# 机器学习与深度学习

## ——逻辑回归

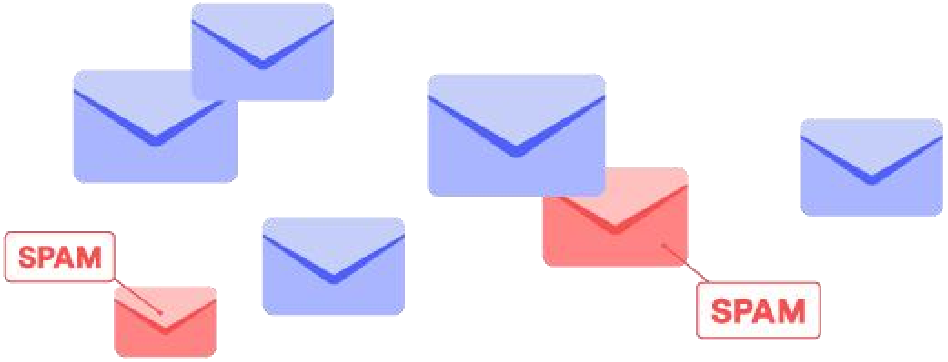
Personal Website: https://www.miaopeng.info/
Email: miaopeng@stu.scu.edu.cn
Github: https://github.com/MMeowhite
Youtube: https://www.youtube.com/@pengmiao-bmm

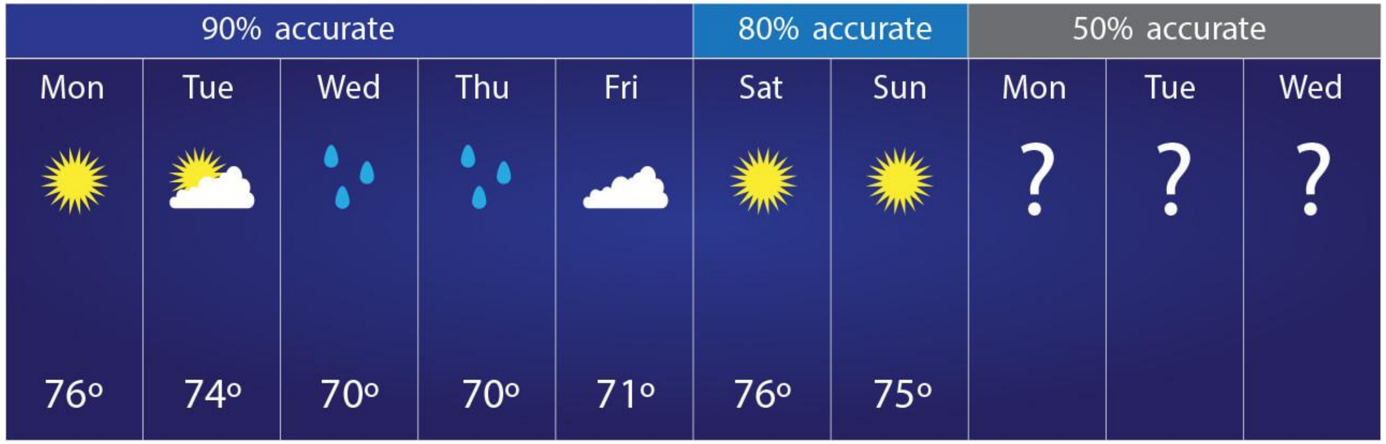
# 目录章节

CONTENTS

# ▶ 为什么学逻辑回归？

➢ 如何预测天气、垃圾邮件识别、疾病诊断？【二分类问题：否/是->0/1】




- 分类任务入门必修：逻辑回归是最经典、最易理解的**二分类**模型（0/1），数学原理清晰，易于初次学习理解。
- 同样通过**回归系数可以解释每个特征对结果的影响大小和方向**，是后续深度模型不可替代的优势。
- 扩展性好，可通过正则化防止过拟合，也能推广到**多分类**（One-vs-Rest、Softmax），同时也为后面学习帮助理解概率预测、最大似然估计、分类决策边界等概念打下基础。

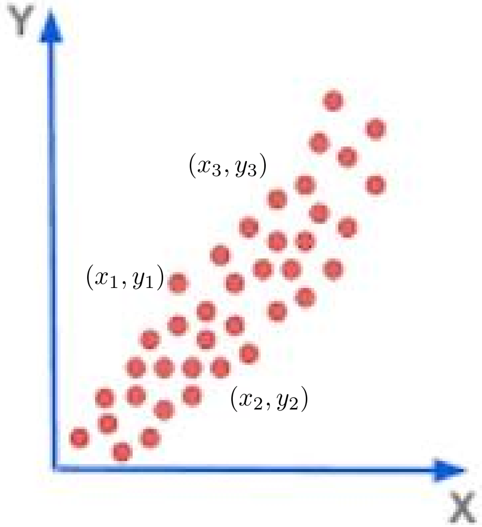
**线性回归是机器学习的第二级台阶。**

# ▶ 什么是逻辑回归？

- 定义：逻辑回归（Logistic Regression）是一种用于分类（尤其是二分类）任务的统计与机器学习模型
- 公式：将输入特征的线性组合 $z = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon$
  - 通过Sigmoid函数映射到(0，1)区间：

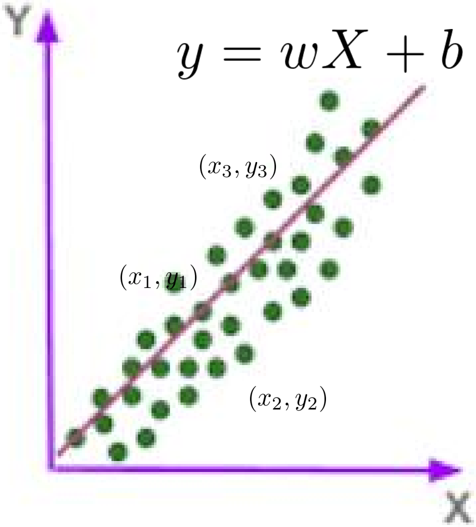
$$\hat{p} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- 决策规则：设定阈值（常用 0.5），如果大于阈值最后输出为1（正类），小于阈值最后输出为0（负类）。

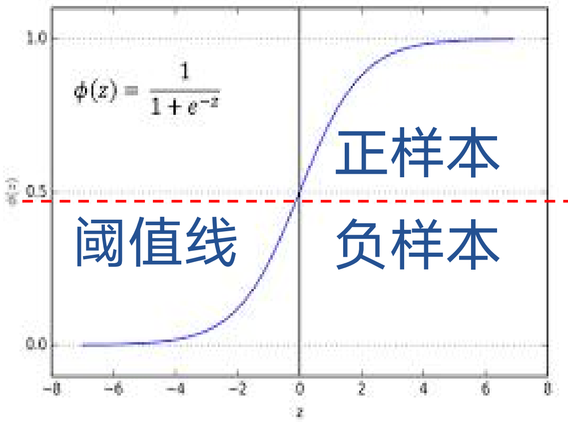



通过所有点的误差($\hat{y}_i - y_i$)的和最小【最小二乘法】

确定参数w和b




映射



# ▶ 逻辑回归的严谨完整形式

➢ 假设我们有一组观测数据，共m个样本，每个样本有n个特征：

- 第i个样本的输入特征为向量：$\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}]^T$
- 对应的目标值为：$y^{(i)}$ 注意y的取值范围为：$y^{(i)} \in \{0,1\}$

➢ 我们先通过一个线性模型来拟合这些数据，然后通过sigmoid函数输出分类的概率：

$$z^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_n x_n^{(i)}$$
$$\hat{p}^{(i)} = \sigma(z^{(i)})$$

注：$z^{(i)}$表示模型对第i个样本的线性输出值，$\beta_i$表示第i个特征的回归系数，$\beta_0$表示截距，$x^{(i)}$表示该样本的第i个特征输入，$p^{(i)}$表示输出概率。

➢ 由于**逻辑回归的输出是一个二元事件发生的概率**，而在概率论里如果响应变量只能取两个值P(y=1|x)=p，P(y=0|x)=1-p，最自然的建模方式就是假设它服从伯努利分布（Bernoulli distribution），它正好描述了单次二元试验中“成功”（1）或“失败”（0）的概率。

- 于是对于m个样本，独立性假设下总似然估计：

$$L(\beta) = \prod_{i=1}^{m} \hat{p}_i^{y_i} (1 - \hat{p}_i)^{1-y_i}$$

- 取对数之后得到对数似然（log-likelihood）：

$$\mathcal{L}(\beta) = \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

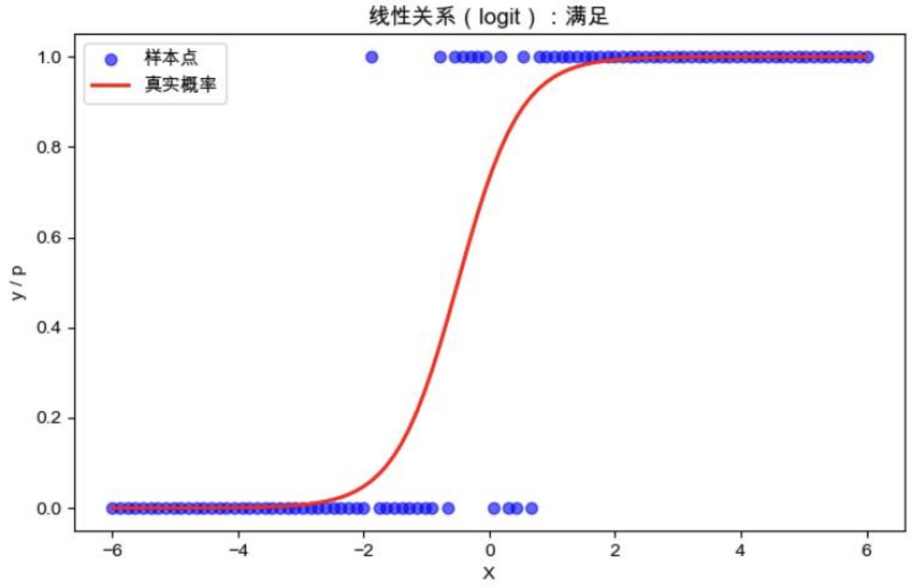
- 训练的目标就是最大化对数似然，等价于最小化负对数似然（常取平均形式），即：

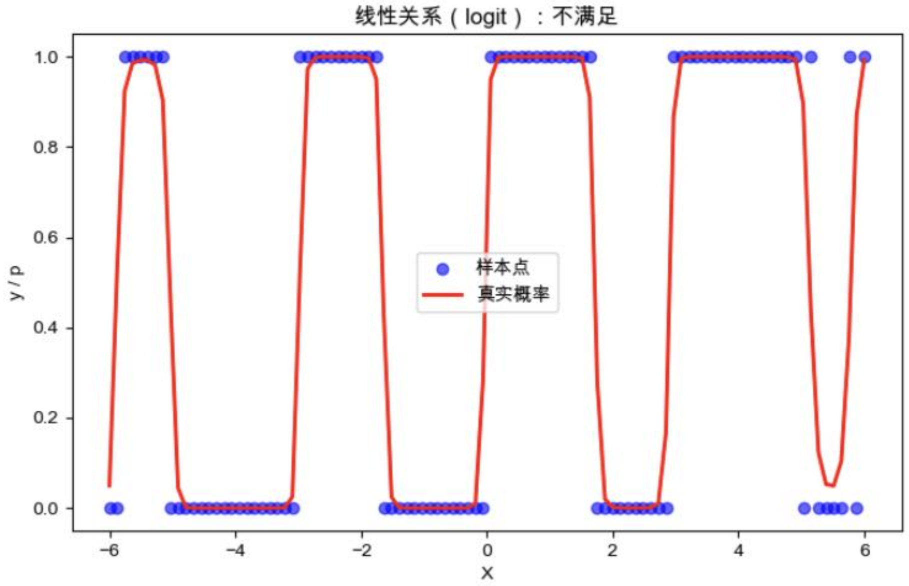
$$\max_{\beta} \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)] \rightarrow \min_{\beta} -\frac{1}{m} \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

# ▶ 模型假设（非常重要！）

➢ 逻辑回归模型的假设类似于线性回归模型的假设，如果不满足以下的假设，那么模型拟合效果肯定不佳：

- 1）**正确的函数形式**：目标变量y的**对数几率**与自变量之间存在线性关系：



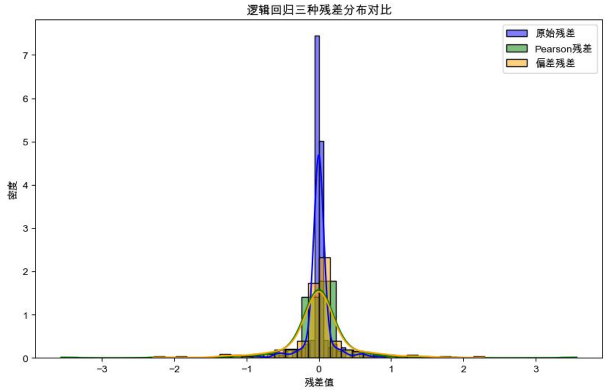

- 根据sigmoid公式，可把概率转换回线性输出，提示需要满足一定的线性关系：

$$\log \frac{P(y=1|X)}{1-P(y=1|x)} = \beta_0 + \beta_1 X_1 + \cdots \beta_n X_n$$

**逻辑回归建立在类似线性回归的假设上，脱离这些假设，模型就失去了它应有的解释力。**

# ▶ 扩展：逻辑回归中的残差

- 在逻辑回归中，残差**不再是“真实值减预测值”这种简单的数值差**，而是对二分类概率预测误差的度量方式。
  - 1）**原始残差（Raw Residual）**：定义为真实类别（0或1）减去预测概率，反映了模型预测概率与实际结果的直接差异。
  - 2）**Pearson残差**：原始残差除以伯努利分布的标准差，用于衡量标准化后的偏差大小，方便比较不同样本的残差。
  - 3）**偏差残差（Deviance Residual）**：基于似然函数推导，用于衡量每个样本对整体模型偏差的贡献，在逻辑回归中更常用，因为它与模型的极大似然估计框架一致。



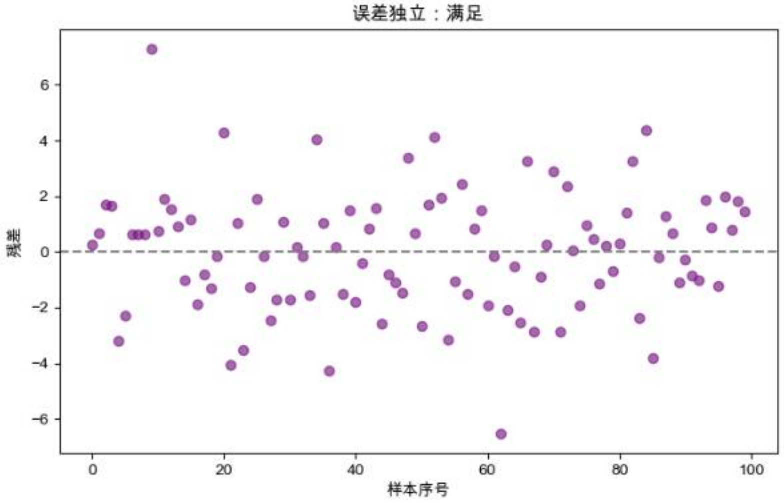

- ◆ 原始残差（蓝色）：是实际标签y与预测概率的差值，分布较为集中，通常在-1到1之间。
- ◆ Pearson残差（绿色）：对原始残差进行了标准化，考虑了概率的方差，分布更宽，能更好反映异常点。
- ◆ 偏差残差（橙色）：基于对数似然损失，分布通常更偏斜，能突出模型拟合不好的样本。

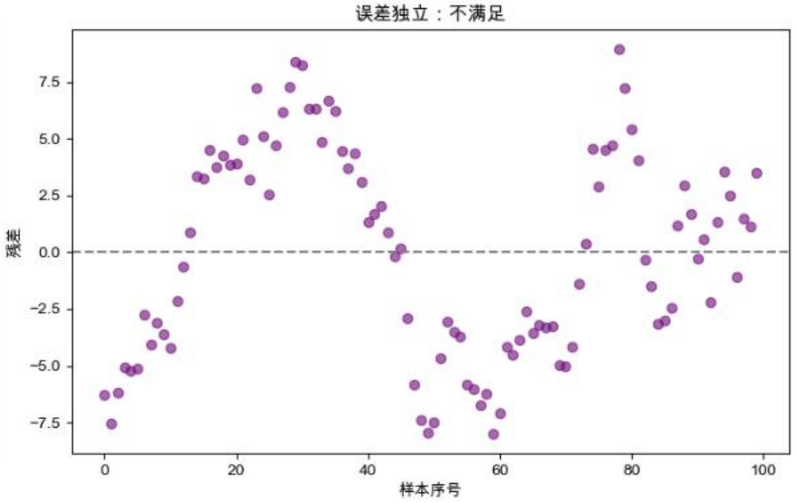
**逻辑回归的残差是对预测概率与真实分类之间差异的度量，可以用原始残差、标准化的Pearson残差或基于似然的偏差残差来表示。**

# ▶ 模型假设（非常重要！）

➢ 逻辑回归模型的假设类似于线性回归模型的假设，如果不满足以下的假设，那么模型拟合效果肯定不佳：

- 2）**误差独立**：残差之间不相关。各个样本的残差（预测误差）彼此之间不应存在系统性关联，也就是说，一个样本的预测误差不应该影响另一个样本的误差。



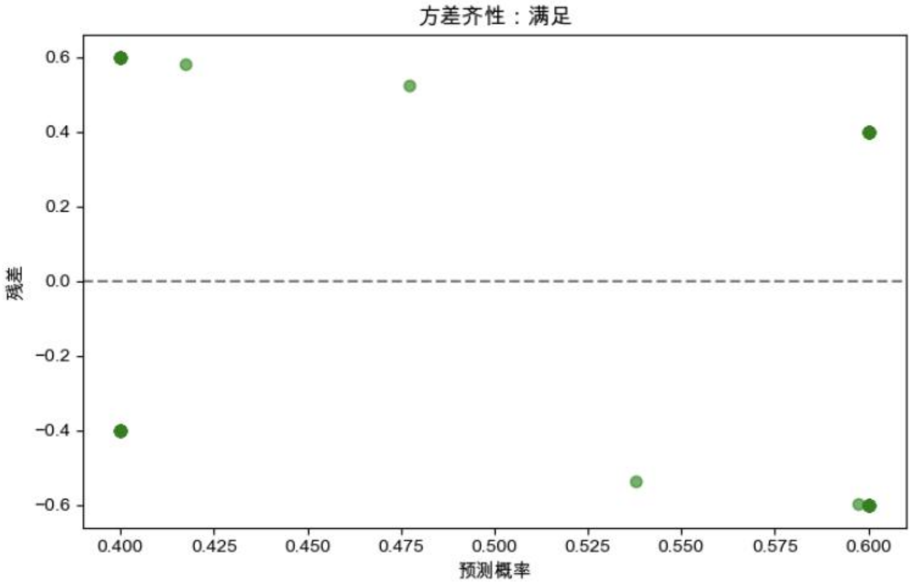

- **时间序列数据**误差就会相互影响，例如：股票价格，气温预测，用户行为数据等。

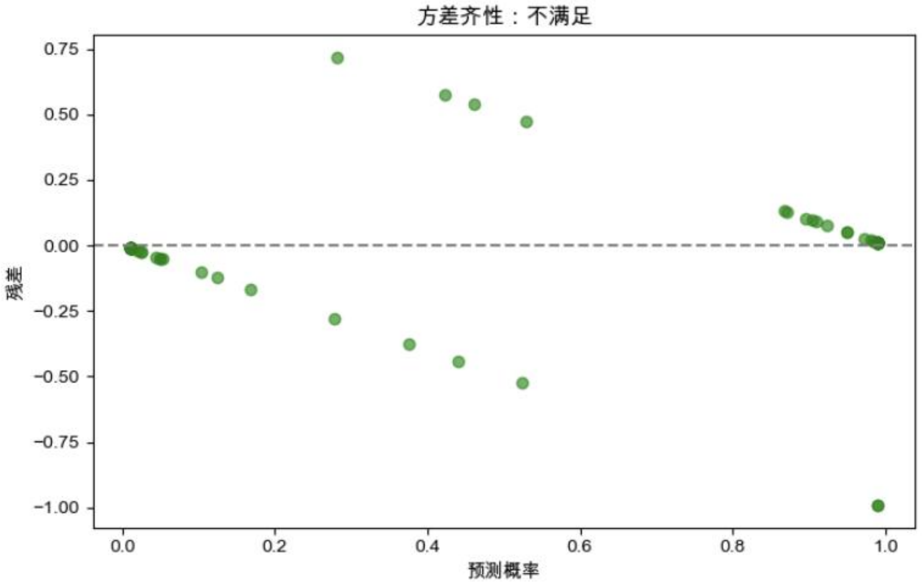
逻辑回归建立在类似线性回归的假设上，脱离这些假设，模型就失去了它应有的解释力。

# ▶ 模型假设（非常重要！）

➢ 逻辑回归模型的假设类似于线性回归模型的假设，如果不满足以下的假设，那么模型拟合效果肯定不佳：

- 3）**方差齐性**：残差方差相等。无论预测值大还是小，模型的误差都应该“差不多大”。



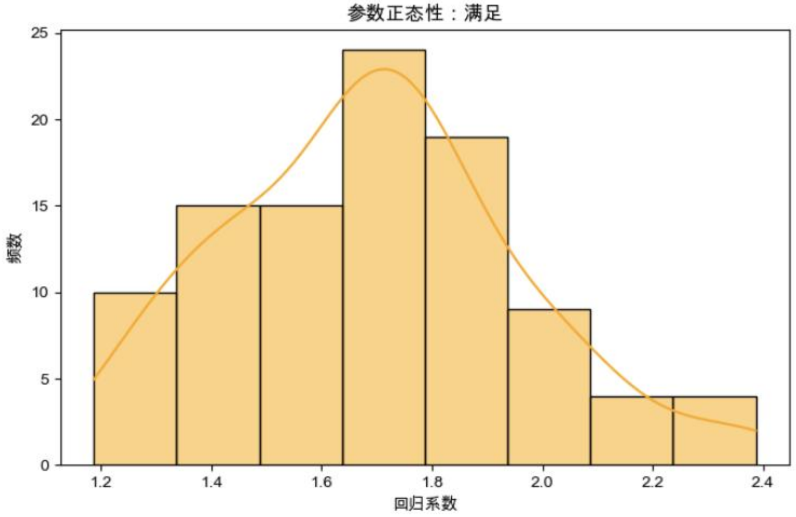

- 模型在所有样本上的预测波动要保持一致，例如右图在预测概率较小的时候残差明显较小，但是预测概率越大的时候残差就较大。

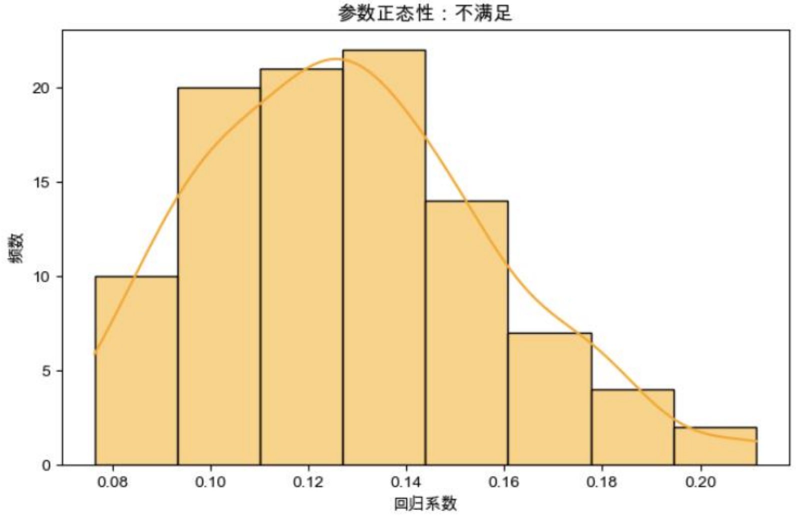
**逻辑回归建立在类似线性回归的假设上，脱离这些假设，模型就失去了它应有的解释力。**

# ▶ 模型假设（非常重要！）

➢ 逻辑回归模型的假设类似于线性回归模型的假设，如果不满足以下的假设，那么模型拟合效果肯定不佳：

- 4）**正态性**：残差服从正态分布。线性回归模型中的残差（error term）应该服从均值为 0 的正态分布。




- 目的：为了推理和检验；提高模型置信度；小样本场景更敏感。

**线性回归建立在一组数学假设上，脱离这些假设，模型就失去了它应有的解释力。**

# ▶ 小结

➢ 为什么需要学习逻辑回归？

- 逻辑回归可作为线性回归的拓展，学习逻辑回归能够帮助我们理解概率建模与分类的基本原理，并为更复杂的机器学习模型打下理论与方法论基础。

➢ 什么是逻辑回归？

- 用是一种用于二分类问题的广义线性模型，它通过对输入特征的线性组合施加 Sigmoid 函数，输出属于某一类别的概率，并以最大似然估计进行参数求解。

$$z^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_n x_n^{(i)}$$
$$\hat{p}^{(i)} = \sigma(z^{(i)})$$

$$\mathcal{L}(\beta) = \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$
$$\max_{\beta} \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)] \rightarrow \min_{\beta} -\frac{1}{m} \sum_{i=1}^{m} [y_i log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

➢ 模型假设：

- 线性可加性：X和log(Y)之间是线性关系。
- 同方差性：误差方差相等。
- 独立性：观测值之间相互独立。
- 正态性：误差服从正态分布。

**线性回归是一种在满足线性、独立、同方差、正态等假设下，用线性方程刻画自变量与因变量关系的基础预测与分析方法。**

# 目录章节

CONTENTS

# ▶ 模型求解的目标：最小化负对数似然

➢ 首先进行数据的统一准备：

- 记训练集（$x_i$,$y_i$），i=1,...,m，每个$x_i \in R^n$，$y_i \in \{0, 1\}$。设计矩阵为：$X \in \mathbb{R}^{m \times n}$
- 线性部分：

$$z_i = x_i^T \beta$$

- Sigmoid映射部分：

$$p_i \equiv \hat{p}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

- 对数似然：

$$\mathcal{L}(\beta) = \sum_{i=1}^{m} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

- 通常将其转换为最小化负对数似然（平均形式或总和形式都可）

$$J(\beta) = -\mathcal{L}(\beta) = -\sum_{i=1}^{m} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

- 求得最小化负对数似然情况下的参数值即为训练的模型：

$$\min_{\beta} -\sum_{i=1}^{m} [y_i \log(p_i) + (1 - y_i) \log(p_i)]$$

# ▶ 模型求解的方法的基础：梯度与Hessian

➢ 首先第一步是求关键导数，是**所有方法的出发点**：

- 对单样本项：

$$\mathcal{L}_i = y_i \log p_i + (1 - y_i) \log(1 - p_i)$$

- 各分量进行求导：

$$\frac{\partial p_i}{\partial z_i} = p_i(1 - p_i) \qquad \frac{\partial z_i}{\partial \beta} = x_i$$

$$\begin{aligned}\frac{\partial \mathcal{L}_i}{\partial \beta} &= \frac{\partial \mathcal{L}_i}{\partial p_i} \cdot \frac{\partial p_i}{\partial \beta} \\ &= y_i \frac{1}{p_i} \frac{\partial p_i}{\partial \beta} + (1 - y_i) \frac{1}{1 - p_i} \frac{\partial (1 - p_i)}{\partial \beta} \\ &= y_i(1 - p_i)x_i - (1 - y_i)p_i x_i \\ &= (y_i - p_i)x_i\end{aligned}$$

➢ 因此总体（负对数似然J的梯度，矩阵形式）：

$$\nabla J(\beta) = -\nabla \mathcal{L}(\beta) = \sum_{i=1}^{m} (p_i - y_i)x_i = X^T(p - y)$$

- X为特征矩阵标签： $X^T \in \mathbb{R}^{n \times m}$
- p为各个输出概率： $\mathbf{p} = (p_1, \cdots, p_n)^T \in \mathbb{R}^{n \times 1}$
- y为列标签： $\mathbf{y} = (y_1, \cdots, y_n)^T \in \mathbb{R}^{n \times 1}$

# ▶ 模型求解的方法：1）梯度下降法

- 目标：使用一阶信息（梯度）做迭代优化。
  - 根据之前梯度推导以及梯度下降更新公式（批量梯度下降）：

$$\beta^{(t+1)} = \beta^{(t)} - \lambda_t \nabla J(\beta^{(t)}) = \beta^{(t)} - \lambda_t \mathbf{X}^T(\mathbf{p}^{(t)} - \mathbf{y})$$

  - 其中：$\mathbf{p}^{(\mathbf{t})} = \sigma(X\beta^{(t)})$ ，为步长（学习率）
- 复杂度：
  - 每次迭代计算 $\mathbf{p}^{(\mathbf{t})} = \sigma(X\beta^{(t)})$ (O(mn))，然后再乘 $\mathbf{X}^T(\mathbf{p} - \mathbf{y})$ (O(mn))，所以复杂度为**O(mn)**
- SGD/mini-batch：用单样本或小批次近似梯度，更新为：

$$\beta \leftarrow \beta - \lambda \sum_{i \in \mathcal{B}} (p_i - y_i) x_i$$

  - 好处是每步成本低，适合极大数据；缺点是噪声、需要学习率调度（如衰减、动量、Adam 等）。
  - 步长选择：固定步长、逐步衰减、或者线性/Armijo/backtracking 行搜索；步长太大会发散，太小会收敛慢。

# ▶ 模型求解的方法：2）牛顿法

➢ 核心思想：**利用二阶信息（Hessian）做二阶泰勒近似，进行快速（二阶）收敛的更新：**

- Newton更新：

$$\beta^{(t+1)} = \beta^{(t)} - H(\beta^{(t)})^{-1}\nabla J(\beta^{(t)})$$

- Hessian（二阶导）可根据一阶导再求导可得：$H(\beta) = \nabla^2 J(\beta) = \sum_{i=1}^{n} p_i(1-p_1)x_i x_i^T = \mathbf{X}^T\mathbf{W}\mathbf{X}$
- 代入：

$$\beta^{(t+1)} = \beta^{(t)} - (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{p}-\mathbf{y})$$

- **IRLS（迭代重加权最小二乘）的等价形式**：定义“工作响度”z：

$$z = \mathbf{X}\beta^{(t)} + \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p})$$

- 则 Newton 更新等价于解加权最小二乘问题：

$$\beta^{(t+1)} = \text{argmin}_\beta \frac{1}{2}(z-\mathbf{X}\beta)^T\mathbf{X}(z-\mathbf{X}\beta)$$

- **其正规方程$X^TWX\beta=X^TWz$而解为$\beta=（X^TWX）^{-1}X^TWz$。把z展开科研算得到与上面Newton更新一致。**

# ▶ 模型求解的方法：3）拟牛顿法

➢ 由于牛顿法需要显式计算并求逆 Hessian，代价大且在高维不实用。拟牛顿法的想法是：用一个容易维护的矩阵Bk（近似Hessian）或它的拟Hk来代替真实Hessian，并逐步更新这个近似矩阵，使其逐步接近真实的Hessian，同时满足重要的约束条件（Secant条件）：

- 由梯度的泰勒展开：

$$\nabla f(x_{k+1}) \approx \nabla f(x_k) + \nabla^2 f(x_k)(x_{k+1} - x_k)$$

- 定义：

$$s_k = s_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

- 若$B_{k+1}$是Hessian的近似，我们希望它满足：

$$B_{k+1} s_k = y_k$$

- 这就是secant方程：表示用$B_{k+1}$能够重现两个点之间梯度的线性变化。对应地，对逆近似$H_{k+1}$则要求：

$$H_{k+1} y_k = s_k$$

- 对更新形式的约束与选择：1）满足secant条件；2）保持对称（Hessian对称）；3）若可能则保持正定（以保证搜索方向为下降方向）；4）更新尽量“最小改变”（即不要无谓地偏离原来信息）

# ▶ 模型求解的方法：3）拟牛顿法

- BFGS更新（Hessian近似$B_{k+1}$的形式）：
  - 标准的BFGS Hessian近似更新为：

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

  - 显然满足对更新形式的约束与选择：即这是对称的；满足secant方程；在$B_k$正定且$y_k^T s_k>0$的情况下，$B_{k+1}$仍是正定的。

$$s_k = s_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

- BFGS 的逆更新（更常用于实现：逆 Hessian $H_k=B_k^{-1}$的更新）：
  - 实际实现里通常维护$H_k$，因为搜索的方向是$-H_k\nabla f_k$。其等价的逆更新公式是：

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_l^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k}$$

  - 这个更新在$H_k$初始为正定时能保持正定性，且是满足secant条件下的“最小修正”解。
- 算法主循环：
  - **1.计算搜索方向$-H_k\nabla f_k$ -> 2.更新参数$\beta$ <- $\beta$ - $H_k\nabla f_k$ -> 3.计算$s_k$，$y_k$并更新$H_{k+1}$，如此循环直至最后的梯度为0。**

# ▶ 扩展：BFGS更新公式推导

- 直观出发点与约束：
  - 我们要维护一个对称矩阵Bk作为 Hessian 的近似（$B_k = \nabla^2 f$）。
  - 在一次迭代中计算出位移$s_k=x_{k+1}-x_k$和梯度增量$y_k=\nabla f(x_{k+1}) - \nabla f(x_k)$。
  - 根据牛顿迭代法，自然要求新的近似Bk+1要求满足secant条件，即：$B_{k+1}s_k = y_k$
  - 同时还希望Bk+1是对称的，并且尽量不偏离之前的Bk（即做尽可能小的修正），且更新秩尽可能低（便于计算）。

- 基于上述，假设Bk+1与Bk的差是秩-2（即两个秩-1矩阵之和）的对称修正，写成两个秩-1矩阵之和为，即设：

$$B_{k+1} = B_k + U + V$$

  - 我们选择U与V为对称形式的秩-1矩阵（即形如$auu^T$），并把自然候选选为：

$$U = a(B_k s_k)(B_k s_k)^T, \quad V = b y_k y_k^T$$

  - 这样即满足了U，V都是堆成且秩至多为1，U+V的秩至多为2。用向量u=Bksk简记，上式变为：

$$B_{k+1} = B_k + auu^T + b y_k y_k^T$$

# ▶ 扩展：BFGS更新公式推导

➢ 下一步我们用secant条件$B_{k+1}s_k=y_k$来解a，b：

- 代入secant条件得到：

$$B_{k+1}s_k + au(u^Ts_k) + by_k(y_k^Ts_k) = y_k$$

- 由于$u=B_ks_k$，因此$u^Ts_k = s_k^TB_ks_k$。记$S=s_k^TB_ks_k$与$Y=y_k^Ts_k$，代入并整理：

$$u + auS + by_kY = y_k \to (1+aS)u + bYy_k = y_k$$

➢ 这是关于向量u和$y_k$的线性组合等于$y_k$。为了让等式在向量空间成立，一个自然且最简单的选择是让系数满足：

- 前项消去：

$$1 + aS = 0 \to a = -\frac{1}{S} = -\frac{1}{s_k^TB_ks_k}$$

- 后项使得：

$$bY = 1 \to b = \frac{1}{Y} = \frac{1}{y_k^Ts_k}$$

- 带回U，V的定义，就得到：

$$\boxed{B_{k+1} = B_k - \frac{B_ks_ks_k^TB_k}{s_k^TB_ks_k} + \frac{y_ky_k^T}{y_k^Ts_k}}$$

# ▶ 扩展：BFGS更新公式推导

- $B_k$是Hessian矩阵（目标函数的二阶导数）的近似矩阵，而$H_k$是$B_k$的逆矩阵，叫做逆Hessian近似矩阵。
  - 利用矩阵求逆更新的Sherman-Morrison-Woodbury公式求$H_{k+1}=B_{k+1}^{-1}$，这个公式指明，对于矩阵更新：

$$A + UCV^T$$

  - 其逆矩阵更新为：

$$(A + UCV^T)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^T A^{-1}U)^{-1}V^T A^{-1}$$

- 把Bk+1写成Bk加上两个秩一矩阵的差：

$$B_{k+1} = B_k + UCV^T$$

  - 其中：

$$U = \begin{bmatrix} y_k & B_k s_k \end{bmatrix} \qquad V = \begin{bmatrix} y_k & B_k s_k \end{bmatrix} \qquad C = \begin{bmatrix} \frac{1}{y_k^T s_k} & 0 \\ 0 & -\frac{1}{s_k^T B_k s_k} \end{bmatrix}$$

  - 套用Sherman-Morrison-Woodbury公式做矩阵运算，经过代数整理之后，得到：

$$\boxed{H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}}$$

# ▶ 模型评估：目标与基本指标

➢ 逻辑回归的评估目标通常是判断它在真实应用中能否可靠地预测，并且能否在不同场景下稳定表现：

- **分类准确性**：确保模型能够正确区分正类和负类。
- **概率预测能力**：不仅分类正确，还能输出可信的概率值（如 0.8 代表“非常可能是正类”）。
- **泛化能力**：在新数据上的表现稳定，避免过拟合。
- **可解释性**：特征系数和概率输出便于业务理解和使用。

➢ 评估的基本指标可分为**分类性能**和**概率质量**两大类：

- 分类性能类指标：包括**准确率**（Accuracy）、**精确率**（Precision）、**召回率**（Recall/Sensitivity）、F1值（Precision和Recall的调和平均）。
- 概率质量类指标：AUC (Area Under the ROC Curve)、ROC曲线、对数损失（Log Loss）、Brier Score。

**逻辑模型评估的核心是验证其分类正确性与概率预测质量，常用指标包括准确率、精确率、召回率、F1、AUC、对数损失等。**

# ▶ 模型评估：分类性能指标

➢ 分类性能类指标主要依赖于混淆矩阵（Confusion Matrix）：

| | 预测正类 | 预测负类 |
|---|---|---|
| 实际正类 | 真阳性，TP | 假阴性，FN |
| 实际负类 | 假阳性，FP | 真阴性，TN |

◆ 1）准确率：

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

● 所有预测中即预测正负类正确的概率。

◆ 2）精确率：

$$Precision = \frac{TP}{TP + FP}$$

● 预测为正的样本中，有多少是真的正类。

◆ 3）召回率：

$$Recall = \frac{TP}{TP + FN}$$

● 实际正类中，有多少被预测对。

◆ 4）F1值：

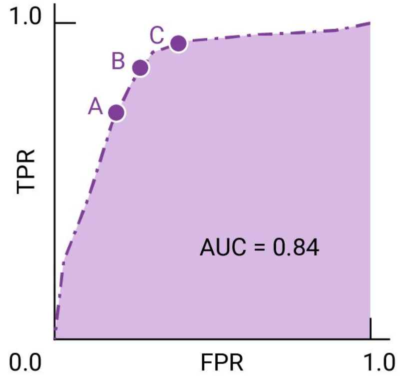
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

● 平衡精确率与召回率。

# ▶ 模型评估：概率质量类指标

➢ 概率质量类指标主要评估概率预测的好坏。

◆ 1）ROC曲线与AUC (Area Under the ROC Curve)：




- 假设判定阈值为t，模型预测概率为p：

$$\mathrm{TPR}(t) = \frac{TP(t)}{P} = P(\hat{p} \geq t | Y = 1)$$

$$\mathrm{FPR}(t) = \frac{FP(t)}{N} = P(\hat{p} \geq t | Y = 0)$$

- ROC曲线函数表达为：

$$\mathrm{ROC} : t \mapsto (\mathrm{FPR}(t), \mathrm{TPR}(t)), \quad t \in [0, 1]$$

- ROC曲线本质上是以假阳性率（FPR）为横轴，真阳性率（TPR）为纵轴绘制的曲线，随着判定阈值（threshold）从高到低变化，计算对应的FPR和TPR点集。
- AUC就是ROC曲线的面积，衡量模型区分正负样本的能力，0.5 为随机水平，1 为完美分类。

◆ 2）对数损失（Log loss）：

$$\mathrm{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

- 概率越接近真实标签，损失越小。

◆ 3）Brier Score：预测概率与真实标签之间的均方误差

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2$$

- Brier Score 越小，表示预测概率越接近真实结果。它综合考虑了概率的校准（预测概率与真实频率的一致性）和判别能力。

# 小结

- 逻辑回归模型的求解主要包含三种方法：梯度下降法、牛顿法、拟牛顿法。
  - 梯度下降法：通过迭代沿着目标函数负梯度方向更新参数，逐步逼近最优解，计算简单但收敛速度较慢。

$$\boxed{\beta \leftarrow \beta - \lambda \sum_{i \in \mathcal{B}} (p_i - y_i) x_i}$$

  - 牛顿法：利用一阶和二阶导数信息直接迭代更新参数，收敛速度快，但需要计算 Hessian 矩阵，代价较高。

$$\boxed{\beta^{(t+1)} = \beta^{(t)} - (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{p} - \mathbf{y})}$$

  - 拟牛顿法：在牛顿法基础上用近似的 Hessian 替代真实二阶导数，兼顾计算效率与收敛速度。

$$\boxed{H_{k+1} = \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k}}$$

**线性回归可用正规方程法或梯度下降法求解，模型好坏可用 MSE、RMSE 衡量误差大小，用 $R^2$ 衡量解释能力。**

# ▶ 小结

➢ 逻辑回归模型的评估主要包含两类指标：分类性能指标和概率质量指标。

| 类别 | 指标名称 | 公式 | 说明 |
|---|---|---|---|
| 分类性能指标 | 准确率 | $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ | 正确预测样本数 / 总样本数，整体正确性。 |
| | 精确率 | $Precision = \frac{TP}{TP+FP}$ | 在预测为正类的样本中，真正为正类的比例。 |
| | 召回率/灵敏度 | $Recall = \frac{TP}{TP+FN}$ | 在实际正类样本中，被正确预测为正类的比例。 |
| | F1分数 | $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$ | 精确率与召回率的调和平均，综合考量模型性能。 |
| | ROC-AUC | $\mathrm{ROC}: t \mapsto (\mathrm{FPR}(t), \mathrm{TPR}(t)), \quad t \in [0,1]$ | 反映模型区分正负样本的能力，曲线下的面积越大越好。 |
| 概率质量指标 | 对数似然 (Log-Loss) | $\mathrm{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$ | 度量预测概率与真实标签的差异，值越小说明概率预测越准确。 |
| | Brier Score | $\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2$ | 预测概率与真实标签之间的均方误差，越小越好。 |

**线性回归可用正规方程法或梯度下降法求解，模型好坏可用 MSE、RMSE 衡量误差大小，用$R^2$衡量解释能力。**

# 目录章节

CONTENTS

# ▶ 模型扩展与改进——正则化

➢ 1）岭回归（Ridge Regression）：

- 在损失函数中加入**L2范数**：

$$J(\theta) = -\mathcal{L}(\theta) + \frac{\lambda}{2}||\theta||_2^2 = -\sum_{i=1}^{m}[y_i \log p_i + (1-y_i)\log(1-p_i)] + \boxed{\frac{\lambda}{2}\sum_{j=1}^{m}\beta_j^2}$$

- 意义：防止过拟合，稳定参数估计，使模型更加平滑。

➢ 2）Lasso 回归（L1正则）：

- 在损失函数中加入**L1范数**：

$$J(\theta) = -\mathcal{L}(\theta) + \lambda||\theta||_1 = -\sum_{i=1}^{m}[y_i \log p_i + (1-y_i)\log(1-p_i)] + \boxed{\lambda\sum_{j=1}^{m}\beta_j}$$

- 意义：增加参数的稀疏性，做特征选择，适合高维稀疏数据。

➢ 3）弹性网回归（Elastic Net）

- **综合 L1 + L2**：

$$J(\theta) = -\mathcal{L}(\theta) + \boxed{\lambda_1||\theta||_1 + \lambda_2||\theta||_2^2}$$

- 意义：结合L1和L2，兼顾特征选择和稳定性。

# ▶ 模型扩展与改进——其他

## ➢ 1）多项逻辑回归（Multinomial Logistic Regression）：

- 假设类别数为K，样本xi属于第k类的概率为：

$$P(Y_i = k|\mathbf{x}_i) = \frac{\exp(\mathbf{x}_i^T \beta_k)}{\sum_{l=1}^{K} \exp(\mathbf{x}_i^T \beta_l)}, \quad k = 1, \cdots, K$$

- 其中通常固定K=0作为基准类。
- 对应的对数似然为：

$$\mathcal{L}(\beta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{1}_{\{y_i=k\}} \log P(Y_i = k|\mathbf{x}_i)$$

- 应用场景：多类别预测。

## ➢ 2）分层逻辑回归（Hierarchical Logistic Regression）：

- 对不同层次引入随机效应，假设两层：

$$\text{logit}(P(Y_{ij} = 1)) = \mathbf{x}_{ij}^T \beta + u_j$$

- i表示第j组中的第i个样本。
- $u_j \sim N(0, \sigma_u^2)$为随机效应，捕捉组内相关性
- 参数估计通常用**广义线性混合模型（GLMM）**方法。

# ▶ 模型扩展与改进——其他

- 3）加入交互项和多项式特征：
  - 将原始特征向量x扩展为包含交互项和多项式项的新特征向量z【让逻辑回归（本来是线性决策边界）能够捕捉非线性关系和变量之间的相互作用。】：

$$\mathbf{z} = [x_1, x_2, \cdots, x_d, x_1x_2, x_1^2, \cdots]$$

  - 然后用逻辑回归模型：

$$p = \sigma(\mathbf{z}^T \beta)$$

  - 应用场景：特征之间存在相互作用效应；变量与结果是非线性关系；特征影响在某些条件下才显现；提高线性模型的表现而不换模型。
- 4）加权逻辑回归：
  - 给每个样本一个非负权重$w_i$，加权对数似然为：

$$\mathcal{L}(\beta) = \sum_{i=1}^{m} w_i[y_i \log p_i + (1 - y_i)\log(1 - p_i)]$$

  - 对不同样本赋予不同权重，解决样本不平衡或重要性不同的问题。
  - 应用场景：样本或类别不平衡时采用效果最佳；成本敏感学习；样本代表性不同（抽样或调查数据）；处理数据稀疏但重要的样本。

# ▶ 模型扩展与改进——其他

➢ 5）**在线学习和增量训练**

- 适合大规模数据和流数据，模型能持续更新而非一次性训练。
- 通过逐批更新模型参数，实现持续学习，避免一次性加载全部数据，节省内存并适应数据分布变化。
- 应用场景：大规模数据集、持续更新模型等场景。

➢ 6）**模型融合与集成**

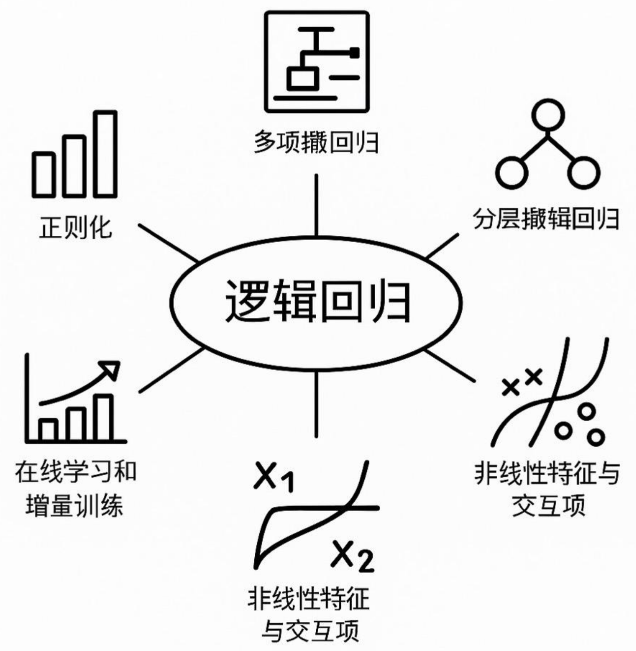
- 将逻辑回归与其他模型（如决策树、随机森林、梯度提升树）结合，提升预测性能。
- 如逻辑回归作为基学习器或用于后续融合。

➢ 7）**广义线性模型（GLM）**

- 逻辑回归是GLM的一种，可以根据需要选择其他连接函数（如Probit回归）。
- 在输入端仍是线性部分$\eta=X\beta$，在输出端用链接函数$g(\mu)=\eta$连接不同分布。
- 常用实例：逻辑回归（Logistic Regression）：二分类；Poisson 回归：计数数据；Gamma 回归：正值连续变量。

# ▶ 小结

- 逻辑回归可通过多种扩展提升性能，包括正则化、多项分类、分层建模、加权训练等，以适应不同数据特征和任务需求。
- 可引入非线性特征、交互项或广义线性模型扩展，增强表达能力并保持可解释性。
- 在大规模与流式数据场景下，可采用在线学习、增量训练或与其他模型融合，提升适应性与预测精度。




**逻辑回归可通过正则化、多类别与分层扩展、加权训练、非线性特征、GLM扩展、模型融合及在线学习等方式增强适应性、表达力与预测性能。**

# 目录章节

CONTENTS

# ▶ 算法实现：numpy实现

➢ 实现：MyLogisticRegression类，第一步：先定义类及初始化方法及内部（私有_）方法。

```python
class MyLogisticRegression:
    def __init__(self, learning_rate=0.01, n_iters=1000,
                 fit_intercept=True, method="gd",
                 alpha=0.0, l1_ratio=0.0):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.fit_intercept = fit_intercept
        self.method = method
        self.alpha = alpha  # 正则化强度
        self.l1_ratio = l1_ratio  # L1占比，0=L2，1=L1
        self.theta = None

    def _add_intercept(self, X):
        if self.fit_intercept:
            intercept = np.ones((X.shape[0], 1))
            return np.hstack((intercept, X))
        return X

    def _sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def _loss(self, theta, X, y):
        z = X @ theta
        y_pred = self._sigmoid(z)
        # 逻辑回归负对数似然
        log_loss = -np.mean(y * np.log(y_pred + 1e-8) + (1 - y) * np.log(1 - y_pred + 1e-8))
        # 正则化项，截距theta[0]不正则化
        reg_l1 = self.alpha * self.l1_ratio * np.sum(np.abs(theta[1:]))
        reg_l2 = 0.5 * self.alpha * (1 - self.l1_ratio) * np.sum(theta[1:] ** 2)
        return log_loss + reg_l1 + reg_l2

    def _gradient(self, theta, X, y):
        z = X @ theta
        y_pred = self._sigmoid(z)
        error = y_pred - y
        grad = (1 / len(y)) * (X.T @ error)
        # 正则化梯度，截距theta[0]不正则化
        if self.alpha > 0:
            # L1梯度用次梯度（符号函数），对theta[0]置0
            grad_reg_l1 = self.alpha * self.l1_ratio * np.sign(theta)
            grad_reg_l1[0] = 0
            # L2梯度
            grad_reg_l2 = self.alpha * (1 - self.l1_ratio) * theta
            grad_reg_l2[0] = 0
            grad += grad_reg_l1 + grad_reg_l2
        return grad
```

- 类定义及初始化：可选参数包括学习率、迭代次数、是否加截距、正则化强度alpha和L1比例l1_ration（Elastic Net）。
- _add_intercept方法（_表示私有）：用于在特征矩阵前加一列1，实现截距。
- _sigmoid方法（_表示私有）：实现sigmoid函数运算。
- _loss方法（_表示私有）：实现损失函数：

$$J(\theta) = -\mathcal{L}(\theta) + \lambda(\alpha||\theta||_1 + \frac{1}{2}(1-\alpha)||\theta||_2^2)$$

- _gradient方法（_表示私有）：实现梯度计算：

$$\nabla J(\theta) = -\nabla\mathcal{L}(\theta) + \lambda(\alpha\text{sign}(\theta) + (1-\alpha)||\theta||)$$

# ▶ 算法实现：numpy实现

➢ 实现：MyLogisticRegression类，第二步：实现核心的模型训练方法：fit(X, y)。

```python
def fit(self, X, y):
    X = np.array(X)
    y = np.array(y)
    X = self._add_intercept(X)
    self.theta = np.zeros(X.shape[1])

    if self.method == "newton":
        for _ in range(self.n_iters):
            z = X @ self.theta
            y_pred = self._sigmoid(z)
            W = np.diag(y_pred * (1 - y_pred))
            # 海森矩阵
            if self.alpha > 0 and self.l1_ratio < 1:
                reg_hessian = self.alpha * (1 - self.l1_ratio) * np.eye(len(self.theta))
            else:
                reg_hessian = 0
            if isinstance(reg_hessian, int):
                H = (1 / len(y)) * (X.T @ W @ X)
            else:
                reg_hessian[0, 0] = 0
                H = (1 / len(y)) * (X.T @ W @ X) + reg_hessian
            H_inv = np.linalg.pinv(H)
            self.theta -= self.learning_rate * (H_inv @ self._gradient(self.theta, X, y))

    elif self.method == "lbfgs":
        from scipy.optimize import minimize
        def loss(theta):
            return self._loss(theta, X, y)
        def grad(theta):
            return self._gradient(theta, X, y)
        result = minimize(loss, self.theta, jac=grad, method='L-BFGS-B')
        self.theta = result.x

    elif self.method == "gd":
        for _ in range(self.n_iters):
            self.theta -= self.learning_rate * self._gradient(self.theta, X, y)

    else:
        raise ValueError("Unsupported optimization method. Use 'newton', 'lbfgs', or 'gd'.")
```

- 数据转换为numpy数组，并加截距项。
- 根据method参数，选择实现牛顿法、BFGS法、梯度下降法。
- 牛顿法根据公式更新：

$$\beta^{(t+1)} = \beta^{(t)} - H(\beta^{(t)})^{-1}\nabla J(\beta^{(t)})$$

- L-BFGS根据公式更新，直接采用封装好的minimize函数：

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}$$

- 梯度下降法更新：

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} - \lambda_t \nabla J(\beta^{(t)}) \\ &= \beta^{(t)} - \lambda_t \mathbf{X}^T(\mathbf{p}^{(t)} - \mathbf{y})\end{aligned}$$

# ▶ 算法实现：numpy实现

➢ 实现：MyLogisticRegression类，第三步：实现核心的模型预测、评估方法。

```python
def predict_proba(self, X):
    X = self._add_intercept(np.array(X))
    return self._sigmoid(X @ self.theta)

def predict(self, X, threshold=0.5):
    proba = self.predict_proba(X)
    return (proba >= threshold).astype(int)

def score(self, X, y):
    y_pred = self.predict(X)
    return np.mean(y_pred == y)
```

- predict_proba方法返回每个样本属于正类（1）的概率。
- predict方法返回每个样本的类别预测（0或1）
- score方法评估模型预测的准确率。

# ▶ 算法实现：pytorch实现（练习）

➤ 利用Pytorch实现关键步骤：

```python
class TorchLogisticRegression(nn.Module):
    def __init__(self, learning_rate=0.01, n_iters=1000,
                 fit_intercept=True, method="gd",
                 alpha=0.0, l1_ratio=0.0, device="cpu"):
        super(TorchLogisticRegression, self).__init__()
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.fit_intercept = fit_intercept
        self.method = method
        self.alpha = alpha  # 正则化强度
        self.l1_ratio = l1_ratio  # L1比例
        self.device = device

        self.theta = None  # 权重参数（包括截距）

    def _add_intercept(self, X):
        if self.fit_intercept:
            intercept = torch.ones((X.shape[0], 1), device=self.device)
            return torch.cat((intercept, X), dim=1)
        return X

    def _sigmoid(self, z):
        return torch.sigmoid(z)

    def _loss(self, y_pred, y, theta):
        pass

    def fit(self, X, y):
        X = torch.tensor(X, dtype=torch.float32, device=self.device)
        y = torch.tensor(y, dtype=torch.float32, device=self.device)

        X = self._add_intercept(X)
        n_features = X.shape[1]

        # 初始化参数
        self.theta = nn.Parameter(torch.zeros(n_features, device=self.device))

        if self.method == "gd":  # 梯度下降
            optimizer = optim.SGD([self.theta], lr=self.learning_rate)

            for _ in range(self.n_iters):
                optimizer.zero_grad()
                y_pred = self._sigmoid(X @ self.theta)
                loss = self._loss(y_pred, y, self.theta)
                loss.backward()
                optimizer.step()

        elif self.method == "lbfgs":  # 拟牛顿 L-BFGS
            optimizer = optim.LBFGS([self.theta], lr=self.learning_rate, max_iter=self.n_iters)

            def closure():
                optimizer.zero_grad()
                y_pred = self._sigmoid(X @ self.theta)
                loss = self._loss(y_pred, y, self.theta)
                loss.backward()
                return loss

            optimizer.step(closure)

        elif self.method == "newton":
            # PyTorch 没有直接封装 IRLS (Newton-Raphson)
            # 这里手写实现：Hessian + gradient
            for _ in range(self.n_iters):
                y_pred = self._sigmoid(X @ self.theta)
                W = torch.diag((y_pred * (1 - y_pred)).detach())
                H = (X.T @ W @ X) / len(y)
                grad = torch.autograd.grad(self._loss(y_pred, y, self.theta), self.theta)[0]
                H_inv = torch.linalg.pinv(H)
                with torch.no_grad():
                    self.theta -= self.learning_rate * (H_inv @ grad)

        else:
            raise ValueError("Unsupported method: use 'gd', 'lbfgs', or 'newton'.")

    def predict_proba(self, X):
        X = torch.tensor(X, dtype=torch.float32, device=self.device)
        X = self._add_intercept(X)
        return self._sigmoid(X @ self.theta).detach().cpu().numpy()

    def predict(self, X, threshold=0.5):
        proba = self.predict_proba(X)
        return (proba >= threshold).astype(int)

    def score(self, X, y):
        y_pred = self.predict(X)
        return (y_pred == y).mean()
```

- 任务：补充方框中的核心代码部分即可。主要实现_loss函数即可，这是核心部分

# ▶ 算法实战：广告点击预测

➢ 如之前一样，首先第一步就是导入数据，并查看数据的情况，决定下一步该干什么。

```python
import pandas as pd

# 读取数据
data = pd.read_csv('advertising.csv')

# 查看信息
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object
 5   City                      1000 non-null   object
 6   Male                      1000 non-null   int64
 7   Country                   1000 non-null   object
 8   Timestamp                 1000 non-null   object
 9   Clicked on Ad             1000 non-null   int64
dtypes: float64(3), int64(3), object(4)
memory usage: 78.3+ KB
```

| | Daily Time Spent on Site | Age | Area Income | Daily Internet Usage | Ad Topic Line | City | Male | Country | Timestamp | Clicked on Ad |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 68.95 | 35 | 61833.90 | 256.09 | Cloned 5thgeneration orchestration | Wrightburgh | 0 | Tunisia | 2016-03-27 00:53:11 | 0 |
| 1 | 80.23 | 31 | 68441.85 | 193.77 | Monitored national standardization | West Jodi | 1 | Nauru | 2016-04-04 01:39:02 | 0 |
| 2 | 69.47 | 26 | 59785.94 | 236.50 | Organic bottom-line service-desk | Davidton | 0 | San Marino | 2016-03-13 20:35:42 | 0 |
| 3 | 74.15 | 29 | 54806.18 | 245.89 | Triple-buffered reciprocal time-frame | West Terrifurt | 1 | Italy | 2016-01-10 02:31:19 | 0 |
| 4 | 68.37 | 35 | 73889.99 | 225.58 | Robust logistical utilization | South Manuel | 0 | Iceland | 2016-06-03 03:36:18 | 0 |

● 可以看见没有null数据，但是有几列的数据类型为object，需要转换为OneHot编码。

# ▶ 算法实战：房价预测

➢ 根据上面的分析，进行数据预处理，并进行数据集划分（训练集、测试集）。

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

# 1.独热编码处理类别变量
encoder = OneHotEncoder(sparse_output=False)
categorical_cols = ['Ad Topic Line', 'City', 'Country', 'Timestamp']
X_categorical = encoder.fit_transform(data[categorical_cols])
data = data.drop(columns=categorical_cols)
data = pd.concat([data, pd.DataFrame(X_categorical)], axis=1)

 # 查看编码后的数据结构
data.info()

# 2. 标签与特征分离
X = data.drop(columns=['Clicked on Ad'], axis=1)
X.columns = X.columns.astype(str)
y = data['Clicked on Ad']

# 3. 分割数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 查看划分后的形状
print(f"X_train shape: {X_train.shape}, X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}, y_test shape: {y_test.shape}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Columns: 3212 entries, Daily Time Spent on Site to 3205
dtypes: float64(3209), int64(3)
memory usage: 24.5 MB
X_train shape: (800, 3211), X_test shape: (200, 3211)
y_train shape: (800,), y_test shape: (200,)
```

- 经过Onehot独热编码之后，新增非常多列，该几列的值全部转换为0/1，并且Null全部被消除，训练机划分数据形状也正常，可以进行下一步。

# ▶ 算法实战：房价预测

➢ 定义模型进行训练、预测，最后通过**准确率、召回率、F1分数等**指标评估效果。

```python
# 使用L1正则化的逻辑回归
from sklearn.linear_model import LogisticRegression as SklearnLogisticRegression
model = SklearnLogisticRegression(penalty='l1', solver='liblinear', C=10.0, max_iter=1000)
model.fit(X_train, y_train)
acc = model.score(X_test, y_test)
print(f"Sklearn L1 Logistic Regression Accuracy: {acc:.4f}\n")

# 使用L2正则化的逻辑回归
model = SklearnLogisticRegression(penalty='l2', solver='lbfgs', C=10.0, max_iter=1000)
model.fit(X_train, y_train)
acc = model.score(X_test, y_test)
print(f"Sklearn L2 Logistic Regression Accuracy: {acc:.4f}\n")

# 使用弹性网正则化的逻辑回归
model = SklearnLogisticRegression(penalty='elasticnet', solver='saga',
                                  l1_ratio=0.5, C=10.0, max_iter=1000)
model.fit(X_train, y_train)
acc = model.score(X_test, y_test)
print(f"Sklearn Elastic Net Logistic Regression Accuracy: {acc:.4f}\n")

# random forest回归
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_acc = rf_model.score(X_test, y_test)
print(f"Random Forest Classifier Accuracy: {rf_acc:.4f}\n")

# xgboost回归
from xgboost import XGBClassifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
                          n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)
xgb_acc = xgb_model.score(X_test, y_test)
print(f"XGBoost Classifier Accuracy: {xgb_acc:.4f}\n")
```

```
Sklearn L1 Logistic Regression Accuracy: 0.9500

/opt/homebrew/anaconda3/envs/ml-dl-fullstack-guide/lib/python3.13/site-pa
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
  n_iter_i = _check_optimize_result(
Sklearn L2 Logistic Regression Accuracy: 0.9300

/opt/homebrew/anaconda3/envs/ml-dl-fullstack-guide/lib/python3.13/site-pa
  warnings.warn(
Sklearn Elastic Net Logistic Regression Accuracy: 0.4450

Random Forest Classifier Accuracy: 0.9400

/opt/homebrew/anaconda3/envs/ml-dl-fullstack-guide/lib/python3.13/site-pa
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost Classifier Accuracy: 0.9350
```

- 可见L1正则化对于该任务的提升效率不大，而比较复杂的模型的拟合效果较好。

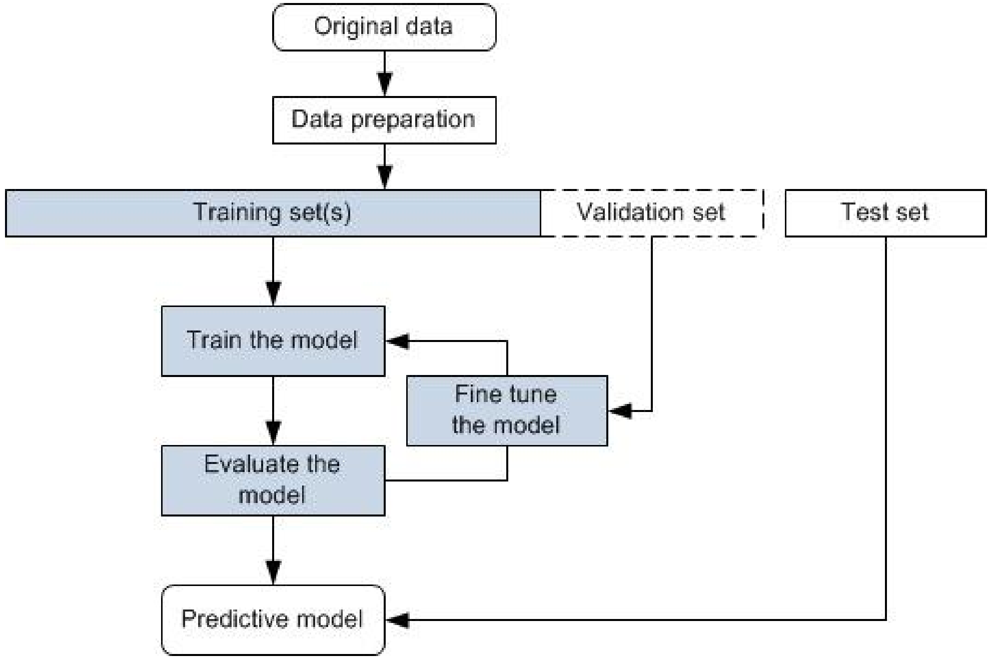
# ▶ 小结

➢ 模型实现：

- 主要的方法：初始化参数（__init__），拟合方法（fit），预测方法（predict），评估指标（score），实现这些方法的私有方法（_xxx）

➢ 模型实战：

- 数据准备：导入数据、清洗缺失值、特征选择/编码。
- 数据划分：训练集 / 测试集（常用70%/30%或80%/20%）。
- 特征预处理：标准化/归一化/去NULL值/独热编码等。
- 模型训练：1）梯度下降法；2）牛顿法；3）拟牛顿法等。
- 模型评估：通过准确率，精准率，召回率，F1分数等进行评估。




**逻辑回归类的实现核心是完成参数初始化、在 fit() 中通过不同方法求解模型系数、在 predict() 中计算预测概率和类别标签，通过分类评价指标进行评估。**

# 目录章节

CONTENTS

# ▶ 总结

➢ **逻辑回归的概念与原理**

✓ 逻辑回归用于处理分类问题，本质是通过Logistic 函数（Sigmoid）将线性组合映射到 [0,1] 的概率空间。

✓ 其核心思想建立自变量与因变量属于某一类别的对数几率（logit）之间的线性关系。

✓ 模型假设：1）线性关系（自变量与 logit）；2）独立性；3）方差齐性；4）正态分布。

➢ **逻辑回归的求解与评估**

✓ 逻辑回归的参数求解常用三种方法：1）梯度下降；2）牛顿法（IRLS）；3）拟牛顿法（如 BFGS）。评估指标分为两类：分类性能指标：准确率（Accuracy）、精确率（Precision）、召回率（Recall）、F1-score；概率质量指标：ROC-AUC、对数似然（Log-Likelihood）、Brier Score 等。

➢ **逻辑回归的扩展与改进**

✓ 线性回归的扩展与改进主要有：1）添加正则项；2）多分类逻辑回归（One-vs-Rest，Softmax 回归）；3）针对特殊需求发展变种（加权、多项式、在线增量、GLM）等。

**逻辑回归通过 sigmoid 函数将线性回归输出映射为概率，实现分类预测，参数可用梯度下降或迭代加权最小二乘法求解，并通过分类与概率指标评估性能，同时可扩展至正则化、多分类与非线性场景以增强适用性。**

# 感谢聆听

Personal Website: https://www.miaopeng.info/

Email: miaopeng@stu.scu.edu.cn

Github: https://github.com/MMeowhite

Youtube: https://www.youtube.com/@pengmiao-bmm