

回归 (Regression) 和分类 (Classification) .....	6
什么是 Gradient Descent (梯度下降) ? .....	7
数学公式 (最核心的一行) .....	7
梯度 = 多个偏导数组成的向量 .....	7
为什么“朝向最低点”也可能是错误的方向? .....	8
所以正确的“下降方向”要满足: .....	9
如何求导数? .....	9
什么是 K-Nearest Neighbors? .....	10
KNN 的优缺点 .....	11
优点: .....	11
缺点: .....	11
什么是 Linear Squares Regression? .....	12
图示理解 .....	12
应用场景: .....	12
最小二乘法的核心思想 .....	12
方法一: 解析解 (闭式解) .....	13
方法二: 梯度下降 (Gradient Descent) .....	13
反积分/不定积分: 用导数推原目标方程 .....	14
向量函数和矩阵函数的反积分 (机器学习中常见) .....	15
性最小二乘回归 (Linear Least Squares Regression) 中 L1 和 L2 regularization 正则化 ...	15
什么是正则化? .....	15
L2 Regularization (也叫 Ridge Regression) .....	16
L1 Regularization (也叫 Lasso Regression) .....	17
通常我们应该 exclude (排除) bias term (偏置项) 不参与正则化 .....	18
什么是 Logistic Regression (逻辑回归) ? .....	19
应用场景 .....	19
总结流程: .....	20
最大似然估计 (Maximum Likelihood Estimation, MLE) .....	20
为啥要取对数 (Log-Likelihood) ? .....	20
Logistic Regression 多类分类 (Multinomial Classification) .....	22
Softmax Logistic Regression (多项式逻辑回归) .....	23

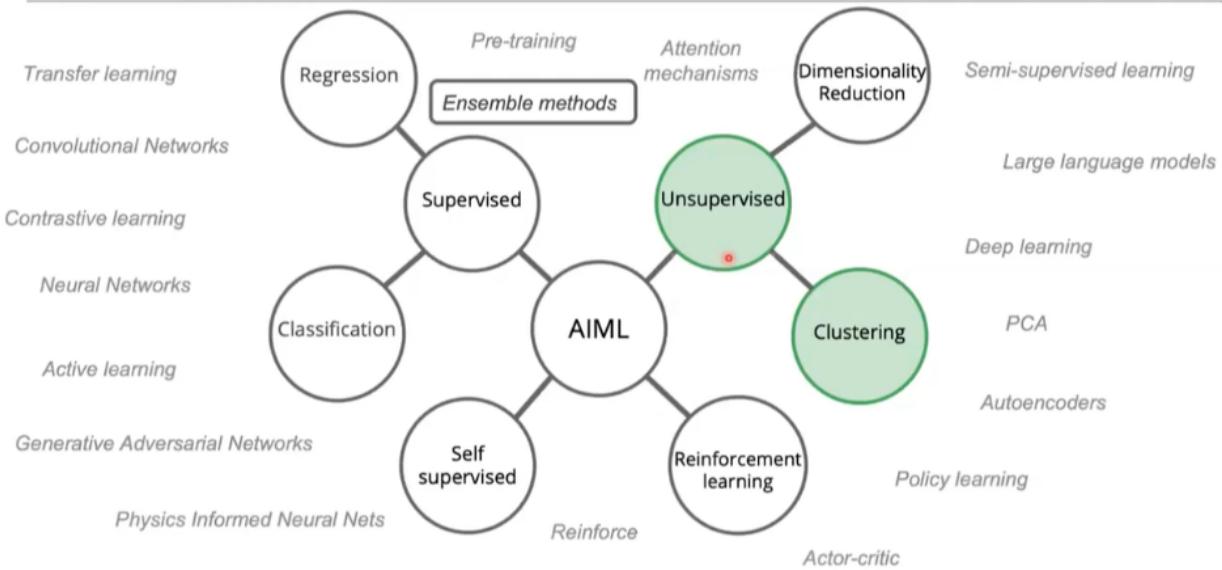
什么是 Logits? .....	24
Support Vector Machine (支持向量机, 简称 SVM) .....	24
决策树的构建核心——如何分裂节点? .....	25
1. 决策树回归 (Decision Tree Regression) .....	27
<span style="color: green;">✓</span> 基本定义.....	27
<span style="color: blue;">💡</span> 工作原理: .....	27
2. 自助聚合 (Bootstrap Aggregation, Bagging) .....	28
3. 随机森林回归 (Random Forest Regression) .....	28
1. MDI — Mean Decrease in Impurity (平均不纯度减少) .....	29
2. PFI — Permutation Feature Importance (特征置换重要性) .....	30
Feature Engineering .....	30
Feature Scaling.....	30
小结建议.....	31
Feature Filtering (特征筛选) .....	31
Neural Network 简称 NN .....	32
核心结构组成.....	32
神经网络如何学习.....	33
神经网络如何“学习” .....	37
那为什么还要激活函数 (Activation Function) ? .....	38
直观例子.....	39
<span style="color: red;">1</span> 没有激活函数: .....	39
<span style="color: red;">2</span> 有 ReLU 激活函数: .....	39
<span style="color: red;">3</span> 有 Sigmoid 激活函数: .....	39
Model Training .....	41
Bias–Variance tradeoff (为何有权衡) .....	42
实践中的常见技巧与影响因素.....	42
如何在实践中估计 bias 和 variance? .....	42
Model Training .....	43
K-Fold 交叉验证 (K-Fold Cross Validation).....	43
模型选择 (Model Selection) .....	43
(2) 调参 (Hyperparameter Tuning) .....	44
Unsupervised learning (无监督学习) .....	44

2. 无监督学习能做什么? .....	44
① Clustering (聚类) —— 将数据分组 .....	44
② Dimensionality Reduction (降维) —— 将数据压缩到更少维度.....	45
③ Density Estimation (密度估计) .....	45
④ Association Rule Learning (关联规则) .....	45
K-means 聚类算法怎么工作? .....	46
K-means 的工作流程 .....	46
1. 选择簇的数量 $k$ .....	46
2. 随机初始化 $k$ 个聚类中心 (centroids) .....	46
3. 分配步骤: 将每个样本分配到最近的中心.....	46
4. 更新步骤: 重新计算每个簇的中心.....	47
5. 重复步骤 3 和 4, 直到收敛.....	47
直观理解.....	47
算法的优缺点.....	47
优点.....	47
缺点.....	47
层次聚类 (Hierarchical Clustering) 是什么? .....	48
Dimensionality Reduction (降维) 是什么? .....	49
🌈 为什么需要降维? .....	49
① 可视化 (最常见) .....	49
② 减少噪音, 提高模型效果.....	50
③ 加速模型训练.....	50
④ 缓解过拟合.....	50
🔥 降维方法总结 (核心方法) .....	50
A 类: 线性降维.....	50
⭐ 1. PCA (Principal Component Analysis) .....	50
原理 (直观版) .....	50
特点.....	51
面试最爱的点.....	51
PCA 的标准步骤 (8 步) .....	51
① 数据标准化 (可选) .....	51

② 计算均值向量 $\mu$ .....	51
③ 数据中心化.....	52
④ 计算协方差矩阵 $S$ .....	52
⑤ 求协方差矩阵的特征值与特征向量.....	52
⑥ 按特征值大小排序.....	52
⑦ 选择前 $k$ 个主成分（降维） .....	53
⑧ 将数据投影到新轴上.....	53
★ 最终结果.....	53
1. Orthogonality with $e_1$ .....	54
2. Orthogonality with $e_2$ .....	54
3. Unit normalization .....	55
★ 2. LDA (Linear Discriminant Analysis) .....	56
原理.....	56
B 类: 非线性降维 (Manifold Learning) .....	56
★ 1. t-SNE (t-distributed Stochastic Neighbor Embedding) .....	56
原理 (直观) .....	56
特点.....	57
一、t-SNE 是什么? .....	57
★ 2. UMAP (Uniform Manifold Approximation and Projection) .....	57
优点.....	57
C 类: 稀疏/特征选择类降维.....	57
★ 1. Feature Selection.....	58
★ 2. Autoencoder (深度学习降维) .....	58
原理.....	58
如何选择方法? .....	58
一个简单直观例子 (超好懂) .....	58
PCA 做的事: .....	58
t-SNE 做的事: .....	59
Autoencoder 做的事: .....	59
什么是 Ensemble Learning? .....	59
为什么 Ensemble 能提高准确率? .....	59

🔧 Ensemble 的三大经典方法.....	60
① Bagging (Bootstrap Aggregating) — "多数投票" 系列.....	60
② Boosting — 逐个纠错，让模型越来越强.....	61
③ Stacking (Stacked Generalization) — “模型的模型”.....	61
⌚ 工程师视角下的优缺点（你会非常受用）.....	62
✓ 优点.....	62
✗ 缺点（也是 AI Infra 要考虑的）.....	63
📍 一个简单例子，让你完全理解.....	63
⌚ 总结（最重要的记住这几点）.....	64

# Ontology



## 回归 (Regression) 和分类 (Classification)

在人工智能 (AI) 和机器学习中，**回归 (Regression)** 和 **分类 (Classification)** 是两种常见的监督学习任务。它们的主要区别在于预测目标的类型。

项目	回归 (Regression)	分类 (Classification)
目標类型	连续值 (实数)	离散类别 (标签)
举例	预测房价、股票价格、温度等	识别猫/狗、是否 (离散标签)、垃圾邮件识别等
输出	实数值, 如 123.45	类别标签, 如 "狗"、"猫" 或 0/1
常见评估指标	均方误差 (MSE)、平均绝对误差 (MAE)、R <sup>2</sup> 等	准确率、精确率、召回率、F1 分数等

- **回归模型示例：**
  - 线性回归 (Linear Regression)
  - 决策树回归 (Decision Tree Regressor)
  - 支持向量回归 (SVR)
- **分类模型示例：**
  - 逻辑回归 (Logistic Regression) (名字叫回归, 其实是分类)
  - 支持向量机 (SVM)
  - 决策树分类器 (Decision Tree Classifier)

- 神经网络分类器（如 Softmax 输出）

小技巧：如何判断是回归还是分类？如果你预测的目标是：

- 一个 **具体数值** → 回归
- 一个 **标签、类别** → 分类

## 什么是 Gradient Descent (梯度下降) ？

梯度下降是一种用于优化机器学习模型参数的算法。它的目标是：

**逐步减少损失函数 (Loss Function)** 的值，直到找到一个最小值（最优解）。

想象你站在一座山上（损失函数的高点），天气太黑你看不到山下，但你可以：

- 每次往**最陡的方向**往下走一步（这一步就是“梯度”）
- 不停走，直到你走到**山谷底部**（即损失最小）

这个过程就叫做 **梯度下降 (Gradient Descent)**

## 数学公式（最核心的一行）

更新参数  $\theta$  的方式是：

$$\theta = \theta - \alpha \cdot \nabla L(\theta)$$

- $\theta$ : 模型的参数（如权重 w、偏置 b）
- $\alpha$ : 学习率 (learning rate) —— 控制每一步走多远
- $\nabla L(\theta)$ : 损失函数对参数的**导数**（梯度）

## 梯度 = 多个偏导数组成的向量

梯度  $\nabla f(x)$  是一个向量，它有两个属性：

属性	含义	类比
方向	函数在该点增长最快的方向	“你朝哪个方向爬山最快”
大小 (模长)	在这个方向上函数上升的速度	“你爬得有多陡”

如果  $\theta$  是一个向量（比如模型有多个参数），那么：

$$\nabla J(\theta) = \left[ \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \dots, \frac{\partial J}{\partial \theta_n} \right]$$

这是一个梯度向量，每个分量表示：对某个参数单独微调，损失函数会怎么变。

**梯度 ( $\nabla L(\theta)$ ) 是使函数值 (损失) 增加最快的方向。** 在梯度下降中，我们关心的是相反的方向（负梯度就是 gradient descent）：**负梯度方向 ( $-\nabla L(\theta)$ ) 是函数值减小最快的方向**，

所以我们沿着这个方向去优化模型参数，逐步让损失最小。用一句话解释梯度的“方向”意义：梯度就像“指南针”，指向函数上升最快的方向，而我们走“负梯度方向”，就是下山最快的路。

- **方向**：告诉你该往哪调参数
- **长度 (大小)**：告诉你函数变化有多快 (斜率多陡)

当你走得越靠近最优点 (函数极小值) 时：

- 梯度的大小变小 (趋近于 0)
- 表示：已经快到底部，下降变慢 → 收敛

**Gradient Descent 最陡下降方向** 永远是损失函数的**负梯度向量**： $-\nabla J(\theta)$   
它总是和等高线 (contour lines) 垂直。

为什么“朝向最低点”也可能是错误的方向？

有时候，一个方向看起来是“朝向最小值的方向”，  
但它不是当前位置下降最快的方向。

举个比喻：

- 想象你站在一个斜坡上，坡底在远处。
- **最陡的方向**是你脚下直接向下的方向 (负梯度方向)。

- 虽然坡底在远方斜对角线方向，但我们必须一步步按“最陡”走，不是直接跳过去。

所以正确的“下降方向”要满足：

- 是 **负梯度方向**:  $-\nabla J(\theta)$
- 与当前点的 **等高线垂直**
- 不一定正对着最低点，但一定是**局部最快下降方向**

如何求导数？

你可以记住一些常用的导数公式：

函数 $f(x)$	导数 $f'(x)$
$x^n$	$nx^{n-1}$
$\sin x$	$\cos x$
$\cos x$	$-\sin x$
$e^x$	$e^x$
$\ln x$	$\frac{1}{x}$

- 常数乘法法则：

$$\frac{d}{dx}[c \cdot f(x)] = c \cdot f'(x)$$

- 和/差法则：

$$\frac{d}{dx}[f(x) \pm g(x)] = f'(x) \pm g'(x)$$

- 乘法法则（积法则）：

$$\frac{d}{dx}[f(x) \cdot g(x)] = f'(x)g(x) + f(x)g'(x)$$

- 除法法则（商法则）：

$$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

- 链式法则（复合函数）：

$$\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$$

Formula type	Formula
$\cos(a + b)$	$\cos a \cos b - \sin a \sin b$
$\cos(a - b)$	$\cos a \cos b + \sin a \sin b$

梯度下降是一种**优化算法**，不是模型本身，用于**最小化损失函数**，帮助训练模型，比如线性回归、逻辑回归、神经网络等。

应用场景	描述
深度学习（神经网络）	所有神经网络模型的训练都依赖梯度下降来优化权重。
线性/逻辑回归	拟合数据时使用梯度下降优化模型参数。
自然语言处理（NLP）	训练语言模型、词向量等。
计算机视觉（CV）	图像识别、目标检测中训练 CNN 时使用。
推荐系统	训练矩阵分解模型、因子分解机等时优化目标函数。
强化学习	用于策略梯度、值函数优化等。

## 什么是 K-Nearest Neighbors？

KNN 是一种**监督学习算法**，用于分类或回归，核心思想是：

“看你周围的邻居是谁，你就可能是谁。”

- 给定一个样本点（测试点）
- 找到训练集中距离它最近的 K 个样本
- 观察这 K 个样本的“标签”：
  - 分类：多数票决定类别
  - 回归：平均值作为预测值

通常用**欧几里得距离**（Euclidean Distance）：

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \dots}$$

但也可以用**曼哈顿距离**、**余弦距离**等。

类型	方法	输出
分类	多数邻居投票	类别标签
回归	邻居的平均值或加权平均	数值结果

超参数 K 的选择很重要

- K 太小 → 对噪声太敏感，容易过拟合
- K 太大 → 平滑但可能欠拟合

常用的做法是用交叉验证选一个最好的 K。

## KNN 的优缺点

优点：

- 非常简单，容易理解
- 无需训练过程（懒惰学习）
- 对多分类问题很自然

缺点：

- 大数据时速度慢（预测时需要计算所有距离）
- 特征尺度敏感（需要标准化）
- 维度灾难：高维数据时效果差

KNN 是一个\*\*懒惰学习（lazy learning）\*\*的分类/回归算法。预测时直接计算与训练数据的距离来决定结果。

应用场景	描述
小样本学习	数据量不大时效果较好，不需要模型训练。
图像识别	图像特征向量之间的相似度判断。
文本分类	使用词向量或 TF-IDF 后，进行相似度分类。
推荐系统	基于用户或物品的相似度推荐。
异常检测	找出与大多数点距离较远的点（即离群点）。

# 什么是 Linear Squares Regression?

线性最小二乘回归是一种统计方法，用于拟合一条直线（或更高维的超平面）来描述两个或多个变量之间的关系。目标是让这条线尽可能“贴近”数据点。

## 图示理解

想象你有一些散点图的数据点，你要画一条直线/曲线，使得所有点到直线的垂直距离的平方和最小。这就是最小二乘回归的直观意义。

## 应用场景：

- 根据房屋面积预测价格
- 根据广告费用预测销售额
- 根据温度预测冰淇淋销量

假设我们有一组样本数据点：

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

我们想找一条直线：

$$y = wx + b$$

其中：

- $w$  是斜率 (weight)
- $b$  是截距 (bias)
- $x$  是输入， $y$  是预测输出

## 最小二乘法的核心思想

我们希望找到一组  $w$  和  $b$ ，使得预测值和真实值之间的差距（即误差）**平方和最小**。

这个误差叫做 损失函数 (Loss Function) , 形式如下:

$$L(w, b) = \sum_{i=1}^n (y_i - (wx_i + b))^2$$

这叫做最小二乘损失函数 (Least Squares Loss)

我们的目标是:

$$\min_{w,b} \sum_{i=1}^n (y_i - (wx_i + b))^2$$

## 方法一：解析解（闭式解）

通过求导，解这个最小值问题，可以得到：

$$w = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$b = \bar{y} - w\bar{x}$$

## 方法二：梯度下降 (Gradient Descent)

当变量很多或者用矩阵形式处理时，通常使用梯度下降进行迭代求解。

在线性回归中，我们的目标是最小化以下损失函数 (误差平方和)

$$J(w, b) = \frac{1}{2n} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \mathbf{x}_i + b))^2$$

其中：

- $\mathbf{x}_i$  是第  $i$  个样本的特征向量 (可能有多维)
- $y_i$  是对应的真实值
- $\mathbf{w}$  是权重向量
- $b$  是偏置
- $n$  是样本数量
- 损失函数前的  $\frac{1}{2n}$  是为了在求导时更简洁 (数学方便)

梯度下降算法步骤：

1. 初始化参数：随机设定  $\mathbf{w}, b$
2. 计算梯度（导数）：
  - $\nabla_{\mathbf{w}} = -\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i) \mathbf{x}_i$
  - $\nabla_b = -\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)$
3. 更新参数：
  - $\mathbf{w} := \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}}$
  - $b := b - \alpha \cdot \nabla_b$

其中：

- $\alpha$  是学习率 (learning rate)：决定每次“走多远”
- $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + b$ : 模型预测值
- 4. 重复步骤 2–3，直到损失函数收敛（变化很小）或达到最大迭代次数。

## 反积分/不定积分：用导数推原目标方程

“反积分”的过程在数学上就是求 不定积分（Indefinite Integral），也就是 从导数反推原函数 的过程。

函数 $f(x)$	不定积分 $\int f(x) dx$
$x^n \quad (n \neq -1)$	$\frac{x^{n+1}}{n+1} + C$
$\frac{1}{x}$	$(\ln x  + C)$
$e^x$	$e^x + C$
$a^x$	$\frac{a^x}{\ln a} + C$
$\sin x$	$-\cos x + C$
$\cos x$	$\sin x + C$
$\sec^2 x$	$\tan x + C$
$\frac{1}{1+x^2}$	$\arctan x + C$

其中， $C$  是积分常数，因为不定积分的结果总是差一个常数项。

## 向量函数和矩阵函数的反积分（机器学习中常见）

1. 向量对向量的二次型：

$$\frac{d}{dw} \left( \frac{1}{2} w^T A w \right) = Aw \Rightarrow \int Aw dw = \frac{1}{2} w^T A w + C$$

其中  $A$  是对称矩阵， $w$  是向量。

---

2. 向量的线性形式：

$$\frac{d}{dw} (b^T w) = b \Rightarrow \int b dw = b^T w + C$$

---

3. 欧几里得范数平方：

$$\frac{d}{dw} \left( \frac{1}{2} \|Xw - y\|^2 \right) = X^T(Xw - y)$$

从这也可以反推：

$$\int X^T(Xw - y) dw \stackrel{\downarrow}{=} \frac{1}{2} \|Xw - y\|^2 + C$$

---

## 性最小二乘回归（Linear Least Squares Regression）中 L1 和 L2 regularization 正则化

问题：在 LSR 里当特征太多 or 训练数据太少，会发生什么？

- 模型容易过拟合（在训练集上表现很好，在新数据上表现差）
- 权重  $w$  会变得不稳定（尤其在特征共线性时）

解决方案：加上 正则项（Regularization Term）

### 什么是正则化？

正则化 = 惩罚复杂模型，鼓励简单权重（小且稀疏）

我们在原始损失函数后面加一个“惩罚项”来约束权重  $w$ ，防止它们变得太大或太复杂。

## L2 Regularization (也叫 Ridge Regression)

✓ 惩罚项:

$$\lambda \|w\|_2^2 = \lambda \sum_i w_i^2$$

✓ 完整损失函数:

$$\text{Loss}(w) = \|Xw - y\|^2 + \lambda \sum_i w_i^2$$

解释:

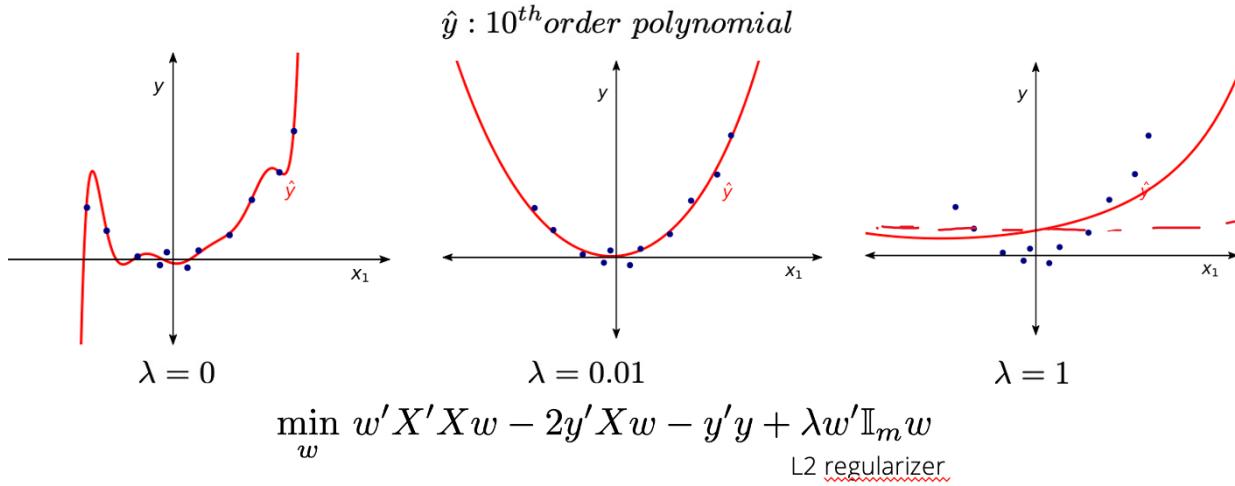
- 惩罚所有权重过大的情况
- 越大的 w, 惩罚越重
- 结果: 权重被压小, 但通常不会完全变成 0
- 有效地降低模型复杂度, 改善泛化能力
- 后边的部分, lambda 部分是一个 circle constraint 因为  $w_i^2$  加总 < 一个 constraint

效果:

- 收缩 (shrinkage) 模型, 但不会让特征“消失”
- 更适用于所有特征都有用, 但不希望它们太激进

Lambda 或者  $w_i$  越大, 后面的权重越大, 是一个固定的数, 前面的 x 和 y 的影响越小。曲线越扁, 因为后边权重是个 constant, 当权重 lambda 无限大时候, 整个曲线最后会变成一个横线。

## LLS with Regularization



## L1 Regularization (也叫 Lasso Regression)

✓ 惩罚项:

$$\lambda \|w\|_1 = \lambda \sum_i |w_i|$$

✓ 完整损失函数:

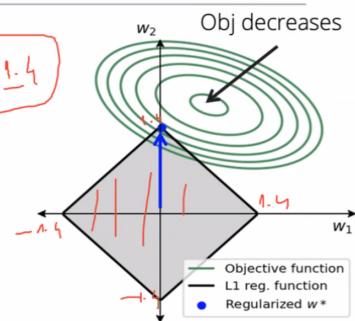
$$\text{Loss}(w) = \|Xw - y\|^2 + \lambda \sum_i |w_i|$$

解释:

- 惩罚权重的“绝对值”
- 结果: 部分权重会被压成 **0**
- 即: 自动选择特征 (特征选择能力)
- 后边的部分, lambda 部分是一个 diamond constraint

## Solving L1 regularization problem

- Obj L1 regression involves an absolute operator
- Lasso Regression does not have closed-form solution
- Lasso cost function is not differentiable at  $w_i = 0$
- $|w_1|$



效果：

- 更适用于数据中存在很多冗余特征时
- 能“丢弃”一些无关特征 → 稀疏模型

特性	L1 正则化 (Lasso)	L2 正则化 (Ridge)
惩罚形式	sum	w_i
结果权重	稀疏（可能为 0）	收缩（趋向于小）
特征选择能力	✓ 自动剔除不重要的特征	✗ 不会将权重设为 0
稳定性	对噪声敏感（不稳定）	更稳定，抗多重共线性
解法	不可导（需迭代优化）	可直接求闭式解（解析解）
适用场景	特征多，但只有少数有用	所有特征都有用，但需抑制过拟合

- 特征非常多，可能很多无用 → L1 (Lasso)
- 想让模型更稳定，避免权重过大 → L2 (Ridge)
- 想要两者兼顾 → 使用 Elastic Net，结合了 L1 + L2

通常我们应该 exclude (排除) bias term (偏置项) 不参与正则化

问题	答案
需要正则化 bias term 吗？	✗ 通常不需要
为什么？	避免不必要的抑制输出偏移；提高模型表现
在代码里需要特别排除吗？	如果手动实现，需要排除；库（如 sklearn）会自动处理

## 1. 偏置项的作用不同于权重项

- 权重项  $w$  控制模型对输入特征的响应；
- 偏置项  $b$  控制模型的输出整体偏移 (intercept)；
- 对偏置项加正则化可能会导致模型在整体输出上出现不必要的偏移。

## 2. 防止欠拟合

如果你正则化偏置项（也就是惩罚它），在某些情况下会导致模型整体能力下降：

- 输出容易被拉向 0；
- 模型欠拟合的可能性增加，特别是在数据不以原点为中心时。

# 什么是 Logistic Regression (逻辑回归) ?

Logistic Regression 是用来做二分类问题的模型，输出一个“属于某个类别的概率”。虽然叫“回归”，但它是一个 **分类算法（特别是二分类）**。之所以叫“回归”，是因为它基于线性回归的思想来建模。

## 应用场景

- 是否是垃圾邮件 (Spam / Not Spam)
- 用户是否会点击广告 (Yes / No)
- 肿瘤是否是恶性 (Malignant / Benign)
- 信贷用户是否会违约 (Default / No Default)

### Step 1：构造一个线性函数

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b = \mathbf{w}^T \mathbf{x} + b$$

这部分跟线性回归一样，表示模型的“加权和输出”。

### Step 2：套上 Sigmoid 函数 (非线性转换)

我们用 sigmoid 把  $z$  映射到  $(0, 1)$  之间：

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

得到的  $\hat{y}$  就是“预测为正类的概率”。

### Step 3：根据概率做分类决策

$$\begin{aligned}\hat{y} \geq 0.5 &\Rightarrow \text{预测为 1 类} \\ \hat{y} < 0.5 &\Rightarrow \text{预测为 0 类}\end{aligned}$$

你也可以设置 0.5 以外的阈值 (threshold) 来自定义敏感性。

模型训练：用交叉熵损失函数 (Cross-Entropy)

Logistic Regression 不用 MSE (均方误差)，而是：

$$L(\hat{y}, y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

解释：

- 当预测和实际一样时，损失小；
- 当预测错得离谱，损失会变得非常大。

训练目标是最小化所有样本的总损失，优化参数  $w$  和  $b$ 。

常见限制：

问题	描述
只能做线性分类	数据需要可用直线（或超平面）分隔
对异常值敏感	极端值会影响模型
特征共线性问题	特征之间强相关可能导致不稳定
不适用于多类别（默认）	不原生支持多分类问题（但可以扩展）

## 总结流程：

1. 准备数据（清洗、标准化）
2. 构造特征矩阵  $\mathbf{X}$  和标签  $y$
3. 初始化权重和偏置
4. 计算线性组合  $z = \mathbf{w}^T \mathbf{x} + b$
5. 应用 sigmoid 函数得到概率
6. 使用 log loss 作为损失函数
7. 用梯度下降优化权重
8. 判断概率是否大于 0.5 进行分类

## 最大似然估计（Maximum Likelihood Estimation, MLE）

Logistic Regression 背后的核心数学思想就是 **最大似然估计（Maximum Likelihood Estimation, MLE）**。最大似然估计的目标是：找到一组参数，使得在这些参数下，训练数据出现的“概率”最大。用你看到的数据，来反过来“倒推”出最可能生成这些数据的参数。

构造似然函数（Likelihood Function）

假设数据集中有  $m$  个样本，那么联合概率（也就是似然函数）是：

$$L(\mathbf{w}, b) = \prod_{i=1}^m \sigma(z_i)^{y_i} \cdot (1 - \sigma(z_i))^{1-y_i}$$

MLE 的目标是：最大化这个似然函数，也就是找一组  $w, b$  让  $L$  最大。

## 为啥要取对数（Log-Likelihood）？

由于乘积不好计算（尤其是梯度优化时），我们通常取对数变成：

$$\ell(\mathbf{w}, b) = \log L(\mathbf{w}, b) = \sum_{i=1}^m [y_i \log(\sigma(z_i)) + (1 - y_i) \log(1 - \sigma(z_i))]$$

这就叫做 **对数似然函数 (Log-Likelihood)**。

最大化 mle 函数，取相反数，用导数求最小。

Minimize  $L$  by setting  $\frac{\partial L}{\partial w} = 0$

$$\frac{\partial L(w)}{\partial w} = \sum_{i=1}^{n=10} (g(w'x^{(i)}) - y^{(i)})x^{(i)}$$

Where  $w = \{w_0, w_1, w_2, w_3\}$ , so we take *feature vector*

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial w_3} = 0$$

:

## ④ L2 正则化的核心思想

| 在损失函数中加入一个项，用于惩罚权重  $w$  的大小（防止过拟合）：

$$L_{\text{reg}}(w) = L(w) + \lambda \cdot \frac{1}{2} \|w\|_2^2$$

其中：

- $L(w)$ : 原始逻辑回归的负对数似然损失 (log loss)
- $\|w\|_2^2 = \sum_{j=1}^d w_j^2$ : 权重向量的 L2 范数平方
- $\lambda > 0$ : 正则化强度 (超参数)

原始逻辑回归损失函数：

$$L(w) = - \sum_{i=1}^n \left[ y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

加上 L2 正则化之后：

$$L_{\text{reg}}(w) = L(w) + \lambda \cdot \frac{1}{2} \sum_{j=1}^d w_j^2$$

| 注意：偏置项  $w_0$  通常不参与正则化，可以单独处理。

## 对损失函数求梯度（含 L2）

假设不对偏置项  $w_0$  正则化：

$$\frac{\partial L_{\text{reg}}}{\partial w_j} = \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)} + \lambda w_j, \quad \text{for } j > 0$$

而：

$$\frac{\partial L_{\text{reg}}}{\partial w_0} = \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)}) \cdot 1$$

$\lambda$ 值	影响
很小（接近 0）	几乎没有正则化 → 模型更容易过拟合
很大	强正则化 → 模型权重被压小，可能欠拟合
合适的值	能够平衡模型的拟合能力和泛化能力

## Logistic Regression 多类分类（Multinomial Classification）

多类分类的三种实现方式：

One-vs-One(一对一)

One-vs-Rest (OvR, 一对其余)

对每一个类都训练一个二分类器，把这个类视为正类，其余所有类为负类。

预测时，所有分类器都对样本打分，选得分最高的类。

优点：简单、可解释

缺点：可能对类不平衡敏感，分类器之间可能冲突

Multinomial (Softmax) 逻辑回归

一次性学习所有类之间的区分，不再是多个二分类器。

输出是一个多项分布（概率向量），使用 **softmax** 函数将结果归一化为概率。

在 `sklearn` 中，使用 `multi_class='multinomial'` 配置。

## Softmax Logistic Regression (多项式逻辑回归)

假设：

- 有  $K$  个类别
- 输入特征为  $\mathbf{x} \in \mathbb{R}^n$
- 每个类别  $k$  都有一个权重向量  $\mathbf{w}_k$

类别  $k$  的得分：

$$z_k = \mathbf{w}_k^T \mathbf{x}$$

Softmax 函数用于归一化：

$$P(y = k | \mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}$$

最终选择概率最大的类作为预测结果。

所以整体含义是：把第  $i$  个类别的“指数得分”除以“所有类别的指数得分之和”，就得到了它在这组类别中被认为是“正确类别”的概率。

假设模型输出的 logits (未归一化得分) 是：

$$\mathbf{z} = [2.0, 1.0, 0.1]$$

计算 Softmax：

1. 先计算指数值：

$$e^{2.0} \approx 7.39, \quad e^{1.0} \approx 2.72, \quad e^{0.1} \approx 1.105$$

2. 求和：

$$7.39 + 2.72 + 1.105 \approx 11.215$$

3. 计算每个类别的概率：

$$\text{Softmax}(z_1) = \frac{7.39}{11.215} \approx 0.659$$

$$\text{Softmax}(z_2) = \frac{2.72}{11.215} \approx 0.243$$

$$\text{Softmax}(z_3) = \frac{1.105}{11.215} \approx 0.098$$

✓ 得到最终概率分布：

$$[0.659, 0.243, 0.098]$$



表示模型认为类别 1 的概率最高。

为什么用指数函数？

因为指数函数会：

放大差距 (得分差距越大，概率差越明显)

保持正数（避免负概率）

这让模型可以“更自信”地区分不同类别。

## 什么是 Logits？

**Logits** 是神经网络或分类模型在输出层最后一层给出的 原始预测分数，还没有经过像 **Softmax** 这样的归一化操作。

通俗理解：

假设模型要分类为 3 个类别，

那么模型最后输出的向量可能是：

$\text{logits}=[2.0, 1.0, 0.1]$

这些数字 不是概率，只是表示模型对每个类别的“信心”或“打分”。

这些得分可能是 正数、负数、任意实数。

为什么叫“未归一化得分”？

因为这些输出：

没有限制范围（不像概率那样介于 0 到 1）

总和不等于 1

不能直接用于解释为概率

所以叫做“未归一化”。

只有将 logits 经过 Softmax 函数之后，才能转化为 概率分布，用于分类判断。

一句话总结： **Logits** 是模型的原始判断，而 **Softmax** 是模型的概率解释。

情况	是否使用 logits
训练分类模型时	✓ 使用 logits 直接输入交叉熵损失 (CrossEntropyLoss)
需要概率预测时	✗ 不能用 logits，必须经过 Softmax
可视化或解释模型输出时	✗ logits 难以解释，需转为概率

## Support Vector Machine（支持向量机，简称 SVM）

**SVM 的核心思想是：**

找一条最优的“分界线”（高维中是超平面），把不同类别的数据尽可能地分开，并让这条线离数据点尽量远，即“最大间隔分类”。

你有两类点（红色和蓝色），它们在二维平面上分布。SVM 会找到一条线：

- 把红点和蓝点分开；
- 并让这条线距离两边最近的点最远；
- 这些最近的点叫：**Support Vectors**（支持向量）。

## 数学思想（简化版）

我们想找一个 超平面  $H$ , 使得:

$$H(x) = w \cdot x + b = 0$$

其中:

- $w$  是权重向量 (决定方向)
- $b$  是偏移量 (决定平面位置)

目标是使间隔  $\text{margin}$  最大, 即:

$$\text{maximize} \frac{2}{\|w\|}$$

并满足分类条件:

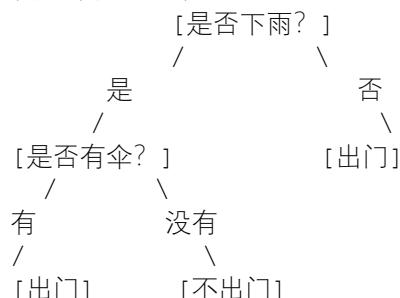
$$y_i(w \cdot x_i + b) \geq 1 \quad \text{for all } i$$

这是一个 凸优化问题, 可以通过拉格朗日乘子等方法求解。

## 什么是决策树 (Decision Tree) ?

决策树是一种树形结构的模型, 每一个节点表示对一个特征的判断, 每一个分支表示判断结果, 每一个叶子节点则对应一个预测输出 (类别或数值)。

### 决策树的基本结构



## 决策树的构建核心 —— 如何分裂节点?

关键: 选出能最好地区分数据的特征进行划分。

常用的划分标准:

**信息增益 (Information Gain) :**

适用于 ID3 算法

衡量划分前后数据不确定性的减少量

使用 **熵 (Entropy)** 作为衡量标准

**信息增益率 (Gain Ratio) :**

用于 C4.5 算法

解决信息增益偏向多值特征的问题

**基尼指数 (Gini Index) :**

用于 CART 算法

衡量“纯度”——越纯的节点 Gini 值越低

基尼指数 (Gini Impurity) 是一种衡量一个集合中数据“纯度”的指标，用于选择最优特征进行节点划分。

- 纯度高：集合中大多数样本属于同一类别 → 基尼指数低
- 纯度低：类别混杂 → 基尼指数高

它回答的是：

如果我们随机从这个集合中挑选两个样本，它们属于不同类别的概率是多少？

对于一个数据集  $D$ ，它的基尼指数定义为：

$$Gini(D) = 1 - \sum_{k=1}^K p_k^2$$

其中：

- $K$ : 类别总数
- $p_k$ : 第  $k$  类样本在数据集中所占的比例

### ✓ 举个例子

假设你有一个节点，其中样本如下：

类别	数量
正例	4
反例	6

总共 10 个样本：

- $p_{\text{正}} = 0.4$
- $p_{\text{反}} = 0.6$

$$Gini = 1 - (0.4)^2 - (0.6)^2 = 1 - 0.16 - 0.36 = 0.48$$

👉 表示这个节点的“杂乱度”为 0.48，越高越杂乱。

### ✓ 举个划分例子（手动算一下）

假设我们要根据一个二元特征（如“是否下雨”）来划分数据：

- 原始数据：10个样本（4正，6反） $\rightarrow Gini = 0.48$
- 划分后：
  - 子集1（下雨）：4个样本（4正，0反） $\rightarrow Gini = 0$
  - 子集2（不下雨）：6个样本（0正，6反） $\rightarrow Gini = 0$

加权平均基尼：

$$Gini_{split} = \frac{4}{10} \cdot 0 + \frac{6}{10} \cdot 0 = 0$$

|  说明这个划分非常好，完全纯净！

如果另一种划分方式得到的  $Gini_{split} = 0.3$ ，那显然我们选择  $Gini_{split} = 0$  的这个划分。

找 feature 能够让 gini 系数最小。

## 1. 决策树回归（Decision Tree Regression）

### ✓ 基本定义

决策树回归是一种通过不断划分输入空间、在每个区域内输出常数的方式来进行预测的模型，适用于连续数值型目标变量（回归任务）。

### 工作原理：

- 把输入特征空间划分成若干个区域（例如根据某个特征是否小于某个阈值）。
- 每个区域内的输出值为该区域内训练样本目标值的平均值。
- 不断选择最佳划分点（最小化 MSE）来分裂节点，直到满足停止条件。

损失函数：均方误差（MSE）

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

优点

- 易于理解、可解释性强
- 可以处理非线性关系
- 不需要特征缩放

缺点

- 容易过拟合（特别是深树）
- 对小变化敏感（高方差）

## 2. 自助聚合（Bootstrap Aggregation, Bagging）

### 基本定义

Bagging 是一种通过“多个模型平均”来减少模型方差的方法，常用于决策树这类高方差模型。

🧠 工作流程：

从原始训练集中，有放回地采样生成多个数据子集（Bootstrap samples）。

对每个子集，训练一个模型（如决策树回归）。

对于新样本，所有模型各自预测，再对这些预测值取平均作为最终结果。

$$\hat{y} = \frac{1}{T} \sum_{t=1}^T h_t(x)$$

其中  $h_t(x)$  是第  $t$  个模型的预测结果。

优点

- 显著降低过拟合风险
- 增强模型稳定性（抗噪声能力强）

Bootstrap 采样采的是“数据样本”，而不是“特征”！

每次 **Bootstrap** 采样是从原始训练集的所有样本中“有放回”地随机抽取一批样本，用于训练模型，而每个样本仍然保留全部特征。

## 3. 随机森林回归（Random Forest Regression）

### 基本定义

随机森林 = Bagging + 再加入特征随机性，进一步减少模型之间的相关性。可以看作是“一群随机生长的决策树的集成”，是 Bagging 的改进版。

工作流程：

对每棵树：

- 使用 bootstrap 采样构建训练子集
- 在每次划分节点时，只随机选取部分特征（例如  $\text{sqrt}(\text{总特征数})$ ）来考虑划分

### 2. 对测试样本：

- 所有树独立预测
- 将预测值平均（回归）或投票（分类）

随机森林带来的两个主要好处：

技术	目的	效果
Bootstrap 采样	降低过拟合（方差）	每棵树不同训练数据
特征子集随机采样	减少树之间的相关性	提高泛化能力

缺点	描述
不易解释	相比单棵树，整体模型为黑盒
占用内存多	每棵树都要保存，树数量多时模型大
预测速度慢	需要所有树都预测一次后再平均

内容	是否使用 Bootstrap?	是否采样特征?
<b>Bagging / Bootstrap Aggregation</b>	<input checked="" type="checkbox"/> 采样“样本”	<input type="checkbox"/> 不采样特征（除非特意指定）
<b>Random Forest</b>	<input checked="" type="checkbox"/> 样本	<input checked="" type="checkbox"/> 特征（在每个分裂点随机选一部分特征）

两个在 **随机森林（Random Forest）** 中常用的特征重要性（Feature Importance）度量方法：

### 1. MDI — Mean Decrease in Impurity (平均不纯度减少)

#### 原理：

随机森林是由很多决策树组成的。

每棵树在训练时会选择最能“降低不纯度”的特征进行分裂（比如使用 Gini 或 Entropy）。

MDI 就是：统计每个特征在所有树中的所有分裂点，带来的总“Gini/熵”降低是多少。

这个“降低”越多，说明该特征对分类结果的“贡献”越大。

#### 优点：

高效（训练模型后直接给出）

内置支持，使用方便

#### 缺点：

对于有很多无用特征或特征相关性高的数据，可能产生**偏差**（如偏向有更多取值的数值型特征）

不能用于非树模型（比如 SVM、线性回归）

## 2. PFI — Permutation Feature Importance (特征置换重要性)

### ✓ 原理：

模型训练完毕后，逐个打乱（打乱顺序）每个特征的值。

每打乱一个特征，就用模型在测试集上重新做预测，看性能下降了多少（例如 RMSE 或 accuracy 下降）。

如果打乱一个特征后性能下降很多 → 说明这个特征非常重要。

### ✓ 优点：

模型无关 (**model-agnostic**)：可以用于任何模型（线性模型、SVM、神经网络等）

直接衡量对预测性能的影响 → 更贴近实际效果

能处理相关特征更好（因为只改变一个特征，观察变化）

### ⚠ 缺点：

**慢**：每个特征都要打乱 + 重新预测，计算成本高

对随机性敏感（所以一般要多次重复置换）

# Feature Engineering

## Feature Scaling

### 常见数据缩放与变换方法总结

方法	所属类型	数学定义 / 核心思想	变换后特点	典型应用场景	操作
MinMaxScaler	线性缩放	$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$	将特征压缩到固定区间 (默认 [0, 1])	神经网络、KNN、SVM 等 对特征量纲敏感的模型	
StandardScaler	线性标准化	$x' = \frac{x - \mu}{\sigma}$	均值为 0，方差为 1， 保持分布形状	线性模型 (Logistic/Linear Regression)、SVM、PCA	
RobustScaler	线性标准化 (抗异常值)	$x' = \frac{x - \text{median}}{\text{IQR}}$	不受极端值影响 (基于中位数和四分位距)	数据中存在 outlier 时，如金融数据	
Normalizer	向量归一化	将每个样本的特征向量缩放为单位范数 (L2/L1)	每个样本长度为 1	文本特征 (TF-IDF)、余弦相似度计算前	
Log Transform	非线性变换	$x' = \log(x + c)$ (c 用于避免 log(0))	压缩大值、拉伸小值， 使分布更接近正态	特征高度偏态 (如收入、价格、点击数)	
Quantile Transformer	非线性变换 (分位数)	将数据映射到指定分布 (如正态或均匀分布)	消除异常值影响，结果分布平滑	特征分布极不均匀、需要正态分布输入的模型	
Power Transformer (Box-Cox / Yeo-Johnson)	非线性变换 (幂次)	$\downarrow$ $x' = \frac{x^\lambda - 1}{\lambda}$ (Box-Cox)	使分布更接近高斯分布、减小偏态	特征偏态明显、回归模型、线性假设模型	

## 小结建议

如果 模型依赖距离度量（如 KNN、SVM、神经网络）→ 用 **MinMaxScaler / StandardScaler**

如果 数据有异常值 → 用 **RobustScaler** 或 **QuantileTransformer**

如果 特征分布偏斜（如收入、销量）→ 用 **Log / Power Transform**

如果 想要正态分布输入 → **PowerTransformer** 或

**QuantileTransformer(output\_distribution='normal')**

## Feature Filtering (特征筛选)

目的：减少输入特征数量，去掉无关、冗余或噪声特征，从而：

降低模型复杂度

提高泛化能力

加快训练速度

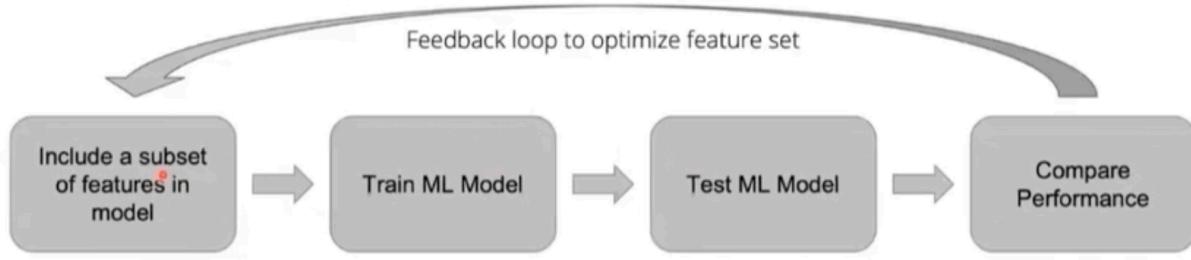
减少过拟合

类型	方法	核心思想	适用场景
统计过滤法 (Filter methods)	<b>VarianceThreshold</b>	删除方差很小的特征（几乎不变）	初步过滤掉“常量列”
	相关系数过滤 (Correlation filter)	计算每个特征与目标值的相关系数（如Pearson、Spearman），删除低相关的特征	回归或连续目标
	<b>Chi-square 检验 (<math>\chi^2</math>)</b>	检查分类特征与目标类别的独立性	分类问题（类别特征）
	<b>Mutual Information (互信息)</b>	衡量特征和目标之间的信息共享程度	非线性关系也可检测
包裹法 (Wrapper methods)	<b>Recursive Feature Elimination (RFE)</b>	用模型评估特征重要性，递归移除最不重要的特征	对模型性能最敏感但计算较慢
嵌入法 (Embedded methods)	<b>Lasso (L1 正则化)</b>	通过正则化让部分权重为零来实现特征选择	高维数据，线性模型
	<b>Tree-based feature importance</b>	使用决策树、随机森林等模型的特征重要性	自动选出非线性重要特征

Filter method: Variance inflation factor

Wrapper method:

Wrapper methods provide another approach to feature selection



- Method to select features can be done randomly, individually, combinatorially.
- Depending on choice of method, feedback loop can be automated, or require user insights
- If K features, an exhaustive search for the best feature subset would cost:  
 $C(K,1)+C(K,2)+C(K,3)....+C(K,K)$  models to train and test

## Neural Network 简称 NN

是模仿人脑神经元连接方式的一种数学模型。它由很多“神经元”（neurons）组成，每个神经元接收输入、做加权求和、加上偏置（bias），然后通过一个非线性函数（activation function）输出结果。

神经网络是一个可以“自动学习输入与输出关系”的函数逼近器。  
换句话说，它能从数据中“学出”一个复杂函数。

### 核心结构组成

一个典型的神经网络由三种层组成：

1. **输入层（Input Layer）**
  - 接收原始数据（如图片像素、文本特征等）。
2. **隐藏层（Hidden Layers）**
  - 通过权重（weights）和激活函数（activation function）计算中间特征。
3. **输出层（Output Layer）**
  - 生成最终的预测结果（如分类概率、回归值等）

每一层的神经元都会接收上一层的输出，做如下运算：

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$a = f(z)$$

其中：

- $x_i$ : 输入特征
- $w_i$ : 权重（决定每个输入的重要性）
- $b$ : 偏置项
- $f$ : 激活函数（如 ReLU, Sigmoid, Tanh）
- $a$ : 输出结果

## 神经网络如何学习

神经网络的“学习”就是不断调整权重  $w$  和偏置  $b$  的过程。

### 1. 前向传播 (Forward Propagation)

输入数据经过每层计算，生成最终输出。

### 2. 计算损失 (Loss)

通过损失函数 (Loss Function) 计算预测值与真实值的误差。

常见损失函数如：

- MSE (均方误差)
- Cross-Entropy (交叉熵)

### 3. 反向传播 (Backpropagation)

利用链式法则 (chain rule) 计算每个权重对损失的影响。

### 4. 梯度下降 (Gradient Descent)

根据梯度更新权重，最小化损失函数：

类型	特点	典型应用
MLP (多层感知机)	最基础的前馈结构	tabular 数据、简单分类
CNN (卷积神经网络)	通过卷积提取局部特征	图像识别、目标检测
RNN (循环神经网络)	具有时间序列记忆能力	语音识别、文本生成
LSTM / GRU	改进的 RNN，解决长期依赖问题	时间序列预测
Transformer	基于注意力机制 (Attention)	机器翻译、ChatGPT、BERT 等

[https://docs.pytorch.org/tutorials/beginner/basics/buildmodel\\_tutorial.html](https://docs.pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)

在 **neural network** (神经网络) 中, **activation function** (激活函数) 是每一层神经元输出前所使用的非线性函数, 它决定了神经元的输出并赋予网络非线性能力。没有激活函数, 网络只能表示线性关系, 这会严重限制它的表达能力。下面详细讲解常见激活函数及应用场景:

### 1 Sigmoid 函数

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

特点:

- 输出范围:  $0 \sim 1$
- 平滑、可微
- 适合概率输出 (如二分类任务)

缺点:

- 容易 **梯度消失** (当输入很大或很小时, 梯度接近 0)
- 计算指数略慢

### 2 Tanh (双曲正切) 函数

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

特点:

- 输出范围:  $-1 \sim 1$
- 零中心化 (比 sigmoid 更适合隐藏层)

缺点:

同样存在 **梯度消失问题**

### 3 ReLU (Rectified Linear Unit)

公式:

$$\text{ReLU}(x) = \max(0, x)$$

特点：

- 输出范围： $0 \sim \infty$
- 计算简单，收敛快
- 减少梯度消失问题

缺点：

- **Dead Neurons:** 如果神经元一直输出 0，梯度永远为 0

#### 4 Leaky ReLU / Parametric ReLU

公式：

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

- $\alpha$  通常为 0.01
- 改进了 ReLU 的“Dead Neurons”问题

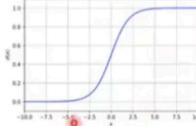
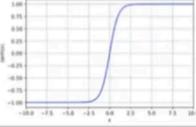
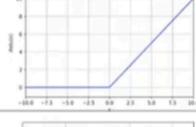
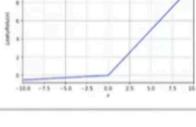
#### 5 Softmax (多分类输出)

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

特点：

- 输出范围： $0 \sim 1$ ，且所有输出之和为 1
- 常用于 多分类任务的输出层

层类型	常用激活函数	原因/场景
隐藏层	ReLU, Leaky ReLU	避免梯度消失，加速训练
二分类输出层	Sigmoid	输出概率
多分类输出层	Softmax	输出概率分布
零中心化隐藏层	Tanh	输出 $-1 \sim 1$ ，更易训练

	Advantages & Key properties	Disadvantages
Sigmoid	 <ul style="list-style-type: none"> <li>Output squashed (non-zero centered)</li> <li>Common in last layers of binary classification networks</li> </ul>	<ul style="list-style-type: none"> <li>May result in vanishing gradients</li> <li>May be slow to train (weak gradients away from zero)</li> <li>May be slow to compute (exp functions)</li> </ul>
Tanh	 <ul style="list-style-type: none"> <li>Output squashed (zero centered)</li> <li>Common in hidden layers of classification networks</li> </ul>	<ul style="list-style-type: none"> <li>May result in vanishing gradients</li> <li>May be slow to train (weak gradients away from zero)</li> <li>May be slow to compute (exp functions)</li> </ul>
ReLU	 <ul style="list-style-type: none"> <li>Overcomes vanishing gradients</li> <li>Hence preferred in deep networks</li> <li>Fast to compute (simple f and f')</li> </ul>	<ul style="list-style-type: none"> <li>Gradients at all negative inputs become zero</li> <li>Dead nodes</li> </ul>
Leaky ReLU	 <ul style="list-style-type: none"> <li>Overcomes vanishing gradients</li> <li>Overcomes dead nodes</li> </ul>	

“梯度消失”（vanishing gradient）是训练深层神经网络时常见的核心问题之一，它会导致网络学习停滞、权重几乎不更新。我们来一步一步讲清楚：当网络很深时（比如十几层、几十层），梯度在反向传播过程中会层层相乘。

如果每层的梯度（或激活函数的导数）都小于 1，比如 0.9，那么经过多层乘积后：

$$(0.9)^{20} = 0.12, (0.9)^{50} = 0.005$$

👉 梯度越来越小，最后几乎变成 0。

结果：前面的层几乎不再更新——网络“学不动了”。、

激活函数如何导致梯度消失？

以 sigmoid 举例：

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

它的导数是：

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

当  $x$  很大或很小时，sigmoid 的输出趋近 0 或 1，此时导数非常小（接近 0）。

因此在深层网络中，sigmoid 层层相乘后梯度迅速趋近 0。

表现症状

当你训练神经网络时：  
 loss 几乎不下降  
 早期层（靠近输入层）参数几乎不变化  
 模型收敛特别慢  
 这些往往都是梯度消失的表现。

方法	原理
✓ 使用 <b>ReLU / LeakyReLU</b>	避免梯度接近 0 的区间
✓ 使用 <b>Batch Normalization</b>	保持每层输入分布稳定，防止梯度极端值
✓ 使用 <b>Residual Connection</b> (残差结构)	比如 ResNet，让梯度可以直接传递到前层
✓ 使用 合适的权重初始化	如 Xavier 或 He 初始化，保持方差稳定

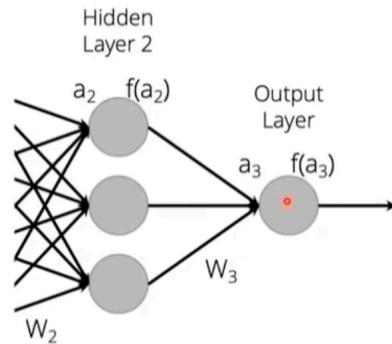
## Caution

Linear (identity):  
 Passes its input to the output without any change

Sigmoid:  
 Output constrained to: [0,1]

Tanh:  
 Output constrained to: [-1, +1]

ReLU:  
 Output constrained to: [0, +inf]



Regression: In the last (output) layer of a NN, use a Linear activation function. Other activation functions should be used judiciously in the last layer. E.g. if the problem calls only for non-zero output values, then ReLU is appropriate to use in the last layer.

## 神经网络如何“学习”

神经网络通过一个叫 **反向传播 (Backpropagation)** 的算法来学习。

步骤如下：

1. 前向传播 (**Forward Pass**)
    - 计算当前权重下的输出与预测误差。
  2. 计算损失函数 (**Loss Function**)
    - 衡量预测值与真实值之间的差距。
- 常见例子：

$$L = \frac{1}{2}(y_{true} - y_{pred})^2$$

- 反向传播 (Backward Pass)

- 根据链式法则计算每个权重对误差的影响。

- 梯度下降 (Gradient Descent)

- 更新权重，使误差变小。

$$w_{new} = w_{old} - \eta \cdot \frac{\partial L}{\partial w}$$

(其中  $\eta$  是学习率)

这个循环会重复很多次，直到模型学会了规律。

## 那为什么还要激活函数 (Activation Function) ?

如果没有激活函数，整个神经网络就只是线性函数的堆叠：

$$y = W_3(W_2(W_1x + b_1) + b_2) + b_3$$

但数学上你可以证明：

线性函数的组合仍然是线性函数。

👉 也就是说，不管你堆多少层，  
没有激活函数的神经网络都只能学习“直线关系”。

而现实世界是非线性的：

- 图像识别：猫和狗的像素差异不是线性的
- 语音识别：音频信号变化复杂
- 股票预测：涨跌关系不是线性的

激活函数的作用就是给网络“打破线性”，  
让它能表示“弯曲的、复杂的”关系。

## 直观例子

假设输入是  $x = 2$ , 权重  $w = 3$ , 偏置  $b = -4$ 。

① 没有激活函数:

$$y = 3 \times 2 - 4 = 2$$

结果始终是线性变化。

② 有 ReLU 激活函数:

$$a = \max(0, 2) = 2$$

→ 如果输入太小（负数），输出会被“截断”为 0，  
模型可以学习“某些特征只在特定条件下起作用”。

③ 有 Sigmoid 激活函数:

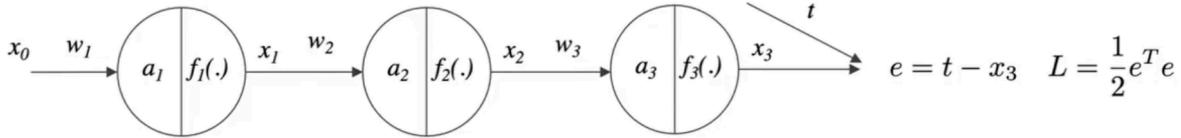
$$a = \frac{1}{1 + e^{-2}} \approx 0.88$$

→ 输出被压缩在 (0,1), 可以代表“概率”。

一道计算题:

# Backpropagation

- Multiple layers. Input and output are scalars. Single data point. No bias.



$$\frac{\partial L}{\partial w_3} = \delta_3 x_2, \text{ where } \delta_3 = -(t - x_3) f'_3(a_3)$$

$$\frac{\partial L}{\partial w_2} = \delta_2 x_1, \text{ where } \delta_2 = \delta_3 w_3 f'_2(a_2)$$

$$\frac{\partial L}{\partial w_1} = \delta_1 x_0, \text{ where } \delta_1 = \delta_2 w_2 f'_1(a_1)$$

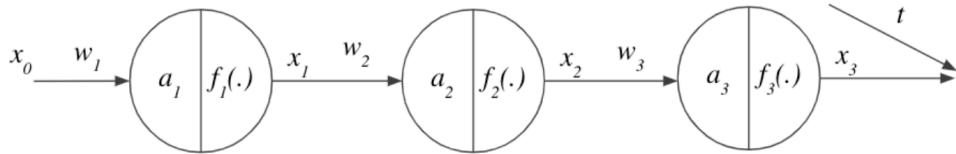
NN delta rule

$$\delta_i = \begin{cases} -(t - x_i) f'_i(a_i) & \text{if } x_i \text{ is the output} \\ \delta_{i+1} w_{i+1} f'_i(a_i) & \text{otherwise} \end{cases}$$

$$w_i^{new} \leftarrow w_i - \lambda \frac{\partial L}{\partial w_i} \quad (\text{gradient descent})$$

Consider the following network, with  $x_0 = 2$ ,  $w_1 = -1$ ,  $w_2 = 3$ ,  $w_3 = 7$ , and linear (identity) activation functions.

Compute  $\partial L / \partial w_3$ ,  $\partial L / \partial w_2$ ,  $\partial L / \partial w_1$  provided that  $t = -40$



**Open-ended short-answer response:**

$$\partial L / \partial w_3 =$$

$$\partial L / \partial w_2 =$$

$$\partial L / \partial w_1 =$$

## 1 Forward pass

$$x_0 = 2, w_1 = -1, w_2 = 3, w_3 = 7, t = -40$$

$$x_1 = (-1)(2) = -2$$

$$x_2 = 3(-2) = -6$$

$$x_3 = 7(-6) = -42$$

$$e = t - x_3 = (-40) - (-42) = 2$$

$$L = \frac{1}{2}(2)^2 = 2$$

## 2 Backpropagation

$$\frac{\partial L}{\partial x_3} = -(t - x_3) = x_3 - t = -42 - (-40) = -2$$

Now chain backward:

$$\begin{aligned}\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial x_3} \frac{\partial x_3}{\partial w_3} = (-2)(x_2) = (-2)(-6) = 12 \\ \frac{\partial L}{\partial x_2} &= \frac{\partial L}{\partial x_3} \frac{\partial x_3}{\partial x_2} = (-2)(w_3) = (-2)(7) = -14 \\ \frac{\partial L}{\partial w_2} &= \frac{\partial L}{\partial x_2} \frac{\partial x_2}{\partial w_2} = (-14)(x_1) = (-14)(-2) = 28 \\ \frac{\partial L}{\partial x_1} &= \frac{\partial L}{\partial x_2} \frac{\partial x_2}{\partial x_1} = (-14)(w_2) = (-14)(3) = -42 \\ \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial x_1} \frac{\partial x_1}{\partial w_1} = (-42)(x_0) = (-42)(2) = -84\end{aligned}$$

### Final Gradients

$$\boxed{\frac{\partial L}{\partial w_1} = -84, \frac{\partial L}{\partial w_2} = 28, \frac{\partial L}{\partial w_3} = 12}$$

## Model Training

$$\text{Err}(x) = \underbrace{\left( E[\hat{f}(x)] - f(x) \right)^2}_{\text{Bias}} + \underbrace{E\left[ \left( \hat{f}(x) - E[\hat{f}(x)] \right)^2 \right]}_{\text{Variance}} + \underbrace{\sigma_e^2}_{\text{Irreducible Error}}$$

**Expected Error = Bias<sup>2</sup> + Variance + Irreducible Noise**

注意这里有两个关键点：

1. 用的是 **Bias** 的平方 (Bias<sup>2</sup>)，不是 Bias
2. 它描述的是 **期望误差** (Expected prediction error)，不是某次训练的单点 error

- **Bias (偏差)** : Bias[\hat{f}(x)] = \hat{f}(x) - f(x)。表示学习算法在无限次重复训练后平均预测与真实函数 f 的偏离。偏差高表示模型“欠拟合”——模型太简单，无法表达真实关系。

- **Variance** (方差) :  $\text{Var}[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \bar{f}(x))^2]$ 。表示当训练集改变（或算法随机性）时，模型预测的波动。方差高表示模型“过拟合”——对训练数据噪声敏感，训练集变化会让预测大幅波动。
- **Irreducible noise** (不可约误差) :  $\sigma^2$ 。这是数据本身的噪声（例如测量误差、被忽略的变量），算法无法通过建模消除。

## Bias–Variance tradeoff (为何有权衡)

通常，当你增加模型复杂度（比如从线性回归到高次多项式、从小决策树到大决策树、从少量基函数到深网络）：

- **Bias** ↓ (减小) : 更复杂模型能拟合更复杂的  $f$ 。
- **Variance** ↑ (增加) : 更复杂模型更易捕捉训练集噪声，导致对不同训练集敏感。

所以总误差 ( $\text{bias}^2 + \text{variance}$ ) 在模型复杂度上常呈 U 型：一端欠拟合（高 bias，低 variance），另一端过拟合（低 bias，高 variance）。train data 准，换成 test data variance 就升高。最佳复杂度位于 U 型底部。不可约误差  $\sigma^2$  无法消除，只能接受。

## 实践中的常见技巧与影响因素

- **增大训练集 (N ↑)** : 主要减少 方差 (更多数据稳定估计)，对 bias 影响小。
- **正则化 (L2/L1/early stopping)** : 通过限制模型复杂度增加 bias、减少 variance，通常降低测试误差。
- **模型集成 (bagging, random forest)** : 通过平均多个模型大幅降低方差，bias 变化小 (bagging 不显著改变偏差)。
- **Boosting**: 往往降低 bias (通过逐步纠正)，但可能增加方差或过拟合，需要正则化。
- **交叉验证**: 用于估计泛化误差，帮助在 bias 与 variance 间选择合适复杂度/超参数。

## 如何在实践中估计 bias 和 variance?

真实情况下不知道  $f(x)$ ，所以不能直接计算严格的 bias。但可以用实验性方法估算：

1. 用相同数据分布反复采样多个训练集（或用 bootstrap），在固定的测试点  $x$  上记录各次  $\hat{f}^{(i)}(x)$ 。
2. 估算  $\bar{f}(x)$  为这些预测的平均，估算方差为这些预测的样本方差，估算  $\text{bias}^2$  为  $(\bar{f}(x) - \text{approx } f(x))^2$ 。  
如果没有  $f(x)$ ，可以用大量训练数据拟合的“高精度模型”或用测试集观测  $y$  的平均扣除噪声近似  $f(x)$ （但这是近似）。

- 在实践中更常见的是观察**学习曲线**（训练/验证误差随样本量或模型复杂度的变化），从曲线形状判断偏差或方差主导。

## Model Training

### K-Fold 交叉验证 (K-Fold Cross Validation)

**目的：**更稳定地估计模型性能，避免因为一次随机划分 train/test 而导致评估结果偏差。

**原理：**

- 将训练集划分成 K 个相等大小的“fold”（例如 K=5）。
- 每次使用其中 1 个 fold 作为验证集，其余 K-1 个作为训练集。
- 训练 K 次，得到 K 个性能分数，然后取平均。

K-Fold 能帮助：

- 防止过拟合 (overfitting)
- 更准确地比较不同模型

### 模型选择 (Model Selection)

1) 比较不同模型：

使用相同的交叉验证框架，比较多个模型的平均得分，选择表现最好的模型。

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

models = {
    "logistic": LogisticRegression(),
    "random_forest": RandomForestClassifier()
}

for name, m in models.items():
    score = cross_val_score(m, X, y, cv=5, scoring='f1').mean()
    print(name, score)
```

## (2) 调参 (Hyperparameter Tuning)

- 用 **GridSearchCV** 或 **RandomizedSearchCV**。
- 自动搜索最优参数组合。

```
from sklearn.model_selection import GridSearchCV

params = {'alpha': [0.1, 1, 10]}
ridge = Ridge()
grid = GridSearchCV(ridge, params, cv=5)
grid.fit(X, y)

print("Best alpha:", grid.best_params_)
print("Best score:", grid.best_score_)
```

## Unsupervised learning (无监督学习)

是机器学习中的一种方法，它的核心是：没有标签（没有  $y$ ），模型自己从数据中发现结构、模式或规律。

换句话说：

监督学习 = “别人告诉你答案”，  
无监督学习 = “自己找规律”。

在 无监督学习 中：只有  $X$ ，没有  $y$

## 2. 无监督学习能做什么？

主要有以下几类任务：

---

### ① Clustering (聚类) —— 将数据分组

模型自动从数据中找“天然的分组方式”。

典型算法：

- K-Means
- Hierarchical Clustering (层次聚类)
- DBSCAN

用途例子：

- 将客户分成不同群体（市场营销）
- 图像分组
- 用户行为模式发现

---

## ② Dimensionality Reduction (降维) —— 将数据压缩到更少维度

例子：PCA、t-SNE、UMAP

用途：

- 特征压缩，加速模型
- 数据可视化（将高维数据降到 2D 或 3D）
- 去除噪声

---

## ③ Density Estimation (密度估计)

找出哪些区域数据多、哪些异常。

算法：

- Gaussian Mixture Model (GMM)
- Kernel Density Estimation (KDE)

用途：

- 异常检测 (Anomaly Detection)
- 模式识别

---

## ④ Association Rule Learning (关联规则)

寻找规律或关联，例如：

买啤酒的人也常买尿布

算法：

- Apriori
- FP-Growth

用途：

- 商品推荐
- 市场篮分析

## K-means 聚类算法怎么工作？

K-means 聚类算法的工作过程可以概括为“初始化 → 分配 → 更新 → 重复”。它是一种常用的无监督学习算法，用来把数据分成  $k$  个簇。

下面是它的详细步骤：

---

### K-means 的工作流程

#### 1. 选择簇的数量 $k$

你必须先决定要分多少类（簇）。

---

#### 2. 随机初始化 $k$ 个聚类中心（centroids）

通常随机选取数据中的  $k$  个点作为初始中心。

---

#### 3. 分配步骤：将每个样本分配到最近的中心

对于每个数据点  $x$ ，计算它到每个中心的距离（一般用欧氏距离），分配给距离最短的那个中心。

这一步完成后，会得到  $k$  个簇。

---

#### 4. 更新步骤：重新计算每个簇的中心

对于每个簇，将簇中所有点的均值作为新的中心：

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$$

---

#### 5. 重复步骤 3 和 4，直到收敛

收敛通常意味着：

- 簇分配不再变化，或
  - 中心点移动变化非常小，或
  - 达到最大迭代次数
- 

### 直观理解

K-means 的目标是找到簇，使得簇内的点彼此尽量接近，其数学目标是最小化簇内平方误差（Within-Cluster Sum of Squares, WCSS）：

$$\sum_{j=1}^k \sum_{x \in C_j} \|x - \mu_j\|^2$$

---

### 算法的优缺点

#### 优点

- 简单易实现
- 速度快，适合大规模数据
- 效果直观

#### 缺点

- 需要预先指定  $k$

- 对初始中心敏感
  - 只能找到“球形”簇，对复杂形状簇不适合
  - 对噪声点敏感
- 

## 层次聚类 (Hierarchical Clustering) 是什么？

层次聚类是一种 不需要预先指定聚类数  $k$  的聚类方法。

它通过不断 合并 或 拆分 数据点来形成一个“层次结构”，通常用 树状图 (dendrogram) 表示。

### 🧠 两大类型

1. Agglomerative (自底向上) ← 最常见

步骤：

每个点一开始都是一个独立 cluster

找到距离最近的两个 cluster

合并它们

重复，直到只剩一个 cluster

得到一棵树（从叶子向上）

2. Divisive (自顶向下)

从一个 cluster 含所有数据点开始

按某种方式拆分成两个最合适的子 cluster

对每个子 cluster 继续拆分

一直到每个点是自己的 cluster

这个方法不常用。

### 🔍 关键区别：如何定义“两个 cluster 的距离”？

这是层次聚类的核心，称为 **linkage** (连接方式)

1. Single linkage (最短距离)

两个簇最近的两个点的距离

→ 容易拉成长“链”，适合弯曲数据（但噪声敏感）

2. Complete linkage (最长距离)

两个簇最远的两个点

→ 簇紧凑、圆形

3. Average linkage (平均距离)

两个簇之间所有点对距离的平均

→ 稳健且常用

4. Ward's method (最小化簇内方差增加)

→ sklearn 默认

→ 类似 KMeans 的思路，但无须指定 k

---

### 树状图 (Dendrogram)

层次聚类最独特的地方：

你得到一张 树状图，可以观察：

每次合并的顺序

你可以在某个高度“切”这棵树 → 得到不同数量的簇

高度越大，越难合并 → 表示两个簇不相似

就像给你一个 可视化的聚类结构。

特点	层次聚类	KMeans
是否要指定 k	 不需要 (可通过树状图选择)	 必须指定
cluster 形状	任意	近似圆形
结果稳定性	 稳定	 依赖初始化
计算复杂度	较高 $O(n^2)$	较低 $O(n)$
适合数据	小到中规模	大规模

## Dimensionality Reduction (降维) 是什么？

降维就是把高维数据压缩成低维表示，同时尽量保持原本的数据结构、信息、或相似性。

你可以把它理解为：

“把原来复杂、噪音多、难以可视化的数据，用更少的特征重新表达，而且不失去本质信息。”

---

## 为什么需要降维？

### 1 可视化 (最常见)

高维数据无法直接画图。

降维后可以把数据映射到 2D 或 3D 进行可视化展示 (聚类、分布等)。

## 2 减少噪音，提高模型效果

很多高维数据有：

- 冗余特征
- 高度相关的特征
- 噪音特征

降维可以帮助模型更专注在真正有用的信息上，避免 **维度灾难** (**curse of dimensionality**) 。

## 3 加速模型训练

低维数据计算量更小，训练更快。

## 4 缓解过拟合

当特征太多、数据不够多时容易过拟合。

降维 = 自动做 regularization。

---

# 🔥 降维方法总结（核心方法）

我把它们分为 3 大类，让你一眼看清：

---

## A 类：线性降维

### ⭐ 1. PCA (Principal Component Analysis)

机器学习里最经典的降维方法。

#### 原理（直观版）

把高维数据投影到一组新的正交坐标轴上，让投影后的数据方差最大。  
这些新轴就是主成分（principal components）。

换句话说：

PCA 找到数据变化最大的方向来作为新的坐标轴，从而实现降维、去噪、压缩、可视化。

## 特点

- 快、稳定、可解释
- 捕捉线性结构
- 保留最多“信息量”（方差）

## 面试最爱的点

- PCA 使用 **SVD or eigen decomposition**
- 第一个主成分 = 最大方差方向
- PCA 是 **无监督**
- 可以作为 **preprocessing** 用在几乎所有模型里

## PCA 的标准步骤 (8 步)

标准化 → 求均值 → 中心化 → 协方差 → 特征分解 → 排序 → 选主成分 → 投影

### ① 数据标准化 (可选)

对于量纲不同的特征，先做 normalize 或 standardize。

最常用：

$$x' = \frac{x - \mu}{\sigma}$$

如果所有特征量纲相近，可以跳过。

---

### ② 计算均值向量 $\mu$

对每个特征（列）计算平均值：

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$$

---

### ③ 数据中心化

$$X_{centered} = X - \mu$$

这一步是 PCA 必要步骤（必须将数据平移到原点）。

---

### ④ 计算协方差矩阵 $S$

$$S = \frac{1}{n-1} X_{centered}^\top X_{centered}$$

在 Python 中：

```
s = np.cov(X_centered.T)
```

协方差矩阵反映各特征之间的相关性。

---

### ⑤ 求协方差矩阵的特征值与特征向量

$$Sv_i = \lambda_i v_i$$

- $\lambda_i$ : 特征值（方差大小）
- $v_i$ : 特征向量（主成分方向）

在 Python 中：

```
w, v = np.linalg.eig(S)
```

---

### ⑥ 按特征值大小排序

从大到小排序：

- 最大的  $\lambda_1 \rightarrow$  第一主成分（最重要）

- 第二大  $\lambda_2 \rightarrow$  第二主成分
  - ...
- 

## ⑦ 选择前 k 个主成分 (降维)

如果你想降到 k 维:

$$W_k = [v_1, v_2, \dots, v_k]$$

这是列向量矩阵。

---

## ⑧ 将数据投影到新轴上

变换公式:

$$Z = X_{centered} W_k$$

每一行是一个样本，每一列是一个主成分得分。

---

## ⭐ 最终结果

- $W_k$ = 主成分方向 (新轴)
- $Z$ = 主成分空间中的数据 (降维后的数据)

The size of the covariance matrix = min( number of features, number of samples)

We're looking for the third eigenvector  $e_3 = [i, j, k]$  that's **unit length** and **orthogonal** to the first two:

- $e_1 = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0]$

- $e_2 = [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0]$

(You wrote  $e_1$  twice; I'm treating the second one as  $e_2$ .)

Because this is a 3D problem with three eigenvalues, we usually assume the eigenvectors form an **orthonormal basis**. So  $e_3$  must satisfy:

1.  $e_3 \cdot e_1 = 0$
2.  $e_3 \cdot e_2 = 0$
3.  $\|e_3\| = 1$

Let  $e_3 = [i, j, k]$ .

### 1. Orthogonality with $e_1$

$$e_3 \cdot e_1 = i \cdot \frac{1}{\sqrt{2}} + j \cdot \frac{1}{\sqrt{2}} + k \cdot 0 = \frac{1}{\sqrt{2}}(i + j) = 0$$

So:

$$i + j = 0 \Rightarrow j = -i$$

### 2. Orthogonality with $e_2$

$$e_3 \cdot e_2 = i \cdot \frac{1}{\sqrt{2}} + j \cdot (-\frac{1}{\sqrt{2}}) + k \cdot 0 = \frac{1}{\sqrt{2}}(i - j) = 0$$

So:

$$i - j = 0 \Rightarrow j = i$$

From the two results:

- $j = -i$
- $j = i$

The only solution is:

$$i = 0, j = 0$$

So  $e_3 = [0, 0, k]$ .

### 3. Unit normalization

$$\|e_3\|^2 = 0^2 + 0^2 + k^2 = 1 \Rightarrow k^2 = 1 \Rightarrow k = \pm 1$$

So the unit third eigenvector is:

$$e_3 = [0, 0, 1] \text{ or } e_3 = [0, 0, -1]$$

Both are valid eigenvectors; the sign doesn't matter unless the problem specifies a particular orientation.

So the values are:

- $i = 0$
- $j = 0$
- $k = \pm 1$  (often we just pick  $k = 1$ , so  $e_3 = [0, 0, 1]$ ).

协方差矩阵的结构是：

$$S = \begin{bmatrix} \text{Var}(x) & \text{Cov}(x, y) \\ \text{Cov}(x, y) & \text{Var}(y) \end{bmatrix}$$

- $\text{Var}(x) = \frac{1}{n-1} \sum (x_i - \bar{x})^2$
- $\text{Var}(y) = \frac{1}{n-1} \sum (y_i - \bar{y})^2$
- $\text{Cov}(x, y) = \frac{1}{n-1} \sum (x_i - \bar{x})(y_i - \bar{y})$

简单理解：

| 协方差就是“两个变量一起偏离自己平均值的程度”。

算出来的结果是：

$$S = \begin{bmatrix} 0.61655556 & 0.61544444 \\ 0.61544444 & 0.71655556 \end{bmatrix}$$

你可以重点理解两点：

- 对角线是每一维自己的方差 ( $x$  的方差 ~0.6166,  $y$  的方差 ~0.7166)
- 非对角线是  $x$  与  $y$  的协方差 (~0.6154, 正数 →  $x$ 、 $y$  正相关)

---

## ⭐ 2. LDA (Linear Discriminant Analysis)

也叫 Fisher Discriminant。

与 PCA 不同：

- PCA 不看 label
- LDA 是 **supervised** (有标签)

原理

- 增大类间距离 (between-class variance)
- 减小类内距离 (within-class variance)

常用于分类前的降维。

---

---

## B 类：非线性降维 (Manifold Learning)

适合数据在高维中弯曲成“流形”，比如人脸、语音等。

### ⭐ 1. t-SNE (t-distributed Stochastic Neighbor Embedding)

🎨 最强数据可视化神器：

高维 → 2D/3D 的漂亮聚类图，用于：

- NLP embeddings
- 图像 embeddings
- 复杂结构可视化

原理 (直观)

- 在高维构建“相似度概率分布”
- 在低维也构建相似度
- 让两个分布差距最小 (KL divergence)

## 特点

- 非常适合可视化
- 不适合用于训练模型的实际降维
- 结果不稳定（随机性大）

## 一、t-SNE 是什么？

**t-SNE** (**t-distributed Stochastic Neighbor Embedding**) 是一种 **非线性降维 / 可视化** 方法，常用来把 **高维数据** (几十、几百、几千维) 映射到 **2D 或 3D**，让你可以画出“聚类结构”和“流形结构”。

它的目标是：

在低维空间中，**尽量保持“局部邻居关系”**：

原来高维空间里“互相很近的点”，在 2D 图里也要靠近；

原来“很远的点”，即使不严格保持距离，至少不要挤在一起。

---

## ★ 2. UMAP (Uniform Manifold Approximation and Projection)

现代比 t-SNE 更快、更稳定的替代方案。

## 优点

- 保持局部结构更好
- 比 t-SNE 快几十倍
- 更适合做 feature embedding 的 pre-processing

---

## C 类：稀疏/特征选择类降维

降维不仅是“变换”，也可以是“挑特征”。

## ★ 1. Feature Selection

- 选前 k 个最重要特征
- 如 tree-based feature importance
- 或 mutual information

## ★ 2. Autoencoder (深度学习降维)

通过神经网络学习低维表示。

### 原理

- 用 encoder 把高维压缩成低维 latent vector
- 再用 decoder reconstruct 原数据
- 低维 latent 就是降维结果

适合非线性复杂数据。

---

## 🧠 如何选择方法？

场景	最适合的方法
可视化 2D/3D	t-SNE, UMAP
模型前降维	PCA
有标签的降维	LDA
大量非线性数据	Autoencoder
特征太多/稀疏矩阵	Feature selection + PCA
需要解释性	PCA

## 一个简单直观例子 (超好懂)

假设你有一个 1000 维的向量（例如图像、文本 embedding）。

PCA 做的事：

“你这 1000 维里很多信息是相关的，比如相同的模式反复出现。我帮你找出最重要的那几个方向，让你只要用 10–50 维就能表达绝大多数有用内容。”

t-SNE 做的事：

“我不管全局结构，我只关心谁和谁像，让你 2D 图看起来有分群。”

Autoencoder 做的事：

“我让我自己的神经网络学习一种方式，在 32 维里就能把数据重新构造回来。”

## 什么是 Ensemble Learning？

一句话：

**Ensemble** = 多个模型一起投票 / 合作 → 获得比单一模型更好的结果。

类比现实世界：

- 一个医生可能会判断错
- 但 **10** 个医生一起判断 → 错误率显著降低

机器学习中也是：

- 单个模型有偏差、有弱点
- 多个模型组合 → 彼此互相补充彼此的缺点

## 为什么 Ensemble 能提高准确率？

原因非常简单：

不同模型的错误不完全一样 → 合在一起错误会抵消；  
共同正确的部分会被放大。

你可以想象：

- 模型 A 对 case1 表现好，但 case2 较差

- 模型 B 刚好对 case2 表现好
- 模型 C 对噪音鲁棒性强

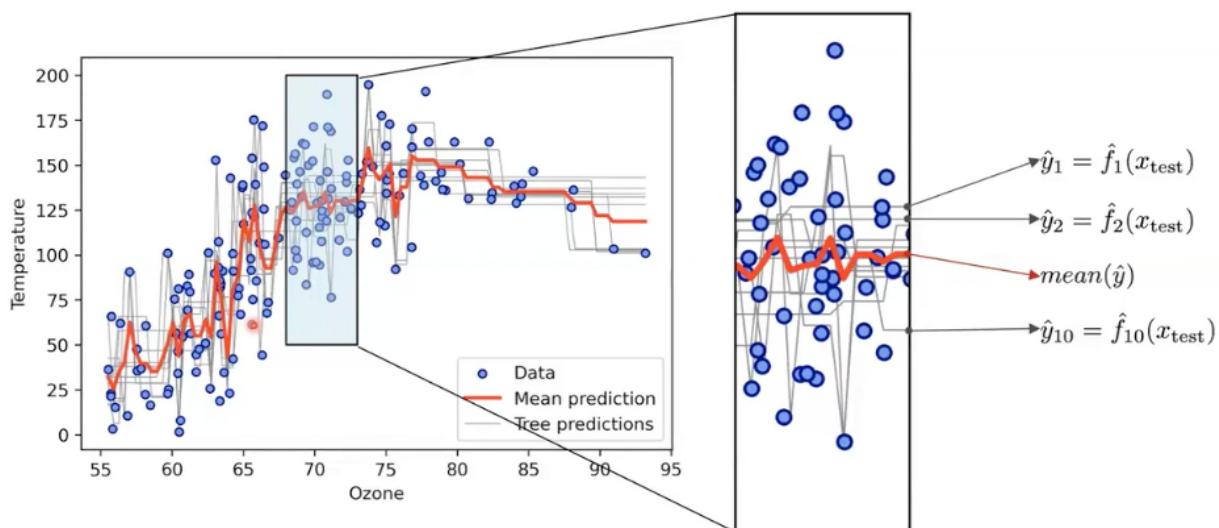
→ Ensemble = A + B + C = 一个更稳定、更强的整体

这就是集成学习效果好的根本原因。

## 🔧 Ensemble 的三大经典方法

### 1 Bagging (Bootstrap Aggregating) — "多数投票" 系列

典型模型：随机森林 Random Forest



思想：

1. 对训练集进行随机采样 (bootstrap) → 得到多个子训练集
2. 每个子训练集训练一个独立模型
3. 最后对结果做 投票 / 平均

作用：

- ✓ 降低 variance (模型不容易 overfit)
- ✓ 非常稳定且简单
- ✓ 训练可以并行，非常适合大规模分布式训练

作为 Software Engineer, 你可以直接类比:

**Bagging** 就是跑多个版本、多个 shard 的模型 → 最后合并结果。

---

## 2 Boosting — 逐个纠错，让模型越来越强

典型模型: **XGBoost**、**AdaBoost**、**LightGBM**

思想:

1. 第一个模型先学
2. 第二个模型专门学习 第一个模型做错的部分
3. 第三个模型学习前面模型的弱点
4. ...重复下去
5. 最终合成强模型

Boosting 非常强，几乎在传统 ML 时代称霸 Kaggle。

作用:

- ✓ 大幅降低 bias (模型更聪明)
- ✓ 效果通常比单模更好
- ✓ 用于 tabular data 时几乎无敌

你可以把它想象成:

每个新模型都像新实习生，专门补前一轮的漏洞 → 最终团队越来越强。

---

## 3 Stacking (Stacked Generalization) — “模型的模型”

这是最工程化的 ensemble。

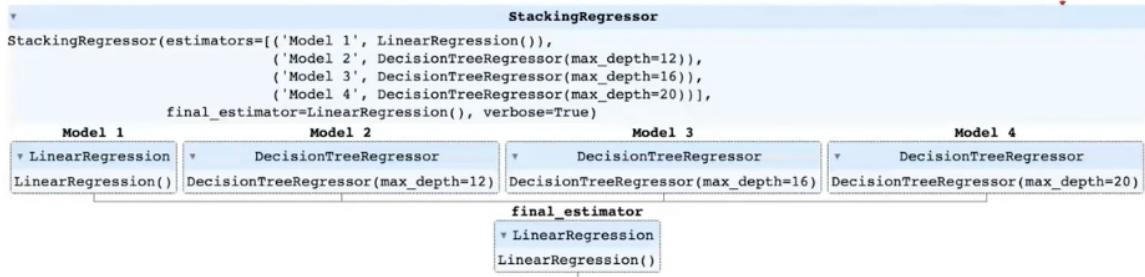
思想:

1. 训练多个不同类型模型 (如 LR、RF、XGBoost、NN)
2. 把它们的预测结果作为 新的输入特征
3. 再训练一个“meta-model (元模型)”来综合判断

- 例如 Logistic Regression, LightGBM, Neural Network

## Stacking

- Train L strong models  $M_i$ :  $i:1..L$ .
  - $M_i$  are usually of different types. E.g. a mix of NN, SVR, LR, DT
  - Each  $M_i$  can be a bagged or boosted ensemble itself
- Output of each model forms a new feature for a subsequent  $M_{final}$ 
  - In effect, original data is transformed, where the dimensionality of the new data is L
  - $M_{final}$  is trained on the new data
- Aims to improve overall performance



作用：

- ✓ 不同模型信息被充分提取
- ✓ 效果通常高于 bagging 和 boosting
- ✓ 是工业界常用的 ensemble 技术
- ✓ 用于 Kaggle / ranking 系统 / 推荐系统非常常见

类比：

多个专家给意见 → 再由一个“总专家”决定最终结果。

## 🎯 工程师视角下的优缺点（你会非常受用）

### ✓ 优点

- 模型性能显著提升
- 更稳定，不容易过拟合
- 适用于所有数据类型
- 既适合 batch 也适合实时 prediction

工业界使用场景包括：

- 推荐系统
  - 广告点击预测
  - 风控系统（银行/支付）
  - 搜索排序
  - 保险定价
  - 医学诊断
- 

## ✖ 缺点（也是 AI Infra 要考虑的）

作为做 AI infra 的 SWE，你应该知道：

- 推理成本变高（多个模型 → latency 变高）
- 部署更复杂（多个模型要 versioning + rollback）
- 监控复杂（每个模型漂移情况不同）
- 资源成本（compute/memory/GPU）上升

解决方法：

- 模型蒸馏（Distillation）→ 把 ensemble 蒸成单模型
- 模型 cache
- 分布式 serving
- 使用 Triton / Ray Serve / KFServing 做多模型管理

这些都是 AI Infra 会常遇到的主题。

---

## 📌 一个简单例子，让你完全理解

假设你做一个 spam classifier：

- 模型 A 对广告邮件很敏感
- 模型 B 对诈骗邮件很敏感
- 模型 C 对病毒邮件很敏感

单个模型都会漏判，但 ensemble：

```
Prediction = majority_vote(A, B, C)
```

准确率立刻提升。

---

## 🎯 总结（最重要的记住这几点）

1. **Ensemble** = 多模型合作 → 抵消错误 → 增强优势
2. 三大方法： **Bagging**、**Boosting**、**Stacking**
3. 工业界表现最强： **Boosting & Stacking**
4. **AI Infra** 必须处理： **Latency + Cost + Deployment Complexity**
5. **Distillation** 是 ensemble 落地在生产的 key 技术