局部加权线性回归、逻辑回归、随机梯度与牛顿法

前言

在前几篇文章中，我们介绍了最小二乘的推导过程和批量梯度下降的推导过程，这一篇文章实际上是对上述内容的扩展，也就是如标题：局部加权回归、逻辑回归与牛顿法。

这篇文章需要读者掌握前置几篇文章的知识，同时掌握高等数学、线性代数、概率基础

局部加权线性回归 (Locally Weighted Linear Regression)

在往期的案例数据中，我们都是在讨论类线性模型的回归问题，没有考虑过非线性或者模型复杂、数据波动大等情况。现在，我们就来处理，当我们出现这样的模型时的解决方案：局部加权线性回归。

为了更好地理解这个东西，我们有必要去把这个名词做拆解，一个一个进行说明：

1. 线性回归：这个名词应该不陌生了，不懂的去看 线性回归 和 线性回归推导、误差分析、最大似然估计 + 正太分布推最小二乘
2. 局部：这个词是处理非线性模型的关键点所在。我们只需要关注某一点 x_i 的周围 $\cdots y_{i-2}$、y_{i-1}、y_i、y_{i+1}、$y_{i+2}\cdots$ y 值即可，通过这些值我们可以近似估计模型的 y_i 值
3. 加权：在局部的基础上，靠近 x_i 的 x 被赋予更高的权重，给距离 x_i 较远的点赋予较低的权重（甚至权重为0，忽略它们）。距离的衡量通常使用**核函数 (Kernel Function)（*核函数就是为了让线性模型拥有处理非线性问题的处理能力*）**。

公式预览

1. **权重计算：** $W^{(i)} = exp(-\frac{(x^{(i)} - x)^2}{2\tau^2})$
2. **加权正规方程：** $X^T W X\theta = X^T X y$
3. **参数解：** $\theta = (X^\top W X)^{-1} X^\top W y$
4. **预测：** $\hat{y} = [1 \quad x]\theta$

推导过程

局部线性回归推导

案例

数据：

- 训练点： $(1,1),\ (2,2),\ (3,2)$
- 查询点： $x = 1.5$
- 带宽： $\tau = 1$

步骤1：计算权重（高斯核）

$$
\begin{aligned}
w^{(1)} &= \exp\left(-\frac{(1-1.5)^2}{2\cdot 1^2}\right) = e^{-0.125} \approx 0.8825 \\
w^{(2)} &= \exp\left(-\frac{(2-1.5)^2}{2}\right) = e^{-0.125} \approx 0.8825 \\
w^{(3)} &= \exp\left(-\frac{(3-1.5)^2}{2}\right) = e^{-1.125} \approx 0.3247
\end{aligned}
$$

步骤2：构建矩阵

$$
X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}
$$

$$
\mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}
$$

$$
W = \begin{bmatrix} 0.8825 & 0 & 0 \\ 0 & 0.8825 & 0 \\ 0 & 0 & 0.3247 \end{bmatrix}
$$

步骤3：计算关键矩阵

1. $X^\top W = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0.8825 & 0 & 0 \\ 0 & 0.8825 & 0 \\ 0 & 0 & 0.3247 \end{bmatrix} = \begin{bmatrix} 0.8825 & 0.8825 & 0.3247 \\ 0.8825 & 1.765 & 0.9741 \end{bmatrix}$
2. $X^\top W X = \begin{bmatrix} 0.8825 & 0.8825 & 0.3247 \\ 0.8825 & 1.765 & 0.9741 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 2.0897 & 3.6216 \\ 3.6216 & 7.4862 \end{bmatrix}$
3. $X^\top W \mathbf{y} = \begin{bmatrix} 0.8825\cdot 1 + 0.8825\cdot 2 + 0.3247\cdot 2 \\ 0.8825\cdot 1 + 1.765\cdot 2 + 0.9741\cdot 2 \end{bmatrix} = \begin{bmatrix} 3.2969 \\ 6.3607 \end{bmatrix}$

步骤4：求解 θ

解方程：

$$
\begin{bmatrix} 2.0897 & 3.6216 \\ 3.6216 & 7.4862 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 3.2969 \\ 6.3607 \end{bmatrix}
$$

- 行列式： $\det = (2.0897 \times 7.4862) - (3.6216 \times 3.6216) \approx 2.528$
- 逆矩阵：

$$(X^\top W X)^{-1} \approx \frac{1}{2.528} \begin{bmatrix} 7.4862 & -3.6216 \\ -3.6216 & 2.0897 \end{bmatrix} \approx \begin{bmatrix} 2.960 & -1.432 \\ -1.432 & 0.826 \end{bmatrix}$$

- 计算 θ ：

$$\theta = \begin{bmatrix} 2.960 & -1.432 \\ -1.432 & 0.826 \end{bmatrix} \begin{bmatrix} 3.2969 \\ 6.3607 \end{bmatrix} \approx \begin{bmatrix} 0.65 \\ 0.53 \end{bmatrix}$$

步骤5：带入模型进行预测

$$\hat{y} = \theta_0 + \theta_1 x = 0.65 + 0.53 \times 1.5 = \boxed{1.445}$$

注意事项（AI）

1. **矩阵可逆性**：若 $X^\top W X$ 奇异，奇异不可逆（如局部点共线），需添加正则化项 λI 。
2. **计算效率**：每个查询点需重新计算权重并求逆，复杂度 $O(mn^2 + n^3)$ （ n 为特征数）。
3. **带宽选择**：通过交叉验证优化 τ ：
   - τ 过大 → 欠拟合（接近全局线性回归）
   - τ 过小 → 过拟合（对噪声敏感）
4. **核函数选择**：除高斯核外，也可用Epanechnikov核等。

> **核心思想**：对每个查询点赋予邻近样本不同权重，通过加权最小二乘法拟合局部线性模型，从而灵活捕捉非线性关系。

逻辑回归（Logistic regression）

我需要告诉各位的是，这里所讲的逻辑回归本质上是线性回归的推导，逻辑回归可以看作是线性回归的一种扩展，用于处理分类问题。具体来说，逻辑回归的线性部分（β0+β1X1+β2X2+⋯+βnXn）与线性回归的线性部分相同，但逻辑回归通过一个非线性函数（逻辑函数或sigmoid函数）将线性部分的输出映射到0和1之间（二元），从而得到一个概率值。

到此，我就需要给各位介绍今天的主角，根据线性回归推导出的逻辑回归模型：

$$\boxed{P(y = 1 \mid x; \theta) = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}}$$

其中：

- $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ （ θ 为参数向量）
- $g(z) = \dfrac{1}{1 + e^{-z}}$ 是 **Sigmoid 函数**（或称 Logistic 函数），它将任意实数 z 映射到 $(0, 1)$ 区间

好，现在问题来了，这个公式是怎么来的？直接上推导吧！

逻辑回归模型推导

逻辑回归的假设函数设计基于以下逻辑：

1. 从线性回归到分类问题

- **线性回归**：直接输出连续值 $\theta^T x$ （范围 $(-\infty, +\infty)$）。
- **分类问题**：需要预测离散类别（如二分类中的 0 或 1）。但直接使用线性回归的输出作为概率不合理（概率需在 [0,1] 区间）。

2. 引入概率建模

目标是估计 样本 x 属于类别 1 的概率：

$$P(y = 1 \mid x; \theta) = ?$$

需要将线性组合 $\theta^T x$ （范围 $(-\infty, +\infty)$）映射到 [0,1] 区间。

3. 选择 Sigmoid 函数

Sigmoid 函数 $g(z)$ 是理想选择：

- 当 $z \to +\infty$， $g(z) \to 1$
- 当 $z \to -\infty$， $g(z) \to 0$
- 当 $z = 0$， $g(z) = 0.5$
- 单调连续，便于求导优化。

4. 概率比值的对数（Log Odds）

核心思想：**用线性模型建模“对数几率”**（Log-Odds）。

定义“几率”（Odds）为属于正类的概率与负类概率的比值：

$$\text{Odds} = \frac{P(y = 1 \mid x)}{P(y = 0 \mid x)} = \frac{P(y = 1 \mid x)}{1 - P(y = 1 \mid x)}$$

取对数得到 **Logit 函数**：

$$\log \left(\frac{P(y = 1 \mid x)}{1 - P(y = 1 \mid x)} \right) = \theta^T x$$

5. 反解出概率形式

从 Logit 反推概率表达式：

$$\log\left(\frac{P}{1-P}\right) = \theta^T x \implies \frac{P}{1-P} = e^{\theta^T x} \implies P = \frac{e^{\theta^T x}}{1+e^{\theta^T x}} = \frac{1}{1+e^{-\theta^T x}}$$

最终得到：

$$\boxed{P(y=1 \mid x;\theta) = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}}$$

一些补充：

为什么不用其他函数？

- **Sigmoid 的替代方案**：Probit 函数（基于正态分布），但 Sigmoid 计算更简单且导数易求（毕竟你在前边的二乘也见识过正态分布的公式复杂程度）。
- **与损失函数的关联**：Sigmoid 的导数形式 $g'(z) = g(z)(1-g(z))$ 天然适配交叉熵损失函数的优化（下文提到了什么是交叉损失函数）。

理解假设函数的由来，能更好地掌握逻辑回归如何将线性边界转化为概率预测，以及为何它成为二分类问题的经典方法。

为什么要取对数？

到此我们就知道，**逻辑回归模型Sigmoid是由 线性模型（拟合对数几率） + 取自然对数（映射到连续空间）+ 贝叶斯概率模型（可能不需要）** 推导出来的，但是你们有没有想过，为什么构建时候要取对数？

这个问题可以从公式入手：

- “对数几率”的意义，取对数前本身是关于 1 的连续离散值，但是当取对数之后，我们的值就变成了：
  - 当 Odds > 1（P>0.5），Logit > 0
  - 当 Odds = 1（P=0.5），Logit = 0
  - 当 Odds < 1（P<0.5），Logit < 0
- YES，关于0的连续离散值。于是我们的 Logit 的取值范围就变成了 (-∞, +∞)！这是一个不受限制的连续值

拓展：Sigmoid求导性质

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} && (1)\\
&= \frac{1}{(1+e^{-z})^2}(e^{-z}) && (2)\\
&= \frac{1}{1+e^{-z}}\cdot\left(1-\frac{1}{1+e^{-z}}\right) && (3)\\
&= g(z)(1-g(z)) && (4)
\end{aligned}
$$

浅浅来一下深入：逻辑回归推随机梯度上升（Stochastic gradient ascent）

在确定了模型之后，我们需要找到合适的 θ 的值。这里采用之前使用的最大似然来抉择参数（假设函数可以直接看作概率分布）。

首先，二元分类符合伯努利分布，我们假设：

$$
\begin{aligned}
P(y=1 \mid x;\theta) &= h_\theta(x) && (5)\\
P(y=0 \mid x;\theta) &= 1-h_\theta(x) && (6)
\end{aligned}
$$

重写上面的公式合二为一，得到：

$$
P(y \mid x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}
$$

假定 m 个样本之间相互独立，我们可以得到参数 θ 的似然函数如下：

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X;\theta) && (7)\\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) && (8)\\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1-h_\theta(x^{(i)})\right)^{1-y^{(i)}} && (9)
\end{aligned}
$$

$$
h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}
$$

逻辑回归公式放这，不然上去翻麻烦（下文（12-15）求导会用）

与之前类似，为了计算方便，我们最大化对数似然函数：

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) && (10)\\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})) && (11)
\end{aligned}
$$

上边两步我们得到**对数似然函数 $\ell(\theta)$（log-likelihood function）。它就是 交叉熵损失（cross-entropy loss） 的负值（缺少负号和平均系数）。机器学习中常定义损失函数为 $J(\theta) = -\frac{1}{m}\ell(\theta)$**

（注：**在信息论当中，我们也可以使用相对熵推导出交叉熵**）。（这部分其实与我们在 线性回归推导、误差分析、最大似然估计 + 正太分布推最小二乘 中推导最小二乘法很相似）。接下来要做的就是找到 θ 使得 $\ell(\theta)$ 最大，找最大值那就求导呗（其实第三步也类似哈），由于这里是找最大值而非最小值，所以使用梯度上升（gradient ascent），参数的更新规则是 $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$，对于随机梯度上升（每次只考虑一个样本），求导过程如下：

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) && (12) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x && (13) \\
&= \left(y(1 - g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j && (14) \\
&= \left(y - h_\theta(x) \right) x_j && (15)
\end{aligned}$$

综上所述，我们得到**随机梯度上升**的更新规则是：

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

和批量梯度的区别在于，这家伙不会在 学习率 α 前乘 $\frac{1}{m}$，在更新的时候不会一股脑梭哈，而是**随机**。

注：上升是 $+\alpha$，下降就是 $-\alpha$，逻辑回归中，交叉熵损失（下降）是最小化负对数似然（正对数是上升，负对数不就是下降了），因此梯度下降和上升本质等价（符号相反罢了）

牛顿法（Newton's Law）

而另外一种算法也可以来求解 $\ell(\theta)$ 的最大值，也就是**牛顿法**。这个方法，你甚至可以在高中数学中的章节最后扩展看到，是一个非常优雅的递归+求导求局部最优解的方法。

事实上这个方法作者本人再熟悉不过了（很久之前用C语言实现过一维版本的

从一维到多维的推导

1. 一维情况（求解方程 $f(x) = 0$）

牛顿迭代核心思想：

用函数的 **切线** 近似代替原函数，用切线的零点逼近原函数的零点。

推导步骤：

1. **初始点**：从初始猜测 x_0 开始。
2. **泰勒展开**：在点 x_n 处对 $f(x)$ 进行一阶泰勒展开（其实这里不使用泰勒展开也可）：

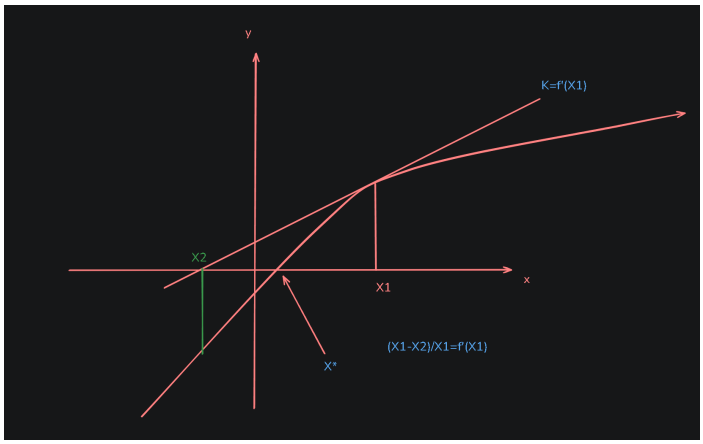
$$f(x) \approx f(x_n) + f'(x_n)(x - x_n)$$

or

$$f'(x_n) = \frac{f(x_n)}{(x_n - x_{n+1})}$$

$$\downarrow$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



3. **求切线零点**：令近似方程等于零：

$$f(x_n) + f'(x_n)(x - x_n) = 0$$

4. **解出下一个迭代点** x_{n+1}：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

5. **迭代**：重复上述过程直至收敛。

几何解释：

在 x_n 处作切线，该切线与 x 轴的交点即为 x_{n+1}（见下图）。

代码块

```
y
|
|    f(x)
|     /|
|    / |
|   /  | 切线: y = f(xₙ) + f'(xₙ)(x - xₙ)
|  /   |
| /    |
|/_____|________ x
      xₙ₊₁  xₙ
```

看一眼，我们的重点不在这里，机器学习大部分都是多维数据量，谁会在一维徘徊?

2. 多维情况（求解方程组 $F(\mathbf{x}) = \mathbf{0}$）

核心思想：

用 **线性逼近** （雅可比矩阵，线性代数）代替原函数，用线性方程组的解逼近原方程组的解。

说白了就是n维代表了n个方程，进行求偏导，代入数据迭代，最终逼近正确解。

推导步骤：

1. **初始点**：从初始向量 $\mathbf{x}_0$ 开始。
2. **泰勒展开**：在点 $\mathbf{x}_n$ 处对向量函数 $F(\mathbf{x})$ 进行一阶泰勒展开：

$$F(\mathbf{x}) \approx F(\mathbf{x}_n) + J_F(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}n)$$

其中 J_F 是雅可比矩阵（Jacobian Matrix），元素为：

$$(J_F)_{ij} = \frac{\partial F_i}{\partial x_j}$$

3. **令线性逼近等于零**：

$$F(\mathbf{x}_n) + J_F(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n) = \mathbf{0}$$

4. **解出下一个迭代点 $\mathbf{x}_{n+1}$**：

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J_F(\mathbf{x}_n)^{-1} F(\mathbf{x}_n)$$

5. **迭代**：重复直至收敛。

下面将多维牛顿法应用于**似然函数**的优化，我们需要找到 $\ell(\theta)$ 的最大值，即 $\ell\prime(\theta) = 0$ 的点，因此令 $f(\theta) = \ell\prime(\theta)$，我们可以得到逻辑回归的牛顿方法更新公式：

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

而对于 θ 为向量的情况，牛顿方法的多维形式如下（又被称为**牛顿-拉夫逊方法**）：

$$\theta := \theta - H^{-1} \nabla_\theta \ell(\theta)$$

这里我需要解释的是，$\ell''(\theta)$ 就是一个 $(n+1) \times (n+1)$ 的 **黑塞矩阵**，也就是**二阶偏导矩阵**，这里**取黑塞矩阵的逆代替分母**了，后边的 $\nabla_\theta \ell(\theta)$ 是**似然函数对于每个 θi 求一阶偏导数构成的向量**。还是很好理解的。

黑塞矩阵：

1. 原表示：

$$(H_f)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

2. 带入似然函数表示：

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$

感知器学习算法

19世纪60年代，感知器被看作是大脑工作中独立神经元的粗糙的模型，但其实人脑的运作比这个算法要复杂的多。

修改一下逻辑回归方法，“强迫”它输出的值要么是 要么是 。要实现这个目的，很自然就应该把函数 的定义修改一下，改成一个阈值函数，由此得到其假设函数为：

$$h_\theta(x) = g(\theta^T x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 (16) \\ 0 & \text{if } \theta^T x < 0 (17) \end{cases}$$

然后，使用如下参数更新规则（其实就是上文的随机梯度上升，但是 $h_\theta(x)$ 变了）：

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

这便是一个完整的感知器算法。

虽然感知器算法很优秀，但是很难对感知器算法赋予有意义的概率解释，也很难使用最大似然估计算法来推出感知器学习算法。

关键点总结：

情况	迭代公式	矩阵作用
一维求根	$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$	一阶导数
多维求根	$\mathbf{x}_{n+1} = \mathbf{x}_n - J_F^{-1} F(\mathbf{x}_n)$	雅可比矩阵逆
优化（求极小值）	$\mathbf{x}_{n+1} = \mathbf{x}_n - H_f^{-1} \nabla f(\mathbf{x}_n)$	Hessian（黑塞） 矩阵逆

类牛顿法 和 类梯度算法 是有一定区别的。当面对大量的数据时，牛顿法的时间复杂度要比梯度高（因为要处理二阶偏导的黑塞矩阵），但是牛顿法公式收敛速度要比类梯度算法快。所以结论就是：**小规模建议用牛顿，大规模建议用梯度。**