \exp(w_{c'}^T x)} \quad (8.33)$$

一个稍微不同的变体称为**条件 logit 模型**，它对每个数据样本在**不同的类别集合**上进行归一化；当我们要建模用户在提供给他们的不同物品集合之间所作的选择时，这个模型会很有用。

现在引入一些记号。令

$$\mu_{ic} = p(y_i = c | x_i, W) = S(\eta_i)_c,$$

其中 $\eta_i = W^\top x_i$ 是一个 $C \times 1$ 的向量。又令 $y_{ic} = \mathbf{I}(y_i = c)$ 为 y_i 的 one-of- C 编码；因此 y_i 是一个比特向量，且当且仅当 $y_i = c$ 时第 c 位为 1。按照 (Krishnapuram et al., 2005)，令 $w_C = 0$ 以保证可识别性，并定义

$$w = \text{vec}(W(:, 1:C-1))$$

是一个 $D \times (C-1)$ 的列向量。用这些记号，对数似然可写为

$$\ell(W) = \log \prod_{i=1}^N \prod_{c=1}^C \mu_{ic}^{y_{ic}} = \sum_{i=1}^N \sum_{c=1}^C y_{ic} \log \mu_{ic} \quad (8.34)$$

$$= \sum_{i=1}^N \left(\sum_{c=1}^C y_{ic} w_c^\top x_i - \log \sum_{c=1}^C \exp(w_c^\top x_i) \right). \quad (8.35)$$

定义负对数似然为

$$f(w) = -\ell(w). \quad (8.36)$$

接下来我们来计算该式子的梯度和海森矩阵。由于 w 具有分块结构，记号会稍微复杂一点，但思想很简单。我们先定义克罗内克积 $A \otimes B$ 。若 A 是 $m \times n$ 矩阵而 B 是 $p \times q$ 矩阵，则 $A \otimes B$ 是一个 $mp \times nq$ 的分块矩阵：

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}. \quad (8.37)$$

回到我们的任务，可以证明（练习 8.4）梯度为

$$g(W) = \nabla f(w) = \sum_{i=1}^N (\mu_i - y_i) \otimes x_i, \quad (8.38)$$

其中

$$y_i = (\mathbf{I}(y_i = 1), \dots, \mathbf{I}(y_i = C-1)), \quad \mu_i(W) = (p(y_i = 1 | x_i, W), \dots, p(y_i = C-1 | x_i, W))$$

都是长度为 $C-1$ 的列向量。举例来说，若 $D = 3$ 个特征、 $C = 3$ 个类别，则

$$g(W) = \sum_i \begin{pmatrix} (\mu_{i1} - y_{i1})x_{i1} \\ (\mu_{i1} - y_{i1})x_{i2} \\ (\mu_{i1} - y_{i1})x_{i3} \\ (\mu_{i2} - y_{i2})x_{i1} \\ (\mu_{i2} - y_{i2})x_{i2} \\ (\mu_{i2} - y_{i2})x_{i3} \end{pmatrix}. \quad (8.39)$$

换句话说，对每个类别 c ，第 c 列权重的导数为

$$\nabla_{w_c} f(W) = \sum_i (\mu_{ic} - y_{ic}) x_i. \quad (8.40)$$

这与二元逻辑回归的形式相同，即“误差项 $\times x_i$ ”。（事实证明，这对**指数族**分布是一个普遍性质，我们将在第 9.3.2 节看到。）

还可以证明（练习 8.4）**海森矩阵**是下面这个分块的 $D(C-1) \times D(C-1)$ 矩阵：

$$H(W) = \nabla^2 f(w) = \sum_{i=1}^N (\text{diag}(\mu_i) - \mu_i \mu_i^\top) \otimes (x_i x_i^\top). \quad (8.41)$$

例如，当 3 个特征、3 个类别时，有

$$H(W) = \sum_i \begin{bmatrix} \mu_{i1} - \mu_{i1}^2 & -\mu_{i1}\mu_{i2} \\ -\mu_{i1}\mu_{i2} & \mu_{i2} - \mu_{i2}^2 \end{bmatrix} \otimes \begin{bmatrix} x_{i1}x_{i1} & x_{i1}x_{i2} & x_{i1}x_{i3} \\ x_{i2}x_{i1} & x_{i2}x_{i2} & x_{i2}x_{i3} \\ x_{i3}x_{i1} & x_{i3}x_{i2} & x_{i3}x_{i3} \end{bmatrix} \quad (8.42)$$

$$= \sum_i \begin{bmatrix} (\mu_{i1} - \mu_{i1}^2)X_i & -\mu_{i1}\mu_{i2}X_i \\ -\mu_{i1}\mu_{i2}X_i & (\mu_{i2} - \mu_{i2}^2)X_i \end{bmatrix}, \quad X_i = x_i x_i^\top. \quad (8.43)$$

也就是说，第 (c, c') 个分块子矩阵为

$$H_{c,c'}(W) = \sum_i \mu_{ic} (\delta_{c,c'} - \mu_{i,c'}) x_i x_i^\top. \quad (8.44)$$

这同样是一个**正定矩阵**，因此极大似然解是**唯一的**。

现在考虑最小化

$$\tilde{f}(W) = -\log p(D | W) - \log p(W), \quad (8.45)$$

其中 $p(W) = \prod_c N(w_c | 0, V_0)$ 。新的目标、其梯度与海森为

$$\tilde{f}(W) = f(W) + \frac{1}{2} \sum_c w_c^\top V_0^{-1} w_c, \quad (8.46)$$

$$\tilde{g}(W) = g(W) + V_0^{-1} (\sum_c w_c), \quad (8.47)$$

$$\tilde{H}(W) = H(W) + I_C \otimes V_0^{-1}. \quad (8.48)$$

（原文以上三式在段落末尾重复了一次。）

解释 (基础且详尽)

1) 模型与记号一眼明白

- 我们有 C 个类别，特征向量 $x \in \mathbb{R}^D$ 。
- $W = [w_1, \dots, w_C] \in \mathbb{R}^{D \times C}$ ；第 c 列 w_c 是第 c 类的权重。
- Softmax (多类 logistic) :

$$p(y = c \mid x) = \frac{e^{w_c^\top x}}{\sum_{c'} e^{w_{c'}^\top x}}.$$

- 为避免“平移不变”造成的不可识别（所有列同时加同一常量对概率没影响），我们固定 $w_C = 0$ ，只学 $C - 1$ 列。

2) 对数似然 (8.34)(8.35)

- One-hot $y_{ic} \in \{0, 1\}$ 使 $\prod_c \mu_{ic}^{y_{ic}}$ 只留下真实类别的那一项；取对数得 $\sum_c y_{ic} \log \mu_{ic}$ 。
- 再把 μ_{ic} 展开，就得到“线性项 – log 归一化项”（后者叫对数配分函数）。

3) 梯度 (8.38)(8.39)(8.40): 误差 × 特征

- 记 μ_{ic} 为模型对“第 i 个样本属于第 c 类”的概率，而 y_{ic} 为真实标签。
- 对第 c 类的权重：

$$\nabla_{w_c} f = \sum_i (\mu_{ic} - y_{ic}) x_i.$$

这和二分类完全同形：预测–真实 就是误差。

- 把所有类别堆起来，就得到 (8.38) 的克罗内克积写法，便于一次写出整块梯度。

4) 海森 (8.41)–(8.44): 块结构 + 正定

- 对每个样本 i , $\text{diag}(\mu_i) - \mu_i \mu_i^\top$ 是 softmax 的“类概率协方差”；
- 再与 $x_i x_i^\top$ 做克罗内克积，累加得到整体海森。
- 子块形式 (8.44): 当 $c = c'$ 得到正项 $\mu_{ic}(1 - \mu_{ic}) X_i$ ；当 $c \neq c'$ 得到负项 $-\mu_{ic}\mu_{ic'} X_i$ 。
- 该海森正定（在可识别化约束下），所以 NLL 凸、MLE 唯一。这也解释了 softmax 回归好优化、用牛顿/拟牛顿/IRLS 都很稳。

5) 带高斯先验 (L_2 正则) 的负对数后验 (8.45)–(8.48)

- 先验 $p(W) = \prod_c N(w_c \mid 0, V_0)$ 等价于在目标里加 $\frac{1}{2} \sum_c w_c^\top V_0^{-1} w_c$ （即每一列的 L_2 正则）。

- 因此：

- 目标： $\tilde{f} = f + \frac{1}{2} \sum_c w_c^\top V_0^{-1} w_c$ 。
- 更常见的梯度写法是逐列加 $V_0^{-1} w_c$ ：**

$$\tilde{g}(W) = g(W) + (I_{C-1} \otimes V_0^{-1}) w.$$

原文式 (8.47) 写成 $V_0^{-1}(\sum_c w_c)$ 疑似排版简写/笔误；按块向量形式应为“对每个 c 各加 $V_0^{-1} w_c$ ”。

- 海森： $\tilde{H} = H + I_{C-1} \otimes V_0^{-1}$ (每个类别块都加 V_0^{-1})。

若只对 w_c 的非偏置部分正则化，可把对应的行/列在 V_0^{-1} 置 0。

6) 训练要点 (简明)

- 梯度下降/Adam**：用 (8.40) 的梯度 (加上正则项) 即可。
- 牛顿/IRLS/拟牛顿**：用 (8.41) 的海森或其近似 (BFGS / L-BFGS)，或解加权最小二乘的等价形式。
- 数值稳定**：计算 softmax 时用“减最大值”的技巧：

$$\text{softmax}(z)_c = \exp(z_c - \max z) / \sum_c \exp(z_c - \max z).$$

7) 直觉小图

- 二分类用 sigmoid；多分类用 softmax (“多把拉杆”，把概率分给 C 类)。
- 每类的梯度都是“**我以为是这类的概率 – 它是否真是这类**”乘以 x_i ，再对样本求和。
- 海森像是“类别相关的协方差 \times 特征二次型”，因此整体凸、好优化。
- 加高斯先验 = L_2 正则：既防止过拟合，又让海森“更正定”，优化更顺滑。

小结

- 多分类逻辑回归 (softmax/最大熵) 把 $p(y=c | x)$ 设为 $\exp(w_c^\top x)$ 的归一化。
- 对数似然的梯度**： $\nabla_{w_c} f = \sum_i (\mu_{ic} - y_{ic}) x_i$ 。
- 海森**： $\sum_i (\text{diag } \mu_i - \mu_i \mu_i^\top) \otimes (x_i x_i^\top)$ ，正定 \Rightarrow 解唯一。
- 加高斯先验 (L_2)** 后，只需在目标/梯度/海森中加入对应的 V_0^{-1} 项即可。

没看懂这一块在讲什么，和前面有什么区别

明白！这段（8.3.7 多分类逻辑回归）其实是在说：把前面“**二分类的逻辑回归 (sigmoid)** 推广到**多分类 (softmax)**”。下面用最直白的方式梳理它和前面的区别、每个符号在干嘛、以及你需要真正记住的点。

一句话大意

- 前面：**只有两类（0/1），用 sigmoid 输出“是 1 类的概率”。参数是一个向量 w 。
- 这里：**有 C 个类别，用 softmax 一次性输出这 C 个概率（加起来=1）。参数变成一个矩阵 $W = [w_1, \dots, w_C]$ ，每一列对应一个类别的权重。

核心公式对比（看出“换了啥”）

内容	二分类（前面）	多分类（本节）
概率模型	$p(y = 1 x) = \sigma(w^\top x)$	$p(y = c x) = \frac{e^{w_c^\top x}}{\sum_{c'=1}^C e^{w_{c'}^\top x}}$ (softmax)
标签表示	$y \in \{0, 1\}$	one-hot 向量 $y_i = (y_{i1}, \dots, y_{iC})$, 真实类那一位=1, 其余=0
参数	$w \in \mathbb{R}^D$	$W = [w_1, \dots, w_C] \in \mathbb{R}^{D \times C}$ (常设 $w_C = 0$ 消除冗余)
对数似然	$\sum_i [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$	$\sum_i \sum_c y_{ic} \log \mu_{ic}$; 其中 $\mu_{ic} = p(y = c x_i, W)$
梯度	$\nabla_w f = \sum_i (\mu_i - y_i) x_i$	对每个类 c : $\nabla_{w_c} f = \sum_i (\mu_{ic} - y_{ic}) x_i$
海森 (形状)	$X^\top S X$ (标量权重)	分块矩阵: $\sum_i (\text{diag}(\mu_i) - \mu_i \mu_i^\top) \otimes (x_i x_i^\top)$
凸性/唯一性	凸、解唯一	仍然凸（做了可识别化），解唯一
L2 正则	$+\lambda \ w\ ^2$	每列都加: $+\frac{1}{2} \sum_c w_c^\top V_0^{-1} w_c$ (通常就是各列的 L2)

小结：梯度的形状没变——依旧是“预测 – 真实（误差） \times 特征”。只是现在对**每个类别**各有一份。

关键新概念用白话讲

1) 为什么要设 $w_C = 0$ (可识别性) ?

softmax 只看各类的相对分数 $w_c^\top x$ 。如果所有列同时加同一个向量，概率不变 → 参数不唯一。把最后一列固定为 0，相当于把它当“基准类”，就唯一了。

2) 克罗内克积 (\otimes) 是干嘛的?

只是为了把所有类别的梯度/海森一次性打包写得紧凑；理解上你完全可以记住“对每个类别 c 的梯度就是 $\sum_i (\mu_{ic} - y_{ic})x_i$ ”就够了。

3) 为啥海森是“分块”的?

因为现在有 $C-1$ 组参数（不算被固定的那一列）。不同类别之间在 softmax 里是联动的（一个类的概率变大，别的会变小），所以二阶导的矩阵自然出现类与类之间的耦合块。但好消息是：整体仍然正定、问题仍是凸的，优化不难。

4) 和“one-vs-rest (OvR) ”有什么不同?

- 本节的 softmax 回归：一个模型，联合学习所有类别，输出一组和为 1 的概率。
- OvR：训练 C 个二分类器，彼此独立，最后再比较分数。
实践里 softmax 更一致（概率可比、总和=1），通常是默认首选。

5) 和前面真的有什么“本质区别”吗?

只有两点：

- 输出从 1 个概率变成 C 个概率 (softmax)；
- 参数从一个向量变成多列（每类一列），因此梯度/海森按类别分块写。
除此之外，训练套路、正则化思路、凸性结论都和二分类版几乎一样。

一个小例子 (更直观)

假设 $C = 3$ 、 $D = 2$ ，

$$x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad w_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad w_3 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

打分： $w_1^\top x = 2$, $w_2^\top x = 0$, $w_3^\top x = -2$ 。

softmax 概率（近似）：[0.867, 0.117, 0.016] (总和=1)。

若真实类别是 2 类 (one-hot: [0, 1, 0])，则

- 类1的误差： $\mu_1 - y_1 = 0.867 - 0 = 0.867$
- 类2的误差： $0.117 - 1 = -0.883$
- 类3的误差： $0.016 - 0 = 0.016$

各类的梯度贡献都是“误差 × 特征”：

$$\nabla_{w_1} f \propto 0.867 \cdot x, \quad \nabla_{w_2} f \propto (-0.883) \cdot x, \quad \nabla_{w_3} f \propto 0.016 \cdot x.$$

直觉：把分太高的类往下拉，把真实类往上推，别的类小修小补。

训练与正则化

- **优化**：照旧用 GD/Adam 或二阶法（牛顿、拟牛顿、IRLS 的多类版本）。
- **正则化**：给每一列 w_c 加 L2（或高斯先验），就像二分类那样稳定泛化；海森等式里表现为“对角块 + λI ”。

你需要记住的最少要点

1. 二分类：sigmoid；多分类：softmax。
2. 梯度结构不变：(预测 - 真实) × 特征，只是每个类别各一份。
3. 设 $w_C = 0$ 只是为了解除参数冗余。
4. 问题依旧是**凸的、好优化**；正则化照旧能提升泛化。
5. $C = 2$ 时，softmax 退化回 sigmoid（就是前面的二分类）。

如果你愿意，我也可以把多分类的训练过程写成“伪代码版步骤”，或者用你自己的一个小数据当例子，带你手算一两步 😊

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.4 Bayesian logistic regression

It is natural to want to compute the full posterior over the parameters, $p(w|D)$, for logistic regression models. This can be useful for any situation where we want to associate confidence intervals with our predictions (e.g., this is necessary when solving contextual bandit problems, discussed in Section 5.7.3.1). Unfortunately, unlike the linear regression case, this cannot be done exactly, since there is no convenient conjugate prior for logistic regression. We discuss one simple approximation below; some other approaches include MCMC (Section 24.3.3.1), variational inference (Section 21.8.1.1), expectation

propagation (Kuss and Rasmussen 2005), etc. For notational simplicity, we stick to binary logistic regression.

8.4.1 Laplace approximation

In this section, we discuss how to make a Gaussian approximation to a posterior distribution. The approximation works as follows. Suppose $\theta \in \mathbb{R}^D$. Let

$$p(\theta|D) = 1/Z e^{-E(\theta)} \quad (8.49)$$

where $E(\theta)$ is called an energy function, and is equal to the negative log of the unnormalized log posterior, $E(\theta) = -\log p(\theta, D)$, with $Z = p(D)$ being the normalization constant. Performing a Taylor series expansion around the mode θ^* (i.e., the lowest energy state) we get

$$E(\theta) \approx E(\theta^*) + (\theta - \theta^*)^T g + \frac{1}{2} (\theta - \theta^*)^T H (\theta - \theta^*) \quad (8.50)$$

where g is the gradient and H is the Hessian of the energy function evaluated at the mode:

$$g = \nabla E(\theta^*) \quad (8.51)$$

Since θ^* is the mode, the gradient term is zero. Hence

$$\hat{p}(\theta|D) \approx 1/Z e^{-E(\theta^*)} \exp(-\frac{1}{2}(\theta - \theta^*)^T H (\theta - \theta^*)) \quad (8.52)$$

$$= N(\theta|\theta^*, H^{-1}) \quad (8.53)$$

$$Z = p(D) \approx \hat{p}(\theta|D)d\theta = e^{-E(\theta^*)}(2\pi)^D/2|H|^{1/2} \quad (8.54)$$

The last line follows from normalization constant of the multivariate Gaussian. Equation 8.54 is known as the Laplace approximation to the marginal likelihood.

Therefore Equation 8.52 is sometimes called the Laplace approximation to the posterior. However, in the statistics community, the term "Laplace approximation" refers to a more sophisticated method (see e.g. (Rue et al. 2009) for details). It may therefore be better to use the term "Gaussian approximation" to refer to Equation 8.52. A Gaussian approximation is often a reasonable approximation, since posteriors often become more "Gaussian-like" as the sample size increases, for reasons analogous to the central limit theorem. (In physics, there is an analogous technique known as a saddle point approximation.)

已思考 33s

下面先给**完整逐字翻译**（不遗漏），然后做**通俗解释**（把每个式子和结论都讲清楚；我也会把符号的小问题顺手规范一下，意义不变）。

译文

8.4 贝叶斯逻辑回归

对于逻辑回归模型，想要计算参数的完整后验 $p(\theta | D)$ 是很自然的需求。当我们希望给预测结果配上置信区间时，这会很有用（例如在**情境赌博机**问题中就需要这样做，见第 5.7.3.1 节）。不幸的是，不同于线性回归情形，这里**不能精确地做到**，因为逻辑回归没有方便的**共轭先验**。下面我们讨论一种简单近似；其他做法包括 MCMC（见 24.3.3.1 节）、变分推断（见 21.8.1.1 节）、期望传播（Kuss and Rasmussen 2005）等。为简化记号，我们仅讨论**二元**逻辑回归。

8.4.1 拉普拉斯近似

本节讨论如何对后验分布做**高斯近似**。做法如下。设 $\theta \in \mathbb{R}^D$ ，令

$$p(\theta | D) = \frac{1}{Z} e^{-E(\theta)} \quad (8.49)$$

其中 $E(\theta)$ 称为**能量函数**，等于未归一化后验的负对数： $E(\theta) = -\log p(\theta, D)$ ， $Z = p(D)$ 是归一化常数。围绕**众数** θ^{**} （即最低能量点）做泰勒展开，有

$$E(\theta) \approx E(\theta^{**}) + (\theta - \theta^{**})^\top g + \frac{1}{2}(\theta - \theta^{**})^\top H(\theta - \theta^{**}) \quad (8.50)$$

其中 g 是梯度、 H 是能量在众数处的海森矩阵：

$$g = \nabla E(\theta)|_{\theta^{**}}, \quad H = \frac{\partial^2 E(\theta)}{\partial \theta \partial \theta^\top} \Big|_{\theta^{**}}. \quad (8.51)$$

由于 θ^{**} 是众数，梯度项为 0。于是

$$p(\theta | D) \approx \frac{1}{Z} e^{-E(\theta^{**})} \exp(-\frac{1}{2}(\theta - \theta^{**})^\top H(\theta - \theta^{**})) \quad (8.52)$$

$$= N(\theta | \theta^{**}, H^{-1}) \quad (8.53)$$

$$Z = p(D) \approx \int p(\theta | D) d\theta = e^{-E(\theta^{**})} (2\pi)^{D/2} |H|^{-1/2}. \quad (8.54)$$

最后一行来自多元高斯的归一化常数。式 (8.54) 被称为**边缘似然**的拉普拉斯近似。因此，式 (8.52) 有时被称为后验的拉普拉斯近似。不过在统计学界，“拉普拉斯近似”一词指的是更复杂的方法（详见 Rue 等, 2009），因此把式 (8.52) 称为“**高斯近似**”或许更好。当样本量增大时，后验往往会因类似中心极限定理的原因而更“像高斯”，因此高斯近似通常是合理的。（在物理中有一个类似的技巧称为**鞍点近似**。）

通俗解释（基础且详尽）

这段总体在讲什么？

- 以前（第 8.3 节）我们用的是**点估计**：MLE 或 MAP，给出一个最优的 w 。
- 现在（8.4）进入**贝叶斯视角**：不只给一个点，而是要整个**参数分布** $p(w | D)$ 。有了它，就能量化**不确定性**（例如给预测概率一个置信区间/可信区间），在探索-利用（情境赌博机）等问题里非常关键。
- 但逻辑回归**没有共轭先验**，后验没法“写成一个已知分布”并精确求出。于是用**近似推断**。本节讲的是**拉普拉斯 (= 高斯) 近似**：把后验在众数附近用高斯来近似。

关键思路（一步到位）

- 定义“能量” $E(\theta) = -\log p(\theta | D)$ （负的联合对数）。
 - 等价地， $-\log p(\theta | D) = E(\theta) - \log Z$ ，所以最小化 E 就是最大化后验（求 MAP）。
- 找到**众数** θ^* （即 MAP）： $\theta^* = \arg \max_{\theta} p(\theta | D)$ 。
 - 在逻辑回归里，这就是把**NLL + 正则项最小化**（前面 8.3.6 的 L_2 正则，对应高斯先验）。
- 在 θ^* 附近，用**二次函数**（二阶泰勒）去近似 $-\log p(\theta | D)$ 。
 - 形象地说：把原本凹凸不平的山谷，在谷底附近“抛物面化”。
- 二次函数对应的分布正是**高斯** $N(\theta^*, H^{-1})$ 。
 - 其中 H 是能量 E 在 θ^* 处的海森（即 $-\log$ 后验的二阶导）。
 - 因为是“碗形”才像高斯，所以需要 H **正定**（在逻辑回归 + L_2 正则的凸设定下通常成立）。
- 同时还能得到**模型证据（边缘似然）的近似（式 8.54）**，可用于**模型比较/超参选择**。

把它套到逻辑回归上（很实用的版本）

- 设先验 $w \sim N(0, \Sigma_0)$ （常见的 L_2 正则）。
- 能量**：

$$E(w) = \underbrace{\text{NLL}(w)}_{\text{负对数似然}} + \underbrace{\frac{1}{2} w^\top \Sigma_0^{-1} w}_{\text{先验惩罚}} + \text{常数.}$$

- 众数 (MAP)** w^* ：用 8.3 节的优化器（IRLS/牛顿/拟牛顿）求解最小化 $E(w)$ 。
- 海森**（在 w^* ）：

$$H = \nabla^2 E(w^*) = \underbrace{X^\top S X}_{\text{来自似然}} + \underbrace{\Sigma_0^{-1}}_{\text{来自先验}},$$

其中 $S = \text{diag}(\mu_i(1 - \mu_i))$, $\mu_i = \sigma(w^{*\top} x_i)$ 。

- 后验高斯近似**：

$$p(w \mid D) \approx N(w \mid w^*, H^{-1}).$$

- **边缘似然** (用于选 Σ_0 或 λ) :

$$\log p(D) \approx -E(w^*) - \frac{1}{2} \log |H| + \frac{D}{2} \log(2\pi).$$

这能带来什么 (预测与不确定性) ?

- **参数不确定性**: 对每个权重 w_j 有近似方差 $(H^{-1})_{jj}$, 这给了参数层面的可信区间。
- **预测不确定性**: 想算

$$p(y = 1 \mid x, D) = \int \sigma(w^\top x) p(w \mid D) dw.$$

用高斯近似后, $w^\top x \sim N(m, s^2)$ (其中 $m = w^{*\top} x$, $s^2 = x^\top H^{-1} x$) 。

该积分没有闭式, 但常用**近似** (很准的经验公式) :

$$E[\sigma(a)] \approx \sigma\left(\frac{m}{\sqrt{1 + \frac{\pi}{8}s^2}}\right).$$

这就给出了**带不确定性校正**的预测概率; 当 s^2 大 (不确定性高) 时, 概率会被“往 0.5 收缩”, 更保守。

和前面 (频率学派的 MLE/MAP) 有什么区别?

- **前面**: 只给一个点 (w_{MLE} 或 w_{MAP}), 没有不确定性刻画。
- **这里**: 给出整个 (近似的) **分布** $N(w^*, H^{-1})$, 能:
 - 量化参数和预测的不确定性 (可信区间、置信带) ;
 - 用边缘似然近似做模型选择;
 - 在探索-利用等需要“置信上界/汤普森采样”的任务里自然适用。

什么时候它靠谱 / 可能不太准?

- **靠谱**: 数据量较大、后验比较单峰且“像个碗”, 逻辑回归 + L_2 正则的常见场景。
- **可能不准**: 数据很少、后验严重偏斜/多峰、或可分导致曲面极陡 (尽管 L_2 会缓解)。这时可考虑 MCMC、变分、EP 等更强的近似。

小结 (把步骤背下来)

1. 选先验 (如 $w \sim N(0, \Sigma_0)$) 。
2. 求 MAP: 最小化 $E(w) = NLL(w) + \frac{1}{2}w^\top \Sigma_0^{-1} w$, 得 w^* 。
3. 算海森 H (或其近似) 并取协方差 H^{-1} 。
4. 得后验近似 $N(w^*, H^{-1})$, 可出参数/预测的可信区间; 边缘似然近似也随之得到。

这样, 你就把“点估计的逻辑回归”升级成“能量化不确定性的贝叶斯逻辑回归 (高斯近似版) ”了。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.4.2 Derivation of the BIC

We can use the Gaussian approximation to write the log marginal likelihood as follows, dropping irrelevant constants:

$$\log p(D) \approx \log p(D|\theta^*) + \log p(\theta^*) - \frac{1}{2} \log |H| \quad (8.55)$$

The penalization terms which are added to the $\log p(D|\theta^*)$ are sometimes called the Occam factor, and are a measure of model complexity. If we have a uniform prior, $p(\theta) \propto 1$, we can drop the second term, and replace θ^* with the MLE, $\hat{\theta}$.

We now focus on approximating the third term. We have $H = N \sum_{i=1}^D H_i$, where $H_i = \nabla \nabla \log p(D_i|\theta)$. Let us approximate each H_i by a fixed matrix H^* . Then we have $\log |H| = \log |N H^*| = \log(N \det |H^*|) = D \log N + \log |H^*|$ (8.56)

where $D = \dim(\theta)$ and we have assumed H is full rank. We can drop the $\log |H^*|$ term, since it is independent of N , and thus will get overwhelmed by the likelihood. Putting all the pieces together, we recover the BIC score (Section 5.3.2.4):

$$\log p(D) \approx \log p(D|\hat{\theta}) - D \log N \quad (8.57)$$

下面先给**完整逐字翻译**，再做**通俗解释**（把每一步近似为什么成立都讲清楚；并指出常见的前提与局限）。

译文

8.4.2 BIC 的推导

我们可以用高斯近似把**对数边缘似然**写成下面这样（丢弃与模型比较无关的常数项）：

$$\log p(D) \approx \log p(D | \theta^*) + \log p(\theta^*) - \frac{1}{2} \log |H| \quad (8.55)$$

加在 $\log p(D | \theta^*)$ 上的这些惩罚项有时称为**奥卡姆因子** (Occam factor)，它是对模型复杂度的一种度量。若我们使用**均匀先验** $p(\theta) \propto 1$ ，就可以丢掉第二项，并把 θ^* 换成

MLE, 记为 $\hat{\theta}$ 。

我们现在把注意力放到第三项的近似上。我们有 $H = \sum_{i=1}^N H_i$, 其中 $H_i = \nabla \nabla \log p(D_i | \theta)$ 。

让我们把每个 H_i 都近似为同一个固定矩阵 \hat{H} 。于是

$$\log |H| = \log |N \hat{H}| = \log (N^D |\hat{H}|) = D \log N + \log |\hat{H}| \quad (8.56)$$

其中 $D = \dim(\theta)$, 且我们假设 H 满秩。我们可以丢掉 $\log |\hat{H}|$ 这一项, 因为它与 N 无关, 因此会被似然项的规模所“淹没”。把各部分合并起来, 我们就得到 **BIC 评分** (见 5.3.2.4 节) :

$$\log p(D) \approx \log p(D | \hat{\theta}) - \frac{D}{2} \log N. \quad (8.57)$$

解释 (基础且详尽)

1) 这段在干什么?

- 目标: 给出对数边缘似然 $\log p(D)$ 的一个可计算近似, 用于模型比较/选模。
- 方法: 用上一小节的高斯 (拉普拉斯) 近似: 把后验在众数附近近似成 $N(\theta^{(*)}, H^{-1})$, 从而得到

$$\log p(D) \approx \log p(D | \theta^{(*)}) + \log p(\theta^{(*)}) - \frac{1}{2} \log |H| + \text{常数}.$$

这里的 $-\frac{1}{2} \log |H|$ 就是“奥卡姆因子”(复杂度惩罚)。

2) 为什么能把第三项写成 $D \log N + \text{常数}$?

- 数据独立同分布时, 似然是各样本似然的乘积; 其**曲率 (海森) **在众数附近可写作

$$H = \sum_{i=1}^N H_i, \quad H_i = \nabla \nabla \log p(D_i | \theta) \text{ (在 } \theta^{(*)} \text{ 处评估).}$$

(很多教材用负号把它定义成“观测信息矩阵”, 数值上结论一致; 这里按原文记号。)

- 当样本量大、每个样本“贡献的曲率”差不多时, 可以用同一个 \hat{H} 来近似每个 H_i :

$$H \approx N \hat{H}.$$

- 行列式的基本性质: $|N \hat{H}| = N^D |\hat{H}|$ (因为把矩阵整体放大 N 倍, 相当于对每个特征方向都乘 N , 乘积放大 N^D)。

取对数即

$$\log |H| \approx D \log N + \log |\hat{H}|.$$

- $\log |\hat{H}|$ 与 N 无关，在大样本比较里可视作常数，被前面的对数似然尺度“淹没”，因此丢掉。

3) 合并：得到 BIC

把（均匀先验下） $\log p(\theta^{(*)})$ 去掉，把 $\theta^{(*)}$ 改成 $\hat{\theta}$ ，并把
 $-\frac{1}{2} \log |H| \approx -\frac{1}{2}(D \log N + \text{常数})$ 中的常数丢掉，就得到

$$\log p(D) \approx \log p(D | \hat{\theta}) - \frac{D}{2} \log N.$$

这就是 **BIC**（贝叶斯信息准则）：**最大化**它等价于在“拟合好”与“越复杂罚得越重”之间折中。

- 第一项：训练拟合度（对数似然）。
- 第二项：复杂度惩罚，随参数维度 D 和样本量 N 增大而改变：
 - D 越大，罚得越重；
 - N 越大， $\log N$ 越大，对过度复杂模型惩罚更强（“数据越多，越能区分谁在真学、谁在凑巧”）。

4) 奥卡姆因子直觉

$-\frac{1}{2} \log |H|$ 衡量了后验高斯的“体积”（协方差的行列式的倒数的对数）。

- **简单模型**：曲率大（碗“窄”）， $|H|$ 大 $\rightarrow -\frac{1}{2} \log |H|$ 更负（惩罚小），支持度高。
- **复杂模型**：曲率小（碗“宽”）， $|H|$ 小 \rightarrow 惩罚大。

这就是“**奥卡姆剃刀**”：在解释数据足够好时，**更简单的模型更受青睐**。

5) 一个小算例（感受量级）

- $N = 1000, D = 5$ 。
- 惩罚项 $= \frac{D}{2} \log N = 2.5 \times \log 1000 \approx 2.5 \times 6.907 = 17.27$ 。
- 也就是：更复杂 5 个参数，**需要提升至少约 17.3 的对数似然**，才“值得”。

6) 与 AIC 的区别（顺带）

- **AIC**： $AIC = -2 \log p(D | \hat{\theta}) + 2D$ ，更偏向**预测误差**的无偏估计；
- **BIC**： $BIC = -2 \log p(D | \hat{\theta}) + D \log N$ ，来源于**边缘似然（Bayes 证据）**近似，样本越多对复杂度惩罚越强；
- 大样本时，BIC 更“保守”，更偏向**选简单模型**。

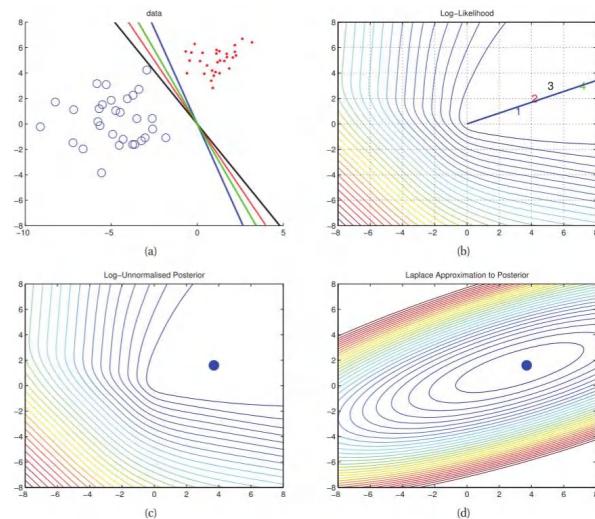
7) 成立的前提与注意事项

- 模型是**正则的**、参数维度 D 固定、后验**单峰且 $\theta^{(*)}$ 附近“像高斯”**；

- H 满秩 (曲率不是退化的) ;
- 大样本近似 ($\log |\hat{A}|$ 可视为常数) ;
- 对**奇异模型** (如混合模型、层次模型中有不可识别性) 或强先验、非 i.i.d. 情形, BIC 可能不可靠, 这时可考虑更精细的证据近似或交叉验证。

小结

- 从**拉普拉斯 (高斯)** 近似出发, $\log p(D) \approx \log p(D | \hat{\theta}) - \frac{D}{2} \log N$ (忽略常数) 。
- 第一项衡量拟合好坏, 第二项是“奥卡姆因子”惩罚复杂度。
- 这就是 BIC 的来龙去脉: 用一个**简单可算的式子**近似比较不同模型的 Bayes 证据。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.5 (a) Two-class data in 2d. (b) Log-likelihood for a logistic regression model. The line is drawn from the origin in the direction of the MLE (which is at infinity). The numbers correspond to 4 points in parameter space, corresponding to the lines in (a). (c) Unnormalized log posterior (assuming vague spherical prior). (d) Laplace approximation to posterior. Based on a figure by Mark Girolami. Figure generated by logregLaplaceGirolamiDemo.

8.4.3 Gaussian approximation for logistic regression Now let us apply the Gaussian approximation to logistic

regression. We will use a Gaussian prior of the form $p(w) = N(w|0, V_0)$, just as we did in MAP estimation. The approximate posterior is given by

$$p(w|D) \approx N(w|\hat{w}, H^{-1}) \quad (8.58)$$

where $\hat{w} = \operatorname{argmin}_w E(w)$, $E(w) = -(\log p(D|w) + \log p(w))$, and $H = \nabla^2 E(w)|\hat{w}$.

As an example, consider the linearly separable 2D data in Figure 8.5(a). There are many parameter settings that correspond to lines that perfectly separate the training data; we show 4 examples. The likelihood surface is shown in Figure 8.5(b), where we see that the likelihood is unbounded as we move up and to the right in parameter space, along a ridge where $w_2/w_1 = 2.35$ (this is indicated by the diagonal line). The reasons for this is that we can maximize the likelihood by driving $\|w\|$ to infinity (subject to being on this line), since large regression weights make the sigmoid function very steep, turning it into a step function. Consequently the MLE is not well defined when the data is linearly separable.

To regularize the problem, let us use a vague spherical prior centered at the origin, $N(w|0, 100I)$. Multiplying this spherical prior by the likelihood surface results in a highly skewed posterior, shown in Figure 8.5(c). (The posterior is skewed because the likelihood function “chops off” regions of parameter space (in a “soft” fashion) which disagree with the data.) The MAP estimate is shown by the blue dot. Unlike the MLE, this is not at infinity. The Gaussian approximation to this posterior is shown in Figure 8.5(d). We see that this is a symmetric distribution, and therefore not a great approximation. Of course, it gets the mode correct (by construction), and it at least represents the fact that there is more uncertainty along the southwest-northeast direction (which corresponds to uncertainty about the orientation of separating lines) than perpendicular to this. Although a crude approximation, this is surely better than approximating the posterior by a delta function, which is what MAP estimation does.

已思考 45s

下面先给**完整逐字翻译**, 再做**通俗解释** (把图里每个细节、式子里的符号和背后的直觉都讲清楚)。

译文

图 8.5

- (a) 二分类的二维数据。
- (b) 逻辑回归模型的对数似然。那条直线是从原点朝着 MLE 的方向画出的（MLE 在无穷远处）。图中的数字对应于参数空间里的 4 个点，它们与 (a) 图中的 4 条直线一一对应。
- (c) 未归一化的对数后验（假设一个“模糊的球形先验”）。
- (d) 对后验的拉普拉斯近似。基于 Mark Girolami 的一幅图。该图由 `logregLaplaceGirolamiDemo` 生成。

8.4.3 逻辑回归的高斯近似

现在把高斯近似用到逻辑回归上。我们使用与 MAP 估计相同形式的高斯先验 $p(w) = N(w | 0, V_0)$ 。近似后的后验为

$$p(w | D) \approx N(w | \hat{w}, H^{-1}) \quad (8.58)$$

其中 $\hat{w} = \arg \min_w E(w)$, $E(w) = -(\log p(D | w) + \log p(w))$, 并且 $H = \nabla^2 E(w)|_{\hat{w}}$ 。

作为例子, 考虑图 8.5(a) 中**线性可分**的二维数据。很多组参数都会给出**完美分开**训练集的直线; 我们展示了 4 个例子。(b) 图画出了似然面, 在参数空间中沿着一条脊线(其方向满足 $w_2/w_1 = 2.35$, 图中以对角线表示)向右上走时, 似然会**不断增大且没有一个有限的最大值**。原因是我们可以让 $\|w\| \rightarrow \infty$ (并保持在这条方向上), 这样回归权重变得很大, Sigmoid 函数极陡, 极限变成一个阶跃函数, 从而把训练样本的概率质量推到极端。因此, 当数据线性可分时, **MLE 没有良好定义** (不存在有限的最优 w)。

为使问题可解, 我们在原点放一个“模糊的、球形的”先验 $N(w | 0, 100I)$ 。把这个球形先验与似然相乘后的后验分布**高度偏斜**, 见图 8.5(c)。(后验之所以偏斜, 是因为似然会以“柔和”的方式把与数据冲突的参数区间“切掉”。) 蓝点是 MAP 估计——与 MLE 不同, 它**不在无穷远**。对该后验的高斯(拉普拉斯)近似见图 8.5(d)。可以看到这是一个**对称分布**, 因此并不是一个非常好的近似。当然, 它(按构造)把**众数**放对了, 并且至少刻画出了这样一个事实: 沿着**西南—东北方向**(对应“分隔直线的朝向”这件事)的不确定性要比垂直方向更大。虽然这种近似较为粗糙, 但仍然明显优于把后验近似成一个 **δ 函数**(这正是 MAP/点估计所做的)。

通俗解释 (基础且详尽)

1) 这段的目的

把第 8.4.1 节的高斯/拉普拉斯近似真正应用到逻辑回归上，解决线性可分数据里 MLE 不存在（参数发散）的难题，并展示先验+拉普拉斯近似能给出一个**有限、有不确定性刻画的后验**。

2) 高斯先验 + 拉普拉斯近似：配方一行记

- 选先验: $w \sim N(0, V_0)$ (例如 $V_0 = 100I$ 表示“模糊/宽”的球形先验）。
- 构造能量: $E(w) = \text{NLL}(w) + \frac{1}{2}w^\top V_0^{-1}w$ 。
- 先求**众数/极小点** \hat{w} (就是 MAP)。
- 用 \hat{w} 处的海森 $H = \nabla^2 E(\hat{w})$ 得到

$$p(w | D) \approx N(w | \hat{w}, H^{-1}).$$

3) 为什么线性可分时 MLE “跑到无穷大”？

- 线性可分 → 总存在一条直线把正负类完全分开。
- 让 w 沿着这条“正确方向”不断放大，Sigmoid 越来越像“硬阈值”，对训练样本的预测概率趋近 1 (正类) / 0 (负类)。
- 于是对数似然不断上升并趋近其上界 (0)，但没有任何**有限的** w 真正达到该上界
→ **没有有限解**。
- 图 8.5(b) 里的“斜脊线 $w_2/w_1 = 2.35$ ”就是“能把数据分开”的方向；沿这个方向把 $\|w\|$ 做大，似然就一路变好。

4) 先验如何“拉回”发散的解？

- 球形高斯先验 $N(0, 100I)$ 给参数加了“弹簧”：太大的权重会被惩罚。
- 于是**后验 = 似然 × 先验在某个有限的位置达最大** (即图中的蓝点 MAP)。
- 但由于似然的形状沿那条脊线“拉得很长”，乘上先验后的后验就会**偏斜** (不是椭圆对称)，见图 8.5(c)。

5) 拉普拉斯 (高斯) 近似的好与坏

- **做法：**在 MAP 点 \hat{w} 附近把 $-\log$ 后验用**二次函数**近似 (就像把谷底看成椭圆碗)，得到高斯。
- **好处：**
 - 给出**参数的不确定性** (协方差 H^{-1})，比 MAP 的“ δ 函数” (只给一个点、方差 0) 强得多；
 - 求起来很快：只需一次二阶信息 (或其近似)。
- **局限：**
 - 真实后验若**偏斜/长尾/多峰**，高斯 (对称椭圆) 会不太准；

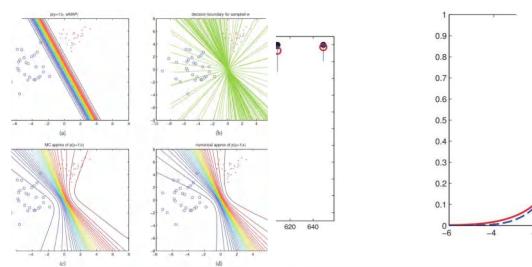
- 图 8.5(d) 就体现了：真实后验很偏，但高斯是对称的，只能大致反映“哪个方向更不确定”。

6) 图 8.5 怎么读

- (a) 蓝圈 vs 红点两团数据，右侧几条彩色直线都能把它们分开。
- (b) 以 $w = (w_1, w_2)$ 为坐标画出的**对数似然等高线**：沿着斜线方向（斜率 2.35）往右上，似然不断变好 \rightarrow MLE 在无穷远。数字 1–4 表示参数空间中的 4 个点，与 (a) 的 4 条直线对应。
- (c) 乘以先验后的**未归一化对数后验**：出现一个明显的**歪斜峰**；蓝点是 MAP。
- (d) **拉普拉斯近似**：以 MAP 为中心画出的椭圆形高斯——对称、能量等高线是“**椭圆圈**”。

小结

- 线性可分时，逻辑回归的 MLE **不存在** ($\|w\| \rightarrow \infty$ 沿一条脊线)。
- 加一个**宽的高斯先验** \rightarrow 后验有**有限的众数 (MAP)**。
- 在 MAP 附近用**拉普拉斯/高斯近似**: $p(w | D) \approx N(\hat{w}, H^{-1})$ 。
- 该近似**粗但有用**：正确找到了众数，还提供了“哪个方向不确定性更大”的信息，比单纯的 MAP (δ 函数) 要好得多。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.6 Posterior predictive distribution for a logistic regression model in 2d. Top left: contours of $p(y=1|x, \hat{w}_{map})$. Top right: samples from the posterior predictive distribution. Bottom left: Averaging over these samples. Bottom right: moderated output (probit approximation). Based on a figure by Mark Girolami. Figure generated by logregLaplaceGirolamiDemo.

Figure 8.7 (a) Posterior predictive density for SAT data. The red circle denotes the posterior mean, the blue cross the posterior median, and the blue lines denote the 5th and 95th percentiles of the predictive distribution. Figure generated by logregSATdemoBayes. (b) The logistic (sigmoid) function $\text{sigm}(x)$ in solid red, with the rescaled probit function $\Phi(\lambda x)$ in dotted blue superimposed. Here $\lambda = \pi/8$, which was chosen so that the derivatives of the two curves match at $x = 0$. Based on Figure 4.9 of (Bishop 2006b). Figure generated by probitPlot. Figure generated by probitRegDemo.

8.4.4 Approximating the posterior predictive

Given the posterior, we can compute credible intervals, perform hypothesis tests, etc., just as we did in Section 7.6.3.3 in the case of linear regression. But in machine learning, interest usually focusses on prediction. The posterior predictive distribution has the form

$$p(y|x, D) = p(y|x, w)p(w|D)dw \quad (8.59)$$

Unfortunately this integral is intractable. The simplest approximation is the plug-in approximation, which, in the binary case, takes the form

$$p(y=1|x, D) \approx p(y=1|x, E[w]) \quad (8.60)$$

where $E[w]$ is the posterior mean. In this context, $E[w]$ is called the Bayes point. Of course, such a plug-in estimate underestimates the uncertainty. We discuss some better approximations below.

8.4.4.1 Monte Carlo approximation

A better approach is to use a Monte Carlo approximation, as follows:

$$p(y=1|x, D) \approx \frac{1}{S} \sum_{s=1}^S \text{sigm}((w_s)^T x) \quad (8.61)$$

where $w_s \sim p(w|D)$ are samples from the posterior. (This technique can be trivially extended to the multi-class case.) If we have approximated the posterior using Monte Carlo, we can reuse these samples for prediction. If we made a Gaussian approximation to the posterior, we can draw independent samples from the Gaussian using standard methods. Figure 8.6(b) shows samples from the posterior predictive for our 2d example. Figure 8.6(c) shows the average of these samples. By averaging over multiple predictions, we see that the uncertainty in the decision boundary “splays out” as we move further from the training data. So although the decision boundary is

linear, the posterior predictive density is not linear. Note also that the posterior mean decision boundary is roughly equally far from both classes; this is the Bayesian analog of the large margin principle discussed in Section 14.5.2.2. Figure 8.7(a) shows an example in 1d. The red dots denote the mean of the posterior predictive evaluated at the training data. The vertical blue lines denote 95% credible intervals for the posterior predictive; the small blue star is the median. We see that, with the Bayesian approach, we are able to model our uncertainty about the probability a student will pass the exam based on his SAT score, rather than just getting a point estimate.

8.4.4.2 Probit approximation (moderated output)

If we have a Gaussian approximation to the posterior $p(w|D) \approx N(w|mN, VN)$, we can also compute a deterministic approximation to the posterior predictive distribution, at least in the binary case. We proceed as follows:

$$p(y=1|x, D) \approx \text{sigm}(w^T x)p(w|D)dw = \text{sigm}(a)N(a|\mu_a, \sigma^2_a)da \quad (8.62)$$

$$a = w^T x \quad (8.63)$$

$$\mu_a = E[a] = m^T Nx \quad (8.64)$$

$$\sigma^2_a = \text{var}[a] = p(a|D)[a^2 - E[a^2]]da \quad (8.65)$$

$$= p(w|D)[(w^T x)^2 - (m^T Nx)^2]dw = x^T V N x \quad (8.66)$$

Thus we see that we need to evaluate the expectation of a sigmoid with respect to a Gaussian. This can be approximated by exploiting the fact that the sigmoid function is similar to the probit function, which is given by the cdf of the standard normal:

$$\Phi(a) = \int_{-\infty}^a N(x|0, 1)dx \quad (8.67)$$

Figure 8.7(b) plots the sigmoid and probit functions. We have rescaled the axes so that $\text{sigm}(a)$ has the same slope as $\Phi(\lambda a)$ at the origin, where $\lambda^2 = \pi/8$. The advantage of using the probit is that one can convolve it with a Gaussian analytically:

$$\Phi(\lambda a)N(a|\mu, \sigma^2)da = \Phi(a)(\lambda^2 - 2 + \sigma^2)^{-1/2} \quad (8.68)$$

$$\text{sigm}(a)N(a|\mu, \sigma^2)da \approx \text{sigm}(\kappa(\sigma^2)\mu) \quad (8.69)$$

$$\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2} \quad (8.70)$$

Applying this to the logistic regression model we get the following expression (first suggested in (Spiegelhalter and Lauritzen 1990)):

$$p(y=1|x, D) \approx \text{sigm}(\kappa(\sigma^2 a)\mu_a) \quad (8.71)$$

Figure 8.6(d) indicates that this gives very similar results to

the Monte Carlo approximation. Using Equation 8.71 is sometimes called a moderated output, since it is less extreme than the plug-in estimate. To see this, note that $0 \leq \kappa(\sigma^2) \leq 1$ and hence

$$\text{sigm}(\kappa(\sigma^2)\mu) \leq \text{sigm}(\mu) = p(y=1|x, \hat{w}) \quad (8.72)$$

where the inequality is strict if $\mu = 0$. If $\mu > 0$, we have $p(y=1|x, \hat{w}) > 0.5$, but the moderated prediction is always closer to 0.5, so it is less confident. However, the decision boundary occurs whenever $p(y=1|x, D) = \text{sigm}(\kappa(\sigma^2)\mu) = 0.5$, which implies $\mu = \hat{w}^T x = 0$. Hence the decision boundary for the moderated approximation is the same as for the plug-in approximation. So the number of misclassifications will be the same for the two methods, but the log-likelihood will not. (Note that in the multiclass case, taking into account posterior covariance gives different answers than the plug-in approach: see Exercise 3.10.3 of (Rasmussen and Williams 2006).)

已思考 19s

下面先给**完整逐字翻译**（不遗漏），再做**通俗解释**（把每个式子、图和结论都讲清楚；遇到常见易混淆处我会随手备注）。

译文

图 8.6 逻辑回归在二维中的**后验预测分布**。左上： $p(y=1 | x, \hat{w}_{MAP})$ 的等高线。右上：从**后验预测分布**中采样得到的决策边界。左下：对这些样本进行**平均**后的结果。右下：**温和化输出**（probit 近似）。基于 Mark Girolami 的一幅图。图由 `logregLaplaceGirolamiDemo` 生成。

图 8.7 (a) SAT 数据的后验预测密度。红色圆圈表示后验预测的**均值**，蓝色叉号表示**中位数**，蓝色竖线表示后验预测分布的 5% 与 95% **分位数**。图由 `logregSATdemoBayes` 生成。
(b) 实线红色是 logistic (sigmoid) 函数 $\text{sigm}(x)$ ，叠加虚线蓝色是重新缩放后的 probit 函数 $\Phi(\lambda x)$ 。这里 $\lambda = \pi/8$ ，它是为了在 $x = 0$ 处让两条曲线的**导数相同**而选取的。基于 (Bishop 2006b) 的图 4.9。图由 `probitPlot` 与 `probitRegDemo` 生成。

8.4.4 近似计算后验预测

在得到后验之后，我们就可以像第 7.6.3.3 节（线性回归）那样计算可信区间、做假设检验等。不过在机器学习里，我们通常更关心**预测**。后验预测分布为

$$p(y \mid x, D) = \int p(y \mid x, w) p(w \mid D) dw \quad (8.59)$$

不幸的是，这个积分是不可求解析解的。最简单的近似是Plug-in (代入) 近似；在二分类情形下，它写成

$$p(y = 1 \mid x, D) \approx p(y = 1 \mid x, E[w]) \quad (8.60)$$

其中 $E[w]$ 是后验均值。在此语境下， $E[w]$ 被称为 Bayes 点。当然，这种代入估计会低估不确定性。下面讨论更好的近似。

8.4.4.1 Monte Carlo 近似

一个更好的办法是用 Monte Carlo 近似：

$$p(y = 1 \mid x, D) \approx \frac{1}{S} \sum_{s=1}^S \text{sigm}((w^{(s)})^\top x) \quad (8.61)$$

其中 $w^{(s)} \sim p(w \mid D)$ 是从后验抽样得到的（这个技巧对多分类可直接推广）。如果我们已经用 Monte Carlo 近似了后验，那么可以复用这些样本来做预测；如果我们用的是高斯近似的后验，则可以从该高斯独立采样。图 8.6(b) 给出了二维示例的后验预测采样。图 8.6(c) 展示了这些样本的平均。通过对多次预测求平均，我们看到随着远离训练数据，决策边界的不确定性“向外张开”。因此，虽然决策边界本身是线性的，后验预测密度却不是线性的。还要注意，后验均值的决策边界大致离两类样本等远；这是第 14.5.2.2 节“大间隔原则”的一个贝叶斯类比。图 8.7(a) 给出了一个一维示例：红点是训练点处的后验预测均值，蓝色竖线是 95% 可信区间，小蓝星是中位数。可见，贝叶斯方法能够刻画“学生根据 SAT 分数通过考试的概率不确定性”，而不仅给一个点估计。

8.4.4.2 Probit 近似 (温和化输出)

若后验已有高斯近似 $p(w \mid D) \approx N(w \mid m_N, V_N)$ ，在二分类的情况下我们还可以计算一种确定性的后验预测近似。步骤如下：

$$p(y = 1 \mid x, D) \approx \int \text{sigm}(w^\top x) p(w \mid D) dw = \int \text{sigm}(a) N(a \mid \mu_a, \sigma_a^2) da \quad (8.62)$$

$$a = w^\top x \quad (8.63)$$

$$\mu_a = E[a] = m_N^\top x \quad (8.64)$$

$$\sigma_a^2 = \text{var}[a] = \int p(a \mid D) [a^2 - E[a]^2] da = \int p(w \mid D) [(w^\top x)^2 - (m_N^\top x)^2] dw = \text{J}(\bar{x})$$

于是我们需要对“sigmoid 在高斯分布下的期望”做近似。可利用 sigmoid 与 probit (标准正态的 cdf) 很接近这一事实：

$$\Phi(a) = \int_{-\infty}^a N(x | 0, 1) dx \quad (8.67)$$

图 8.7(b) 画出了两者；我们把坐标缩放，使 $\text{sigm}(a)$ 在原点的斜率与 $\Phi(\lambda a)$ 相同（文中给 $\lambda = \pi/8$ 以匹配斜率；常见推导得到 $\lambda = \sqrt{\pi/8}$ ，两者皆用于近似）。使用 probit 的好处是它与高斯的卷积有解析式：

$$\int \Phi(\lambda a) N(a | \mu, \sigma^2) da = \Phi\left(\frac{\mu}{\sqrt{\lambda^{-2} + \sigma^2}}\right) \quad (8.68)$$

因此有近似

$$\int \text{sigm}(a) N(a | \mu, \sigma^2) da \approx \text{sigm}(\kappa(\sigma^2)\mu) \quad (8.69)$$

$$\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2} \quad (8.70)$$

把它用于逻辑回归就得到（最早由 Spiegelhalter 和 Lauritzen, 1990 提出）：

$$p(y = 1 | x, D) \approx \text{sigm}(\kappa(\sigma_a^2)\mu_a) \quad (8.71)$$

图 8.6(d) 显示它与 Monte Carlo 近似的结果非常接近。式 (8.71) 的预测有时称为**温和化 (moderated) 输出**，因为它比代入估计更“不极端”。原因是 $0 \leq \kappa(\sigma^2) \leq 1$ ，因此

$$\text{sigm}(\kappa(\sigma^2)\mu) \leq \text{sigm}(\mu) = p(y = 1 | x, \hat{w}) \quad (8.72)$$

当 $\mu = 0$ 时不等式严格成立。若 $\mu > 0$ ，代入法给 $p(y = 1 | x, \hat{w}) > 0.5$ ，而“温和化”预测更靠近 0.5，置信度更低。不过，决策边界由 $p(y = 1 | x, D) = \text{sigm}(\kappa(\sigma^2)\mu) = 0.5$ 给出，这意味着 $\mu = \hat{w}^\top x = 0$ 。因此，**温和化近似与代入近似的决策边界相同**，两者的误分类数也相同，但对数似然不同。（注意在多分类情形，考虑后验协方差与简单代入法会得到不同答案：见 Rasmussen & Williams, 2006 的练习 3.10.3。）

通俗解释（基础且详尽）

1) “后验预测”到底是什么？

- 不是用一个固定的 w 去预测；而是把**参数不确定性也平均进去**：

$$\underbrace{p(y | x, D)}_{\text{最终预测}} = \int \underbrace{p(y | x, w)}_{\text{用某个 } w \text{ 的预测}} \underbrace{p(w | D)}_{\text{学到的后验}} dw.$$

- 这让我们得到**可信区间**（比如 5%-95%），而不仅是单点概率。

2) 三种近似方法的差别

- **代入法 (Plug-in)** : 把 w 换成后验均值 (或 MAP) , 快但低估不确定性。图 8.6(a) 的等高线很“锋利”。
- **Monte Carlo**: 从后验采样 w , 把对应的预测平均起来。最直观, 图 8.6(b)(c) 显示决策边界随着远离数据而**“向外张开”**。
- **Probit 近似 (温和化输出)** : 当后验已近似为高斯时, 用解析近似

$$\text{sigm}(\kappa\mu) \quad (\mu = m_N^\top x, \kappa = (1 + \pi\sigma_a^2/8)^{-1/2})$$

计算预测, 快且与 MC 结果接近 (图 8.6(d)) 。

3) “温和化”直觉

- $\sigma_a^2 = x^\top V_N x$ 大表明“该方向的参数不确定性大”。 $\kappa(\sigma_a^2) \in (0, 1]$ 会把对数几率拉向 0, 从而把概率拉向 0.5 → 别太自信。
- 但由于只是把 μ 乘以 κ , 0.5 的等值线 (决策边界) 不变: $\mu = 0 \Rightarrow \hat{w}^\top x = 0$ 。

4) 如何读两张图

- **图 8.6:**
 - (a) 用 \hat{w}_{MAP} 的等高线;
 - (b) 从后验采样的很多 w 画出的直线 (每条一条决策边界) ;
 - (c) 对 (b) 的预测平均后, 概率带变宽、边界弯起来;
 - (d) 用式 (8.71) 的确定性近似, 和 (c) 类似。
- **图 8.7(a):** 对每个 SAT 分数, 红圈是后验预测均值, 蓝星中位数, 蓝线 95% 区间 → 展示“这分数下通过考试的概率”的不确定性。
- **图 8.7(b):** sigmoid 与缩放的 probit 曲线几乎重合, 说明用 probit 做解析近似是合理的。 (严格推导常得 $\lambda = \sqrt{\pi/8}$ 以匹配在 0 处的斜率, 文中给 $\lambda = \pi/8$ 可视为近似/记号取值。)

小结

- 后验预测把参数不确定性也整合进预测;
- MC 平均最直观精确, Probit 近似很快而且效果接近;
- “温和化输出”会减小过度自信, 但决策边界不变;
- 图示告诉我们: 即使模型 (线性决策边界) 是线性的, 整合不确定性后的预测面可以是弯的——这才更符合“离数据远就更不确定”的常识。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.4.5 Residual analysis (outlier detection)

It is sometimes useful to detect data cases which are "outliers". This is called residual analysis or case analysis. In a regression setting, this can be performed by computing $r_i = y_i - \hat{y}_i$, where $\hat{y}_i = \hat{w}^T x_i$. These values should follow a $N(0, \sigma^2)$ distribution, if the modelling assumptions are correct. This can be assessed by creating a qq-plot, where we plot the N theoretical quantiles of a Gaussian distribution against the N empirical quantiles of the r_i . Points that deviate from the straightline are potential outliers.

Classical methods, based on residuals, do not work well for binary data, because they rely on asymptotic normality of the test statistics. However, adopting a Bayesian approach, we can just define outliers to be points which which $p(y_i|\hat{y}_i)$ is small, where we typically use $\hat{y}_i = \text{sigm}(\hat{w}^T x_i)$. Note that \hat{w} was estimated from all the data. A better method is to exclude (x_i, y_i) from the estimate of w when predicting y_i . That is, we define outliers to be points which have low probability under the cross-validated posterior predictive distribution, defined by

$$p(y_i|x_i, x_{-i}, y_{-i}) = p(y_i|x_i, w) \prod_{j \neq i} p(y_j | x_j, w) p(w) dw \quad (8.73)$$

This can be efficiently approximated by sampling methods (Gelfand 1996). For further discussion of residual analysis in logistic regression models, see e.g., (Johnson and Albert 1999, Sec 3.4).

已思考 17s

下面先给**完整逐字翻译**，再做**通俗解释**（把每个概念、式子、做法都讲清楚；文末给出可直接操作的小步骤）。

译文

8.4.5 残差分析（离群点检测）

有时识别“离群 (outlier) ”的数据样本很有用。这叫作**残差分析或个案分析**。在**回归场景**里，可以计算

$$r_i = y_i - \hat{y}_i, \quad \text{其中 } \hat{y}_i = \hat{w}^\top x_i.$$

如果建模假设正确，这些残差应当服从 $N(0, \sigma^2)$ 分布。可以用 **qq-plot** 来检验：把高斯分布的 N 个**理论分位数**与残差 r_i 的 N 个**经验分位数**作图对照。偏离直线的点就是潜在的离群点。

基于残差的经典方法在**二元数据**上效果不好，因为它们依赖于检验统计量的**渐近正态性**。不过，采用**贝叶斯方法**，我们可以把“那些 $p(y_i | \hat{y}_i)$ 很小的点”定义为离群点，通常取

$$\hat{y}_i = \text{sigm}(\hat{w}^\top x_i).$$

注意， \hat{w} 是用**全部数据**估计出来的。更好的方法是在预测 y_i 时**把 (x_i, y_i) 从估计 w 的数据里排除**。也就是说，把“在交叉验证的后验预测分布之下概率很小”的点定义为离群点，该分布定义为

$$p(y_i | x_i, x_{-i}, y_{-i}) = \underbrace{\int p(y_i | x_i, w) \left[\prod_{j \neq i} p(y_j | x_j, w) p(w) \right] dw}_{\text{去掉第 } i \text{ 个样本的后验 (未归一化)}}. \quad (8.73)$$

这可以用**采样方法**高效近似 (Gelfand 1996)。关于逻辑回归的残差分析更多讨论，见 Johnson and Albert (1999) 第 3.4 节等。

通俗解释 (基础且详尽)

1) 为什么“残差检测离群”在逻辑回归上不灵？

- 经典做法：**在线性回归里**， $r_i = y_i - \hat{y}_i$ 应该像 $N(0, \sigma^2)$ 。画 qq-plot，看哪些点偏离直线 → 可能是离群。
- 但**逻辑回归里** $y_i \in \{0, 1\}$ ，不存在“近似正态的残差”这一前提；很多“残差检验”依赖的渐近正态性在这里不稳当，灵敏度/特异度都差。

2) 贝叶斯视角更自然：看“这点的概率有多小”

- 逻辑回归给的是**概率**： $\hat{y}_i = \sigma(\hat{w}^\top x_i)$ 。
- 如果真实 $y_i = 1$ 但 \hat{y}_i 很小（或 $y_i = 0$ 但 \hat{y}_i 很大），那这点对当前模型来说就是“不太可能” → 候选离群点。
- 问题： \hat{w} 是用**所有样本**（包括这点本身）估计的，存在“**自我影响/双重使用**”——不够客观。

3) 更严谨：**留一交叉验证 (LOO)**的后验预测概率

- 把第 i 个样本拿掉，用剩余数据得到的（贝叶斯）后验来预测它：

$$p(y_i \mid x_i, D_{-i}) = \int p(y_i \mid x_i, w) p(w \mid D_{-i}) dw,$$

其中 $D_{-i} = \{(x_j, y_j) : j \neq i\}$ 。

- 若这个概率很小（比如 $\min\{p, 1-p\} < \text{阈值}$ ），就更有理由把它视为离群点。
- 文献里通常用采样/重要性采样来近似 LOO (Gelfand 1996；现代常用 PSIS-LOO)。

4) 怎么“量化小”？

- 对二分类，常看

$$u_i = \min\{p(y_i = 1 \mid x_i, D_{-i}), 1 - p(y_i = 1 \mid x_i, D_{-i})\}.$$

u_i 越接近 0，越“不可思议”。

- 也可看对数似然贡献 $\log p(y_i \mid x_i, D_{-i})$ 是否异常小（与其它样本或某个分位数相比）。

5) 怎样高效近似 LOO 概率？

- 若你有后验样本 $\{w^{(s)}\}$ （比如拉普拉斯/高斯近似抽样、MCMC 等）：
 - 朴素**：对每个 i 重新拟合去掉 i 的模型（开销大）。
 - 重要性采样/PSIS-LOO**：用全数据后验样本，给每个样本一个“去掉 i ”的**重要性权重**，再对 $\sigma((w^{(s)})^\top x_i)$ 加权平均——**不必重训**，效率高、效果好（现代做法）。

6) 与传统“残差/诊断”的关系

- 经典二分类里也有**皮尔逊残差**、**偏差残差**等，但它们依赖大样本近似。
- 贝叶斯 LOO-PPD（后验预测分布）方法避免这些近似假设，直接用“这点在模型下的概率”来判断，直观而稳健。

7) 实操小流程（你可以照抄）

- 选一个先验，拟合**贝叶斯逻辑回归**（可用拉普拉斯/高斯近似、MCMC、或变分）。
- 拿到**后验样本** $\{w^{(s)}\}_{s=1}^S$ 或高斯近似 $N(m, V)$ 。
- 对每个样本点 i ：
 - 用**PSIS-LOO/重要性采样**近似 $p(y_i \mid x_i, D_{-i})$
 $\approx \sum_s \tilde{w}_i^{(s)} \sigma((w^{(s)})^\top x_i)$ ，
 其中 $\tilde{w}_i^{(s)}$ 是经稳定化后的权重（和为 1）。
 - 计算 $u_i = \min\{p_i, 1 - p_i\}$ 或 $\log p_i$ ，并与阈值或分位数比较。

4. 标记 u_i 特别小的点为**潜在离群点**；结合**可视化**（如把这些点在平面上标红）与**领域知识**做最终判断。

小提醒：离群≠错误。它可能意味着**录入错误/异常值**，也可能意味着**数据中存在新机制/新群组**，需要进一步建模（加特征、分层模型、混合模型等）。

小结

- 传统残差诊断在**连续回归**可用，但在**二分类**上不可靠。
- 用**贝叶斯后验预测**（最好是 LOO 版本）来定义“离群点 = 概率异常小的样本”，更稳健、直观。
- 计算上可用采样法（重要性采样/PSIS-LOO）高效近似，无需为每个样本重训模型。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.5 Online learning and stochastic optimization

Traditionally machine learning is performed offline, which means we have a batch of data, and we optimize an equation of the following form

$$f(\theta) = \frac{1}{N} \sum_{i=1}^N f(\theta, z_i) \quad (8.74)$$

where $z_i = (x_i, y_i)$ in the supervised case, or just x_i in the unsupervised case, and $f(\theta, z_i)$ is some kind of loss function. For example, we might use

$$f(\theta, z_i) = -\log p(y_i | x_i, \theta) \quad (8.75)$$

in which case we are trying to maximize the likelihood.

Alternatively, we might use

$$f(\theta, z_i) = L(y_i, h(x_i, \theta)) \quad (8.76)$$

where $h(x_i, \theta)$ is a prediction function, and $L(y, \hat{y})$ is some other loss function such as squared error or the Huber loss. In frequentist decision theory, the average loss is called the risk (see Section 6.3), so this overall approach is called empirical risk minimization or ERM (see Section 6.5 for details). However, if we have streaming data, we need to perform online learning, so we can update our estimates as each new data point arrives rather than waiting until “the end” (which may never occur). And even if we have a batch of data, we might want to treat it like a

stream if it is too large to hold in main memory. Below we discuss learning methods for this kind of scenario.

8.5.1 Online learning and regret minimization

Suppose that at each step, "nature" presents a sample z_k and the "learner" must respond with a parameter estimate θ_k . In the theoretical machine learning community, the objective used in online learning is the regret, which is the averaged loss incurred relative to the best we could have gotten in hindsight using a single fixed parameter value:

$$\text{regret}_k = \sum_{t=1}^k f(\theta_t, z_t) - \min_{\theta^* \in \Theta} \sum_{t=1}^k f(\theta^*, z_t) \quad (8.77)$$

For example, imagine we are investing in the stock-market. Let θ_j be the amount we invest in stock j , and let z_j be the return on this stock. Our loss function is $f(\theta, z) = -\theta^T z$. The regret is how much better (or worse) we did by trading at each step, rather than adopting a "buy and hold" strategy using an oracle to choose which stocks to buy. One simple algorithm for online learning is online gradient descent (Zinkevich 2003), which is as follows: at each step k , update the parameters using

$$\theta_{k+1} = \text{proj}_{\Theta}(\theta_k - \eta_k g_k) \quad (8.78)$$

where $\text{proj}_V(v) = \arg\min_{w \in V} \|w - v\|^2$ is the projection of vector v onto space V , $g_k = \nabla f(\theta_k, z_k)$ is the gradient, and η_k is the step size. (The projection step is only needed if the parameter must be constrained to live in a certain subset of \mathbb{R}^d . See Section 13.4.3 for details.) Below we will see how this approach to regret minimization relates to more traditional objectives, such as MLE. There are a variety of other approaches to regret minimization which are beyond the scope of this book (see e.g., Cesa-Bianchi and Lugosi (2006) for details).

下面先给**完整逐字翻译**, 再做**通俗解释** (把每个符号和式子都讲清楚; 最后给一个可操作的小步骤表)。

译文

8.5 在线学习与随机优化

传统上, 机器学习是**离线 (offline) **进行的, 这意味着我们手里有一批数据, 并且我们要优化如下形式的式子

$$f(\theta) = \frac{1}{N} \sum_{i=1}^N f(\theta, z_i) \quad (8.74)$$

其中在有监督情形 $z_i = (x_i, y_i)$, 在无监督情形则只有 x_i ; $f(\theta, z_i)$ 是某种损失函数。

例如, 我们可以用

$$f(\theta, z_i) = -\log p(y_i | x_i, \theta) \quad (8.75)$$

此时我们在做最大似然。或者我们也可以用

$$f(\theta, z_i) = L(y_i, h(x_i, \theta)) \quad (8.76)$$

其中 $h(x_i, \theta)$ 是预测函数, $L(y, \hat{y})$ 是其它损失函数, 如**平方误差**或**Huber 损失**。在频率学派的决策理论中, **平均损失**叫作**风险** (risk) (见 6.3 节), 所以这种整体做法称为**经验风险最小化 (ERM)** (细节见 6.5 节)。

然而, 如果我们面对的是**数据流 (streaming)**, 就需要进行**在线学习 (online learning)**: 每到来一个新样本就更新一次估计, 而不是等到“最后”(最后可能永远不会来)。即便我们手里是一个批量数据, 如果它大到**主存放不下**, 我们也可能要把它当作数据流来处理。下面我们讨论适合这种场景的学习方法。

8.5.1 在线学习与遗憾最小化

设在每一步, “自然”给出一个样本 z_k , 而“学习者”必须回应一个参数估计 θ_k 。在理论机器学习社区里, 在线学习使用的目标是**遗憾 (regret)**, 它是: 相对于“事后回看 (hindsight) 时用一个**固定**参数能做到的最好结果”, 我们所**平均**多付出的损失:

$$\text{regret}_k = \frac{1}{k} \sum_{t=1}^k f(\theta_t, z_t) - \min_{\theta^* \in \Theta} \frac{1}{k} \sum_{t=1}^k f(\theta^{*}, z_t). \quad (8.77)$$

例如, 想象我们在股市投资。令 θ_j 是我们在第 j 支股票上投入的金额, 令 z_j 是这支股票的回报。我们的损失函数是 $f(\theta, z) = -\theta^\top z$ 。遗憾衡量的是: 我们每一步都在交易, 与其相对比“由神谕选择买哪些股票、然后买入并持有”的策略, 我们究竟多好或多差。

一个简单的在线学习算法是**在线梯度下降 (online gradient descent, OGD)** (Zinkevich 2003): 在每一步 k , 用下面的式子更新参数

$$\theta_{k+1} = \text{proj}_{\Theta} (\theta_k - \eta_k g_k) \quad (8.78)$$

其中 $\text{proj}_V(v) = \arg \min_{w \in V} \|w - v\|_2$ 是把向量 v 投影到集合 V 上; $g_k = \nabla f(\theta_k, z_k)$ 是梯度, η_k 是步长。(只有当参数必须被限制在 \mathbb{R}^D 的某个子集里时才需要投影步骤。细节见 13.4.3 节。) 下面我们会看到, 这种**遗憾最小化**的做法如何与更传统的目标 (如 MLE)

相关。关于遗憾最小化还有很多其它方法，不在本书范围内（详见 Cesa-Bianchi 和 Lugosi 2006）。

通俗解释（基础且详尽）

1) 离线 vs 在线：到底差在哪儿？

- 离线/批量：先把整批数据放好，一次性优化平均损失（式 8.74）。
- 在线/流式：数据一条条来；你必须边来边学，且通常只能保留很少历史信息（内存受限）。在线学习不仅适合“真·数据流”，也适合超大批量（当整批装不下内存时）。

2) 为什么式 (8.74)-(8.76) 熟悉？

- (8.75) 把损失取成负对数似然 \Rightarrow 最大似然 (MLE)；
- (8.76) 是通用预测损失（平方、Huber 等），本质都在做 ERM——最小化经验平均损失 (= 频率学派里的“风险”）。

3) “遗憾 (regret) ”是什么直觉？

- 在线场景下，每一步你都给出当前参数 θ_t ，因此也就承担了该步损失 $f(\theta_t, z_t)$ 。
- 回头看全部 k 步，如果当初就知道未来全部数据，用一个固定的最优参数 θ^* 会取得 $\frac{1}{k} \sum f(\theta^*, z_t)$ 的最小平均损失。
- 遗憾 = 你实际走法的平均损失 - 这个“事后最优固定策略”的平均损失。
- 好的在线算法力求遗憾随 k 增长趋近 0（或增长得尽量慢，如 $O(1/\sqrt{k})$ ）。

股市类比（文中例子）：

- 令 θ 是你的持仓向量， z 是各股票当期收益向量。
- 损失 $f(\theta, z) = -\theta^\top z$ （收益越高损失越小）。
- 遗憾衡量“频繁调仓”的你 vs “一开始就买好并持有”的神谕策略，长期看差了多少。

4) 在线梯度下降 (OGD) (8.78) 怎么用？

- 到第 k 步来了一个样本 z_k ：
 1. 计算当前梯度 $g_k = \nabla f(\theta_k, z_k)$ 。
 2. 做一步梯度下降： $\theta_k - \eta_k g_k$ 。
 3. 如需满足约束（如 $\|\theta\|_2 \leq R$ 、 $\theta \geq 0$ 、概率单纯形等），把结果投影回可行集合 Θ ： $\text{proj}_\Theta(\cdot)$ 。
- 选步长 η_k （常见： $\eta_k \propto 1/\sqrt{k}$ 或使用自适应步长/线搜索）。
- OGD 有经典的**遗憾上界**：在凸、Lipschitz 条件下可达 $O(RG\sqrt{k})$ （常数略）。

小提示：当 $f(\theta, z)$ 是单个样本的损失时，OGD 就是我们熟悉的**随机梯度下降 (SGD)**；当把若干样本组成**小批 (mini-batch) **时，就是小批 SGD。这就把“遗憾最小化”的理论与“MLE/ERM 的优化”连接起来了。

5) 为什么有时要“投影”？

- 有些问题参数必须满足约束：
 - 概率向量: $\theta \in \Delta = \{ \theta \geq 0, \mathbf{1}^\top \theta = 1 \}$;
 - 正则化半径: $\|\theta\|_2 \leq R$;
 - 盒约束: $a \leq \theta \leq b$ 。
- 直接梯度步可能走出可行域；**投影**把它拉回去，保证可行且有理论保证（见 13.4.3）。

6) 和 MLE/ERM 的关系

- 批量 ERM 优化 $\frac{1}{N} \sum f(\theta, z_i)$ 。
- OGD/SGD 每次用**一个样本**（或一小批）做一步下降，在期望上等价于对批量目标做梯度下降，且内存/计算友好。
- 因此：**在线学习** ≈ **随机优化**；一个关心**遗憾**（在线对比事后最优），一个关心**训练目标值**（ERM/MLE），但算法形式高度相似。

迷你操作流程（可直接套用）

- 定义损失** $f(\theta, z)$: 如 $-\log p(y | x, \theta)$ 或 $L(y, h(x, \theta))$ 。
- 设定可行域** Θ （如无约束/半径/单纯形）与步长规则 η_k 。
- 对数据流 z_k （或超大批量）循环：
 - 计算 $g_k = \nabla_\theta f(\theta_k, z_k)$;
 - $\theta_{k+1} = \text{proj}_\Theta(\theta_k - \eta_k g_k)$ 。
- 监控：训练损失/验证指标，必要时调步长或用自适应法（AdaGrad/Adam 等）。

小结

- 离线 ERM：优化平均损失；在线学习：**样本到就学**，常用 OGD/SGD。
- 遗憾衡量**“我们 vs 事后最优固定参数”的差距；好算法让遗憾随时间变小。
- OGD 一步: $\theta_{k+1} = \text{proj}_\Theta(\theta_k - \eta_k \nabla f(\theta_k, z_k))$ 。
- 在实践中，它既适合**数据流**，也适合**超大数据**，并与 MLE/ERM 的优化自然衔接。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.5.2 Stochastic optimization and risk minimization

Now suppose that instead of minimizing regret with respect to the past, we want to minimize expected loss in the future, as is more common in (frequentist) statistical learning theory. That is, we want to minimize

$$f(\theta) = E [f(\theta, z)] \quad (8.79)$$

where the expectation is taken over future data.

Optimizing functions where some of the variables in the objective are random is called stochastic optimization.

Suppose we receive an infinite stream of samples from the distribution. One way to optimize stochastic objectives such as Equation 8.79 is to perform the update in Equation 8.78 at each step. This is called stochastic gradient descent or SGD (Nemirovski and Yudin 1978). Since we typically want a single parameter estimate, we can use a running average:

$$\theta_k = \frac{1}{k} \sum_{t=1}^k \theta_t \quad (8.80)$$

This is called Polyak-Ruppert averaging, and can be implemented recursively as follows:

$$\theta_k = \theta_{k-1} - \eta_k (\theta_{k-1} - \theta_k) \quad (8.81)$$

See e.g., (Spall 2003; Kushner and Yin 2003) for details.

8.5.2.1 Setting the step size

We now discuss some sufficient conditions on the learning rate to guarantee convergence of SGD. These are known as the Robbins-Monro conditions:

$$\lim_{k \rightarrow \infty} \eta_k = 0, \quad \sum_{k=1}^{\infty} \eta_k < \infty. \quad (8.82)$$

The set of values of η_k over time is called the learning rate schedule. Various formulas are used, such as $\eta_k = 1/k$, or the following (Bottou 1998; Bach and Moulines 2011):

$$\eta_k = (\tau_0 + k)^{-\alpha} \quad (8.83)$$

where $\tau_0 \geq 0$ slows down early iterations of the algorithm, and $\alpha \in (0, 1]$ controls the rate at which old values are forgotten.

The need to adjust these tuning parameters is one of the main drawbacks of stochastic optimization. One simple heuristic (Bottou 2007) is as follows: store an initial subset of the data, and try a range of η values on this subset;

then choose the one that results in the fastest decrease in the objective and apply it to all the rest of the data. Note that this may not result in convergence, but the algorithm can be terminated when the performance improvement on a hold-out set plateaus (this is called early stopping).

8.5.2.2 Per-parameter step sizes

One drawback of SGD is that it uses the same step size for all parameters. We now briefly present a method known as adagrad (short for adaptive gradient) (Duchi et al. 2010), which is similar in spirit to a diagonal Hessian approximation. (See also (Schaul et al. 2012) for a similar approach.) In particular, if $\theta_i(k)$ is parameter i at time k , and $g_i(k)$ is its gradient, then we make an update as follows:

$$\theta_i(k+1) = \theta_i(k) - \eta g_i(k) \tau_0 + s_i(k) \quad (8.84)$$

where the diagonal step size vector is the gradient vector squared, summed over all time steps. This can be recursively updated as follows:

$$s_i(k) = s_i(k-1) + g_i(k)^2 \quad (8.85)$$

The result is a per-parameter step size that adapts to the curvature of the loss function. This method was originally derived for the regret minimization case, but it can be applied more generally.

8.5.2.3 SGD compared to batch learning

If we don't have an infinite data stream, we can "simulate" one by sampling data points at random from our training set. Essentially we are optimizing Equation 8.74 by treating it as an expectation with respect to the empirical distribution.

In theory, we should sample with replacement, although in practice it is usually better to randomly permute the data and sample without replacement, and then to repeat. A single such pass over the entire data set is called an epoch.

In this offline case, it is often better to compute the gradient of a mini-batch of B data cases. If $B = 1$, this is standard SGD, and if $B = N$, this is standard steepest descent. Typically $B \sim 100$ is used. Although a simple first-order method, SGD performs surprisingly well on some problems, especially ones with large data sets (Bottou 2007). The intuitive reason for this is that one can get a fairly good estimate of the gradient by looking at just a few examples. Carefully evaluating precise gradients using

large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space. As an extreme example, suppose we double the training set by duplicating every example. Batch methods will take twice as long, but online methods will be unaffected, since the direction of the gradient has not changed (doubling the size of the data changes the magnitude of the gradient, but that is irrelevant, since the gradient is being scaled by the step size anyway). In addition to enhanced speed, SGD is often less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise". Consequently it is quite popular in the machine learning community for fitting models with non-convex objectives, such as neural networks (Section 16.5) and deep belief networks (Section 28.1).

已思考 14s

下面先给**完整逐字翻译**, 再做**通俗解释** (把每个式子、名词、直觉和操作步骤都讲清楚)。

译文

8.5.2 随机优化与风险最小化

现在假设我们不再对过去的损失做最小化（最小化遗憾），而是像（频率学派的）统计学习理论里更常见的那样，要最小化**未来的**期望损失。也就是要最小化

$$f(\theta) = \mathbb{E}[f(\theta, z)] \quad (8.79)$$

其中期望是对将来的数据取的。目标函数里有一部分变量是随机的、需要被期望掉，这种优化叫作**随机优化**。假设我们能从该分布收到一条**无限的数据流**。优化 8.79 这类随机目标的一种方法，是在每一步都按式 (8.78) 做更新。这叫作**随机梯度下降** (SGD) (Nemirovski 和 Yudin, 1978)。由于我们通常只想要一个单一的参数估计，可以用**滑动平均**：

$$\bar{\theta}_k = \frac{1}{k} \sum_{t=1}^k \theta_t \quad (8.80)$$

这叫作 Polyak–Ruppert 平均，可以递推地实现如下：

$$\bar{\theta}_k = \bar{\theta}_{k-1} - \frac{1}{k}(\bar{\theta}_{k-1} - \theta_k) \quad (8.81)$$

更多细节见 Spall (2003) ; Kushner 和 Yin (2003) 。

8.5.2.1 步长的设定

我们现在给出一些能保证 SGD 收敛的充要条件（充分条件）用于学习率，这些叫作 Robbins–Monro 条件：

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty. \quad (8.82)$$

$\{\eta_k\}$ 随时间的取值序列叫作**学习率日程** (schedule)。常见的公式有 $\eta_k = 1/k$ ，或者下面这个 (Bottou 1998; Bach 和 Moulines 2011)：

$$\eta_k = (\tau_0 + k)^{-\kappa} \quad (8.83)$$

其中 $\tau_0 \geq 0$ 用来放慢算法在早期几步的速度， $\kappa \in (0.5, 1]$ 用来控制“遗忘旧值”的速度。

需要调这些超参数，是随机优化的主要缺点之一。一个简单的**经验法则** (Bottou 2007) 是：先保存一小段初始数据，在这部分数据上尝试一系列 η ，选择目标下降最快的那个，然后把它用到其余数据上。需要注意的是，这未必保证收敛；但当验证集上的表现进入平台期时可以停止（这叫早停）。

8.5.2.2 按参数设定步长

SGD 的一个缺点是**所有参数共用同一个步长**。我们简要介绍一种方法 adagrad (adaptive gradient, 自适应梯度) (Duchi 等, 2010)，思想与“海森矩阵的对角近似”相似（另见 Schaul 等, 2012 的类似方法）。具体地，若 $\theta_i(k)$ 是第 i 个参数在时刻 k 的值， $g_i(k)$ 是它的梯度，则更新为

$$\theta_i(k+1) = \theta_i(k) - \eta \frac{g_i(k)}{\tau_0 + s_i(k)} \quad (8.84)$$

其中“对角步长向量”由**历次梯度的平方和**构成，递推为

$$s_i(k) = s_i(k-1) + g_i(k)^2. \quad (8.85)$$

这样得到的就是**按参数自适应的步长**，能随损失函数的曲率而变化。该方法最初是在**遗憾最小化**的设定下推导的，但可以更广泛地使用。

8.5.2.3 SGD 与批量学习的比较

如果我们没有无限的数据流，可以通过从训练集**随机采样**来“模拟”数据流。本质上，我们把式 (8.74) 当成对**经验分布**的期望来优化。

理论上，采样应当是**有放回**的；不过在实践里，通常做法是先把数据**随机打乱后无放回**地遍历一遍，再重复。遍历一遍全数据集叫作一个 epoch。

在这种离线情况下，通常更好地做法是对一个大小为 B 的**小批量**求梯度。 $B = 1$ 时，就是标准 SGD； $B = N$ 时，就是标准的**最速下降**。典型地使用 $B \approx 100$ 。尽管只是一个简单的一阶方法，SGD 在某些问题上表现惊人地好，尤其是**大数据集** (Bottou 2007)。直观原因是：只看少量样本也能得到一个**相当不错的**梯度估计；而在大数据上精确计算梯度常常是浪费，因为下一步还要再算一次。与其如此，不如用**带噪声的估计**快速在参数空间里移动。作为一个极端例子，假设我们把训练集**复制一遍**，使规模翻倍。**批量**方法需要两倍时间；**在线**方法不受影响，因为梯度的**方向**没变（数据翻倍只会把梯度的**幅度**翻倍，但这无关紧要，因为我们还会再用步长缩放它）。除了速度提升，SGD 由于额外的“噪声”，也更不容易卡在**浅的局部极小**里。因此，它在机器学习界很流行，用于拟合**非凸**目标的模型，比如神经网络（第 16.5 节）和深度置信网络（第 28.1 节）。

通俗解释（基础且详尽）

1) “随机优化 / 风险最小化”与“遗憾最小化”的区别

- **遗憾最小化**：和过去对比，要求“到目前为止我走得不比事后最优差太多”。
- **风险最小化**：面向未来数据，最小化期望损失 $E[f(\theta, z)]$ (式 8.79)，这就是频率学派的**泛化风险**。
- 算法形式：两者都可用**一步一更新**的梯度法 (式 8.78)，所以 SGD 同时连接了两种视角。

2) SGD + Polyak–Ruppert 平均：为什么好用？

- 单步 SGD 的估计有噪声；对历次 θ_t 做**简单平均** (式 8.80) 能显著降低方差，常常比最后一步 θ_k 更准。递推式 (8.81) 不用存全历史，很省内存。

3) Robbins–Monro 条件直觉 (式 8.82)

- $\sum \eta_k = \infty$ ：总步长必须**无限**，否则可能在到达最优前就“停了”。
- $\sum \eta_k^2 < \infty$ ：步长平方可和**必须有限**，否则噪声会让你一直抖、收不住。
- 例如 $\eta_k = c/(k + \tau_0)$ 或 $(k + \tau_0)^{-\kappa}$ 且 $\kappa \in (0.5, 1]$ 都满足这两个条件。

实操建议：

- 选 $\kappa \approx 0.5 \sim 0.75$, τ_0 让前几十/上百步更稳；

- 也可先在一个小子集网格搜索几种初始 η , 挑下降最快的; 整体用**早停**保障泛化。

4) Adagrad 的意义 (式 8.84–8.85)

- $s_i(k) = \sum g_i(k)^2$ 越大, 说明第 i 个参数方向“历史上梯度大/曲率大”, 步长就被**自动缩小**; 反之增大。
- 这是**每个参数自适应步长**的简单做法, 能减少手工调参压力。
- 说明: 常见实现用 $\sqrt{s_i(k)}$ 或加一个小 ϵ 在分母, 这与文中形式只差一个常数尺度, 思想一致。

5) 为什么小批量 (mini-batch) 常选 $B \sim 100$?

- $B = 1$: 噪声大但**更新频繁**;
- $B = N$: 噪声小但**每步昂贵**;
- 取中间值可以更好地利用并行/向量化, 同时保持足够随机性。
- 多数现代实践中, B 取 32/64/128/256 (由硬件决定) 都很常见。

6) SGD 的三个“好处”

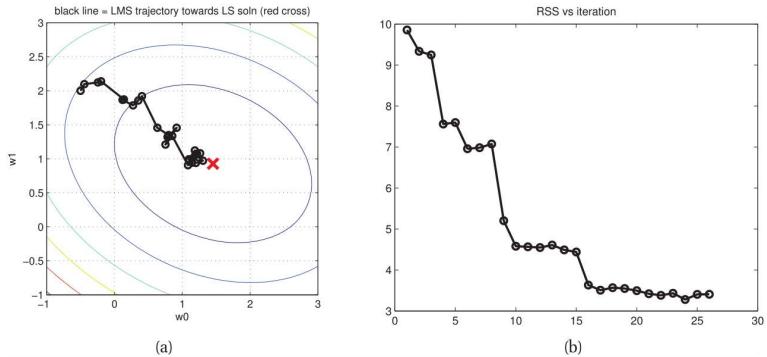
- 速度**: 用少量样本估计梯度 → 每秒更新更多次。
- 鲁棒**: 噪声帮助跳出浅局部极小。
- 可扩展**: 数据翻倍不影响每步成本; 只需多走几个 epoch。

7) 一页纸“怎么用”

- 定义损失** $f(\theta, z)$ (如负对数似然)。
- 选择日程** $\eta_k = (\tau_0 + k)^{-\kappa}$ 或恒定 η +**早停**。
- (可选) Adagrad: 维护 s_i 并用式 (8.84) 更新。
- 小批量**: 每步取 B 个样本, 梯度取平均。
- 输出**: 用 Polyak–Ruppert 平均的 $\bar{\theta}_k$ 作为最终模型。
- 监控**: 训练/验证损失曲线; 当验证集进入平台期就停 (早停)。

小结

- 随机优化最小化的是**未来期望损失** (风险), 算法核心就是 SGD。
- 学习率日程**要满足 Robbins–Monro 条件; 可配合**早停**。
- Adagrad给出**按参数自适应步长**;
- 小批量** SGD在大数据/非凸模型 (如神经网络) 上效果尤其好: 更快、更不易陷入浅局部极小。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.8 Illustration of the LMS algorithm. Left: we start from $\theta = (-0.5, 2)$ and slowly converging to the least squares solution of $\hat{\theta} = (1.45, 0.92)$ (red cross). Right: plot of objective function over time. Note that it does not decrease monotonically. Figure generated by LMSdemo.

8.5.3 The LMS algorithm

As an example of SGD, let us consider how to compute the MLE for linear regression in an online fashion. We derived the batch gradient in Equation 7.14. The online gradient at iteration k is given by

$$g_k = x_i(\theta^T k x_i - y_i) \quad (8.86)$$

where $i = i(k)$ is the training example to use at iteration k . If the data set is streaming, we use $i(k) = k$; we shall assume this from now on, for notational simplicity.

Equation 8.86 is easy to interpret: it is the feature vector x_k weighted by the difference between what we predicted, $\hat{y}^k = \theta^T k x_k$, and the true response, y_k ; hence the gradient acts like an error signal. After computing the gradient, we take a step along it as follows:

$$\theta_{k+1} = \theta_k - \eta_k (\hat{y}_k - y_k) x_k \quad (8.87)$$

(There is no need for a projection step, since this is an unconstrained optimization problem.) This algorithm is called the least mean squares or LMS algorithm, and is also known as the delta rule, or the Widrow-Hoff rule.

Figure 8.8 shows the results of applying this algorithm to the data shown in Figure 7.2. We start at $\theta = (-0.5, 2)$ and converge (in the sense that $\|\theta_k - \theta_{k-1}\|_2^2$ drops below a threshold of 10^{-2}) in about 26 iterations. Note that LMS

may require multiple passes through the data to find the optimum. By contrast, the recursive least squares algorithm, which is based on the Kalman filter and which uses second-order information, finds the optimum in a single pass (see Section 18.2.3). See also Exercise 7.7.

下面先给**完整逐字翻译**, 再做**通俗解释** (把每个式子、名词和图里的结论都讲清楚; 最后附上可直接照抄的伪代码与实用要点)。

译文

图 8.8 LMS 算法示意图。 左: 我们从 $\theta = (-0.5, 2)$ 出发, 缓慢收敛到最小二乘解 $\hat{\theta} = (1.45, 0.92)$ (红色叉号)。右: 目标函数随时间的曲线。注意它**并不单调下降**。图由 *LMSdemo* 生成。

8.5.3 LMS 算法

作为 SGD 的一个例子, 来看看如何以**在线方式为线性回归计算 MLE**。我们在式 (7.14) 推导了批量梯度。第 k 次迭代的**在线梯度**为

$$g_k = x_i(\theta_k^\top x_i - y_i) \quad (8.86)$$

其中 $i = i(k)$ 是在第 k 次迭代所使用的训练样本。如果数据是**流式**的, 我们取 $i(k) = k$; 为了记号简单, 下面都这样假设。式 (8.86) 很容易理解: 它是**特征向量** x_k 乘以“我们预测的 $\hat{y}_k = \theta_k^\top x_k$ ”与“真实响应 y_k ”的差, 因此这个梯度就像一个**误差信号**。计算梯度后, 按如下方式走一步:

$$\theta_{k+1} = \theta_k - \eta_k (\hat{y}_k - y_k) x_k \quad (8.87)$$

(因为这是一个无约束优化问题, 所以不需要投影步骤。) 该算法称为**最小均方 (least mean squares, LMS) 算法**, 也叫**delta 规则**或**Widrow–Hoff 规则**。图 8.8 展示了把该算法应用到图 7.2 数据的结果。我们从 $\theta = (-0.5, 2)$ 起步, 并在大约 26 次迭代后“收敛”(即 $\|\theta_k - \theta_{k-1}\|_2^2$ 低于阈值 10^{-2})。请注意, LMS 可能需要对数据进行**多次遍历**才能找到最优。相比之下, **递归最小二乘 (RLS) **算法——它基于卡尔曼滤波并使用二阶信息——**一次遍历**就能找到最优 (见 18.2.3 节)。另见习题 7.7。

通俗解释 (基础且详尽)

1) 它在做什么?

- 我们要拟合线性回归 $\hat{y} = \theta^\top x$ 。
- 在线地、一点点读入样本 (x_k, y_k) , 每来一个样本就把参数 θ 沿着**当前样本的负梯度方向**更新一次。
- 这就是**随机梯度下降 (SGD) **在平方误差上的具体形式, 传统上叫 LMS / Widrow–Hoff / delta 规则。

2) 式子怎么理解?

- 单样本平方误差: $\frac{1}{2}(\hat{y}_k - y_k)^2$ 。
- 对 θ 的梯度: $\nabla_{\theta} \frac{1}{2}(\hat{y}_k - y_k)^2 = (\hat{y}_k - y_k)x_k$ 。
- 所以更新 (8.87) = 旧参数 - 步长 \times “误差 \times 特征”。
 - 若预测过大 ($\hat{y}_k > y_k$) , θ 朝着 $-x_k$ 方向调小;
 - 若预测过小 ($\hat{y}_k < y_k$) , θ 朝着 $+x_k$ 方向调大。
- 就像用当前样本给参数一个“纠偏”。

3) 图 8.8 该怎么看?

- 左图:** 等高线是最小二乘目标 (凸“碗”), 红叉是真正的最优 $\hat{\theta}$ 。黑色折线是 LMS 的走法: 每次沿当下样本给出的方向迈一小步, 所以路线有些**锯齿/抖动**, 但总体朝最优靠近。
- 右图:** 目标值 (残差平方和) 随迭代变化**不是严格单调**——因为每次只看一个样本, 梯度有噪声, 偶尔会涨一点, 整体仍然往下。

4) 它和批量/二阶法的差别

- LMS (SGD)** : 每次用 1 个样本, 便宜、可流式、可能要多次遍历。
- 批量最速下降**: 每次用全数据的精确梯度, 贵但稳定。
- RLS (递归最小二乘)** : 相当于利用“二阶信息”(卡尔曼滤波思想), **一次遍历**即可到最优, 但实现与代价更高。

5) 怎么用 (伪代码)

```

初始化 θ
for k = 1, 2, 3, ... (数据流或多轮遍历)
    读入 (x_k, y_k)
    ̂_k = θT x_k
    e_k = ̂_k - y_k
    θ = θ - η_k * e_k * x_k
end
输出 θ (或做 Polyak 平均得到更平滑的 θ)

```

实用要点

- **学习率 η_k** : 可用常数 (配合早停) 或衰减 (如 $\eta_k = \eta_0 / (\tau_0 + k)^{\kappa}$, $\kappa \in (0.5, 1]$)。
- **特征标准化**: 使各维尺度相近, 便于步长、提高稳定性。
- **偏置/截距**: 在 x_k 里加常数 1, 就能同时学偏置项。
- **多轮遍历**: 一般需要多个 epoch; 当 $\|\theta_k - \theta_{k-1}\|^2$ 很小或验证误差不再下降时停止。
- **小批量**: 把 (8.87) 的单样本换成平均多个样本的误差, 折中噪声与效率。
- **与 RLS**: 若需要超快收敛/一次遍历且能承担二阶代价, 可用 RLS; 否则 LMS 简单稳健。

小结

- LMS = 在线/随机梯度下降在**线性回归**上的具体更新:

$$\theta_{k+1} = \theta_k - \eta_k (\theta_k^\top x_k - y_k) x_k.$$

- 它用**误差 × 特征**纠偏参数, 每步只看一个样本, 所以目标值**不必单调**但总体下降。
- 可能需要**多次遍历**; 若要一次到最优, 可用基于二阶信息的 RLS。

翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

8.5.4 The perceptron algorithm Now let us consider how to fit a binary logistic regression model in an online manner. The batch gradient was given in Equation 8.5. In the online case, the weight update has the simple form

$$\theta_k = \theta_{k-1} - \eta_k g_i = \theta_{k-1} - \eta_k (\mu_i - y_i) x_i \quad (8.88)$$

where $\mu_i = p(y_i = 1 | x_i, \theta_k) = E[y_i | x_i, \theta_k]$. We see that this has exactly the same form as the LMS algorithm. Indeed, this property holds for all generalized linear models (Section 9.3).

We now consider an approximation to this algorithm. Specifically, let

$$\hat{y}_i = \arg \max_{y \in \{0, 1\}} p(y | x_i, \theta) \quad (8.89)$$

represent the most probable class label. We replace $\mu_i = p(y = 1 | x_i, \theta) = \text{sigm}(\theta^\top x_i)$ in the gradient expression with \hat{y}_i . Thus the approximate gradient becomes

$$g_i \approx (\hat{y}_i - y_i) x_i \quad (8.90)$$

It will make the algebra simpler if we assume $y \in \{-1, +1\}$ rather than $y \in \{0, 1\}$. In this case, our prediction becomes $\hat{y}^i = \text{sign}(\theta^T x_i)$ (8.91)

Then if $\hat{y}^i = -1$, we have made an error, but if $\hat{y}^i = +1$, we guessed the right label.

At each step, we update the weight vector by adding on the gradient. The key observation is that, if we predicted correctly, then $\hat{y}^i = y^i$, so the (approximate) gradient is zero and we do not change the weight vector. But if x_i is misclassified, we update the weights as follows: If $\hat{y}^i = 1$ but $y^i = -1$, then the negative gradient is $-(\hat{y}^i - y^i)x_i = -2x_i$; and if $\hat{y}^i = -1$ but $y^i = 1$, then the negative gradient is $-(\hat{y}^i - y^i)x_i = 2x_i$. We can absorb the factor of 2 into the learning rate η and just write the update, in the case of a misclassification, as

$$\theta_k = \theta_{k-1} + \eta y_i x_i \quad (8.92)$$

Since it is only the sign of the weights that matter, not the magnitude, we will set $\eta_k = 1$.

One can show that this method, known as the perceptron algorithm (Rosenblatt 1958), will converge, provided the data is linearly separable, i.e., that there exist parameters θ such that predicting with $\text{sign}(\theta^T x)$ achieves 0 error on the training set. However, if the data is not linearly separable, the algorithm will not converge, and even if it does converge, it may take a long time. There are much better ways to train logistic regression models (such as using proper SGD, without the gradient approximation, or IRLS, discussed in Section 8.3.4). However, the perceptron algorithm is historically important: it was one of the first machine learning algorithms ever derived (by Frank Rosenblatt in 1957), and was even implemented in analog hardware. In addition, the algorithm can be used to fit models where computing marginals $p(y|x, \theta)$ is more expensive than computing the MAP output, $\arg \max_y p(y|x, \theta)$; this arises in some structured-output classification problems. See Section 19.7 for details.

已思考 43s

下面先给**完整逐字翻译，再做通俗解释**（把式子、结论都讲清楚，并给出可抄的伪代码与直觉图景）。

译文

8.5.4 感知机 (perceptron) 算法

现在来考虑如何以**在线**方式拟合一个**二元逻辑回归**模型。批量情形的梯度见式 (8.5)。在在线情形下，权重的更新有如下简单形式

$$\theta_k = \theta_{k-1} - \eta_k g_i = \theta_{k-1} - \eta_k (\mu_i - y_i) x_i \quad (8.88)$$

其中 $\mu_i = p(y_i = 1 | x_i, \theta_k) = E[y_i | x_i, \theta_k]$ 。我们看到它与 LMS 算法的形式完全相同。事实上，这个性质对**广义线性模型**（见 9.3 节）都成立。

现在我们考虑这个算法的一个**近似**。具体地，让

$$\hat{y}_i = \arg \max_{y \in \{0,1\}} p(y | x_i, \theta) \quad (8.89)$$

表示**最可能的**类别标签。我们把梯度表达式里的 $\mu_i = p(y = 1 | x_i, \theta) = \text{sigm}(\theta^\top x_i)$ 用 \hat{y}_i 来替换。于是近似梯度变为

$$g_i \approx (\hat{y}_i - y_i) x_i. \quad (8.90)$$

为使代数更简洁，下面假设 $y \in \{-1, +1\}$ 而不是 $\{0, 1\}$ 。在这种情况下，我们的预测变为

$$\hat{y}_i = \text{sign}(\theta^\top x_i). \quad (8.91)$$

此时，如果 $\hat{y}_i y_i = -1$ ，说明我们**出错**；如果 $\hat{y}_i y_i = +1$ ，说明我们**猜对了**标签。

每一步，我们通过把梯度**加到**权重向量上来更新它。关键观察是：如果预测正确，则 $\hat{y}_i = y_i$ ，于是（近似）梯度为 0，我们不改变权重。但如果 x_i 被误分，权重如下更新：若 $\hat{y}_i = 1$ 而 $y_i = -1$ ，则**负梯度**为 $-(\hat{y}_i - y_i)x_i = -2x_i$ ；若 $\hat{y}_i = -1$ 而 $y_i = 1$ ，则负梯度为 $-(\hat{y}_i - y_i)x_i = 2x_i$ 。我们可以把这个 2 吸收到学习率 η 里，于是在**误分类**时的更新可写为

$$\theta_k = \theta_{k-1} + \eta_k y_i x_i. \quad (8.92)$$

由于只关心权重的**符号**而非其大小，我们取 $\eta_k = 1$ 。

可以证明，这个方法称为**感知机算法** (Rosenblatt, 1958)，在**数据线性可分**时会收敛——也就是说，存在某个 θ 使得用 $\text{sign}(\theta^\top x)$ 在训练集上达到 0 错误。然而，如果**数据不可分**，算法将**不收敛**；即便可分，也可能需要很久才收敛。训练逻辑回归有更好的方法（比如**真正的 SGD**，不做上面的梯度近似，或第 8.3.4 节的 IRLS）。不过，感知机在历史上很重要：它是最早的机器学习算法之一 (Frank Rosenblatt 1957 年提出)，甚至被做成过模拟硬件。此外，当计算边缘概率 $p(y_i | x, \theta)$ 比计算 MAP 输出 $\arg \max_y p(y | x, \theta)$ 更昂贵时（在一些**结构化输出**分类问题中会发生），该算法仍然有用。细节见 19.7 节。

通俗解释（基础且详尽）

1) 这段在讲什么？

- 从在线逻辑回归出发，先给出**正确的在线梯度更新**：
 $\theta \leftarrow \theta - \eta(\mu - y)x$, 其中 $\mu = \sigma(\theta^\top x)$ 。
- 然后做了一个**粗近似**：把“概率 μ ”直接换成“**硬判决** \hat{y} ”——这一步把“平滑的逻辑损失”变成了“只看对不对”。
- 用 $y \in \{-1, +1\}$ 表示后，得到著名的**感知机更新**：
 - 对了**：不更新；
 - 错了**： $\theta \leftarrow \theta + yx$ 。

2) 为什么感知机更新是这个方向？

- 若真实 $y = +1$ 而被判成 -1 ，说明 $\theta^\top x < 0$ 。把 θ 加上 $+x$ (乘 $y = +1$)，会**增大** $\theta^\top x$ ，推动它变正，从而下次更可能判成 $+1$ 。
- 若真实 $y = -1$ 而被判成 $+1$ ，把 θ 加上 $-x$ (乘 $y = -1$)，会**减小** $\theta^\top x$ ，下次更可能判成 -1 。
- 几何直觉： $\theta^\top x = 0$ 是超平面；误分时把超平面沿着 yx 方向**往正确一侧挪**。

3) 与 LMS / 逻辑回归 SGD 的关系

- LMS/逻辑回归 SGD**：用的是**软误差**（平方或逻辑损失）的**真实梯度**，更新是 $(\hat{y} - y)x$ 或 $(\mu - y)x$ 。
- 感知机**：把 μ 用 $\hat{y} = \text{sign}(\theta^\top x)$ 近似 → 只在**错时更新**，且步长不依赖误差大小。
- 因此感知机是一个“**更粗**的、只看对错的“SGD 版本”，计算更便宜但也更“莽”。

4) 收敛性与局限

- 可分时**：有**感知机收敛定理**，一定会在有限步内找到一个**零训练误差**的分离超平面（并且与间隔有关的“错误次数上界”可被证明）。
- 不可分时**：可能**来回震荡**，不收敛。
- 即使可分，找到的超平面不一定“最优”（比如**间隔不大**）；现代更喜欢用**逻辑回归/最大间隔等方法**。

5) 为什么步长可以设 1？

- 感知机只看符号**：把 θ 乘以任意正常数，分类结果都不变。
- 因此把常数因子（比如前文的 2）吸进学习率，再把学习率取 1，不影响分类边界（只是影响迭代速度/路径）。

6) 何时还会用感知机?

- 当计算 $p(y | x, \theta)$ (需要求和/积分) 很贵, 但拿到 $\arg \max_y p(y | x, \theta)$ 相对容易时——例如**结构化预测** (序列标注、配对、匹配等) 里, 可以用“**结构化感知机**”训练 (见 19.7 节)。

7) 可直接套用的伪代码

```

输入: 训练流  $\{(x_i, y_i)\}$ ,  $y_i \in \{-1, +1\}$ 
初始化  $\theta = 0$ 
for i = 1, 2, 3, ... // 单次或多次遍历
    if sign( $\theta^T x_i$ )  $\neq y_i$  then // 误分才更新
         $\theta \leftarrow \theta + y_i x_i$ 
    end if
end for
输出  $\theta$ 

```

若用 {0,1} 标签, 把判断改成 $\hat{y} = I[\theta^T x \geq 0]$, 更新 $\theta \leftarrow \theta + (y - \hat{y})x$ 。

小结

- 在线逻辑回归的**正确更新**: $\theta \leftarrow \theta - \eta(\mu - y)x$, 与 LMS 同形。
 - 感知机**用“硬判决”近似 μ , 于是“对了不动、错了加 yx ”:
- 若误分: $\theta \leftarrow \theta + yx$
- 可分则收敛; 不可分则可能不收敛。现代更推荐用**真正的 SGD / IRLS**训练逻辑回归; 感知机主要有历史价值与在**结构化预测**中的实用价值。

翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

8.5.5 A Bayesian view

Another approach to online learning is to adopt a Bayesian view. This is conceptually quite simple: we just apply Bayes rule recursively:

$$p(\theta|D_{1:k}) \propto p(D_k|\theta)p(\theta|D_{1:k-1}) \quad (8.93)$$

This has the obvious advantage of returning a posterior instead of just a point estimate. It also allows for the

online adaptation of hyper-parameters, which is important since cross-validation cannot be used in an online setting. Finally, it has the (less obvious) advantage that it can be quicker than SGD. To see why, note that by modeling the posterior variance of each parameter in addition to its mean, we effectively associate a different learning rate for each parameter (de Freitas et al. 2000), which is a simple way to model the curvature of the space. These variances can then be adapted using the usual rules of probability theory. By contrast, getting second-order optimization methods to work online is more tricky (see e.g., (Schraudolph et al. 2007; Sunehag et al. 2009; Bordes et al. 2009, 2010)). As a simple example, in Section 18.2.3 we show how to use the Kalman filter to fit a linear regression model online. Unlike the LMS algorithm, this converges to the optimal (offline) answer in a single pass over the data. An extension which can learn a robust non-linear regression model in an online fashion is described in (Ting et al. 2010). For the GLM case, we can use an assumed density filter (Section 18.5.3), where we approximate the posterior by a Gaussian with a diagonal covariance; the variance terms serve as a per-parameter step-size. See Section 18.5.3.2 for details. Another approach is to use particle filtering (Section 23.5); this was used in (Andrieu et al. 2000) for sequentially learning a kernelized linear/logistic regression model.

已思考 27s

下面先给**完整逐字翻译**, 再做**通俗解释** (把符号、直觉与做法讲清楚; 末尾给出可抄的“在线贝叶斯”小模板)。

译文

8.5.5 一种贝叶斯视角

另一种进行在线学习的方法, 是采取**贝叶斯**的视角。这在概念上非常简单: 我们只需**递归地应用贝叶斯公式**:

$$p(\theta | D_{1:k}) \propto p(D_k | \theta) p(\theta | D_{1:k-1}) \quad (8.93)$$

这样做的一个显而易见的优点是: 它返回的是**后验分布**, 而不仅仅是一个点估计。它还允许对**超参数**进行在线自适应, 这很重要, 因为在在线情形中**不能使用交叉验证**。最后, 它

还有一个不那么显而易见的好处：它可能比 SGD 更快。原因在于：除了对每个参数的均值进行建模之外，我们还同时建模其**后验方差**，等价于为每个参数分配了**不同的学习率**（de Freitas 等, 2000），这是一种刻画空间曲率的简单方式。随后，这些方差可以用通常的概率论规则进行自适应更新。与此相对，要把**二阶优化方法**做成在线版本则更棘手（见如 Schraudolph 等, 2007; Sunehag 等, 2009; Bordes 等, 2009、2010）。

一个简单的例子是在第 18.2.3 节中，我们展示了如何用**卡尔曼滤波器**在线地拟合线性回归模型。与 LMS 算法不同，它在**单次遍历**数据时就会收敛到离线（全量）最优解。一个能够在线学习鲁棒**非线性回归**模型的扩展见 Ting 等 (2010)。

对 GLM 情形，我们可以使用**假设密度滤波**（第 18.5.3 节）：用具有**对角协方差**的高斯来近似后验；这些**方差项就充当“逐参数的步长”**。细节见第 18.5.3.2 节。另一条途径是使用**粒子滤波**（第 23.5 节）；Andrieu 等 (2000) 用它来顺序学习核化的线性/逻辑回归模型。

通俗解释（基础且详尽）

1) 递归贝叶斯 = 在线更新

- 记 $D_{1:k} = \{D_1, \dots, D_k\}$ 为到第 k 个时刻的全部数据（一次来一个）。
- **先验 \rightarrow 后验 \rightarrow 新先验**：拿上一步的后验 $p(\theta | D_{1:k-1})$ 当作本步的先验，与新样本的似然 $p(D_k | \theta)$ 相乘并归一化，就得到新的后验 $p(\theta | D_{1:k})$ 。这就是式 (8.93)。
- 输出不是一个点 $\hat{\theta}$ ，而是一整个**分布**（均值 + 方差），因此自带**不确定性信息**。

2) 为什么可能比 SGD“更快”？

- 在贝叶斯更新里，我们不仅维护参数的**均值** m ，还维护其**协方差/方差** V 。
- 更新公式里会自然出现“**按参数自适应的步长**”，例如：
 - 在**在线线性-高斯模型**（卡尔曼滤波）：步长就是**卡尔曼增益**，由当前不确定性 V 和观测噪声自动算出；不需要手调 η 。
 - 这等价于在做一种**二阶信息**（曲率）感知的更新，但计算与实现更稳当。
- 因为“步长自动选得合理”，往往**比手工调的 SGD 收敛更快/更稳**。

3) 与二阶在线优化的对比

- 直接把牛顿法、BFGS 等做成在线版要处理很多技术细节（稳定性、存储、噪声），文献提示“更棘手”。
- **贝叶斯做法**通过维护 V （或其对角近似），顺带就有了“二阶味道”的自适应步长。

4) 几个常用实例

- **线性回归 + 卡尔曼滤波** (第 18.2.3) :
 - 状态 = 参数 θ ; 测量 = 新样本误差。
 - 单次遍历就能达到**离线最优** (而 LMS/SGD 往往要多轮)。
- **GLM + 假设密度滤波 (ADF)** :
 - 真实后验通常非高斯, 于是把它**近似成**“均值 m + 对角方差 v ”的高斯。
 - 每来一个样本, 用近似规则更新 m, v ; 其中 v 起到“每个参数的学习率”的作用 (方差大 → 步长大, 反之小)。
- **粒子滤波**:
 - 用一组带权样本 (粒子) 来近似 $p(\theta | D_{1:k})$, 适合**非线性、非高斯**的模型。
 - 已用于**核化的线性/逻辑回归的顺序学习** (Andrieu 等, 2000)。

5) 为什么“在线调超参”很重要?

- 在线场景**不能做 K 折交叉验证** (未来数据还没来)。
- 贝叶斯框架可以给超参 (如噪声方差、正则强度) 设先验, 并和参数一起**联合更新** (或用边际似然/证据递推近似), 从而随时间**自适应**。

6) 何时选贝叶斯在线 vs SGD?

- **更倾向贝叶斯在线**:
 - 需要**不确定性** (置信区间、风险控制、探索-利用) ;
 - 想要**自动步长/每参数自适应**而少调参;
 - 希望**一遍数据**就接近最优 (如传感器流、低延迟应用)。
- **更倾向 SGD**:
 - 模型十分大、近似推断代价高;
 - 只要求点估计, 且可容忍多轮遍历与手调学习率。

在线贝叶斯的小模板 (你可以直接照这个做)

初始化: 先验 $p(\theta) = \text{N}(\mu_0, \Sigma_0)$ // 或其它可解析/可近似的先验

```
for k = 1, 2, 3, ... // 数据流  $D_k$  逐个到达
    // 1) 似然:  $p(D_k | \theta)$ 
    // 2) 递归后验更新 (精确或近似):
        //   - 线性高斯: 用卡尔曼更新  $m, \Sigma$ 
        //   - GLM: 用假设密度滤波/拉普拉斯一步近似更新  $m, \text{diag}(\Sigma)$ 
        //   - 更复杂: 用粒子滤波更新  $\{\theta^s, w^s\}$ 
    // 3) 需要预测时, 对  $\theta$  的后验做积分 (可用 MC 或解析近似)
```

end

输出：后验 $p(\theta | D_{\{1:k\}})$ 的均值/方差（以及预测不确定性）

小结

- 在线贝叶斯的核心是一句式 (8.93): **新后验 \propto 新似然 \times 旧后验。**
- 它输出**分布**而非点估计，能**在线调超参**，并通过维护每个参数的**方差**自动产生“**逐参数步长**”，因此在很多场景里**更快/更稳于 SGD**。
- 典型实现：线性回归用**卡尔曼滤波**，GLM 用**假设密度滤波**，复杂/非高斯情况用**粒子滤波**。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.6 Generative vs discriminative classifiers

In Section 4.2.2, we showed that the posterior over class labels induced by Gaussian discriminant analysis (GDA) has exactly the same form as logistic regression, namely $p(y=1|x) = \text{sigm}(w^T x)$. The decision boundary is therefore a linear function of x in both cases. Note, however, that many generative models can give rise to a logistic regression posterior, e.g., if each class-conditional density is Poisson, $p(x|y=c) = \text{Poi}(x|\lambda_c)$. So the assumptions made by GDA are much stronger than the assumptions made by logistic regression. A further difference between these models is the way they are trained. When fitting a discriminative model, we usually maximize the conditional log likelihood $\sum_i \log p(y_i|x_i, \theta)$, whereas when fitting a generative model, we usually maximize the joint log likelihood, $\sum_i \log p(y_i, x_i|\theta)$. It is clear that these can, in general, give different results (see Exercise 4.20). When the Gaussian assumptions made by GDA are correct, the model will need less training data than logistic regression to achieve a certain level of performance, but if the Gaussian assumptions are incorrect, logistic regression will do better (Ng and Jordan 2002). This is because discriminative models do not need to model the

distribution of the features. This is illustrated in Figure 8.10. We see that the class conditional densities are rather complex; in particular, $p(x|y=1)$ is a multimodal distribution, which might be hard to estimate. However, the class posterior, $p(y=c|x)$, is a simple sigmoidal function, centered on the threshold value of 0.55. This suggests that, in general, discriminative methods will be more accurate, since their “job” is in some sense easier. However, accuracy is not the only important factor when choosing a method. Below we discuss some other advantages and disadvantages of each approach.

下面先给**完整逐字翻译**，再做**通俗解释**（把概念、式子和结论都讲清楚，适合数学基础较弱时阅读）。

译文

8.6 生成式分类器 vs 判别式分类器

在第 4.2.2 节中，我们展示过：由**高斯判别分析** (GDA) 所诱导的**类别后验与逻辑回归完全同形**，具体就是

$$p(y=1 | x) = \text{sigm}(w^\top x).$$

因此，在这两种方法中，**决策边界**都是 x 的线性函数。注意，许多**生成式模型**都可能得到一个**逻辑回归形状的后验**；例如，当每个类条件密度都是泊松分布时， $p(x | y=c) = \text{Poi}(x | \lambda_c)$ 也会得到这种形式。所以，GDA 所做出的假设**比逻辑回归所做的假设要强得多**。

这两类模型的另一个差别在于**训练方式**。拟合判别式模型时，我们通常**最大化条件对数似然**

$$\sum_{i=1}^N \log p(y_i | x_i, \theta);$$

而拟合生成式模型时，我们通常**最大化联合对数似然**

$$\sum_{i=1}^N \log p(y_i, x_i | \theta).$$

显然，一般而言这两种目标会给出**不同的结果**（见练习 4.20）。当 GDA 的高斯假设**正确**时，要达到某一性能水平，它所需的训练数据**少于**逻辑回归；但若高斯假设**不正确**，逻辑回归会更好 (Ng 和 Jordan, 2002)。原因是判别式模型**不需要去建模特征的分布**。图 8.10 做了说明：类条件密度很复杂；尤其是 $p(x | y=1)$ 是**多峰分布**，可能很难估计。然而类别后验 $p(y=c | x)$ 却是一个**简单的 S 形函数**，中心在阈值 0.55 附近。这提示我们：一般来说，**判别式方法会更准确**，因为它们的“任务”某种意义上更简单。

不过，选择方法时**准确率并不是唯一因素**。下面我们将讨论每种方法的其他优缺点。

通俗解释（基础且详尽）

1) 先分清两类方法在“建模什么”

- **生成式** (GDA 等)：为每个类建立**特征的分布** $p(x | y)$, 再配上先验 $p(y)$, 最后用贝叶斯公式得到

$$p(y | x) \propto p(x | y) p(y).$$

例如 GDA 假设 $p(x | y = c)$ 是**高斯**。
- **判别式** (逻辑回归等)：直接建模 $p(y | x)$ 或者直接学一个判别函数 $h(x)$ (如最大间隔), 不需要写出 $p(x | y)$ 。

2) 为什么两者都会给出“线性边界”？

- 只要 $p(y = 1 | x) = \text{sigm}(w^\top x)$, 令其等于 0.5 得到 $w^\top x = 0$ ——这就是**线性超平面**。
- GDA 在“同协方差高斯”的假设下可推出这一点；逻辑回归是**直接假设**这一形式。

3) 谁的假设更强？

- **GDA 更强**: 必须假设 $p(x | y)$ 的具体形状 (如高斯、泊松等)。
- **逻辑回归更弱**: 只假设后验的**对数几率**是 $w^\top x$ 的线性函数, 对 x 的分布本身**不关心**。

4) 训练目标为什么不同、会带来什么差别？

- **判别式最大化** $\sum \log p(y_i | x_i)$: 只关心“**给定 x, y 的概率**”。
- **生成式最大化** $\sum \log p(y_i, x_i)$: 既要把类别分好, 又要把**特征的概率密度**也拟合好。
- 所以当生成式的分布假设**正确**时, 它“信息利用更充分”, **样本效率更高** (少数据也能好)；
 但假设一旦**偏了**, 就会在拟合 $p(x | y)$ 上“浪费精力”甚至被误导, **判别式**反而更稳健。

5) 为什么图 8.10 支持判别式更容易“准确”？

- 真正的 $p(x | y)$ 可能很复杂 (多峰、非对称等), 难估计;
- **但我们要的只是** $p(y | x)$: 往往是**平滑的 S 形**、只在阈值附近变化。
- “难题→简单题”的映射: 直接学后验 (判别式) 更容易把“关键的边界”学准。

6) 何时选生成式、何时选判别式？

- **生成式的优势 (除准确率外)**
 - 能**生成样本、补全缺失特征** (可把缺失维积分掉), 自然支持**半监督/先验知识**;

- 小数据时若模型假设靠谱，往往比判别式更好（样本效率）；
- 需要联合建模 x, y 的问题（如异常检测、仿真）。
- 判别式的优势
 - 目标对准“分类/预测”，通常精度高、鲁棒；
 - 不必估计 $p(x | y)$ ，特征很多、分布复杂时更省心；
 - 优化常是凸的（逻辑回归），训练稳定。

7) 一句话记忆

- GDA：假设强、若对就省数据，错了就受伤；
 - Logistic：假设弱、只学边界，通常更准但可能更耗数据。
 - 两者在某些设定下后验同形（sigmoid），但“到这个式子”的路线不同，带来样本效率与鲁棒性上的差别。
-

小结

- 两者都可能得到 $p(y = 1 | x) = \text{sigm}(\mathbf{w}^\top \mathbf{x})$ （线性边界），但 GDA 的前置假设更强。
- 训练目标不同：判别式最大化条件似然；生成式最大化联合似然 → 结果一般不同。
- 样本效率 vs. 假设鲁棒性：假设正确 → GDA 更省数据；假设不准 → 逻辑回归更好。
- 选法不只看准确率，还要看是否需要生成能力、处理缺失、样本量大小、以及是否有可信的分布假设。



已上传的图片

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.10 The class-conditional densities $p(x|y = c)$ (left) may be more complex than the class posteriors $p(y = c|x)$

(right). Based on Figure 1.27 of (Bishop 2006a). Figure generated by generativeVsDiscrim.

8.6 Generative vs discriminative classifiers

In Section 4.2.2, we showed that the posterior over class labels induced by Gaussian discriminant analysis (GDA) has exactly the same form as logistic regression, namely $p(y=1|x) = \text{sigm}(w^T x)$. The decision boundary is therefore a linear function of x in both cases. Note, however, that many generative models can give rise to a logistic regression posterior, e.g., if each class-conditional density is Poisson, $p(x|y=c) = \text{Poi}(x|\lambda_c)$. So the assumptions made by GDA are much stronger than the assumptions made by logistic regression. A further difference between these models is the way they are trained. When fitting a discriminative model, we usually maximize the conditional log likelihood $\sum_{i=1}^N \log p(y_i|x_i, \theta)$, whereas when fitting a generative model, we usually maximize the joint log likelihood, $\sum_{i=1}^N \log p(y_i, x_i|\theta)$. It is clear that these can, in general, give different results (see Exercise 4.20). When the Gaussian assumptions made by GDA are correct, the model will need less training data than logistic regression to achieve a certain level of performance, but if the Gaussian assumptions are incorrect, logistic regression will do better (Ng and Jordan 2002). This is because discriminative models do not need to model the distribution of the features. This is illustrated in Figure 8.10. We see that the class conditional densities are rather complex; in particular, $p(x|y=1)$ is a multimodal distribution, which might be hard to estimate. However, the class posterior, $p(y=c|x)$, is a simple sigmoidal function, centered on the threshold value of 0.55. This suggests that, in general, discriminative methods will be more accurate, since their "job" is in some sense easier. However, accuracy is not the only important factor when choosing a method. Below we discuss some other advantages and disadvantages of each approach.

下面先给**完整逐字翻译**，再做**通俗解释**（把图里细节、两类方法的区别和何时用哪种都讲清楚）。

译文

图 8.10 左图：类条件密度 $p(x | y = c)$ 可能比右图的类后验 $p(y = c | x)$ 更复杂。基于 (Bishop 2006a) 图 1.27。图由 *generativeVsDiscrim* 生成。

8.6 生成式分类器 vs 判别式分类器

在第 4.2.2 节中，我们展示过：由高斯判别分析（GDA）所诱导的类别后验与逻辑回归完全同形，即

$$p(y = 1 | x) = \text{sigm}(w^\top x).$$

因此在两种方法中，决策边界都是 x 的线性函数。需要注意的是，许多生成式模型都能产生一个逻辑回归形状的后验；例如若每个类条件密度都是泊松分布，

$$p(x | y = c) = \text{Poi}(x | \lambda_c),$$

也会得到这种形式。所以，GDA 所作的假设远强于逻辑回归的假设。

这两类模型的另一个区别在于训练方式。拟合判别式模型时，我们通常最大化条件对数似然

$$\sum_{i=1}^N \log p(y_i | x_i, \theta),$$

而拟合生成式模型时，我们通常最大化联合对数似然

$$\sum_{i=1}^N \log p(y_i, x_i | \theta).$$

显然，二者一般会得到不同的结果（见练习 4.20）。当 GDA 的高斯假设正确时，要达到某一性能水平，它所需的训练数据少于逻辑回归；但如果高斯假设不正确，逻辑回归会更好 (Ng 和 Jordan, 2002)。这是因为判别式模型不需要去建模特征的分布。图 8.10 做了说明：我们看到类条件密度相当复杂；尤其 $p(x | y = 1)$ 是多峰分布，可能很难估计。然而类后验 $p(y = c | x)$ 却是一个简单的 S 形函数，其中心在阈值 0.55 附近。这表明，通常判别式方法会更准确，因为它们的“任务”在某种意义上更简单。不过，选择方法时准确率并非唯一因素。下面我们将讨论两种途径各自的其他优缺点。

通俗解释（基础且详尽）

1) 读图 8.10：为什么“左难右易”

- **左图画的是每个类别里 x 的分布 $p(x | y = c)$:**
黑实线 $p(x | y = 1)$ 明显多峰 (好几座小山) , 红虚线 $p(x | y = 2)$ 偏在右侧。要把这两条曲线都拟合准确, 模型得很复杂 (混合分布、核密度等) 。
- **右图画的是后验 $p(y = c | x)$:**
黑线 $p(y = 1 | x)$ 与红线 $p(y = 2 | x)$ 一上一下, 整体就是平滑的 S 形; 绿色竖线是两类后验相等 (约 0.55) 的位置——**决策阈值**。
也就是说, 虽然左边各类的“形状”很复杂, 但右边“我究竟属于哪类”的概率却非常简单。

结论: 若你的目标只是“分哪个类” (学 $p(y | x)$) , 通常**直接学判别式**更省心。

2) 两类方法到底在“建什么”

- **生成式:** 先建 $p(x | y)$ (每类里特征怎样分布) 再配合 $p(y)$, 用贝叶斯公式得到 $p(y | x)$ 。例: GDA 假设高斯, 朴素贝叶斯假设条件独立。
- **判别式:** 直接建 $p(y | x)$ (如逻辑回归) , 或直接学一个决策函数 (SVM 的间隔) 。

3) 训练目标为什么不一样、影响何在

- 判别式最大化 $\sum \log p(y_i | x_i)$: 只对**分类**负责。
- 生成式最大化 $\sum \log p(y_i, x_i)$: 既要分类, 还要把 x 的**分布**也学好。
- 结果:
 - **假设真** (比如各类确实高斯) \rightarrow 生成式**样本效率高** (少数据就能好) 。
 - **假设偏** (真实分布多峰/长尾/非高斯) \rightarrow 判别式**更鲁棒、通常更准**, 因为它不被 $p(x | y)$ 的错设牵着走。

4) 为什么两者都可能给线性边界

- 当后验形如 $\text{sigm}(w^\top x)$ 时, 令其等于 0.5 得到 $w^\top x = 0$ ——**线性超平面**。
- GDA 在“同协方差高斯”等条件下可以推导出这个形式; 逻辑回归则是直接假设并拟合它。

5) 选型建议 (除准确率外再看什么)

- **选择生成式**如果:
 - 你需要**生成能力** (能采样/填补缺失) ;
 - 数据很少且你对 $p(x | y)$ 的假设**可信**;
 - 需要自然地融入**先验知识**、做**半监督/异常检测**等。
- **选择判别式**如果:
 - 目标是**预测/分类**, 特征分布复杂或维度高;

- 你更在意**稳健与精度**, 不想费力建 $p(x | y)$;
- 期望**凸优化**、训练稳定 (逻辑回归就是) 。

一句话记忆

- **GDA**: 假设强, 真则**省数据**, 假则吃亏;
- **逻辑回归**: 假设弱, 通常**更准更稳**;
- 左图复杂是 $p(x | y)$ 的难, 右图简单是 $p(y | x)$ 的易——这就是“**判别式工作更容易**”的直觉来源。



翻译并解释以下内容, 翻译不能有遗漏, 解释请基础且详尽, 因为我的数学基础不太好。

Figure 8.9 (a) Multinomial logistic regression for 5 classes in the original feature space. (b) After basis function expansion, using RBF kernels with a bandwidth of 1, and using all the data points as centers. Figure generated by logregMultinomKernelDemo.

8.6.1 Pros and cons of each approach

- Easy to fit? As we have seen, it is usually very easy to fit generative classifiers. For example, in Sections 3.5.1.1 and 4.2.4, we show that we can fit a naive Bayes model and an LDA model by simple counting and averaging. By contrast, logistic regression requires solving a convex optimization problem (see Section 8.3.4 for the details), which is much slower.
- Fit classes separately? In a generative classifier, we estimate the parameters of each class conditional density independently, so we do not have to retrain the model when we add more classes. In contrast, in discriminative

models, all the parameters interact, so the whole model must be retrained if we add a new class. (This is also the case if we train a generative model to maximize a discriminative objective Salojarvi et al. (2005).)

- Handle missing features easily? Sometimes some of the inputs (components of x) are not observed. In a generative classifier, there is a simple method for dealing with this, as we discuss in Section 8.6.2. However, in a discriminative classifier, there is no principled solution to this problem, since the model assumes that x is always available to be conditioned on (although see (Marlin 2008) for some heuristic approaches).
- Can handle unlabeled training data? There is much interest in semi-supervised learning, which uses unlabeled data to help solve a supervised task. This is fairly easy to do using generative models (see e.g., (Lasserre et al. 2006; Liang et al. 2007)), but is much harder to do with discriminative models.
- Symmetric in inputs and outputs? We can run a generative model “backwards”, and infer probable inputs given the output by computing $p(x|y)$. This is not possible with a discriminative model. The reason is that a generative model defines a joint distribution on x and y , and hence treats both inputs and outputs symmetrically.
- Can handle feature preprocessing? A big advantage of discriminative methods is that they allow us to preprocess the input in arbitrary ways, e.g., we can replace x with $\varphi(x)$, which could be some basis function expansion, as illustrated in Figure 8.9. It is often hard to define a generative model on such pre-processed data, since the new features are correlated in complex ways.
- Well-calibrated probabilities? Some generative models, such as naive Bayes, make strong independence assumptions which are often not valid. This can result in very extreme posterior class probabilities (very near 0 or 1). Discriminative models, such as logistic regression, are usually better calibrated in terms of their probability estimates.

We see that there are arguments for and against both kinds of models. It is therefore useful to have both kinds in your “toolbox”. See Table 8.1 for a summary of the classification and regression techniques we cover in this book.

下面先给**完整逐字翻译**, 再做**通俗解释** (把图里细节和每条优缺点都讲清楚; 数学符号很少, 读起来不吃力)。

译文

图 8.9

- (a) 在原始特征空间中做 5 类的**多项式 (多类)**逻辑回归。
- (b) 先做**基函数展开**: 使用带宽为 1 的 RBF 核, 并把**所有训练样本都当作核中心**。图由 `logregMultinomKernelDemo` 生成。

8.6.1 各方法的优缺点

- **好拟合吗?** 正如我们所见, **生成式分类器**通常很容易拟合。例如在 3.5.1.1 和 4.2.4 节里, 朴素贝叶斯和 LDA 只需**计数与求均值/协方差**就能训练。相比之下, **逻辑回归**需要解一个**凸优化问题** (细节见 8.3.4), 速度要慢得多。
- **能否把每个类别分开独立拟合?** 在**生成式分类器**里, 我们分别估计每个**类条件分布**的参数, 因而**添加新类别时无需重训旧类别**。反之在**判别式模型**中, 所有参数是**耦合**的, 一旦加新类, **整个模型都得重训**。 (即使把生成式模型拿来、却用判别式目标训练, 也会有同样问题; 见 Salojarvi 等, 2005。)
- **能否轻松处理缺失特征?** 有时输入 x 的某些分量**没有观测到**。在**生成式分类器**中, 这个问题有一个**简单做法** (见 8.6.2 节)。但在**判别式分类器**中没有一个“原则性”的通用解, 因为判别式模型默认**总能拿到完整的 x** (不过 Marlin, 2008 给出了一些启发式方法)。
- **能否利用无标签训练数据?** 半监督学习 (用无标签数据辅助有监督任务) 很受关注。对**生成式模型**来说比较容易 (如 Lasserre 2006; Liang 2007), 但对**判别式模型**则困难得多。
- **在输入和输出上是否对称?** **生成式模型**可以“反向运行”, 通过计算 $p(x | y)$ 来推断在给定输出时可能的输入; **判别式模型**做不到。原因是生成式模型定义的是 x, y 的**联合分布**, 因此对输入与输出是**对称**处理的。
- **能否方便做特征预处理?** **判别式方法**一大优点是允许我们**任意预处理输入**, 例如把 x 换成 $\phi(x)$ (某种基函数展开), 如图 8.9 所示。而在这样的预处理后数据上去**定义生成式模型**常常很难, 因为新特征之间可能出现**复杂相关**。
- **概率是否校准得好?** 一些**生成式模型** (如**朴素贝叶斯**) 做了很强的独立性假设, 现实中往往不成立, 结果会得到**很极端的后验概率** (接近 0 或 1)。**判别式模型** (如**逻辑回归**) 在“概率估计的校准性”方面通常更好。

可以看到，这两类模型都有支持和反对的理由。因此，工具箱里同时备着两类方法是有用的。本书覆盖的分类与回归技术的总结见表 8.1。

通俗解释（基础且详尽）

图 8.9 在讲什么？

- (a) **线性多类逻辑回归**：在原始二维平面上，每个类别的决策边界都是直线，所以分区都是**多边形**，拐角直来直去。
- (b) **核-RBF 多类逻辑回归**：先把原始点 x 通过很多**RBF 基函数**映射到高维后，再做逻辑回归。
 - RBF（径向基）形如 $\phi_j(x) = \exp(-\|x - c_j\|^2/(2\sigma^2))$ 。这里把每个训练样本当作一个中心 c_j ，带宽 (σ) 取 1。
 - 结果：模型能刻画**弯曲/不规则**的边界，局部细节被捕捉到，分类区域更贴合数据。

直观记忆：**线性** → 直线边；**核+基函数展开** → 弯弯曲曲的边。

每条“优缺点”用白话补充

1. 训练是否容易

- 生成式（朴素贝叶斯/LDA）：像做统计表，**数一数、求一求平均/协方差**就行。
- 判别式（逻辑回归）：要解优化问题（虽然是凸的，但要**迭代**），**慢一些**。

2. 加新类别要不要重训

- 生成式：每类各自估参数，**不动旧类就能加新类**。
- 判别式：参数耦合，**加新类要整体重训**。

3. 缺失特征

- 生成式：有 $p(x | y)$ 的模型，可以把**缺失维积分掉/求和**，自然解决。
- 判别式：只有 $p(y | x)$ ，如果 x 缺维，**没得条件** → 只能靠启发式（插值、均值填充等），**不原则**。

4. 无标签数据

- 生成式：有 $p(x)$ 或 $p(x | y)$ 的说法，**无标签也能学到结构**，再辅助分类。
- 判别式：只关心 $p(y | x)$ ，**无标签数据帮不上忙**（需要额外技巧才行）。

5. 输入输出对称

- 生成式：有联合分布 $p(x, y)$ ，所以既能算 $p(y | x)$ ，也能算 $p(x | y)$ （**反推输入**）。

- 判别式：只有条件分布，**反推做不了**。

6. 特征预处理的自由度

- 判别式：随便造特征 $\phi(x)$ (多项式、RBF、文本特征等)，**直接喂给分类器**。
- 生成式：要给 $\phi(x)$ 写出 $p(\phi(x) | y)$ ，常常**很难** (相关性复杂)。

7. 概率校准

- 朴素贝叶斯常因独立性假设不真 → 概率会**过度自信** (接近 0/1)。
- 逻辑回归的概率通常**更接近真实频率** (更“稳”)。

什么时候用谁？（一眼能记）

- **生成式**：样本少、想处理缺失/无标签、需要生成/反推输入、有靠谱的分布假设。
 - **判别式**：只想要**好预测**、特征复杂/高维、愿意做特征工程 (如图 8.9(b) 的核映射)、**要概率校准好**。
-

小结

- 线性多类逻辑回归给**直线**边界；加上核的**基函数展开**可得到**非线性**边界。
- 生成式 vs 判别式：训练方便/可利用无标签/能处理缺失 (生成式) vs 预处理自由/概率更准/通常更高精度 (判别式)。
- 两类方法各有用武之地，**工具箱里都要有**。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.6.2 Dealing with missing data

Sometimes some of the inputs (components of x) are not observed; this could be due to a sensor failure, or a failure to complete an entry in a survey, etc. This is called the missing data problem (Little. and Rubin 1987). The ability to handle missing data in a principled way is one of the biggest advantages of generative models. To formalize our assumptions, we can associate a binary response variable $r_i \in \{0, 1\}$, that specifies whether each value x_i is observed or not. The joint model has the form $p(x_i, r_i | \theta, \varphi) = p(r_i | x_i, \varphi)p(x_i | \theta)$, where φ are the parameters controlling whether the item is observed or not. If we

assume $p(r_i|xi, \phi) = p(r_i|\phi)$, we say the data is missing completely at random or MCAR. If we assume $p(r_i|xi, \phi) = p(r_i|x_{oi}, \phi)$, where x_{oi} is the observed part of xi , we say the data is missing at random or MAR. If neither of these assumptions hold, we say the data is not missing at random or NMAR. In this case, we have to model the missing data mechanism, since the pattern of missingness is informative about the values of the missing data and the corresponding parameters. This is the case in most collaborative filtering problems, for example. See e.g., (Marlin 2008) for further discussion. We will henceforth assume the data is MAR.

When dealing with missing data, it is helpful to distinguish the cases when there is missingness only at test time (so the training data is complete data), from the harder case when there is missingness also at training time. We will discuss these two cases below. Note that the class label is always missing at test time, by definition; if the class label is also sometimes missing at training time, the problem is called semi-supervised learning.

8.6.2.1 Missing data at test time

In a generative classifier, we can handle features that are MAR by marginalizing them out. For example, if we are missing the value of x_1 , we can compute

$$p(y = c|x_2:D, \theta) \propto p(y = c|\theta)p(x_2:D|y = c, \theta) \quad (8.94)$$

$$= p(y = c|\theta)x_1 p(x_1, x_2:D|y = c, \theta) \quad (8.95)$$

If we make the naive Bayes assumption, the marginalization can be performed as follows:

$$x_1 p(x_1, x_2:D|y = c, \theta) = x_1 p(x_1|\theta_1c) \prod_{j=2}^D p(x_j|\theta_jc) = \prod_{j=2}^D p(x_j|\theta_jc) \quad (8.96)$$

where we exploited the fact that $x_1 p(x_1|y = c, \theta) = 1$.

Hence in a naive Bayes classifier, we can simply ignore missing features at test time. Similarly, in discriminant analysis, no matter what regularization method was used to estimate the parameters, we can always analytically marginalize out the missing variables (see Section 4.3):

$$p(x_2:D|y = c, \theta) = N(x_2:D|\mu_c, 2:D, \Sigma_c, 2:D, 2:D) \quad (8.97)$$

8.6.2.2 Missing data at training time

Missing data at training time is harder to deal with. In particular, computing the MLE or MAP estimate is no longer a simple optimization problem, for reasons discussed in Section 11.3.2. However, soon we will study a variety of more sophisticated algorithms (such as EM

algorithm, in Section 11.4) for finding approximate ML or MAP estimates in such cases.

下面先给**完整逐字翻译**（不遗漏），再做**通俗解释**（把每个概念、公式与做法讲清楚；尽量用直白比喻）。

译文

8.6.2 处理缺失数据

有时输入（向量 x 的某些分量）并没有被观测到；这可能是传感器失灵，或问卷里没有填写等。这被称为**缺失数据问题**（Little 和 Rubin, 1987）。能够以**有原则**的方式处理缺失数据，是**生成式模型**最大的优势之一。为了形式化我们的假设，我们给每个数值 x_i 关联一个二元响应变量 $r_i \in \{0, 1\}$ ，它指明该值是否被观测到。联合模型写成

$$p(x_i, r_i | \theta, \phi) = p(r_i | x_i, \phi) p(x_i | \theta),$$

其中 ϕ 是控制“该项是否被观测到”的参数。若我们假设 $p(r_i | x_i, \phi) = p(r_i | \phi)$ ，就说数据是**完全随机缺失**（MCAR）。若我们假设

$p(r_i | x_i, \phi) = p(r_i | x_i^o, \phi)$ ，其中 x_i^o 是 x_i 的**已观测部分**，就说数据是**随机缺失**（MAR）。如果这两种假设都不成立，就说数据是**非随机缺失**（NMAR）。在这种情况下，我们必须对**缺失机制**建模，因为缺失的模式本身对缺失值以及相关参数是有信息的。多数协同过滤问题就是这种情形。更多讨论见 Marlin (2008)。以下我们假设数据为 MAR。

处理缺失数据时，最好区分两种情况：**只在测试时**（test time）有缺失（因此训练集是完整的），以及**训练时也有缺失**（更难）。下面分别讨论这两种情形。注意：按照定义，**测试时类别标签总是未知的**；如果训练时偶尔也缺标签，这个问题叫**半监督学习**。

8.6.2.1 测试时的缺失

在一个**生成式分类器**里，对 MAR 的特征，我们可以把它们**积分消掉（边缘化）**。例如，若缺少 x_1 的值，可以计算

$$p(y = c | x_{2:D}, \theta) \propto p(y = c | \theta) p(x_{2:D} | y = c, \theta) \quad (8.94)$$

$$= p(y = c | \theta) \int p(x_1, x_{2:D} | y = c, \theta) dx_1. \quad (8.95)$$

若作**朴素贝叶斯假设**，边缘化可这样做：

$$\int p(x_1, x_{2:D} \mid y = c, \theta) dx_1 = \int_{j=2}^D p(x_j \mid \theta_{1c}) \prod_{j=2}^D p(x_j \mid \theta_{jc}) dx_1 = \prod_{j=2}^D p(x_j \mid \theta_{jk}) \quad (8.96)$$

其中用到了 $\int p(x_1 \mid y = c, \theta) dx_1 = 1$ 的事实。因此，在**朴素贝叶斯分类器**里，测试时可以**直接忽略**缺失特征。类似地，在**判别分析**（高斯判别）中，无论用什么正则化估计参数，我们都能**解析地**把缺失变量边缘化（见 4.3 节）：

$$p(x_{2:D} \mid y = c, \theta) = N(x_{2:D} \mid \mu_{c,2:D}, \Sigma_{c,2:D,2:D}). \quad (8.97)$$

8.6.2.2 训练时的缺失

训练时的缺失更难处理。特别地，计算 MLE 或 MAP 估计不再是一个简单的优化问题（原因见 11.3.2 节）。不过，接下来我们会学习一系列更高级的算法（例如 EM 算法，见 11.4 节），用来在这种情形下寻找近似的 ML 或 MAP 解。

通俗解释（基础且详尽）

1) 先弄清三种“缺失机制”

- **MCAR (完全随机缺失)**：丢不丢这项，和它的真实值、也和其它任何特征都无关。像掷硬币决定是否漏掉。→ 最好处理。
- **MAR (随机缺失)**：丢不丢这项，只取决于已经看到的那部分特征（或外部已观测变量）。例如：收入信息是否缺失，可能与“是否勾选隐私选项（已知）”有关，但与“真实收入值本身（未知）”无关。→ 还能靠建模搞定（本文默认）。
- **NMAR (非随机缺失)**：丢不丢这项取决于它自己真实的值或其他未观测因素（比如收入越高越不报）。→ 必须对缺失机制 $p(r \mid x)$ 也建模，否则会偏。

记忆口诀：**MCAR** = Magic coin toss；**MAR** = Measurable/observed info drives missing；**NMAR** = Need model missingness.

2) 为什么生成式模型更擅长？

- 生成式模型有 $p(x \mid y)$ （甚至 $p(x, y)$ ）这类**联合/条件分布**，你可以对缺失维做积分（把看不见的那部分“抹掉”）。
- 判别式 $p(y \mid x)$ 默认 x 是**完整的**；一旦缺维，就缺乏一个规范的“把它积分掉”的通路（更多是启发式填补）。

3) 测试时缺失怎么做（两例）？

- **朴素贝叶斯**：特征条件独立 → 缺的那一维的似然 $\int p(x_1 \mid \theta) dx_1 = 1$ ，于是**直接忽略缺失维**，只把其它维的 $p(x_j \mid \theta)$ 相乘即可（式 8.96）。

- **高斯判别/线性判别分析 (LDA/QDA)** : 多元高斯的任意子向量仍是高斯 \rightarrow 用子均值 $\mu_{2:D}$ 、子协方差 $\Sigma_{2:D,2:D}$ 的密度即可 (式 8.97)。这就是把缺失维**解析边缘化**。

小提示：这比“先用均值填补再丢进分类器”更原则，且常更准。

4) 训练时缺失为什么更难？

- 现在不仅测试点缺维，**训练样本**也缺。要学参数 θ 时，目标里包含了对缺失值的**积分/求和**，导致 MLE/MAP 不再是“普通凸优化”。
- 经典解法是 EM：
 - E 步：在当前参数下，对缺失值/隐变量求**条件期望**；
 - M 步：在“填充了期望”的充分统计量上**最大化**。
- 这背后的通用思想叫**不完全数据的最大似然**（细节在第 11 章）。

5) 标签缺失与半监督

- 测试时标签本来就未知。
 - **训练时有些标签也缺** \rightarrow **半监督学习**：生成式方法（有 $p(x)$ 或 $p(x | y)$ ）天然能把**无标签样本**也利用上；判别式需要额外技巧。
-

实操小抄

- **若缺失只发生在测试时**
 - 朴素贝叶斯：**忽略缺失维**，只乘其余维的似然；
 - LDA/QDA：用**子高斯密度** $N(x_{obs} | \mu_{obs}, \Sigma_{obs,obs})$ ；
 - 其它生成式：把缺失维**边缘化积分**。
 - **若训练也缺失 (MAR)**
 - 首选 EM / 变分 / MCMC 等**近似推断**求 MLE/MAP；
 - 若是判别式模型，常需**数据插补** + **训练**或“生成式-判别式混合”方案。
 - **若怀疑 NMAR**
 - 需要显式建 $p(r | x)$ (缺失机制)，否则会系统性偏差（推荐参考专门文献，如 Marlin 2008）。
-

小结

- MCAR/MAR/NMAR 描述“为什么会缺”；仅当 MAR/MCAR 时，可不建缺失机制。
- **生成式模型能把缺失特征边缘化**，测试时处理非常自然；

- 训练时缺失需要 EM 等推断算法。
- 这正是生成式方法在工程实践中一个“看得见的优势”。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.11 Example of Fisher's linear discriminant. (a) Two class data in 2D. Dashed green line = first principal basis vector. Dotted red line = Fisher's linear discriminant vector. Solid black line joins the class-conditional means. (b) Projection of points onto Fisher's vector shows good class separation. (c) Projection of points onto PCA vector shows poor class separation.

8.6.3 Fisher's linear discriminant analysis (FLDA)

Discriminant analysis is a generative approach to classification, which requires fitting an MVN to the features. As we have discussed, this can be problematic in high dimensions. An alternative approach is to reduce the dimensionality of the features $x \in RD$ and then fit an MVN to the resulting low-dimensional features $z \in RL$. The simplest approach is to use a linear projection matrix, $z = Wx$, where W is a $L \times D$ matrix. One approach to finding W would be to use PCA (Section 12.2); the result would be very similar to RDA (Section 4.2.6), since SVD and PCA are essentially equivalent. However, PCA is an unsupervised technique that does not take class labels into account. Thus the resulting low dimensional features are not necessarily optimal for classification, as illustrated in Figure 8.11. An alternative approach is to find the matrix W such that the low-dimensional data can be classified as well as possible using a Gaussian class-conditional density model. The assumption of Gaussianity is reasonable since we are computing linear combinations of (potentially non-

Gaussian) features. This approach is called Fisher's linear discriminant analysis, or FLDA.

FLDA is an interesting hybrid of discriminative and generative techniques. The drawback of this technique is that it is restricted to using $L \leq C - 1$ dimensions, regardless of D , for reasons that we will explain below. In the two-class case, this means we are seeking a single vector w onto which we can project the data. Below we derive the optimal w in the two-class case. We then generalize to the multi-class case, and finally we give a probabilistic interpretation of this technique.

下面先给**完整逐字翻译**，再做**通俗解释**（把图 8.11 的含义、FLDA 的直觉、与 PCA 的区别、为什么最多取到 $C - 1$ 维都讲清楚）。

译文

图 8.11 Fisher 线性判别的示例。

- (a) 二类、二维数据。绿色虚线 = 第一主成分基向量 (PCA)。红色点划线 = Fisher 的线性判别向量。黑色实线连接两类的类条件均值。
- (b) 把数据点投影到 Fisher 向量上，显示出很好的类别分离。
- (c) 把数据点投影到 PCA 向量上，类别分离较差。

8.6.3 Fisher 的线性判别分析 (FLDA)

判别分析 (discriminant analysis) 是一个**生成式**的分类方法，需要对特征拟合一个**多元高斯分布 (MVN)**。正如我们讨论过的，在高维里这可能会有问题。另一种做法是**先把特征 $x \in \mathbb{R}^D$ 的维度降到更低的 $z \in \mathbb{R}^L$** ，然后再对这个低维特征去拟合多元高斯。最简单的做法是用一个**线性投影矩阵 $z = Wx$** ，其中 W 是 $L \times D$ 的矩阵。

找 W 的一种方法是用 PCA (第 12.2 节)；其结果会与 RDA (第 4.2.6 节) 非常相似，因为 SVD 与 PCA 本质等价。但是，PCA 是**无监督**技术，不考虑类别标签。因此得到的低维特征**不一定对分类最优**，如图 8.11 所示。

另一种方法是寻找这样的矩阵 W ：在高斯类条件密度模型下，投影后的低维数据**尽可能容易被正确分类**。由于我们在做特征的线性组合 (即使原特征非高斯)，**假设高斯**是合理的。这个方法称为 **Fisher 线性判别分析 (FLDA)**。

FLDA 是一个**判别式与生成式的有趣混合**。它的一个缺点是：**无论原始维数 D 多大，最终只能使用 $L \leq C - 1$ 个维度** (C 是类别数)，原因我们稍后解释。对二分类来说，这意味着

着我们只需求一个向量 w , 把数据投影到这条直线上。下面我们将推导二分类情形下的最优 w ; 然后推广到多分类情形, 最后给出该技术的概率化解释。

通俗解释 (基础且详尽)

1) 图 8.11 说明了什么?

- (a) 二维散点图:
 - 黑实线连接两类的平均位置 (类均值)。
 - **绿虚线 (PCA 方向) **是数据总体方差最大的方向——它只关心“数据分散到哪里”, 不看标签。
 - 红点划线 (Fisher 方向) 专门挑一个方向, 让两类在这个方向上尽量拉开, 而类内在这个方向上尽量紧。
- (b) 把点投影到 Fisher 方向, 两类直方图几乎不重叠 → 便于设阈值分类。
- (c) 把点投影到 PCA 方向, 两类分布明显重叠 → 分类效果差。

一眼记住: PCA 找“整体最散方向”, FLDA 找“最能分开类别的方向”。

2) 为什么要先降维再做高斯判别?

- 在高维 (特征多、样本相对少) 时, 直接估计高斯的协方差矩阵不稳/难。
- 先用线性投影 $z = Wx$ 把维数降到 L (典型地很小), 再对 z 做高斯判别, 估计更稳、计算更省。

3) FLDA 的核心目标 (直观版)

- 设二类的投影均值为 m_1, m_2 , 类内投影方差为 s_1^2, s_2^2 。
- Fisher 的思想: 找方向 w 让

$$(类间距离) / (类内方差) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \text{ 最大.}$$

- 这样两类在这条线上的中心相距远、各自分布又窄 → 一维上容易分类。

更正式地, 会得到广义特征值问题: 最大化 $w^\top S_B w / (w^\top S_W w)$, 其中 S_B = 类间散度、 S_W = 类内散度。解是 $S_W^{-1}(\mu_1 - \mu_2)$ (二类时)。

4) 为什么最多只有 $C - 1$ 维?

- 类间散度 S_B 的秩最多是 $C - 1$ (只有 C 个类均值, 去掉整体均值后最多独立 $C - 1$ 个方向)。
- 因此最多只能找到 $C - 1$ 个有区分力的投影方向。

- 这与“用 C 个点（类均值）张成的子空间最多是 $C - 1$ 维”是一个直觉。

5) FLDA 与 PCA 的差别与联系

- PCA：无监督，只最大化总方差；方向与分类**不一定相关**。
- FLDA：有监督，最大化“类间散度 / 类内散度”；专为**分类设计**。
- 两者都是**线性投影**；PCA 常用 SVD 计算，FLDA 常用**广义特征分解**或解线性方程 $S_W w \propto \mu_1 - \mu_2$ （二类）。

6) FLDA 是“判别+生成”的混合

- **判别部分**：选 W 时用到了标签 y ，直接优化可分性；
- **生成部分**：在投影后的空间里，假设每类是**高斯**，再做**高斯判别**（LDA/QDA）。

7) 一个简单的使用流程（二类）

1. 计算两类的样本均值 μ_1, μ_2 ；类内协方差 $S_W = S_1 + S_2$ 。
2. 方向 $w \propto S_W^{-1}(\mu_1 - \mu_2)$ （求解线性方程即可）。
3. 把样本投影 $z = w^\top x$ ，在一维上按阈值分类（可配合高斯判别或直接选中点阈值）。
4. 多类时解广义特征值问题，取最大的 $C - 1$ 个特征向量组成 W 。

小结

- PCA 找方差最大的方向；FLDA 找**区分度**最大的方向（类间远、类内紧）。
- 先用 $z = Wx$ 降到 $L \leq C - 1$ 维，再用高斯类条件模型分类，是 FLDA 的套路。
- 图 8.11 清楚展示：Fisher **方向**上两类分得好，而 PCA **方向**上重叠严重 → 无监督降维未必利于分类。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.6.3.1 Derivation of the optimal 1d projection

We now derive this optimal direction w , for the two-class case, following the presentation of (Bishop 2006b, Sec 4.1.4). Define the class-conditional means as
 $\mu_1 = \frac{1}{N_1} \sum_{i:y_i=1} x_i, \mu_2 = \frac{1}{N_2} \sum_{i:y_i=2} x_i$ (8.98)

Let $m_k = w^T \mu_k$ be the projection of each mean onto the line w . Also, let $z_i = w^T x_i$ be the projection of the data onto the line. The variance of the projected points is proportional to

$$s_2^2 = \frac{1}{n} \sum_{i=1}^n (z_i - m_k)^2 \quad (8.99)$$

The goal is to find w such that we maximize the distance between the means, $m_2 - m_1$, while also ensuring the projected clusters are “tight”:

$$J(w) = (m_2 - m_1)^2 s_2^2 + s_1^2 \quad (8.100)$$

We can rewrite the right hand side of the above in terms of w as follows

$$J(w) = w^T S_B w + w^T S_W w \quad (8.101)$$

where S_B is the between-class scatter matrix given by

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \quad (8.102)$$

and S_W is the within-class scatter matrix, given by

$$S_W = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_1)(x_i - \mu_1)^T + \frac{1}{n} \sum_{i=1}^n (x_i - \mu_2)(x_i - \mu_2)^T \quad (8.103)$$

To see this, note that

$$w^T S_B w = w^T (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T w = (m_2 - m_1)(m_2 - m_1) \quad (8.104)$$

and

$$\begin{aligned} w^T S_W w &= \frac{1}{n} \sum_{i=1}^n w^T (x_i - \mu_1)(x_i - \mu_1)^T w + \frac{1}{n} \sum_{i=1}^n w^T (x_i - \mu_2)(x_i - \mu_2)^T w \\ &= (x_i - \mu_1)^T w (x_i - \mu_1) + (x_i - \mu_2)^T w (x_i - \mu_2) \end{aligned} \quad (8.105)$$

$$= i:y_i=1 (z_i - m_1)^2 + i:y_i=2 (z_i - m_2)^2 \quad (8.106)$$

Equation 8.101 is a ratio of two scalars; we can take its derivative with respect to w and equate to zero. One can show (Exercise 12.6) that that $J(w)$ is maximized when

$$S_B w = \lambda S_W w \quad (8.107)$$

where

$$\lambda = \frac{w^T S_B w}{w^T S_W w} \quad (8.108)$$

Equation 8.107 is called a generalized eigenvalue problem. If S_W is invertible, we can convert it to a regular eigenvalue problem:

$$S_B^{-1} S_W w = \lambda w \quad (8.109)$$

However, in the two class case, there is a simpler solution.

In particular, since

$$S_B w = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T w = (\mu_2 - \mu_1)(m_2 - m_1) \quad (8.110)$$

then, from Equation 8.109 we have

$$\lambda w = S_B^{-1} S_W w = (\mu_2 - \mu_1)(m_2 - m_1) \quad (8.111)$$

$$w \propto S_B^{-1} S_W w = (\mu_2 - \mu_1) \quad (8.112)$$

Since we only care about the directionality, and not the scale factor, we can just set

$$w = S_B^{-1} S_W w = (\mu_2 - \mu_1) \quad (8.113)$$

This is the optimal solution in the two-class case. If $SW \propto I$, meaning the pooled covariance matrix is isotropic, then w is proportional to the vector that joins the class means. This is an intuitively reasonable direction to project onto, as shown in Figure 8.11.

已思考 21s

下面先给**完整逐字翻译**, 再做**通俗解释** (把每一步推导都讲清楚; 适合数学基础薄弱时阅读)。

译文

8.6.3.1 最优一维投影的推导

现在我们来为二分类情形推导这个最优方向 w (参考 Bishop 2006b 第 4.1.4 节的表述)。

把类条件均值定义为

$$\mu_1 = \frac{1}{N_1} \sum_{i: y_i=1} x_i, \quad \mu_2 = \frac{1}{N_2} \sum_{i: y_i=2} x_i \quad (8.98)$$

令 $m_k = w^\top \mu_k$ 为每个均值在直线 w 上的投影。又令 $z_i = w^\top x_i$ 为数据在该直线上的投影。投影点的方差正比于

$$s_k^2 = \sum_{i: y_i=k} (z_i - m_k)^2 \quad (8.99)$$

我们的目标是寻找 w , 使得均值间的距离 $m_2 - m_1$ 最大, 同时投影后的簇“尽量紧”:

$$J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \quad (8.100)$$

上式右边可以改写成与 w 有关的形式

$$J(w) = \frac{w^\top S_B w}{w^\top S_W w} \quad (8.101)$$

其中, **类间散度矩阵** S_B 为

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^\top \quad (8.102)$$

类内散度矩阵 S_W 为

$$S_W = \sum_{i: y_i=1} (x_i - \mu_1)(x_i - \mu_1)^\top + \sum_{i: y_i=2} (x_i - \mu_2)(x_i - \mu_2)^\top \quad (8.103)$$

要验证式 (8.101), 注意

$$w^\top S_B w = w^\top (\mu_2 - \mu_1)(\mu_2 - \mu_1)^\top w = (m_2 - m_1)(m_2 - m_1) \quad (8.104)$$

以及

$$\begin{aligned} & \begin{aligned} w^\top S_W w &= \sum_{i:y_i=1} w^\top (x_i - \mu_1)(x_i - \mu_1)^\top w + \sum_{i:y_i=2} w^\top (x_i - \mu_2)(x_i - \mu_2)^\top w \end{aligned} \\ & \quad \&= \sum_{i:y_i=1} (z_i - m_1)^2 + \sum_{i:y_i=2} (z_i - m_2)^2 \end{aligned} \quad (8.105) \quad (8.106)$$

式 (8.101) 是两个标量的比值; 我们可以对 w 求导并令其为零。可证明 (见习题 12.6) $J(w)$ 在以下条件下取最大值:

$$S_B w = \lambda S_W w \quad (8.107)$$

其中

$$\lambda = \frac{w^\top S_B w}{w^\top S_W w} \quad (8.108)$$

式 (8.107) 称为**广义特征值问题**。若 S_W 可逆, 可化为普通特征值问题:

$$S_W^{-1} S_B w = \lambda w \quad (8.109)$$

不过在二类情况下有更简单的解。特别地, 因为

$$S_B w = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^\top w = (\mu_2 - \mu_1)(m_2 - m_1) \quad (8.110)$$

由式 (8.109) 得

$$\lambda w = S_W^{-1}(\mu_2 - \mu_1)(m_2 - m_1) \quad (8.111)$$

即

$$w \propto S_W^{-1}(\mu_2 - \mu_1) \quad (8.112)$$

由于我们只关心方向而非尺度, 可直接取

$$w = S_W^{-1}(\mu_2 - \mu_1) \quad (8.113)$$

这就是二分类时的最优解。若 $S_W \propto I$, 即合并协方差矩阵各向同性, 则 w 与**连接两类均值的向量成正比**。这是一个直观合理的投影方向, 如图 8.11 所示。

通俗解释 (基础且详尽)

1) 我们要最大化的量是什么?

- 分子 $(m_2 - m_1)^2$: 两类在投影线上的**均值间距** (越大越好)。
- 分母 $s_1^2 + s_2^2$: 两类在投影线上的**类内散布** (越小越好)。
- 于是 $J(w)$ = “**类间远 / 类内近**”，找使其最大的方向 w 。

这正是 Fisher 判别的思想：在某条线上看，两类中心尽量拉开，各自尽量紧凑。

2) 为什么能写成 $\frac{w^\top S_B w}{w^\top S_W w}$?

- $S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^\top$ 是把“**两类均值差**”打包成矩阵；
 $w^\top S_B w = (w^\top(\mu_2 - \mu_1))^2 = (m_2 - m_1)^2$ ——正是分子。
- S_W 把“**类内偏差的协方差**”求和；
 $w^\top S_W w = \sum(w^\top(x_i - \mu_{y_i}))^2 = \sum(z_i - m_{y_i})^2$ ——正是分母。

3) 如何求最优 w ?

- 这是一个 Rayleigh 商最大化问题；等价于

$$\max_w w^\top S_B w \quad \text{s.t. } w^\top S_W w = 1$$

用拉格朗日乘子法一推，得到一阶条件 $S_B w = \lambda S_W w$ ，即式 (8.107)。

- 若 S_W 可逆，化成普通特征值问题 $(S_W^{-1} S_B)w = \lambda w$ ，取最大的特征值方向即最优。
- 二类特化：** S_B 是一个秩 1 的外积矩阵（只有一个非零特征值），于是直接得到

$$w \propto S_W^{-1}(\mu_2 - \mu_1)$$

这一步把“沿均值差方向走”，用 S_W^{-1} 做了“**白化/伸缩**”，抵消不同方向的类内方差。

4) 两个重要特例

- 各向同性：** $S_W \propto I$ 时， $w \propto (\mu_2 - \mu_1)$ ——把数据投到“连接两类均值”的直线上是最优的（直觉完全吻合）。
- 奇异 S_W** （样本少、维度大常见）：可以加微小正则 $S_W + \lambda I$ 再求解（称为正则化 FLDA）。

5) 怎么用这个 w 做分类？

- 计算 $z = w^\top x$ （把样本投到这条线上）。
- 计算两类投影均值 m_1, m_2 和（必要时）方差；
- 若先验相等且方差近似相同，阈值可取 $\frac{m_1 + m_2}{2}$ ；更一般地可在 1D 上做高斯判别。

6) 与 LDA/逻辑回归的关系

- 若每类是高斯且同协方差，Fisher 方向与 LDA 的贝叶斯最优线性边界一致；这时它和逻辑回归得到的直线边界也**同形**。

小结

- Fisher 的目标：最大化 $\frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$ ；
- 得到广义特征值条件 $S_B w = \lambda S_W w$ ；二类的闭式解

$$w = S_W^{-1}(\mu_2 - \mu_1)$$

- 若类内协方差各向同性，最优方向就是“**连接两类均值**”的方向。



翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

Figure 8.12 (a) PCA projection of vowel data to 2d. (b) FLDA projection of vowel data to 2d. We see there is better class separation in the FLDA case. Based on Figure 4.11 of (Hastie et al. 2009). Figure generated by fisherDiscrimVowelDemo, by Hannes Bretschneider.

8.6.3.2 Extension to higher dimensions and multiple classes

We can extend the above idea to multiple classes, and to higher dimensional subspaces, by finding a projection matrix W which maps from D to L so as to maximize $J(W) = |W\Sigma B W^T| / |W\Sigma W W^T|$ (8.114)
where

$$\Sigma B = c N_c N (\mu_c - \mu)(\mu_c - \mu)^T \quad (8.115)$$

$$\Sigma W = c N_c N \Sigma_c \quad (8.116)$$

$$\Sigma_c = 1 N_c i: y_i = c (x_i - \mu_c)(x_i - \mu_c)^T \quad (8.117)$$

The solution can be shown to be

$$W = \Sigma^{-1/2} W U \quad (8.118)$$

where U are the L leading eigenvectors of $\Sigma^{-1/2} W \Sigma B \Sigma^{-1/2} W^T$

$-1/2 W^T \Sigma W$, assuming ΣW is non-singular. (If it is singular, we can first perform PCA on all the data.) Figure 8.12 gives an example of this method applied to some $D = 10$ dimensional speech data, representing $C = 11$ different vowel sounds. We see that FLDA gives better class separation than PCA. Note that FLDA is restricted to finding at most a $L \leq C - 1$ dimensional linear subspace, no matter how large D , because the rank of the between class covariance matrix Σ_B is $C - 1$. (The -1 term arises because of the μ term, which is a linear function of the μ_c .) This is a rather severe restriction which limits the usefulness of FLDA.

已思考 8s

下面先给**完整逐字翻译**, 再做**通俗解释** (把每个公式、步骤与直觉讲透; 最后给出可操作的小算法)。

译文

图 8.12 (a) 将元音数据用 PCA 投影到 2 维。 (b) 将元音数据用 FLDA 投影到 2 维。可以看到在 FLDA 情况下类别分离得更好。基于 (Hastie 等, 2009) 的图 4.11。图由 Hannes Bretschneider 的 `fisherDiscrimVowelDemo` 生成。

8.6.3.2 推广到高维与多类别

我们可以把上面的思想推广到**多类别**, 以及**更高维的子空间**, 方法是寻找一个把 D 维映到 L 维的投影矩阵 W , 使下面的目标最大:

$$J(W) = \frac{|W \Sigma_B W^T|}{|W \Sigma_W W^T|} \quad (8.114)$$

其中

$$\Sigma_B = \sum_c \frac{N_c}{N} (\mu_c - \mu)(\mu_c - \mu)^T \quad (8.115)$$

$$\Sigma_W = \sum_c \frac{N_c}{N} \Sigma_c \quad (8.116)$$

$$\Sigma_c = \frac{1}{N_c} \sum_{i:y_i=c} (x_i - \mu_c)(x_i - \mu_c)^T \quad (8.117)$$

可以证明其解为

$$W = \Sigma_W^{-1/2} U \quad (8.118)$$

其中 U 是矩阵 $\Sigma_W^{-1/2} \Sigma_B \Sigma_W^{-1/2}$ 的前 L 个特征向量（假定 Σ_W 可逆）。若 Σ_W 奇异，可以先对所有数据做一次 PCA。

图 8.12 给出了该方法在一个 $D = 10$ 维的语音数据上的示例，这个数据包含 $C = 11$ 个不同的元音。可以看到 FLDA 比 PCA 有更好的类别分离。需要注意，**无论 D 多大，FLDA 最多只能找到 $L \leq C - 1$ 维的线性子空间**，因为类间协方差矩阵 Σ_B 的秩是 $C - 1$ 。（这个“-1”是由式中的总体均值 μ 引起的，它是各类均值 μ_c 的线性函数。）这是一种较为严格的限制，会影响 FLDA 的适用性。

通俗解释（基础且详尽）

1) 这些矩阵都是什么？

- μ_c : 第 c 类的样本均值； μ : **总体均值**。 N_c 为第 c 类样本数， $N = \sum_c N_c$ 。
- Σ_c : 第 c 类的**类内协方差**（式 8.117）。
- Σ_W : 把各类的 Σ_c 按样本占比加权平均得到的**总体类内协方差**（式 8.116）。
- Σ_B : 各类均值相对整体均值 μ 的离散程度——**类间协方差**（式 8.115）。

直觉： Σ_B 越大 → 各类的“中心”彼此越远； Σ_W 越小 → 每一类自己越紧。

2) 目标函数 $J(W)$ 的含义

$$J(W) = \frac{|W \Sigma_B W^\top|}{|W \Sigma_W W^\top|}$$

- 分子是投影后**类间散度的体积**（行列式=体积）；
- 分母是投影后**类内散度的体积**。
→ 我们找一个 W 让“**类间大 / 类内小**”。

这就是多维版本的 Fisher 思想（一维时是 $(m_2 - m_1)^2/(s_1^2 + s_2^2)$ ）。

3) 为什么解是 $W = \Sigma_W^{-1/2} U$ ？

- 先把数据在类内协方差的意义下**白化** ($\Sigma_W^{-1/2}$)，把不同方向的类内尺度调成一致；
- 在白化后的空间里，只需要找能让类中心分得最开的 L 个正交方向——这正是 $\Sigma_W^{-1/2} \Sigma_B \Sigma_W^{-1/2}$ 的前 L 个**特征向量** U ；
- 最终把这组方向映回原空间，得到 $W = \Sigma_W^{-1/2} U$ 。

这与二类情形 $w \propto \Sigma_W^{-1}(\mu_2 - \mu_1)$ 完全一致，只是推广到多类与多维。

4) 为什么最多 $L \leq C - 1$ 维?

- Σ_B 是若干向量 $(\mu_c - \mu)$ 的加权外积之和；这些向量之和为 0 (因为 $\mu = \sum_c \frac{N_c}{N} \mu_c$)，所以它们至多张成一个 $C - 1$ 维子空间。
- 因此 $\text{rank}(\Sigma_B) = C - 1 \rightarrow$ 最多只能提取 $C - 1$ 个有判别力的方向。

5) 与 PCA 的对比 (图 8.12)

- PCA 只最大化总体方差，不用标签 \rightarrow 在元音数据上，类别在 2D 平面里仍有明显混叠。
- FLDA 用到标签，显式最大化“类间/类内” \rightarrow 同样 2D 投影里，各类别分得更开，分类更容易。

6) 数值/实现注意

- Σ_W 在高维小样本 ($D \gg N$) 时常奇异。做法：
 1. 先对全部数据做 PCA 降到不奇异的维数 (或加小的 λI 正则化)；
 2. 再按上式求 W 。
- 选择投影维数：理论上线性可分方向不超过 $C - 1$ ，实务中可取 $L = \min(C - 1, \text{需要的维数})$ 。

7) 简单可抄的步骤 (多类 FLDA)

1. 计算各类均值 μ_c 与总体均值 μ 。
2. 计算各类协方差 Σ_c ，再得 $\Sigma_W = \sum_c \frac{N_c}{N} \Sigma_c$ 。
3. 计算 $\Sigma_B = \sum_c \frac{N_c}{N} (\mu_c - \mu)(\mu_c - \mu)^\top$ 。
4. 若 Σ_W 奇异：先 PCA 或用 $\Sigma_W + \lambda I$ 。
5. 求 $\Sigma_W^{-1/2} \Sigma_B \Sigma_W^{-1/2}$ 的前 $L \leq C - 1$ 个特征向量 U 。
6. 令 $W = \Sigma_W^{-1/2} U$ ，做降维 $z = Wx$ 。
7. 在 z 空间里用高斯类条件模型 (或最近均值等) 分类。

小结

- 多类/多维 FLDA 通过最大化 $|W \Sigma_B W^\top| / |W \Sigma_W W^\top|$ 找到最“能分类”的线性子空间；
- 解等价于白化类内协方差后取类间协方差的主方向；
- 天然上限 $L \leq C - 1$ 源于类间协方差的秩；
- 实证上 (图 8.12)，FLDA 投影比 PCA 更能拉开类别。

翻译并解释以下内容，翻译不能有遗漏，解释请基础且详尽，因为我的数学基础不太好。

8.6.3.3 Probabilistic interpretation of FLDA

To find a valid probabilistic interpretation of FLDA, we follow the approach of (Kumar and Andreo 1998; Zhou et al. 2009). They proposed a model known as heteroscedastic LDA (HLDA), which works as follows. Let W be a $D \times D$ invertible matrix, and let $z_i = Wx_i$ be a transformed version of the data. We now fit full covariance Gaussians to the transformed data, one per class, but with the constraint that only the first L components will be class-specific; the remaining $H = D - L$ components will be shared across classes, and will thus not be discriminative. That is, we use

$$p(z_i|\theta, y_i = c) = N(z_i|\mu_c, \Sigma_c) \quad (8.119)$$

$$\mu_c = (m_c; m_0) \quad (8.120)$$

$$\Sigma_c = S_c \otimes S_0 \quad (8.121)$$

where m_0 is the shared H dimensional mean and S_0 is the shared $H \times H$ covariance. The pdf of the original (untransformed) data is given by

$$p(x_i|y_i = c, W, \theta) = |W|N(Wx_i|\mu_c, \Sigma_c) \quad (8.122)$$

$$= |W|N(WLx_i|m_c, S_c) N(WHx_i|m_0, S_0) \quad (8.123)$$

where $W = WLWH$. For fixed W , it is easy to derive the MLE for θ . One can then optimize W using gradient methods.

In the special case that the Σ_c are diagonal, there is a closed-form solution for W (Gales 1999). And in the special case the Σ_c are all equal, we recover classical LDA (Zhou et al. 2009). In view of this result, it should be clear that HLDA will outperform LDA if the class covariances are not equal within the discriminative subspace (i.e., if the assumption that Σ_c is independent of c is a poor assumption). This is easy to demonstrate on synthetic data, and is also the case on more challenging tasks such as speech recognition (Kumar and Andreo 1998). Furthermore, we can extend the model by allowing each class to use its own projection matrix; this is known as multiple LDA (Gales 2002).

下面先给**完整逐字翻译**, 再做**通俗解释** (把模型、公式里每个符号、训练步骤与结论都讲清楚; 适合数学基础较弱时阅读)。

译文

8.6.3.3 FLDA 的概率化解释

为了给 FLDA 找到一个合理的**概率模型**解释, 我们遵循 (Kumar and Andreo, 1998; Zhou et al., 2009) 的做法。他们提出了一个叫作**异方差 LDA (HLDA, heteroscedastic LDA)** 的模型, 工作方式如下。设 W 是一个 $D \times D$ 的**可逆矩阵**, 并令

$z_i = Wx_i$ 为数据的一个**线性变换**。现在我们在这个变换后的空间里, 对每个类别各自拟合一个**满协方差高斯分布**, 但施加一个约束: **只有前 L 个分量是类别特有的**; 其余 $H = D - L$ 个分量在各类别之间**共享**, 因此不是判别性的。也就是说, 我们使用

$$p(z_i | \theta, y_i = c) = N(z_i | \mu_c, \Sigma_c) \quad (8.119)$$

$$\mu_c = \begin{pmatrix} m_c \\ m_0 \end{pmatrix} \quad (8.120)$$

$$\Sigma_c = \begin{pmatrix} S_c & 0 \\ 0 & S_0 \end{pmatrix} \quad (8.121)$$

其中 m_0 是共享的 H 维均值, S_0 是共享的 $H \times H$ 协方差。

原始 (未变换) 数据的概率密度为

$$p(x_i | y_i = c, W, \theta) = |W| N(Wx_i | \mu_c, \Sigma_c) \quad (8.122)$$

$$= |W| N(W_L x_i | m_c, S_c) N(W_H x_i | m_0, S_0), \quad (8.123)$$

其中 $W = \begin{pmatrix} W_L \\ W_H \end{pmatrix}$ (把 W 按行分成前 L 行与后 H 行)。在给定 W 的情况下, 很容易推出 θ 的极大似然估计; 随后可以用**梯度法**来优化 W 。

在一个特殊情形下, 若所有 Σ_c 都是**对角阵**, 则存在 W 的**闭式解** (Gales, 1999)。再者, 当所有 Σ_c 相等时, 我们可以**退化回经典 LDA** (Zhou et al., 2009)。由此可见, 当在判别子空间内各类的协方差并不相等 (也就是“ Σ_c 与 c 无关”的假设很差) 时, **HLDA 会优于 LDA**。这在合成数据上很容易演示, 在更具挑战性的任务 (如语音识别) 上也成立 (Kumar and Andreo, 1998)。此外, 我们还可以把模型扩展为**每个类别使用自己的投影矩阵**, 这称为 multiple LDA (Gales, 2002)。

通俗解释 (基础且详尽)

1) HLDA 在干嘛? (一句话)

找一个线性变换 $z = Wx$, 使得在这个新坐标里:

- 前 L 维是“**判别维**”: 不同类在这 L 维上的**均值** m_c 和/或**协方差** S_c 可以不同;
- 后 $H = D - L$ 维是“**公共噪声维**”: 不管哪一类, 都共享同一个**均值** m_0 和**协方差** S_0
-

这样就把“有助于分类的信息”集中到前 L 维, 剩下的维度当作共同背景噪声处理。

2) 为什么式 (8.122) 里有 $|W|$?

这是**变量变换的雅可比行列式**: 从 x 变到 $z = Wx$, 密度要乘以 $|\det W|$ 才能保体积一致。它保证 $p(x)$ 是一个合法的概率密度。

3) 如何训练?

- 已知 W :
 - 判别维 (前 L 维) : 对每个类 c 的 $\{W_L x_i\}$ 求均值/协方差, 得 m_c, S_c 。
 - 噪声维 (后 H 维) : 把所有类的 $\{W_H x_i\}$ 合在一起求均值/协方差, 得 m_0, S_0 。
→ 这就是 $\theta = \{m_c, S_c, m_0, S_0\}$ 的**极大似然闭式更新**。
- 优化 W : 固定 θ , 对对数似然 $\sum_i \log p(x_i | y_i, W, \theta)$ 对 W 求梯度并上升。可交替进行, 直到收敛 (看起来像 EM, 但这里直接做坐标上升也行)。

4) 与 LDA/FLDA 的关系

- 所有 Σ_c 相等 (同协方差, homoscedastic) → HLDA 退化为**经典 LDA/FLDA**: 边界是**线性的**; 取到的投影子空间就是我们前面推导的 W 。
- Σ_c 不相等 (异方差, heteroscedastic) → LDA 的假设被破坏, 性能下降; 而 HLDA 允许不同类在判别维上有不同的协方差 S_c , 因此**更灵活, 效果更好**。边界一般为**二次** (与 QDA 类似), 但只在 L 维判别子空间里发生。

5) 为什么要把协方差分块成这样?

- 把“能区分类别的信息”都放在前 L 维 (均值差/协方差差), 把“区分无关的变化”放在后 H 维并共享, 从而**提高判别信噪比**、减少需要估计的参数量 (后 H 维只要一个共享的 m_0, S_0)。

6) 两个重要特例

- 对角 Σ_c 有闭式 W (Gales, 1999) : 说明在“各维度近似独立”的假设下, 最佳线性变换能直接算出。

- 所有 Σ_c 相等 \Rightarrow 经典 LDA：这也给了 FLDA 的概率学解释——它是在一个特定的高斯生成模型（同协方差）下的最优线性判别投影。

7) multiple LDA 是什么？

让每个类各自一套投影矩阵 W_c ，进一步提升灵活性（非共享的几何结构）。当然参数更多，需要更多数据/正则化以防过拟合。

8) 何时用 HLDA / 它的优势

- 当你怀疑/检验发现：不同类别在关键子空间内的方差结构不同（“异方差”）时，用 HLDA 往往显著优于 LDA。
- 语音识别等任务里，发音人的差异、声学条件等会导致类内协方差不相等，因此 HLDA 常优。

小结（一句话）

HLDA = 给 FLDA 装上一个明确的高斯生成模型：先找一个线性变换，把“可判别的 L 维”与“共享噪声的 H 维”分开；若各类协方差在判别维上相等，退化为 LDA；若不等，HLDA 更合适、通常更准，还可扩展到 multiple LDA（每类自己的投影）。

基于本对话里我贴过的整章原文，输出：
①知识点总结
(分点+关键词+用途) ②子章节关联 (依赖/对比/递进+关系句，附Mermaid可选) ③本章总结。

① 知识点总结 (分点 + 关键词 + 用途)

- 逻辑回归的两条路
 - 关键词：Generative vs Discriminative、GDA、Logistic
 - 用途：理解同一线性决策边界可由生成式或判别式产生；选模时权衡假设强弱与数据需求。
- 模型定义 (8.2)
 - 关键词：sigmoid/sigm、 $p(y|x, w) = \text{Ber}(\text{sigm}(w^\top x))$
 - 用途：二分类概率建模的标准起点；阈值 0.5 \Rightarrow 线性决策边界法向量为 w 。
- 极大似然与目标函数 (8.3.1)
 - 关键词：NLL/交叉熵、对偶标签 $\tilde{y} \in \{-1, +1\}$ 、凸性
 - 用途：明确优化目标；因为凸 \Rightarrow 全局最优唯一；为后续优化法奠基。

- **梯度与海森矩阵 (8.3.1)**
 - 关键词: $g = X^\top(\mu - y)$ 、 $H = X^\top SX$ 、 $S = \text{diag}(\mu_i(1 - \mu_i))$
 - 用途: 任意一阶/二阶优化都要用到的基本量。
- **最陡下降/线搜索/动量 (8.3.2)**
 - 关键词: $\theta_{k+1} = \theta_k - \eta_k g_k$ 、Line Search、Zig-zag、Momentum
 - 用途: 一阶优化的收敛/数值稳定; 理解步长过大/过小的影响与改良。
- **牛顿法 (8.3.3)**
 - 关键词: 二阶泰勒、 $\theta_{k+1} = \theta_k - H_k^{-1} g_k$ 、正定性、截断牛顿
 - 用途: 快速收敛; 当非凸/负曲率时的安全策略与数值实现。
- **IRLS (8.3.4)**
 - 关键词: 迭代加权最小二乘、工作响应 z_k 、权矩阵 S_k
 - 用途: 把逻辑回归的牛顿步写成“每步解一次加权最小二乘”, 工程上好实现。
- **拟牛顿与 L-BFGS (8.3.5)**
 - 关键词: BFGS、Hessian 近似、低秩更新、L-BFGS
 - 用途: 大规模问题的常用黑箱优化器 (内存友好、收敛稳)。
- **L_2 正则 (8.3.6)**
 - 关键词: MAP、ridge、过度自信 (线性可分时 $\|w\| \rightarrow \infty$)
 - 用途: 提高泛化、数值稳定; 轻松并入任何梯度法: 目标/梯度/海森各加 λ 项。
- **多类逻辑回归 (8.3.7)**
 - 关键词: Softmax、分块梯度、克罗内克积、Hessian 块结构、正定
 - 用途: 从二类推广到 C 类; 优化仍是凸的, 便于用二阶/拟牛顿。
- **贝叶斯逻辑回归 (8.4)**
 - 关键词: 后验 $p(w|D)$ 、不可共轭、近似
 - 用途: 给出不确定性 (置信区间、带不确定的预测)。
- **拉普拉斯/高斯近似 (8.4.1)**
 - 关键词: 在 MAP 处二阶展开、 $N(\theta^*, H^{-1})$ 、BIC (8.4.2)
 - 用途: 近似后验 & 边际似然; 可做模型比较与正则强度直观解释 (Occam 因子)。
- **后验预测分布 (8.4.4)**
 - 关键词: Plug-in、Monte Carlo、Probit 近似/Moderated output、 $\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2}$
 - 用途: 把参数不确定性传递到预测; MC 最准、Probit 快速闭式。
- **残差/异常点分析 (8.4.5)**

- 关键词：交叉验证式后验预测 $p(y_i|x_i, x_{-i}, y_{-i})$
- 用途：在二元数据上更合适的“异常点”判定。
- **在线学习与随机优化 (8.5)**
 - 关键词：Regret、Online GD、SGD、Polyak–Ruppert 平均、Robbins–Monro 步长
 - 用途：流式/大规模训练；保证逐步收敛与高效更新。
- **自适应步长 (8.5.2)**
 - 关键词：学习率日程、Adagrad $s_i = \sum g_i^2$
 - 用途：按参数曲率自动缩放步长，免调或少调学习率。
- **LMS/感知机 (8.5.3–8.5.4)**
 - 关键词：LMS（线回归在线 MLE）、Perceptron（0/1 近似梯度、线性可分收敛）
 - 用途：两种经典在线算法；理解 SGD 与 GLM 的关系。
- **在线的贝叶斯视角 (8.5.5)**
 - 关键词：递推 Bayes、卡尔曼滤波、ADF、粒子滤波
 - 用途：在线同时估计均值与方差 = “参数级别自适应步长”。
- **优缺点对比 (8.6.1)**
 - 关键词：训练难度、增类成本、缺失/无标签、输入/输出对称、特征预处理、概率校准
 - 用途：为选型提供 checklist。
- **缺失数据 (8.6.2)**
 - 关键词：MCAR/MAR/NMAR、生成式边缘化、NB/LDA 的解析处理、EM（训练期缺失）
 - 用途：测试缺失可直接积分；训练缺失用 EM/变分等。
- **Fisher 线性判别 (8.6.3)**
 - 关键词：最大化 $\frac{w^\top S_B w}{w^\top S_W w}$ 、二类解 $w \propto S_W^{-1}(\mu_2 - \mu_1)$ 、多类 $L \leq C - 1$
 - 用途：监督降维 + 生成式判别的桥梁；在低维空间用高斯类条件分类。
- **FLDA 的概率解释 (HLDA) (8.6.3.3)**
 - 关键词：异方差 LDA、 $z = Wx$ 、判别子空间（类特异均值/协方差）+ 共享噪声子空间、Multiple LDA
 - 用途：当类协方差不等时优于 LDA；语音等任务实证更强。

② 子章节关联（依赖 / 对比 / 递进）

- 8.1 → 8.2 (铺垫 → 定义) : 阐明判别式路线, 给出二分类概率模型。
- 8.2 → 8.3 (定义 → 拟合) : 从交叉熵目标出发, 推导梯度/海森, 进入优化器家族。
- 8.3.1 → 8.3.2/3/4/5 (一阶 ↔ 二阶 ↔ 近二阶) : 最陡下降 (步长难) → 牛顿 (快但要海森) → IRLS (把牛顿化为 WLS) → BFGS/L-BFGS (省内存的海森近似)。
- 8.3.6 (正则) 贯穿 8.3.*: 在目标/梯度/海森中加 λ 即可; 为可分数据稳定化。
- 8.3.7 (多类) 继承 8.3.1 的结构: 梯度/海森保持“误差 × 特征”的形态, 仍凸。
- 8.3 → 8.4 (频率学 → 贝叶斯) : 在 MAP 之上引入参数不确定性; 拉普拉斯近似把 8.3 的海森再次用起来; BIC 由此而来。
- 8.4.4 (预测) 对比三法: Plug-in (快但乐观) ↔ Monte Carlo (准) ↔ Probit 近似 (快而稳)。
- 8.5 (离线 → 在线) : 将 8.3 的优化搬到流式场景; SGD/步长/自适应与 LMS/Perceptron 连接 GLM。
- 8.6 (判别式 ↔ 生成式对比) : 基于 8.1 的两条路线系统评估; 引出**缺失数据与监督降维**两个生成式优势应用。
- 8.6.2 (缺失) 利用生成式 $p(x|y)$ 进行边缘化; 训练期缺失链接到 11 章的 EM。
- 8.6.3 (FLDA) 把“监督降维”与“高斯判别”结合;
8.6.3.1 → 8.6.3.2 → 8.6.3.3 为**推导 → 多类/高维扩展 → 概率解释 (HLDA) **的递进链。

可选 Mermaid (主脉络) :

```
graph LR
    A[8.1 导论] --> B[8.2 模型定义]
    B --> C[8.3 拟合(梯度/海森)]
    C --> C2[8.3.2 最陡下降/线搜/动量]
    C --> C3[8.3.3 牛顿/截断牛顿]
    C --> C4[8.3.4 IRLS]
    C --> C5[8.3.5 BFGS/L-BFGS]
    C --> C6[8.3.6 L2 正则]
    C --> C7[8.3.7 多类 Softmax]
    C --> D[8.4 贝叶斯LR(拉普拉斯/BIC/预测)]
    C --> E[8.5 在线/SGD/Adagrad/LMS/Perceptron/贝叶斯在线]
    A --> F[8.6 生成式vs判别式]
    F --> G[8.6.2 缺失数据(MCAR/MAR/NMAR)]
    F --> H[8.6.3 FLDA]
    H --> H1[8.6.3.1 二类最优投影]
    H --> H2[8.6.3.2 多类与高维]
    H --> H3[8.6.3.3 概率解释(HLDA)]
```

关系句示例:

- “8.3 的海森不仅用于牛顿步, 也在 8.4 的拉普拉斯近似里充当后验协方差的逆。”
- “8.6.2 的测试期缺失能被生成式直接边缘化; 而 8.5 的在线学习更关心到达式的更新, 不解决缺失本身。”
- “8.6.3 的 FLDA 是监督降维; 当类协方差不等时, 8.6.3.3 的 HLDA 比 LDA 更合理。”

③ 本章总结 (一屏速记)

- **核心主题：**以逻辑回归为代表，系统讲解从**凸优化到贝叶斯近似、从离线到在线训练**的一整套工具，并在结尾与**生成式方法**深入对比，延伸到**缺失数据与Fisher判别/HLDA等监督降维方法**。
- **方法光谱：**
 - **判别式：**交叉熵 + (GD/牛顿/IRLS/BFGS) + 正则 + Softmax；
 - **贝叶斯：**拉普拉斯后验、高斯/MC 预测、BIC；
 - **在线：**SGD/步长日程/Adagrad、LMS、Perceptron、递推 Bayes；
 - **生成式优势场景：**缺失/半监督/反向推断；监督降维 (FLDA/HLDA) 。
- **实践要点：**
 1. 二类/多类 LR：首选 L-BFGS 或 IRLS；数据可分要加 L_2 ；需要不确定性用拉普拉斯 + Monte Carlo/Probit。
 2. 大数据/流式：用 mini-batch SGD + 合理学习率（或 Adagrad/Adam 系），必要时做 Polyak 平均与早停。
 3. 缺失特征：生成式边缘化 > 简单填补；训练期缺失用 EM。
 4. 类别结构复杂：考虑 FLDA/HLDA 做监督降维，再配合高斯判别或逻辑回归。
- **总览：**本章把“**怎么建模 $p(y|x)$** ”、“**怎么高效稳定地求解**”、以及“**何时该换成生成式视角**”三者打通。读者可据此在实际问题间自如切换：**判别式拿精度，生成式拿结构与鲁棒性**；需要不确定性就走贝叶斯近似，需要规模就走在线/SGD。