

# 尚硅谷大模型技术之机器学习

(作者: 尚硅谷研究院)

版本: V1.1.0

## 第 1 章 机器学习概述

机器学习 (Machine Learning, ML) 主要研究计算机系统对于特定任务的性能，逐步进行改善的算法和统计模型。通过输入海量训练数据对模型进行训练，使模型掌握数据所蕴含的潜在规律，进而对新输入的数据进行准确的分类或预测。

机器学习是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸优化、算法复杂度理论等多门学科。



### 机器学习是什么

- 什么是学习
  - 从人的学习说起
  - 学习理论；从实践经验中总结
  - 在理论上推导；在实践中检验
  - 通过各种手段获取知识或技能的过程
- 机器怎么学习?
  - 处理某个特定的任务，以大量的“经验”为基础
  - 对任务完成的好坏，给予一定的评判标准
  - 通过分析经验数据，任务完成得更好了

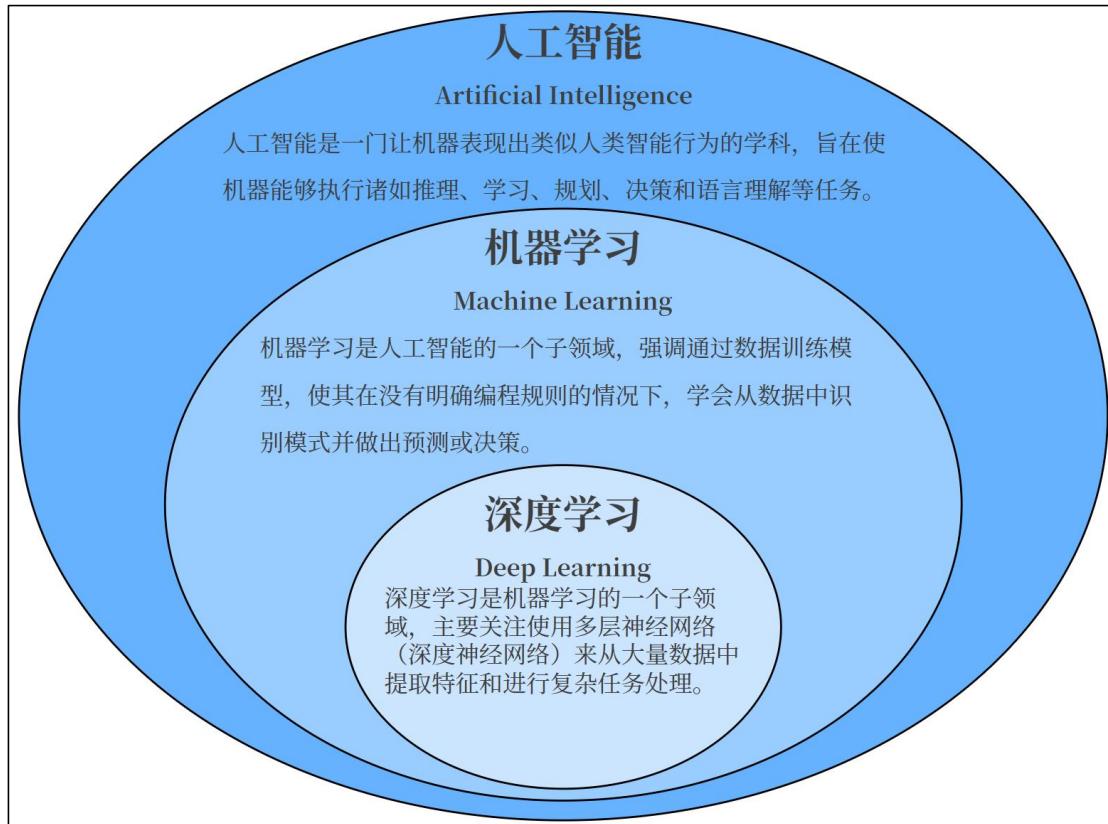


让天下没有难学的技术

## 1.1 人工智能、机器学习与深度学习

- 人工智能 (AI) 是计算机科学的一个广泛领域，目标是让机器具备类人的“智能”，包括自然语言处理 (NLP)、计算机视觉、机器人技术、专家系统等等；具体实现手段多种多样，包括规则系统、符号逻辑、统计方法、机器学习等。

- 机器学习是 AI 的一个子领域，核心是通过数据驱动，让计算机进行学习并改进性能，自动发现规律（模式），并利用这些规律进行预测或决策。
- 深度学习是机器学习的一个子领域，基于多层神经网络处理复杂任务。



## 1.2 发展历史（了解）

### 1.2.1 早期探索：20世纪 50-70 年代

人工智能研究处于“推理期”，人们认为只要能赋予机器逻辑推理能力，机器就能具有智能。

- 图灵提出“图灵测试”，定义机器智能的基本标准
- 弗兰克·罗森布拉特提出感知机算法，这是最早的人工神经网络模型之一，用于线性分类问题
- 亚瑟·李·塞谬尔提出“机器学习”一词，并设计了一个用于玩跳棋的学习算法

### 1.2.2 知识驱动与专家系统：20世纪 70-80 年代

随着研究推进，人们逐渐认识到，仅具有逻辑推理能力是远远实现不了人工智能的。部分人认为，要使机器具有智能，就必须设法使机器拥有知识。计算机硬件逐步发展，人工智能的研究进入以规则为主的“知识驱动”阶段。

- 出现众多专家系统，如 MYCIN（用于医疗诊断）
- 引入了决策树（如 ID3 算法）
- 统计学习理论开始成形，例如贝叶斯定理在机器学习中的应用

### 1.2.3 数据驱动与统计学习：20世纪80年代-21世纪

专家系统面临“知识工程瓶颈”。机器学习成为一个独立的学科领域，各种机器学习技术百花初绽。基于神经网络的连接主义逐步发展，但其局限性也开始凸显。统计学习登场并占据主流，代表技术是支持向量机。互联网的发展带来了大量数据，统计学方法逐渐成为机器学习的核心。

- 决策树算法得到进一步发展（如 C4.5 算法）
- 支持向量机（SVM）被提出，成为一种强大的分类工具
- 无监督学习方法开始成熟，例如 K-means 聚类
- 随机森林和 Boosting 方法（如 AdaBoost）引入，提升了集成学习的性能

### 1.2.4 深度学习崛起：21世纪初

随着计算能力（特别是 GPU 的发展）和数据规模的快速增长，深度学习技术得以崛起。深度神经网络主导，突破了图像、语音和文本领域的性能瓶颈。

- 深度信念网络（DBN）被提出，标志着深度学习研究的复兴
- AlexNet 在 ImageNet 图像分类竞赛中获胜，证明了卷积神经网络（CNN）的强大性能
- 生成对抗网络（GAN）被提出，用于生成图像和其他生成任务
- Transformer 架构在论文《Attention is All You Need》中提出，彻底改变了自然语言处理领域

### 1.2.5 大模型与通用人工智能：2020年代

深度学习进入规模化应用阶段，大语言模型和多模态模型的出现推动了机器学习的新高潮。模型参数规模空前，技术以通用性和泛化能力为目标。

- 自然语言处理：如 OpenAI 的 GPT 系列模型、Google 的 BERT
- 多模态模型：如 OpenAI 的 CLIP、DeepMind 的 Gato
- 自监督学习成为重要方向，降低了对人工标注数据的依赖
- 强化学习和深度学习结合，应用于 AlphaGo、AlphaFold 等项目

### 1.3 机器学习应用领域

今天，在计算机科学的诸多分支学科领域中，无论是多媒体、图形学，还是网络通信、软件工程，乃至体系结构、芯片设计，都能找到机器学习技术的身影，尤其是在计算机视觉、自然语言处理等计算机应用技术领域，机器学习已成为最重要的技术进步源泉之一，并为许多交叉学科提供了重要的技术支撑。

#### 图像处理与计算机视觉

图像分类：识别图像中的主要对象  
目标检测：如自动驾驶中的行人和障碍物检测  
人脸识别：用于安防、支付认证  
图像分割：医疗影像分析中的病灶检测  
图像生成：生成高质量图像或进行风格迁移

#### 自然语言处理

文本分类：垃圾邮件检测、情感分析  
机器翻译：如 Google 翻译、DeepL  
语音识别：语音转文字，如智能助手中的语音输入  
聊天机器人：用于客户服务、教育等领域  
文本生成：创作新闻、摘要或代码生成

#### 推荐系统

电商平台：商品推荐  
内容平台：个性化视频推荐  
音乐流媒体：歌单生成  
社交网络：好友推荐

#### 医疗与健康

疾病诊断：分析医疗影像识别病变  
药物研发：通过机器学习加速新药发现  
健康监测：智能穿戴设备监测心率、睡眠等  
个性化医疗：根据患者数据优化治疗方案

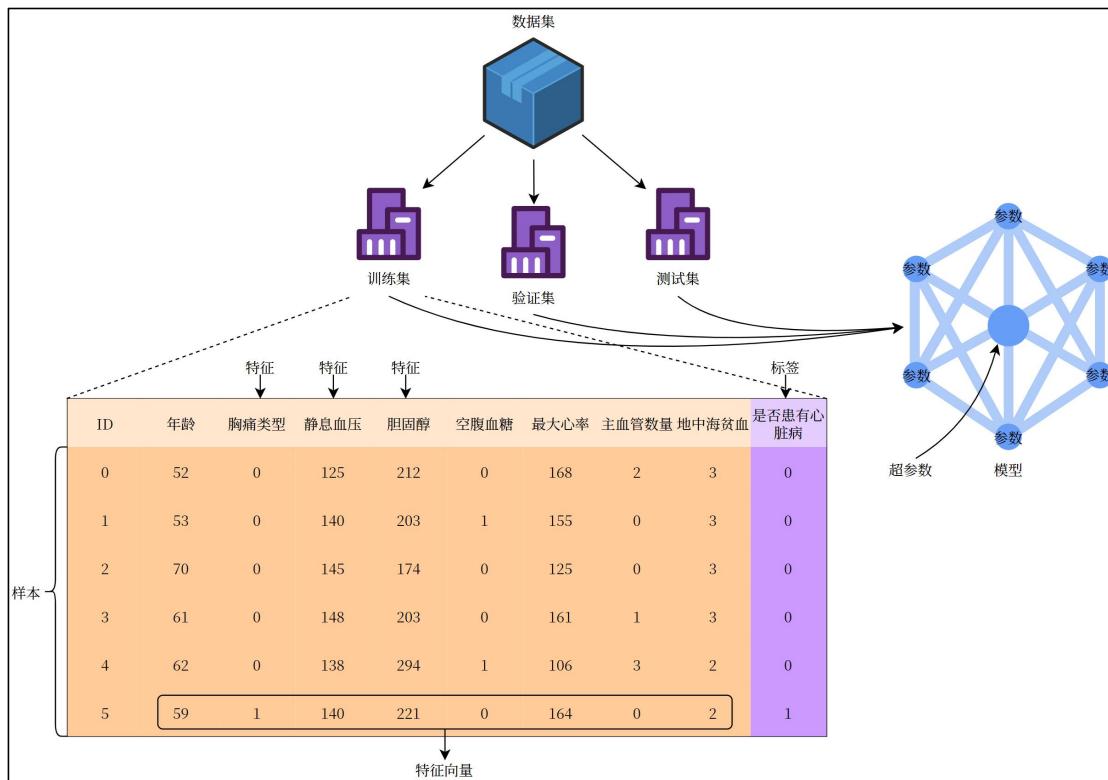
#### 金融科技

信用评分：评估用户的信用风险  
欺诈检测：实时监测异常交易行为  
量化交易：基于算法预测市场趋势并执行交易  
风险管理：预测金融市场风险，提高决策效率

#### 自动驾驶与智能交通

自动驾驶：自动驾驶汽车  
车道保持与避障：使用机器学习处理传感器数据  
智能交通系统：优化红绿灯调度，减少拥堵  
交通流预测：预测交通状况，为导航提供实时优化

### 1.4 基本术语



- **数据集 (Data Set) :** 多条记录的集合。
  - **训练集 (Training Set) :** 用于训练模型的数据。
  - **验证集 (Validation Set) :** 用于调节超参数的数据。
  - **测试集 (Test Set) :** 用于评估模型性能的数据。
- **样本 (Sample) :** 数据集中的一条记录是关于一个事件或对象的描述，称为一个样本。
- **特征 (Feature) :** 数据集中一列反映事件或对象在某方面的表现或性质的事项，称为特征或属性。
- **特征向量 (Feature Vector) :** 将样本的所有特征表示为向量的形式，输入到模型中。
- **标签 (Label) :** 监督学习中每个样本的结果信息，也称作目标值 (target)。
- **模型 (Model) :** 一个机器学习算法与训练后的参数集合，用于进行预测或分类。
- **参数 (Parameter) :** 模型通过训练学习到的值，例如线性回归中的权重和偏置。
- **超参数 (Hyper Parameter) :** 由用户设置的参数，不能通过训练自动学习，例如学习率、正则化系数等。

## 第 2 章 机器学习基本理论

## 2.1 机器学习三要素

机器学习的方法一般主要由三部分构成：模型、策略和算法，可以认为：

$$\text{机器学习方法} = \text{模型} + \text{策略} + \text{算法}$$

- 模型（model）：总结数据的内在规律，用数学语言描述的参数系统
- 策略（strategy）：选取最优模型的评价准则
- 算法（algorithm）：选取最优模型的具体方法

## 2.2 机器学习方法分类

机器学习的方法种类繁多，并不存在一个统一的理论体系能够涵盖所有内容。从不同的角度，可以将机器学习的方法进行不同的分类：

- 通常分类：按照有无监督，机器学习可以分为 **有监督学习**、**无监督学习** 和 **半监督学习**，除此之外还有 **强化学习**。
- 按模型分类：根据模型性质，可以分为概率模型/非概率模型，线性/非线性模型等。
- 按学习技巧分类：根据算法基于的技巧，可以分为贝叶斯学习、核方法等。



### 机器学习主要分类



有监督学习



无监督学习

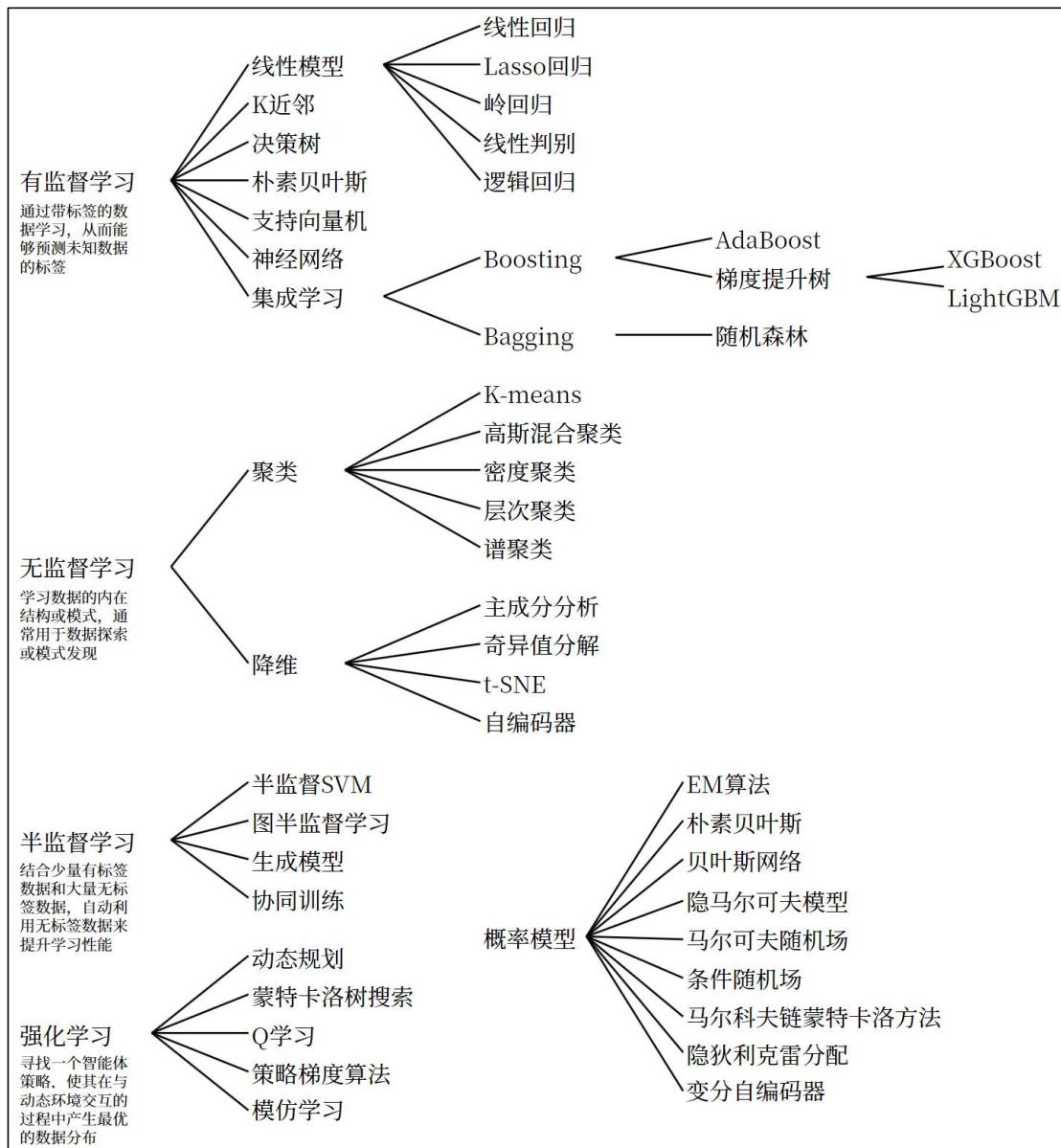


强化学习

- **有监督学习**：提供数据并提供数据对应结果的机器学习过程。
- **无监督学习**：提供数据并且不提供数据对应结果的机器学习过程。
- **强化学习**：通过与环境交互并获取延迟返回进而改进行为的学习过程。

让天下没有难学的技术

各种类型的机器学习方法可以用下图汇总展示：



## 2.3 建模流程

我们可以以监督学习为例，考察一下机器学习的具体过程。



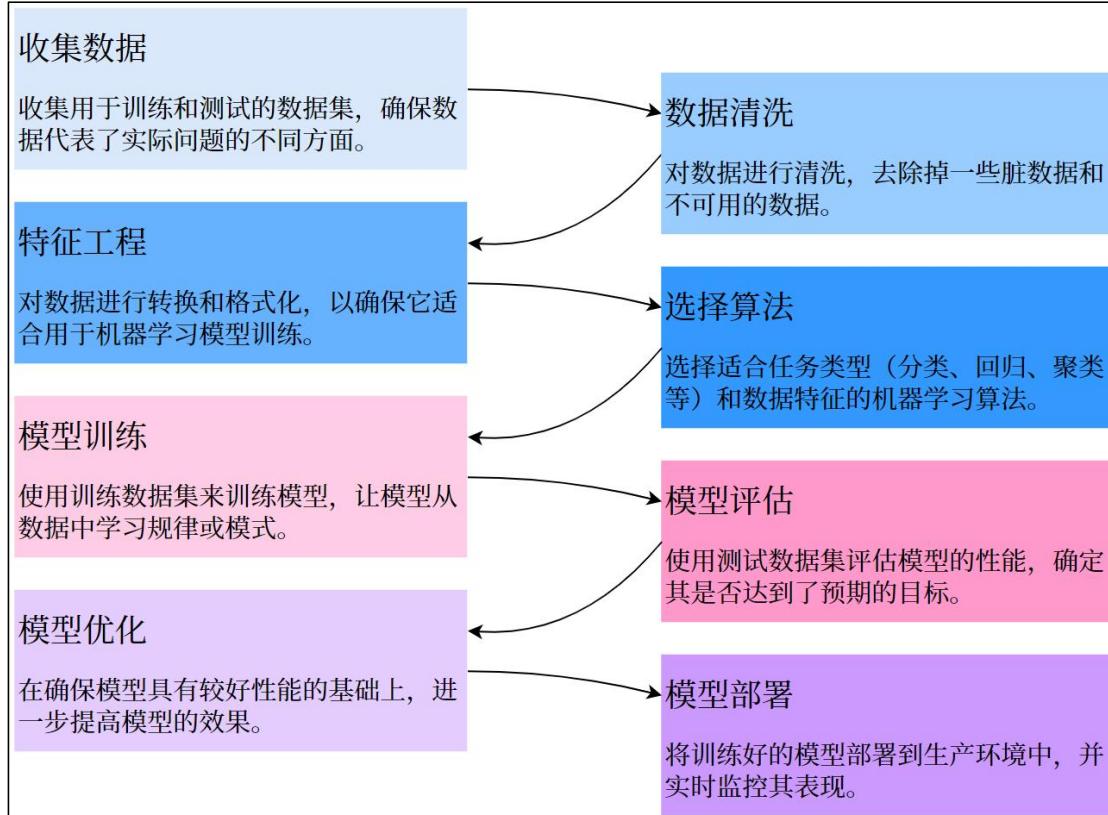
## 监督学习过程示例

	所在街区	房屋价格	住房面积	住房格局	是否学区	是否售出
	海淀	7000000	120	三室一厅	是	是
	朝阳	6000000	100	二室一厅	否	否
	昌平	5000000	120	二室一厅	否	是
	大兴	6500000	150	三室一厅	否	?

让天下没有难学的技术

可以看到，机器学习是由数据驱动的，核心是利用数据来“训练模型”；模型训练的结果需要用一定方法来进行评估、优化，最终得到一个成熟的学习模型；最后就可以用这个模型来进行预测和解决问题了。

总结监督学习建模的整体流程如下：



## 2.4 特征工程

### 2.4.1 什么是特征工程

**特征工程**（Feature Engineering）是机器学习过程中非常重要的一步，指的是通过对原始数据的处理、转换和构造，生成新的特征或选择有效的特征，从而提高模型的性能。简单来说，特征工程是将原始数据转换为可以更好地表示问题的特征形式，帮助模型更好地理解和学习数据中的规律。优秀的特征工程可以显著提高模型的表现；反之，忽视特征工程可能导致模型性能欠佳。

实际上，特征工程是一个迭代过程。特征工程取决于具体情境。它需要大量的数据分析和领域知识。其中的原因在于，特征的有效编码可由所用的模型类型、预测变量与输出之间的关系以及模型要解决的问题来确定。在此基础上，辅以不同类型的数据集（如文本与图像）则可能更适合不同的特征工程技术。因此，要具体说明如何在给定的机器学习算法中最好地实施特征工程可能并非易事。

### 2.4.2 特征工程的内容

#### 1) 特征选择

从原始特征中挑选出与目标变量关系最密切的特征，剔除冗余、无关或噪声特征。这样可以减少模型的复杂度、加速训练过程、并减少过拟合的风险。

特征选择不会创建新特征，也不会改变数据结构。

#### (1) 过滤法 (Filter Method)

基于统计测试（如卡方检验、相关系数、信息增益等）来评估特征与目标变量之间的关系，选择最相关的特征。

#### (2) 包裹法 (Wrapper Method)

使用模型（如递归特征消除 RFE）来评估特征的重要性，并根据模型的表现进行特征选择。

#### (3) 嵌入法 (Embedded Method)

使用模型本身的特征选择机制（如决策树的特征重要性，L1 正则化的特征选择）来选择最重要的特征。

## 2) 特征转换

对数据进行数学或统计处理，使其变得更加适合模型的输入要求。

#### (1) 归一化 (Normalization)

将特征缩放到特定的范围（通常是 0 到 1 之间）。适用于对尺度敏感的模型（如 KNN、SVM）。

#### (2) 标准化 (Standardization)

通过减去均值并除以标准差，使特征的分布具有均值 0，标准差 1。

#### (3) 对数变换

对于有偏态的分布（如收入、价格等），对数变换可以将其转化为更接近正态分布的形式。

#### (4) 类别变量的编码

独热编码 (One-Hot Encoding)：将类别型变量转换为二进制列，常用于无序类别特征。

标签编码 (Label Encoding)：将类别型变量映射为整数，常用于有序类别特征。

目标编码 (Target Encoding)：将类别变量的每个类别替换为其对应目标变量的平均值或其他统计量。

频率编码 (Frequency Encoding)：将类别变量的每个类别替换为该类别在数据集中的出现频率。

### 3) 特征构造

特征构造是基于现有的特征创造出新的、更有代表性的特征。通过组合、转换、或者聚合现有的特征，形成能够更好反映数据规律的特征。

#### (1) 交互特征

将两个特征组合起来，形成新的特征。例如，两个特征的乘积、和或差等。

例如，将年龄与收入结合创建新的特征，可能能更好地反映某些模式。

#### (2) 统计特征

从原始特征中提取统计值，例如求某个时间窗口的平均值、最大值、最小值、标准差等。

例如，在时间序列数据中，你可以从原始数据中提取每个小时、每日的平均值。

#### (3) 日期和时间特征

从日期时间数据中提取如星期几、月份、年份、季度等特征。

例如，将“2000-01-01”转换为“星期几”、“是否节假日”、“月初或月末”等特征。

### 4) 特征降维

当数据集的特征数量非常大时，特征降维可以帮助减少计算复杂度并避免过拟合。通过降维方法，可以在保持数据本质的情况下减少特征的数量。

#### (1) 主成分分析 (PCA)

通过线性变换将原始特征映射到一个新的空间，使得新的特征（主成分）尽可能地保留数据的方差。

#### (2) 线性判别分析 (LDA)

一种监督学习的降维方法，通过最大化类间距离与类内距离的比率来降维。

#### (3) t-SNE (t-Distributed Stochastic Neighbor Embedding, t 分布随机近邻嵌入)

一种非线性的降维技术，特别适合可视化高维数据。

#### (4) 自编码器 (Auto Encoder)

一种神经网络模型，通过压缩编码器来实现数据的降维。

### 2.4.3 常用方法

对于一个模型来说，有些特征可能很关键，而有些特征可能用处不大。

例如：

某个特征取值较接近，变化很小，可能与结果无关。

某几个特征相关性较高，可能包含冗余信息。

因此，**特征选择** 在特征工程中是最基本、也最常见的操作。

另外，在训练模型时有时也会遇到维度灾难，即特征数量过多。我们希望能在确保不丢失重要特征的前提下减少维度的数量，来降低训练模型的难度。所以在特征工程中，也经常会用到 **特征降维** 方法。

### 1) 低方差过滤法

对于特征的选择，可以直接基于方差来判断，这是最简单的。低方差的特征意味着该特征的所有样本值几乎相同，对预测影响极小，可以将其去掉。

```
from sklearn.feature_selection import VarianceThreshold

# 低方差过滤：删除方差低于 0.01 的特征
var_thresh = VarianceThreshold(threshold=0.01)
X_filtered = var_thresh.fit_transform(X)
```

### 2) 相关系数法

通过计算特征与目标变量或特征之间的相关性，筛选出高相关性特征（与目标相关）或剔除冗余特征（特征间高度相关）。

#### (1) 皮尔逊相关系数

皮尔逊相关系数（Pearson Correlation）用于衡量两个变量的线性相关性，取值范围[-1,1]。

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

- 正相关：值接近 1，说明特征随目标变量增加而增加。
- 负相关：值接近-1，说明特征随目标变量增加而减少。
- 无关：值接近 0，说明特征和目标变量无明显关系。

例如，现有一数据集包括不同渠道广告投放金额与销售额。

使用 `pandas.DataFrame.corrwith(method="pearson")` 计算各个特征与标签间的皮尔逊相关系数。

```
import pandas as pd

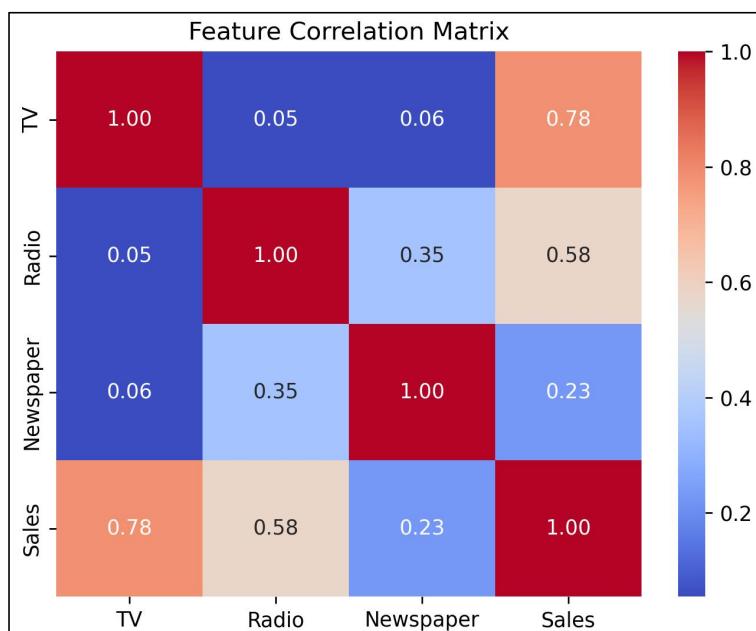
advertising = pd.read_csv("data/advertising.csv")
advertising.drop(advertising.columns[0], axis=1, inplace=True)
advertising.dropna(inplace=True)
X = advertising.drop("Sales", axis=1)
y = advertising["Sales"]
```

```
# 计算皮尔逊相关系数
print(X.corrwith(y, method="pearson"))
# TV          0.782224
# Radio       0.576223
# Newspaper   0.228299
# dtype: float64
```

使用 pandas.DataFrame.corr(method="pearson")计算皮尔逊相关系数矩阵。

```
import seaborn as sns
import matplotlib.pyplot as plt

# 计算皮尔逊相关系数矩阵
corr_matrix = advertising.corr(method="pearson")
# 可视化热力图
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Matrix")
plt.show()
```



## (2) 斯皮尔曼相关系数

斯皮尔曼相关系数（Spearman's Rank Correlation Coefficient）的定义是等级变量之间的皮尔逊相关系数。用于衡量两个变量之间的单调关系，即当一个变量增加时，另一个变量是否总是增加或减少（不要求是线性关系）。适用于非线性关系或数据不符合正态分布的情况。

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)}$$

其中：

- $d_i$ 是两个变量的等级之差

- $n$ 是样本数

斯皮尔曼相关系数的取值范围为[ -1,1]:

- $\rho = 1$ : 完全正相关 (一个变量增加, 另一个变量也总是增加)。
- $\rho = -1$ : 完全负相关 (一个变量增加, 另一个变量总是减少)。
- $\rho = 0$ : 无相关性。

例如, 现有一组每周学习时长与数学考试成绩的数据:

$X$	$y$
5	55
8	65
10	70
12	75
15	85
3	50
7	60
9	72
14	80
6	58

按数值由小到大排出 $X$ 、 $y$ 的等级, 并计算等级差:

$X$	$R_X$	$y$	$R_y$	$d = R_X - R_y$	$d^2$
5	2	55	2	0	0
8	5	65	5	0	0
10	7	70	6	1	1
12	8	75	8	0	0
15	10	85	10	0	0
3	1	50	1	0	0
7	4	60	4	0	0
9	6	72	7	-1	1
14	9	80	9	0	0
6	3	58	3	0	0

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)} = 1 - \frac{6 \times 2}{10(10^2 - 1)} = 1 - 0.0121 = 0.9879$$

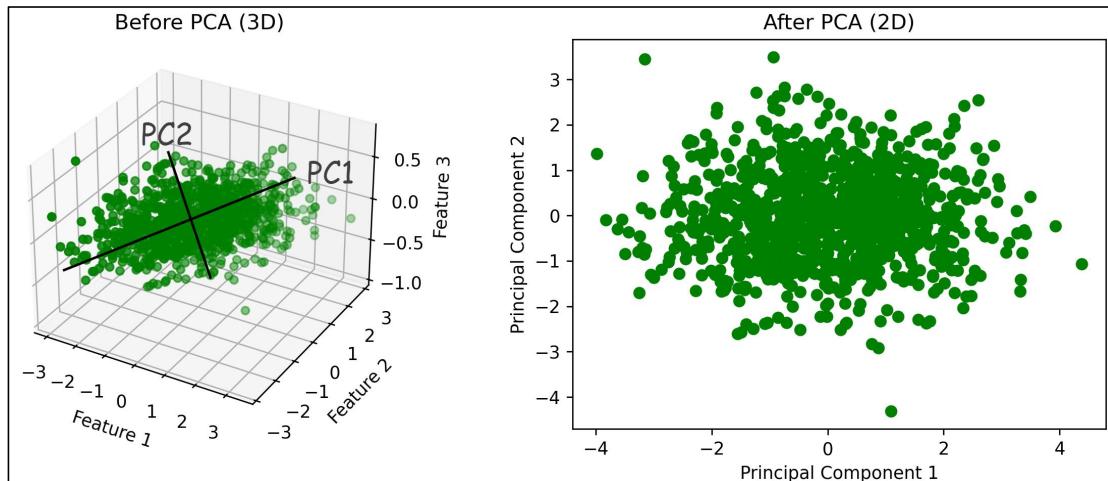
使用 `pandas.DataFrame.corrwith(method="spearman")`计算斯皮尔曼相关系数。

```
import pandas as pd

# 每周学习时长
X = [[5], [8], [10], [12], [15], [3], [7], [9], [14], [6]]
# 数学考试成绩
y = [55, 65, 70, 75, 85, 50, 60, 72, 80, 58]
# 计算斯皮尔曼相关系数
X = pd.DataFrame(X)
y = pd.Series(y)
print(X.corrwith(y, method="spearman"))
# 0.987879
```

### 3) 主成分分析 (PCA)

主成分分析 (Principal Component Analysis, PCA) 是一种常用的降维技术，通过线性变换将高维数据投影到低维空间，同时保留数据的主要变化模式。



使用 `sklearn.decomposition.PCA` 进行主成分分析。参数 `n_components` 若为小数则表示保留多少比例的信息，为整数则表示保留多少个维度。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

n_samples = 1000
# 第1个主成分方向
component1 = np.random.normal(0, 1, n_samples)
# 第2个主成分方向
component2 = np.random.normal(0, 0.2, n_samples)
# 第3个方向（噪声，方差较小）
noise = np.random.normal(0, 0.1, n_samples)
# 构造3维数据
X = np.vstack([component1 - component2, component1 + component2, component2 + noise]).T

# 标准化
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# 应用PCA，将3维数据降维到2维
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_standardized)
```

```
# 可视化
# 转换前的 3 维数据可视化
fig = plt.figure(figsize=(12, 4))
ax1 = fig.add_subplot(121, projection="3d")
ax1.scatter(X[:, 0], X[:, 1], X[:, 2], c="g")
ax1.set_title("Before PCA (3D)")
ax1.set_xlabel("Feature 1")
ax1.set_ylabel("Feature 2")
ax1.set_zlabel("Feature 3")
# 转换后的 2 维数据可视化
ax2 = fig.add_subplot(122)
ax2.scatter(X_pca[:, 0], X_pca[:, 1], c="g")
ax2.set_title("After PCA (2D)")
ax2.set_xlabel("Principal Component 1")
ax2.set_ylabel("Principal Component 2")
plt.show()
```

## 2.5 模型评估和模型选择（重点）

### 2.5.1 损失函数

对于模型一次预测结果的好坏，需要有一个度量标准。

对于监督学习而言，给定一个输入  $X$ ，选取的模型就相当于一个“决策函数” $f$ ，它可以输出一个预测结果  $f(X)$ ，而真实的结果（标签）记为  $Y$ 。 $f(X)$  和  $Y$  之间可能会有偏差，我们就用一个**损失函数**（loss function）来度量预测偏差的程度，记作  $L(Y, f(X))$ 。

- 损失函数用来衡量模型预测误差的大小；损失函数值越小，模型就越好；
- 损失函数是  $f(X)$  和  $Y$  的非负实值函数；

常见的损失函数有：

#### 1) 0-1 损失函数

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$$

#### 2) 平方损失函数

$$L(Y, f(X)) = (Y - f(X))^2$$

#### 3) 绝对损失函数

$$L(Y, f(X)) = |Y - f(X)|$$

#### 4) 对数似然损失函数

$$L(Y, P(Y|X)) = -\log P(Y|X)$$

### 2.5.2 经验误差

给定一个训练数据集，数据个数为 n:

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

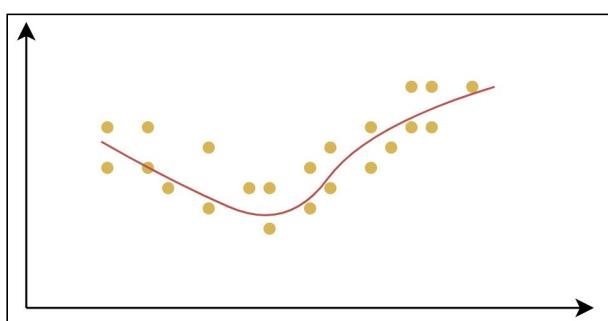
根据选取的损失函数，就可以计算出模型  $f(X)$  在训练集上的平均误差，称为训练误差，也被称作 **经验误差** (empirical error) 或 **经验风险** (empirical risk)。

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i))$$

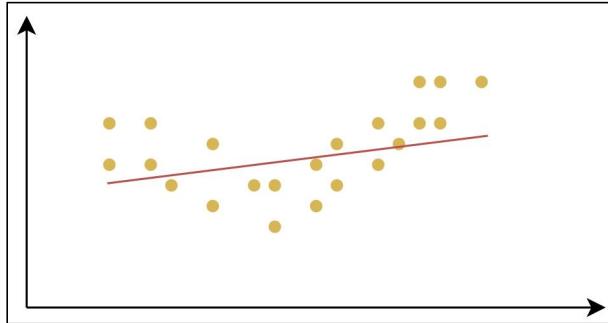
类似地，在测试数据集上平均误差，被称为测试误差或者 **泛化误差** (generalization error)。一般情况下对模型评估的策略，就是考察经验误差；当经验风险最小时，就认为取到了最优的模型。这种策略被称为 **经验风险最小化** (empirical risk minimization, ERM)。

### 2.5.3 欠拟合与过拟合

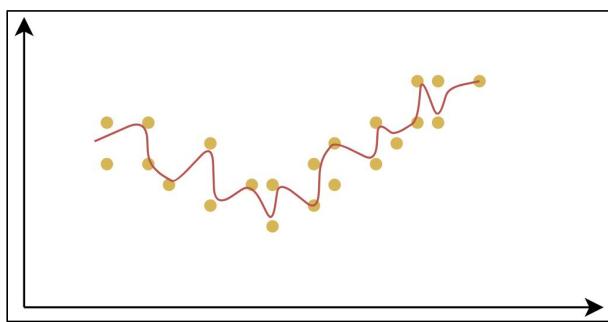
**拟合** (Fitting) 是指机器学习模型在训练数据上学习到规律并生成预测结果的过程。理想情况下，模型能够准确地捕捉训练数据的模式，并且在未见过的新数据（测试数据）上也有良好的表现；即模型具有良好的 **泛化能力**。



**欠拟合** (Underfitting)：是指模型在训练数据上表现不佳，无法很好地捕捉数据中的规律。这样的模型不仅在训练集上表现不好，在测试集上也同样表现差。

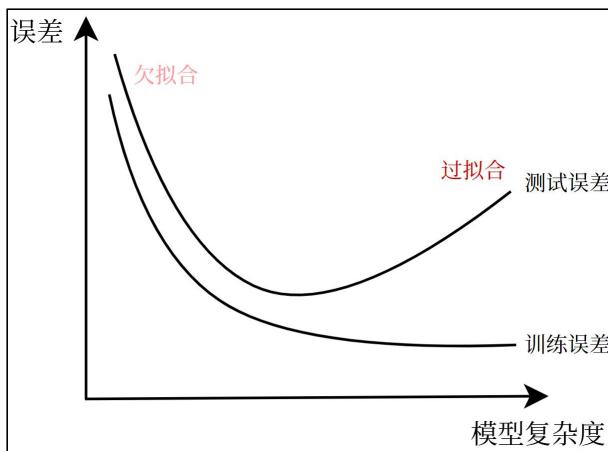


**过拟合 (Overfitting)**：是指模型在训练数据上表现得很好，但在测试数据或新数据上表现较差的情况。过拟合的模型对训练数据中的噪声或细节过度敏感，把训练样本自身的一些特点当作了所有潜在样本都会具有的一般性质，从而失去了泛化能力。



产生欠拟合和过拟合的根本原因，是模型的复杂度过低或过高，从而导致测试误差（泛化误差）偏大。

- 欠拟合：模型在训练集和测试集上误差都比较大。模型过于简单，高偏差。
- 过拟合：模型在训练集上误差较小，但在测试集上误差较大。模型过于复杂，高方差。



## 1) 产生原因与解决办法

### (1) 欠拟合

产生原因：

- 模型复杂度不足：模型过于简单，无法捕捉数据中的复杂关系。
- 特征不足：输入特征不充分，或者特征选择不恰当，导致模型无法充分学习数据的模式。
- 训练不充分：训练过程中迭代次数太少，模型没有足够的时间学习数据的规律。
- 过强的正则化：正则化项设置过大，强制模型过于简单，导致模型无法充分拟合数据。

解决办法：

- 增加模型复杂度：选择更复杂的模型。
- 增加特征或改进特征工程：添加更多的特征或通过特征工程来创造更有信息量的特征。
- 增加训练时间：增加训练的迭代次数，让模型有更多机会去学习。
- 减少正则化强度：如果使用了正则化，尝试减小正则化的权重，以让模型更灵活。

## (2) 过拟合

产生原因：

- 模型复杂度过高：模型过于复杂，参数太多。
- 训练数据不足：数据集太小，模型能记住训练数据的细节，但无法泛化到新数据。
- 特征过多：特征太多，模型可能会“记住”数据中的噪声，而不是学到真正的规律。
- 训练过长：训练时间过长，导致模型学习到训练数据中的噪声，而非数据的真正规律。

解决办法：

- 减少模型复杂度：降低模型的参数数量、使用简化的模型或降维来减小模型复杂度。
- 增加训练数据：收集更多数据，或通过数据增强来增加训练数据的多样性。
- 使用正则化：引入 L1、L2 正则化，避免过度拟合训练数据。
- 交叉验证：使用交叉验证技术评估模型在不同数据集上的表现，以减少过拟合的风险。
- 早停：训练时，当模型的验证损失不再下降时，提前停止训练，避免过度拟合训练集。

## 2) 代码演示

使用多项式在  $x \in [-3, 3]$  上拟合  $\sin(x)$ :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

plt.rcParams["font.sans-serif"] = ["KaiTi"]
plt.rcParams["axes.unicode_minus"] = False

def polynomial(x, degree):
    """构成多项式, 返回 [x^1, x^2, x^3, ..., x^n]"""
    return np.hstack([x**i for i in range(1, degree + 1)])

# 生成随机数据
X = np.linspace(-3, 3, 300).reshape(-1, 1)
y = np.sin(X) + np.random.uniform(-0.5, 0.5, 300).reshape(-1, 1)
fig, ax = plt.subplots(1, 3, figsize=(15, 4))
ax[0].plot(X, y, "yo")
ax[1].plot(X, y, "yo")
ax[2].plot(X, y, "yo")

# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建线性回归模型
model = LinearRegression()

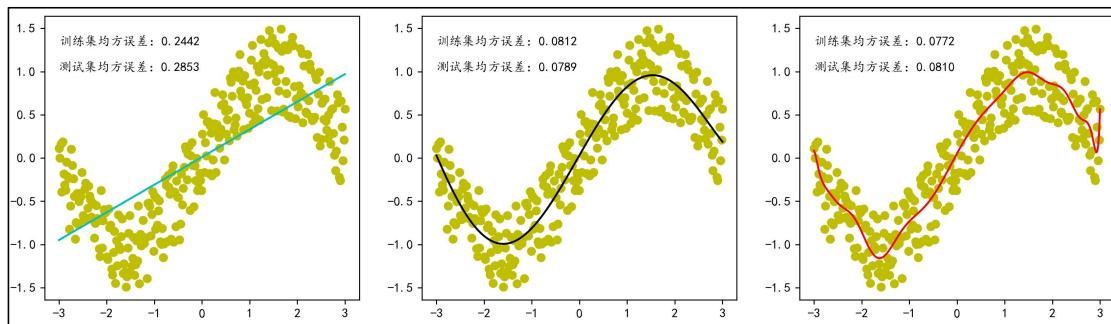
# 欠拟合
x_train1 = x_train
x_test1 = x_test
model.fit(x_train1, y_train) # 模型训练
y_pred1 = model.predict(x_test1) # 预测
ax[0].plot(np.array([[-3], [3]]), model.predict(np.array([[-3], [3]])), "c") # 绘制曲线
ax[0].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred1):.4f}")
ax[0].text(-3, 1.3, f"训练集均方误差: {mean_squared_error(y_train, model.predict(x_train1)):.4f}")
```

```

# 恰好拟合
x_train2 = polynomial(x_train, 5)
x_test2 = polynomial(x_test, 5)
model.fit(x_train2, y_train) # 模型训练
y_pred2 = model.predict(x_test2) # 预测
ax[1].plot(X, model.predict(polynomial(X, 5)), "k") # 绘制曲线
ax[1].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred2):.4f}")
ax[1].text(-3, 1.3, f"训练集均方误差: {mean_squared_error(y_train, model.predict(x_train2)):.4f}")

# 过拟合
x_train3 = polynomial(x_train, 20)
x_test3 = polynomial(x_test, 20)
model.fit(x_train3, y_train) # 模型训练
y_pred3 = model.predict(x_test3) # 预测
ax[2].plot(X, model.predict(polynomial(X, 20)), "r") # 绘制曲线
ax[2].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred3):.4f}")
ax[2].text(-3, 1.3, f"训练集均方误差: {mean_squared_error(y_train, model.predict(x_train3)):.4f}")
plt.show()

```



当多项式次数较低时，模型过于简单，拟合效果较差。

当多项式次数增加后，模型复杂度适中，拟合效果较好，训练误差和测试误差均较低。

当多项式次数继续增加，模型变得过于复杂，过度学习了噪声，导致训练误差较低而测试误差较高。

## 2.5.4 正则化

正则化 (Regularization) 是一种在训练机器学习模型时，在损失函数中添加额外项，来惩罚过大的参数，进而限制模型复杂度、避免过拟合，提高模型泛化能力的技术。

如在平方损失函数中加入正则化项  $\lambda \sum_{i=1}^k \omega_i^2$ :

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：[尚硅谷官网](#)

$$Loss = \frac{1}{n} \left( \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 + \lambda \sum_{i=1}^k \omega_i^2 \right)$$

- 原损失函数  $\sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2$  的目的：更好的拟合数据集。
- 正则化项  $\lambda \sum_{i=1}^k \omega_i^2$  的目的：减小参数的大小，从而降低模型的复杂度。

这里的  $\lambda$  是 **正则化系数**，用来表示惩罚项的权重。正则化系数不属于模型的参数，无法通过训练学习得到，需要在模型训练开始之前手动设置，这种参数被称为“**超参数**”。

两者相互平衡，在模型的拟合能力（偏差）和复杂度之间找到最佳折中。

常见的正则化技术有 L1 正则化和 L2 正则化。

### 1) L1 正则化 (Lasso 回归)

L1 正则化在损失函数中加入参数的绝对值之和：

$$Loss_{L1} = \text{原} Loss + \lambda \sum_{i=1}^k |\omega_i|$$

L1 正则化通过惩罚模型参数的绝对值，使得部分权重趋近 0 甚至变为 0。这会导致特征选择，即模型会自动“丢弃”一些不重要的特征。L1 正则化有助于创建稀疏模型（即许多参数为 0）。在解决回归问题时，使用 L1 正则化也被称为“Lasso 回归”。

$\lambda$  超参数控制着正则化的强度。较大的  $\lambda$  值意味着强烈的正则化，会使模型更简单，可能导致欠拟合。而较小的  $\lambda$  值则会使模型更复杂，可能导致过拟合。

### 2) L2 正则化 (Ridge 回归, 岭回归)

L2 正则化在损失函数中加入参数的平方之和：

$$Loss_{L2} = \text{原} Loss + \lambda \sum_{i=1}^k \omega_i^2$$

L2 正则化通过惩罚模型参数的平方，使得所有参数都变得更小，但不会将参数强行压缩为 0。它会使得模型尽量平滑，从而防止过拟合。

在解决回归问题时，使用 L2 正则化也被称为“岭回归”。

### 3) ElasticNet 正则化(弹性网络回归)

ElasticNet 正则化结合了 L1 和 L2 正则化，通过调整两个正则化项的比例来取得平衡，从而同时具备稀疏性和稳定性的优点。

$$Loss_{ElasticNet} = \text{原} Loss + \lambda \left( \alpha \sum_{i=1}^n |\omega_i| + \frac{1-\alpha}{2} \sum_{i=1}^n \omega_i^2 \right)$$

$\alpha \in [0,1]$ , 决定 L1 和 L2 的权重。

#### 4) 正则化案例

同样以使用多项式在  $x \in [-3,3]$  上拟合  $\sin(x)$  为例, 分别不使用正则化、使用 L1 正则化、使用 L2 正则化进行拟合。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error

plt.rcParams["font.sans-serif"] = ["KaiTi"]
plt.rcParams["axes.unicode_minus"] = False

def polynomial(x, degree):
    """构成多项式, 返回 [x^1,x^2,x^3,...,x^n]"""
    return np.hstack([x**i for i in range(1, degree + 1)])

# 生成随机数据
X = np.linspace(-3, 3, 300).reshape(-1, 1)
y = np.sin(X) + np.random.uniform(-0.5, 0.5, X.size).reshape(-1, 1)
fig, ax = plt.subplots(2, 3, figsize=(15, 8))
ax[0, 0].plot(X, y, "yo")
ax[0, 1].plot(X, y, "yo")
ax[0, 2].plot(X, y, "yo")

# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
x_train1 = polynomial(x_train, 20)
x_test1 = polynomial(x_test, 20)

# 拟合
model = LinearRegression()
model.fit(x_train1, y_train) # 模型训练
y_pred3 = model.predict(x_test1) # 预测
ax[0, 0].plot(X, model.predict(polynomial(X, 20)), "r") # 绘制曲线
```

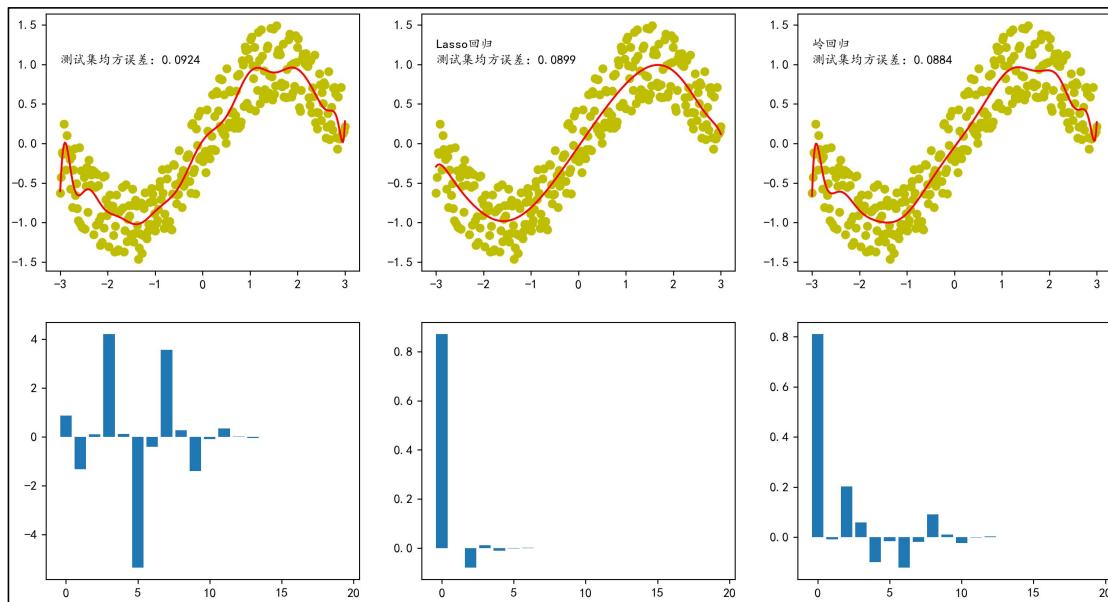
```

ax[0, 0].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred3)}")
ax[1, 0].bar(np.arange(20), model.coef_.reshape(-1)) # 绘制所有系数

# L1 正则化-Lasso 回归
lasso = Lasso(alpha=0.01)
lasso.fit(x_train1, y_train) # 模型训练
y_pred3 = lasso.predict(x_test1) # 预测
ax[0, 1].plot(X, lasso.predict(polynomial(X, 20)), "r") # 绘制曲线
ax[0, 1].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred3)}")
ax[0, 1].text(-3, 1.2, "Lasso 回归")
ax[1, 1].bar(np.arange(20), lasso.coef_) # 绘制所有系数

# L2 正则化-岭回归
ridge = Ridge(alpha=1)
ridge.fit(x_train1, y_train) # 模型训练
y_pred3 = ridge.predict(x_test1) # 预测
ax[0, 2].plot(X, ridge.predict(polynomial(X, 20)), "r") # 绘制曲线
ax[0, 2].text(-3, 1, f"测试集均方误差: {mean_squared_error(y_test, y_pred3)}")
ax[0, 2].text(-3, 1.2, "岭回归")
ax[1, 2].bar(np.arange(20), ridge.coef_) # 绘制所有系数
plt.show()

```



## 2.5.5 交叉验证

交叉验证（Cross-Validation）是一种评估模型泛化能力的方法，通过将数据集划分为多个子集，反复进行训练和验证，以减少因单次数据划分带来的随机性误差。通过交叉验证能

更多 Java -大数据 -前端 -python 人工智能资料下载，可百度访问：[尚硅谷官网](#)

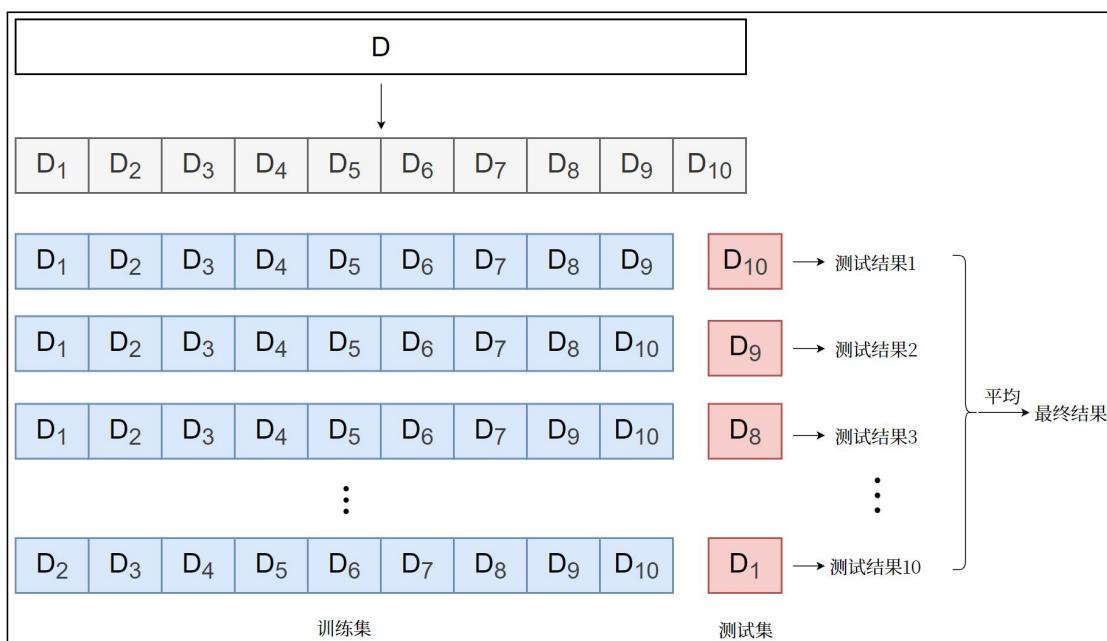
更可靠地估计模型在未知数据上的表现。亦能避免因单次数据划分不合理导致的模型过拟合或欠拟合。

### 1) 简单交叉验证 (Hold-Out Validation)

将数据划分为训练集和验证集（如 70%训练，30%验证）。结果受单次划分影响较大，可能高估或低估模型性能。

### 2) k 折交叉验证 (k-Fold Cross-Validation)

将数据均匀分为 k 个子集（称为“折”），每次用 k-1 折训练，剩余 1 折验证，重复 k 次后取平均性能。充分利用数据，结果更稳定。



### 3) 留一交叉验证 (Leave-One-Out, LOO)

每次仅留一个样本作为验证集，其余全部用于训练，重复直到所有样本都被验证一次。适用于小数据集，计算成本极高。

## 2.6 模型求解算法

正则化可以有效防止过拟合，增强模型的泛化能力。这时模型的评估策略，就是让结构化的经验风险最小，即增加了正则化项的损失函数最小，称为 **结构风险最小化** (Structural Risk Minimization, SRM)。

$$\min \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i)) + \lambda J(\theta)$$

这其实就是求解一个 **最优化问题**。代入训练集所有数据  $(x_i, y_i)$ ，要求最小值的目标函数就是模型中参数  $\theta$  的函数。

具体求解的算法，可以利用数学公式直接计算解析解，也可以使用迭代算法。

### 2.6.1 解析法

如果模型损失函数的最小值可以通过数学公式进行严格推导，得到一个解析解，那么就直接得到了最优模型的全部参数。这种方法称作解析法。

#### 1) 特点

- 适用条件：目标函数必须可导，且导数方程有解析解。
- 优点：直接且精确；计算高效；
- 缺点：适用条件较为苛刻；特征维度较大时，矩阵求逆计算复杂度极高。

#### 2) 应用示例

- 线性回归问题：可以采用“最小二乘法”求得解析解。

$$Loss_{MSE} = \frac{1}{n} (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})$$

$$\nabla Loss_{MSE} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) = 0$$

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- 线性回归 L2 正则化（Ridge 回归，岭回归）

可以得到解析解如下：

$$Loss_{L2} = \frac{1}{n} (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) + \frac{1}{n} \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$$

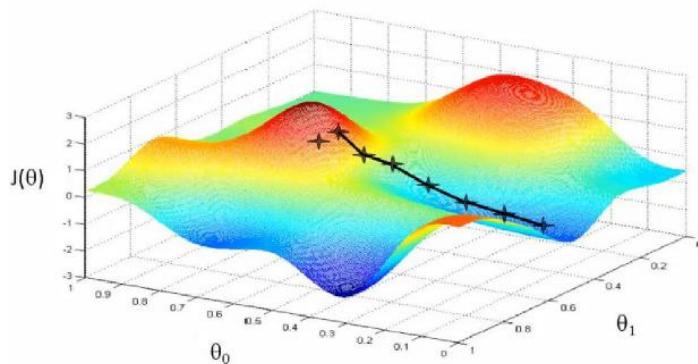
$$\nabla Loss_{L2} = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) + \frac{2}{n} \lambda \boldsymbol{\beta} = 0$$

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

由于加入的对角矩阵  $\lambda \mathbf{I}$  就像一条“山岭”，因此 L2 正则化也称作“岭回归”。

### 2.6.2 梯度下降法（重点）

**梯度下降法**（gradient descent）是一种常用的一阶优化方法，是求解无约束优化问题最简单、最经典的方法之一。梯度下降法是迭代算法，基本思路就是先选取一个适当的初始值  $\theta_0$ ，然后沿着梯度方向或者负梯度方向，不停地更新参数，最终取到极小值。



- 梯度方向：函数变化增长最快的方向（变量沿此方向变化时函数增长最快）
- 负梯度方向：函数变化减少最快的方向（变量沿此方向变化时函数减少最快）

因为损失函数是系数的函数，那么如果系数沿着损失函数的负梯度方向变化，此时损失函数减少最快，能够以最快速度下降到极小值。

$$\theta_{k+1} = \theta_k - \alpha \cdot \nabla L(\theta_k)$$

这里的 $\nabla L(\theta_k)$ 是参数取值为 $\theta_k$ 时损失函数 $L$ 的梯度； $\alpha$ 是每次迭代的“步长”，被称为“**学习率**”。学习率也是一个常见的超参数，需要手动设置，选择不当会导致收敛失败。

## 1) 特点

- 梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。
- 优点：适用性广；计算简单；
- 缺点：收敛速度慢；可能陷入局部最优。

## 2) 分类

### (1) 批量梯度下降 (Batch Gradient Descent, BGD)

每次迭代使用全部训练数据计算梯度。

- 优点：稳定收敛。
- 缺点：计算开销大。

### (2) 随机梯度下降 (Stochastic Gradient Descent, SGD)

每次迭代随机选取一个样本计算梯度。

- 优点：速度快，适合大规模数据。
- 缺点：梯度更新方向不稳定，优化过程震荡较大，可能难以收敛。

### (3) 小批量梯度下降 (Mini-batch Gradient Descent, MBGD)

每次迭代使用一小批样本（如 32、64 个）计算梯度。

平衡了 BGD 的稳定性和 SGD 的速度，是最常用的方法。

### 3) 梯度下降法计算步骤

- (1) 初始化参数：随机选择初始参数
- (2) 计算梯度：在当前参数下，计算损失函数的梯度
- (3) 更新参数：沿负梯度方向调整参数
- (4) 重复迭代：直到满足停止条件（如梯度接近零、达到最大迭代次数等）

### 4) 代码实现

我们以一个单变量函数为例，介绍梯度下降法的代码实现。

设  $f(x) = x^2$ ，求  $x$  为何值时， $f(x) = 2$ 。

目标函数  $J(x) = (f(x) - 2)^2 = (x^2 - 2)^2$ ，原问题等价于求  $x$  为何值时目标函数取得最小值。

使用梯度下降法求解。

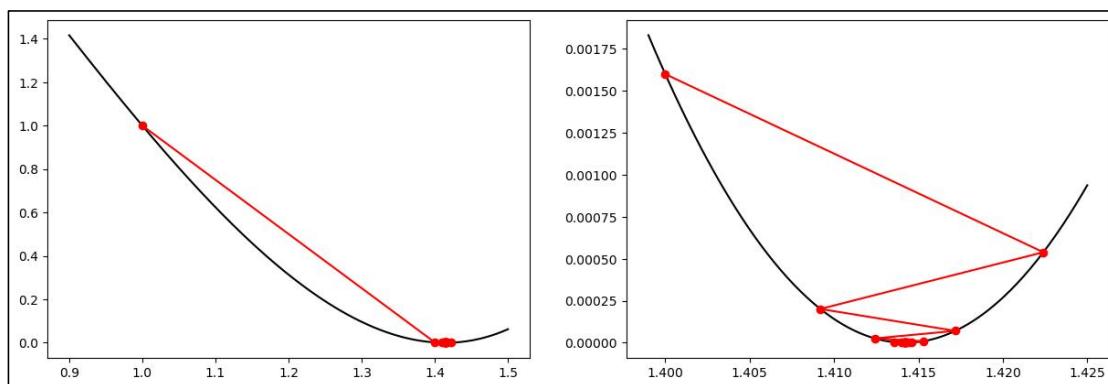
- $x$  初始值取 1
- 学习率  $\alpha$  取 0.1
- $\frac{dJ(x)}{dx} = 2 \times (x^2 - 2) \times 2 \times x = 4x^3 - 8x$

第 1 步： $x_1 = 1$ ,  $J(x_1) = 1$ ,  $J(x_1)' = -4$ ,  $x_2 = x_1 - \alpha J(x_1)' = 1.4$

第 2 步： $x_2 = 1.4$ ,  $J(x_2) = 0.0016$ ,  $J(x_2)' = -0.2240$ ,  $x_3 = x_2 - \alpha J(x_2)' = 1.4224$

⋮

第 n 步： $x_n = 1.414213$ ,  $J(x_n) = 2.53 \times 10^{-12}$ ,  $J(x_n)' = 8.8817 \times 10^{-15}$



示例代码：

```
def J(x):
    """目标函数"""
    return (x**2 - 2) ** 2
```

```

def gradient(x):
    """梯度"""
    return 4 * x**3 - 8 * x

x = 1 # x 的初始值
alpha = 0.1 # 学习率
while (j := J(x)) > 1e-30: # 当目标函数的值小于 10 的-30 次幂时停止计算
    print(f"x={x}\tJ={j}")
    grad = gradient(x) # 求解梯度
    x = x - alpha * grad # 更新参数

```

## 5) 应用示例

- L1 正则化 (Lasso 回归)

梯度下降法求解的推导过程如下：

$$\begin{aligned}
 Loss_{L1} &= \frac{1}{n} \left( \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 + \lambda \sum_{j=1}^k |\omega_j| \right) \\
 \frac{\partial Loss_{L1}}{\partial \omega_j} &= \frac{1}{n} \left( 2 \sum_{i=1}^n x_{ij}(f(\mathbf{x}_i) - y_i) + \lambda \cdot \text{sign}(\omega_j) \right) \\
 \text{其中 } \text{sign}(\omega_j) &= \begin{cases} 1, & \omega_j > 0 \\ 0, & \omega_j = 0 \\ -1, & \omega_j < 0 \end{cases}
 \end{aligned}$$

$$\text{参数更新: } \omega_j \leftarrow \omega_j - \alpha \left( \frac{2}{n} \sum_{i=1}^n x_{ij}(f(\mathbf{x}_i) - y_i) + \frac{\lambda}{n} \cdot \text{sign}(\omega_j) \right)$$

可见 L1 正则化项的梯度是一个常数  $\frac{\lambda}{n}$ , 当  $\omega_j$  很小时会直接变成 0, 导致稀疏性。

- L2 正则化 (Ridge 回归, 岭回归)

梯度下降法求解的推导过程如下：

$$\begin{aligned}
 Loss_{L2} &= \frac{1}{n} \left( \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 + \lambda \sum_{j=1}^k \omega_j^2 \right) \\
 \frac{\partial Loss_{L2}}{\partial \omega_j} &= \frac{1}{n} \left( 2 \sum_{i=1}^n x_{ij}(f(\mathbf{x}_i) - y_i) + 2\lambda\omega_j \right)
 \end{aligned}$$

$$\text{梯度更新: } \omega_j \leftarrow \omega_j - \alpha \left( \frac{2}{n} \sum_{i=1}^n x_{ij}(f(\mathbf{x}_i) - y_i) + \frac{2\lambda}{n} \omega_j \right)$$

可见 L2 正则化项的梯度是  $\frac{2\lambda}{n} \omega_j$ , 相当于在每次更新时都对  $\omega_j$  进行缩小, 但不会直接变为 0。

### 2.6.3 牛顿法和拟牛顿法（了解）

牛顿法也是求解无约束最优化问题的常用方法, 核心思想是利用目标函数的二阶导数信息, 通过迭代逐渐逼近极值点。

$$\theta_{k+1} = \theta_k - H^{-1}(\theta_k) \cdot \nabla L(\theta_k)$$

这里的  $H^{-1}(\theta_k)$  表示损失函数  $L$  黑塞矩阵的逆在点  $\theta_k$  的取值。

- 优点: 收敛速度快; 精度高;
- 缺点: 计算复杂; 可能发散。

由于牛顿法中需要计算黑塞矩阵的逆  $H^{-1}(\theta_k)$ , 这一步比较复杂; 所以可以考虑用一个  $n$  阶正定矩阵来近似代替它, 这种方法称为“拟牛顿法”。

牛顿法和拟牛顿法一般用于解决中小规模的凸优化问题。

## 2.7 模型评价指标（重点）

对学习的泛化性能进行评估, 不仅需要有效可行的实验估计方法, 还需要有衡量模型泛化能力的评价指标, 也叫做性能度量 (performance measure)。

### 2.7.1 回归模型评价指标

模型的评价指标用于衡量模型在训练集或测试集上的性能, 评估结果反映了模型预测的准确性和泛化能力。

对于回归问题, 最常用的性能度量是“均方误差” (Mean Squared Error, MSE)。

#### 1) 平均绝对误差 (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |f(\mathbf{x}_i) - y_i|$$

➤ MAE 对异常值不敏感, 解释直观。适用于数据包含异常值的场景。

#### 2) 均方误差 (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2$$

➤ MSE 会放大较大误差，对异常值敏感。适用于需要惩罚大误差的场景。

### 3) 均方根误差 (RMSE)

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2}$$

➤ 与 MSE 类似，但量纲与目标变量一致。适用于需要直观误差量纲的场景。如果一味地试图降低 RMSE，可能会导致模型对异常值也拟合度也很高，容易过拟合。

### 4) R<sup>2</sup> (决定系数)

$$R^2 = 1 - \frac{\sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

➤ 测量模型对目标变量的解释能力，越接近 1 越好，对异常值敏感。

## 2.7.2 分类模型评价指标

对于分类问题，最常用的指标就是“准确率”（Accuracy），它定义为分类器对测试集正确分类的样本数与总样本数之比。此外还有一系列常用的评价指标。

### 1) 混淆矩阵

混淆矩阵（Confusion Matrix）是用于评估分类模型性能的工具，展示了模型预测结果与实际标签的对比情况。

对于二分类问题，混淆矩阵是一个  $2 \times 2$  矩阵：

预测值 真实值	正例	负例
正例	真正例 (True Positive, TP)	假负例 (False Negative, FN)
负例	假正例 (False Positive, FP)	真负例 (True Negative, TN)

例如，有 10 个样本。6 个是猫，4 个是狗。假设以猫为正例，模型预测对了 5 个猫，2 个狗。

		预测值	猫	狗
真实值	猫	5	1	
	狗	2	2	

使用 `sklearn.metrics.confusion_matrix` 查看混淆矩阵：

```
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix

label = ["猫", "狗"] # 标签
y_true = ["猫", "猫", "猫", "猫", "猫", "猫", "狗", "狗", "狗", "狗"] # 真实值
y_pred1 = ["猫", "猫", "狗", "猫", "猫", "猫", "猫", "猫", "狗", "狗"] # 预测值
matrix1 = confusion_matrix(y_true, y_pred1, labels=label) # 混淆矩阵
print(pd.DataFrame(matrix1, columns=label, index=label))
sns.heatmap(matrix1, annot=True, fmt='d', cmap='Greens')
```

## 2) 准确率 (Accuracy)

正确预测的比例。

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

		正例	负例
真实值	正例	真正例 (True Positive, TP)	假负例 (False Negative, FN)
	负例	假正例 (False Positive, FP)	真负例 (True Negative, TN)

上述案例中，准确率  $= \frac{5+2}{10} = 0.7$ 。

```
from sklearn.metrics import accuracy_score

label = ["猫", "狗"] # 标签
y_true = ["猫", "猫", "猫", "猫", "猫", "猫", "狗", "狗", "狗", "狗"] # 真实值
y_pred1 = ["猫", "猫", "狗", "猫", "猫", "猫", "猫", "猫", "狗", "狗"] # 预测值
accuracy = accuracy_score(y_true, y_pred1)
```

```
print(accuracy)
```

### 3) 精确率 (Precision)

预测为正例的样本中实际为正例的比例，也叫查准率。

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

预测值 \ 真实值	正例	负例
正例	真正例 (True Positive, TP)	假负例 (False Negative, FN)
负例	假正例 (False Positive, FP)	真负例 (True Negative, TN)

上述案例中，精确率  $= \frac{5}{5+2} = 0.7143$ 。

```
from sklearn.metrics import precision_score

label = ["猫", "狗"] # 标签
y_true = ["猫", "猫", "猫", "猫", "猫", "猫", "狗", "狗", "狗", "狗"] # 真实值
y_pred1 = ["猫", "猫", "狗", "猫", "猫", "猫", "猫", "猫", "狗", "狗"] # 预测值
precision = precision_score(y_true, y_pred1, pos_label="猫") # pos_label 指定正例
print(precision)
```

### 4) 召回率 (Recall)

实际为正类的样本中预测为正类的比例，也叫查全率。

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

预测值 \ 真实值	正例	负例
正例	真正例 (True Positive, TP)	假负例 (False Negative, FN)
负例	假正例 (False Positive, FP)	真负例 (True Negative, TN)

上述案例中，召回率  $= \frac{5}{5+1} = 0.8333$ 。

```
from sklearn.metrics import recall_score

label = ["猫", "狗"] # 标签
```

```

y_true = ["猫", "猫", "猫", "猫", "猫", "猫", "狗", "狗", "狗", "狗"] # 真实值
y_pred1 = ["猫", "猫", "狗", "猫", "猫", "猫", "猫", "猫", "狗", "狗"] # 预测值
recall = recall_score(y_true, y_pred1, pos_label="猫") # pos_label 指定正例
print(recall)

```

## 5) F1 分数 (F1 Score)

精确率和召回率的调和平均。

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

上述案例中, F1 分数 =  $\frac{2 \times \frac{5}{5+2} \times \frac{5}{5+1}}{\frac{5}{5+2} + \frac{5}{5+1}} = 0.7692$ 。

```

from sklearn.metrics import f1_score

label = ["猫", "狗"] # 标签
y_true = ["猫", "猫", "猫", "猫", "猫", "猫", "狗", "狗", "狗", "狗"] # 真实值
y_pred1 = ["猫", "猫", "狗", "猫", "猫", "猫", "猫", "猫", "狗", "狗"] # 预测值
f1 = f1_score(y_true, y_pred1, pos_label="猫") # pos_label 指定正例
print(f1)

```

在代码中, 我们可通过 `sklearn.metrics.classification_report` 生成分类任务的评估报告, 包括精确率、召回率、F1 分数等。

```

from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, labels=[], target_names=None)
# y_true: 真实标签
# y_pred: 预测的标签
# labels: 可选, 指定需要计算的类别列表 (默认计算所有出现过的类别)
# target_names: 可选, 类别名称 (默认使用 labels 指定的类别号)

```

例如:

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# 生成一个二分类数据集
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=100)

```

```
# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=100)

# 训练一个逻辑回归模型
model = LogisticRegression()
model.fit(x_train, y_train)

# 预测
y_pred = model.predict(x_test)

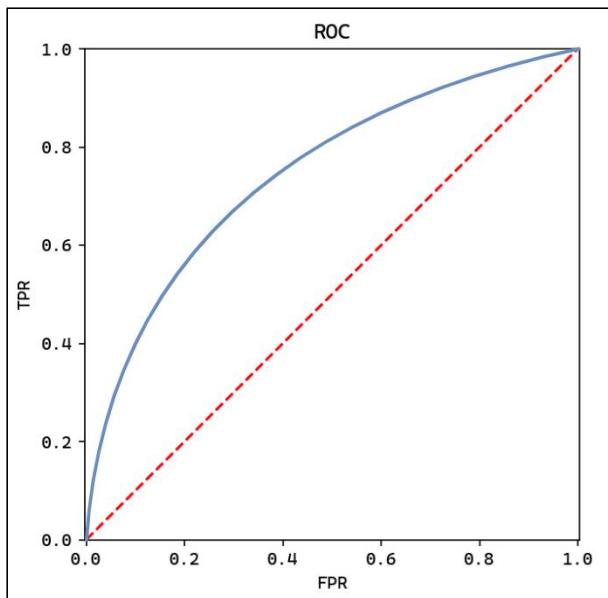
# 生成分类报告
report = classification_report(y_test, y_pred)
print(report)
```

## 6) ROC 曲线

- 真正例率 (TPR)：实际为正例，被预测为正例的比例，即召回率。
- 假正例率 (FPR)：实际为负例，被预测为正例的比例。
- 阈值 (Threshold)：根据阈值将概率转换为类别标签。

$$TPR = \frac{TP}{\text{实际正例数}}$$
$$FPR = \frac{FP}{\text{实际负例数}}$$

ROC 曲线 (Receiver Operating Characteristic Curve, 受试者工作特征) 是评估二分类模型性能的工具，以假正例率 (FPR) 为横轴，以真正例率 (TPR) 为纵轴，展示不同阈值下模型的表现。绘制 ROC 曲线时，从高到低调整阈值，计算每个阈值的 TPR 和 FPR 并绘制所有阈值的点，形成 ROC 曲线。



## 7) AUC 值

AUC 值代表 ROC 曲线下的面积，用于量化模型性能。AUC 值越大，模型区分正负类的能力越强，模型性能越好。AUC 值=0.5 表示模型接近随机猜测，AUC 值=1 代表完美模型。

可通过 `sklearn.metrics.roc_auc_score` 计算 AUC 值。

```
from sklearn.metrics import roc_auc_score

auc_score = roc_auc_score(y_true, y_score)
# y_true: 真实标签 (0 或 1)
# y_score: 正例的概率值或置信度
```

例如：

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# 生成一个二分类数据集
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=100)

# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=100)

# 训练一个逻辑回归模型
model = LogisticRegression()
```

```

model.fit(x_train, y_train)

# 预测概率值（取正类的概率）
y_pred_proba = model.predict_proba(x_test)[:, 1]

# 计算 AUC
auc_score = roc_auc_score(y_test, y_pred_proba)
print(auc_score)

```

## 8) 案例：绘制 ROC 曲线

假设一个二分类模型的真实标签和模型输出概率如下：

样本	真实标签	模型输出概率		样本	真实标签	模型输出概率
1	1	0.9		1	1	0.9
2	0	0.4		4	1	0.8
3	1	0.6	按模型输出概率排序 →	6	0	0.7
4	1	0.8		3	1	0.6
5	0	0.2		8	0	0.5
6	0	0.7		2	0	0.4
7	1	0.3		7	1	0.3
8	0	0.5		5	0	0.2

调整阈值，计算 TPR 和 FPR：

$$\text{阈值} = 0.9: TPR = \frac{0}{4} = 0, FPR = \frac{0}{4} = 0, \text{点坐标}(0, 0)$$

$$\text{阈值} = 0.8: TPR = \frac{1}{4} = 0.25, FPR = \frac{0}{4} = 0, \text{点坐标}(0, 0.25)$$

$$\text{阈值} = 0.7: TPR = \frac{2}{4} = 0.5, FPR = \frac{0}{4} = 0, \text{点坐标}(0, 0.5)$$

$$\text{阈值} = 0.6: TPR = \frac{2}{4} = 0.5, FPR = \frac{1}{4} = 0.25, \text{点坐标}(0.25, 0.5)$$

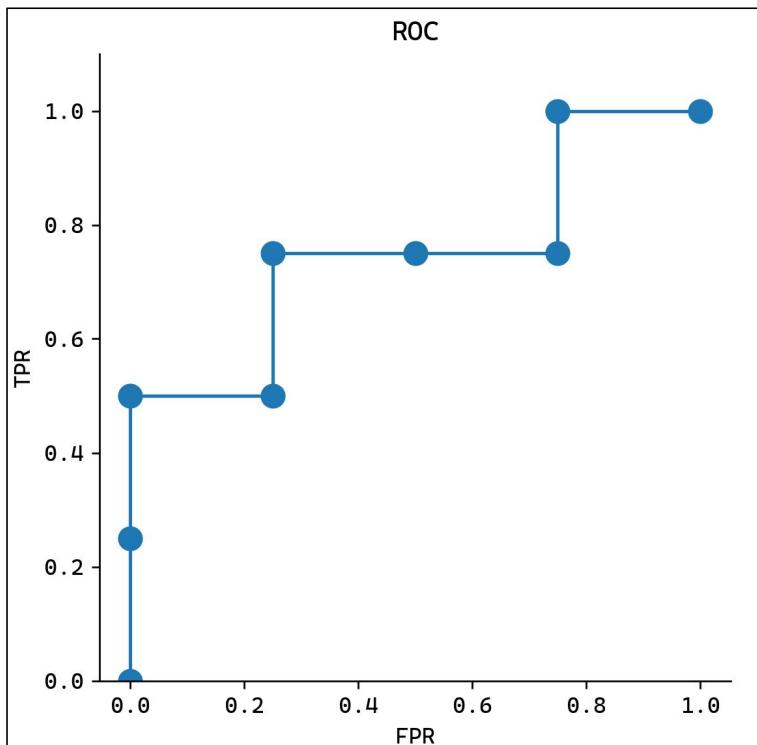
$$\text{阈值} = 0.5: TPR = \frac{3}{4} = 0.75, FPR = \frac{1}{4} = 0.25, \text{点坐标}(0.25, 0.75)$$

$$\text{阈值} = 0.4: TPR = \frac{3}{4} = 0.75, FPR = \frac{2}{4} = 0.5, \text{点坐标}(0.5, 0.75)$$

$$\text{阈值} = 0.3: TPR = \frac{3}{4} = 0.75, FPR = \frac{3}{4} = 0.75, \text{点坐标}(0.75, 0.75)$$

$$\text{阈值} = 0.2: TPR = \frac{4}{4} = 1, FPR = \frac{3}{4} = 0.75, \text{点坐标}(0.75, 1)$$

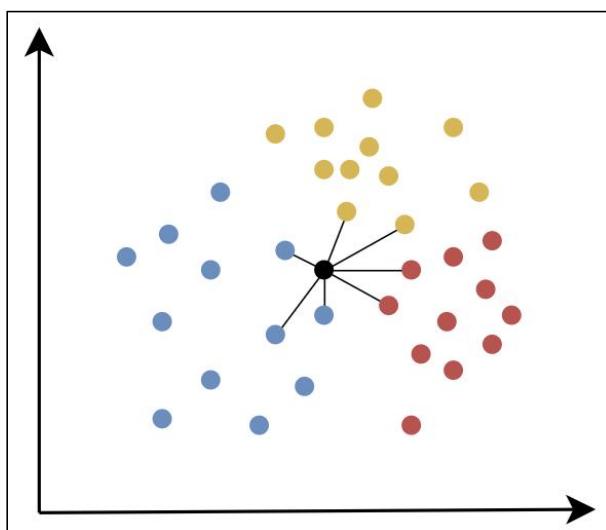
根据坐标点绘制 ROC 曲线：



## 第 3 章 KNN 算法

### 3.1 KNN 算法介绍

K 近邻算法（K-Nearest Neighbors, KNN）是一种基本的分类与回归方法，属于监督学习算法。其核心思想是通过计算给定样本与数据集中所有样本的距离，找到距离最近的 K 个样本，然后根据这 K 个样本的类别或值来预测当前样本的类别或值。



#### 3.1.1 工作原理

计算距离：计算待分类样本与训练集中每个样本的距离。

选择 K 个近邻：根据计算的距离，选择距离最近的 K 个样本。

投票或平均：

- 分类任务：统计 K 个近邻各类别的数量，将待分类样本归为数量最多的类别。
- 回归任务：取 K 个近邻的平均值作为预测结果。

### 3.1.2 关键参数

距离度量方法：选择合适的距离度量方法，常见的有欧氏距离、曼哈顿距离、切比雪夫距离、闵可夫斯基距离等。

K 值：K 值的选择对结果影响很大。K 值过小容易过拟合，K 值过大则可能欠拟合。

### 3.1.3 优缺点

KNN 优点：

- 简单直观，易于理解和实现。
- 无需训练过程，直接利用训练数据进行预测。

KNN 缺点：

- 计算量大，尤其是训练集较大时。
- 对噪声数据较敏感。

### 3.1.4 API 使用

#### 1) 分类

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2) # KNN 分类模型，K 值为 2
X = [[2, 1], [3, 1], [1, 4], [2, 6]] # 特征
y = [0, 0, 1, 1] # 标签
knn.fit(X, y) # 模型训练
knn.predict([[4, 9]]) # 预测
```

#### 2) 回归

```
from sklearn.neighbors import KNeighborsRegressor

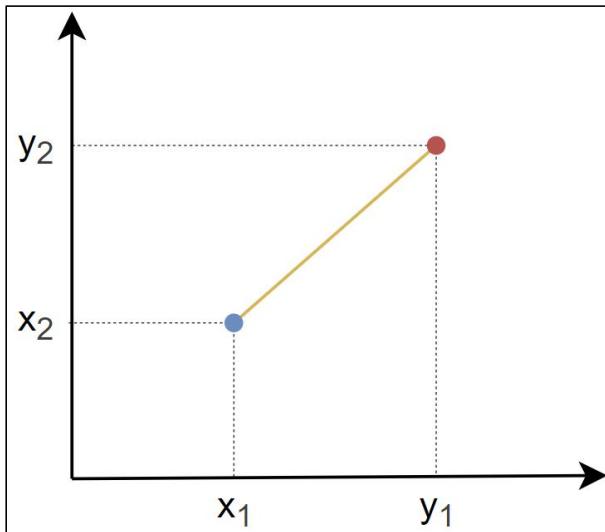
knn = KNeighborsRegressor(n_neighbors=2) # KNN 回归模型，K 值为 2
X = [[2, 1], [3, 1], [1, 4], [2, 6]] # 特征
y = [0.5, 0.33, 4, 3] # 标签
knn.fit(X, y) # 模型训练
```

```
knn.predict([[4, 9]]) # 预测
```

## 3.2 常见距离度量方法（了解）

### 3.2.1 欧氏距离

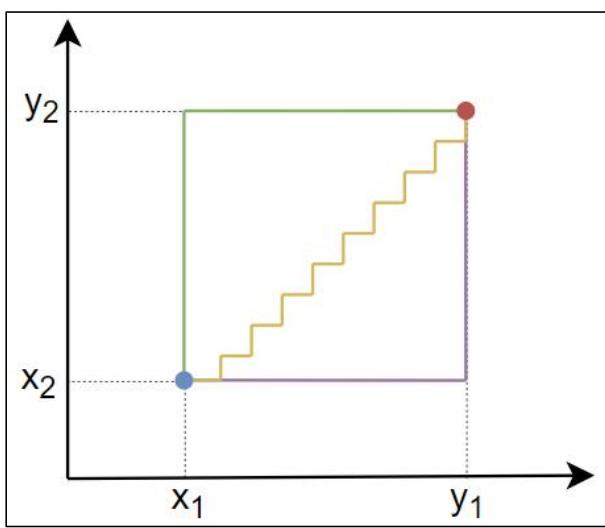
欧几里得距离（Euclidean Distance）是指连接两点的线段的长度。



点 $x(x_1, \dots, x_n)$ 和 $y(y_1, \dots, y_n)$ 之间的欧氏距离 $d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$ 。

### 3.2.2 曼哈顿距离

曼哈顿距离（Manhattan Distance）是两点在标准坐标系上的绝对轴距之和。



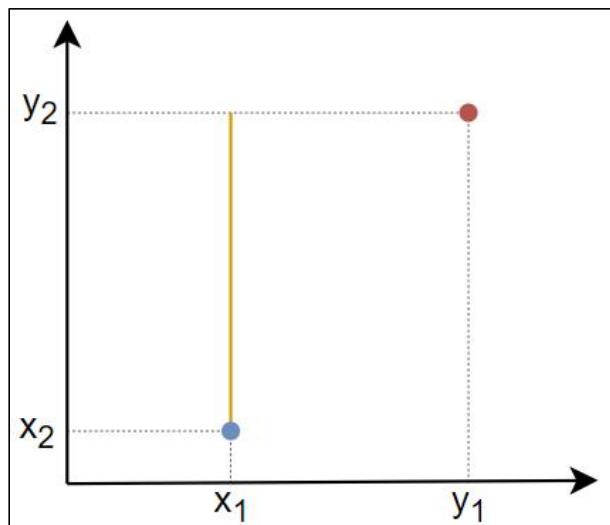
点 $x(x_1, \dots, x_n)$ 和 $y(y_1, \dots, y_n)$ 之间的曼哈顿距离 $d(x, y) = \sum_{i=1}^n |x_i - y_i|$ 。

曼哈顿距离得名于纽约曼哈顿的街道布局。由于曼哈顿的街道多为规则的网格状，车辆只能沿水平和垂直方向行驶，无法直接斜穿。因此，两点之间的实际行驶距离是沿街道行走的距离，而非直线距离。



### 3.2.3 切比雪夫距离

切比雪夫距离（Chebyshev Distance）是两点各坐标数值差的最大值。



点 $x(x_1, \dots, x_n)$ 和 $y(y_1, \dots, y_n)$ 之间的切比雪夫距离 $d(x, y) = \max (|x_i - y_i|)$ 。

在国际象棋中，国王可以横向、纵向或斜向移动一格。其从起点到终点的最少步数就等于两点之间的切比雪夫距离。

	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1	1	1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

### 3.2.4 阎可夫斯基距离

阎可夫斯基距离（Minkowski Distance）是一种用于度量多维空间中两点间距离的通用方法，点 $x(x_1, \dots, x_n)$ 和 $y(y_1, \dots, y_n)$ 之间的阎可夫斯基距离 $d(x, y) = (\sum_{i=1}^n (|x_i - y_i|)^p)^{\frac{1}{p}}$ 。 $p$ 越小，对多个维度的差异更敏感； $p$ 越大，更关注最大维度的差异。

通过调整参数 $p$ ，阎可夫斯基距离可以退化为以下经典距离：

- 曼哈顿距离： $p = 1$ ,  $d(x, y) = \sum_{i=1}^n |x_i - y_i|$ 。
- 欧氏距离： $p = 2$ ,  $d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$ 。
- 切比雪夫距离： $p = \infty$ ,  $d(x, y) = \max(|x_i - y_i|)$ 。

## 3.3 归一化与标准化

### 3.3.1 归一化

#### 1) 定义

将数据按比例缩放到一个固定范围 $[x_{min}, x_{max}]$ （通常是 $[0, 1]$ 或 $[-1, 1]$ ）。

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

$$x' = 2 \times \frac{x - x_{min}}{x_{max} - x_{min}} - 1$$

#### 2) 目的

- 消除量纲差异：不同特征的单位或量纲可能差异巨大（例如身高以米为单位，体重以千克为单位），归一化可消除这种差异，避免模型被大范围特征主导。

- 加速模型收敛：对于梯度下降等优化算法，归一化后特征处于相近的尺度，优化路径更平滑，收敛速度更快。
- 适配特定模型需求：某些模型（如神经网络、K 近邻、SVM）对输入数据的范围敏感，归一化能显著提升其性能。

### 3) 场景

归一化不改变原始分布形状，但对异常值比较敏感。当数据分布有明显边界（如图像像素值、文本词频），或模型对输入范围敏感时可以优先考虑归一化。

### 4) API 使用

```
from sklearn.preprocessing import MinMaxScaler

X = [[2, 1], [3, 1], [1, 4], [2, 6]]
# 归一化，区间设置为(-1,1)
X = MinMaxScaler(feature_range=(-1, 1)).fit_transform(X)
print(X)
```

## 3.3.2 标准化

### 1) 定义

将数据调整为均值为 0、标准差为 1 的标准分布。

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

其中  $\mu = \frac{\sum_{i=1}^n x_i}{n}$  是平均值， $\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)^2}{n}}$  是标准差。

### 2) 目的

适应数据分布：将数据转换为均值为 0、标准差为 1 的分布，适合假设数据服从正态分布的模型（如线性回归、逻辑回归）。

稳定模型训练：标准化后的数据对异常值的敏感度较低（相比归一化），鲁棒性更强。

统一特征尺度：与归一化类似，标准化也能消除量纲差异，但更关注数据的统计分布而非固定范围。

### 3) 场景

大多数场景下标准化更通用，尤其是数据分布未知或存在轻微异常值时。

### 4) API 使用

```
from sklearn.preprocessing import StandardScaler

X = [[2, 1], [3, 1], [1, 4], [2, 6]]
```

```
# 标准化
X = StandardScaler().fit_transform(X)
print(X)
```

## 3.4 案例：心脏病预测

### 3.4.1 数据集说明

Heart Disease 数据集 <https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>。

- 年龄：连续值
- 性别：0-女， 1-男
- 胸痛类型：0-典型心绞痛， 1-非典型心绞痛， 2-非心绞痛， 3-无症状
- 静息血压：连续值， 单位 mmHg
- 胆固醇：连续值， 单位 mg/dl
- 空腹血糖：1-大于 120mg/dl, 0-小于等于 120mg/dl
- 静息心电图结果：0-正常， 1-ST-T 异常， 2-可能左心室肥大
- 最大心率：连续值
- 运动性心绞痛：1-有， 0-无
- 运动后的 ST 下降：连续值
- 峰值 ST 段的斜率：0-向上， 1-水平， 2-向下
- 主血管数量：0 到 3
- 地中海贫血：一种先天性贫血， 1-正常， 2-固定缺陷， 3-可逆缺陷
- 是否患有心脏病：标签， 0-否， 1-是

### 3.4.2 加载数据集

```
import pandas as pd

# 加载数据集
heart_disease = pd.read_csv("data/heart_disease.csv")
# 处理缺失值
heart_disease.dropna()
heart_disease.info()
heart_disease.head()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   年龄        1025 non-null   int64  
 1   性别        1025 non-null   int64  
 2   胸痛类型    1025 non-null   int64  
 3   静息血压    1025 non-null   int64  
 4   胆固醇     1025 non-null   int64  
 5   空腹血糖    1025 non-null   int64  
 6   静息心电图结果 1025 non-null   int64  
 7   最大心率    1025 non-null   int64  
 8   运动性心绞痛 1025 non-null   int64  
 9   运动后的ST下降 1025 non-null   float64 
 10  峰值ST段的斜率 1025 non-null   int64  
 11  主血管数量  1025 non-null   int64  
 12  地中海贫血  1025 non-null   int64  
 13  是否患有心脏病 1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB

```

	年龄	性别	胸痛类型	静息血压	胆固醇	空腹血糖	静息心电图结果	最大心率	运动性心绞痛	运动后的ST下降	峰值ST段的斜率	主血管数量	地中海贫血	是否患有心脏病
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

### 3.4.3 数据集划分

```

from sklearn.model_selection import train_test_split

# 划分特征和标签
X = heart_disease.drop("是否患有心脏病", axis=1) # 特征
y = heart_disease["是否患有心脏病"] # 标签
# 将数据集按 7:3 划分为训练数据与测试数据
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=100)

```

### 3.4.4 特征工程

#### 1) 特征转换

数据集中包含多种类型的特征：

- 类别型特征（需要特殊处理）
  - 胸痛类型：4 种分类（名义变量）
  - 静息心电图结果：3 种分类（名义变量）
  - 峰值 ST 段的斜率：3 种分类（有序变量）
  - 地中海贫血：3 种分类（名义变量）
- 数值型特征（可直接标准化）：年龄、静息血压、胆固醇、最大心率、运动后的 ST 下降、主血管数量
- 二元特征（保持原样）：性别、空腹血糖、运动性心绞痛

对于类别型特征，直接使用整数编码的类别特征会被算法视为有序数值，导致错误的距离计算（例如：会认为 胸痛类型=1 和 胸痛类型=2 之间的差异比 胸痛类型=1 和 胸痛类

型=3 之间差异更小，而实际上它们都是类别）。使用 **独热编码**（One-Hot Encoding）可将类别特征转换为二元向量，消除虚假的顺序关系。

胸痛类型		胸痛类型_0	胸痛类型_1	胸痛类型_2	胸痛类型_3
0		1	0	0	0
1		0	1	0	0
2		0	0	1	0
3		0	0	0	1

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# 数值型特征
numerical_features = ["年龄", "静息血压", "胆固醇", "最大心率", "运动后的 ST 下降", "主血管数量"]
# 类别型特征
categorical_features = ["胸痛类型", "静息心电图结果", "峰值 ST 段的斜率", "地中海贫血"]
# 二元特征
binary_features = ["性别", "空腹血糖", "运动性心绞痛"]
# 创建列转换器
preprocessor = ColumnTransformer(
    transformers=[
        # 对数值型特征进行标准化
        ("num", StandardScaler(), numerical_features),
        # 对类别型特征进行独热编码，使用 drop="first" 避免多重共线性
        ("cat", OneHotEncoder(drop="first"), categorical_features),
        # 二元特征不进行处理
        ("binary", "passthrough", binary_features),
    ]
)
# 执行特征转换
x_train = preprocessor.fit_transform(x_train) # 计算训练集的统计信息并进行转换
x_test = preprocessor.transform(x_test) # 使用训练集计算的信息对测试集进行转换
```

## 2) 避免多重共线性

`drop="first"` 是独热编码中的一个参数，它的核心目的是避免多重共线性（Multicollinearity）。

多重共线性是指特征之间存在高度线性相关关系的现象。例如特征胸痛类型包含 4 个类别（0、1、2、3），若直接进行独热编码会生成 4 个新列（胸痛类型\_0、胸痛类型\_1、胸痛类型\_2、胸痛类型\_3），此时这 4 列满足

$$\text{胸痛类型}_0 + \text{胸痛类型}_1 + \text{胸痛类型}_2 + \text{胸痛类型}_3 = 1$$

这种完全线性相关关系会导致特征矩阵的列之间存在完美共线性。

当特征矩阵存在多重共线性时，模型参数估计会变得不稳定（矩阵不可逆或接近奇异），导致系数估计值方差增大、模型可解释性下降、过拟合等问题。

在独热编码时设置 `drop="first"`，会删除每个类别特征的第 1 列，从而打破完全共线性。比如特征胸痛类型会生成 3 列（胸痛类型\_1、胸痛类型\_2、胸痛类型\_3），此时

$$\text{胸痛类型}_1 = 0, \text{胸痛类型}_2 = 0, \text{胸痛类型}_3 = 0 \text{ 隐含代表 } \text{胸痛类型}_0 = 1$$

胸痛类型_0	胸痛类型_1	胸痛类型_2	胸痛类型_3
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

→

胸痛类型_1	胸痛类型_2	胸痛类型_3
0	0	0
1	0	0
0	1	0
0	0	1

虽然 KNN 不直接受多重共线性影响（不像线性模型），但使用 `drop="first"` 也能够减少冗余特征，提升计算效率。

### 3.4.5 模型训练与评估

```
from sklearn.neighbors import KNeighborsClassifier

# 使用 K 近邻分类模型, K=3
knn = KNeighborsClassifier(n_neighbors=3)
# 模型训练
knn.fit(x_train, y_train)
# 模型评估, 计算准确率
knn.score(x_test, y_test)
```

### 3.4.6 模型的保存

可以使用 Python 的 `joblib` 库保存训练好的模型：

```
import joblib

joblib.dump(knn, "knn_heart_disease")
```

加载先前保存的模型：

```
# 加载模型
knn_loaded = joblib.load("knn_heart_disease")
```

```
# 预测  
y_pred = knn_loaded.predict(x_test[10:11])  
# 打印真实值与预测值  
print(y_test.iloc[10], y_pred)
```

## 3.5 模型评估与超参数调优

### 3.5.1 网格搜索

网格搜索（Grid Search）是一种系统化的超参数调优方法，通过遍历预定义的超参数组合，找到使模型性能最优的参数配置。通过自动化调参避免手动试错，提高效率。

网格搜索通常嵌套交叉验证，与交叉验证结合以提高调参的可靠性：

- 外层循环：遍历参数网格中的每个参数组合。
- 内层循环：对每个参数组合使用交叉验证评估模型性能。

### 3.5.2 对心脏病预测模型进行超参数调优

对模型训练与评估部分进行修改，使用 `sklearn.model_selection.GridSearchCV` 进行交叉验证和网格搜索：

```
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model_selection import GridSearchCV  
  
knn = KNeighborsClassifier()  
# 网格搜索参数，K 值设置为 1 到 10  
param_grid = {"n_neighbors": list(range(1, 10))}  
# GridSearchCV(estimator=模型, param_grid=网格搜索参数, cv=k 折交叉验证)  
knn = GridSearchCV(estimator=knn, param_grid=param_grid, cv=10)  
  
# 模型训练  
knn.fit(x_train, y_train)  
print(pd.DataFrame(knn.cv_results_)) # 所有交叉验证结果  
print(knn.best_estimator_) # 最佳模型  
print(knn.best_score_) # 最佳得分  
  
# 使用最佳模型进行评估  
knn = knn.best_estimator_  
print(knn.score(x_test, y_test))
```

## 第 4 章 线性回归

### 4.1 线性回归简介

### 4.1.1 什么是线性回归

线性回归（Linear Regression）是一种用于建模两个或多个变量之间线性关系的统计方法。它通过拟合一条直线（或超平面）来描述自变量（输入特征）与因变量（输出目标）之间的关联，并可用于预测或分析变量间的影响关系。

假设因变量 $y$ 与自变量 $x_1, x_2, \dots, x_n$ 之间的关系可以用如下线性方程表示：

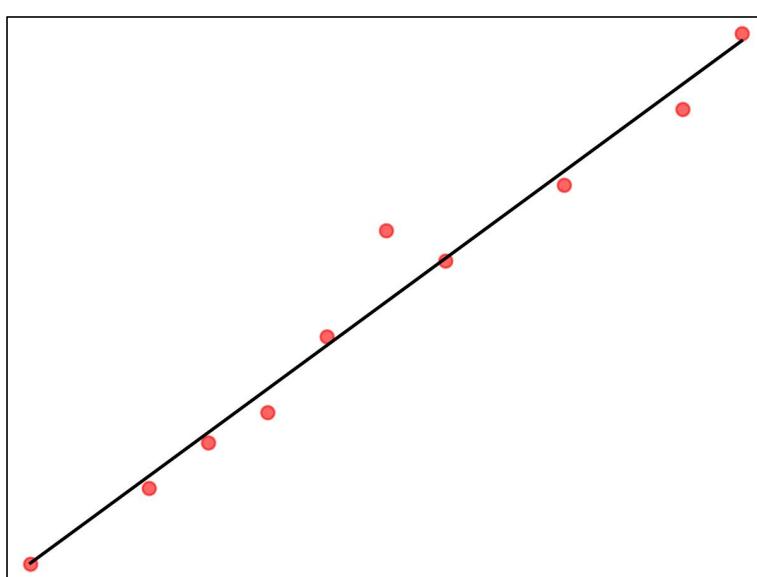
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

- $\beta_0$ : 截距，模型在自变量全为 0 时的基准值
- $\beta_1, \beta_2, \dots, \beta_n$ : 自变量的系数，表示每个自变量对因变量的影响程度

通过估计这些系数，使模型预测值尽可能接近真实值。

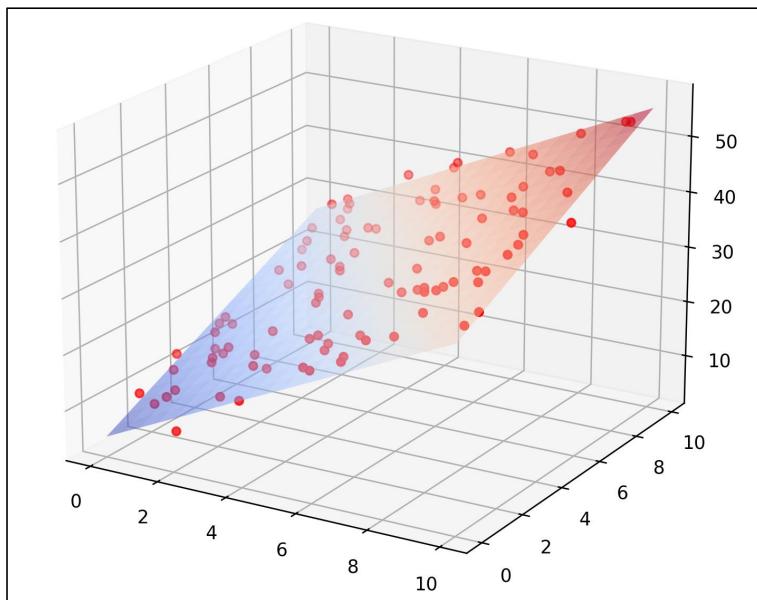
#### 1) 一元线性回归

仅有一个自变量： $y = \beta_0 + \beta_1 x_1$ 。



#### 2) 多元线性回归

包含多个自变量： $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$ 。



#### 4.1.2 线性回归应用场景

- GDP 预测：用历史数据（如投资、消费、出口）建立回归模型，预测 GDP 增长趋势。
- 广告效果评估：量化不同渠道广告投入对销售额的影响，优化预算分配。
- 药物剂量研究：分析药物剂量与患者生理指标（如血压、血糖）之间的关系。
- 产品质量控制：通过生产参数（温度、压力、原料配比）预测产品性能（如强度、耐久）。
- 政策效果评估：分析最低工资政策对就业率的影响，或环保法规对污染排放的抑制作用。
- 气候变化建模：用工业排放量、森林覆盖率等变量预测全球气温变化趋势。

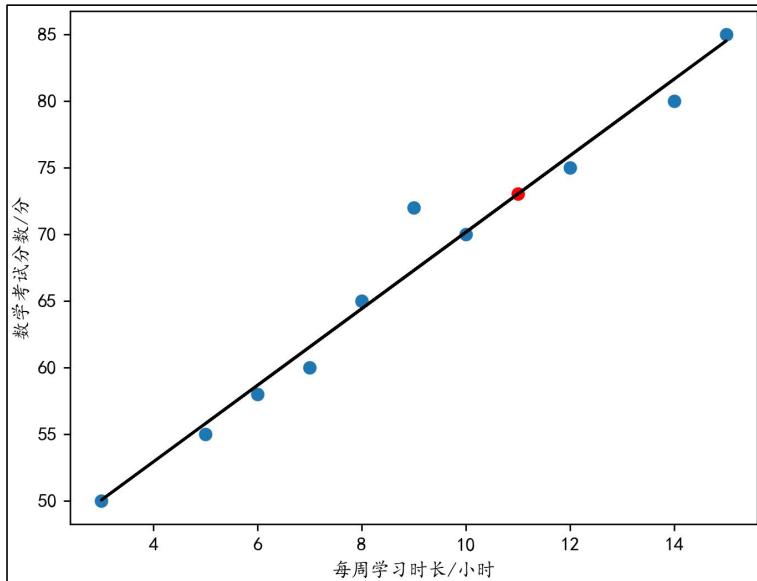
#### 4.1.3 API 使用

例如，某中学教师想研究学生每周学习时间（小时）与数学考试成绩（0-100 分）之间的关系，并预测学生成绩。

```
from sklearn.linear_model import LinearRegression

# 自变量，每周学习时长
X = [[5], [8], [10], [12], [15], [3], [7], [9], [14], [6]]
# 因变量，数学考试成绩
y = [55, 65, 70, 75, 85, 50, 60, 72, 80, 58]
# 实例化线性回归模型
model = LinearRegression()
```

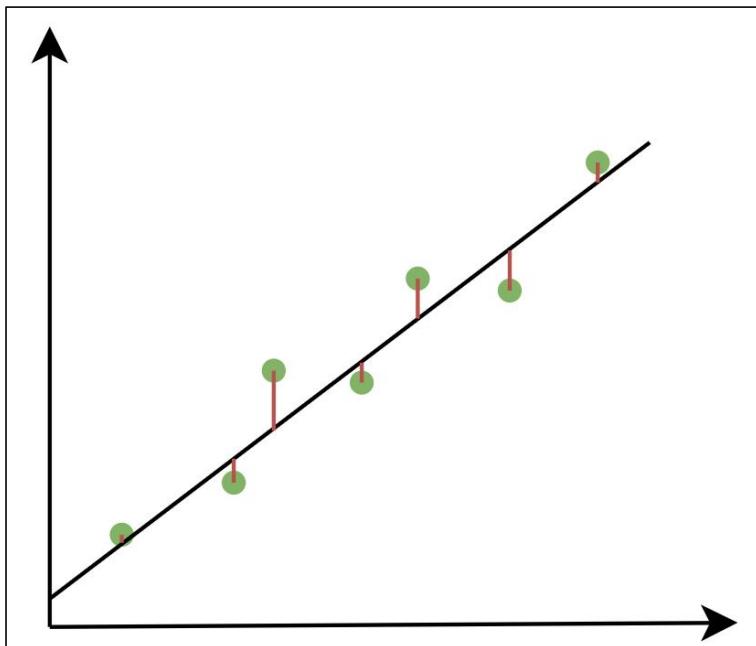
```
# 模型训练
model.fit(X, y)
# 系数, 每周每学习 1 小时, 成绩会增加多少分
print(model.coef_)
# 截距
print(model.intercept_)
# 预测,每周学习 11 小时, 成绩可能是多少分
print(model.predict([[11]]))
```



## 4.2 线性回归求解

### 4.2.1 损失函数

模型的预测值与真实值之间存在误差。



损失函数用于衡量模型预测值与真实值之间的误差，并通过最小化损失函数来优化模型参数。当损失函数最小时，得到的自变量系数就是最优解。

### 1) 均方误差 (MSE)

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

- $n$ : 样本个数。
- $y_i$ : 第  $i$  个样本的真实值。
- $f(x_i)$ : 第  $i$  个样本的预测值。

均方误差是回归任务中最常用的损失函数。均方误差对应了欧氏距离。基于均方误差最小化来进行模型求解的方法称为“最小二乘法”。在线性回归中，最小二乘法就是试图找到一条直线（或超平面），使所有样本到直线（或超平面）上的欧氏距离之和最小。

#### (1) 均方误差的特点

- 均方误差对大误差比较敏感，因为平方项会放大较大的误差（如误差  $2 \rightarrow 4$ ，误差  $10 \rightarrow 100$ ）。
- 均方误差是凸函数，存在全局的唯一最小值，平方项又使损失函数处处可导，便于求解最优参数。
- 最小二乘法（最小化 MSE）的解析解可通过矩阵运算直接求出（如  $\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ ）。

➤ 若误差服从正态分布，则均方误差对应极大似然估计，是最优的损失函数。

(2) 误差服从正态分布时，均方误差与极大似然估计的关系

假设因变量 $y$ 与自变量 $x$ 的关系为 $y_i = \beta^T x_i + \epsilon_i$ ,  $\epsilon_i$ 为误差项。

当误差独立且具有相同分布，并且都服从正态分布时：

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right)$$

将 $y_i$ 与 $x_i$ 代入：

$$p(y_i|x_i; \beta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \beta^T x_i)^2}{2\sigma^2}\right)$$

似然函数：

$$L(\beta) = \prod_{i=1}^n p(y_i|x_i; \beta) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \beta^T x_i)^2}{2\sigma^2}\right)$$

对数似然函数：

$$\begin{aligned} \ln L(\beta) &= \ln \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \beta^T x_i)^2}{2\sigma^2}\right) \\ &= -n \ln \sqrt{2\pi\sigma^2} - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \beta^T x_i)^2 \end{aligned}$$

为使似然函数最大，需求解 $\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \beta^T x_i)^2 = \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - f(x_i))^2$ 的最小值，发现

其与均方误差直接相关。即最大化对数似然函数等价于最小化均方误差。

## 2) 平均绝对误差 (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |f(x_i) - y_i|$$

平均绝对误差受异常值影响较小，但对小误差的惩罚较弱。适用与数据中存在显著异常值（如金融风险预测）的场景。

### 4.2.2 一元线性回归解析解

对于一元线性回归，有

$$f(x_i) = \beta_0 + \beta_1 x_i$$

$$MSE = \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)^2$$

对 $\beta_0$ 求偏导：

$$\frac{\partial MSE}{\partial \beta_0} = \frac{\partial \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)^2}{\partial \beta_0}$$

$$= \frac{2}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)$$

令偏导等于 0:

$$\frac{2}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i) = 0$$

$$\beta_0 = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_1 x_i)$$

$$= \bar{y} - \beta_1 \bar{x}$$

对  $\beta_1$  求偏导:

$$\frac{\partial MSE}{\partial \beta_1} = \frac{\partial \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)^2}{\partial \beta_1}$$

$$= \frac{2}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i) x_i$$

将  $\beta_0 = \bar{y} - \beta_1 \bar{x}$  代入:

$$\begin{aligned} \frac{\partial MSE}{\partial \beta_1} &= \frac{2}{n} \sum_{i=1}^n (x_i(\bar{y} - \beta_1 \bar{x}) + \beta_1 x_i^2 - y_i x_i) \\ &= \frac{2}{n} \sum_{i=1}^n (\beta_1(x_i^2 - x_i \bar{x}) + x_i(\bar{y} - y_i)) \end{aligned}$$

令偏导等于 0:

$$\frac{2}{n} \sum_{i=1}^n (\beta_1(x_i^2 - x_i \bar{x}) + x_i(\bar{y} - y_i)) = 0$$

$$\beta_1 \sum_{i=1}^n (x_i^2 - x_i \bar{x}) = \sum_{i=1}^n x_i(y_i - \bar{y})$$

解得:

$$\begin{aligned}
 \beta_1 &= \frac{\sum_{i=1}^n x_i(y_i - \bar{y})}{\sum_{i=1}^n (x_i^2 - x_i \bar{x})} \\
 &= \frac{\sum_{i=1}^n x_i(y_i - \bar{y})}{\sum_{i=1}^n x_i(x_i - \bar{x})} \\
 &= \frac{\sum_{i=1}^n (x_i - \bar{x} + \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x} + \bar{x})(x_i - \bar{x})} \\
 &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) + n\bar{x}\sum_{i=1}^n (y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x}) + n\bar{x}\sum_{i=1}^n (x_i - \bar{x})}
 \end{aligned}$$

其中  $\sum_{i=1}^n (y_i - \bar{y}) = \sum_{i=1}^n y_i - n\bar{y} = 0$ ,  $\sum_{i=1}^n (x_i - \bar{x}) = \sum_{i=1}^n x_i - n\bar{x} = 0$

所以

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

以前述学生每周学习时间（小时）与数学考试成绩（0-100 分）之间的关系为例：

```
# 自变量，每周学习时长
X = [[5], [8], [10], [12], [15], [3], [7], [9], [14], [6]]
# 因变量，数学考试成绩
y = [55, 65, 70, 75, 85, 50, 60, 72, 80, 58]
```

$$\begin{aligned}
 \beta_1 &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \\
 &= \frac{(5 - 8.9) \times (55 - 67) + (8 - 8.9) \times (65 - 67) + \dots + (6 - 8.9) \times (58 - 67)}{(5 - 8.9)^2 + (8 - 8.9)^2 + \dots + (6 - 8.9)^2} \\
 &= 2.8707
 \end{aligned}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} = 67 - 2.8707 \times 8.9 = 41.4507$$

$$\text{解得 } \beta = \frac{41.4507}{2.8707}$$

### 4.2.3 正规方程法

#### 1) 正规方程法介绍

正规方程法 (Normal Equation) 是一种用于求解线性回归的解析解的方法。它基于最小二乘法，通过求解矩阵方程来直接获得参数值。

将损失函数转换为矩阵形式：

$$\begin{aligned}
MSE &= \frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^n ((\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_m x_{im}) - y_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^n (\boldsymbol{\beta}^T \mathbf{x}_i - y_i)^2 \\
&= \frac{1}{n} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2 \\
&= \frac{1}{n} (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})
\end{aligned}$$

➤  $\mathbf{X}$ :  $n \times (m + 1)$  的矩阵，包含一个全 1 的列

$$\mathbf{X} = \begin{matrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{matrix}$$

➤  $\boldsymbol{\beta}$ :  $(m + 1) \times 1$  的参数向量（包含截距项  $\beta_0$ ）

$$\boldsymbol{\beta} = \begin{matrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{matrix}$$

➤  $\mathbf{y}$ :  $n \times 1$  的因变量向量

对  $\boldsymbol{\beta}$  求偏导

$$\begin{aligned}
\frac{\partial MSE}{\partial \boldsymbol{\beta}} &= \frac{\partial \left( \frac{1}{n} (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) \right)}{\partial \boldsymbol{\beta}} \\
&= \frac{2}{n} \mathbf{X}^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})
\end{aligned}$$

当  $\mathbf{X}^T \mathbf{X}$  为满秩矩阵或正定矩阵时，令偏导等于 0

$$\frac{2}{n} \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} - \frac{2}{n} \mathbf{X}^T \mathbf{y} = \mathbf{0}$$

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}$$

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

正规方程法适用于特征数量较少的情况。当特征数量较大时，计算逆矩阵的复杂度会显著增加，此时梯度下降法更为适用。

## 2) API 使用

```
# fit_intercept: 是否计算偏置
model = sklearn.linear_model.LinearRegression(fit_intercept=True)
model.fit([[0, 3], [1, 2], [2, 1]], [0, 1, 2])
# coef_: 系数
print(model.coef_)
# intercept_: 偏置
print(model.intercept_)
```

#### 4.2.4 梯度下降法

##### 1) 更新公式

梯度下降法 (Gradient Descent) 是一种用于最小化目标函数的迭代优化算法。核心是沿着目标函数 (如损失函数) 的负梯度方向逐步调整参数，从而逼近函数的最小值。梯度方向指示了函数增长最快的方向，因此负梯度方向是函数下降最快的方向。

假设目标函数为  $J(\boldsymbol{\beta})$ ，其中  $\boldsymbol{\beta}$  是模型参数。梯度下降的更新公式为：

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \alpha \cdot \nabla J(\boldsymbol{\beta}_t)$$

$\alpha$ : 学习率 (Learning Rate)，用于控制步长

$\nabla J(\boldsymbol{\beta}_t)$ : 目标函数在  $\boldsymbol{\beta}_t$  处的梯度 (偏导数组成的向量)

##### 2) 计算示例

以前述学生每周学习时间 (小时) 与数学考试成绩 (0-100 分) 之间的关系为例。

自变量 (学习时长)  $\mathbf{x} = [5 \ 8 \ 10 \ 12 \ 15 \ 3 \ 7 \ 9 \ 14 \ 6]^T$ 。

因变量 (数学考试成绩)  $\mathbf{y} = [55 \ 65 \ 70 \ 75 \ 85 \ 50 \ 60 \ 72 \ 80 \ 58]^T$ 。

求  $\boldsymbol{\beta} = [\beta_0 \ \beta_1]^T$  使得损失函数  $J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2$  最小，为此需要求出损失函数相对  $\boldsymbol{\beta}$  的梯度。

$\mathbf{x}$  添加一列 1，转换为  $\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 5 & 8 & 10 & 12 & 15 & 3 & 7 & 9 & 14 & 6 \end{bmatrix}^T$ 。则损失函数  $J(\boldsymbol{\beta}) = \frac{1}{n} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ 。

求得损失函数的梯度

$$\nabla J(\boldsymbol{\beta}) = \frac{2}{n} \mathbf{X}^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})$$

使用梯度下降法求解  $\boldsymbol{\beta} = [\beta_0 \ \beta_1]^T$ 。

- $\boldsymbol{\beta}$  初始值取  $[1 \ 1]^T$
- 学习率  $\alpha$  取 0.01

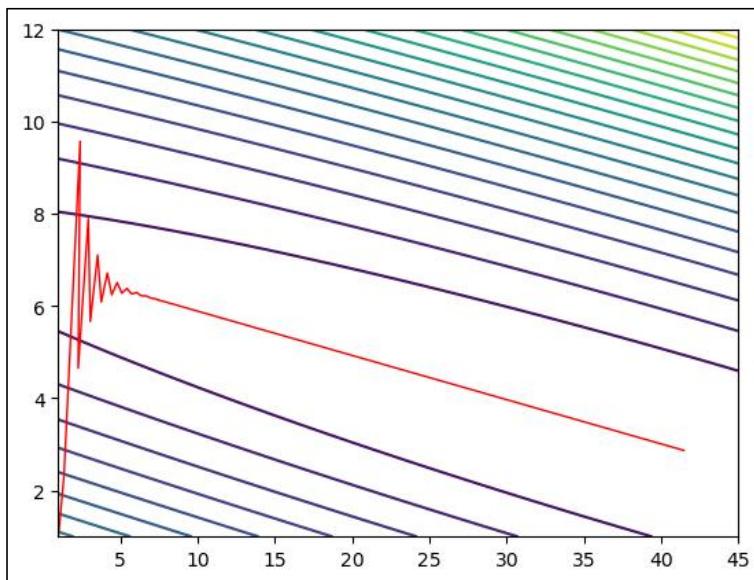
$$\nabla J(\beta) = \frac{2}{n} X^T (X\beta - y)$$

$$\beta_1 = \frac{1}{1}, J(\beta_1) = 3311.3, \nabla J(\beta_1) = \frac{-114.2}{-1067.6}, \beta_2 = \beta_1 - \alpha \nabla J(\beta_1) = \frac{2.142}{11.676}$$

$$\beta_2 = \frac{2.142}{11.676}, J(\beta_2) = 2589.9686, \nabla J(\beta_2) = \frac{78.1168}{936.3284}, \beta_3 = \beta_2 - \alpha \nabla J(\beta_2) = \frac{1.3608}{2.3127}$$

⋮

$$\beta_n = \frac{41.45069394}{2.87070855}, J(\beta_n) = 2.98115413, \nabla J(\beta_n) = \frac{-2.33768560 \times 10^{-12}}{1.17950094 \times 10^{-13}}$$



示例代码：

```
import numpy as np

def J(beta):
    """目标函数"""
    return np.sum((X @ beta - y) ** 2, axis=0).reshape(-1, 1) / n

def gradient(beta):
    """梯度"""
    return X.T @ (X @ beta - y) / n * 2

X = np.array([[5], [8], [10], [12], [15], [3], [7], [9], [14], [6]]) # 自变量, 每周学习时长
y = np.array([[55], [65], [70], [75], [85], [50], [60], [72], [80], [58]]) # 因变量, 数学考试成绩
beta = np.array([[1], [1]]) # 初始化参数
n = X.shape[0] # 样本数
```

```
X = np.hstack([np.ones((n, 1)), X]) # X 添加一列 1，与偏置项相乘
alpha = 1e-2 # 学习率
epoch = 0 # 迭代次数
while (j := J(beta)) > 1e-10 and (epoch := epoch + 1) <= 10000:
    grad = gradient(beta) # 求解梯度
    if epoch % 1000 == 0:
        print(f"beta={beta.reshape(-1)}\tJ={j.reshape(-1)}")
    beta = beta - alpha * grad # 更新参数
```

### 3) 学习率的选择

- 学习率过大：可能导致跳过最优解，甚至发散。
- 学习率过小：收敛速度慢，易陷入局部极小。
- 自适应学习率：高级优化器（如 Adam、Adagrad）动态调整学习率以提升性能。

### 4) 梯度下降法常见问题

#### (1) 特征缩放

通常需要提前对特征进行缩放（如标准化或归一化），以加快收敛速度。

#### (2) 局部极小值、鞍点问题

可能陷入局部极小值（非全局最优解），或遇到鞍点（梯度为零但非极值点）。

解决方案：使用动量（Momentum）、自适应优化器（如 Adam）或二阶方法（如牛顿法）。

### 5) API 使用

```
model = sklearn.linear_model.SGDRegressor(
    loss="squared_error", # 损失函数，默认为均方误差
    fit_intercept=True, # 是否计算偏置
    learning_rate="constant", # 学习率策略
    eta0=0.1, # 初始学习率
    max_iter=1000, # 最大迭代次数
    tol=1e-8, # 损失值小于 tol 时停止迭代
)
model.fit([[0, 3], [1, 2], [2, 1]], [0, 1, 2])
# coef_: 系数
print(model.coef_)
# intercept_: 偏置
print(model.intercept_)
```

## 4.3 案例：广告投放效果预测

### 4.3.1 数据集说明

Advertising 数据集: <https://www.kaggle.com/datasets/tawfikelmetwally/advertising-dataset>.

- ID: 序号
- TV: 电视广告投放金额, 单位千元
- Radio: 广播广告投放金额, 单位千元
- Newspaper: 报纸广告投放金额, 单位千元
- Sales: 销售额, 单位百万元

### 4.3.2 使用线性回归预测广告投放效果

```
import pandas as pd
from sklearn.preprocessing import StandardScaler # 标准化
from sklearn.model_selection import train_test_split # 划分数据集
from sklearn.linear_model import LinearRegression, SGDRegressor # 线性回归-正规方程, 线性回归-随机梯度下降
from sklearn.metrics import mean_squared_error # 均方误差

# 加载数据集
advertising = pd.read_csv("data/advertising.csv")
advertising.drop(advertising.columns[0], axis=1, inplace=True)
advertising.dropna(inplace=True)
advertising.info()
print(advertising.head())

# 划分训练集与测试集
X = advertising.drop("Sales", axis=1)
y = advertising["Sales"]
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

# 标准化
preprocessor = StandardScaler()
x_train = preprocessor.fit_transform(x_train) # 计算训练集的均值和标准差,
并标准化训练集
x_test = preprocessor.transform(x_test) # 使用训练集的均值和标准差对测试集
标准化

# 使用正规方程法拟合线性回归模型
normal_equation = LinearRegression()
normal_equation.fit(x_train, y_train)
print("正规方程法解得模型系数:", normal_equation.coef_)
print("正规方程法解得模型偏置:", normal_equation.intercept_)
```

```
# 使用随机梯度下降法拟合线性回归模型
gradient_descent = SGDRegressor()
gradient_descent.fit(x_train, y_train)
print("随机梯度下降法解得模型系数:", gradient_descent.coef_)
print("随机梯度下降法解得模型系数:", gradient_descent.intercept_)

# 使用均方误差评估模型
print("正规方程法均方误差:", mean_squared_error(y_test,
normal_equation.predict(x_test)))
print("随机梯度下降法均方误差:", mean_squared_error(y_test,
gradient_descent.predict(x_test)))
```

## 第 5 章 逻辑回归

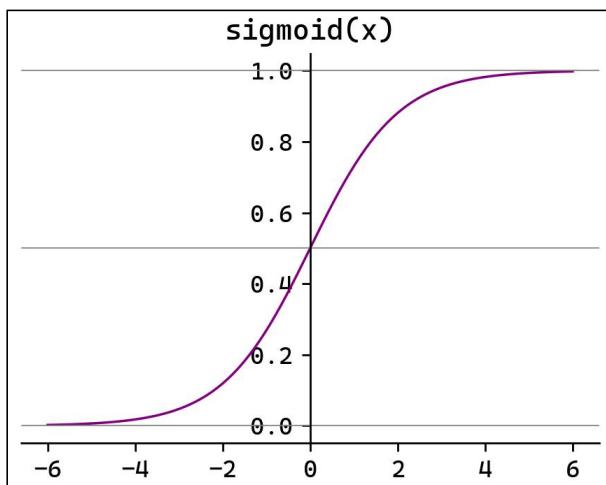
### 5.1 逻辑回归简介

#### 5.1.1 什么是逻辑回归

逻辑回归 (Logistic Regression) 是一种用于解决分类问题的统计方法，尤其适用于二分类问题。尽管名称中有“回归”，但它主要用于分类任务。

逻辑回归通过将线性回归的输出映射到[0,1]区间，来表示某个类别的概率。

常用的映射函数是 sigmoid 函数:  $f(x) = \frac{1}{1+e^{-x}}$ , 其导数  $f'(x) = f(x)(1 - f(x))$ 。



逻辑回归结果可表示为:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

其中  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$  为线性回归输出结果,  $P(y = 1|x)$  表示输出为 1 类的概率。根据逻辑回归结果和阈值来确认最终预测结果，若逻辑回归结果大于阈值则输出为 1 类，反之输出为 0 类。

$$\begin{array}{c}
 \text{线性回归输出结果} \\
 \left[ \begin{array}{ccc} 0.5 & 0 & 0.7 \\ 0.5 & 0.5 & 0.9 \\ 0.1 & 1 & 0.6 \\ 0.6 & 0.1 & 0 \end{array} \right] \left[ \begin{array}{c} \boldsymbol{\beta} \\ -1 \\ 2 \\ 0.5 \end{array} \right] = \left[ \begin{array}{c} -0.15 \\ 0.95 \\ 2.2 \\ -0.4 \end{array} \right]
 \end{array}
 \xrightarrow{\text{sigmoid}}
 \begin{array}{c}
 \text{逻辑回归结果} \\
 \left[ \begin{array}{c} 0.46257015 \\ 0.72111518 \\ 0.90024951 \\ 0.40131234 \end{array} \right]
 \end{array}
 \xrightarrow{\text{与阈值 0.5 比较}}
 \begin{array}{c}
 \text{预测结果} \\
 \left[ \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \right]
 \end{array}$$

### 5.1.2 逻辑回归应用场景

- 信用评分：预测客户是否会违约（违约/不违约）。
- 欺诈检测：预测某笔交易是否是欺诈行为。
- 垃圾邮件检测：预测一封邮件是否是垃圾邮件（垃圾邮件/非垃圾邮件）。
- 广告点击预测：预测用户是否会点击某个广告（点击/不点击）。
- 图像分类：将图像分类为不同的类别（如猫、狗、鸟等）。
- 情感分析：将文本分类为正面、负面或中性情感。
- 产品质量分类：预测产品是否合格。
- 医学诊断：预测患者是否患有某种疾病（患病/未患病）。
- 蛋白质功能预测：基于蛋白质序列和结构特征预测其功能类别。

### 5.1.3 逻辑回归损失函数

逻辑回归的损失函数通常使用对数损失（Log Loss），也称为二元交叉熵损失（Binary Cross-Entropy Loss），用于衡量模型输出的概率分布与真实标签之间的差距。逻辑回归的损失函数来源于最大似然估计（MLE）。

$P(Y|X; \boldsymbol{\beta})$  表示给定输入特征  $\mathbf{x}$  和模型参数  $\boldsymbol{\beta}$  时，因变量  $y$  发生的概率：

$$\begin{aligned}
 P(y=1|\mathbf{x}; \boldsymbol{\beta}) &= \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x})}} \\
 P(y=0|\mathbf{x}; \boldsymbol{\beta}) &= 1 - P(y=1|\mathbf{x}; \boldsymbol{\beta}) = 1 - \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x})}}
 \end{aligned}
 \right\} \xrightarrow{\text{整合}} P(y|\mathbf{x}; \boldsymbol{\beta})$$

$$\begin{aligned}
 P(y|\mathbf{x}; \boldsymbol{\beta}) &= P(y=1|\mathbf{x}; \boldsymbol{\beta})^y (1 - P(y=1|\mathbf{x}; \boldsymbol{\beta}))^{1-y} \\
 &= \left( \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x})}} \right)^y \left( 1 - \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x})}} \right)^{1-y}
 \end{aligned}$$

似然函数  $L(\boldsymbol{\beta})$  表示已知  $y$  的结果，此时模型参数为  $\boldsymbol{\beta}$  的概率：

对于 1 个样本:  $L(\boldsymbol{\beta}) = P(y|x; \boldsymbol{\beta}) = P(y=1|x; \boldsymbol{\beta})^y (1 - P(y=1|x; \boldsymbol{\beta}))^{1-y}$

对于 n 个样本:  $L(\boldsymbol{\beta}) = \prod_{i=1}^n P(y_i|x_i; \boldsymbol{\beta}) = \prod_{i=1}^n P(y_i=1|x_i; \boldsymbol{\beta})^{y_i} (1 - P(y_i=1|x_i; \boldsymbol{\beta}))^{1-y_i}$

取对数似然:  $\log L(\boldsymbol{\beta}) = \sum_{i=1}^n \left( y_i \log P(y_i=1|x_i; \boldsymbol{\beta}) + (1-y_i) \log (1 - P(y_i=1|x_i; \boldsymbol{\beta})) \right)$

拟合的过程就是求解似然函数的最大值, 为了方便优化, 令损失函数

$$\begin{aligned} Loss &= -\frac{1}{n} \log L(\boldsymbol{\beta}) \\ &= -\frac{1}{n} \sum_{i=1}^n \left( y_i \log P(y_i=1|x_i; \boldsymbol{\beta}) + (1-y_i) \log (1 - P(y_i=1|x_i; \boldsymbol{\beta})) \right) \end{aligned}$$

来求解损失函数的最小值。

#### 5.1.4 损失函数的梯度（了解）

$$\begin{aligned} Loss &= -\frac{1}{n} \sum_{i=1}^n \left( y_i \log P(y_i=1|x_i; \boldsymbol{\beta}) + (1-y_i) \log (1 - P(y_i=1|x_i; \boldsymbol{\beta})) \right) \\ &= -\frac{1}{n} (\mathbf{y}^T \log \mathbf{p} + (1-\mathbf{y})^T \log (1 - \mathbf{p})) \end{aligned}$$

其中:

$$\begin{aligned} \mathbf{y} &= [y_1 \quad y_2 \quad \dots \quad y_n]^T \\ \mathbf{p} &= [P(y_1=1|x_1; \boldsymbol{\beta}) \quad P(y_2=1|x_2; \boldsymbol{\beta}) \quad \dots \quad P(y_n=1|x_n; \boldsymbol{\beta})]^T \\ &= \left[ \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x}_1)}} \quad \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x}_2)}} \quad \dots \quad \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x}_n)}} \right]^T \\ &= \frac{1}{1 + e^{-(\mathbf{x}\boldsymbol{\beta})}} \end{aligned}$$

则梯度

$$\begin{aligned}
 \nabla Loss &= \frac{\partial Loss}{\partial \mathbf{p}} \cdot \frac{\partial \mathbf{p}}{\partial \boldsymbol{\beta}} \\
 &= \frac{\partial \left( -\frac{1}{n} (\mathbf{y}^T \log \mathbf{p} + (1-\mathbf{y})^T \log(1-\mathbf{p})) \right)}{\partial \mathbf{p}} \cdot \frac{\partial \left( \frac{1}{1+e^{-(\mathbf{X}\boldsymbol{\beta})}} \right)}{\partial \boldsymbol{\beta}} \\
 &= -\frac{1}{n} \left( \frac{\mathbf{y}}{\mathbf{p}} - \frac{1-\mathbf{y}}{1-\mathbf{p}} \right) \cdot (\mathbf{p} \odot (1-\mathbf{p}) \cdot \mathbf{X}) \\
 &= \frac{1}{n} \mathbf{X}^T (\mathbf{p} - \mathbf{y}) \\
 &= \frac{1}{n} \mathbf{X}^T \left( \frac{1}{1+e^{-(\mathbf{X}\boldsymbol{\beta})}} - \mathbf{y} \right)
 \end{aligned}$$

其中  $\odot$  表示 Hadamard 乘积（逐项乘积）。

## 5.1.5 API 使用

### 1) LogisticRegression 参数说明

scikit-learn 中的 LogisticRegression 类用于实现逻辑回归。它支持二分类和多分类任务，并提供了多种优化算法和正则化选项。

```
# solver: 优化算法
# lbgfs: 拟牛顿法（默认），仅支持 L2 正则化
# newton-cg: 牛顿法，仅支持 L2 正则化
# liblinear: 坐标下降法，适用于小数据集，支持 L1 和 L2 正则化
# sag: 随机平均梯度下降，适用于大规模数据集，仅支持 L2 正则化
# saga: 改进的随机梯度下降，适用于大规模数据，支持 L1、L2 和 ElasticNet 正则化
# penalty: 正则化类型，可选 l1、l2 和 elasticnet
# C: 正则化强度，C 越小，正则化强度越大
# class_weight: 类别权重，balanced 表示自动平衡类别权重，让模型在训练时更关注少数类，从而减少类别不平衡带来的偏差
model = sklearn.linear_model.LogisticRegression(solver="lbgfs",
penalty="l2", C=1, class_weight="balanced")
```

### 2) 案例：心脏病预测

Heart Disease 数据集 <https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>。

- 年龄：连续值
- 性别：0-女，1-男
- 胸痛类型：1-典型心绞痛，2-非典型心绞痛，3-非心绞痛，4-无症状
- 静息血压：连续值，单位 mmHg
- 胆固醇：连续值，单位 mg/dl
- 空腹血糖：1-大于 120mg/dl，2-小于等于 120mg/dl

- 静息心电图结果: 0-正常, 1-ST-T 异常, 2-可能左心室肥大
- 最大心率: 连续值
- 运动性心绞痛: 1-有, 2-无
- 运动后的 ST 下降: 连续值
- 峰值 ST 段的斜率: 1-向上, 2-水平, 3-向下
- 主血管数量: 0 到 3
- 地中海贫血: 一种先天性贫血, 3-正常, 6-固定缺陷, 7-可逆缺陷
- 是否患有心脏病: 标签, 0-否, 1-是

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression

# 加载数据集
heart_disease = pd.read_csv("data/heart_disease.csv")
heart_disease.dropna()

# 划分为训练集与测试集
X = heart_disease.drop("是否患有心脏病", axis=1) # 特征
y = heart_disease["是否患有心脏病"] # 标签
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=100)

# 特征工程
# 数值型特征
numerical_features = ["年龄", "静息血压", "胆固醇", "最大心率", "运动后的 ST
下降", "主血管数量"]
# 类别型特征
categorical_features = ["胸痛类型", "静息心电图结果", "峰值 ST 段的斜率", "地中
海贫血"]
# 二元特征
binary_features = ["性别", "空腹血糖", "运动性心绞痛"]
# 创建列转换器
preprocessor = ColumnTransformer(
    transformers=[
        # 对数值型特征进行标准化
        ("num", StandardScaler(), numerical_features),
        # 对类别型特征进行独热编码, 使用 drop="first"避免多重共线性
        ("cat", OneHotEncoder(drop="first"), categorical_features),
```

```

# 二元特征不进行处理
("binary", "passthrough", binary_features),
]

)
# 执行特征转换
x_train = preprocessor.fit_transform(x_train) # 计算训练集的统计信息并进行
# 转换
x_test = preprocessor.transform(x_test) # 使用训练集计算的信息对测试集进行
# 转换

# 模型训练
model = LogisticRegression()
model.fit(x_train, y_train)

# 模型评估，计算准确率
model.score(x_test, y_test)

```

## 5.2 多分类任务（了解）

逻辑回归通常用于二分类问题，但可以通过一对多（One-vs-Rest, OvR）以及 Softmax 回归（Multinomial Logistic Regression, 多项逻辑回归）来扩展到多分类任务。

### 5.2.1 一对多（OVR）

#### 1) 实现方式

若有 C 个类别，则训练 C 个二分类逻辑回归分类器。

每个分类器将一个类别作为正例、所有其他类别作为反例。

预测时，计算 C 个分类器的输出概率，选取概率最高的类别。

对于类别 c：

$$P(y = c | \mathbf{x}; \boldsymbol{\beta}) = \frac{1}{1 + e^{-(\boldsymbol{\beta}^T \mathbf{x})}}$$

$$\text{Loss} = -\frac{1}{n} \sum_{i=1}^n \left( y_i \log P(y_i = c | \mathbf{x}_i; \boldsymbol{\beta}) + (1 - y_i) \log (1 - P(y_i = c | \mathbf{x}_i; \boldsymbol{\beta})) \right)$$

#### 2) 优缺点

优点：简单易于实现，适用于类别数量较少的情况。

缺点：每个类别训练 1 个分类器，当类别数量较多时，训练时间较长。

#### 3) API

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression(multi_class="ovr")

# 或

from sklearn.multiclass import OneVsRestClassifier

model = OneVsRestClassifier(LogisticRegression())
```

## 5.2.2 Softmax 回归（多项逻辑回归）

### 1) 实现方式

直接扩展逻辑回归到多分类问题，使用 Softmax 函数将模型输出转化为概率分布。

只需训练 1 个逻辑回归模型。

预测时用 1 个模型计算所有类别的概率，选择最大值。

若有 C 个类别，模型将输出 C 个分数。

对于类别  $c$ ,  $P(y = c | \mathbf{x}) = \frac{e^{\beta_c^T \mathbf{x}}}{\sum_{j=1}^C e^{\beta_j^T \mathbf{x}}}$ 。

损失函数  $Loss = -\frac{1}{n} \sum_{i=1}^n \sum_{c=1}^C I(y_i = c) \log P(y_i = c | \mathbf{x}_i)$

其中  $I(y_i = c)$  为示性函数，当  $y_i = c$  时值为 1，反之值为 0。

### 2) 优缺点

优点：只训练 1 个模型，计算高效，分类一致性更好。

缺点：计算 Softmax 需要对所有类别求指数，计算量较高。

### 3) API

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(multi_class="multinomial")

# 对于多分类问题，LogisticRegression 会自动使用 multinomial，因此 multi_class
# 参数可省略

model = LogisticRegression()
```

## 5.3 案例：手写数字识别

### 5.3.1 数据集说明

Digit Recognizer 数据集：<https://www.kaggle.com/competitions/digit-recognizer>。

文件 train.csv 中包含手绘数字（从 0 到 9）的灰度图像，每张图像为  $28 \times 28$  像素，共 784 像素。每个像素有一个 0 到 255 的值表示该像素的亮度。

文件第 1 列为标签，之后 784 列分别为 784 个像素的亮度值。

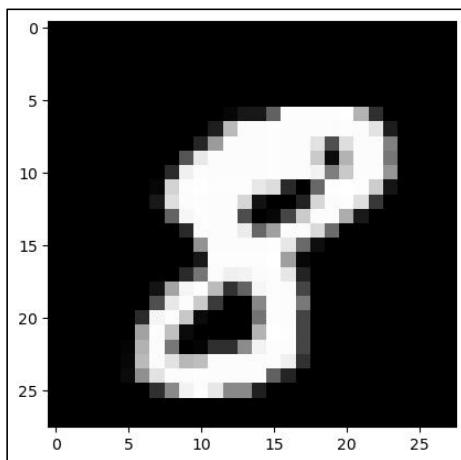
第  $x \times 28 + y$  个像素表示图像第  $x$  行第  $y$  列的像素。

000	001	002	003	...	026	027
028	029	030	031	...	054	055
056	057	058	059	...	082	083
				...		
728	729	730	731	...	754	755
756	757	758	759	...	782	783

### 5.3.2 逻辑回归实现手写数字识别

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression

# 加载数据集
digit = pd.read_csv("data/train.csv")
plt.imshow(digit.iloc[10, 1:].values.reshape(28, 28), cmap="gray")
plt.show()
```



```
# 划分训练集和测试集
X = digit.drop("label", axis=1) # 特征
y = digit["label"] # 标签
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=100)
```

```
# 归一化
preprocessor = MinMaxScaler()
x_train = preprocessor.fit_transform(x_train)
x_test = preprocessor.transform(x_test)

# 模型训练
model = LogisticRegression(max_iter=500)
model.fit(x_train, y_train)

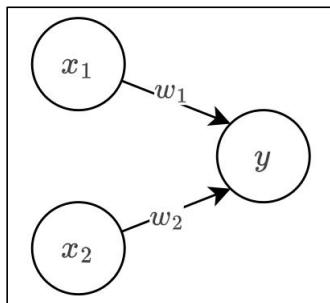
# 模型评估
model.score(x_test, y_test)

# 预测
plt.imshow(digit.iloc[123, 1:].values.reshape(28, 28), cmap="gray")
plt.show()
print(model.predict(digit.iloc[123, 1:].values.reshape(1, -1)))
```

## 第 6 章 感知机

### 6.1 感知机的概念

感知机（Perceptron）是二分类模型，接收多个信号，输出一个信号。感知机的信号只有 0、1 两种取值。下图是一个接收两个输入信号的感知机的例子：



$x_1, x_2$  是输入信号， $y$  是输出信号， $w_1, w_2$  是权重， $\circ$  称为神经元或节点。输入信号被送往神经元时，会分别乘以固定的权重。神经元会计算传送过来的信号的总和，只有当这个总和超过某个界限值时才会输出 1，也称之为神经元被激活。这里将界限值称之为阈值  $\theta$ 。

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

感知机的多个输入信号都有各自的权重，这些权重发挥着控制各个信号的重要性的作用，权重越大，对应信号的重要性越高。

### 6.2 简单逻辑电路

## 6.2.1 与门

尝试使用感知机解决简单的问题，如实现逻辑电路中的与门（AND gate）。与门是具有两个输入和一个输出的门电路，仅在两个输入均为 1 时输出 1，其他时候输出 0。这种输入信号和输出信号对应表称为真值表。

$x_1$	$x_2$	$y$
0	0	0
1	0	0
0	1	0
1	1	1

使用感知机表示与门需要确定能满足上述真值表的  $w_1, w_2, \theta$  的值，满足条件的参数有无数个，如  $(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$  或  $(1.0, 1.0, 1.0)$ 。

## 6.2.2 与非门

再来考虑下与非门（NAND gate），与非门颠倒了与门的输出，仅当  $x_1, x_2$  同时为 1 时输出 0，其他时候输出 1。

$x_1$	$x_2$	$y$
0	0	1
1	0	1
0	1	1
1	1	0

要表示与非门，满足条件的参数也有无数个，如  $(w_1, w_2, \theta) = (-0.5, -0.5, -0.7)$ 。实际上，只要把实现与门的参数值的符号取反即可。

## 6.2.3 或门

再来看一下或门（OR gate）。或门只要有一个输入信号为 1，输出就为 1。

$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	1

$(w_1, w_2, \theta) = (0.5, 0.5, 0)$ 。

## 6.3 感知机的实现

### 6.3.1 简单实现

先来实现与门，定义一函数 AND 接受  $x_1$  和  $x_2$ ，在函数内部初始化参数  $w_1, w_2, \theta$ 。当输入的加权总和超过阈值时返回 1，否则返回 0。

```
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    res = x1 * w1 + x2 * w2
    if res <= theta:
        return 0
    elif res > theta:
        return 1

print(AND(0, 0)) # 0
print(AND(1, 0)) # 0
print(AND(0, 1)) # 0
print(AND(1, 1)) # 1
```

可以以同样的方式实现与非门和或门，不过在此之前先做一些修改。

### 6.3.2 导入权重和偏置

考虑到以后的事情，我们将其修改为另外一种实现形式。将  $\theta$  换为  $-b$ ，表示感知机行为的式子变为如下形式：

$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

$b$  为偏置， $w_1, w_2$  为权重。下面我们使用 Numpy 按上述形式分别实现与门、与非门和或门。

与门：

```
import numpy as np
```

```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

print(AND(0, 0)) # 0
print(AND(1, 0)) # 0
print(AND(0, 1)) # 0
print(AND(1, 1)) # 1
```

偏置 $b$ 与权重 $w_1, w_2$ 的作用是不同的，具体地说 $w_1, w_2$ 是控制输入信号的重要性的参数，而偏置是调整神经元被激活的容易程度的参数。

接下来实现与非门和或门：

```
import numpy as np

def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

print(NAND(0, 0)) # 1
print(NAND(1, 0)) # 1
print(NAND(0, 1)) # 1
print(NAND(1, 1)) # 0

def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
```

```
b = -0.2
tmp = np.sum(w * x) + b
if tmp <= 0:
    return 0
else:
    return 1

print(OR(0, 0)) # 0
print(OR(1, 0)) # 1
print(OR(0, 1)) # 1
print(OR(1, 1)) # 1
```

与门、与非门、或门是具有相同构造的感知机，区别仅在于权重参数的值。因此在与非门和或门的实现中，仅设置权重和偏置的值这一点和与门的实现不同。

## 6.4 感知机的局限

现在来考虑一下异或门（XOR gate）。异或门仅当 $x_1$ 或 $x_2$ 中的一方为 1 时才会输出 1。

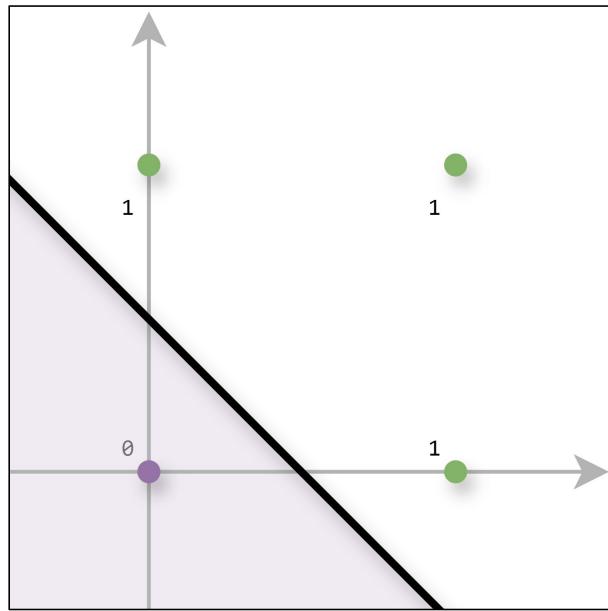
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0

之前的感知机是无法实现这个异或门的，具体原因让我们来分析一下。

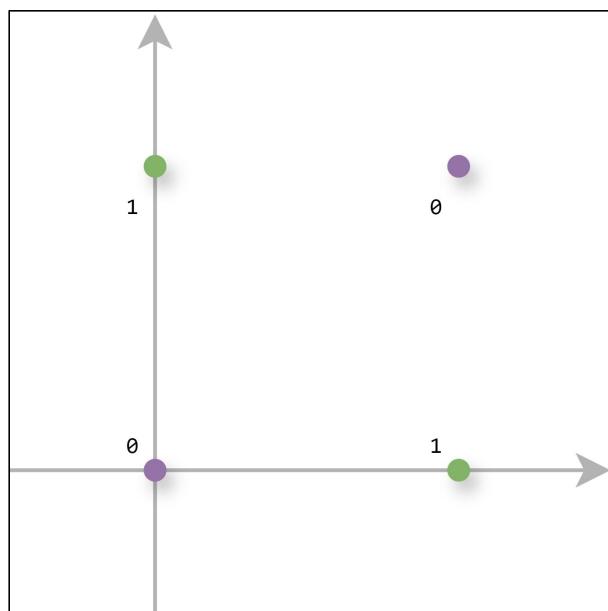
感知机表示或门的式子如下：

$$y = \begin{cases} 0 & (-0.5 + x_1 + x_2 \leq 0) \\ 1 & (-0.5 + x_1 + x_2 > 0) \end{cases}$$

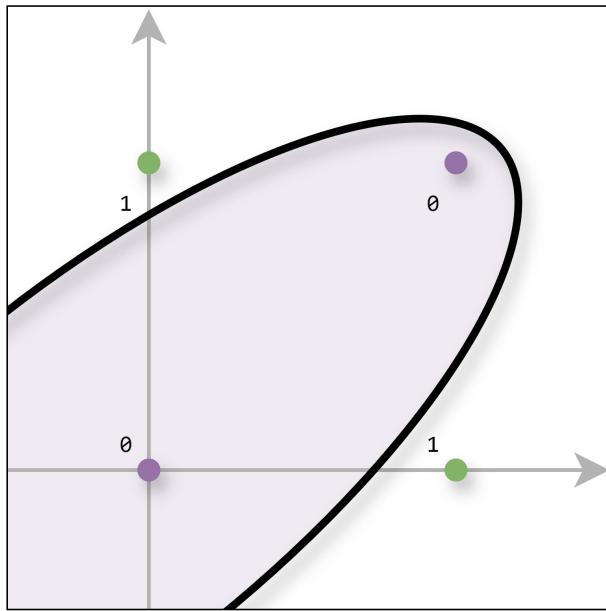
上式表示感知机会生成由直线 $-0.5 + x_1 + x_2 = 0$  划分的两个空间，其中一个空间输出 1，另一个空间输出 0，如图：



如果要实现异或门，是无法使用一条直线将 0 和 1 分开的：



但如果将直线这个限制条件去掉就可以了：

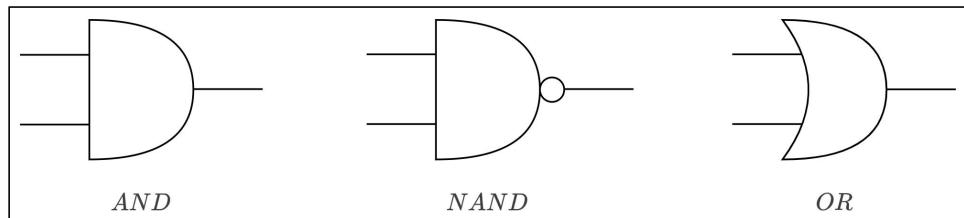


感知机的局限性就在于它只能表示由一条直线划分的空间，而上图中的曲线无法用感知机表示。由曲线划分的空间称为非线性空间，由直线划分的空间称为线性空间。

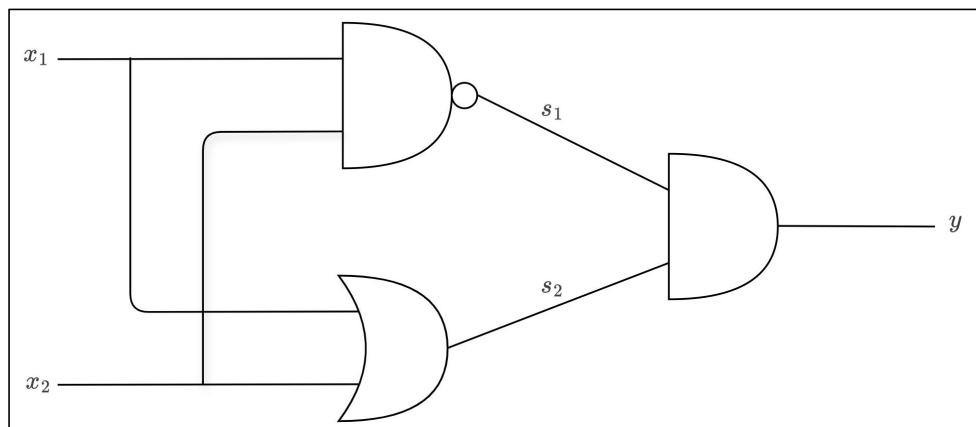
## 6.5 多层感知机

我们先来考虑一下如何使用与门、与非门和或门来组合出异或门。

与门、与非门、或门用下图的符号表示：



将其组合构成异或门：



$x_1$	$x_2$	$s_1$	$s_2$	$y$
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

使用之前实现的与门、与非门和或门代码来实现异或门：

```
import numpy as np

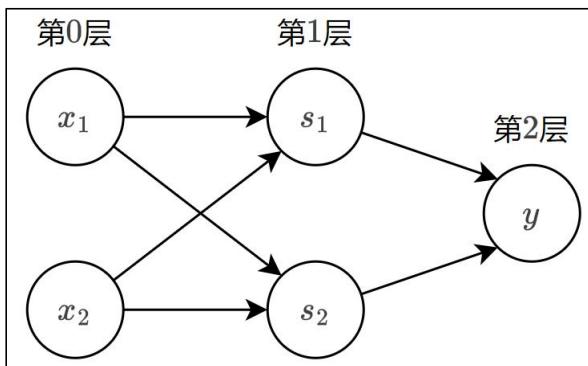
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
```

```
else:  
    return 1  
  
def XOR(x1, x2):  
    s1 = NAND(x1, x2)  
    s2 = OR(x1, x2)  
    y = AND(s1, s2)  
    return y  
  
print(XOR(0, 0)) # 0  
print(XOR(1, 0)) # 1  
print(XOR(0, 1)) # 1  
print(XOR(1, 1)) # 0
```

试着使用感知机的表示方法来表示这个异或门：



如图所示，这是一种多层次感知机结构。

第 0 层的两个神经元接收输入信号，并将信号发送给第 1 层的神经元。第 1 层的神经元将信号发送给第 2 层的神经元。第 2 层的神经元输出结果。

通过叠加层，感知机能进行更加灵活的表示。

感知机是一种非常简单的算法，也是后续学习神经网络的基础。

## 第 7 章 其他监督学习算法（了解）

### 7.1 朴素贝叶斯法

#### 7.1.1 朴素贝叶斯法简介

朴素贝叶斯（naive Bayes）法是一种基于概率的机器学习算法。它基于贝叶斯定理，并假设特征之间相互独立（这就是“朴素”的来源）。朴素贝叶斯法实现简单，学习与预测的

效率都很高，是一种常用方法，在许多场景下表现得非常好，如文本分类（垃圾邮件检测）、情感分析等。

朴素贝叶斯法的核心是贝叶斯定理：

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- $P(Y|X)$ : 后验概率，给定特征 $X$ 时类 $Y$ 的概率。
- $P(X|Y)$ : 条件概率，类 $Y$ 包含特征 $X$ 的概率。
- $P(Y)$ : 先验概率，类 $Y$ 的概率。
- $P(X)$ : 特征 $X$ 的概率。

例如，判断一封邮件是不是垃圾邮件（ $Y=$ 垃圾邮件， $X=$ 邮件内包含“免费”这个词）：

- $P(Y|X)$ : 邮件包含“免费”时是垃圾邮件的概率。
- $P(X|Y)$ : 垃圾邮件中包含“免费”的概率。
- $P(Y)$ : 邮件是垃圾邮件的概率。
- $P(X)$ : 邮件包含“免费”的概率。

朴素贝叶斯假设所有特征相互独立，这使得我们无需考虑特征之间复杂的依赖关系，极大简化了条件概率的计算：

$$P(X_1, X_2, \dots, X_n|Y) = P(X_1|Y) \cdot P(X_2|Y) \cdot \dots \cdot P(X_n|Y) = \prod_{j=1}^n P(X_j|Y)$$

### 7.1.2 极大似然估计

在朴素贝叶斯法中，学习意味着估计先验概率和条件概率。可以应用极大似然估计法估计相应的概率。

先验概率 $P(Y = C_k)$ （ $Y$ 为类 $C_k$ ）的极大似然估计：

$$P(Y = C_k) = \frac{\sum_{i=1}^N I(y_i = C_k)}{N}, \quad k = 1, 2, \dots, K$$

条件概率 $P(X_j = a_{jl}|Y = C_k)$ （ $Y$ 为类 $C_k$ 时第 $j$ 个特征 $X_j$ 为 $a_{jl}$ ）的极大似然估计：

$$P(X_j = a_{jl}|Y = C_k) = \frac{\sum_{i=1}^N I(x_{ji} = a_{jl}, y_i = C_k)}{\sum_{i=1}^N I(y_i = C_k)}$$

$$j = 1, 2, \dots, n; \quad l = 1, 2, \dots, L; \quad k = 1, 2, \dots, K$$

其中 $I$ 为示性函数，取值为 1 或 0。

### 7.1.3 贝叶斯估计

使用极大似然估计可能会出现所要估计的概率值为 0 的情况，这会影响到后验概率的计算结果，使分类产生偏差。解决方法是改用贝叶斯估计。

先验概率的贝叶斯估计：

$$P_\lambda(Y = C_k) = \frac{\sum_{i=1}^N I(y_i = C_k) + \lambda}{N + K\lambda}, \quad k = 1, 2, \dots, K$$

条件概率的贝叶斯估计：

$$P_\lambda(X_j = a_{jl}|Y = C_k) = \frac{\sum_{i=1}^N I(x_{ji} = a_{jl}, y_i = C_k) + \lambda}{\sum_{i=1}^N I(y_i = C_k) + L\lambda}$$

$$j = 1, 2, \dots, n; \quad l = 1, 2, \dots, L; \quad k = 1, 2, \dots, K$$

式中  $\lambda \geq 0$ ，当  $\lambda = 0$  时就是极大似然估计，常取  $\lambda = 1$ ，这时称为拉普拉斯平滑。

#### 7.1.4 学习与分类过程

学习时，计算先验概率：

$$P(Y = C_k), \quad k = 1, 2, \dots, K$$

和条件概率：

$$P(X_j = a_{jl}|Y = C_k), \quad j = 1, 2, \dots, n; \quad l = 1, 2, \dots, L; \quad k = 1, 2, \dots, K$$

分类时，根据给定的实例  $x = (x_1, x_2, \dots, x_n)$  计算后验概率：

$$P(Y = C_k) \prod_{j=1}^n P(X_j = x_j|Y = C_k), \quad k = 1, 2, \dots, K$$

并确定实例  $x$  属于哪一个类别：

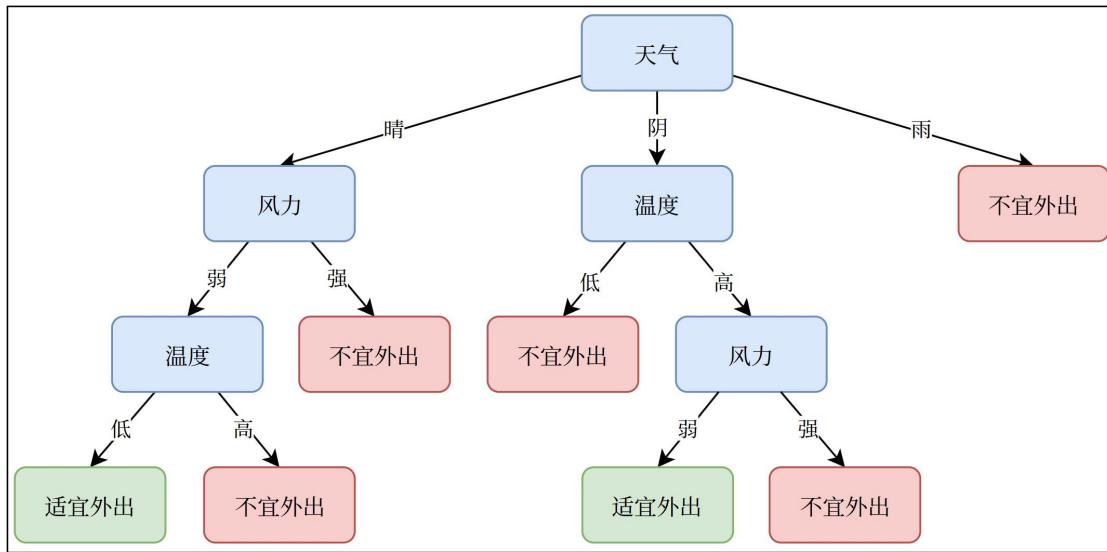
$$y = \operatorname{argmax}_{C_k} P(Y = C_k) \prod_{j=1}^n P(X_j = x_j|Y = C_k), \quad k = 1, 2, \dots, K$$

## 7.2 决策树

### 7.2.1 决策树简介

决策树（Decision Tree）是一种基于树形结构的算法，根据一系列条件判断逐步划分数据，缩小范围，最终得出预测结果。决策树由 4 部分组成：

- 根节点：树的节点，包含所有数据。
- 内部节点：表示特征上的判断条件。
- 分支：根据判断条件分出的路径。
- 叶节点：最终分类或回归的结果。



决策树适用于需要规则化、可解释性和快速决策的场景，尤其在数据特征明确、样本量适中的情况下表现良好。在复杂任务中，它常作为基础模型，与集成学习结合（如随机森林、梯度提升树）以提升性能。

## 7.2.2 决策树工作过程

决策树的学习通常包括 3 个步骤：特征选择、决策树的生成和决策树的剪枝。

如果特征数量很多，可以在决策树学习之前对特征进行选择，只留下对训练数据有足够的分类能力的特征。

学习时通常是递归地选择最优特征，并根据该特征对训练数据进行划分，使得对各个子数据集有一个最好的分类。首先构建根结点，将所有训练数据都放在根结点。选择一个最优特征，按照这一特征将训练数据集划分成子集，使得各个子集有一个在当前条件下最好的分类。如果这些子集已经能够被基本正确分类，那么构建叶结点，并将这些子集分到所对应的叶结点中去；如果还有子集不能被基本正确分类，那么就对这些子集选择新的最优特征，继续对其进行划分并构建相应的结点。如此递归直至所有训练数据子集被基本正确分类，或者没有合适的特征为止。最后每个子集都被分到叶结点上，即都有了明确的类，这就生成了一棵决策树。决策树的每次划分都相当于在特征空间中引入一个超平面将当前空间一分为二。

以上方法生成的决策树可能对训练数据有很好的分类能力，但对未知的测试数据却未必，即可能发生过拟合现象。因此需要对已生成的树自下而上进行剪枝，将树变得更简单，从而使它具有更好的泛化能力。具体地，就是去掉过于细分的叶结点，使其回退到父结点或更高的结点，然后将父结点或更高的结点改为新的叶结点。

决策树的生成只考虑局部最优，决策树的剪枝则考虑全局最优。

### 7.2.3 特征选择与决策树生成

特征选择在于选取对训练数据具有分类能力的特征，这样可以提高决策树学习的效率。如果利用一个特征进行分类的结果与随机分类的结果没有很大差别，则称这个特征是没有分类能力的，经验上扔掉这样的特征对决策树学习的精度影响不大。通常特征选择的准则是信息增益或信息增益率。

#### 1) 信息熵

信息熵（Entropy）是表示随机变量不确定性的度量，设 $X$ 是一个取有限个值的离散随机变量，其概率分布为：

$$P(X = x_i) = p_i, \quad i = 1, 2, \dots, n$$

随机变量 $X$ 的熵定义为：

$$H(X) = - \sum_{i=1}^n p_i \log p_i$$

若 $p_i = 0$ ，则定义 $0 \log 0 = 0$ 。通常式中对数以 2 或 e 为底。熵只依赖于 $X$ 的分布，与 $X$ 的取值无关。熵越大，随机变量的不确定性就越大。

设有随机变量 $(X, Y)$ ，其联合概率分布为：

$$P(X = x_i, Y = y_j) = p_{ij}, \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, m$$

条件熵 $H(Y|X)$ 表示在已知随机变量 $X$ 的条件下随机变量 $Y$ 的不确定性：

$$H(Y|X) = \sum_{i=1}^n P(X = x_i) H(Y|X = x_i)$$

#### 2) 信息增益与 ID3

决策树学习应用信息增益（Information Gain）准则选择特征，给定训练集数据 $D$ 和特征 $A$ ，熵 $H(D)$ 表示对数据集进行分类的不确定性，条件熵 $H(D|A)$ 表示在特征 $A$ 给定条件下对数据集 $D$ 进行分类的不确定性。两者之差即信息增益：

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

表示由于特征 $A$ 而使得对数据集 $D$ 的分类的不确定性减少的程度。对数据集 $D$ 而言，信息增益依赖于特征，不同的特征具有不同的信息增益，信息增益大的特征具有更强的分类能力。在进行特征选择时，对训练数据集 $D$ 计算每个特征的信息增益，并比较他们的大小，选择信息增益最大的特征。

假设：

- 训练数据集  $D$  有  $|D|$  个样本
- 特征  $A$  有  $n$  个不同的取值  $\{a_1, a_2, \dots, a_n\}$ , 根据特征  $A$  将  $D$  划分为  $n$  个子集  $D_1, D_2, \dots, D_n$ ,  $D_i$  的样本个数为  $|D_i|$
- 有  $K$  个类  $C_k$ , 每个类有  $|C_k|$  个样本
- 子集  $D_i$  中属于类  $C_k$  的样本的集合为  $D_{ik}$  ( $D_{ik} = D_i \cap C_k$ ),  $D_{ik}$  的样本个数为  $|D_{ik}|$ 。

信息增益计算方法如下：

(1) 计算数据集  $D$  的熵  $H(D)$ :

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

(2) 计算特征  $A$  对数据集  $D$  的条件熵  $H(D|A)$ :

$$H(D|A) = \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \sum_{k=1}^K \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

(3) 计算信息增益:

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

ID3 算法在决策树各个节点上应用信息增益选择特征，递归地构建决策树。具体方法是：

从根节点开始，计算所有可能的信息增益，选择信息增益最大的特征作为节点特征，由该特征的不同取值建立子节点，再依次对子节点进行上述操作，直到所有特征信息增益均很小或无特征可选为止，最终生成一颗决策树。

### 3) 信息增益率与 C4.5

使用信息增益划分训练数据集的特征，会倾向于选择取值较多的特征。而使用信息增益率（Information Gain Ratio）可以对这一问题进行校正，这是特征选择的另一个准则。

特征  $A$  对训练数据集  $D$  的信息增益率  $g_R(D, A)$  定义为信息增益  $g(D, A)$  与训练数据集  $D$  关于特征  $A$  的值的熵  $H_A(D)$  之比：

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)}$$

其中  $H_A(D) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$ ,  $n$  是特征  $A$  的取值个数。

C4.5 算法与 ID3 算法相似，但对其进行了改进，使用信息增益率来选择特征。

### 4) 基尼指数与 CART

有 $K$ 个类，样本属于第 $k$ 类的概率为 $p_k$ ，则概率分布的基尼指数：

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

对于给定的样本集合 $D$ ，其基尼指数：

$$Gini(D) = 1 - \sum_{k=1}^K \left( \frac{|C_k|}{|D|} \right)^2$$

这里 $C_k$ 是 $D$ 中属于第 $k$ 类的样本子集， $K$ 是类的个数。

如果样本集合 $D$ 根据特征 $A$ 是否取某一可能值 $a$ 被划分为两个子集 $D_1, D_2$ ，则在特征 $A$ 的条件下，集合 $D$ 的基尼指数：

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

基尼指数 $Gini(D)$ 表示集合 $D$ 的不确定性，基尼指数 $Gini(D, A)$ 表示经 $A = a$ 划分后集合 $D$ 的不确定性。基尼指数越大样本集合的不确定性也越大，这与信息熵相似。

CART 决策树是一棵二叉树，根据基尼指数生成决策树，对训练数据集 $D$ 的每个特征 $A$ 的每一个可能的取值 $a$ ，计算 $A = a$ 时的基尼指数，选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点，并生成两个子节点，将训练数据集依特征分配到两个子节点中。重复上述过程，直到节点中样本数小于阈值、或样本集的基尼指数小于阈值、或没有个更多特征。

## 5) CART 回归树

一颗回归树对应着输入空间（特征空间）的划分，以及在划分上的输出值。将输入空间划分为 $M$ 个单元 $R_1, R_2, \dots, R_M$ ，并且在每个单元 $R_m$ 上有一个固定输出 $c_m$ 的回归树模型可表示为：

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m)$$

在训练数据集所在的空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树。每次划分的目标是找到一个特征 $j$ 和该特征的一个切分点 $s$ ，使得划分后的误差最小。在对输入空间进行划分时，将特征 $j$ 和它的取值 $s$ （通常是连续特征取值的中间点）作为切分变量和切分点，定义两个区域：

$$R_1(j, s) = \{x | x_j \leq s\} \text{ 和 } R_2(j, s) = \{x | x_j > s\}$$

然后寻找最优切分变量 $j$ 和最优切分点 $s$ 。具体地，使用平方误差作为损失函数，求解：

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

解出最优切分变量 $j$ 和最优切分点 $s$ 后，使用最优切分变量 $j$ 和最优切分点 $s$ 后划分两个区域。重复上述过程直到满足停止条件为止。最终叶节点的输出值为该区域内所有样本目标值的均值：

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m(j,s))$$

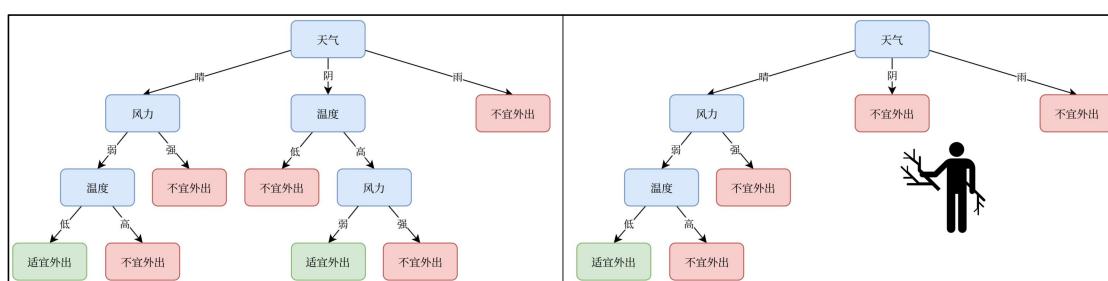
由此生成一颗回归树，这样的回归树称之为最小二乘回归树。

#### 7.2.4 决策树的剪枝

决策树出现过拟合的原因在于学习时过多的考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树。为了避免过拟合，可以对已生成的决策树进行剪枝，从决策树上裁掉一些子树或叶节点，并将其根节点或父节点作为新的叶节点，从而简化模型。决策树的剪枝通常分为预剪枝（Pre-Pruning）和后剪枝（Post-Pruning）。

预剪枝是在决策树生成过程中，通过设置一些限制条件提前停止树的生长，避免过度分裂。常见的停止条件有：限制最大树深度、限制每个节点最小样本数、限制最小的误差减少量、限制最大叶节点数量等。但过于严格的停止条件可能导致欠拟合，并且可能难以确定最佳阈值，需要多次尝试。

后剪枝是在决策树完全生成之后，基于某种评估准则从底部向上逐步判断是否移除分支。常见的后剪枝方法有代价复杂度剪枝（Cost-Complexity Pruning, CCP）和减少误差剪枝（Reduced Error Pruning, REP）等。



代价复杂度剪枝需要首先计算子树的损失函数：

$$C_\alpha(T) = C(T) + \alpha|T|$$

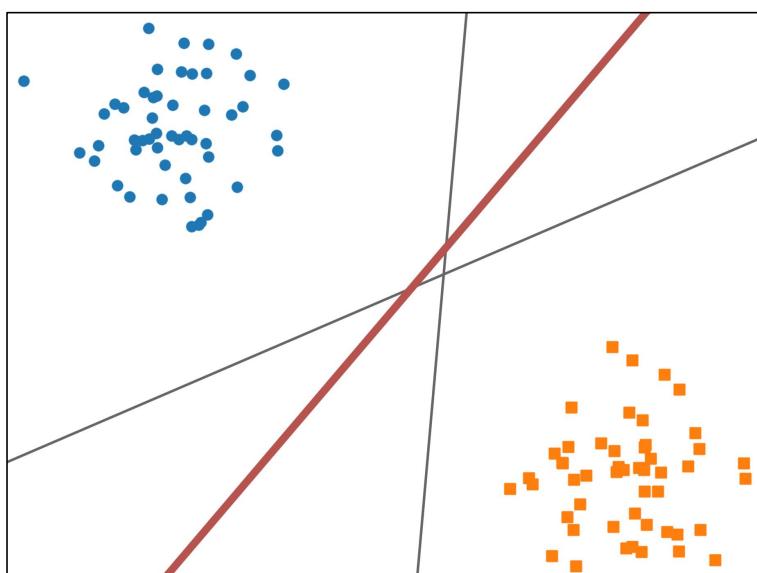
- $T$ 为任意子树， $|T|$ 为子树的叶节点个数
- $C(T)$ 为对训练数据的预测误差（如基尼指数）

- $\alpha \geq 0$  为参数，权衡训练数据的拟合程度和模型的复杂度。 $\alpha$ 越大越倾向于剪掉子树，生成更简单的树。
- $C_\alpha(T)$  为参数是  $\alpha$  时子树  $T$  的整体损失  
从  $\alpha = 0$  开始，逐渐增加  $\alpha$ 。对于每个  $\alpha$ ，选择剪掉使  $C_\alpha(T)$  最小的子树。生成一系列剪枝后的树，使用交叉验证选择最优的  $\alpha$  和对应的树。  
减少误差剪枝则是直接基于验证集的误差来判断是否剪枝。剪枝过程遍历每个内部节点，将该子树替换为一个叶节点并在验证集上比较替换前后的误差，如果替换后误差没有增加则剪掉该子树。

## 7.3 支持向量机

### 7.3.1 支持向量机简介

支持向量机（Support Vector Machines, SVM）是一种二分类模型。其核心目标是寻找一个“间隔最大”的超平面将不同类别的数据点分隔开。这个超平面在二维空间中是一条直线，在三维空间中是一个平面，在更高维空间中则是一个超平面。



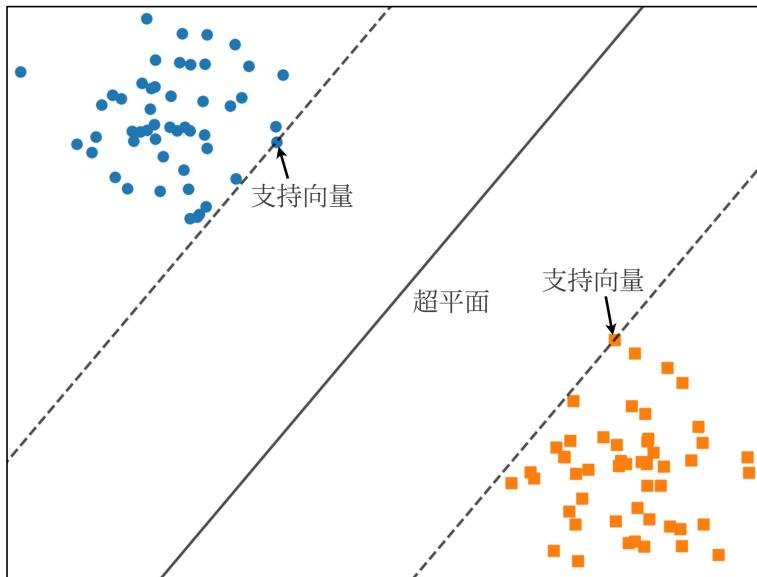
间隔最大代表该超平面与最近的数据点之间的距离最大，这使得支持向量机有较强的泛化能力。

支持向量机学习方法包括由简至繁的模型：线性可分支持向量机、线性支持向量机以及非线性支持向量机。

### 7.3.2 线性可分支持向量机-硬间隔

#### 1) 硬间隔

当训练样本线性可分时，此时可以通过最大化硬间隔来学习线性可分支持向量机。硬间隔是指超平面能够将不同类的样本完全划分开。距离超平面最近的几个样本点称为支持向量，它们直接决定超平面的位置和方向，只要支持向量不变，超平面就不会变。



在样本空间中，超平面可表示为

$$\mathbf{w}^T \mathbf{x} + b = 0$$

其中  $\mathbf{w} = (w_1, w_2, \dots, w_n)$  为法向量，决定了超平面的方向； $b$  为位移项，决定了超平面与原点之间的距离。将超平面记为  $(\mathbf{w}, b)$

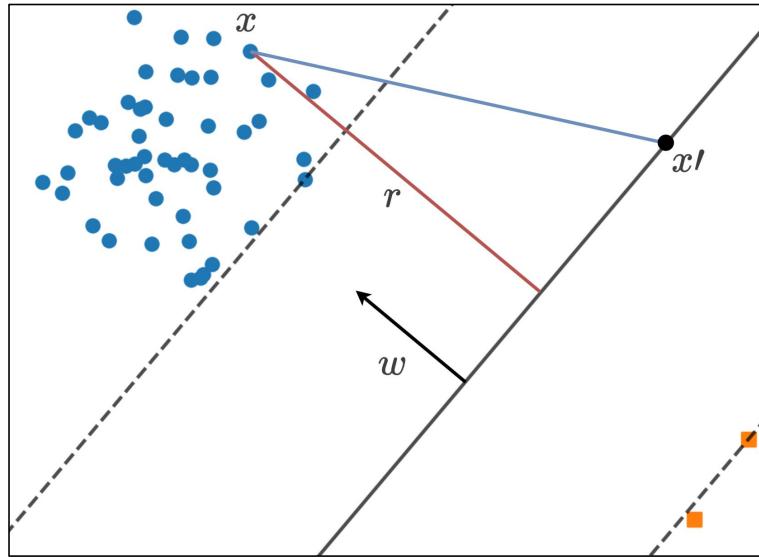
相应的分类函数称为线性可分支持向量机：

$$f(x) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

## 2) 间隔与最大间隔

$\mathbf{x}'$  为超平面上一点， $\mathbf{w}^T \mathbf{x}' + b = 0$ ，样本空间中任一点  $\mathbf{x}$  到超平面  $(\mathbf{w}, b)$  的距离为：

$$r = \left| \frac{\mathbf{w}^T (\mathbf{x} - \mathbf{x}')}{\|\mathbf{w}\|} \right| = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}$$



记每个样本点 $\mathbf{x}_i$ 的类别为 $y_i$ , 该样本点的函数间隔 $\hat{\gamma}_i = y_i(\mathbf{w}^T \mathbf{x} + b)$ , 表示分类预测的正确性及确信度, 若超平面 $(\mathbf{w}, b)$ 能将样本正确分类, 则有

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b > 0, & y_i = +1 \\ \mathbf{w}^T \mathbf{x}_i + b < 0, & y_i = -1 \end{cases}$$

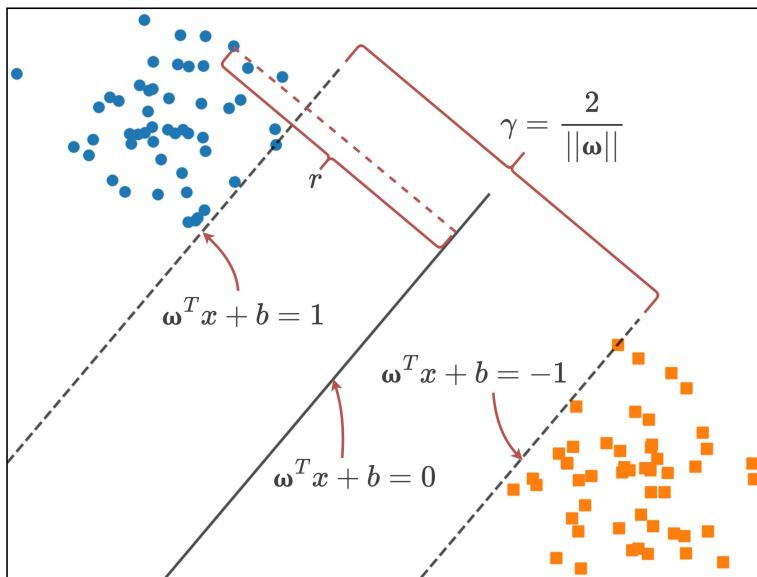
此时有

$$y_i(\mathbf{w}^T \mathbf{x} + b) = |\mathbf{w}^T \mathbf{x} + b| > 0$$

$$r_i = \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|} = \frac{y_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} = \frac{\hat{\gamma}_i}{\|\mathbf{w}\|}$$

我们注意到对 $\mathbf{w}, b$ 进行缩放变换 $\mathbf{w} \rightarrow \lambda \mathbf{w}, b \rightarrow \lambda b$ 时, 不会改变超平面, 也不会改变 $r$ 的值, 但函数间隔 $\hat{\gamma}_i = y_i(\mathbf{w}^T \mathbf{x}_i + b)$ 会随着缩放 $\mathbf{w}$ 和 $b$ 而发生变化。也就是说可以通过缩放 $\mathbf{w}$ 和 $b$ 来任意缩放函数间隔 $\hat{\gamma}_i$ 而不改变 $r_i$ 。

因此令支持向量到超平面的函数间隔 $\hat{\gamma}_i = 1$ , 此时支持向量到超平面的距离 $r_i = \frac{\hat{\gamma}_i}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$ 。两个异类支持向量到超平面的距离之和 $\gamma = \frac{2}{\|\mathbf{w}\|}$ ,  $\gamma$ 被称为“间隔”。



欲找到具有最大间隔的超平面，也就是求在约束 $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ 下最大的 $\gamma$ :

$$\begin{aligned} & \max_{\mathbf{w}, b} \frac{2}{\|\mathbf{w}\|} \\ \text{s.t. } & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

等价于：

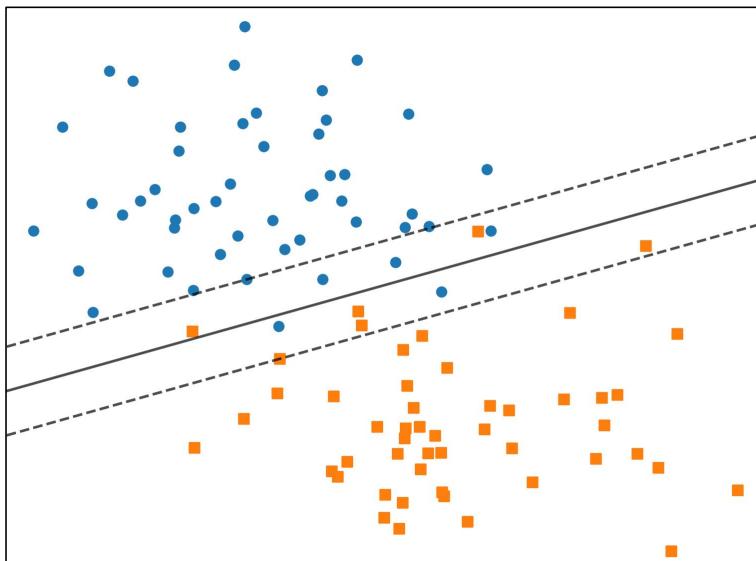
$$\begin{aligned} & \min_{\mathbf{w}, b} \frac{\|\mathbf{w}\|^2}{2} \\ \text{s.t. } & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

s.t. 为 Subject to，意为约束。

上式就是支持向量机的基本型。对上式使用拉格朗日乘子法得到其对偶问题，从对偶问题中解出拉格朗日乘子，进而解出 $\mathbf{w}, b$ ，即可得到具有最大间隔的超平面。

### 7.3.3 线性支持向量机-软间隔

先前我们假定训练样本在样本空间中线性可分，但现实中很可能并非如此，此时我们无法找出一个合适的超平面将所有样本点完全正确划分。通常训练数据集中会有一些特异点，如果将这些特异点去掉，剩下大部分样本点是线性可分的。这时我们可以放宽条件，允许某些样本分错，为此我们引入软间隔。



线性不可分意味着某些样本点 $(\mathbf{x}_i, y_i)$ 不能满足约束条件 $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ 。为解决这个问题，可以对每个样本点引进一个松弛变量 $\xi_i \geq 0$ ，使得函数间隔加上松弛变量 $\geq 1$ 。这时约束条件变为：

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i$$

同时为了在最大化间隔的时候使不满足约束的样本尽可能少，目标函数中引入对误分类的惩罚：

$$\frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n \xi_i$$

这里 $C > 0$  为惩罚系数， $C$ 值越大对误分类的惩罚越大。

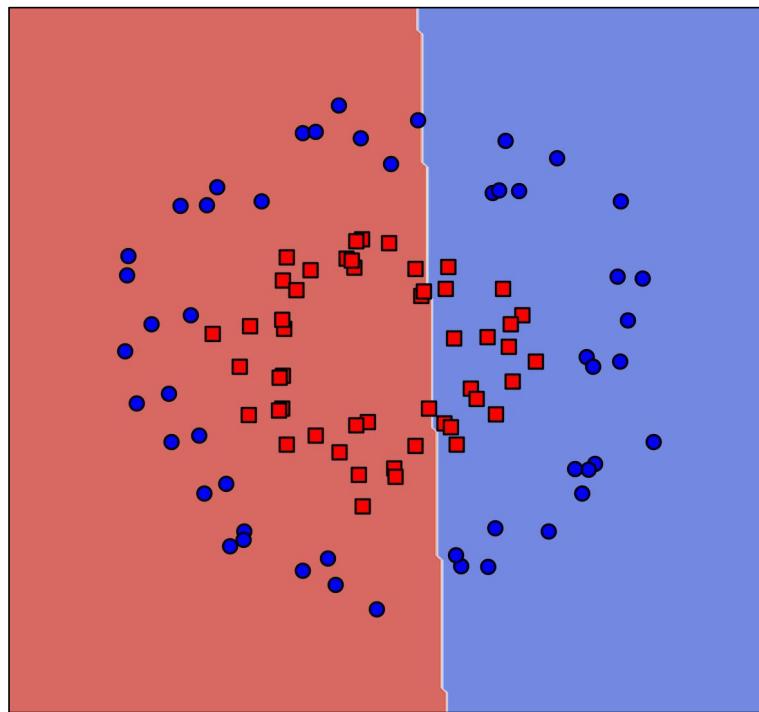
线性不可分的线性支持向量机的学习问题可表示如下：

$$\begin{aligned} & \min_{\mathbf{w}, b, \xi} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n \xi_i \\ & \text{s.t. } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \end{aligned}$$

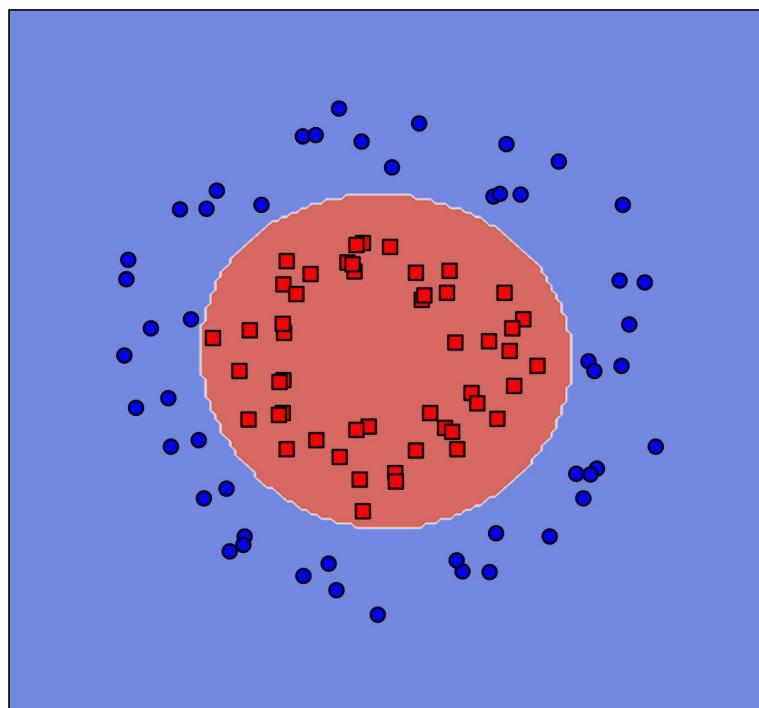
这就是软间隔支持向量机。

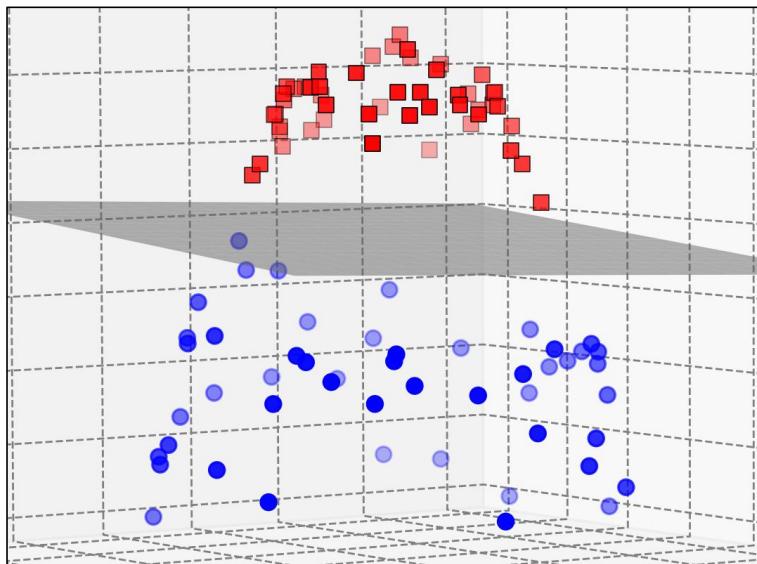
#### 7.3.4 非线性支持向量机-核函数

非线性分类问题是通过利用非线性模型才能很好地进行分类的问题，如下图是无法直接使用超平面对其分类的：



这时我们可以通过核函数将数据从原始空间映射到高维特征空间，使得数据在高维特征空间线性可分，将原本的非线性问题转换为线性问题。使用核技巧学习非线性支持向量机，等价于隐式地在高维特征空间中学习线性支持向量机。





$\phi(\mathbf{x})$ 表示将 $\mathbf{x}$ 映射后的特征向量，在特征空间中超平面可表示为：

$$\mathbf{w}^T \phi(\mathbf{x}) + b = 0$$

并且有：

$$\begin{aligned} & \min_{\mathbf{w}, b} \frac{\|\mathbf{w}\|^2}{2} \\ \text{s. t. } & y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 \end{aligned}$$

将其转化为对偶问题并求解，但其对偶问题涉及到计算 $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ ，这是样本 $\mathbf{x}_i$ 和样本 $\mathbf{x}_j$ 映射到特征空间之后的内积。由于特征空间维度很高，因此直接计算 $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ 通常非常困难，为了避开这一障碍，设想这样一个函数：

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

$\kappa$ 就是核函数。显然若已知映射函数 $\phi$ 就可写出核函数，但实际任务中我们通常不知道 $\phi$ 的形式，那么如何判断给定的 $\kappa$ 是否是核函数呢？

实际上，只要一个对称函数所对应的核矩阵半正定，它就能作为核函数使用。这个定义在构造核函数时很有用，但对于一个具体函数 $\kappa$ ，检验它是否为正定核函数不容易，因此实际中往往使用已有的核函数。

核函数的选择也是支持向量机最大的变数，若核函数选择不合适，意味着将样本映射到了一个不合适的特征空间，很可能导致性能不佳。下面是几种常用的核函数：

线性核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
多项式核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$
高斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ ^2}{2\sigma^2}\right)$
拉普拉斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ }{\sigma}\right)$
Sigmoid 核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i^T \mathbf{x}_j + \theta)$

## 7.4 集成学习

集成学习（Ensemble Learning）通过某种策略组合多个个体学习器的预测结果来提高整体的预测能力。只包含同种类型的个体学习器的集成称为同质集成，例如决策树集成中全都是决策树，同质集成中的个体学习器亦称基学习器，相应的学习算法称为基学习算法。包含不同类型的个体学习器的集成称为异质集成，例如同时包含决策树和神经网络。

集成学习有三大经典方法：Boosting、Bagging 和 Stacking。

Boosting（提升方法）按顺序训练模型，每个模型关注前一个模型的错误，通过加权调整来优化整体预测。如 AdaBoost 通过给错分的样本更大的权重，逐步改进；梯度提升树用梯度下降法优化损失函数；XGBoost 和 LightGBM 是高效的梯度提升树变种。Boosting 主要关注于降低偏差。

Bagging（Bootstrap Aggregating，自助聚合）从原始数据集中通过有放回的对样本采样生成多个子数据集，分别训练多个独立模型，最后通过投票（分类）或平均（回归）得到结果。随机森林则是在 Bagging 基础上随机选择特征子集训练每棵树。Bagging 主要关注于降低方差。

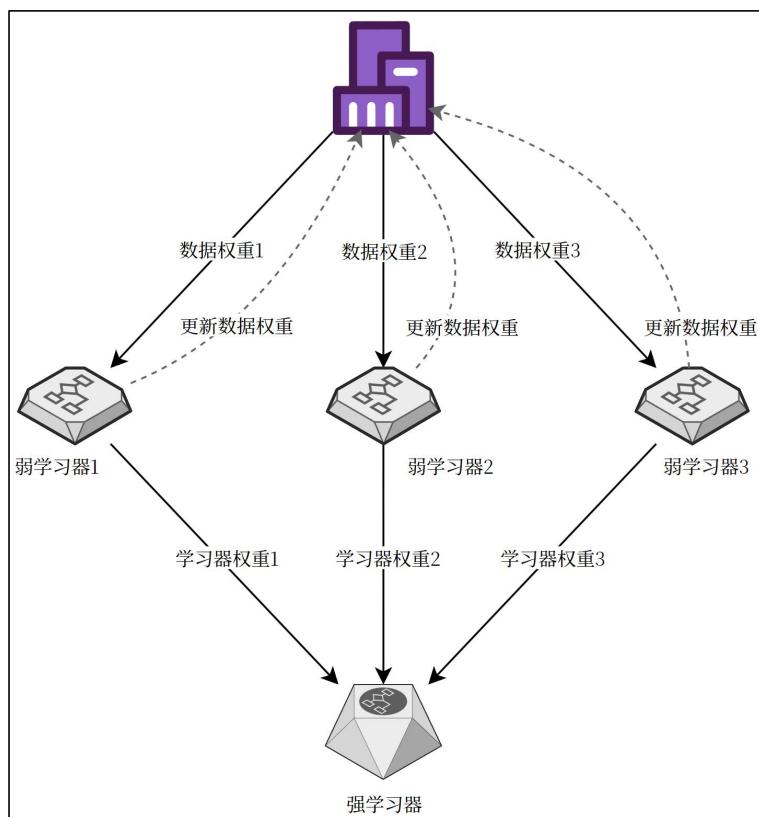
Stacking（堆叠）训练多个不同类型的个体学习器，之后使用一个元模型综合多个个体学习器的预测。灵活性强，能结合多种模型的优势。

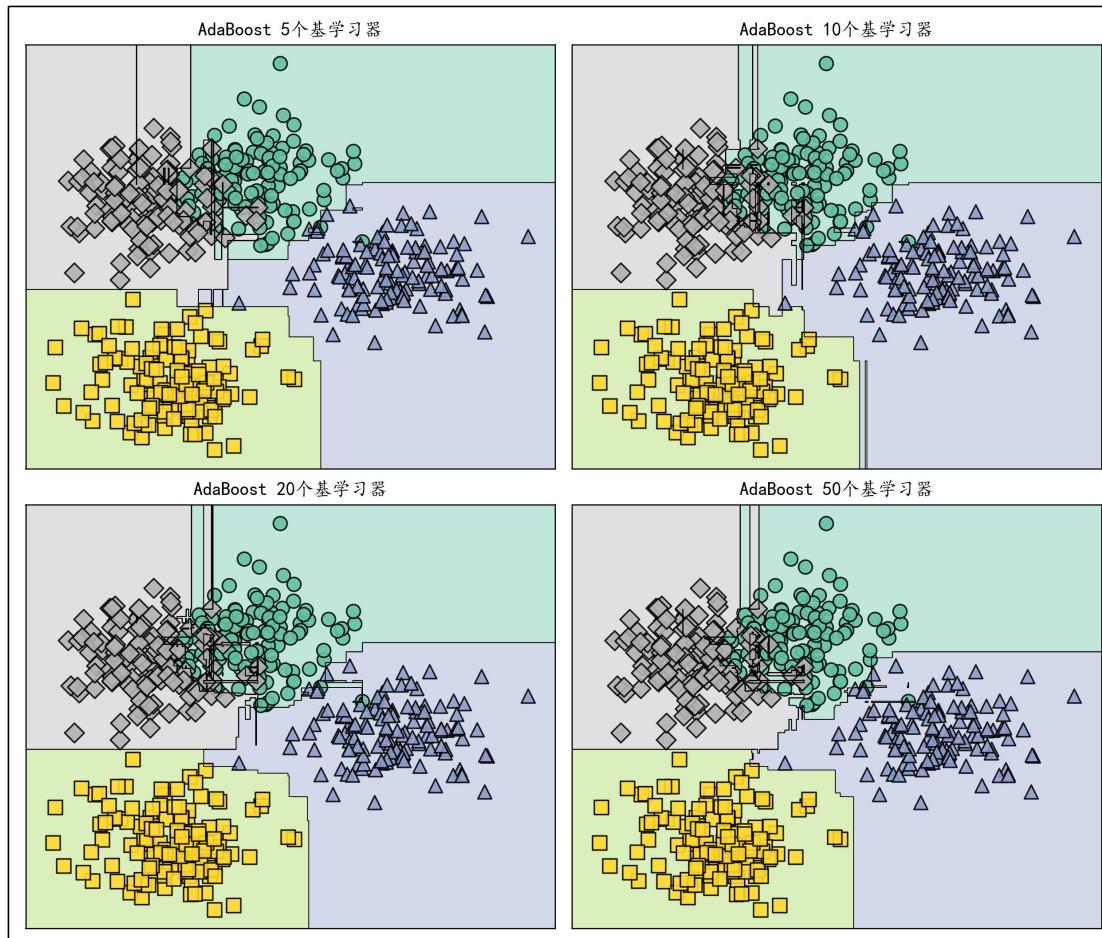
### 7.4.1 AdaBoost

在概率近似正确学习的框架中，一个概念如果存在一个多项式的学习算法能够学习它，并且正确率很高，就称这个概念是强可学习的；一个概念如果存在一个多项式的学习算法能够学习它，但正确率仅比随机猜测略好，就称这个概念是弱可学习的。后来证明，强可学习与弱可学习是等价的。那么如果已经发现了弱学习算法，能否通过某种方式将其提升为强学习算法？

对于分类问题而言，给定一个训练样本集，求比较粗糙的分类规则（弱分类器）要比求精确的分类规则（强分类器）容易的多。Boosting 就是从弱学习算法出发，反复学习，得到一系列弱分类器，然后组合这些弱分类器构成一个强分类器。AdaBoost 通常使用单层决策树作为基学习器，单层决策树也被称为决策树桩（Decision Stump）。

大多数 Boosting 都是改变训练数据的概率分布（权重分布），针对不同的训练数据分布调用弱学习算法学习一系列弱分类器。AdaBoost（Adaptive Boosting，自适应提升）的做法是提高被前一轮弱分类器错误分类的样本的权重，降低被正确分类的样本的权重。这样一轮后弱学习器会更加关注那些没有被正确分类的数据。同时采用加权多数表决的方法，加大分类误差率小的弱分类器的权重，减小分类误差率大的弱分类器的权重。





AdaBoost 工作流程如下：

初始化训练数据权重分布，通常均匀分布：

$$D_{m=1} = (w_{1,1}, \dots, w_{1,i}, \dots, w_{1,N}), \quad w_{1i} = \frac{1}{N}; \quad i = 1, 2, \dots, N; \quad m = 1, 2, \dots, M$$

使用具有权重分布  $D_m$  的训练数据集学习，得到基本分类器：

$$G_m(x): \chi \rightarrow \{-1, +1\} \quad (G_m(x) \text{ 将输入映射到两个值 } -1 \text{ 和 } +1)$$

计算  $G_m(x)$  在训练数据集上的分类误差率：

$$e_m = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i)$$

计算弱分类器  $G_m(x)$  的权重：

$$a_m = \frac{1}{2} \ln \frac{1 - e_m}{e_m}$$

更新训练数据集的权重分布：

$$D_{m+1} = (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N})$$

$$\text{其中: } w_{m+1,i} = \frac{w_{m,1} \exp(-a_m y_i G_m(x_i))}{\sum_{i=1}^N w_{m,i} \exp(-a_m y_i G_m(x_i))}, \quad i = 1, 2, \dots, N$$

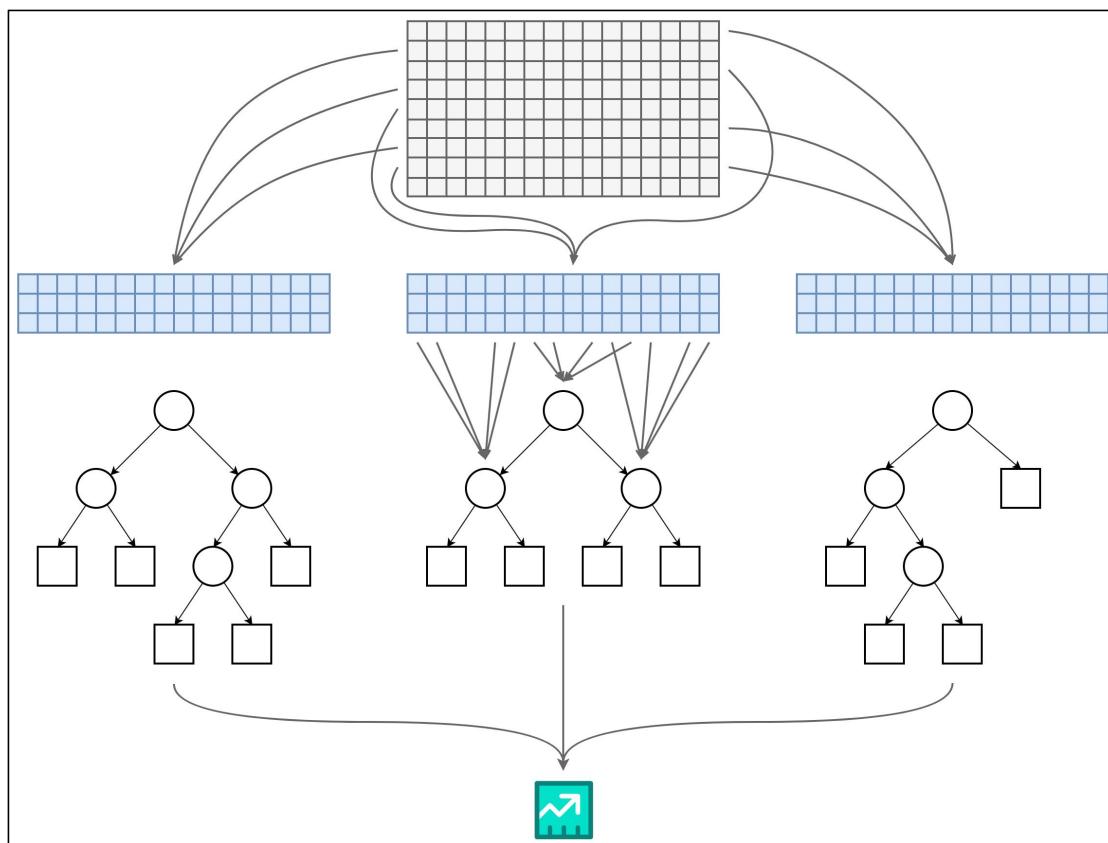
构建基本分类器的线性组合，得到最终分类器：

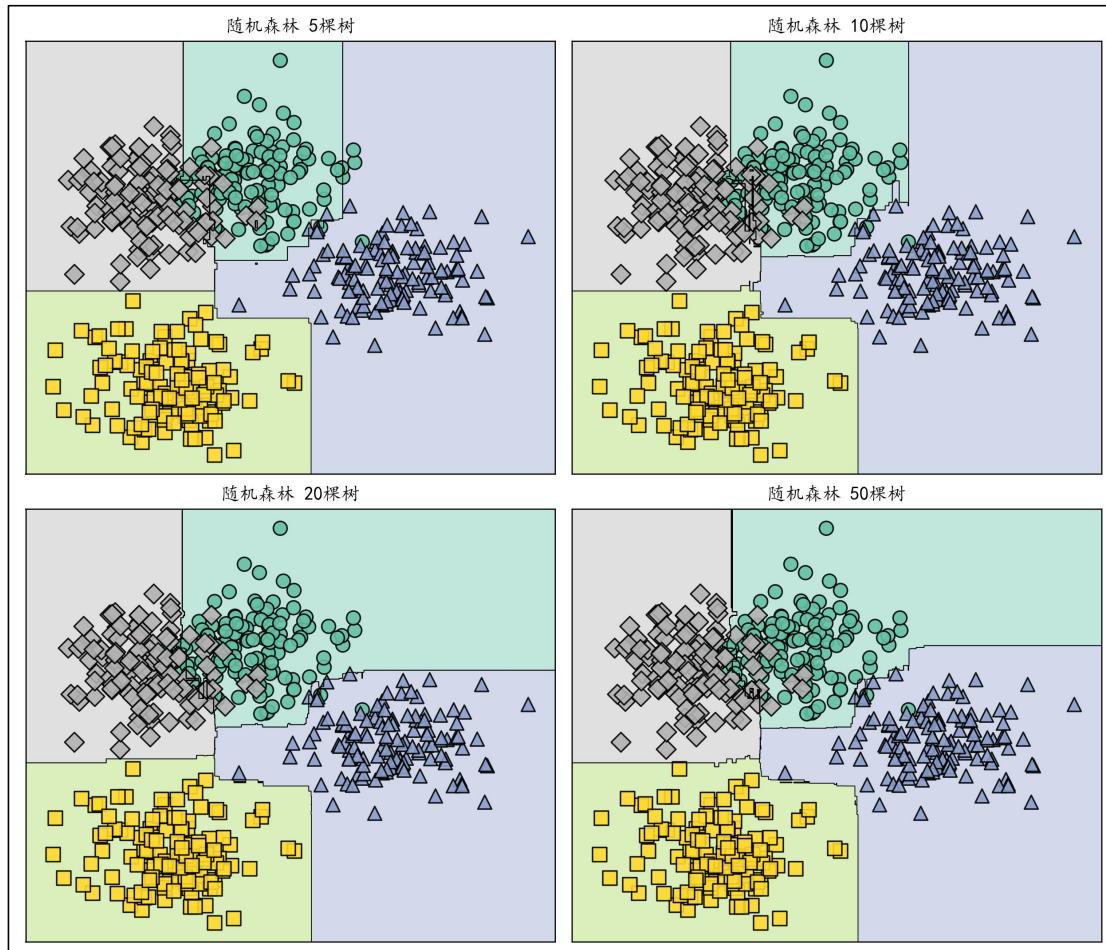
$$G(x) = \text{sign}\left(\sum_{m=1}^M a_m G_m(x)\right)$$

### 7.4.2 随机森林

随机森林是 Bagging 的一个变体，在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树训练过程中引入了随机属性选择。

具体来说，传统决策树在选择划分特征时是在当前节点的特征集合（假定有  $d$  个特征）中选择最优特征。而在随机森林中，基决策树的每个节点先从该节点的特征集合中随机选择一个包含  $k$  个特征的子集，然后再从这个子集中选择一个最优特征用于划分。参数  $k$  控制着随机性的引入程度，若  $k = d$ ，则基决策树的生成与传统决策树相同；若  $k = 1$ ，则随机选择一个属性用于划分。一般推荐  $k = \log_2 d$ 。





随机森林简单易实现，但在很多任务中都展现出了强大性能，被誉为“代表集成学习技术水平的方法”。Bagging 中基学习器的多样性仅来自于样本扰动，而随机森林中基学习器的多样性不仅来自样本扰动，还来自特征扰动，这就使得最终集成的泛化性能可通过基学习器之间差异度的增加而进一步提升。

## 第 8 章 无监督学习

### 8.1 聚类

#### 8.1.1 聚类简介

聚类（Clustering）旨在将数据集中的样本分成若干个簇，使得同一个簇内的对象彼此相似，不同簇间的对象差异较大。聚类是一种无监督学习算法，不需要预先标记数据的标签，完全依赖数据本身内在结构和特征来进行分组，最终簇所对应的概念语义需由使用者来把握和命名。

聚类的核心是“物以类聚”，具体通过以下步骤实现：

- 定义相似性：选择一个度量标准（如欧氏距离，余弦相似度）来衡量对象之间的相似性或距离。
- 分组：根据相似性将对象分配到不同的簇中。
- 优化：通过迭代或直接计算，调整簇的划分，使簇内相似性最大化，簇间差异最大化。

聚类应用场景：

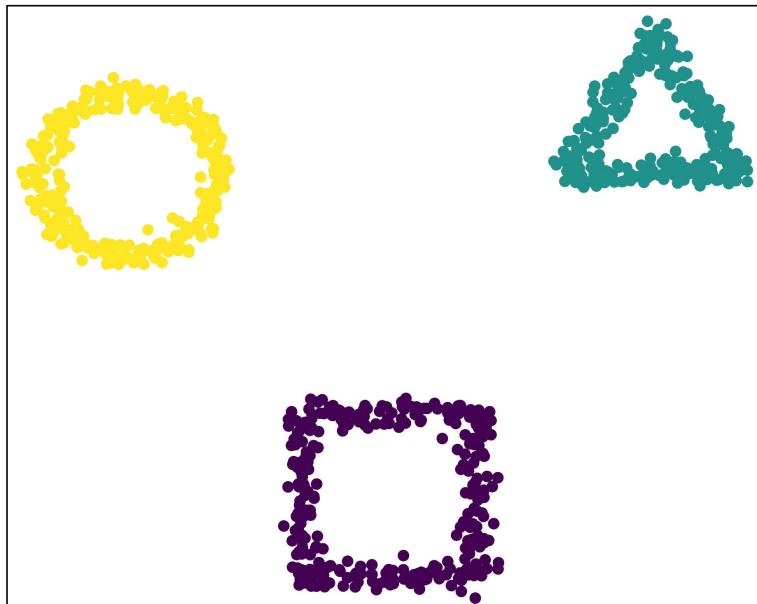
- 市场细分：将消费者按购买习惯分组。
- 图像分割：将图像像素按颜色或纹理聚类。
- 异常检测：识别不属于任何主要簇的异常点。
- 生物信息：对基因表达数据进行分组。

### 8.1.2 常见聚类算法

#### 1) K 均值聚类

##### (1) K 均值聚类介绍

K 均值聚类 (K-means) 是基于样本集合划分的聚类方法，将样本集合划分为  $k$  个子集构成  $k$  个簇，将  $n$  个样本分到  $k$  个簇中，每个样本到其所属簇的中心的距离最小。每个样本只能属于一个簇，所以 K 均值聚类是硬聚类。



K 均值聚类归结为样本集合的划分，通过最小化损失函数来选取最优的划分  $C^*$ 。

首先使用欧氏距离平方作为样本之间的距离：

$$d(x_i, x_j) = \|x_i - x_j\|^2$$

定义样本与其所属簇的中心之间的距离总和作为损失函数：

$$W(C) = \sum_{l=1}^k \sum_{C(i)=l} \|x_i - \bar{x}_l\|^2$$

- $\bar{x}_l$ : 第  $l$  个簇的中心。
- $C(i) = l$ : 第  $i$  个样本是否属于簇  $l$ 。

K 均值聚类就是求解最优化问题：

$$C^* = \operatorname{argmin}_C W(C) = \operatorname{argmin}_C \sum_{l=1}^k \sum_{C(i)=l} \|x_i - \bar{x}_l\|^2$$

相似的样本被聚到同一个簇时损失函数最小。这是一个组合优化问题， $n$  个样本分到  $k$  个簇，所有可能的分法数目  $S(n, k) = \frac{1}{k!} \sum_{l=1}^k (-1)^{k-l} \binom{k}{l} k^n$ ，这个数目是指数级的，现实中采用迭代的方法求解。

### (2) K 均值聚类工作流程

- 初始化，随机选择  $k$  个样本点作为初始簇中心。
- 对样本进行聚类，计算每个样本到各个簇中心的距离，将每个样本分到与其最近的簇，构成聚类结果。
- 计算聚类结果中每个簇中所有样本的均值，作为新的簇中心。
- 使用新的簇中心重复上述过程，直到收敛或符合停止条件（例如划分不再改变）。

### (3) K 均值聚类特点

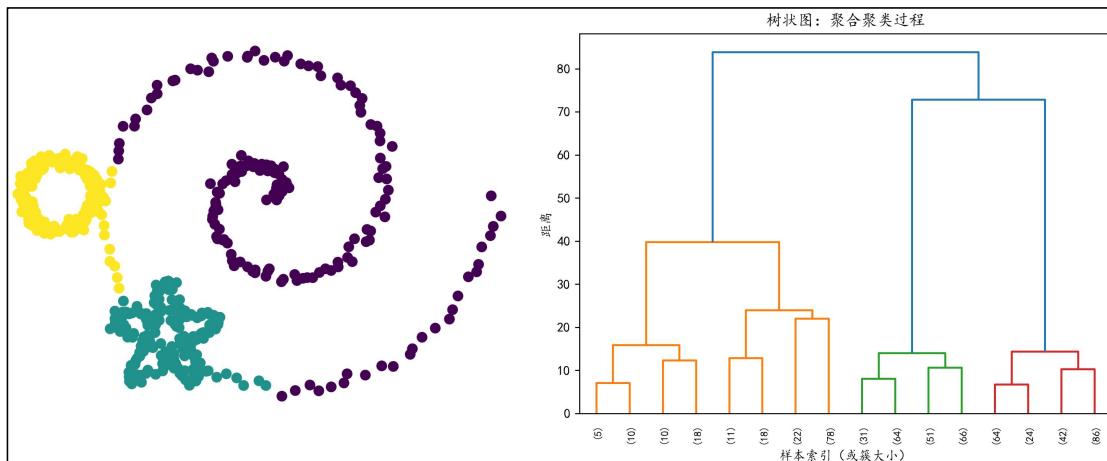
K 均值聚类的初始中心的选择会直接影响聚类结果，并且不适合非凸形状簇。

K 均值聚类需要事先指定簇个数  $k$ ，而实际中最优的  $k$  值是不知道的，需要尝试使用不同的  $k$  值检验聚类结果质量，可以采用二分查找快速找到最优  $k$  值。聚类结果的质量可以用簇的平均直径来衡量，一般地，簇个数变小时平均直径会增加；簇个数变大超过某个值后平均直径会不变，而这个值正是最优的  $k$  值。

## 2) 层次聚类

层次聚类（Hierarchical Clustering）假设簇之间存在层次结构，将样本聚到层次化的簇中。层次聚类有自下而上的聚合方法和自上而下的分裂方法。因为每个样本只属于一个簇，所以层次聚类属于硬聚类。

聚合聚类：开始将每个样本各自分到一个簇，之后将相距最近的两个簇合并，如此往复直至满足停止条件（例如达到预设的簇的个数、每个簇只包含一个样本、簇内样本相似性达到某个阈值等）。



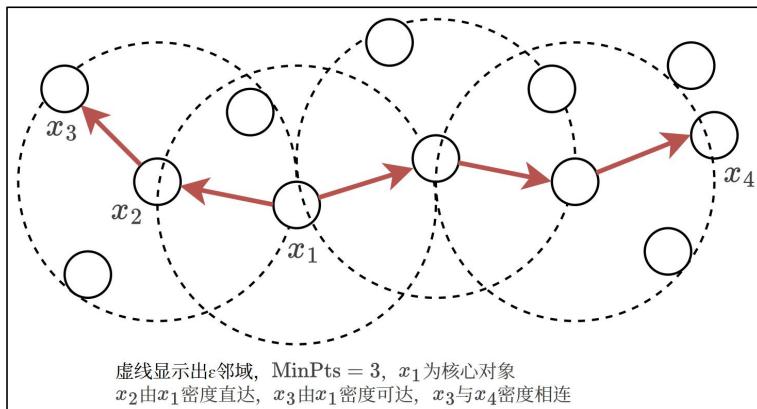
分裂聚类：开始将整个数据集视作一个整体，之后根据某种距离或相似性度量，选择一个现有的簇将其分裂成两个簇，使分裂后子簇内相似性高，子簇间差异大，如此往复直至满足停止条件。

### 3) 密度聚类

密度聚类（Density-Based Clustering）假设聚类结构能通过样本分布的紧密程度确定。通常情况下，密度聚类算法从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展簇以获得最终聚类效果。

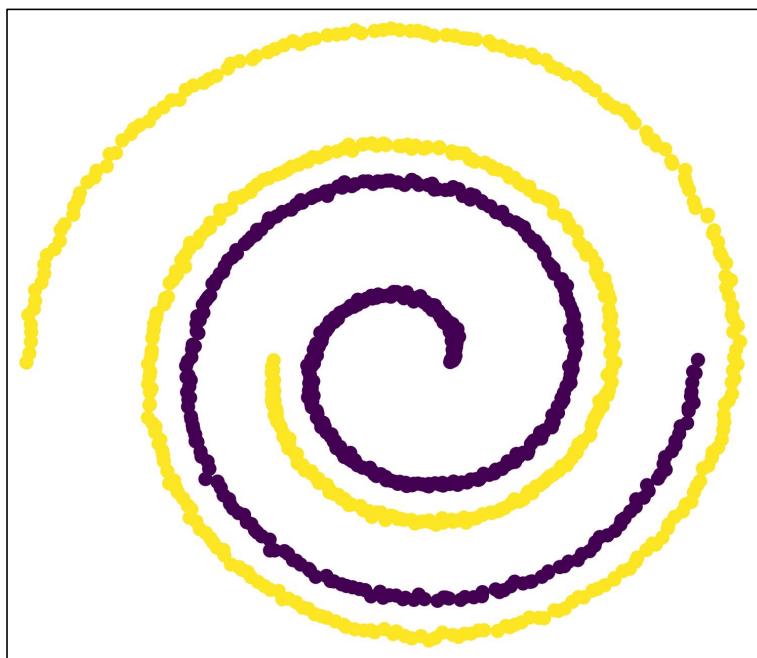
**DBSCAN** 是一种著名的密度聚类算法，基于邻域参数来刻画样本分布的紧密程度。对于给定数据集  $D = \{x_1, x_2, \dots, x_m\}$ ，定义下列概念：

- $\varepsilon$ -邻域：对于  $x_j \in D$ ，其  $\varepsilon$ -邻域包含样本集  $D$  中与  $x_j$  的距离不大于  $\varepsilon$  的样本。
- 核心对象：若  $x_j$  的  $\varepsilon$ -邻域至少包含  $MinPts$  个对象，则  $x_j$  是一个核心对象。
- 密度直达：若  $x_j$  位于  $x_i$  的  $\varepsilon$ -邻域中，且  $x_i$  是核心对象，则称  $x_j$  由  $x_i$  密度直达。
- 密度可达：对  $x_i$  和  $x_j$ ，若存在样本序列  $p_1, p_2, \dots, p_n$ ，其中  $p_1 = x_i$ ,  $p_n = x_j$ ，且  $p_{i+1}$  由  $p_i$  密度直达，则称  $x_j$  由  $x_i$  密度可达。
- 密度相连：对  $x_i$  和  $x_j$ ，存在  $x_k$  使得  $x_i$  和  $x_j$  均由  $x_k$  密度可达，则称  $x_j$  由  $x_i$  密度相连。
- 噪声点：不属于任何簇的点，既不是核心对象也不在核心对象邻域内。



基于这些概念, DBSCAN 将簇定义为由密度可达关系导出的最大密度相连样本集合。

DBSCAN 先根据邻域参数 ( $\epsilon$ 、 $MinPts$ ) 找出所有核心对象, 再以任一核心对象为出发点找出由其密度可达的样本生成一个簇, 直到所有核心对象均被访问过为止。



密度聚类能识别任意形状的簇, 可以自动识别并排除噪声点。但 $\epsilon$ 、 $MinPts$ 的选择对密度聚类结果影响较大, 且密度聚类难以适应密度变化较大的数据集。

### 8.1.3 K-means API 使用

```
kmeans = KMeans(n_clusters=3)
# n_clusters: 指定 K 的值
kmeans.fit(X) # 训练
kmeans.predict(X) # 预测
kmeans.fit_predict(X) # 训练并预测
```

示例代码:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

更多 Java -大数据 -前端 -python 人工智能资料下载, 可百度访问: 尚硅谷官网

```
from sklearn.datasets import make_blobs

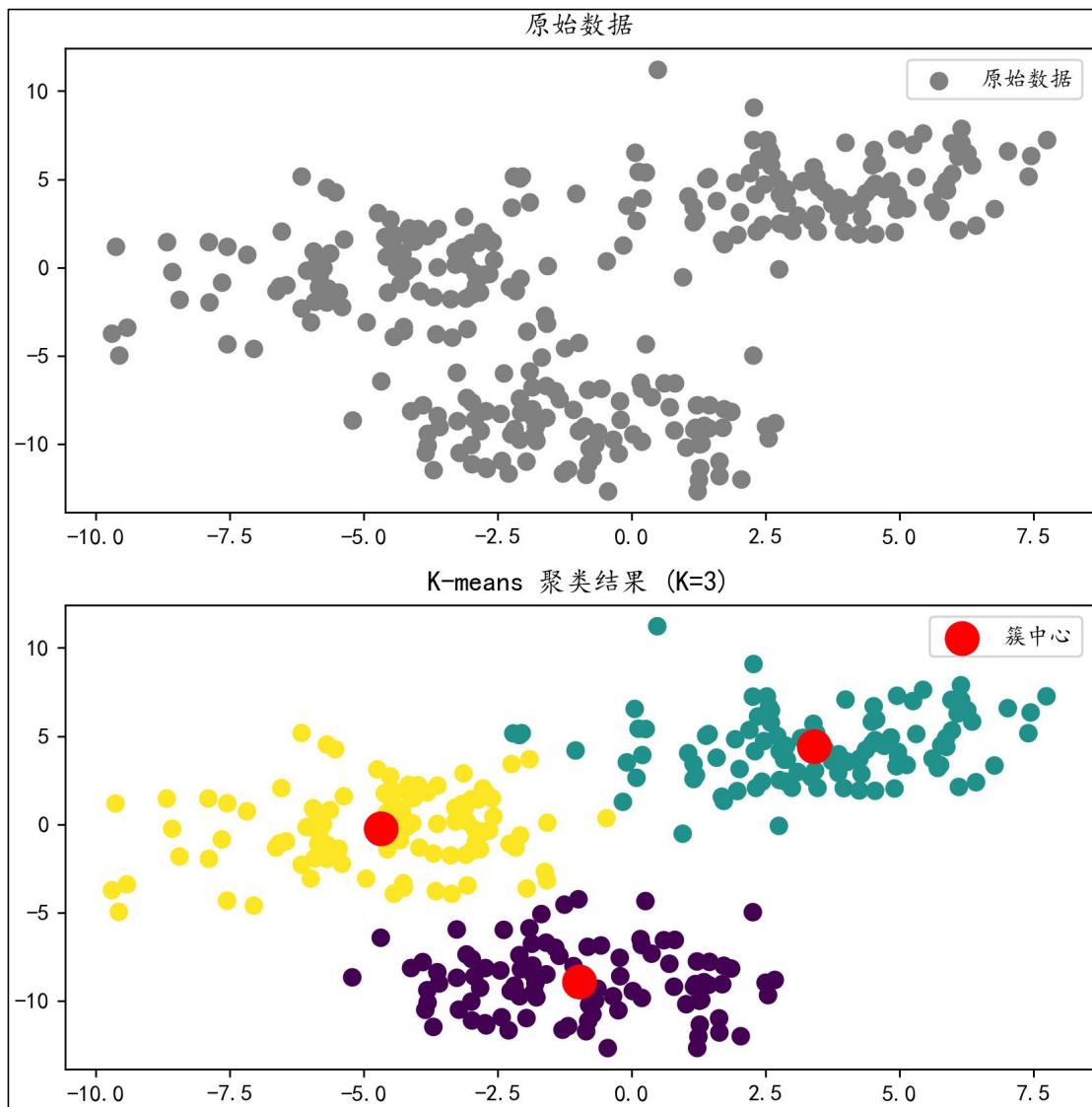
plt.rcParams["font.sans-serif"] = ["KaiTi"]
plt.rcParams["axes.unicode_minus"] = False

# 使用 make_blobs 生成 3 个簇，每个簇 100 个点
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=2)

fig, ax = plt.subplots(2, figsize=(8, 8))
ax[0].scatter(X[:, 0], X[:, 1], s=50, c="gray", label="原始数据")
ax[0].set_title("原始数据")
ax[0].legend()

# 使用 K-Means 聚类
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X) # 预测每个点的簇标签
centers = kmeans.cluster_centers_ # 获取簇中心

ax[1].scatter(X[:, 0], X[:, 1], s=50, c=y_kmeans)
ax[1].scatter(centers[:, 0], centers[:, 1], s=200, c="red", marker="o",
label="簇中心")
ax[1].set_title("K-means 聚类结果 (K=3)")
ax[1].legend()
plt.show()
```



#### 8.1.4 聚类模型评估（了解）

由于聚类任务没有预定义的标签（不像监督学习有真实类别可供比较），所以需要依赖聚类结果和原始数据来衡量模型的好坏，主要关注簇内的紧凑性和簇间的分离性。

##### 1) 轮廓系数（Silhouette Coefficient）

计算每个样本到同簇其他样本的平均距离（内聚度 $a_i$ ）和到最近其他簇样本的平均距离（分离度 $b_i$ ），综合评价聚类紧密度和分离度。

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \in [-1, 1]$$

$s_i$ 的值越接近1，聚类效果越好。总体轮廓系数是所有 $s_i$ 的平均值。

##### 2) 簇内平方和（Within-Cluster Sum of Squares）

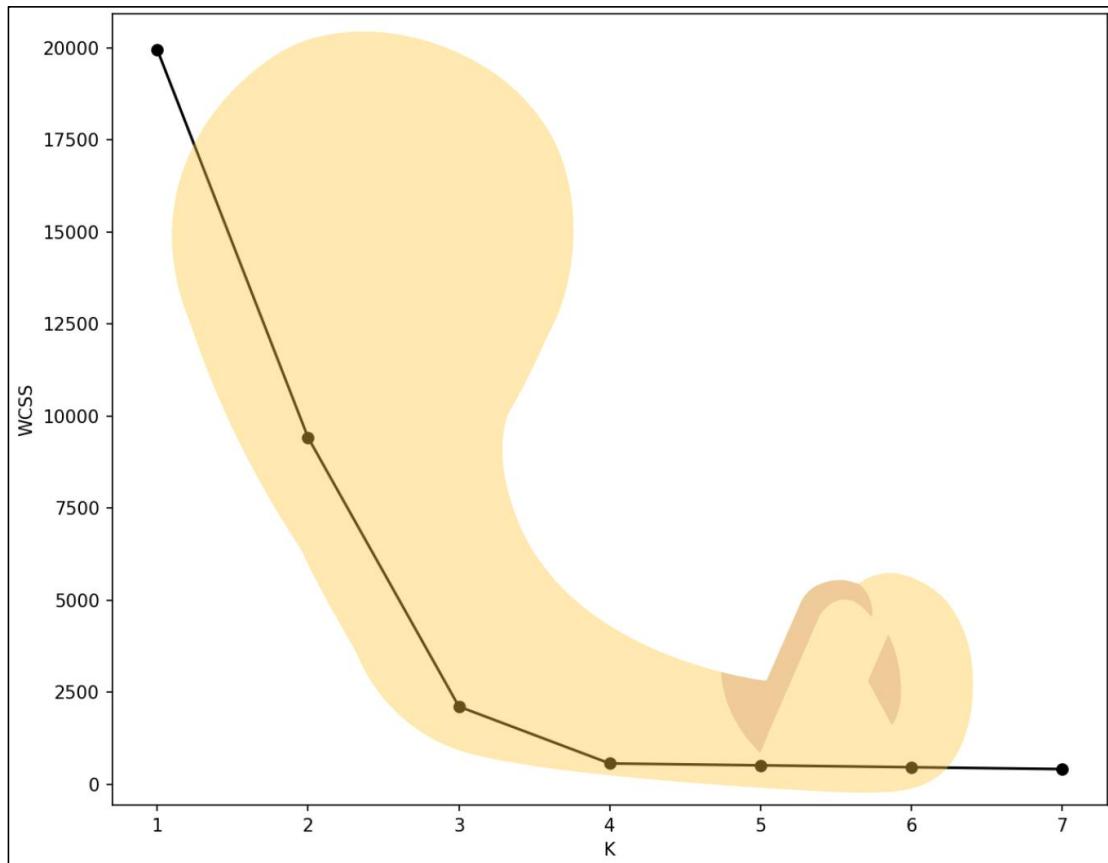
衡量簇内数据点到簇中心的总距离平方和，常用于 K-means。

$$WCSS = \sum_{k=1}^K \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

其中  $\mu_k$  是第  $k$  个簇的中心。

### 3) 肘部法

肘部法用于确定最佳簇数  $K$ ，在使用 K-means 时非常常见，它通过绘制簇数  $K$  和某个聚类质量指标（通常是簇内平方和）的关系曲线，找到一个拐点或“肘部”，即增加簇数带来的收益显著减少的点，这个点通常被认为是最佳的  $K$  值。



### 4) CH 指数 (Calinski-Harabasz Index)

簇间和簇内分散度的比值，也称方差比准则：

$$CH = \frac{\frac{BCSS}{K-1}}{\frac{WCSS}{N-K}}$$

$$BCSS = \sum_{k=1}^K n_k \|\mu_k - \mu\|^2$$

$$WCSS = \sum_{k=1}^K \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

$BCSS$ : 簇间平方和,  $n_k$ 是第 $k$ 个簇的样本数,  $\mu_k$ 为第 $k$ 个簇的中心,  $\mu$ 是所有样本的中心。

$WCSS$ : 簇内平方和。

## 5) 聚类评估 API 使用

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score, calinski_harabasz_score

plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False

# 使用 make_blobs 生成 3 个簇, 每个簇 100 个点
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=5)
# 使用 K-Means 聚类
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X) # 预测每个点的簇标签
centers = kmeans.cluster_centers_ # 获取簇中心
plt.scatter(X[:, 0], X[:, 1], s=50, c=y_kmeans, cmap="viridis")
plt.scatter(centers[:, 0], centers[:, 1], s=200, c="red", marker="*", label="簇中心")
plt.legend()
print("簇内平方和: ", kmeans.inertia_)
print("轮廓系数: ", silhouette_score(X, y_kmeans))
print("CH 指数: ", calinski_harabasz_score(X, y_kmeans))
plt.show()
```

## 8.2 降维（了解）

### 8.2.1 奇异值分解

#### 1) 奇异值分解简介

奇异值分解 (Singular Value Decomposition, SVD) 是一种矩阵因子分解方法, 用于将矩阵分解为更简单的形式, 从而揭示数据的内在结构和特性。通过保留最大的几个奇异值及其对应的奇异向量, 可以近似重构原始矩阵, 减少数据维度, 同时保留主要信息。主成分分析, 潜在语义分析等都用到了奇异值分解。

矩阵的奇异值分解是指将一个非零的实矩阵  $A \in R^{n \times p}$  表示为三个矩阵的乘积 (因子分解) 的形式:

$$A = U\Sigma V^T$$

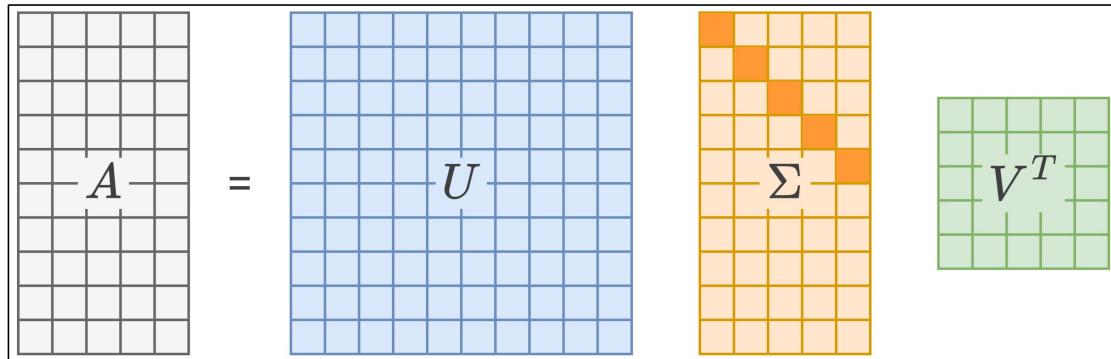
$U$ 是 $n$ 阶正交矩阵:  $UU^T = I$

$V$ 是 $p$ 阶正交矩阵:  $VV^T = I$

$\Sigma$ 是由降序排列的非负的对角元素组成的 $n \times p$ 矩形对角阵:

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p), \quad \sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0, \quad p = \min(n, p)$$

$U\Sigma V^T$ 称为矩阵 $A$ 的奇异值分解,  $\sigma_i$ 称为矩阵 $A$ 的奇异值,  $U$ 的列向量称为左奇异向量,  $V$ 的列向量称为右奇异向量。任一实矩阵一定存在奇异值分解, 且奇异值分解不唯一。



例:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

它的奇异值分解由 $U\Sigma V^T$ 给出,  $U$ 、 $\Sigma$ 、 $V^T$ 分别为:

$$U = \begin{bmatrix} 0 & 0 & \sqrt{0.2} & 0 & \sqrt{0.8} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{0.8} & 0 & -\sqrt{0.2} \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad V^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

## 2) 奇异值分解算法

矩阵 $A$ 的奇异值分解可以通过求对称矩阵 $A^TA$ 的特征值和特征向量得到。 $A^TA$ 的单位化特征向量构成正交矩阵 $V$ 的列;  $A^TA$ 的特征值的平方根为奇异值 $\sigma_i$ , 对其由大到小排列作为对角线元素构成对角矩阵 $\Sigma$ ; 求正奇异值对应的左奇异向量, 再求扩充的 $A^T$ 的标准正交基, 构成正交矩阵 $U$ 的列。具体过程如下:

首先求 $A^TA$ 的特征值和特征向量:

$$(A^TA - \lambda I)x = 0$$

得到特征值并将其由大到小排序:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p \geq 0$$

将特征值带入方程求出对应的特征向量，并将特征向量单位化，得到单位特征向量构成正交矩阵 $V$ ：

$$V = [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \dots \quad \mathbf{v}_p]$$

特征值的平方根构成 $n \times p$ 矩形对角阵 $\Sigma$ ：

$$\Sigma = \text{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \dots, \sqrt{\lambda_p})$$

对 $A$ 的正奇异值计算 $U$ 的列向量， $A$ 的秩为 $r$ ：

$$\mathbf{u}_j = \frac{1}{\sigma_j} A \mathbf{v}_j, \quad j = 1, 2, \dots, r$$

$$U_1 = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \dots \quad \mathbf{u}_r]$$

若 $n > r$ ，则需要补充额外的正交向量使 $U$ 成为 $n \times n$ 矩阵。求 $A^T$ 的零空间的一组标准正交基 $\{\mathbf{u}_{r+1}, \mathbf{u}_{r+2}, \dots, \mathbf{u}_n\}$ ：

$$U_2 = [\mathbf{u}_{r+1} \quad \mathbf{u}_{r+2} \quad \dots \quad \mathbf{u}_n]$$

$$U = [U_1 \quad U_2]$$

以求 $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$ 的奇异值分解为例：

$$A^T A = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

$$(A^T A - \lambda I)x = 0$$

得到齐次线性方程组：

$$\begin{cases} (5 - \lambda)x_1 + 5x_2 = 0 \\ 5x_1 + (5 - \lambda)x_2 = 0 \end{cases}$$

该方程组有非零解的充要条件是：

$$\begin{vmatrix} 5 - \lambda & 5 \\ 5 & 5 - \lambda \end{vmatrix} = 0$$

$$\lambda^2 - 10\lambda = 0$$

解的 $\lambda_1 = 10, \lambda_2 = 0$ ，代入线性方程组，得到对应的单位向量：

$$\mathbf{v}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 1 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

奇异值  $\sigma_1 = \sqrt{\lambda_1} = \sqrt{10}$ ,  $\sigma_2 = \sqrt{\lambda_2} = 0$ , 构造对角矩阵:

$$\Sigma = \begin{bmatrix} \sqrt{10} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

基于  $A$  的正奇异值计算得到列向量  $\mathbf{u}_1$ :

$$\mathbf{u}_1 = \frac{1}{\sigma_1} A \mathbf{v}_1 = \frac{1}{\sqrt{10}} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \\ 0 \end{bmatrix}$$

列向量  $\mathbf{u}_2, \mathbf{u}_3$  是  $A^T$  的零空间  $N(A^T)$  的一组标准正交基, 为此求解下面线性方程组:

$$A^T \mathbf{x} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned} x_1 + 2x_2 + 0x_3 &= 0 \\ x_1 &= -2x_2 \end{aligned}$$

分别取  $(x_2, x_3)$  为  $(1, 0)$  和  $(0, 1)$  得到  $N(A^T)$  的基:

$$\begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

正交基为:

$$\mathbf{u}_2 = \begin{bmatrix} -\frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \\ 0 \end{bmatrix}, \quad \mathbf{u}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

构造正交矩阵  $U$ :

$$U = \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

最终得到矩阵  $A$  的奇异值分解:

$$A = U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

上述过程只是为了说明计算的过程，实际应用中不会使用此算法。实际应用中虽然也会求解 $A^T A$ 的特征值，但不直接计算 $A^T A$ 。

### 3) 紧凑奇异值分解与截断奇异值分解

$A = U\Sigma V^T$ 又称矩阵的完全奇异值分解，实际常用的是奇异值分解的紧凑形式和截断形式。紧凑奇异值分解是与原始矩阵等秩的奇异值分解，截断奇异值分解是比原始矩阵低秩的奇异值分解。

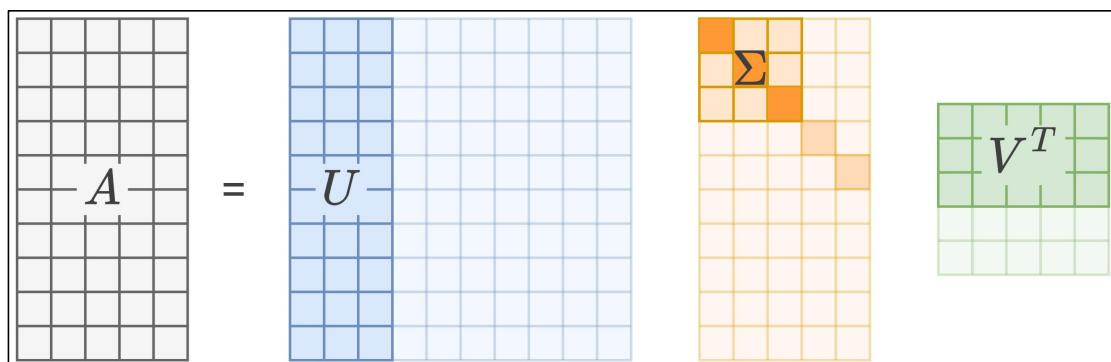
紧凑奇异值  $A = U_r \Sigma_r V_r^T$ ,  $r$  为矩阵  $A$  的秩。 $U_r \in R^{n \times r}$  取  $U$  的前  $r$  列， $\Sigma_r$  是  $r$  阶对角矩阵， $V_r \in R^{p \times r}$  取  $V$  的前  $r$  列。

例如：

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}, \quad r = 3$$

$$U_r = \begin{bmatrix} 0 & 0 & \sqrt{0.2} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sqrt{0.8} \end{bmatrix}, \quad \Sigma_r = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & \sqrt{5} \end{bmatrix}, \quad V_r^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

截断奇异值  $A \approx U_k \Sigma_k V_k^T$  只取最大的  $k$  个奇异值对应的部分。 $U_k \in R^{n \times k}$  取  $U$  的前  $k$  列， $\Sigma_k$  取  $\Sigma$  的前  $k$  个对角线元素， $V_k \in R^{p \times k}$  取  $V$  的前  $k$  列。



例如：

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}, \quad r = 2$$

$$U_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix}, \quad V_2^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A_2 = U_2 \Sigma_2 V_2^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

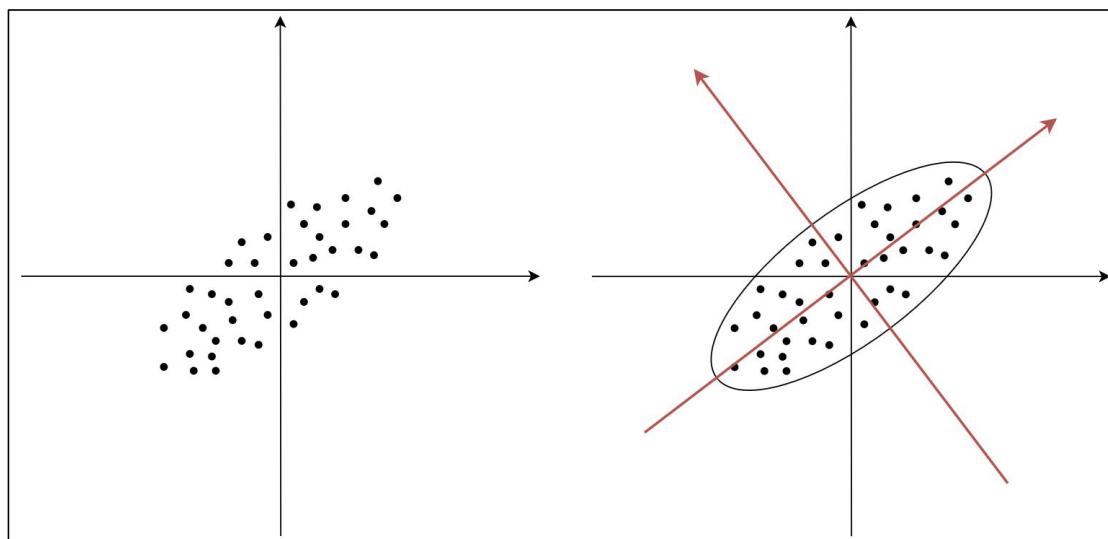
## 8.2.2 主成分分析

### 1) 主成分分析简介

主成分分析（Principal Component Analysis, PCA）是一种常用的无监督学习方法，旨在找到数据中“最重要的方向”，即方差最大的方向，并用这些方向重新表达数据。

在主成分分析过程中，首先将数据的每个特征规范化为平均值为 0 方差为 1，以消除不同特征之间量纲的差异，再使用正交变换把线性相关的原始数据转换为线性无关的新数据（主成分）。主成分彼此正交并且能够最大化地保留原始数据的方差信息。主成分分析主要用于降维和发现数据的基本结构。

主成分分析可直观解释为对数据所在的原始坐标系进行旋转变换，将数据投影到新坐标系的坐标轴上，新坐标系的第一坐标轴、第二坐标轴等分别表示第一主成分、第二主成分等。数据在每一轴上的坐标值的平方表示相应变量的方差，并且这个坐标系是所有可能的新坐标系中，坐标轴上的方差的和最大的。



在数据总体上进行的主成分分析称为总体主成分分析，在有限样本上进行的主成分分析称为样本主成分分析。在实际问题中，通常需要在观测数据上进行主成分分析，也就是样本主成分分析。

传统的主成分分析通过协方差矩阵或相关矩阵的特征值分解进行。

协方差矩阵描述变量之间的方差和协方差，适用于未标准化的数据；

相关矩阵描述变量之间的标准化相关性，适用于标准化的数据。

特征值分解的结果给出了主成分的方向（特征向量）和每个主成分的方差（特征值）。

现在常用的方法是通过奇异值分解进行主成分分析。

## 2) 相关矩阵特征值分解实现主成分分析

给定的样本数据每行为一个样本，每列为一个特征，对其进行规范化使其每列均值为0 方差为1，得到 $X$ 。对于标准化后的数据，其相关矩阵等于协方差矩阵。 $X$ 的相关矩阵：

$$R = \frac{1}{n-1} X^T X \in R^{p \times p}$$

求解特征方程 $|R - \lambda I| = 0$ ，得到 $p$ 个特征值 $\lambda_1, \lambda_2, \dots, \lambda_p$ ，降序排列作为主成分的方差。

将解得的特征值代入特征方程求出对应的单位特征向量 $v_1, v_2, \dots, v_p$ ，作为主成分方向。

根据特征值计算每个主成分的方差贡献率

$$\text{方差贡献率} = \frac{\lambda_i}{\sum_{i=1}^p \lambda_i}, \quad \text{累计方差贡献率} = \frac{\sum_{j=1}^k \lambda_j}{\sum_{i=1}^p \lambda_i}$$

根据累计方差贡献率选择前 $k$ 个主成分（如选择累计方差贡献率大于90%的前 $k$ 个主成分），取 $V$ 的前 $k$ 列构成 $V_k$ 。

将数据投影到主成分方向，得到主成分 $Y = XV_k$ 。

下面是一个主成分分析的例子：

给定样本数据 $X$ 。当使用样本均值替代总体均值时，偏差平方和会系统性低估总体方差，

因此使用样本标准差 $\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$ 代替总体标准差， $n-1$ 提供无偏估计：

$$X = \begin{bmatrix} 2 & 3 & 5 \\ 4 & 6 & 10 \\ 1 & 2 & 3 \\ 5 & 7 & 11 \\ 3 & 4 & 6 \end{bmatrix} \xrightarrow{\text{规范化}} X = \begin{bmatrix} -0.6325 & -0.6751 & -0.5898 \\ 0.6325 & 0.7716 & 0.8847 \\ -1.2649 & -1.1574 & -1.1795 \\ -1.2649 & 1.2538 & 1.1795 \\ 0.0 & -0.1929 & -0.2949 \end{bmatrix}$$

计算相关矩阵，对于标准化后的数据，其相关矩阵等于协方差矩阵：

$$R = \frac{X^T X}{n-1} = \begin{bmatrix} 1.0 & 0.9912 & 0.9791 \\ 0.9912 & 1.0 & 0.9954 \\ 0.9791 & 0.9954 & 1.0 \end{bmatrix}$$

对 $R$ 进行特征值分解，得到特征值和特征向量：

$$\lambda_1 = 2.9772, \lambda_2 = 0.0021, \lambda_3 = 0.0002$$

$$\boldsymbol{v}_1 = \begin{bmatrix} -0.5760 \\ -0.5792 \\ -0.5768 \end{bmatrix}, \boldsymbol{v}_2 = \begin{bmatrix} -0.7530 \\ 0.1013 \\ 0.6502 \end{bmatrix}, \boldsymbol{v}_3 = \begin{bmatrix} 0.3182 \\ -0.8089 \\ 0.4945 \end{bmatrix}$$

选择前两个作为主成分方向进行投影，得到第一、第二主成分：

$$Y = XV$$

$$= \begin{bmatrix} -0.6325 & -0.6751 & -0.5898 \\ 0.6325 & 0.7716 & 0.8847 \\ -1.2649 & -1.1574 & -1.1795 \\ -1.2649 & 1.2538 & 1.1795 \\ 0.0 & -0.1929 & -0.2949 \end{bmatrix} \begin{bmatrix} -0.5760 & -0.7530 \\ -0.5792 & 0.1013 \\ -0.5768 & 0.6502 \end{bmatrix}$$

$$= \begin{bmatrix} 1.0955 & 0.0244 \\ -1.3215 & 0.1771 \\ 2.0794 & 0.0683 \\ -2.1352 & -0.0585 \\ 0.2818 & -0.2113 \end{bmatrix}$$

### 3) 奇异值分解实现主成分分析

通过奇异值分解实现主成分分析可以避免计算协方差矩阵或相关矩阵，可以提高计算效率。

计算过程如下：

对规范化后的样本矩阵 $X$ 进行奇异值分解 $X = U\Sigma V^T$ ， $V$ 的列向量就是 $X$ 的主成分的方向，主成分矩阵 $Y = XV$ 。 $\Sigma$ 则与协方差矩阵的特征值相关。每个主成分的方差贡献率为 $\frac{\sigma_i^2}{\sum_{i=1}^p \sigma_i^2}$ ，用于决定保留多少主成分。

数学验证如下：

给定的样本矩阵每行为一个样本，每列为一个特征，对其进行规范化使其每列均值为0方差为1，得到 $X$ 。 $X$ 的协方差矩阵为：

$$S_X = \frac{X^T X}{n-1}$$

代入奇异值分解 $X = U\Sigma V^T$ ，因为 $U$ 是正交矩阵， $U^T U = I$ ：

$$\begin{aligned} S_X &= \frac{X^T X}{n-1} \\ &= \frac{(U\Sigma V^T)^T (U\Sigma V^T)}{n-1} \\ &= \frac{V\Sigma^T U^T U\Sigma V^T}{n-1} \\ &= V \frac{\Sigma^T \Sigma}{n-1} V^T \\ &= V \Lambda V^T \end{aligned}$$

这正是协方差矩阵的特征值分解形式，其中 $V$ 是特征向量， $\lambda_i = \frac{\sigma_i^2}{n-1}$ 是特征值。