

# lec1 预备知识

## 机器学习概念

机器学习 是一门研究算法的学科，这些算法具备以下特性：

- 提升性能：在特定任务上提高其性能（P）
- 任务导向：针对某项任务（T）
- 经验驱动：基于经验数据（E）
- 非显式编程：通过学习而非显式编程实现
- 任务定义：一个明确的学习任务通常由  $\langle P, T, E \rangle$  三元组定义

## 学习任务类型

	<i>Supervised Learning</i>	<i>Unsupervised Learning</i>
<i>Discrete</i>	classification or categorization	clustering
<i>Continuous</i>	regression	dimensionality reduction

## 正则化方法

L1 正则化(Lasso)

**核心概念：**L1 正则化是在损失函数后加上权重向量的**绝对值之和**。Lasso 全称为 *Least Absolute Shrinkage and Selection Operator*。

**公式：**

$$\text{Loss}_{L1} = \text{Loss}_{Original} + \lambda \sum_{j=1}^p |\beta_j|$$

其中  $\beta_j$  是系数， $\lambda$  是控制惩罚强度的超参数。

L2 正则化(Ridge)

**核心概念：**L2 正则化是在损失函数后加上权重向量的**平方和**。

**公式：**

$$\text{Loss}_{L2} = \text{Loss}_{Original} + \lambda \sum_{j=1}^p \beta_j^2$$

弹性网络

**核心概念：**弹性网络是 L1 和 L2 的结合体。它试图融合两者的优点。

**公式：**

$$\text{Loss}_{EN} = \text{Loss}_{Original} + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2$$

交叉验证

## 2. 核心机制：K-折交叉验证 (K-Fold CV)

这是最主流的方法。假设我们设定  $K = 5$  (即 5-折)：

**具体步骤：**

1. **分组：** 将全部训练数据随机打乱，均匀分成  $K$  等份 (Fold)。
2. **轮转：** 进行  $K$  次训练和验证：
  - **第 1 轮：** 取第 1 份做验证集，其余  $K - 1$  份合并做训练集。记录分数  $E_1$ 。
  - **第 2 轮：** 取第 2 份做验证集，其余  $K - 1$  份合并做训练集。记录分数  $E_2$ 。
  - ...
  - **第 K 轮：** 取第  $K$  份做验证集，其余合并做训练集。记录分数  $E_K$ 。
3. **平均：** 最终的模型得分为这  $K$  次分数的平均值：

$$Score_{final} = \frac{1}{K} \sum_{i=1}^K E_i$$

# lec2 线性回归

---

## 损失函数

0-1 Loss Function

$$L(y^{(i)}, f(x^{(i)})) = \begin{cases} 1, & \text{if } y^{(i)} \neq f(x^{(i)}) \\ 0, & \text{if } y^{(i)} = f(x^{(i)}) \end{cases}$$

Mean Squared Error, MSE

$$L(y^{(i)}, f(x^{(i)})) = (y^{(i)} - f(x^{(i)}))^2$$

Absolute Loss Function

$$L(y^{(i)}, f(x^{(i)})) = |y^{(i)} - f(x^{(i)})|$$

Logarithmic Loss Function (Cross-Entropy Loss Function)

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

$y^{(i)} \in \{0,1\}$ ,  $p^{(i)} = f(x^{(i)})$  is the predicted probability that the  $i$  th sample belongs

## 线性回归

## 简单线性回归 (Simple Linear Regression)

当只有一个特征（比如用“房子面积”预测“房价”）时，公式如下：

$$y = wx + b$$

## 多元线性回归 (Multiple Linear Regression)

当有多个特征（比如用“面积”、“房龄”、“距离地铁距离”来预测“房价”）时：

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

## 梯度下降法

### 批量梯度下降(BGD)

- 怎么做：**用全部数据算一次梯度，更新一次参数
- 优点：**稳定，收敛路径直
- 缺点：**慢，内存要求高
- 一句话：**一步看完全局再走

### 随机梯度下降(SGD)

- 怎么做：**用一个随机样本算梯度，立即更新
- 优点：**快，能跳出局部最优
- 缺点：**震荡大，收敛不稳定
- 一句话：**走一步看一步

### 小批量梯度下降(MBGD)

- 怎么做：**用一小批样本（如32、64个）算梯度
- 优点：**平衡速度与稳定性，适合GPU并行
- 缺点：**需要调批量大小
- 一句话：**看一小片再走，效率和稳定兼顾

方法	每次更新看多少数据	稳定性	速度	常用场景
BGD	全部(100%)	高	慢	小数据、传统ML
SGD	1个样本	低	快	大数据、在线学习
MBGD	一小批(如64个)	中	中	深度学习标配

## 特征归一化与标准化

## Normalization

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

## Standardization

$$x' = \frac{x - \text{mean}(x)}{\text{std}(x)} \quad \text{std}(x) = \sqrt{\frac{\sum(x - \text{mean}(x))^2}{n}}$$

# 自适应学习率优化器

## 1. Momentum (动量法)

- 核心机制:** 模拟物理学中的“惯性”。它在更新参数时，不仅考虑当前的梯度，还累积了**过去的梯度**。
- 主要优势:**
- 平滑路径:** 能够抑制因梯度方向不断变化（如“之”字形震荡）导致的波动。
- 加速收敛:** 在梯度方向一致的维度上加速前进。

## 2. Adagrad (Adaptive Gradient)

- 核心机制:** **自适应**调整学习率。它根据过去梯度的平方和的累积值，为每个参数独立调整学习率。
- 主要优势:**
- 适合稀疏数据:** 出现频率低的特征（梯度更新少）会获得更大的更新步长，而高频特征则更新较小。
- 主要局限:** 随着训练进行，分母上的累积梯度越来越大，导致学习率可能会过早收缩至零，停止学习。

## 3. RMSProp (Root Mean Square Propagation)

- 核心机制:** 它是 Adagrad 的改进版。它不再累积所有的历史梯度平方，而是使用**指数加权移动平均** (Exponentially Decaying Average)。
- 主要优势:**
- 解决学习率消失:** 限制了历史梯度的影响范围，防止学习率过快衰减。
- 动态调整:** 非常适合处理**非平稳目标** (Non-stationary objectives, 如RNN)。

## 4. Adam (Adaptive Moment Estimation)

- 核心机制:** 结合了 **Momentum** 和 **RMSProp** 的优势。它同时计算梯度的一阶矩（均值，类似 Momentum）和二阶矩（方差，类似 RMSProp）。
- 主要优势:**
- 全面高效:** 既有动量的加速效果，又有自适应学习率的调节能力。
- 鲁棒性强:** 对超参数不敏感，通常只需使用默认参数即可获得很好的效果。

# 最小二乘法与正规方程求解

建议参考例子直接记忆参数运算公式：

Examples:

	<b>Size (feet<sup>2</sup>)</b>	<b>Number of bedrooms</b>	<b>number of floors</b>	<b>Age of home (years)</b>	<b>Price (\$1000)</b>
$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$y$
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

## 回归评价标准

MSE(Mean Squared Error)  
均方误差

$$\frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2$$

MAE (Mean absolute Error)  
平均绝对误差

$$\frac{1}{N} \sum_{i=1}^N |y^{(i)} - f(x^{(i)})|$$

RMSE (Root Mean Squared Error)  
均方根误差

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2}$$

R-Squared (r2score) R方/决定系数

$$R^2 = 1 - \frac{\sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2}{\sum_{i=1}^N (y^{(i)} - \bar{y})^2}$$

## lec3 线性分类

logistic 回归(对数几率回归)

- **几率 (Odds)**: 指一个事件发生的概率与不发生概率的比值。

$$\text{Odds} = \frac{p}{1-p}$$

- 如果成功的概率  $p = 0.8$ , 那么几率就是  $0.8/0.2 = 4$  (即 4:1)。
- **对数几率 (Log-Odds / Logit)**: 对几率取自然对数。

$$\text{Logit}(p) = \ln\left(\frac{p}{1-p}\right)$$

- **回归的联系**: 线性回归 ( $y = w^T x + b$ ) 产生的预测值范围是  $(-\infty, +\infty)$ , 而概率  $p$  的范围只能是  $[0, 1]$ 。直接用线性回归预测概率是不合适的。对数几率回归的巧妙之处在于: 它不直接预测概率  $p$ , 而是预测“对数几率”。因为对数几率的范围正好也是  $(-\infty, +\infty)$ , 可以和线性模型完美对应:

$$\ln\left(\frac{p}{1-p}\right) = w^T x + b$$

为了得到我们最终想要的概率  $p$ , 我们将上面的公式反推, 就得到了著名的 **Sigmoid 函数** (S型函数):

$$p(y=1|x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

以下分别是单样本的损失函数, 全数据集的代价函数就是单样本的均值, 学习目标是最小化代价函数

## 1. 损失函数 (Loss Function) - 单样本

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

## 2. 代价函数 (Cost Function) - 全数据集

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

## 3. 学习目标 (Learning Objective)

$$\min_{w,b} J(w, b)$$

# Lda 线性判别分析

## 1. 作用

监督降维 (Supervised Dimensionality Reduction) 与分类。LDA 试图找到一个低维平面，将数据投影上去，以便最好地分离不同的类别。

## 2. 目标

“类内方差小，类间距离大” 将特征空间映射到子空间，使得同类样本在投影点尽可能接近（紧凑），不同类样本的中心在投影点尽可能远离。

## 3. 原理

基于 Fisher 判别准则。假设我们将数据投影到向量上，LDA 通过数学推导找到一个最佳的方向，使得投影后的数据满足上述目标。

#### 4. 散度矩阵公式

假设有  $C$  个类别， $N_j$  为第  $j$  类的样本数， $\mu_j$  为第  $j$  类的均值向量， $\mu$  为所有样本的总均值向量。

- **类内散度矩阵 (Within-class Scatter Matrix,  $S_w$ )** 衡量每个类别内部样本的离散程度（越小越好）：

$$S_w = \sum_{j=1}^C \sum_{x \in X_j} (x - \mu_j)(x - \mu_j)^T$$

#### 3. 类间散度矩阵 ( $S_b$ )

**含义：**度量两个类别中心点（均值）之间的距离（越大越好）。

**公式（二分类情况）：**

$$S_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

#### 5. 优化目标 (广义瑞利商)

我们需要找到投影方向  $w$ ，使得  $J(w)$  最大化：

$$\max_w J(w) = \frac{w^T S_b w}{w^T S_w w}$$

(注：分子代表投影后的类间散度，分母代表投影后的类内散度)

#### softmax 回归

简单来说就是多类别的 logistic 回归。但是因为要多类别所以不能用sigmoid函数，而是用softmax函数

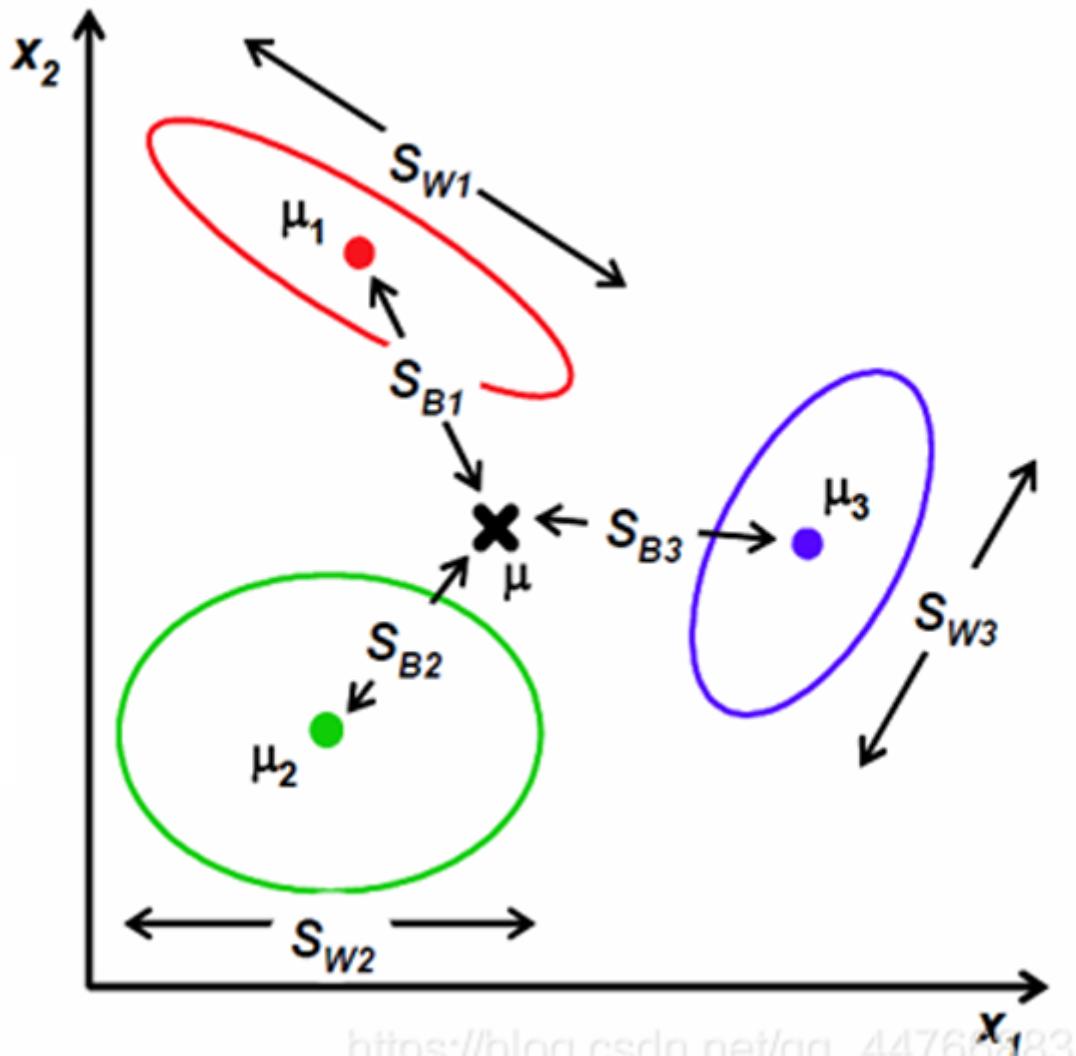
假设线性层的输出为  $z = [z_1, z_2, \dots, z_K]$ ，那么第  $i$  类的预测概率  $\hat{y}_i$  为：

$$\hat{y}_i = \text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

- **分子 ( $e^{z_i}$ )**：通过指数函数，让大的值更大，同时确保所有输出都是正数。
- **分母 ( $\sum e^{z_j}$ )**：归一化项，确保所有类别的概率之和为 1。

在最终决策的时候选用概率最大的那个类别作为最终类别

#### 多类线性判别分析(Multi-class LDA)



类内的散度

矩阵不会发生变化，但是类间散度矩阵要改变：

这是主要变化的地方。它是**各类均值相对于全局均值的加权散度**。

$$S_b = \sum_{k=1}^K N_k (\mu_k - \mu)(\mu_k - \mu)^T$$

- $N_k$  是第  $k$  类的样本数量。
- 这意味着样本数越多的类，其均值离全局均值越远，对  $S_b$  的贡献越大。

## 评估指标

真阳假阳真阴假阴

## Confusion matrix

		Predicted classes	
		1	0
Actual classes	1	True Positive	False Negative
	0	False Positive	True Negative

准确度和错误率

- Accuracy: the ratio of cases when prediction = label

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

- Error rate: the ratio of cases when prediction  $\neq$  label

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

查准与查全

		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

- Precision: the ratio of true class 1 cases in those with prediction 1

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall: the ratio of cases with prediction 1 in all true class 1 cases

$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

敏感度与特异度

- Sensitivity (Recall) :

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

- Specificity:

$$\text{FPR} = \frac{TN}{FP + TN}$$

F1-score

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

或者写作：

$$F1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

## lec4 决策树

---

ID3

**核心逻辑：**选出让系统混乱度（熵）下降最多的特征。

- **熵 (Entropy)：**衡量样本集纯度。

$$H(D) = - \sum_{k=1}^K p_k \log_2 p_k$$

- **条件熵 (Conditional Entropy)：**特征  $A$  将  $D$  划分成  $V$  个子集  $D^v$  后的加权熵。

$$H(D|A) = \sum_{v=1}^V \frac{|D^v|}{|D|} H(D^v)$$

- **决策公式 (Information Gain)：**

$$Gain(D, A) = H(D) - H(D|A)$$

→ 取 Max

## C4.5

**核心逻辑：**在 ID3 基础上除以特征自身的熵，惩罚“分支过多”的特征。

- **固有值 (Intrinsic Value)：**特征  $A$  自身的熵（分支越多，值越大）。

$$IV(A) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

- **决策公式 (Gain Ratio)：**

$$Gain\_Ratio(D, A) = \frac{Gain(D, A)}{IV(A)}$$

→ 取 Max

## CART

**核心逻辑：**严格二分。分类求纯度（Gini），回归求误差（MSE）。

### A. 分类树 (Classification)

- **基尼值 (Gini Index)：**概率  $p_k$  的平方和，比对数运算快。

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

- **特征分裂代价：**特征  $A$  在切分点  $s$  将  $D$  分为  $D_1$  (是) 和  $D_2$  (否)。

$$Gini(D, A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

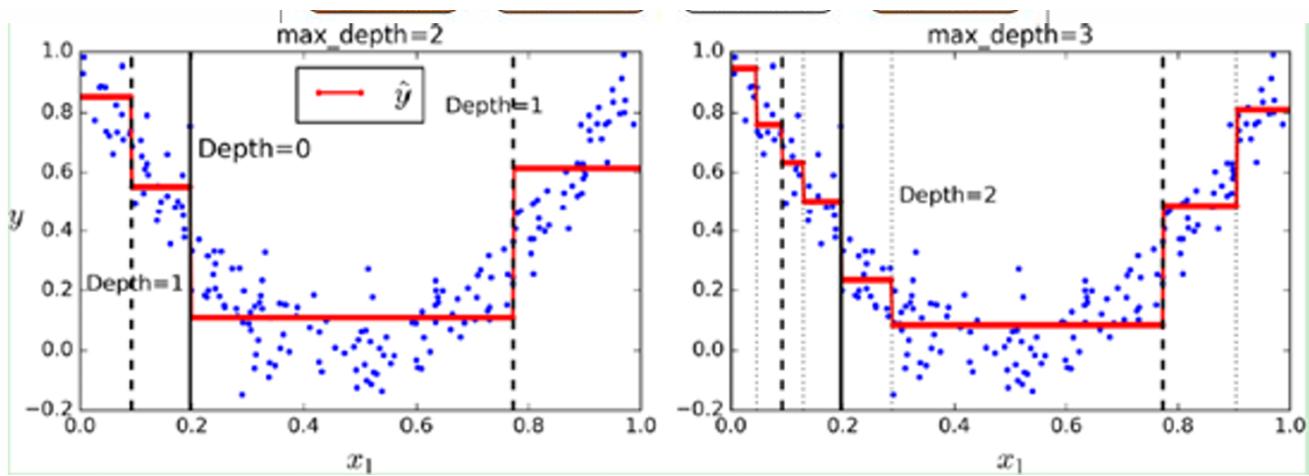
→ 取 Min

### B. 回归树 (Regression)

- **决策公式 (Squared Error)：**寻找切分变量  $j$  和切分点  $s$ ，使两边均方误差之和最小：

$$\min_{j,s} \left[ \sum_{x_i \in R_1} (y_i - c_1)^2 + \sum_{x_i \in R_2} (y_i - c_2)^2 \right]$$

注： $c_1, c_2$  分别为区域  $R_1, R_2$  内  $y$  的均值。



## 剪枝

### 预剪枝

机制：在树的生长过程中，提前停止分裂。即在每次划分前进行评估：如果当前的划分不能带来性能提升（或不满足设定阈值），则停止划分，将当前节点标记为叶节点。

### 后剪枝

机制：先让树完全生长（直到过拟合），然后自底向上地修剪掉多余的子树。即先生成一颗完全树，然后考察去掉某个子树（将其替换为叶节点）后，验证集精度是否提升。

## 1. Reduced-Error Pruning (REP, 错误率降低剪枝)

这是最简单、最直观的方法。

- **核心机制：**它需要一个独立的验证集 (Validation Set) (不能用训练集)。
- **操作：**
  1. 尝试剪掉某个节点（将其变为叶节点，类别由多数票决定）。
  2. 用验证集测试：如果剪枝后的准确率没有下降（甚至上升），就真的剪掉。
  3. 重复直到无法进一步修剪。
- **优点：**简单，线性计算复杂度。
- **缺点：**浪费数据（需要分出一部分数据做验证集），在数据量少时不适用。

## 2. Pessimistic-Error Pruning (PEP, 悲观错误剪枝) / Pessimistic Pruning

这是 C4.5 算法默认使用的剪枝策略。

- **核心机制：**它不需要验证集，直接在训练集上计算。但由于训练集误差通常过于乐观 (Training Error < Generalization Error)，所以它引入了一个“惩罚因子”来人为地\*\*\*“悲观”\*\*估计误差。
- **公式逻辑：**假设某叶节点有  $N$  个样本，误分类了  $E$  个。
  - 乐观误差率 =  $E/N$
  - 悲观误差率 =  $(E + 0.5)/N$  (这个 0.5 就是惩罚项，模拟对未知数据的不确定性)。
- **决策：**如果剪枝后的悲观误差估计值  $\leq$  剪枝前的悲观误差估计值，就剪掉。
- **优点：**不需要额外的验证集，适合小样本数据。

### 3. Cost-Complexity Pruning (CCP, 代价复杂度剪枝)

这是 **CART** 算法使用的策略（刚才提到过的）。

- **核心机制：**引入一个参数  $\alpha$  来平衡“准确率”和“树的复杂度”。
- **公式：**  $Loss = Error(T) + \alpha \times |T_{leaves}|$ 
  - $Error(T)$ : 树的拟合误差。
  - $|T_{leaves}|$ : 叶子节点的数量（代表复杂度）。
- **操作：**随着  $\alpha$  从 0 逐渐增大，我们会得到一系列嵌套的子树，最后通过交叉验证选出最优的  $\alpha$ 。
- **特点：**数学理论最严谨，泛化能力通常最强。

#### 1. Subtree Replacement (子树替换)

这是最常用、最标准的操作。

- **操作：**选中一个子树的根节点，把它整个切掉，替换为一个**叶节点**。
- **叶节点的值：**通常取该子树下所有样本中数量最多的类别（多数表决）。
- **特点：**自下而上进行，结构变化清晰，计算效率高。

#### 2. Subtree Raising (子树提升)

这是一种更复杂的操作，C4.5 中曾使用过，但现代算法较少用。

- **操作：**不仅仅是变叶子，而是把**下层的某个分支（子树）整体提上来**，顶替掉它的父节点（甚至祖父节点）。
  - **例如：**节点 A 是 B 的父节点。剪枝时，删掉 A，把 B 直接提上来连接到 A 的父节点上。
- **特点：**
  - 这会打乱树的层次结构，需要重新计算样本流向。
  - **计算量巨大**（Explanation: 提升子树后，需要重新检查数据是否符合新的分支条件），且效果通常并不比简单的“子树替换”好多少。

## lec5 ANN(人工神经网络)

## lec6 深度学习基础

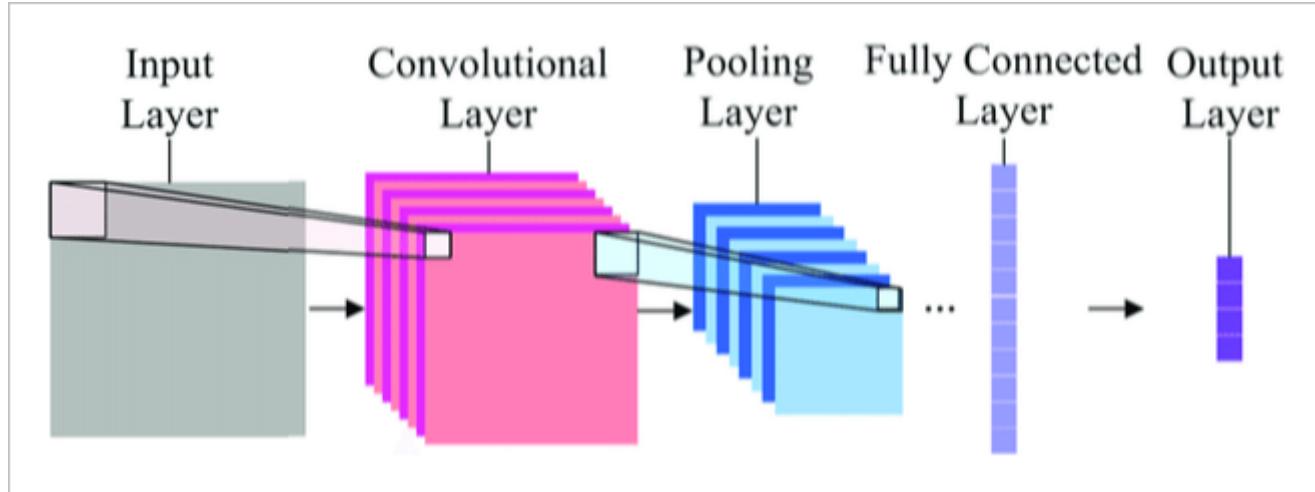
### 卷积神经网络 CNN

**CNN (Convolutional Neural Network)** 是一种专门用于处理**网格状数据**（如图像、视频、语音）的深度学习模型。它通过模仿生物视觉机制，在计算机视觉等领域取得了革命性突破。

## 核心特点：

1. **局部感知**: 不像全连接网络那样每个神经元连接所有输入, CNN只关注输入数据的局部区域
2. **权重共享**: 使用相同的卷积核在整张图像上滑动检测特征, 大大减少参数量
3. **层次化特征提取**: 自动从低级特征(边缘、纹理)到高级特征(物体部件、整体)进行学习

## 主要组件:



### 1. 卷积层 (Convolutional Layer)

- **功能**: 使用卷积核(滤波器)提取局部特征
- **关键参数**:
  - 卷积核大小(如 $3 \times 3$ 、 $5 \times 5$ )
  - 步长(stride)
  - 填充(padding)
- **输出特征图**: 每个滤波器生成一种特征检测图

### 2. 池化层 (Pooling Layer)

- **功能**: 降低特征图尺寸, 增加感受野, 提供平移不变性
- **常见类型**:
  - 最大池化 (Max Pooling) : 取区域最大值
  - 平均池化 (Average Pooling) : 取区域平均值

### 3. 激活函数

- 引入非线性, 常用ReLU及其变体
- ReLU:  $f(x) = \max(0, x)$

### 4. 全连接层 (Fully Connected Layer)

- 位于网络末端, 用于分类或回归任务

## ResNet

## 二、为什么它可以堆叠几百层？

在 ResNet 之前，堆叠层数多了会出现“退化问题”(Degradation Problem)：随着层数增加，准确率反而下降了。这不是过拟合，而是因为网络太深，梯度传不回去，前面的层根本学不到东西。

ResNet 通过以下三个机制解决了这个问题：

### 1. 建立了“梯度高速公路”(Gradient Superhighway)

这是最核心的数学解释。

回忆一下反向传播。在普通网络中，梯度是一层层乘回去的。如果每层的导数都小于 1 (比如 0.9)，乘个 100 次，梯度就变成  $0.9^{100} \approx 0.00002$ ，梯度消失了。

在 ResNet 中，因为输出是  $y = F(x) + x$ 。当我们对  $x$  求导时：

$$\frac{\partial y}{\partial x} = \frac{\partial F(x)}{\partial x} + 1$$

- 那个“1”非常关键！
- 它意味着：在反向传播时，梯度不仅走在那条复杂的卷积路 ( $F(x)$ ) 上，还同时走在那条畅通无阻的直路 ( $x$ ) 上。
- 即使  $F(x)$  那部分的梯度很小甚至为 0，总有一个“1”存在，保证了梯度信号可以无损地从第 100 层直接传回第 1 层。

### 2. 学习“恒等映射”(Identity Mapping) 变得很容易

假设你堆了 100 层，但实际上只需要 50 层就够了。剩下的 50 层是多余的，会捣乱。

- 在普通网络中：想让那多余的 50 层变成“透明”的（即  $H(x) = x$ ），非常难。网络需要把权重  $W$  精确地学成某种样子才能做到输出等于输入。
- 在 ResNet 中：这太简单了！只要把权重  $W$  全部设为 0，那么  $F(x) = 0$ 。
  - 输出公式就变成了  $y = 0 + x = x$ 。
  - 结论：ResNet 只要“偷懒”什么都不做，就能保留上一层的信息。这保证了加深层数至少不会比浅层网络差。

## dropout 和激活函数

## 1. Dropout (随机失活)

### 作用：防止过拟合 (Prevent Overfitting)

在训练神经网络时，如果模型太复杂，它可能会“记住”每一个训练样本的细节，导致在遇到新数据时表现很差。Dropout 就是为了解决这个问题。

- **原理：**在每一次训练迭代中，**随机“关掉”**(使其输出为 0) 一部分神经元。
- **效果：**这样迫使网络不能过度依赖某几个特定的神经元，而是必须让所有神经元都学会提取特征。这增强了模型的泛化能力（鲁棒性）。

**通俗比喻：**就像一个项目团队，如果每天都随机让几个核心员工休假，剩下的员工就必须学会处理所有工作。这样即使某天有人离职，整个团队依然能正常运转。

## 2. 激活函数 (Activation Function)

### 作用：引入非线性 (Introduce Non-linearity)

如果没有激活函数，无论神经网络有多少层，它本质上都只是一个线性的数学公式 ( $y = wx + b$  的叠加)，只能解决简单的线性问题。

- **原理：**在神经元的输出端加上一个非线性的数学函数（如 *ReLU*, *Sigmoid*）。
- **效果：**让神经网络能够逼近任意复杂的函数，从而可以理解图像、声音、自然语言等复杂数据中的非线性规律。

# lec7 SVM 支持向量机

拉格朗日函数(通过对偶问题可以引入内积从而使用核函数)

## 1. 原始问题 (Primal Problem)

SVM的目标是找到一个超平面  $w^T x + b = 0$  来区分正负样本，使得两个类别的间隔 (Margin) 最大化。

我们将这个问题形式化为以下的凸二次规划问题：

- **目标函数：**最小化权重的范数（等价于最大化间隔）。

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

- **约束条件：**所有样本点必须被正确分类，且位于间隔边界之外。

$$y_i(w^T x_i + b) \geq 1, \quad i = 1, \dots, N$$

## 2. 构造拉格朗日函数

为了解决这个带约束的优化问题，我们引入**拉格朗日乘子 (Lagrange Multipliers)**  $\alpha_i \geq 0$  (对应每一个样本点)。

我们将约束条件融合到目标函数中，定义拉格朗日函数  $L(w, b, \alpha)$  如下：

$$L(w, b, \alpha) = \underbrace{\frac{1}{2} \|w\|^2}_{\text{原目标}} - \sum_{i=1}^N \alpha_i \underbrace{[y_i(w^T x_i + b) - 1]}_{\text{约束项}}$$

**直观理解：**如果某个约束没有被满足（即括号内小于0），为了最大化该函数（后续步骤）， $\alpha_i$  会趋向无穷大，从而迫使约束必须满足。

## 3. 从原始问题到对偶问题 (Dual Problem)

原始的优化问题可以表述为：先关于  $\alpha$  最大化（以此来惩罚违反约束的情况），再关于  $w, b$  最小化。即：

$$\min_{w,b} \max_{\alpha \geq 0} L(w, b, \alpha)$$

根据**拉格朗日对偶性**，我们可以交换最小化和最大化的顺序，得到**对偶问题**：

$$\max_{\alpha \geq 0} \min_{w,b} L(w, b, \alpha)$$

## 第一步：求解内部的最小化问题 $\min_{w,b} L$

为了求  $L$  关于  $w$  和  $b$  的极小值，我们需要对它们分别求偏导数并令其为零。

### 1. 对 $w$ 求偏导：

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^N \alpha_i y_i x_i$$

(注意：这说明最优的  $w$  是样本向量  $x_i$  的线性组合)

### 2. 对 $b$ 求偏导：

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^N \alpha_i y_i = 0 \implies \sum_{i=1}^N \alpha_i y_i = 0$$

## 第二步：回代得到对偶形式

将上述求出的  $w$  和约束关系代回拉格朗日函数  $L$  中，消去  $w$  和  $b$ :

$$\begin{aligned} L(w, b, \alpha) &= \frac{1}{2} w^T w - w^T (\sum \alpha_i y_i x_i) - b \sum \alpha_i y_i + \sum \alpha_i \\ &= \frac{1}{2} w^T w - w^T w - 0 + \sum \alpha_i \\ &= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i^T x_j) \end{aligned}$$

## 第三步：最终的对偶问题

现在问题变成了只关于  $\alpha$  的最大化问题（通常我们会加负号变为最小化问题以便求解）：

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i^T x_j) - \sum_{i=1}^N \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \dots, N \end{aligned}$$

#### 4. 为什么要转为对偶问题？（关键点）

转为对偶问题主要有两个巨大的好处：

1. **约束更简单：** 原始问题有不等式约束，且依赖于  $w$  和  $b$ ；对偶问题只有简单的线性约束和非负约束，更容易求解。
2. **自然的引入核函数：** 注意观察对偶问题中的项  $(x_i^T x_j)$ 。这是样本之间的内积。这是引入核函数的关键接口。

直接计算  $\phi(x_i)$  可能非常复杂，甚至是不可能的（如果维度无限）。但是，我们可以找到一个函数  $K(x_i, x_j)$ ，使得：

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

这就是**核技巧 (Kernel Trick)**。它允许我们在低维空间中直接计算出高维空间的内积，而无需显式地写出  $\phi(x)$  的具体形式。

#### 引入核函数后的对偶问题

最终，引入核函数的SVM对偶问题变成了：

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \underbrace{K(x_i, x_j)}_{\text{直接计算核函数}} - \sum_{i=1}^N \alpha_i$$

常见的核函数包括：

- **线性核：**  $K(x, z) = x^T z$
- **多项式核：**  $K(x, z) = (x^T z + 1)^d$
- **高斯核 (RBF)：**  $K(x, z) = \exp(-\frac{\|x-z\|^2}{2\sigma^2})$

## SVM流程

## 阶段一：训练阶段 (Training Phase)

### 1. 输入数据 (Input):

- 接收数据集  $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 。
- 例如： $x$  是图片的像素向量， $y \in \{+1, -1\}$  是类别标签。

### 2. 构建 Gram 矩阵 (Kernel Calculation):

- 动作：**计算所有样本两两之间的核函数值。
- 数据形态：**生成一个  $n \times n$  的对称矩阵  $G$ ，其中  $G_{ij} = K(x_i, x_j)$ 。
- 注：**这一步将样本间的几何相似度转化为了代数数值。

### 3. 求解对偶问题 (Optimization Solver):

- 输入：**Gram 矩阵  $G$  和标签向量  $y$ 。
- 目标：**求解二次规划问题  $\max_{\alpha} \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j G_{ij}$ 。
- 输出：**拉格朗日乘子向量  $\alpha = [\alpha_1, \dots, \alpha_n]^T$ 。

### 4. 模型稀疏化 (Sparsification):

- 动作：**检查向量  $\alpha$ 。
- 结果：**根据互补松弛性，绝大多数  $\alpha_i$  会变为 0。只有少数对应的  $\alpha_i > 0$  的样本被保留，标记为**支持向量**。
- 计算偏置  $b$ ：**选取一个支持向量  $(x_s, y_s)$ ，利用  $y_s(\sum_{i \in SV} \alpha_i y_i K(x_i, x_s) + b) = 1$  反解出  $b$ 。通常会对所有支持向量求平均以增加稳定性。

## 阶段二：预测阶段 (Inference Phase)

当一个新的数据点  $x_{new}$  进入模型时，数据并不经过  $w$ （在核方法中， $w$  维度可能无限大，无法显式存储），而是直接流向支持向量。

### 1. 加权投票：计算新样本与所有**支持向量**的核相似度，并加权求和：

$$score = \sum_{i \in SV} \alpha_i y_i K(x_i, x_{new}) + b$$

**注意：**求和仅针对支持向量进行，非支持向量不参与计算，极大提升了预测效率。

### 2. 判决：

$$\hat{y} = \text{sign}(score)$$

如果  $score > 0$ ，预测为正类；反之预测为负类。

## 核函数

核函数不仅是 SVM 处理非线性问题的核心技巧，其本质是一种计算捷径（Kernel Trick）。

**原理：**在原始低维空间中，数据可能像“异或”逻辑一样交织在一起，无法用一条直线分开（线性不可分）。理论上，我们可以找到一个映射函数  $\phi(x)$ ，将数据投射到高维空间（甚至是无限维空间）。在高维空间中，数据变得线性可分的概率大大增加。

**计算瓶颈与核技巧：**直接计算高维映射  $\phi(x)$  的计算量通常是爆炸性的。核函数的巧妙之处在于：  
我们在高维空间计算两个向量的内积，等同于在低维空间计算一个特定的函数。

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

这意味着我们不需要知道  $\phi(x)$  具体长什么样，也不需要真的去计算高维坐标，只需要算出  $K(x_i, x_j)$  这个标量即可。

常见核函数：

1. **线性核 (Linear Kernel):**  $K(x, z) = x^T z$ 。不做映射，仅用于线性可分数据，速度最快。
2. **多项式核 (Polynomial Kernel):**  $K(x, z) = (x^T z + c)^d$ 。将数据映射到有限的高维空间，通过  $d$  控制维度。
3. **高斯核 / 径向基函数核 (RBF Kernel):**  $K(x, z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ 。
  - 这是最常用的核函数。
  - 它将数据映射到无限维空间。
  - $\sigma$  决定了“邻域”的大小，控制模型的拟合程度。

## 互补松弛性条件

互补松弛性是 KKT 条件 (Karush-Kuhn-Tucker conditions) 中最直观反映 SVM “稀疏性”特质的部分。它是连接原始问题约束与对偶变量  $\alpha$  的桥梁。

**公式表达：**对于每一个训练样本  $i$ , 必须满足：

$$\alpha_i[y_i(w^T x_i + b) - 1] = 0$$

**逻辑推演 (数据的二元状态)：**由于  $\alpha_i \geq 0$  且  $y_i(w^T x_i + b) - 1 \geq 0$  (约束条件), 该乘积为 0 意味着必须通过以下两种情况之一来实现：

- **情况 A: 非支持向量 (Non-Support Vectors)**
  - 样本点被正确分类且远离决策边界, 即  $y_i(w^T x_i + b) > 1$ 。
  - 括号内的项不为 0。
  - 强制推导出  $\alpha_i = 0$ 。
  - 意义：该样本对模型决策边界的确定没有任何贡献，在预测时可以被直接丢弃。
- **情况 B: 支持向量 (Support Vectors)**
  - 样本点恰好落在最大间隔的边缘上, 即  $y_i(w^T x_i + b) = 1$ 。
  - 括号内的项为 0。
  - 允许  $\alpha_i > 0$ 。
  - 意义：这些点是支撑起决策超平面的“柱子”，它们决定了最终模型参数  $w$  和  $b$ 。

## lec8 贝叶斯分类

### 朴素贝叶斯

## 1. 核心思想：概率与逆向推理

朴素贝叶斯不追求通过复杂的函数去拟合数据，它的思路非常直观：

- **基于概率**：它回答的问题是“在这个特征条件下，属于某个类别的概率是多少？”
- **逆向推理**：我们通常通过“原因”推导“结果”。贝叶斯则是根据观察到的“结果”（特征），反推最可能的“原因”（类别）。

## 2. 关键假设：特征条件独立（为什么叫“朴素”？）

这是该算法最核心、也是最具争议的要点。

- **假设内容**：它假设样本中的每一个特征对于类别的判断都是**相互独立**的，互不干扰。
- **举个栗子**：在判断“垃圾邮件”时，算法认为单词“中奖”和单词“汇款”的出现是完全没关系的。
- **意义**：虽然这个假设在现实中很少完全成立（词语之间通常有语境联系），但它极大地**简化了计算复杂度**，让模型在数据量较少时也能高效工作。

## 3. 数学基石：贝叶斯公式

朴素贝叶斯的分类决策基于以下公式。如果你记不住公式，只需要记住这个逻辑：

$$\text{后验概率 (修正后的看法)} \propto \text{似然度 (新证据)} \times \text{先验概率 (原本的看法)}$$

具体公式为：

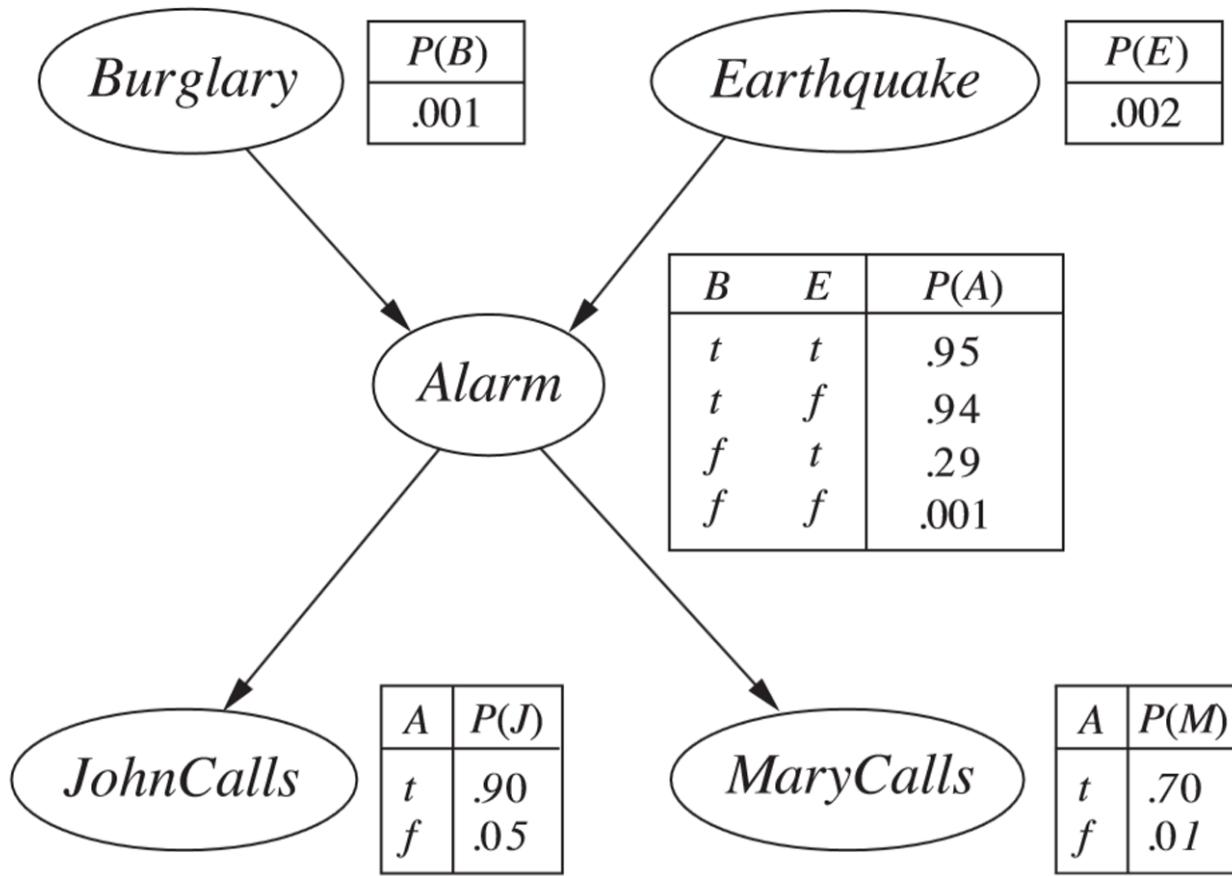
$$P(y|x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i|y)}{P(x_1, \dots, x_n)}$$

- $P(y|x_1, \dots, x_n)$ ：**后验概率**。即看到特征  $x$  后，它属于类别  $y$  的概率（这是我们要求的）。
- $P(y)$ ：**先验概率**。即不看任何特征，该类别本身出现的概率（由历史数据统计得出）。
- $P(x_i|y)$ ：**条件概率（似然度）**。即在类别  $y$  中，特征  $x_i$  出现的概率。
- **决策规则**：对于一个样本，计算它属于所有可能类别的概率，**选概率最大的那个作为结果**。

## 4. 关键技术点：拉普拉斯平滑 (Laplace Smoothing)

这是面试或实战中的一个重要细节。

- **问题**：如果在训练集中，某个特征词从未在某个类别中出现过，那么  $P(x|y) = 0$ 。根据上面的乘法公式，整个概率都会变成 0，这显然不合理。
- **解决**：给分子分母都加上一个很小的常数（通常是 1），保证所有特征的概率都大于 0，避免“一票否决”。

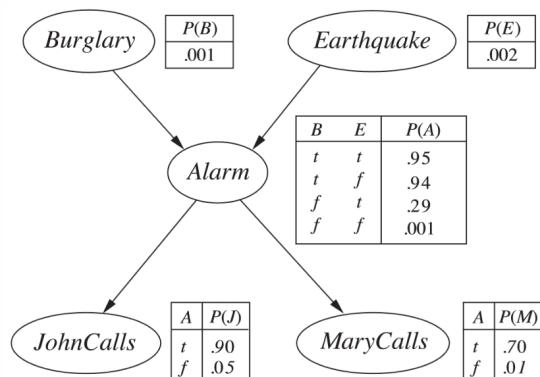


- 贝叶斯网络是完全联合概率分布的一种表示
  - 完全联合概率分布由局部条件概率分布的乘积进行定义（链式法则）

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

- 例如：

$$\begin{aligned} P(j, m, a, \neg b, \neg e) &= P(j|a) P(m|a) P(a|\neg b, \neg e) P(\neg b) P(\neg e) \\ &= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\ &\approx 0.00063 \end{aligned}$$



## 计算方法

### 第一步：条件概率展开 (Bayes' Rule)

利用贝叶斯公式将后验概率转化为联合概率的比值：

$$P(Q | E) = \frac{P(Q, E)}{P(E)} = \frac{P(Q, E)}{\sum_{q \in Q} P(Q = q, E)}$$

### 第二步：边缘化处理 (Marginalization)

如果网络中还存在除  $Q$  和  $E$  之外的其他变量 (隐变量  $Y$ )，需要对这些无关变量进行求和 (求边缘概率)：

$$P(Q, E) = \sum_{y \in Y} P(Q, E, Y)$$

将第一步的分解公式代入：

$$P(Q, E) = \sum_{y_1} \sum_{y_2} \cdots \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

### 第三步：变量消去与因子整合 (Variable Elimination)

这是实际计算中最关键的优化手段。为了避免嵌套求和带来的指数级计算：

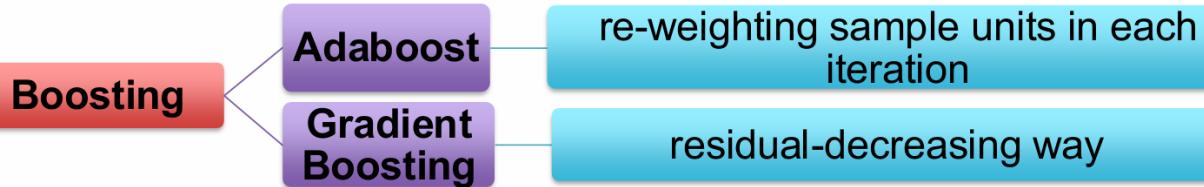
1. 确定求和顺序：选择一个消去变量的顺序 (如从叶子节点到根节点)。
2. 提取因子：将与当前求和变量无关的项移出求和号。
3. 生成中间因子：计算求和结果，生成一个新的因子 (Factor)，再代入下一步计算。

## lec9 集成学习

## Parallel Methods



## Sequential Methods



bagging 集成

Random Forest 随机森林

## 1. 核心原理：两个“随机”

随机森林之所以“随机”，主要体现在以下两个方面：

- **样本随机 (Bootstrap Sampling)**：从原始训练集中，通过**有放回抽样** (Bootstrap) 的方法选取数据。这意味着每一棵树看到的训练数据都是不完全一样的，有的样本可能出现多次，有的可能不出现。
- **特征随机 (Feature Selection)**：在决策树的每一个节点进行分裂时，并不考虑所有特征，而是随机选取一部分特征（通常是  $\sqrt{p}$  个，其中  $p$  为特征总数），再从中选出最优特征进行分裂。

## 2. 训练与预测流程

1. **构建森林**：重复上述两个随机步骤，独立地训练出  $N$  棵决策树。
2. **投票/平均**：
  - \* **分类问题**：采用“少数服从多数”的**投票法** (Voting)，得票最多的类别即为最终结果。
  - **回归问题**：采用所有决策树输出结果的**算术平均值**作为最终预测值。

## 3. 随机森林的优势

- **准确率高**：集成多个弱分类器形成强分类器，泛化能力强。
- **抗过拟合**：两个随机性的引入使得算法对噪声不敏感，不容易陷入过拟合。
- **处理高维数据**：能够处理具有成千上万特征的数据，且不需要进行特征降维。
- **能评估特征重要性**：可以直观地告诉你哪些变量对预测结果影响最大。
- **对缺失值鲁棒**：能够处理缺失数据，并在一定程度上保持准确性。

boosting 集成

AdaBoost

## 1. 关注“差生”(样本权重的动态调整)

在每一轮训练开始时，算法会给所有训练数据分配权重。

- **第一轮：**所有数据平起平坐，权重相同。
- **后续轮次：**如果某个样本在上一轮被分类错误，算法就会提高它的权重；如果分类正确，则降低权重。
- **目的：**强迫下一个弱分类器把精力集中在那些“难以分类”的样本上。

## 2. 话语权分配 (分类器权重的计算)

并不是每个弱分类器都是平等的。AdaBoost 会根据每个弱分类器的准确率给它分配一个“话语权”(权重系数  $\alpha$ )：

- **分类误差越小的分类器，权重越大，在最终投票时说话分量最重。**
- **分类误差越大的分类器，权重越小。**

## 3. 多数票制 (加权投票融合)

最后，将所有训练好的弱分类器组合起来。每个分类器根据自己的判断结果乘以自己的“话语权”，汇总得出最终结果。

## AdaBoost 的数学表达

假设我们有  $M$  个弱分类器  $G_m(x)$ ，最终的强分类器  $G(x)$  为：

$$G(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m G_m(x) \right)$$

其中：

- $G_m(x)$  是第  $m$  个弱分类器的预测结果 (通常为 +1 或 -1)。
- $\alpha_m$  是该分类器的权重，计算公式通常为  $\alpha_m = \frac{1}{2} \ln \left( \frac{1-e_m}{e_m} \right)$  ( $e_m$  为误差率)。

GBDT

## 1. 核心思想：残差学习

GBDT 的精髓在于\*\*“步步为营，纠正错误”。它不是并行训练多棵树（如随机森林），而是采用串行\*\*的方式。每一棵新树的建立，都是为了减少前一轮模型的预测误差。

- **目标：**让预测值不断逼近真实值。
- **手段：**每一棵新生成的树都在拟合上一轮模型留下的**残差**（Residual）。

GBDT 的预测模型是一个**加法模型**：

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

其中  $F_{m-1}(x)$  是前  $m - 1$  棵树的预测和， $h_m(x)$  是当前正在训练的第  $m$  棵树。

如果我们的损失函数（Loss Function）是**平方损失**（Square Loss）：

$$L(y, F(x)) = \frac{1}{2}(y - F(x))^2$$

为了让损失函数最小化，我们希望第  $m$  棵树拟合的值  $h_m(x)$  能够尽可能接近  $y - F_{m-1}(x)$ 。而这个  $y - F_{m-1}(x)$  就是我们所说的**残差**。

## 4. 总结：残差拟合的三个关键点

1. **目标转变：**每一棵树的目标不再是原始标签  $y$ ，而是上一轮留下的预测误差。
2. **树的类型：**为了拟合连续的梯度/残差，GBDT 内部所有的决策树（即使是做分类任务）都必须是**回归树**（CART）。
3. **步长控制**（Learning Rate）：实际操作中，我们通常不会把残差全部加进去，而是乘以一个很小的步长（如 0.1），这叫**收缩**（Shrinkage），能有效防止模型过拟合。

**一句话总结：**GBDT 的残差拟合就是利用每一棵回归树去逼近当前损失函数的负梯度，通过累加这些“修正值”来不断逼近真实目标。

**XGBoost**

## 1. XGBoost (eXtreme Gradient Boosting)

XGBoost 是 GBDT 的大规模并行实现，通过数学上的改进极大提升了精度。

**核心技术：**

- **二阶泰勒展开：** GBDT 只用到一阶导数（残差），而 XGBoost 利用了损失函数的**二阶导数**，使收敛更精确、更准。
- **正则化项：** 在损失函数中加入了  $L_1$  和  $L_2$  正则化（控制叶子节点数和权重），有效防止**过拟合**。
- **预排序 (Pre-sorted)：** 它是其早期的默认分裂算法，能精确找到最佳分裂点，但计算和内存开销较大。
- **缺失值处理：** 自动学习缺失值的默认分裂方向。

**LightGBM**

## 2. LightGBM (Light Gradient Boosting Machine)

由微软开发，核心目标是**快**和**省内存**。它是为了解决 XGBoost 在海量数据下的性能瓶颈而生的。

**核心技术：**

- **直方图算法 (Histogram)：** 不再精确计算分裂点，而是将连续特征划分为多个“桶”(Bins)。这大大减少了内存消耗，并将计算复杂度从  $O(\text{data})$  降到了  $O(\text{bins})$ 。
- **Leaf-wise 生长策略：**
  - **XGBoost (Level-wise)：** 一层一层地长，哪怕某些节点增益很小也长，比较平衡。
  - **LightGBM (Leaf-wise)：** 每次只找**分裂增益最大的**那个节点长，不论它在哪一层。这导致树更深、误差更小，但也更容易过拟合。
- **GOSS (单边梯度采样)：** 只保留梯度大的样本，对梯度小的样本进行随机采样。这样在减少计算量的同时，尽可能保留对减少损失贡献大的样本。
- **EFB (互斥特征捆绑)：** 将许多稀疏的、互斥的特征捆绑成一个，减少特征数量。

lec10 聚类

K-means(**指定类数**)

## 1. 算法流程 (Step-by-Step)

1. **初始化 (Initialization):** 随机选择  $K$  个点作为初始的聚类中心 (Centroids)。
2. **分配 (Assignment):** 计算每个样本点到这  $K$  个中心的距离 (通常使用欧几里得距离)，将每个点分配给距离它最近的中心所属的簇。
3. **更新 (Update):** 针对每个簇，重新计算该簇内所有点的几何中心 (均值)，并将其作为该簇的新中心。
4. **迭代 (Iterate):** 重复步骤 2 和 3，直到满足停止条件。

## 2. 停止条件

通常在以下情况之一发生时停止：

- 中心点的位置不再发生显著变化 (收敛)。
- 样本点的簇分配不再改变。
- 达到了预设的最大迭代次数。

## 3. K-means 的核心特点

- **优点：**原理简单，计算复杂度低  $O(N \cdot K \cdot I)$  ( $N$ 为样本数， $I$ 为迭代次数)，在大数据集上表现高效。
- **缺点：**\*  **$K$  值难确定：**需要预先指定簇的数量。
  - **初始点敏感：**初始中心选得不好可能陷入局部最优 (通常用 **K-means++** 优化)。
  - **形状限制：**只能处理球形或凸分布的数据，对条状或不规则形状效果较差。
  - **异常值敏感：**极端的异常值会严重拉偏中心点。

## 4. 补充：如何选择最优的 $K$ 值？

最常用的方法是 **手肘法 (Elbow Method)**：绘制  $K$  值与 **SSE** (簇内误差平方和) 的关系图。随着  $K$  增大，SSE 会下降。当下降幅度突然减缓，形成一个类似“手肘”的转折点时，对应的  $K$  通常就是最佳选择。

层次聚类 (不用事先定死聚类数量可最后自行选择)

## 1. 初始化（每个样本点自成一派）

将数据集中的每一个样本点  $x_i$  都视为一个独立的簇  $C_i$ 。

- 如果有  $N$  个样本，初始状态就有  $N$  个簇。
- 此时，簇内距离为 0。

## 2. 构建距离矩阵（计算两两亲疏）

计算所有簇与簇之间的初始距离。通常使用**欧几里得距离**，并将其存储在一个对称矩阵中。

- 矩阵的每一个元素  $d_{ij}$  代表簇  $C_i$  和  $C_j$  之间的距离。

## 3. 寻找最小距离并合并（抱团）

在距离矩阵中搜索数值最小的元素（即距离最近的两个簇）。

- 将这两个簇合并为一个新的大簇。
- 此时，总簇数减 1。

## 4. 更新距离（重新定义格局）

这是流程中最关键的一步。合并产生新簇后，需要更新它与其他所有旧簇之间的距离。根据选择的**链接准则（Linkage）**不同，计算方式也不同：

- **最短距离（Single Linkage）**：取两簇间最近点距离。
- **最长距离（Complete Linkage）**：取两簇间最远点距离。
- **Ward's Method（方差最小化）**：计算合并后导致的簇内平方和的增量（这是目前最流行的方法）。

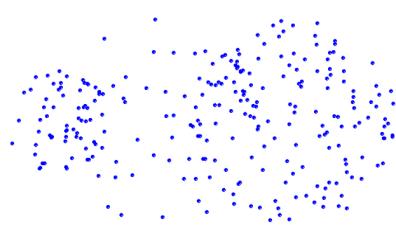
## 5. 迭代与终止（构建树状图）

重复步骤 3 和 4，直到满足以下任一条件：

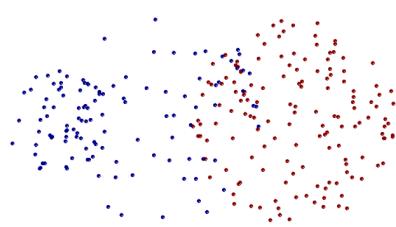
- **所有样本最终合并为一个大簇**（生成完整的树状图）。
- **达到了预设的聚类个数  $K$** 。
- **距离超过了设定的阈值**。

层次聚类中最小和最大标准的缺陷：

## Limitations of MIN



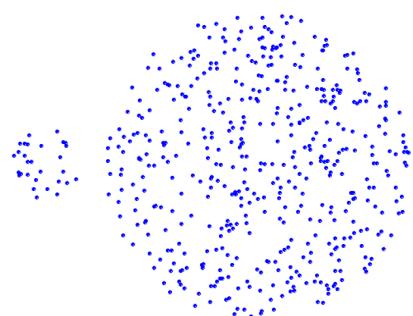
Original Points



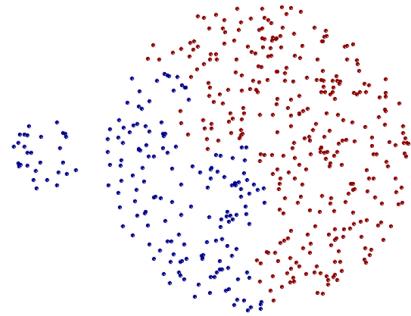
Two Clusters

- Sensitive to noise and outliers

## Limitations of MAX



Original Points



Two Clusters

- Tends to break large clusters
- Biased towards globular clusters

## DBSCAN(密度聚类)

## A. 两个关键参数 (The Parameters)

DBSCAN 不需要你指定聚成几类 ( $K$ )，但它需要你定义什么是“密集”：

1.  $\epsilon$  (Epsilon, 半径)：

- 也就是探测范围。以某个点为圆心，画一个半径为  $\epsilon$  的圆。

2. MinPts (最小点数)：

- 也就是密度的阈值。在这个  $\epsilon$  圆圈里，至少得有多少个点（包括圆心自己），才算“密集”？

## B. 三种点的身份 (Three Types of Points)

根据上面两个参数，每个数据点都会获得一个身份：

1. 核心点 (Core Point)：

- 大佬。它的  $\epsilon$  圈子里至少有 MinPts 个点。它是簇的核心。

2. 边界点 (Border Point)：

- 小弟。它的圈子里点不够多（不够 MinPts），但它落在了某个“核心点”的圈子里。它属于簇的边缘。

3. 噪声点 (Noise Point)：

- 孤独者。既不是核心点，也不是边界点。圈子里没人，也没大哥罩着。它不属于任何簇。

### 3. 算法具体流程

DBSCAN 的过程就像传染病扩散，或者说是\*\*\*“发展下线”\*\*\*：

1. 随机挑一个点  $P$ 。
2. 检查  $P$  的周围：以  $P$  为圆心， $\epsilon$  为半径画圈。
3. 判断身份：
  - 如果圈里点太少：暂时标记为噪声（以后可能会被拉进别人的圈子变成边界点）。
  - 如果圈里点够多：恭喜， $P$  是核心点。创建一个新簇，把圈里所有的点都拉进来。
4. 扩散（发展下线）：
  - 对于刚才被  $P$  拉进来的所有点，挨个检查它们是不是核心点。
  - 如果也是核心点，就继续画圈，把它们周围的点也拉进这个簇。
  - 这种“拉人头”的过程会一直持续，直到簇的边缘再也找不到核心点为止。
5. 换下一个未访问的点，重复上述步骤，直到所有点都被处理过。

### 4. 优缺点分析

优点 (Pros)	缺点 (Cons)
无需指定 K 值：它自己会发现有多少个簇。	参数敏感： $\epsilon$ 和 MinPts 很难调。选小了簇太碎，选大了全连在一起。
发现任意形状：环形、S 形、月牙形都能搞定。	密度不均时失效：如果数据有的地方极密（需要小 $\epsilon$ ），有的地方稀疏（需要大 $\epsilon$ ），DBSCAN 很难用一套参数搞定。
抗噪能力强：自动把离群点标记为 -1 (Noise)，不强行归类。	高维灾难：在高维空间中，“距离”变得很难定义，效果会变差。

## lec11 降维与特征选择

# 降维方法 Dimensionality reduction Method

- Principal Component Analysis (PCA)
- Probabilistic PCA
- Kernel PCA
- Multidimensional Scaling (MDS)
- Locally Linear Embedding (LLE)
- Laplacian Eigenmaps (LE)
- Isometric Mapping(Isomap)
- T-distributed Stochastic Neighbor Embedding (t-SNE)
- Auto-encoders
- Non-negative Matrix Factorization(NMF)
- Canonical Correlation Analysis(CCA)
- Independent Component Analysis (ICA)
- (Linear Discriminant Analysis(LDA)-supervised)
- ...

## PCA 主成分分析

### 顺序求解法

#### 1. 算法一：顺序求解法 (Iterative Algorithms / Sequential)

这种方法通过迭代的方式逐个寻找主成分，而不是一次性计算所有特征向量。

- **适用场景：**适用于超大规模数据集，因为它可以在不计算完整协方差矩阵的情况下找到主要成分。 ↗
- **核心原理：**
  - 第一个主成分指向**方差最大的**方向。 ↗
  - 随后的每一个主成分都与之前的所有主成分**正交**（垂直），并且指向**残差子空间** (Residual Subspace) 中方差最大的方向。 ↗
- **求解方法：**通常可以通过**梯度上升法 (Gradient Ascent)** 来求解。目标是最大化投影后的方差。 ↗

### 特征值分解

## 2. 算法二：特征值分解法 (Sample Covariance Matrix / Eigenvalue Decomposition)

这是最经典的“教科书式”PCA 实现方法，基于协方差矩阵的特征值分解。

- **适用场景：**适用于数据维度不是特别高的情况。 ↴
- **核心流程：** ↴
  1. **去中心化：**将数据矩阵  $X$  的每一列减去均值。
  2. **计算协方差矩阵：**  $\Sigma = \frac{1}{m} XX^T$  (或  $\frac{1}{m-1} XX^T$ )。
  3. **特征值分解：**对  $\Sigma$  进行特征值分解，得到特征值  $\lambda_i$  和特征向量  $u_i$ 。
  4. **排序与选择：**将特征值从大到小排序，选择前  $k$  个最大的特征值对应的特征向量作为 PCA 的基 (Basis Vectors)。

## 奇异值分解 SVD

### 3. 算法三：奇异值分解法 (SVD of the Data Matrix)

这是现代机器学习库（如 Scikit-learn）中最常用的工业级实现方法。

- **适用场景：**通用性最强，可以处理任意大小的矩阵。 ↴
- **核心原理：** ↴
  - 直接对去中心化后的数据矩阵  $X$  进行奇异值分解 (SVD)，无需计算协方差矩阵。
  - 分解公式： $X = USV^T$ 。
- **结果解读：** ↴
  - $U$  矩阵的列向量：**即为主向量 (Principal Vectors) 或主成分方向，且由于正交单位化， $U^T U = I$ 。
  - $S$  矩阵：**对角矩阵，其对角线上的奇异值表示每个特征向量的重要性 (奇异值的平方与特征值成正比)。
  - $V^T$  矩阵：**包含用于重构样本的系数。
- **优势：**相比算法二，SVD 通常数值上更稳定，且对于大型稀疏矩阵有非常高效的随机化算法 (Randomized SVD)。

## LDA 线性判别分析

(上方在线性部分已经提及)

## MDS 和 Isomap

MDS

## 1. MDS (多维缩放, Multidimensional Scaling)

MDS 是一种经典的线性（在某些变体中也可视为非线性框架的基础）降维方法。它的核心目标并不是像 PCA 那样保留数据的方差，而是保留样本之间的距离。

- **核心原理：** MDS 试图在低维空间中找到一组坐标，使得这些点在低维空间中的欧氏距离，尽可能等于它们在原始高维空间中的距离（或差异度）。
  - **输入：** 样本之间的距离矩阵  $D$  ( $N \times N$ , 其中  $d_{ij}$  是样本  $i$  和  $j$  的距离)。注意：MDS 不需要知道原始样本的高维坐标，只需要知道它们之间的距离即可。
  - **输出：** 样本在低维空间（如 2D 或 3D）的坐标  $Z$ 。
  - **目标函数：** 最小化原始距离与低维距离之间的误差，通常形式为：

$$\text{Stress} = \sqrt{\frac{\sum (d_{ij} - \|z_i - z_j\|)^2}{\sum d_{ij}^2}}$$

- **算法流程：**

1. **构建距离矩阵：** 计算原始空间中所有样本对之间的欧氏距离矩阵  $D$ 。
  2. **构建内积矩阵：** 通过“双中心化”操作 (Double Centering)，将距离矩阵  $D$  转换为内积矩阵  $B$ 。这一步利用了余弦定理的性质 ( $d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$ )。
  3. **特征分解：** 对内积矩阵  $B$  进行特征值分解，得到特征值和特征向量。
  4. **计算坐标：** 取前  $k$  个最大的特征值及其对应的特征向量，恢复出低维坐标  $Z$ 。
- **与 PCA 的关系：** 如果距离度量使用的是欧氏距离，那么经典的 MDS (Classical MDS) 算法产生的降维结果与 PCA 是等价的。

Isomap

## 2. Isomap (等度量映射, Isometric Mapping)

Isomap 是流形学习 (Manifold Learning) 的代表算法之一。它本质上是对 MDS 的一种改进，旨在解决非线性数据（如卷曲的瑞士卷数据）的降维问题。

- **核心原理：**在非线性流形 (Manifold) 上，两点之间的真实距离不是直线距离 (欧氏距离)，而是沿着流形表面走的测地线距离 (Geodesic Distance)。
  - **举例：**就像在一张卷起来的纸上，蚂蚁从一头爬到另一头，不能直接穿过空气 (欧氏距离)，而必须沿着纸面爬行 (测地线距离)。
  - Isomap 的策略是：用“测地线距离”替换 MDS 中的“欧氏距离”，然后直接运行 MDS 算法。

- **算法流程：**

1. 构建邻近图 (Neighborhood Graph):

- 对每个样本点  $x_i$ ，找到其最近的  $k$  个邻居（或  $\epsilon$  半径内的邻居）。
- 在这些邻居之间建立连接，边长设为它们之间的欧氏距离。

2. 计算最短路径 (Shortest Path):

- 对于不直接相连的两个点，通过图上的最短路径算法（如 Dijkstra 或 Floyd-Warshall）计算它们之间的距离。
- 这个“图上的最短路径”就是测地线距离的近似值。

3. 运行 MDS:

- 将计算出的“测地线距离矩阵”作为输入，代入经典的 MDS 算法。
- 求解特征值，得到低维坐标。

T-SNE

## 1. 核心思想：把“距离”变成“概率”

t-SNE 的根本思路是：如果在高维空间中两个点距离很近，那么在低维空间中它们也应该很近；反之亦然。

为了量化这种“亲疏关系”，t-SNE 不直接用物理距离，而是使用条件概率：

- 在高维空间 (原始数据)：
  - 使用高斯分布 (Gaussian Distribution) 来转化距离。
  - 对于点  $x_i$ ，如果点  $x_j$  离它越近，那么  $x_j$  被选为  $x_i$  的邻居的概率  $P_{j|i}$  就越大。
  - **直观理解**：就像以  $x_i$  为中心画一个正态分布的圆，落在中心附近的点概率高，边缘的点概率低。
- 在低维空间 (降维后)：
  - 随机初始化一些点  $y_i$ 。
  - 同样计算点之间的相似度概率  $Q_{j|i}$ 。
  - **关键点**：这里不再使用高斯分布，而是使用 t-分布 (Student's t-distribution)。

## 2. 为什么要用 t-分布？(解决“拥挤问题”)

这是 t-SNE 名字中 "t" 的由来，也是它比老一代算法 (SNE) 强的地方。

- **高维空间的“拥挤问题” (The Crowding Problem)**：在高维空间中，点与点之间的空间非常大。当我们把它们强行压扁到 2D 平面时，本来在高维空间中距离“适中”的点，在 2D 平面上会被迫挤在一起，导致分不开。
- **t-分布的作用 (Heavy Tailed / 长尾效应)**：t-分布比高斯分布\*\*“尾巴更长，中间更低”\*\*。
  - 这意味着：在低维空间中，为了获得与高维空间相同的概率（相似度），**中等距离的点必须被推得更远**。
  - **结果**：t-分布强行在低维空间中给“不那么相似”的点之间腾出了空间，从而解决了拥挤问题，让聚类之间的空隙更明显。

### 3. 算法流程

t-SNE 的计算过程本质上是一个优化问题：

1. **计算高维概率  $P$** ：计算原始高维数据中任意两点  $x_i, x_j$  之间的相似度概率  $P_{ij}$ （基于高斯分布）。
2. **初始化低维点  $Y$** ：在 2D 或 3D 空间中随机生成对应的点  $y_i, y_j$ 。
3. **计算低维概率  $Q$** ：计算低维点之间的相似度概率  $Q_{ij}$ （基于 t-分布，自由度为1）。
4. **定义损失函数 (Loss Function)**：使用 **KL 散度 (Kullback-Leibler Divergence)** 来衡量分布  $P$  和分布  $Q$  之间的差异。

$$Cost = KL(P||Q) = \sum \sum P_{ij} \log \frac{P_{ij}}{Q_{ij}}$$

- 如果  $P_{ij}$  高（高维很近）但  $Q_{ij}$  低（低维很远），Cost 会非常大（惩罚这种错误）。

5. **梯度下降 (Gradient Descent)**：移动低维空间中的点  $y$ ，最小化 Cost，直到收敛。

## Autoencoder 自编码器

自编码器由两部分组成，通过无监督方式训练。

- **编码器 (Encoder)**: 将高维输入  $x$  映射到低维隐变量  $z$  (Latent Code)。

$$z = f(Wx + b)$$

通常使用非线性激活函数 (如 ReLU, Sigmoid)。

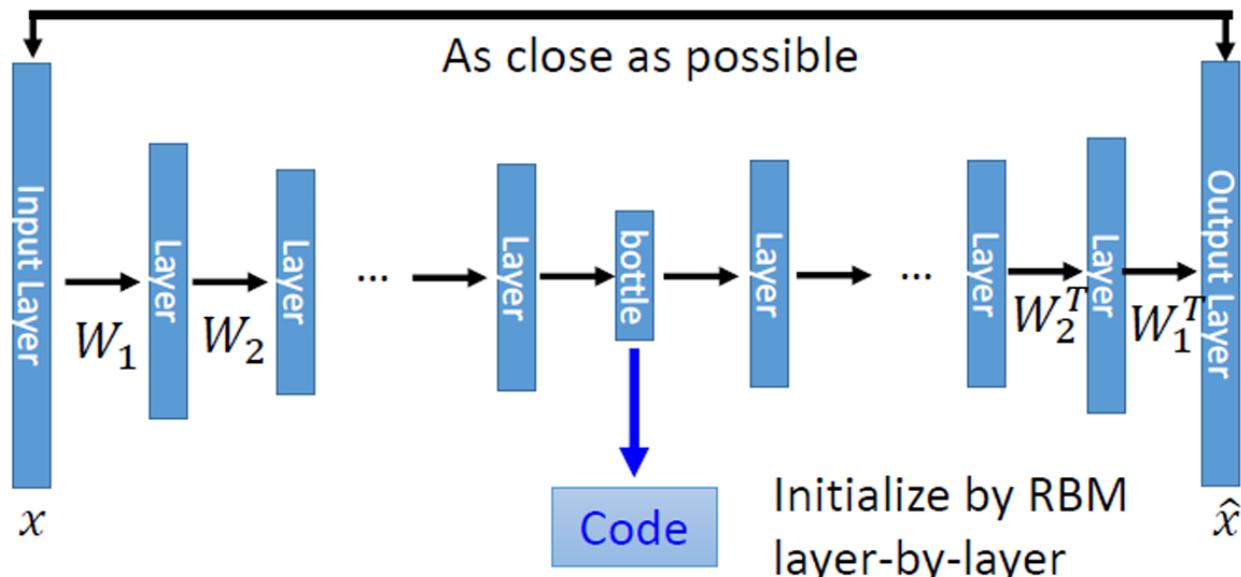
- **瓶颈层 (Bottleneck)**: 这是网络中最窄的一层，维度为  $d$ 。由于  $d < D$  (输入维度)，网络被迫丢弃噪声，只保留最具代表性的特征。
- **解码器 (Decoder)**: 将隐变量  $z$  映射回高维空间，得到重构数据  $\hat{x}$ 。

$$\hat{x} = g(W'z + b')$$

## 2. 优化目标：重构误差

自编码器的训练目标是让输出尽可能接近输入。数学上通常最小化均方误差 (MSE) 或 交叉熵：

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - g(f(x))\|^2$$



## ICA 独立成分分析

经典问题：鸡尾酒会问题

## 一、直观理解：鸡尾酒会问题 (The Cocktail Party Problem)

想象你在一个嘈杂的鸡尾酒会上。

- **场景：**房间里有 3 个人同时在说话（源信号  $s_1, s_2, s_3$ ）。
- **观测：**房间角落放着 3 个麦克风。由于距离不同，每个麦克风录到的声音都是这 3 个人声音的混合（观测信号  $x_1, x_2, x_3$ ）。
  - 麦克风 A 可能听这种混合： $0.8 \times \text{人}_1 + 0.3 \times \text{人}_2 + 0.1 \times \text{人}_3$
  - 麦克风 B 听到另一种混合...
- **任务：**你只有麦克风录到的混乱录音 ( $x$ )，不知道这 3 个人分别说了什么，也不知道麦克风的具体位置。**你能否只通过录音，把这 3 个人的声音单独还原出来？**

这就是盲源分离 (Blind Source Separation) 问题。

ICA 的答案是：能。只要满足两个前提：

1. 这 3 个人的说话是相互独立的（一个人说什么与另一个人无关）。
2. 这些声音信号的分布不是高斯分布 (Non-Gaussian)。

ICA流程

我们将上述问题数学化。

假设有  $n$  个独立的源信号  $s = [s_1, s_2, \dots, s_n]^T$ 。我们要观测到  $n$  个混合信号  $x = [x_1, x_2, \dots, x_n]^T$ 。它们之间通过一个混合矩阵  $A$  关联：

$$x = As$$

- $s$ : 未知的源信号 (Source)。
- $A$ : 未知的混合矩阵 (Mixing Matrix)。
- $x$ : 已知的观测数据 (Observed Data)。

**ICA 的目标：**找到一个解混矩阵  $W$  (即  $A$  的逆矩阵  $A^{-1}$ )，使得：

$$\hat{s} = Wx$$

其中  $\hat{s}$  就是我们还原出来的独立成分。

### 三、核心原理：为什么要“非高斯”？

这是 ICA 最反直觉也最精彩的部分。为什么一定要非高斯分布才能分离？

#### 1. 中心极限定理 (Central Limit Theorem) 的逆向思维

中心极限定理告诉我们：**大量独立随机变量的和，倾向于服从高斯分布（正态分布）。**

- 思考： $x$  是  $s$  的混合（也就是  $s$  的加权和）。
- 推论：根据中心极限定理，混合后的信号  $x$  会比原始信号  $s$  **更像高斯分布**。
- **ICA 的逆向逻辑：**如果我们想从混乱的  $x$  中找回原始的  $s$ ，我们就应该寻找一种变换，使得变换后的结果\*\*“最不像”高斯分布\*\*。

**结论：**非高斯性越强  $\approx$  统计独立性越强。

## 2. 优化目标：最大化非高斯性

ICA 的算法流程本质上就是通过调整矩阵  $W$ ，让输出信号  $y = Wx$  的**非高斯性（Non-Gaussianity）最大化。**

我们如何衡量“非高斯性”？通常用以下两个指标：

- **峰度 (Kurtosis)：**
  - 高斯分布的峰度为 0。
  - 如果峰度绝对值很大（分布很尖或很平），说明是非高斯分布。ICA 会寻找峰度绝对值最大的方向。
- **负熵 (Negentropy)：**
  - 在方差相同的情况下，高斯分布的熵（Entropy）最大（信息最混乱）。
  - 因此，熵越小（负熵越大），信号就越有序，越偏离高斯分布。

## 特征选择

从原始特征集中挑选出最有用的子集，而不是生成新特征。

- **过滤法 (Filter Methods)：**
  - **原理：**使用统计指标（如相关系数、卡方检验、互信息）对特征进行评分和排序，**独立于后续的学习器。**
  - **优点：**速度快，通用性强。
- **包裹法 (Wrapper Methods)：**
  - **原理：**把特征选择看作一个搜索问题，**使用后续学习器的性能作为评价标准。**
  - **策略：**
    - **前向搜索：**从空集开始，每次添加一个效果提升最大的特征。
    - **后向搜索：**从全集开始，每次剔除一个影响最小的特征。
  - **缺点：**计算开销非常大。
- **嵌入法 (Embedded Methods)：**
  - **原理：**特征选择过程与模型训练融为一体。
  - **典型代表：****Lasso (L1 正则化)**。L1 正则化倾向于产生稀疏解，能够自动将不重要特征的权重压缩为 0，从而实现特征选择。