第 3 章
机器学习

在数据驱动的时代，机器学习作为人工智能的核心技术，正深刻改变着各行各业的决策方式与发展模式。从农业生产中作物产量的精准预测，到工业制造中设备故障的提前预警，从金融领域的风险评估，到医疗行业的疾病诊断，机器学习通过从数据中自动学习规律、挖掘隐藏模式，实现了对复杂问题的高效解决。

本章系统梳理机器学习的基础理论与核心方法，涵盖从定义、发展历程到具体算法的完整知识体系。首先介绍机器学习的基本概念与流程，帮助读者建立对这一领域的整体认知；然后重点讲解回归、分类、聚类等三大核心任务，深入剖析线性回归、多项式回归、神经网络、支持向量机、决策树、随机森林、K-Means 等经典算法的原理与应用；最后通过实践案例，展示特征工程、回归、分类等技术在实际场景中的落地方法。无论读者是初学者还是希望深入理解机器学习的学习者，本章都将为其提供从理论到实践的全面指导，助力其掌握用机器学习解决实际问题的核心能力。

通过本章的学习，读者可以了解（或掌握）：

- **机器学习的定义、核心思想、发展历程及主要类型。**
- **机器学习系统的基本组成与机器学习的完整流程。**
- **回归分析的原理及实际案例应用。**
- **神经网络、支持向量机、决策树、随机森林等算法的基本原理与适用场景。**
- **聚类分析的概念与核心算法。**
- **特征工程的基本方法。**

3.1 机器学习概述

3.1.1 定义与核心思想

机器学习（Machine Learning，ML）是人工智能的一个分支，指通过算法和统计模型，使计算机系统能够从数据中自动学习规律，并利用这些规律对未知数据进行预测或决策，而无须显式编程。

机器学习的本质是让机器具备“从经验中学习”的能力，这里的“经验”通常以“数据”形式呈现，如图像、文本、语音等。其关注的是开发能够通过经验（数据）自动提升任务表现的程序。

机器学习的核心思想可以从四个维度进行理解。

1. 从数据中学习规律

摒弃传统编程逻辑：传统编程是通过编写明确的规则来解决问题，即通过“人类编写规则→机器执行”的流程解决问题。例如，手动编写识别猫狗的规则：“尖耳朵+长尾巴=猫”并让机器执行。而机器学习则是让计算机从数据中自动发现模式并构建模型，以应对复杂多变的任务。例如，让机器从大量猫狗图片中自主学习特征，通过数据发现“猫的耳朵更尖、脸型更圆”等模式。

数据即“训练素材”：数据的质量和数量直接影响学习效果。例如，用 10 万张图片训练的图像识别模型，准确率通常高于用 1000 张图片训练的模型。

2. 算法构建学习模型

模型是规律的数学表达：算法通过对数据的学习，生成一个能够拟合数据规律的数学模型。例如，线性回归算法用公式 $y=ax+b$ 拟合房价与面积的关系，其中 a 和 b 是通过数据学习得到的参数。

算法的偏好决定学习方向：不同算法对数据规律有不同的假设（如决策树倾向于学习分层规则，神经网络擅长捕捉复杂的非线性关系），类似人类用不同方法总结经验（如列表法、思维导图法）。

3. 优化与迭代

损失函数量化学习误差：用损失函数（如均方误差、交叉熵）衡量模型预测与真实数据的差距。例如，预测房价时，用损失函数计算预测值与实际价格的差值的平方和。

迭代优化调整模型参数：通过梯度下降等方法，不断调整模型参数（如线性回归中的 a 和 b），使损失函数最小，类似学生通过回顾错题并调整解题思路以提高成绩。

4. 泛化能力实现未知预测

从特殊到一般的归纳推理：模型在训练数据中学习到的规律需能推广到未见过的数据中。例如，用 10 万张猫狗图片训练的模型，需能准确识别第（10 万+1）张新图片。

避免“过拟合”与“欠拟合”。

过拟合：模型过度记忆训练数据的细节（如把“某只猫的白色毛发”当作所有猫的特征），导致新数据识别失败。

欠拟合：模型未能捕捉数据的本质规律（如仅用“是否有尾巴”区分猫狗），使预测准确率低。

3.1.2 机器学习系统的基本组成

一个完整的机器学习系统如同精密协作的“智能工厂”，通常由多个关键组件协同工作。其核心组件可分为数据层、算法层、模型层、应用层。

1. 数据层

在数据处理流程中，数据采集是源头环节。数据来源丰富多样，涵盖传感器、数据库和网络爬虫等。传感器中的摄像头、麦克风可采集图像、音频等数据；数据库提供结构化数据；网络爬虫则负责获取非结构化的文本与图像。

通常需要对采集到的数据进行预处理。数据清洗会剔除缺失值、异常值，例如，在房价数据中，去除“1 亿元/平方米”这样明显错误的数据，保证数据质量；数据转换能将非结构化数据转变为结构化数据，如对文本进行分词处理，将图像转化为像素矩阵；标准化和归一化操作则能统一数据尺度，避免因数值范围差异影响后续模型的训练和分析，比如，把身高数据的单位统一为“米”。

经过预处理的数据需要被存储与管理。对于海量数据，常采用分布式存储技术（如 Hadoop）以实现高效存储和处理。同时，数据版本控制至关重要，它能确保训练数据的可追溯性，方便后续审查和优化。此外，我们还会将数据划分为训练集、验证集和测试集，常见的划分比例为 70% : 15% : 15%，用于模型的训练、调整和评估，从而保障模型的可靠性和准确性。

2. 算法层

机器学习的核心在于学习算法的选择与应用，其中涵盖监督学习、无监督学习和强化学习三大类算法，它们各自适用于不同的任务场景。

监督学习基于已有标注数据进行训练，如线性回归能通过分析房屋面积、地段等因素预测连续的房价数值；决策树则常用于分类，通过构建树形决策模型，可精准识别邮件是否为垃圾邮件。

无监督学习用于处理无标注数据，旨在挖掘数据内在结构和规律。K-Means 聚类算法能依据用户行为数据的相似性，将用户划分成不同群组，便于企业进行精准营销；主成分分析（PCA）则专注于降维，通过提炼关键特征，压缩数据维度，在减少数据存储空间和计算成本的同时，避免信息的大量丢失。

强化学习依靠智能体与环境的交互来学习，以 Q-learning 为例，机器人在执行任务时，通过环境给予的奖励或惩罚反馈，不断调整自身动作策略，逐步掌握最优行为模式，实现高效完成任务的目标。

在模型训练过程中，优化算法发挥着重要作用。梯度下降及其变种（如 Adam 优化器）通过不断迭代更新模型参数，寻找使损失函数最小的参数组合，以提升模型的预测精度和泛化能力，确保模型在实际应用中稳定可靠。

为了更便捷地实现这些算法，开发者常借助各类开源算法库与框架。Scikit-learn 作为传统机器学习的得力工具，提供了丰富且易于调用的算法接口；TensorFlow 和 PyTorch 则在深度学习领域占据主导地位，凭借强大的计算能力和灵活的网络搭建功能，助力研究人员和开发者构建复杂的深度学习模型，加速科研与应用落地。

3．模型层

模型层是承载知识、实现预测与决策的核心载体。模型选择是承上启下的关键步骤，需要依据任务类型来选择合适的算法。

模型构建与训练是赋予模型能力的重要阶段。首先是架构设计，不同的任务类型适配不同的模型结构，例如，在图像识别领域，卷积神经网络（CNN）因其独特的卷积层和池化层设计，能够自动提取图像特征，成为主流选择；而在处理文本序列时，循环神经网络（RNN）及其变体（如 LSTM、GRU）可以捕捉文本中的时序依赖关系，从而高效处理文本数据。确定架构后，训练过程会在强大的计算资源（如 GPU 或 TPU）上进行，通过不断用训练数据迭代优化模型参数，模型会逐步学习到数据中的规律与特征。

模型构建完成并非终点，还需对模型进行全面的评估与优化。在模型评估阶段，不同类型的任务对应不同的评估指标，分类任务常用准确率、精确率、召回率等指标衡量模型的分类效果，而回归任务则使用均方误差来评估模型预测值与真实值之间的偏差。若评估结果未达预期，就需要运用优化手段对模型进行改进，超参数搜索通过调整学习率、神经网络层数等超参数，找到模型性能的最优解；正则化技术（如 L1、L2 正则化）则能有效防止模型过拟合，提升模型在新数据上的泛化能力。

4．应用层

应用层作为机器学习价值的输出端，主要通过预测、决策服务以及人机交互界面实现其功能。在预测、决策服务方面，实时推理能够将训练好的模型部署为 API 接口，例如，电商推荐系统便能利用此功能实时返回契合用户偏好的商品列表；离线分析则侧重于批量处理历史数据，银行常常借助这种方式，运用模型对贷款申请人的风险进行批量评估。而人机交互界面一方面可以将模型的预测结果进行可视化呈现，比如用图表直观展示疾病预测概率；另一方面还能将模型集成到业务系统中，自动驾驶汽车的控制中枢就是一个典型例子。

3.1.3　发展历程

1．萌芽期（20 世纪 50 至 70 年代）：算法理论奠基

20 世纪 50 至 70 年代是机器学习的萌芽期，这一时期的研究以符号主义为主导，关注逻辑推理与人工规则设计，但由于计算能力的限制，技术发展较为缓慢。

1950 年，艾伦·图灵（Alan Turing）在其里程碑论文《计算机器与智能》中提出了著名的“图灵测试”，从哲学层面探讨了机器能否具备智能的问题，为人工智能和机器学习奠定了思想基础。

1957 年，弗兰克·罗森布拉特（Frank Rosenblatt）发明了感知机（Perceptron），这是首个能够实现二分类任务的线性模型，也被视为现代神经网络的雏形。然而，1960 年，马文·明斯基（Marvin Minsky）在著作《感知机》中尖锐指出，感知机无法解决线性不可分问题（如

异或运算），这一论述导致神经网络研究陷入低谷，引发了人工智能领域的第一次“寒冬”。

尽管神经网络研究受挫，但概率统计方法开始崭露头角。1969 年，朱迪亚·珀尔（Judea Pearl）提出了贝叶斯网络，将概率论引入机器学习，为不确定性推理（如医疗诊断、风险评估）提供了新的数学工具。

这一时期的研究虽然受限于计算能力和理论瓶颈，但为后续机器学习的发展奠定了重要的算法和理论基础，神经网络、概率模型等关键思想的雏形均在此时期形成。

2. 经典发展期（20 世纪 80 年代至 2000 年）：统计学习崛起

20 世纪 80 年代至 2000 年是机器学习的经典发展期，这一时期的核心特征是统计学习理论走向成熟，各类算法百花齐放，研究重点聚焦于小数据场景下的模型优化，多个领域取得突破性进展。

20 世纪 80 年代初，决策树算法（ID3、C4.5）问世，其利用信息增益构建树形分类模型，成为早期机器学习的重要工具（如鸢尾花分类）。与此同时，反向传播（Back Propagation，BP）算法被重新发现，成功解决了多层神经网络的参数优化问题，使得 BP 神经网络在语音识别等任务中崭露头角，为深度学习的后续发展埋下伏笔。

20 世纪 90 年代，弗拉基米尔·瓦普尼克（Vapnik）提出支持向量机（SVM），结合核技巧（Kernel Trick）高效处理高维数据分类问题，并在手写数字识别任务上超越传统算法。这一时期，统计学习理论体系正式形成，为机器学习模型的泛化能力提供了坚实的数学基础。此外，集成学习方法（如随机森林、Adaboost）兴起，其通过组合多个弱分类器显著提升预测精度，成为数据竞赛中的常胜将军。同时，强化学习开始崭露锋芒，1992 年，IBM 的 TD-Gammon 在西洋跳棋中击败人类冠军，展示了强化学习在博弈领域的潜力，为后来的 AlphaGo 奠定了基础。

进入 21 世纪，概率图模型（如隐马尔可夫模型（HMM）、条件随机场（CRF））在自然语言处理（NLP）领域占据主导地位，广泛应用于词性标注、语音识别等任务。

这一时期，机器学习从早期的符号逻辑转向数据驱动的统计方法，形成了包括决策树、SVM、神经网络、概率模型等经典算法体系，并在模式识别、NLP、游戏 AI 等领域取得显著成果，为后续大数据时代的深度学习浪潮铺平了道路。

3. 深度学习复兴期（2000 年至今）：大数据与算力驱动

21 世纪 10 年代，深度学习（Deep Learning）迎来爆发式发展，大数据与算力成为核心驱动力，多层神经网络突破了算力与数据瓶颈。这一阶段，多层神经网络在计算机视觉、NLP 等感知任务上达到甚至超越人类水平，彻底改变了人工智能的发展轨迹。

2012 年，AlexNet 在 ImageNet 图像识别竞赛中夺冠，将 Top-5 错误率从 26%大幅降至 15%，宣告深度学习正式崛起。AlexNet 的成功得益于卷积神经网络（CNN）的应用，其利用局部感知和权值共享减少参数，特别适合图像特征提取，同时 GPU 并行计算技术也让训练速度相比 CPU 提升 10 倍以上。

2013 年，Hinton 团队提出“深度学习”概念，进一步推动多层神经网络研究。其中，循环神经网络（RNN）及其变种 LSTM/GRU 有效解决了序列建模问题，在机器翻译、语音识别等领域被广泛应用；生成对抗网络（GAN）则完成了由随机噪声生成逼真人脸等图像生成任务，甚至能够创作艺术画作。

2017 年，Vaswani 等人提出 Transformer 架构，其核心是引入自注意力（Self-Attention）机制，这一创新彻底重构了 NLP 任务的技术路径。基于 Transformer 架构的大规模预训练语言模型（如 BERT、GPT 等）相继涌现，在文本分类、问答系统、语义理解等多项 NLP 任务上不断刷新性能纪录。Transformer 架构具备显著的长序列建模优势，不仅克服了传统 RNN 模型存在的梯度消失问题，还通过并行计算机制大幅提升了模型训练与推理效率，为 NLP 领域的技术突破奠定了重要基础。

步入 20 世纪 20 年代，深度学习迎来新突破。大规模预训练语言模型如 GPT-3（具有 1750 亿量级参数）实现少样本学习，DALL • E2 能生成图文关联内容；多模态学习兴起，CLIP（Contrastive Language-Image Pre-training）模型联合训练图像与文本特征，实现跨模态理解；强化学习与深度学习结合诞生了 AlphaFold2，成功解决了生物学领域预测蛋白质结构这一 50 年难题。

深度学习的复兴标志着人工智能从“手工特征设计”迈向“数据驱动表征学习”，其成功依赖于三大支柱：大数据、强算力、创新架构。未来，随着模型规模扩大和跨模态能力增强，人工智能正逐步从感知智能迈向认知智能，重塑各行各业。

3.1.4 机器学习的主要类型

机器学习的主要类型包括监督学习、无监督学习、半监督学习、强化学习和迁移学习等。

1. 监督学习

监督学习作为机器学习的重要分支，是一种基于标注数据进行预测的方法。其核心思想在于利用“输入–输出”标注训练模型，赋予模型对新输入数据进行预测的能力，适用于存在明确目标变量（如分类标签、数值结果等）的场景。

在监督学习的算法体系中，分类算法与回归算法占据关键地位。分类算法方面，逻辑回归常用于解决二分类问题，如垃圾邮件识别；支持向量机能够通过超平面划分高维数据，可实现手写数字分类；随机森林则通过集成多棵决策树，提升分类精度，在癌症类型判断中发挥作用。回归算法方面，线性回归通过拟合输入与连续值输出的线性关系，助力房价预测；梯度提升树（GBDT/XGBoost）则通过迭代优化残差，降低预测误差，用于股票价格趋势分析。

凭借强大的预测能力，监督学习在实际场景中被广泛应用。无论是标注图像中的物体类别实现图像识别，将语音转为文字完成语音识别，还是预测用户偏好并构建推荐系统，都体现着监督学习的价值与意义。

2. 无监督学习

无监督学习是一种致力于探索数据内在结构的机器学习方法，其核心优势在于无须标注数据，通过挖掘数据的分布模式或关联性来实现聚类、降维等操作。这种学习方式适用于数据缺乏标签、需要挖掘潜在规律的场景，如用户分群、特征提取等。

在无监督学习的算法体系中，聚类算法、降维算法和关联规则学习是三大核心方向。聚类算法中，K-Means 算法将数据划分为 K 个簇，常用于电商用户分群；DBSCAN 算法则基于密度识别进行聚类，能够发现任意形状的簇，在异常点检测中表现突出。降维算法中，主成分分析（PCA）通过线性变换压缩数据维度，用于图片降维以保留关键特征；自编码器（Autoencoder）则借助神经网络学习数据的压缩表示，实现图像去噪等功能。关联规则学习

中的 Apriori 算法，能够发现数据中的频繁项集，经典案例如“买啤酒的顾客常买尿布”的消费关联分析。

从实际应用来看，无监督学习在市场细分（根据用户行为聚类）、异常检测（识别网络攻击）、推荐系统（基于用户相似度推荐）等领域发挥着重要作用，为缺乏标注数据的场景提供了有效的数据分析手段。

3．半监督学习

半监督学习是一种旨在平衡标注成本与模型性能的机器学习方法，其核心思想是将少量标注数据与大量未标注数据结合使用，以此降低人工标注的成本。这种学习方式适用于标注数据成本高昂的场景，比如医疗影像标注等需要专家参与的情况。

在半监督学习的实践中，主要有 3 种关键方法。

自训练（Self-Training）：通过模型对未标注数据进行预测，将预测置信度高的结果加入训练集，从而扩充标注数据。例如，在舆情分析中，先用少量标注的文本训练模型，再让模型对大量未标注的舆情数据进行预测，将可信度高的预测结果作为新的标注数据继续训练模型。

标签传播（Label Propagation）：基于数据之间的相似度来传播标注信息，适用于图结构数据的分类任务。例如，在社交网络中，已知少量用户的兴趣标签，通过用户之间的连接关系和相似度，将标签传播到未标注的用户节点上。

一致性正则化（Consistency Regularization）：通过强制模型对经过扰动的输入保持输出一致来提升性能，典型应用如对抗训练。例如，在图像分类中，对输入图像添加微小的扰动后，模型的预测结果应与原始图像的预测结果一致，从而增强模型的泛化能力。

从实际应用来看，半监督学习在医疗图像分类领域发挥着重要作用，通过结合少量专家标注的图像和大量未标注数据，其既能保证分类精度，又能降低标注成本；在语音识别任务中，也可通过该方法减少标注工时，提高模型训练效率。

4．强化学习

强化学习是一种通过试错来优化策略的机器学习模式，其核心思想是让智能体与环境进行交互，借助“动作-奖励”的反馈机制学习最优策略，最终目标是最大化长期累积奖励。这种学习模式适用于动态决策场景，如游戏博弈、机器人控制等需要连续选择动作的问题。

强化学习包含五大关键要素：执行动作的智能体（如游戏 AI）、智能体交互的对象环境（如棋盘或物理世界）、环境的当前状况（如棋盘布局）、智能体的选择动作（如下棋落子时的位置），以及动作产生的反馈奖励（如赢棋得+1，输棋得-1）。在算法层面，强化学习主要分为三类。

价值迭代算法：通过估计每个状态的价值来指导决策，典型的如 Q-learning，它会计算每个状态下不同动作的预期奖励。

策略梯度算法：直接优化策略函数，如 AlphaGo 采用的蒙特卡罗树搜索，通过迭代改进落子策略。

演员-评论家算法：结合价值迭代与策略梯度优化，如 OpenAI 的 DDPG 算法，由“演员”生成动作，由“评论家”评估动作价值，形成闭环优化。

从实际应用来看，强化学习在多个领域展现出强大能力：在游戏领域，AlphaGo 击败人类围棋冠军、Dota 2 AI 完成复杂团队协作；在机器人控制领域，其可完成无人机避障等导航任务；

在资源调度方面，其能用于数据中心能耗优化等场景，通过动态调整策略实现效率最大化。

5．迁移学习

迁移学习是一种实现知识跨领域复用的机器学习方法，其核心思想是把在源领域中学习到的知识迁移到目标领域，有效解决小数据场景下的模型训练难题。这种学习模式适用于目标领域数据稀缺的场景，如罕见病诊断、小众语言翻译等数据难以大量获取的场景。

从迁移类型来看，迁移学习主要分为两种。

同构迁移：源领域与目标领域的数据分布较为相似，例如，将在 ImageNet 数据集上预训练的模型迁移到宠物图像分类任务中，由于图像数据的基本特征相似，模型能快速适应新任务。

异构迁移：源领域与目标领域差异较大，例如，从图像识别领域迁移到视频动作识别领域，需要克服数据形态（静态图像或动态视频）和特征提取方式的差异。

在技术实现上，预训练–微调与领域自适应是两大核心方案。

预训练–微调（Pre-train+Fine-tune）：以 BERT 在自然语言处理中的应用为例，先让模型在海量文本数据中进行预训练，学习通用的语言表征，再针对具体任务（如情感分析、问答系统）用少量标注数据进行微调，大幅降低对目标领域数据量的依赖。

领域自适应（Domain Adaptation）：通过对抗训练等方式对齐源领域与目标领域的特征分布，例如，将模拟驾驶场景中训练的模型迁移到真实路况时，通过对抗网络让模型无法区分数据是来自模拟环境还是真实环境，从而提升模型在目标领域的泛化能力。

迁移学习的实际应用价值显著：在医学影像分析中，可将利用自然图像预训练的模型迁移至肺部 CT 检测，解决医疗数据标注困难的问题；在低资源语言翻译领域，可将英语–中文等常见语言翻译模型迁移至英语-斯瓦希里语等小众语言翻译任务，大幅减少对目标语言数据的依赖，推动跨语言沟通的技术落地。

3.1.5 机器学习流程

机器学习的流程是一个系统性的过程，从数据处理到模型部署，每个环节都至关重要。以下是其核心步骤及详细说明。

1．问题定义与目标明确

在机器学习项目中，问题定义与目标明确是首要环节，其核心任务在于精准确定机器学习需要解决的具体问题类型（如分类、回归、聚类等），并清晰设定相应的评估指标。以“垃圾邮件识别”为例，首先需明确该任务属于二分类问题，即判断邮件是垃圾邮件或非垃圾邮件；其次要选定评估指标，可采用准确率（正确分类的邮件占比）、召回率（正确识别出的垃圾邮件占实际垃圾邮件的比例）或 F1 分数（准确率与召回率的调和均值）。通过这些指标来衡量模型的分类性能，可确保机器学习解决方案与实际需求高度契合。

2．数据收集与预处理

在机器学习流程中，数据收集与预处理是夯实模型基础的关键环节，主要涵盖数据收集和数据预处理两大核心步骤。

1）数据收集

数据来源丰富多样，包括公开数据集（如用于手写数字识别的 MNIST、用于图像分类的

ImageNet）、企业内部业务数据，以及通过网络爬虫获取的公开网络信息等。在数据收集过程中，需严格遵循三大原则：确保数据与目标任务高度相关（如推荐系统需收集用户行为数据）、具备足够的代表性（覆盖各类场景样本），遵守数据合法性要求（注意隐私保护，避免敏感信息采集）。

2）数据预处理

数据预处理包含数据清洗、数据集成、数据转换、数据划分等步骤。

（1）数据清洗。

缺失值处理：针对数据中可能存在的缺失值，可采用删除含缺失值的样本（适用于缺失比例极低时），或通过均值、中位数填充（如年龄特征的缺失值可用样本均值填充）等方式，保证数据完整性。

异常值处理：通过统计方法（如 *Z*-score）或可视化方法（如箱线图）识别离群点，根据业务逻辑修正或删除异常值（如用户消费金额出现负数时，可核查后修正或删除）。

（2）数据集成。

当数据来自多个源头时，需进行格式统一与合并处理。例如，将不同系统导出的用户数据合并时，需统一时间戳格式（如均转换为 UTC 时间）、数值单位（如金额单位统一为“元”），确保数据一致性。

（3）数据转换。

标准化/归一化：将特征缩放到统一范围内，以避免模型受数值尺度影响。*Z*-score 标准化（基于均值和标准差）适用于服从正态分布的数据（如特征标准化后均值为 0、标准差为 1）；Min-Max 归一化则将特征线性映射到[0,1]区间，适用于区间敏感的模型（如神经网络）。

特征编码：将类别变量（如“颜色”特征的“红、绿、蓝”）转为数值形式。独热编码（One-Hot Encoding）适用于无顺序的类别（如“国籍”），可生成稀疏二进制向量；标签编码（Label Encoding）则为有序类别（如“学历”的“小学、中学、大学”）分配递增数值，提升计算效率。

（4）数据划分。

为科学评估模型性能，需将数据集按约定比例划分为训练集（用于模型参数学习）、验证集（调优超参数，避免过拟合）和测试集（最终评估模型泛化能力）。例如，1000 条样本数据中，700 条用于训练，150 条用于验证，150 条用于测试，确保模型在未知数据上的表现可被客观衡量。

3. 特征工程

特征工程的核心目标是从原始数据中提取或构造对模型更有效的特征，主要包含两大关键环节。一是特征选择，即通过卡方检验、信息增益等统计方法，或 L1 正则化等机器学习方法，筛选出对模型预测最具价值的关键特征，以降低数据维度，避免冗余信息干扰。二是特征构造，这需要结合具体的领域知识，从原始数据中生成新的有效特征，如从日期数据中提取“月份”“星期几”等隐含信息，让数据更贴合模型的学习需求，提升模型的性能表现。

4. 模型选择与训练

在机器学习流程中，模型选择与训练是将数据转化为实际预测能力的核心环节，主要包含模型选择与训练优化两大步骤。

（1）模型选择：适配问题与数据的关键决策。

模型选择需要基于问题类型（如分类、回归、聚类）匹配算法，同时综合考虑三大因素。

数据规模：小规模数据（如数千条样本）更适合逻辑回归、决策树等简单模型，避免过拟合；大规模数据（如百万级样本）则可借助深度学习（如 CNN、Transformer）挖掘复杂特征。

计算资源：深度学习模型（如 GPT）需 GPU/TPU 支持并行计算，而传统机器学习模型（如随机森林）在 CPU 环境下即可高效运行。

可解释性需求：对于医疗、金融等对决策透明度要求高的领域，倾向于选择可解释的决策树、逻辑回归（如通过特征权重分析影响因素），而非黑箱式的深度学习模型。

（2）训练优化：参数优化与算法迭代的核心过程。

在模型训练过程中，需要对模型的关键超参数进行初始化操作，这些超参数包括学习率、迭代次数、神经网络层数等，并通过训练集数据优化参数。例如，学习率决定模型更新的步长，过小会导致收敛缓慢，过大则可能导致跳过最优解；迭代次数需平衡训练精度与效率，避免过度训练导致过拟合。

在训练方式上，梯度下降算法的选择包括：

- 批量梯度下降（BGD）：每次使用全部训练数据计算梯度并更新参数，收敛过程稳定，但数据量庞大时计算开销极高（如百万级样本需遍历全量数据）。
- 随机梯度下降（SGD）：每次仅用一个样本更新参数，计算效率极高且易跳出局部最优解，但更新方向波动大，可能导致收敛过程振荡。
- 小批量梯度下降（Mini-Batch GD）：折中方案，每次选取一小批数据（如 32/64 样本）计算梯度，既减少 SGD 的波动问题，又避免 BGD 的计算负担，是实际应用中的主流选择。

例如，在银行小额贷款风险评估场景中（数据量中等、需解释性），选择逻辑回归或 XGBoost，通过小批量梯度下降训练，既能保证风险预测精度，又能通过特征重要性解释影响贷款审批的关键因素（如收入、征信记录）。在图像识别场景中（大规模数据、黑箱容忍度高），采用 CNN 模型，搭配 Mini-Batch GD 训练，利用 GPU 并行计算加速，通过调整学习率和迭代次数优化图像分类准确率。

5. 模型评估与调优

在机器学习中，模型评估与调优是保障模型性能的核心环节，利用系统性的评估方法与优化策略，可有效提升模型在实际场景中的泛化能力。

为了客观衡量模型性能，通常采用交叉验证和测试集评估两种方法。交叉验证（如 K 折交叉验证）将训练集划分为 K 份，依次以 K−1 份训练、1 份验证，循环 K 次后取平均指标，有效避免过拟合。测试集评估则要求严格隔离测试数据，仅在最终评估时使用，以检验模型的真实泛化能力。

评估指标因任务类型而异，如分类任务关注准确率、精确率、召回率、F1 分数以及 ROC AUC 曲线，回归任务常用均方误差（MSE）、平均绝对误差（MAE）和决定系数（R^2）。

提升机器学习模型性能的核心在于超参数优化和解决过拟合/欠拟合问题。

在超参数优化方面，主要有三种主流方法：

（1）网格搜索：通过穷举预设参数组合（如决策树的最大深度和学习率），来寻找最优

解，虽然计算成本较高，但能保证全面覆盖。

（2）随机搜索：随机采样参数组合的方式，特别适合高维参数空间，在保证效果的同时显著提高了效率。

（3）贝叶斯优化：基于概率模型预测最优参数，相比于传统方法，贝叶斯优化更加智能高效。

针对过拟合问题（即模型因过度记忆训练数据噪声而丧失泛化能力），可以通过多种方法进行改善。例如，引入 L1/L2 正则化惩罚项来约束模型复杂度，采用早停机制在训练过程中提前终止迭代，或者使用集成学习方法（如随机森林）通过模型多样性来抑制过拟合。

而对于欠拟合问题（模型未能充分捕捉数据内在规律），则需要采取相反的策略：通过特征工程增加特征复杂度（如特征交叉组合），选用更强大的模型架构（如用神经网络替代简单线性模型），或者适当降低正则化强度来释放模型的学习能力。这些方法的合理运用能够显著提升模型在实际应用中的表现。

通过系统地评估与调优，模型能够在准确性与泛化性之间达到平衡，从而在实际应用中发挥最佳性能。

6. 模型部署、监控与迭代

模型部署是将训练好的模型转化为实际可应用形式的关键环节。这一过程需要把经过评估和调优的模型，集成到不同的应用场景中：可以通过 Flask 或 Django 等框架将模型部署为 Web 服务，以 API 接口的形式提供预测功能，方便前端系统或其他服务调用；也可以将模型嵌入移动应用中，借助 TensorFlow Lite、PyTorch Mobile 等工具，使模型在手机等移动设备上运行，实现离线预测；还可以将模型部署到华为云、AWS SageMaker 等云平台上，利用云服务的计算和存储资源，以及对大规模数据实时推理和处理的支持，满足高并发、高可用的业务需求。

模型部署的核心是让训练好的模型从开发环境走向实际应用，为业务提供智能化的决策支持。

模型监控与迭代是保障其长期有效运行的关键环节。实时监控需持续跟踪模型在实际业务中的性能表现，如预测准确率是否下降、推理延迟是否超标等，一旦发现异常，需及时排查数据漂移（如输入数据分布变化）或概念漂移（如业务规则变更）等问题。而持续迭代则要求根据新增数据重新训练模型或动态调整模型参数，使模型能够持续适应变化的业务场景与需求，确保机器学习系统始终保持最佳性能状态。

3.2 回归分析

3.2.1 回归分析概述

回归分析是统计学与机器学习领域用于探究变量间数量依赖关系的核心方法，其本质是通过建立数学模型 $y \approx f(X)$，刻画因变量（目标变量，通常用 y 表示）对一个或多个自变量（解释变量，用 $X = x_1, x_2, \cdots, x_n$ 表示）的响应规律，通过最小化预测误差（损失函数）来优化模型参数，从而实现对连续型结果的预测与因果关系解释。

例如，通过降雨量、气温、施肥量预测农作物产量；基于用户年龄、收入、消费习惯预

估其信用卡消费额度；利用房屋面积、地段、房龄等变量预测房价。

1. 回归分析模型的基本形式

假设因变量与自变量之间存在函数关系：

$$y = f(x_1, x_2, \cdots, x_n) + \varepsilon$$

其中：

（1）f为确定性函数（如线性函数、多项式函数等），描述变量间的规律。

（2）ε为随机误差项（不可观测的随机因素，如测量误差、未纳入模型的变量影响），通常假设$\varepsilon \sim N(0,\sigma^2)$（正态分布）。

2. 回归分析的主要类型

1）线性回归

线性回归假设因变量与自变量之间存在线性关系，模型形式为$y=wx+b$。线性回归的应用场景包括房价预测（房屋面积、地段等线性影响价格）、销售额预测（广告投入、客流量等线性关联销量）等。

其局限性在于无法拟合复杂非线性关系，对异常值敏感。

2）多项式回归

多项式回归通过添加自变量的高次项（如x^2, x^3）将线性回归模型扩展为非线性回归模型，以拟合非线性关系。多项式回归的应用场景包括化学反应速率随温度变化的曲线拟合（可能呈二次或三次关系）、人口增长预测（早期呈线性增长，后期可能因资源限制呈多项式衰减）等。

其局限性在于高次项易导致过拟合，需结合正则化或交叉验证来控制复杂度。

3）岭回归/Lasso 回归

岭回归/Lasso 回归通过加入 L2/L1 正则化解决共线性问题。其应用场景包括金融风险建模（如多经济指标存在相关性时，用岭回归稳定系数）、基因表达分析（Lasso 可筛选关键基因特征，剔除冗余变量）等。

4）逻辑回归

逻辑回归虽然名为回归，实为分类算法（预测概率），用于预测二分类或多分类结果（如“是/否”“良性/恶性”）。逻辑回归通过 sigmoid 函数将线性组合$wx+b$映射到(0,1)区间，输出属于正类的概率。逻辑回归的应用场景包括信用卡违约预测（“会违约”或“不会违约”）、疾病诊断（“患病”或“健康”）等。

5）非线性回归

非线性回归不假设变量间具有线性或多项式关系，通过非线性函数或算法拟合复杂规律，常见类型包括：

（1）决策树回归：通过递归划分特征空间，构建树状结构预测连续值（如 XGBoost、随机森林回归）。

（2）支持向量机回归（SVR）：利用核函数（如径向基函数（RBF））将数据映射到高维空间，拟合非线性关系。

（3）神经网络回归：通过多层神经元网络（如全连接网络、CNN）学习任意复杂的非线性映射，适用于图像、文本等非结构化数据。

非线性回归的应用场景包括股票价格波动预测（非线性时间序列）、自动驾驶中的障碍

物距离估计（基于传感器数据的非线性建模）等。

回归分析作为预测建模的基础工具，既可以直接用于实际问题，也可作为更复杂模型的组成部分。理解回归分析是掌握机器学习的重要基础。

3.2.2 线性回归

线性回归的核心思想是通过拟合最佳线性方程（即回归线）来最小化预测值与真实值之间的误差。

1. 一元线性回归模型

一元线性回归模型也称为简单线性回归模型，模型形式为：

$$y = ax + b + \varepsilon$$

其中，y 是因变量，x 是自变量，a 是权重系数，b 是纵截距，ε 是误差项，ε 服从正态分布，均值为 0，方差为 σ^2。

模型中 a 和 b 称为回归系数，误差项 ε 的存在主要是为了平衡等号两边的值，是模型无法解释的部分。

一元线性回归研究一个自变量与一个因变量之间的关系。例如，研究房屋面积与房价之间的关系，其中房屋面积是自变量，房价是因变量。

线性回归分析中的误差项是预测值和真实值之间的差，要想得到理想的拟合线，就必须使误差项 ε 最小。由于误差项是 y 与 $ax+b$ 的差，结果可能为正值或负值，因此使误差项 ε 最小的问题可转换为使误差平方和（SSE）最小的问题（最小二乘法的思路），误差平方和可以表示为：

$$f(a,b) = \sum_{i=1}^{n} \varepsilon^2 = \sum_{i=1}^{n} (y_i - (ax_i + b))^2$$

由于建模时自变量值和因变量值都是已知的，因此求解误差平方和最小值就是求解函数 $f(a,b)$ 的最小值，而该函数的参数就是回归系数 a 和 b。

在 sklearn（在代码中导入 Scikit-learn 时使用的包名）中，一元线性回归通过 sklearn.linear_model.LinearRegression 实现。

【例 3-1】根据职场发展规律，工作年限增长通常会伴随收入水平提升。已知某行业从业人员的工作年限数据[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]（单位：年），以及与之对应的收入数据[3500, 4200, 5600, 6300, 7800, 8900, 10200, 11500, 13200, 14800]（单位：元）。请使用 sklearn 构建并训练线性回归模型，计算模型的均方误差和决定系数，并绘制实际数据点和回归直线的可视化图表。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 已知数据
years_experience = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
salary = np.array([3500, 4200, 5600, 6300, 7800, 8900, 10200, 11500, 13200, 14800])
```

```
# 准备数据
X = years_experience.reshape(-1, 1)  # 特征矩阵（工作年限）
y = salary  # 目标变量（收入）

# 构建并训练模型
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# 模型评估
slope = model.coef_[0]  # 斜率（回归系数）
intercept = model.intercept_  # 截距
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"回归方程: 收入 = {slope:.2f} × 工作年限 + {intercept:.2f}")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")

# 可视化
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='实际数据')
plt.plot(X, y_pred, color='red', linewidth=2, label='回归直线')
plt.xlabel('工作年限(年)')
plt.ylabel('收入(元)')
plt.title('工作年限与收入的线性回归模型')
plt.legend()
plt.grid(True)
plt.show()
```

运行结果为：

```
回归方程: 收入 = 1254.55 × 工作年限 + 1700.00
均方误差(MSE): 111454.55
决定系数(R²): 0.99
```

绘制的图像如图 3-1 所示。

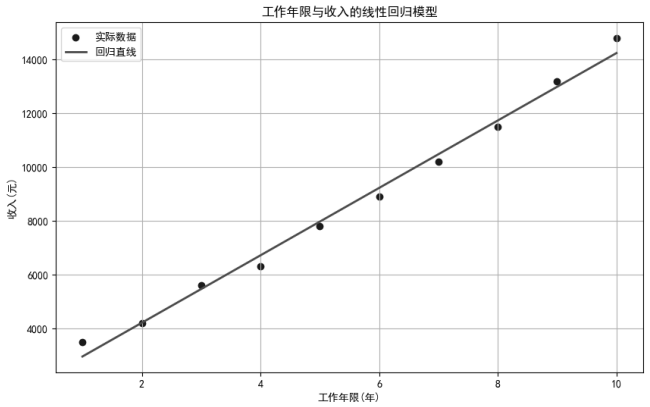

图 3-1　工作年限与收入的线性回归模型

【例 3-2】根据农业知识，在一定范围内，温度升高可能促进小麦生长。已知平均温度数据[15, 17, 19, 21, 23, 25, 27, 29, 30]（单位：℃），以及与之对应的小麦产量数据[424, 433,512, 596, 548, 588, 718, 689, 676]（单位：kg/亩）。请使用 sklearn 构建并训练线性回归模型，计算模型的均方误差和决定系数，并绘制实际数据点和回归直线的可视化图表。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据（平均温度与小麦产量）
temperatures = np.array([15, 17, 19, 21, 23, 25, 27, 29, 30])
wheat_yield =np.array([424, 433,512, 596, 548, 588, 718, 689, 676])

# 数据准备
X = temperatures.reshape(-1, 1)  # 特征矩阵（温度）
y = wheat_yield  # 目标变量（产量）

# 创建并训练线性回归模型
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# 获取模型参数
slope = model.coef_[0]  # 斜率（回归系数）
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"回归方程: 小麦产量 = {slope:.2f} × 温度 + {intercept:.2f}")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")

# 可视化
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='green', label='实际数据')
plt.plot(X, y_pred, color='red', linewidth=2, label='回归直线')
plt.xlabel('平均温度(° C)')
plt.ylabel('小麦产量(kg/亩)')
plt.title('平均温度与小麦产量的线性回归模型')
plt.legend()
plt.grid(True)
plt.show()
```

运行结果为：

```
回归方程: 小麦产量 = 18.94 × 温度 + 142.42
均方误差(MSE): 1246.02
决定系数($R^2$): 0.88
```

绘制的图像如图 3-2 所示。

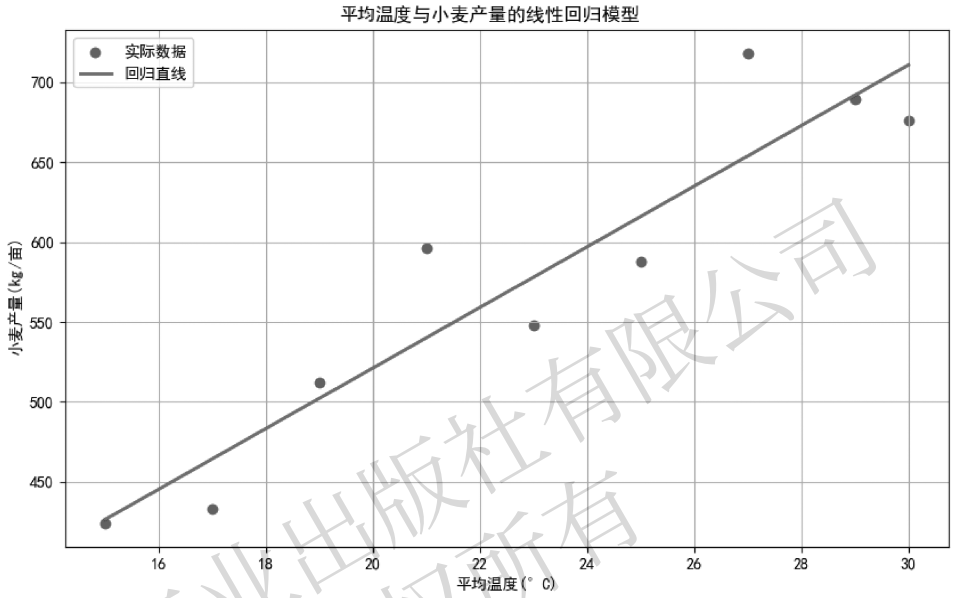

图 3-2　平均温度与小麦产量的线性回归模型

【例 3-3】根据工业生产经验，培训时间越长，工人技能越熟练，产品合格率往往越高。已知工人的平均培训时间为[5, 10, 15, 20, 25, 30, 35, 40]（单位：小时），以及与之对应的产品合格率[60, 62, 68, 74, 73, 76, 85, 87]（省略%）。请使用 sklearn 构建并训练线性回归模型，计算模型的均方误差和决定系数，并绘制实际数据点和回归直线的可视化图表。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据（培训时间与产品合格率）
training_hours = np.array([5, 10, 15, 20, 25, 30, 35, 40])
X = training_hours.reshape(-1, 1)  # 特征矩阵（培训时间）
y = [60, 62, 68, 74, 73, 76, 85, 87]  # 目标变量（合格率）

# 创建并训练线性回归模型
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
```

```
# 获取模型参数
slope = model.coef_[0]  # 斜率（回归系数）
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"回归方程: 产品合格率 = {slope:.2f} × 培训时间 + {intercept:.2f}")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")

# 可视化
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='实际数据')
plt.plot(X, y_pred, color='red', linewidth=2, label='回归直线')
plt.xlabel('培训时间(小时)')
plt.ylabel('产品合格率(%)')
plt.title('培训时间与产品合格率的线性回归模型')
plt.legend()
plt.grid(True)
plt.show()
```

运行结果为：

```
回归方程: 产品合格率 = 0.78 × 培训时间 + 55.61
均方误差(MSE): 3.55
决定系数(R²): 0.96
```

绘制的图像如图 3-3 所示。

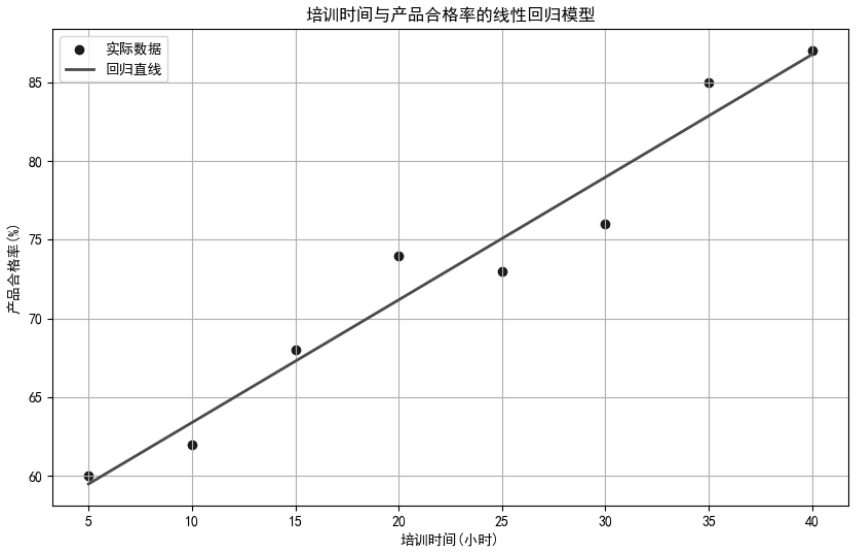

图 3-3　培训时间与产品合格率的线性回归模型

2．多元线性回归模型

对于多变量线性回归（多元线性回归），模型形式为：

$$y=\beta_0+\beta_1x_1+\beta_2x_2+\cdots+\beta_px_p+\varepsilon \tag{3.1}$$

其中，$x_1,x_2,\cdots,x_p$ 是自变量，y 是因变量，β_0 是纵截距，$\beta_1,\beta_2,\cdots,\beta_p$ 是 $x_1,x_2,\cdots,x_p$ 的系数，ε 是误差项，服从正态分布。

为了求出式（3.1），需要估计出 $\beta_1,\beta_2,\cdots,\beta_p$ 和 ε。通过 n 次观测，得到 n 组数据：

$$(y_i,x_{i1},x_{i2},\cdots,x_{ip}) \qquad i=1,2,\cdots,n \qquad (n>p)$$

结合式（3.1），可以写出如下方程组的形式：

$$\begin{cases} y_1=\beta_0+\beta_1x_{11}+\beta_2x_{12}+\cdots+\beta_px_{1p}+\varepsilon_1 \\ y_2=\beta_0+\beta_1x_{21}+\beta_2x_{22}+\cdots+\beta_px_{2p}+\varepsilon_2 \\ \vdots \\ y_n=\beta_0+\beta_1x_{n1}+\beta_2x_{n2}+\cdots+\beta_px_{np}+\varepsilon_n \end{cases} \tag{3.2}$$

其中，$\varepsilon_1,\varepsilon_2,\cdots,\varepsilon_n$ 独立同分布。

根据线性代数的知识，上式可以用矩阵形式表示为：

$$y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},\ X=\begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix},\ \beta=\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix},\ \varepsilon=\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

$$\boldsymbol{y}=\boldsymbol{X}\cdot\boldsymbol{\beta}+\boldsymbol{\varepsilon}$$

其中，$\boldsymbol{y}$ 表示因变量的向量，$\boldsymbol{X}$ 为特征矩阵，$\boldsymbol{\beta}$ 为权重向量，$\boldsymbol{\varepsilon}$ 是误差项。

自变量 $x_1,x_2,\cdots,x_p$ 是确定性变量，不是随机变量，且要求矩阵 $\boldsymbol{X}$ 的秩为 $p+1<n$，也就是说，矩阵 $\boldsymbol{X}$ 的列之间是线性无关的。

【例 3-4】对公司销售额（万元）进行预测，考虑广告投入（万元）、研发支出（万元）和员工数量三个因素的影响。已知

广告投入：[202,448,370,206,171,288,120,202,221,314]

研发支出：[100,100,165,225,245,165,60,155,55,165]

员工数量：[245,285,105,415,395,200,260,475,385,205]

销售额：[4124,6935,6513,7627,7512,6254,2699,6512,4255,6506]

通过模拟数据和多元线性回归模型，量化每个因素对销售额的影响程度（回归系数），评估模型整体预测能力（MSE 和 R^2），并通过可视化图表直观展示各因素与销售额的关系。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据(广告投入、研发支出和员工数量）
advertising = np.array([202,448,370,206,171,288,120,202,221,314])
rd_spending = np.array([100,100,165,225,245,165,60,155,55,165])
```

```
employees = np.array([245,285,105,415,395,200,260,475,385,205])

# 销售额
sales = [4124,6935,6513,7627,7512,6254,2699,6512,4255,6506]

# 数据准备
X = np.column_stack((advertising, rd_spending, employees))  # 特征矩阵
y = sales  # 目标变量

# 创建并训练多元线性回归模型
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# 获取模型参数
coef_adv, coef_rd, coef_emp = model.coef_  # 各特征的系数
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"回归方程: 销售额 = {coef_adv:.2f}×广告投入 + {coef_rd:.2f}×研发支出 + {coef_emp:.2f}×员工数量 + {intercept:.2f}")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")


# 3D 可视化（广告投入、研发支出与销售额）
plt.figure(figsize=(12, 10))
ax = plt.subplot( projection='3d')
ax.scatter(advertising, rd_spending, y, c='blue', marker='o', alpha=0.6, label='实际数据')

# 生成网格以绘制回归平面
x_surf = np.linspace(min(advertising), max(advertising), 20)
y_surf = np.linspace(min(rd_spending), max(rd_spending), 20)
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

# 计算回归平面上的点（固定员工数量为平均值）
z_surf = coef_adv * x_surf + coef_rd * y_surf + coef_emp * np.mean(employees) + intercept
ax.plot_surface(x_surf, y_surf, z_surf, color='red', alpha=0.3, label='回归平面')

ax.set_xlabel('广告投入(万元)')
ax.set_ylabel('研发支出(万元)')
ax.set_zlabel('销售额(万元)')
ax.set_title('广告投入、研发支出与销售额的关系')
plt.tight_layout()
```

```
plt.show()
```

运行结果为：

```
回归方程: 销售额 = 10.28×广告投入 + 19.89×研发支出 + 5.08×员工数量 + -1080.94
均方误差(MSE): 2317.35
决定系数(R²): 1.00
```

绘制的图像如图 3-4 所示。

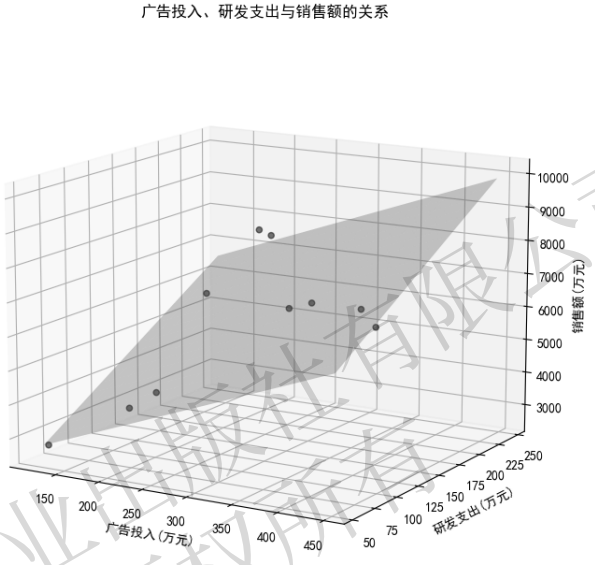

图 3-4　广告投入、研发支出与销售额的关系

【例 3-5】对小麦产量（kg/亩）进行预测，考虑温度（℃）、降雨量（mm）和施肥量（kg/亩）三个因素的影响。已知

温度：[21,18,27,29,25,22,27,19,21,24]

降雨量：[710,730,670,670,735,795,415,310,405,560]

施肥量：[55,165,105,195 ,75, 55,185,150, 50,105]

小麦产量：[441,699,664,995,603,464,809,484,328,570]

通过模拟数据和线性回归模型，量化每个因素对小麦产量的影响程度，评估模型整体预测能力（MSE 和 R^2），并通过可视化图表直观展示各因素与小麦产量的关系。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据（温度、降雨量和施肥量）
temperatures = np.array([21,18,27,29,25,22,27,19,21,24])
rainfalls = np.array([710,730,670,670,735,795,415,310,405,560])
fertilizer = np.array([55,165,105,195,75,55,185,150,50,105])
```

```
# 小麦产量
wheat_yield = [441,699,664,995,603,464,809,484,328,570]

# 数据准备
X = np.column_stack((temperatures, rainfalls, fertilizer))  # 特征矩阵
y = wheat_yield  # 目标变量

# 创建并训练多元线性回归模型
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# 获取模型参数
coef_temp, coef_rain, coef_fert = model.coef_  # 各特征的系数
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"回归方程: 小麦产量 = {coef_temp:.2f}×温度 + {coef_rain:.2f}×降雨量 + {coef_fert:.2f}×施肥量 + {intercept:.2f}")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")

# 3D 可视化（温度、降雨量与产量）
plt.figure(figsize=(12, 10))
ax = plt.subplot(projection='3d')
ax.scatter(temperatures, rainfalls, y, c='green', marker='o', alpha=0.6, label='实际数据')

# 生成网格以绘制回归平面
x_surf = np.linspace(min(temperatures), max(temperatures), 20)
y_surf = np.linspace(min(rainfalls), max(rainfalls), 20)
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

# 计算回归平面上的点
z_surf = coef_temp * x_surf + coef_rain * y_surf + coef_fert * np.mean(fertilizer) + intercept
ax.plot_surface(x_surf, y_surf, z_surf, color='red', alpha=0.3, label='回归平面')

ax.set_xlabel('温度(°C)')
ax.set_ylabel('降雨量(mm)')
ax.set_zlabel('小麦产量(kg/亩)')
ax.set_title('温度、降雨量与小麦产量的关系')
plt.tight_layout()
plt.show()
```

运行结果为：

```
回归方程: 小麦产量 = 21.08×温度 + 0.39×降雨量 + 2.80×施肥量 + -439.72
均方误差(MSE): 295.21
决定系数(R²): 0.99
```

绘制的图像如图 3-5 所示。

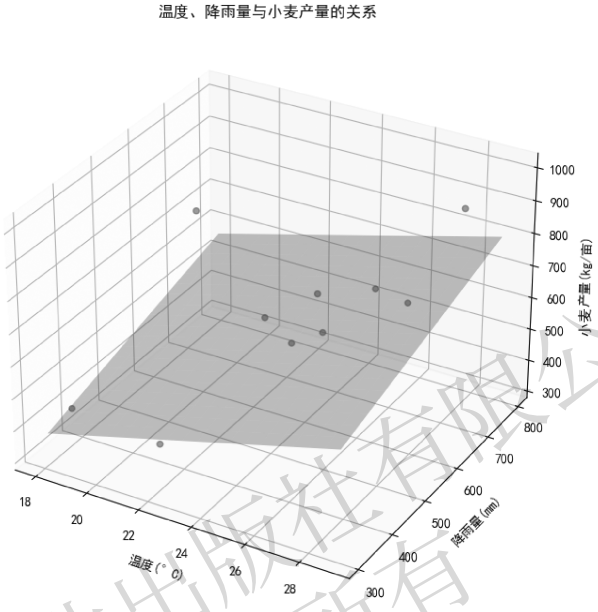

图 3-5 温度、降雨量与小麦产量的关系

【例 3-6】对化工厂产品产量（kg）进行预测，考虑反应温度（℃）、压力（MPa）、催化剂用量（kg）和反应时间（小时）四个因素的影响。已知

反应温度：[151,192,114,171,160,120,182,186,174,174]

压力：[5,15,10,5,10,10,10,10,5,5]

催化剂用量：[3,3,3,1,3,1,1,1,1,1]

反应时间：[4,6,4,4,6,4,6,6,2,6]

产品产量：[895,1636,616,994,1154,643,1283,1329,928,1047]

通过模拟数据和多元线性回归模型，量化每个因素对产品产量的影响程度，评估模型整体预测能力（MSE 和 R^2），并通过可视化图表直观展示各因素与产品产量的关系。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据(反应温度、压力范围、催化剂用量、反应时间）
temperature = np.array([151,192,114,171,160,120,182,186,174,174])
pressure = np.array([5,15,10,5,10,10,10,10,5,5])
catalyst = np.array([3,3,3,1,3,1,1,1,1,1])
time    = np.array([4,6,4,4,6,4,6,6,2,6])
# 产品产量
```

```
    yield_amount = [895,1636,616,994,1154,643,1283,1329,928,1047]

    # 数据准备
    X = np.column_stack((temperature, pressure, catalyst, time))  # 特征矩阵
    y = yield_amount  # 目标变量

    # 创建并训练多元线性回归模型
    model = LinearRegression()
    model.fit(X, y)
    y_pred = model.predict(X)

    # 获取模型参数
    coef_temp, coef_press, coef_cata, coef_time = model.coef_  # 各特征的系数
    intercept = model.intercept_  # 截距

    # 模型评估
    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # 输出结果
    print(f"回归方程：产量 = {coef_temp:.2f}×反应温度 + {coef_press:.2f}×压力 + {coef_cata:.2f}×催化剂用量 +
{coef_time:.2f}×反应时间 + {intercept:.2f}")
    print(f"均方误差(MSE): {mse:.2f}")
    print(f"决定系数(R²): {r2:.2f}")

    # 3D 可视化（温度、压力与产品产量）
    fig = plt.figure(figsize=(14, 6))
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(temperature, pressure, y, c='red', marker='o', alpha=0.6, label='实际数据')

    # 生成网格以绘制回归平面
    x_surf = np.linspace(min(temperature), max(temperature), 20)
    y_surf = np.linspace(min(pressure), max(pressure), 20)
    x_surf, y_surf = np.meshgrid(x_surf, y_surf)

    # 计算回归平面上的点（固定催化剂用量和反应时间为平均值）
    z_surf = coef_temp * x_surf + coef_press * y_surf + coef_cata * np.mean(catalyst) + coef_time * np.mean(time) + intercept
    ax.plot_surface(x_surf, y_surf, z_surf, color='blue', alpha=0.3, label='回归平面')

    ax.set_xlabel('反应温度(°C)')
    ax.set_ylabel('压力(MPa)')
    ax.set_zlabel('产品产量(kg)')
    ax.set_title('反应温度、压力与产品产量的关系')

    # 3D 可视化（催化剂用量、反应时间与产品产量）
    ax = fig.add_subplot(122, projection='3d')
    ax.scatter(catalyst, time, y, c='purple', marker='o', alpha=0.6, label='实际数据')
```

```
# 生成网格以绘制回归平面
x_surf = np.linspace(min(catalyst), max(catalyst), 20)
y_surf = np.linspace(min(time), max(time), 20)
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

# 计算回归平面上的点（固定反应温度和压力为平均值）
z_surf = coef_cata * x_surf + coef_time * y_surf + coef_temp * np.mean(temperature) + coef_press * np.mean(pressure) + intercept
ax.plot_surface(x_surf, y_surf, z_surf, color='blue', alpha=0.3, label='回归平面')

ax.set_xlabel('催化剂用量(kg)')
ax.set_ylabel('反应时间(小时)')
ax.set_zlabel('产品产量(kg)')
ax.set_title('催化剂用量、反应时间与产品产量的关系')

plt.tight_layout()
plt.show()
```

运行结果为：

```
回归方程: 产量 = 9.92×反应温度 + 29.51×压力 + 44.69×催化剂用量 + 28.65×反应时间 + -1027.75
均方误差(MSE): 361.82
决定系数($R^2$): 1.00
```

绘制的图像如图 3-6 所示。

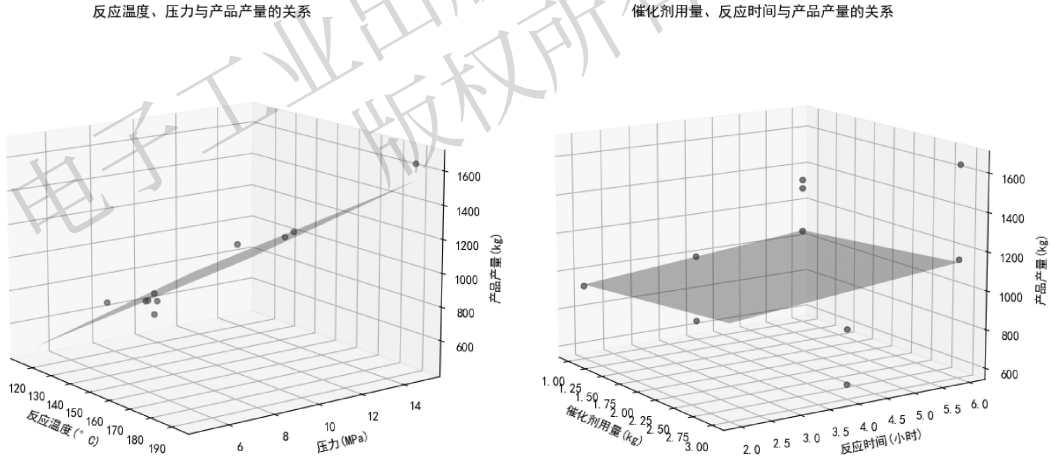

图 3-6 两组主要因素与产品产量的关系

3.2.3 多项式回归

多项式回归（Polynomial Regression）是线性回归的扩展，用于拟合自变量与因变量之间的非线性关系。与简单的线性回归不同，多项式回归通过引入自变量的高次项（如 x^2, x^3），提供了更为复杂的非线性模型来拟合数据。

多项式回归模型的形式为：

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \varepsilon \tag{3.3}$$

其中，x 是自变量，y 是因变量，β_0 是纵截距，$\beta_1, \beta_2, \cdots, \beta_n$ 是 $x, x^2, \cdots, x^n$ 的系数，ε 是误差项。

对于多项式回归模型，其目标是找到一组参数 $\beta_1,\beta_2,\cdots,\beta_n$，使得模型能够尽可能准确地拟合数据。通过引入高次项，使得多项式回归成为非线性模型，但模型对系数 $\beta_1,\beta_2,\cdots,\beta_n$ 而言仍是线性的，因此可通过最小二乘法求解。

多项式回归的核心思想是将非线性问题转化为线性问题，通过构造高次特征，使用线性回归的方法求解高次项的系数。

多项式回归的步骤如下。

（1）特征扩展：根据原始特征 x，生成多项式特征（$x,x^2,\cdots,x^n$）。通过简单的特征扩展，将线性模型升级为非线性模型。

（2）模型训练：对扩展后的特征应用线性回归（最小化 MSE）。

在 sklearn 中，使用 sklearn.preprocessing.PolynomialFeatures 生成多项式特征，将原始特征扩展为多项式组合，即将线性模型扩展为非线性模型，同时保留线性回归的计算效率和可解释性。

【例 3-7】已知两个样本的二维特征为[1, 2], [3, 4]，输出二维特征的多项式扩展。

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

# 原始特征矩阵（2 个样本，2 个特征）
X = np.array([[1, 2], [3, 4]])

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# 查看结果
print("原始特征:\n", X)
print("\n 扩展后的特征:\n", X_poly)
print("\n 特征名称:", poly.get_feature_names_out())
```

运行结果为：

```
原始特征:
 [[1 2]
 [3 4]]

扩展后的特征:
 [[ 1.   1.   2.   1.   2.   4.]
 [ 1.   3.   4.   9. 12. 16.]]

特征名称: ['1' 'x0' 'x1' 'x0^2' 'x0 x1' 'x1^2']
```

【例 3-8】使用多项式回归模型，拟合二次函数 $y=2x^2+3x+1$。计算模型评估指标（MSE），输出模型的系数和截距，绘制多项式回归模型的可视化结果。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# 生成模拟数据
np.random.seed(42)
x = np.linspace(-3, 3, 100).reshape(-1, 1)
y_true = 2 * x**2 + 3 * x + 1
y_noise = y_true + np.random.normal(0, 2, size=x.shape)

degree = 2
poly = PolynomialFeatures(degree=degree, include_bias=False)
x_poly = poly.fit_transform(x)   # 生成 x, x2

model = LinearRegression()
model.fit(x_poly, y_noise)

y_pred = model.predict(x_poly)
mse = mean_squared_error(y_noise, y_pred)
print(f"模型系数: {model.coef_}")
print(f"模型截距: {model.intercept_}")
print(f"MSE: {mse:.4f}")

plt.scatter(x, y_noise, label='观测数据', alpha=0.6)
plt.plot(x, y_true, 'g-', label='真实函数')
plt.plot(x, y_pred, 'r--', label=f'多项式拟合 (degree={degree})')
plt.legend()
plt.title('多项式回归')
plt.grid(True)
plt.show()
```

运行结果为：

```
模型系数: [[3.04597756 2.04072352]]
模型截距: [0.66766832]
MSE: 3.2472
```

绘制的图像如图 3-7 所示。

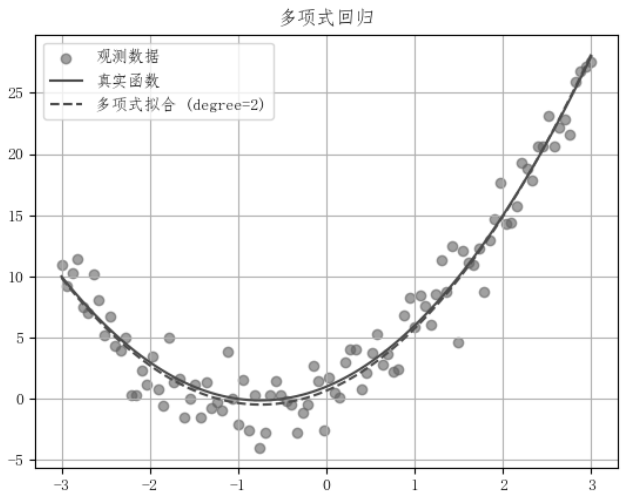

图 3-7　多项式回归模型

【例 3-9】在经济学中，研发投入占比与产出往往呈现边际收益递减的规律：适度增加研发投入可以显著提升企业市场价值（单元：百万元），但过度投入可能导致效率下降。已知

研发投入占比：[8,19,15,12,4,4,2,17,12,14,1,19,17,5,4]（省略%）

市场价值：[1925,1628,1420,2535,1912,1210,1257,1831,2039,1578,1074,2045,2869,1922,1769]

研究企业研发投入占比与市场价值之间的非线性关系，计算模型评估指标（MSE 和 R^2）。可视化展示二次多项式回归模型，并与简单线性回归模型进行对比。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据(研发投入与产出)
rd_investment = np.array([ 8,19,15,12, 4, 4, 2,17,12,14, 1,19,17, 5, 4])
market_value = [1925, 1628, 1420, 2535, 1912, 1210, 1257, 1831, 2039, 1578, 1074, 2045, 2869,1922,1769]

# 数据准备
X = rd_investment.reshape(-1, 1)  # 特征矩阵
y = market_value  # 目标变量

# 创建多项式特征 (degree=2 表示二次多项式)
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# 创建并训练线性回归模型 (拟合多项式特征)
model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)

# 获取模型参数
coef_linear, coef_quadratic = model.coef_  # 线性项和二次项系数
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(f"多项式回归方程: 市场价值 = {intercept:.2f} + {coef_linear:.2f}×研发投入 + {coef_quadratic:.2f}×(研发投入)^2")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R^2): {r2:.2f}")

# 绘制原始数据散点图
plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='blue', alpha=0.6, label='实际数据')
```

```
# 绘制多项式回归曲线
x_range = np.linspace(min(X), max(X), 100).reshape(-1, 1)
x_range_poly = poly.transform(x_range)
plt.plot(x_range, model.predict(x_range_poly), color='red', linewidth=3, label='多项式回归曲线')

# 绘制简单线性回归作为对比
linear_model = LinearRegression()
linear_model.fit(X, y)
plt.plot(x_range, linear_model.predict(x_range), color='green', linestyle='--', linewidth=2, label='简单线性回归')

plt.xlabel('研发投入占比(%)')
plt.ylabel('市场价值(百万元)')
plt.title('企业研发投入占比与市场价值的多项式回归模型')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

运行结果为：

```
多项式回归方程: 市场价值 = 1050.85 + 159.90×研发投入 + -6.14×(研发投入)^2
均方误差(MSE): 141211.66
决定系数(R²): 0.35
```

绘制的图像如图 3-8 所示。

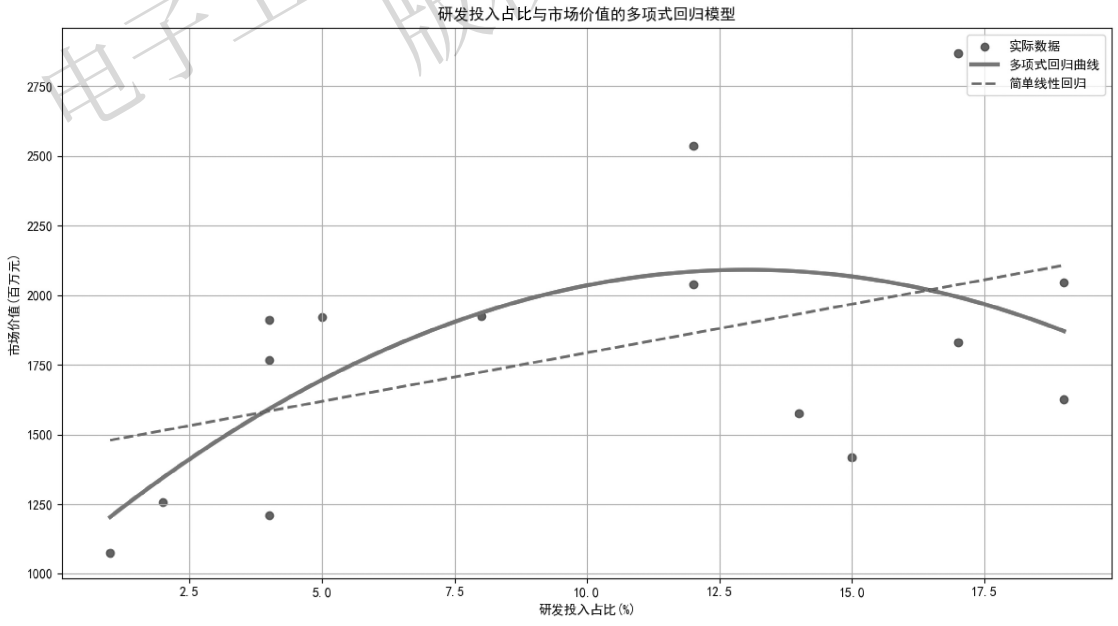

图 3-8　研发投入占比与市场价值的多项式回归模型

【例 3-10】在农业科学中，平均温度（℃）对农作物生长的影响通常呈现钟形曲线特征：平均温度过低或过高都会抑制农作物生长，且存在一个最佳平均温度区间。已知

平均温度：[18,38,31,26,10,10,7,35,26,30,6,39,34,12,11]

农作物产量：[483,167,38,643,650,298,395,241,393,123,330,385,756,610,558]（单位：kg/亩）

模拟平均温度与农作物产量之间的非线性关系，计算模型评估指标（MSE 和 R^2）。可视化展示三次多项式回归模型，并与简单线性回归模型和二次多项式回归模型进行对比。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
plt.rcParams["font.family"] = ["SimHei"]

# 模拟数据(平均温度与农作物产量)
temperatures = np.array([18,38,31,26,10,10, 7,35,26,30, 6,39,34,12,11])
crop_yield =  [483,167, 38,643,650,298,395,241,393,123,330,385,756,610, 558]

# 数据准备
X = temperatures.reshape(-1, 1)  # 特征矩阵
y = crop_yield  # 目标变量

# 创建多项式特征 (degree=3 表示三次多项式)
poly = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly.fit_transform(X)

# 创建并训练线性回归模型 (拟合多项式特征)
model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)

# 获取模型参数
coef_linear, coef_quad, coef_cubic = model.coef_  # 线性、二次和三次项系数
intercept = model.intercept_  # 截距

# 模型评估
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# 输出结果
print(
    f"多项式回归方程: 农作物产量 = {intercept:.2f} + {coef_linear:.2f}×平均温度 + {coef_quad:.2f}×(平均温度)² + {coef_
cubic:.2f}×(平均温度)³")
print(f"均方误差(MSE): {mse:.2f}")
print(f"决定系数(R²): {r2:.2f}")

# 绘制原始数据散点图
plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='green', alpha=0.6, label='实际数据')
```

```
# 绘制多项式回归曲线
x_range = np.linspace(min(X), max(X), 300).reshape(-1, 1)
x_range_poly = poly.transform(x_range)
plt.plot(x_range, model.predict(x_range_poly), color='red', linewidth=3, label='三次多项式回归')

# 绘制简单线性回归模型作为对比
linear_model = LinearRegression()
linear_model.fit(X, y)
plt.plot(x_range, linear_model.predict(x_range), color='blue', linestyle='--', linewidth=2, label='简单线性回归')

# 绘制二次多项式回归模型作为对比
poly2 = PolynomialFeatures(degree=2, include_bias=False)
X_poly2 = poly2.fit_transform(X)
model2 = LinearRegression()
model2.fit(X_poly2, y)
plt.plot(x_range, model2.predict(poly2.transform(x_range)), color='purple', linestyle='-.', linewidth=2, label='二次多项式回归')

plt.xlabel('平均温度（° C)')
plt.ylabel('农作物产量 (kg/亩)')
plt.title('平均温度与农作物产量的多项式回归模型')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

运行结果为：

```
多项式回归方程:农作物产量 = -183.01 + 116.93×平均温度 + -5.56×(平均温度)² + 0.07×(平均温度)³
均方误差(MSE): 32895.69
决定系数(R²): 0.21
```

绘制的图像如图 3-9 所示。

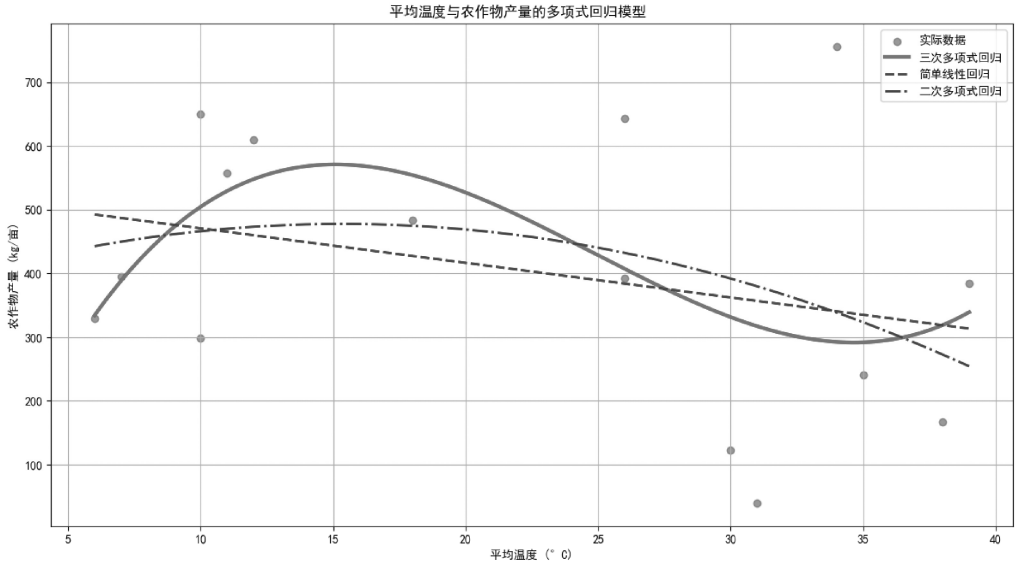

图 3-9　平均温度与农作物产量的多项式回归模型

多项式回归适用于如下场景：

（1）数据呈现非线性趋势：例如，化学反应速率与温度变化之间的指数关系，产品销量与广告投入之间的抛物线关系。

对于非线性问题，多项式回归仅适用于因变量与自变量存在多项式关系的场景，对复杂非线性关系（如正弦函数）效果有限。

（2）线性模型拟合不足：当线性回归的误差（预测值与真实值的差异）呈现系统性偏差时，多项式回归可能更合适。

3.3 分类

分类是我们认识自然和把握世界规律的重要途径。我国传统文化里，“物以类聚，人以群分”和儒家的“格物致知”等典故就蕴含了很多分类思想。从《齐民要术》的农作物分类到基因测序的生物分类，从《本草纲目》对药物的系统分类到现代医学疾病类型的诊断，从围棋的棋谱分类到电商用户画像的区分，分类思维贯穿人类文明发展的各个阶段。

机器学习的分类（Classification）任务，正是让计算机具备这种“辨物识类”的能力。

本节介绍实现分类任务的主要方法：神经网络、支持向量机、决策树、随机森林等。

3.3.1 神经网络

神经网络是模仿生物大脑神经元多层网络结构的人工智能模型，每一层神经元对上一层输入的信息进行加工，最终输出分类结果。它通过大量神经元的连接和权重调整，实现对复杂模式的学习。

1. M-P 神经元模型

人工神经网络的研究起源于 1943 年，心理学家沃伦·麦卡洛克（Warren McCulloch）和数学家沃尔特·皮茨（Walter Pitts）提出的 M-P 神经元模型，奠定了人工神经网络的理论基础，具有跨时代的重要意义。

M-P 神经元模型的关键是模仿生物神经元的信号处理机制，建立可计算的神经元数学模型，如图 3-10 所示。

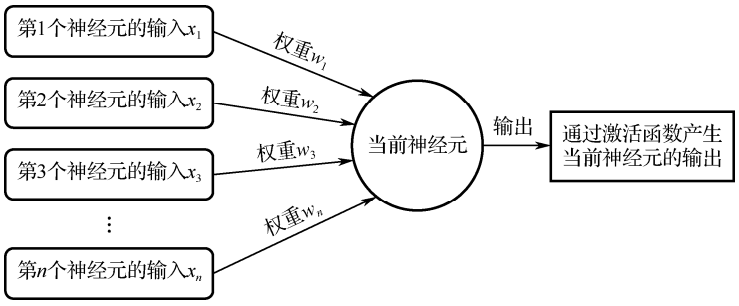

图 3-10　M-P 神经元模型

M-P 神经元模型表示：

将来自 n 个神经元的输入信号借助带有权重的连接输入当前神经元，通过激活函数产生

输出。激活函数 $y=f(s)$是某非线性函数 f 与线性函数 $s=\boldsymbol{w}\cdot\boldsymbol{x}-a$ 的复合，其中，

$$\boldsymbol{w}=(w_1,w_2,\cdots,w_n)^{\mathrm{T}}\ ,\quad \boldsymbol{x}=(x_1,x_2,\cdots,x_n)^{\mathrm{T}}\ ,\quad \boldsymbol{w}\cdot\boldsymbol{x}=\sum_{k=1}^{n}w_kx_k\ 。$$

参数含义如下：

x_k：来自第 k 个神经元输入信号的数值；

w_k：第 k 个输入信号的权重，用来表示该信号的重要性；

a：通常表示当前神经元的一个阈值；

$\sum_{k=1}^{n}w_kx_k$：表示当前神经元接收到的（带权）总输入信息量。

神经元的训练：通过调整参数 $w_1,w_2,\cdots,w_n$ 与 a 的值，使得当前神经元输出的计算结果 y 越来越接近预期值 b。详细来说，该过程是运用高等数学的最小二乘法原理等，求 $w_1,w_2,\cdots,w_n$ 与 a，使得误差函数 $e=|f(\boldsymbol{w}\cdot\boldsymbol{x}-a)-b|$ 取最小值。

我们可以把 M-P 神经元形象地比喻为一个“决策工厂”，每个输入信号就像输送到工厂的原材料，有的携带重要信息（强信号），有的无关紧要（弱信号）。工厂内部有一套严格的生产标准（阈值），当输入原材料的总量达到生产标准时，工厂就会启动生产线，输出一个成品（激发信号）；若原材料不足，则工厂保持静默。这种根据输入综合判断并决定是否“开工”的机制，正是 M-P 神经元模型处理信息的核心逻辑。

2．激活函数

常用的激活函数 $y=f(x)$主要有以下 3 种，f 的不同选择决定了当前神经元的不同输出结果。

1）阶跃函数

$$f(x)=\begin{cases}1, & x\geqslant 0\\ 0, & x<0\end{cases}$$

此时激活函数的输出结果为：

$$y=\begin{cases}1, & \text{当}\sum_{k=1}^{n}w_kx_k\geqslant a\text{时}\\ 0, & \text{当}\sum_{k=1}^{n}w_kx_k<a\text{时}\end{cases}$$

它表示：如果当前神经元接收的总信息量等于或超过了阈值 a，它就会被激活（用 1 模拟兴奋状态）；反之，神经元就会处于抑制状态（用 0 来模拟）。这与现实中生物神经元的兴奋与抑制两种状态非常吻合。然而，阶跃函数不连续、不可导的数学性质，使它无法构造出一些可计算的有效神经网络。

2）挤压函数（sigmoid 函数）

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

该函数是可导的光滑函数，数学性质优于阶跃函数，而且可用于分类，若 sigmoid 函数值大于 0.5，则检测样本大概率为正例；反之，则大概率为负例。特别地，当 x 趋于正无穷时，函数值趋于 1；当 x 趋于负无穷时，函数值趋于 0。但此函数有输出不是以 0 为中心、饱和时梯度消失等不足之处。

3）双曲正切函数（tanh 函数）

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

该函数是 sigmoid 函数的一个变体，取值范围是[−1, 1]，输出以 0 为中心。但饱和时梯度消失的缺陷仍然存在。

关于 sigmoid 函数与 tanh 函数的直观图形和更丰富的解析运算性质，读者可查看《高等数学》教材。

* 延伸阅读：为什么要使用激活函数 $y=f(x)$，而且选择非线性函数？

在本小节第一部分中我们可知，当前神经元对输入信息的参数是通过加权求和的线性运算处理的，若不使用非线性激活函数，则该层的输出结果也会是由线性叠加得到的，再作为下一层神经元的输入值，同样地，最后整个网络的输出结果也是线性的。然而实际问题往往都是非线性的，因此我们需要在输出层使用非线性激活函数，使结果适应现实的非线性空间。

3．神经网络的发展

下面介绍不同神经网络模型的产生背景与发展脉络，以开拓研究思路。

前馈神经网络（FNN）是最早期的可实际运行的人工神经网络。在 M-P 神经元模型的基础上，Rosenblatt 于 1958 年提出感知机，这是首个可学习的前馈神经网络模型，它能够处理线性可分问题，如简单的二分类任务，却无法解决非线性问题，这导致神经网络研究陷入长达十几年的“寒冬期”。直到多层感知机的提出，才打破了这一局面。

在前馈神经网络中，信息只能单向沿输入层–隐藏层–输出层移动，在网络中没有循环回路。由于前馈神经网络对于处理现实中常见的复杂图像识别、自然语言处理等非线性问题的能力十分有限，在 20 世纪后半叶人们迫切地希望改进前馈神经网络的训练方法。

1986 年，Rumelhart、Hinton 和 Williams 等人在《自然》杂志上发表的论文正式提出了 BP 神经网络。这是一种基于误差逆向传播算法训练的多层前馈神经网络，成功解决了多层感知机的训练难题，使神经网络在语音识别、图像识别等领域的应用成为可能，是目前应用最广泛的神经网络之一，也为深度学习的兴起打下基础。

构建 BP 神经网络模型的部分 Python 代码如下：

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),  # 输入层到隐藏层
    Dense(32, activation='relu'),  # 隐藏层
    Dense(1, activation='sigmoid')  # 输出层，二分类问题使用 sigmoid 激活函数
```

近半个多世纪，前馈神经网络、BP 神经网络，还有深度学习领域的卷积神经网络（CNN）、循环神经网络（RNN）、长短时记忆网络（LSTM）等模型相继被学者提出，并成功应用于图像、语音、文本等复杂数据的识别与分类问题中。近期蓬勃发展的 DeepSeek 大模型，其核心机制就是通过深度神经网络来提高信息检索的效率和准确性。

* 思考题：为什么神经网络需要采用多层结构？单层神经网络不能解决复杂问题吗？

解析：单层神经网络只能学习线性分类边界，无法处理非线性问题（如异或运算）。多层神经网络通过非线性激活函数的叠加，能够学习复杂的非线性映射，就像多层滤网能过滤不同粒度的杂质一样，多层神经网络能提取不同层次的特征。

3.3.2 支持向量机

在 3.3.1 节，我们了解到，前馈神经网络（感知机）是 20 世纪 60 年代人们处理分类问题的主要工具，但这类模型只能解决线性可分问题，对“异或（XOR）”等非线性问题几乎无能为力，促使人们探索更强大的工具来突破这一局限性。

另外，很多机器学习方法都采用经验误差最小化的设计方法训练模型，然而，当采集的数据有限时，用于训练的数据集无法完整体现分布特性，使模型的结果不尽如人意。

在这一背景下，支持向量机（Support Vector Machine, SVM）模型应运而生，它在分类算法的模型评估中有着优异的表现。

支持向量机的理论是在面向小样本机器学习问题的统计学习理论基础上建立起来的。不同于经验误差最小化的设计方法，支持向量机采用最大化分类间隔的方式来学习分类模型。

我们从相对简单的线性可分问题开始，探讨支持向量机的分类原理与数学模型。

1. 线性可分支持向量机的原理与数学模型

假定已有某 d 维特征空间上的包含 N 个样本点的训练数据集：

$$S=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\},$$

其中，$\boldsymbol{x}_i\in X=R^d$ 是第 i 个特征向量（亦称实例），$y_i\in Y=\{1,-1\}$ 是 $\boldsymbol{x}_i$ 的类标记，$(\boldsymbol{x}_i,y_i)$ 是第 i 个样本点，$i=1,2,\cdots,N$。当 $y_i=1$ 时，称相应的 $\boldsymbol{x}_i$ 为正例；当 $y_i=-1$ 时，称 $\boldsymbol{x}_i$ 为负例。

线性可分问题的支持向量机目的为：在 d 维特征空间中寻找一个超平面：

$$z=\boldsymbol{w}\cdot\boldsymbol{x}+b=\sum_{k=1}^{d}w_kx_k+b,\quad \boldsymbol{w}=(w_1,w_2,\cdots,w_d)^{\mathrm{T}}\text{ 表示法向量，}b\text{ 为截距}$$

将特征空间中的样本点以最大间隔的方式划分为两部分，法向量 $\boldsymbol{w}$ 指向的一侧是正类，另一侧是负类。该超平面相应的分类决策函数

$$f(\boldsymbol{x})=\mathrm{sign}(\boldsymbol{w}\cdot\boldsymbol{x}+b)$$

称为线性可分支持向量机，其中 sign 是符号函数。

知识链接——超平面：超平面是直线、平面在高维空间的推广。当特征空间维数 $d=1$ 时，$z=w_1x_1+b$ 表示一条直线；当 $d=2$ 时，由高等数学空间解析几何知识，$z=w_1x_1+w_2x_2+b$ 表示平面；当 $d>2$ 时，$z=\boldsymbol{w}\cdot\boldsymbol{x}+b$ 即为超平面。

线性可分支持向量机的原理如图 3-11 所示（以 $d=1$ 为例）。

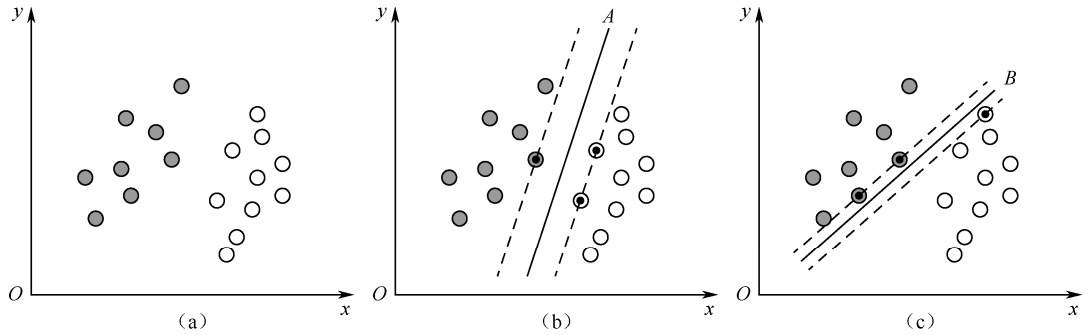

图 3-11　线性可分支持向量机的原理

在图 3-11 中，我们用实心点表示正例，空心点表示负例，能够看到，存在多条直线（例如 A, B）将两类数据分到两侧，而直线 A 的分类效果更好，因为在直观上，它能使得两类点的间隔更大。线性可分支持向量机就是在寻找一条直线 $z = w_1 x_1 + b$，在分隔两类数据点时达到最大间隔。运用高等数学向量代数的知识，可计算得到间隔为：

$$D = \frac{2}{|w_1|}$$

同理，d 维特征空间上的线性可分支持向量机可以用以下数学模型表示：

$$\max_{\boldsymbol{w},b} D = \frac{2}{\|\boldsymbol{w}\|^2}, \quad \text{其中} \|\boldsymbol{w}\|^2 = w_1^2 + w_2^2 + \cdots + w_d^2$$

$$\text{s.t.} \begin{cases} y_i = 1, & \text{当} \boldsymbol{w} \cdot \boldsymbol{x}_i + b \geqslant 1 \text{时} \\ y_i = -1, & \text{当} \boldsymbol{w} \cdot \boldsymbol{x}_i + b \leqslant -1 \text{时} \end{cases} \quad i = 1, 2, \cdots, d$$

该模型可通过高等数学中最优化理论的拉格朗日乘子法求解。

2. 非线性的支持向量机

相较于 3.3.1 节的前馈神经网络，支持向量机的一大优势在于能够用于研究非线性问题。图 3-12 所示的就是一个非线性的支持向量机应用，它无法用一条直线将图片左半部分的两类数据点分隔开，而是用了曲线（圆周）来分隔。对于此类问题，我们就需要构造一个非线性的分类器进行数据分类。

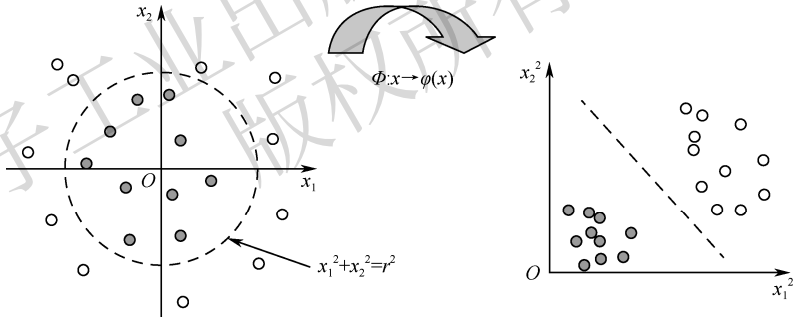

图 3-12　非线性的支持向量机应用

非线性的支持向量机的主要特点是使用核技巧（Kernel Trick）将非线性分类问题转换为线性问题来解决，可以使用以下 Python 命令：

```
from sklearn.svm import SVC
clf = SVC(kernel='rbf')                    # 径向基函数核
clf = SVC(kernel='poly')                   # 多项式函数核
clf = SVC(kernel='sigmoid')                # sigmoid 核
clf = SVC(kernel='precomputed')            # 预计算核矩阵
```

3. 支持向量机的分类参数

我们基于 Scikit-learn 1.0.2 版本（Scikit-learn 不同版本的参数默认值可能变化）对支持向量机分类算法库的常用参数进行了汇总（见表 3-1），方便读者查阅使用。

表 3-1　sklearn.svm.SVC 参数汇总

参数名称	参数说明	数据类型	取值范围
C	正则化参数，C 越大，模型对误分类的惩罚越严格；C 越小，模型可能出现更多误分类	float	正数，默认值 1.0
kernel	核函数类型，用于将输入数据映射到高维空间，解决非线性分类问题	str	'linear'、'rbf'（默认）、'poly'、'sigmoid'、'precomputed'
gamma	核函数的系数，影响决策边界的复杂度	str/float	'scale'（默认）、'auto'或正数
degree	多项式核函数（kernel = 'poly'）的次数。仅在 kernel = 'poly' 时有效，其他核函数下该参数被忽略	int	正整数，默认值 3
coef0	核函数中的常数项	float	实数，默认值 0.0
shrinking	是否使用收缩启发式算法，用于加速优化过程	bool	True（默认）或 False
tol	迭代优化的收敛阈值。当损失函数的变化小于 tol 时，算法停止迭代	float	正数，默认值 1e-3
cache_size	缓存大小（单位：MB），存储核矩阵计算结果，提升训练效率	float	正数，默认值 200.0
verbose	是否启用详细输出，用于调试或监控训练过程	int	非负整数，默认值 0
max_iter	最大迭代次数。设置为-1 时（默认），表示不限制迭代次数，直到收敛	int	整数或-1（默认）
random_state	随机数种子，用于控制随机初始化的结果（如 shrinking 策略中的随机采样），确保结果可复现	int/None/random_state	整数、None（使用默认随机数生成器）或 np.random.RandomState 对象，如 random_state = 42

参数分类说明：

- 优化相关参数：C、tol、max_iter，控制模型复杂度和优化过程的收敛条件。
- 参数调优优先级：通常优先调整 C 和 gamma（尤其是当 kernel = 'rbf'时），其次考虑 kernel。
- 核函数相关参数：kernel、gamma、degree、coef0，根据数据特性选择核函数并调整其形状。
- 计算效率相关参数：shrinking、cache_size、verbose，优化训练速度或监控过程。

3.3.3　决策树

决策树（Decision Tree）是机器学习的一种重要的分类算法，其思想是通过对实例数据特征的不断划分，构建一个树状结构，从而实现对数据的分类或回归预测。

事实上，在决策树的概念正式形成之前，人们就已经有了类似树状结构的思考方式，例如我国的易经，还有生物学家通过对生物特征的逐步判断来对生物进行分类的思想。而能够用于机器学习的成熟的决策树模型则是于 20 世纪 70 年代前后建立起来的。

1．决策树的组成

决策树由根节点、内部节点、叶节点和分支组成，如图 3-13 所示。

➢ 根节点（△）：初始状态，包含全部样本数据。

➢ 内部节点（◯）：也称决策节点，代表对某一特征的判断，其中特征表示数据的属性，如人的年龄、性别、身高、体重等。

➢ 叶节点（□）：表示最终的分类结果。

➢ 分支：从一个节点到另一个节点的连线。

决策树的根节点犹如易经所说的“易有太极”，是万象初始的混沌状态；特征划分的内部节点好像“两仪生四象，四象生八卦”的推演过程；分支延伸如卦象叠加，最终得到叶节点表示的分类结果，如图 3-14 所示。

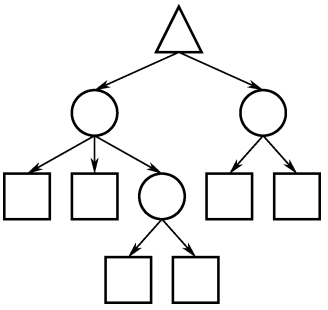

图 3-13　决策树

图 3-14　易经与决策树

2. 决策树的构建原理和步骤

决策树的构建可分为特征选择、决策树的生成、剪枝优化三个步骤。

1）特征选择

特征选择是构建决策树的第一步，需要选取对训练数据有明显分类能力的特征，特征选择的优劣会影响决策树学习能力的强弱。比如，要判断一个水果是苹果还是橘子，若以形状作为特征，就不具备明显的分类能力。

对于不同的决策树算法，特征选择有不同的依据，为此，读者需要了解概率论与信息学的一些概念。

信息熵：用于衡量数据的不确定性，熵值越大，数据的不确定性越高，计算公式为：

$$H(D)=-\sum_{k=1}^{n}p_k\ln p_k \tag{3.4}$$

其中，D 表示样本容量为 n 的数据集，p_k 表示数据集中第 k 类样本所占的比例。

条件熵：描述在已知随机变量 X 的条件下，随机变量 Y 的不确定性，计算公式为：

$$H(Y\mid X)=-\sum_{k=1}^{n}p_kH(Y\mid X=x_k)$$

其中，$H(Y\mid X=x_k)$ 是在条件 $X=x_k$ 下 Y 的条件概率的熵。

信息增益：表示划分前后信息熵的差值，信息增益越大，说明该特征对数据的划分效果

越好。特征 A 对于容量 n 的训练数据集 D 的信息增益计算公式为：

$$\mathrm{Gain}(D,A)=H(D)-H(D\mid A)$$

信息增益率：表示对信息增益的改进，体现了划分后各子节点样本数量的纯度，计算公式为：

$$\mathrm{Gain-Ratio}(D,A)=\frac{\mathrm{Gain}(D\mid A)}{H(D\mid A)}$$

基尼指数：通常在经济学中用来表述收入的不平衡性，也可用作任意不均匀分布的度量。在分类中，基尼指数刻画了数据集 D 的不纯度，定义如下：

$$\mathrm{Gain}(D)=1-\sum_{k=1}^{n}p_k^2$$

2）决策树的生成

我们首先从根节点出发，然后在每一步中选择最优的特征（与不同算法相适应）来进行划分，以确保每次划分都能最大程度地提高数据集的纯度。通过递归地生成子节点，我们不断地将数据集分割成更小的部分，直到满足停止条件。利用这种方式，我们逐步使所有数据都得到了明确的分类，即生成出一棵完整的决策树。决策树生成流程如图 3-15 所示。

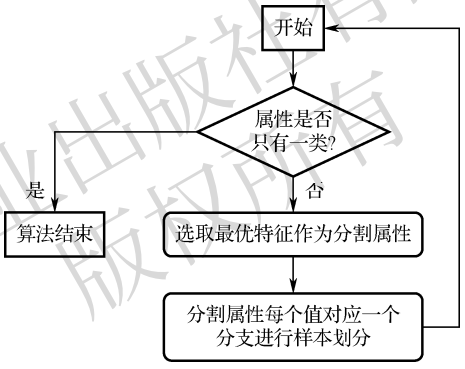

图 3-15　决策树生成流程

3）剪枝优化

递归地生成决策树后，其虽然对训练数据能进行明确的分类，但是对未知的测试数据的分类效果可能不够准确，即存在过拟合现象。

为防止模型过拟合训练数据，提高模型的泛化能力，我们需要通过预剪枝剔除冗余分支，来简化已生成的决策树。具体来说，就是通过极小化决策树的损失函数，对子树或叶节点进行剪裁，并把根节点或父节点作为新的叶节点，达到剪枝的目的。

3．决策树算法简介

1）发展历程

Ross Quinlan 在 1975 年提出的 ID3（Iterative Dichotomiser 3）算法是决策树发展史上的一个重要里程碑，它能够自动从数据中构建决策树，大大提高了决策树的实用性和效率。在 ID3 算法的基础上，Ross Quinlan 又提出了 C4.5 算法，解决了 ID3 算法在处理连续特征和缺失值时的一些问题，使决策树的构建更加合理。

其他研究团队提出了 CART（Classification and Regression Tree）算法，其不仅可应用于

分类问题，还能用于回归问题，生成的决策树是二叉树，具有更广泛的应用场景。

2）常用函数与调用方法

在 Python 中，我们可以使用 sklearn 库来实现 ID3 决策树模型。

例如，使用 DecisionTreeClassifier()函数创建决策树分类器。

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='entropy')          # 创建 ID3 决策树分类器
```

在 sklearn 库中，DecisionTreeClassifier()和 DecisionTreeRegressor()分别用于实现 CART 分类树和 CART 回归树。

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
clf_cart = DecisionTreeClassifier(criterion='gini')        # 创建 CART 分类树
reg_cart = DecisionTreeRegressor()                         # 创建 CART 回归树
```

fit()：用于训练模型。

```
clf.fit(X_train, y_train)                                  # 训练模型
```

predict()：用于进行预测。

```
y_pred = clf.predict(X_test)                               # 进行预测
```

3）算法比较

决策树算法比较如表 3-2 所示。

表 3-2　决策树算法比较

算法	特征核心度量	特点
ID3	信息增益	首次将信息论引入决策树，倾向于选择取值多的特征
C4.5	信息增益率	改进 ID3 缺陷，可处理连续特征和缺失值，用“信息增益率”平衡特征取值数量影响
CART	基尼指数	支持分类与回归，生成二叉树，结构更简洁，是随机森林等集成算法的基础

4．决策树的优点与缺点

决策树的优点与缺点见表 3-3。

表 3-3　决策树的优点与缺点

维度	优点表现	缺点表现
模型假设	无须预先假设数据的先验概率分布，非参数化特性适应多种场景	无法保证能收敛到全局最优解
数据兼容性	支持连续值、离散值、语义数据等多元类型，缺失值处理灵活	高维数据下性能可能下降，特征交互表达能力有限
可解释性	树结构与规则链直观易懂，适合业务场景的决策逻辑可视化	未剪枝时可能导致分类规则复杂，丧失解释优势
抗噪声能力	剪枝机制可抑制噪声影响，缺失值处理策略可减少对数据质量的依赖	天然倾向于拟合训练数据，未优化时易过拟合，泛化能力弱

随机森林能够较好地克服决策树的以上缺点。

3.3.4 随机森林

将决策树与其他机器学习算法相结合，会形成更强大的集成学习方法，随机森林（Random Forest）算法就是其中一种。

1. 随机森林概述

随机森林是通过 Bagging 算法与随机子空间方法，将多棵决策树组合为一体的集成学习方法。

随机森林一词中的“随机”体现为两方面：一是每棵决策树的训练样本是采用 Bagging 算法随机抽取的；二是决策树的每个节点的分裂字段是随机选择的。样本随机性与特征随机性的结合，使随机森林模型具备了很好的性能优势（见表 3-4）。

表 3-4 随机森林的性能优势

对比维度	随机森林	决策树、SVM 等传统算法
抗过拟合能力	强（集成+双重随机性）	弱（单模型易拟合噪声）
处理非线性关系	优（自动学习特征交互）	差（需手动构造交互项）
参数敏感性	低（默认参数表现良好）	高（SVM 需调参）
特征共线性影响	无（特征随机选择）	显著（系数估计不稳定）
小样本性能	良好（Bootstrap 扩充样本）	差（样本不足时方差大）
计算效率	高（并行训练）	中（单模型训练快，但调参耗时）

随机森林通过集成学习和双重随机性设计，实现了对传统机器学习算法的多维度超越，广泛应用于计算机视觉、工业制造、生物医学等领域。

当前随机森林的研究热点有以下几个方向：

- 深度随机森林：结合深度学习的层次特征提取能力；
- 在线随机森林：处理流式数据的增量学习算法；
- 因果随机森林：用于因果推断和反事实预测；
- 量子随机森林：基于量子计算的高效集成模型。

2. 随机森林的构建步骤

- 数据采样（Bagging）：从原始数据集中通过有放回采样生成多个 bootstrap 样本集，每个样本集的样本数量与原始数据集相同。
- 特征随机选择：在构建每棵决策树时，从所有特征中随机选择一部分特征作为候选划分特征。
- 单棵决策树构建：对每个 bootstrap 样本集，使用决策树算法（如 CART 等）构建决策树，节点划分时仅在随机选择的特征中选择最优特征。
- 结果集成：对于分类问题，通过多数投票法确定最终分类结果；对于回归问题，通过平均值确定最终回归结果。

随机森林构建流程如图 3-16 所示。

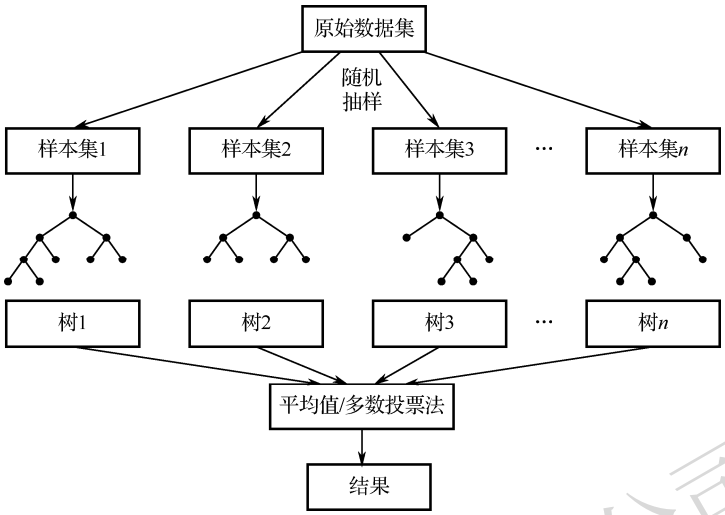

图 3-16　随机森林构建流程

思考：为什么随机森林中每棵树都要使用不同的样本和特征？如果所有树都相同会怎样？

解析：使用不同样本和特征可增加树之间的多样性，降低相关性，从而减少集成模型的方差。若所有树都相同，集成模型将退化为单棵树，无法获得“集体智慧”的优势，准确率和稳定性都会下降。

3.3.5　模型评估指标

由 3.3.1～3.3.4 节可知，机器学习分类算法多种多样，用哪些指标能够评估模型的性能好坏？这是本节介绍的内容。

1. 基本概念

我们假设在分类目标中只有正例（Postitive）和负例（Negative）两类，而分类任务会遇到实际类别和模型预测两种类别。因此模型的分类结果会有以下 4 种情况。

（1）真正例（True Positive，TP）：实际类别为正例，模型预测为正例的情况。

（2）假负例（False Negative，FN）：实际类别为正例，模型预测为负例的情况。

（3）假正例（False Positive，FP）：实际类别为负例，模型预测为正例的情况。

（4）真负例（True Negative，TN）：实际类别为负例，模型预测为负例的情况。

基于以上分析，可提出如表 3-5 所示的混淆矩阵（Confusion Matrix），也称为误差矩阵。

表 3-5　混淆矩阵

混淆矩阵		模型预测	
		正	负
实际类别	正	真正例（TP）	假负例（FN）
	负	假正例（FP）	真负例（TN）

2. 评估指标及其调用

基于混淆矩阵，我们可以提出机器学习分类模型的以下评估指标。不同评估指标从不同维度衡量模型性能，如同用多种工具检测产品质量——有的检查精度，有的检查完整性，有的评估整体稳定性。单一指标易受数据分布影响，需综合多个指标全面评估。

准确率（Accuracy）：预测正确（即将正例判定为正例，负例判定为负例）的样本数占总样本数的比例，公式为：

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} \tag{3.5}$$

准确率是最直观的分类模型评估指标，但在数据不平衡场景下存在局限性。例如，在电商平台的欺诈订单检测中，正常订单占比高达 99.9%，模型看似表现良好，实则丧失了对异常订单的识别能力。因此，准确率需结合其他指标综合评估模型性能。

错误率（Error Rate）：分类错误的样本数占总样本数的比例。由于分对和分错是互斥事件，所以有：

$$\text{Error Rate} = 1 - \text{Accuracy} = \frac{\text{FP} + \text{FN}}{\text{P} + \text{N}}$$

灵敏度（Sensitive），也称为召回率（Recall）或真正率（True Positive Rate，TPR）：所有正例中被分对的比例。计算公式为：

$$\text{Sensitive} = \text{Recall} = \text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}}$$

灵敏度（召回率或真正率）是分类模型中衡量正例预测准确性的核心指标。例如，在疾病诊断场景中，真正率反映了模型能够正确识别出的患病样本的比例。真正率越高，意味着模型捕捉正例的能力越强。

特异度（Specificity）：也称为真阴性率，表示所有负例中被分对的比例，公式为：

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{\text{TN}}{\text{N}}$$

特异度是指模型正确识别负例的能力。特异度高意味着模型在预测负例时很少出现误判。例如，在疾病诊断中，特异度高表明模型不容易将健康人误诊为患者，能有效减少误诊带来的不必要焦虑和医疗资源浪费。

精确率（Precision）：模型预测为正例的样本中，真正属于正例的比例，即表示查得准的概率，因此又称查准率，公式为：

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

例如，在垃圾邮件分类任务中，精确率反映了模型判断为垃圾邮件的邮件里，确实是垃圾邮件的占比。精确率高意味着模型在预测正例时更加可靠，误判情况较少。但精确率存在局限性，当正负例不均衡时，即使模型将大部分样本预测为负例，也可能获得较高的精确率，因此精确率常需结合其他指标共同评估模型性能。

模型评估有时还需要用到一些其他指标。

- F1 分数：精确率和召回率的调和平均值。
- ROC 曲线：受试者工作特征曲线，展示真正例率（TPR）与假正例率（FPR）的关系。
- ROC AUC：ROC 曲线下的面积，衡量模型的整体性能。

- PR 曲线：精确率-召回率曲线，展示精确率和召回率随阈值变化的关系。
- 计算速度：分类器训练和预测需要的时间。
- 鲁棒性：处理缺失值和异常值的能力。
- 可扩展性：处理大数据集的能力。

在 Python 中，我们可以使用 sklearn 库来计算这些评估指标，以下是常用函数的汇总与解释。

- accuracy_score(y_true, y_pred)：计算准确率。y_true 是真实标签，y_pred 是模型的预测标签。例如，真实标签是[1, 0, 1, 0]，预测标签是[1, 1, 1, 0]，通过 accuracy_score([1, 0, 1, 0], [1, 1, 1, 0])就能得到准确率。
- precision_score(y_true, y_pred)：计算精确率。同样，y_true 是真实标签，y_pred 是预测标签。默认情况下，计算的是二分类问题的精确率，如果是多分类问题，可以通过设置 average 参数来指定计算方式，如'micro'、'macro'等。
- recall_score(y_true, y_pred)：计算召回率。参数 y_true 和 y_pred 的含义与前面相同。
- f1_score(y_true, y_pred)：计算 F1 分数。输入真实标签和预测标签，就能得到综合评估的 F1 分数。

思考：在银行信用卡欺诈检测场景中，精确率和召回率哪个指标更为关键？为什么？

解析：召回率更关键。召回率高意味着模型能够尽可能多地识别出真正的欺诈交易。因为漏判（将欺诈交易误判为正常交易）带来的经济损失极其严重，而误判（将正常交易误判为欺诈交易）可通过人工复核等方式解决，相比之下，人们更不能容忍漏判，所以在这种场景下，应优先保证召回率。

3.4 聚类分析

本节介绍的聚类分析是无监督机器学习的一个核心任务。

3.4.1 聚类分析概述

聚类分析（Cluster Analysis）是在没有给定划分类别的情况下，依据数据之间的相似程度对样本进行分组的方法。

根据不同信息维度的特征划分出来的类别通常是不一样的。以分析《全唐诗》为例，聚类分析可以根据诗歌的用词、意象、情感基调等特征，将描写边塞风光的诗归为一类，描绘江南水乡的诗归为另一类；也可以根据诗歌的格律形式，将五言绝句、七言律诗、乐府诗等分别聚类。

我们根据分类对象的不同，将聚类分析分为对样本进行分类的 Q 型聚类分析和对变量进行分类的 R 型聚类分析两大类型。一个聚类算法通常只需要知道如何计算数据对象之间的相似度就可以开始工作了。

1. 相似性度量

在聚类分析中，判断数据对象之间的相似程度，需要借助相似性度量方法。相似性度量的指标有距离与相似系数两大类。

1）三种尺度类型

相似性度量的定义与三种尺度的变量类型有密切关系。

- 间隔尺度：变量用连续的量来表示，如长度、重量、速度、温度、奖金等。
- 有序尺度：变量的度量不用明确数量表示，而用等级来表示，如文化程度，产品等级。
- 名义尺度：变量既无等级关系也无数量关系，如性别、职业、产品的型号、颜色等。

本节主要讨论间隔尺度变量的样品聚类分析方法。

2）距离

常用的距离有欧几里得距离和马氏距离等。给定数据集 $Z=\{Z_1,Z_2,\cdots,Z_p,\cdots,Z_{N_p}\}$，其中 Z_p 是 N_d 维特征空间中的一个特征向量，而 N_p 是特征空间中特征向量的个数。

（1）欧几里得距离。

欧几里得距离又叫欧氏距离，它的定义为：

$$d(\boldsymbol{z}_u,\boldsymbol{z}_w)=\sqrt{\sum_{j=1}^{N_d}(z_{u,j}-z_{w,j})^2}=\|\boldsymbol{z}_u-\boldsymbol{z}_w\| \tag{3.6}$$

欧氏距离即两项间的差，是对每个变量值差的平方和再计算平方根，目的是计算其间的整体距离即不相似性。

欧氏距离主要有以下两个优点。

概念直观：基于几何空间的直线距离计算，符合人类对“距离”的直觉认知，易于理解和解释。

计算简单：仅需对数据点各维度差值的平方求和，再开方，计算复杂度低，适合大规模数据的快速处理。

欧氏距离的缺点是对量纲敏感：若数据各维度的单位或尺度差异大（如身高的单位用厘米、体重的单位用千克），会导致某些维度主导距离计算，影响聚类准确性。

（2）马氏距离。

马氏距离是由印度统计学家马哈拉诺比斯提出的，定义如下：

$$d_M(\boldsymbol{z}_u,\boldsymbol{z}_w)=(\boldsymbol{z}_u-\boldsymbol{z}_w)\sum{}^{-1}(\boldsymbol{z}_u-\boldsymbol{z}_w)^{\mathrm{T}} \tag{3.7}$$

其中，Σ 是数据分布的协方差矩阵。

马氏距离的优点如下。

消除了量纲影响：马氏距离通过协方差矩阵对数据进行标准化，能够自动调整不同特征的尺度，如使得身高（厘米）、体重（千克）等具有不同量纲的特征在距离计算中具有同等重要性，提升聚类准确性。

考虑特征相关性：基于数据的协方差结构计算距离，可有效捕捉特征间的线性关系。例如，在分析金融数据时，能根据股票收益率之间的相关性调整距离度量，避免因简单欧氏距离忽略关联而导致的聚类偏差。

马氏距离的缺点体现在如下方面。

计算复杂度高：需计算协方差矩阵的逆矩阵，当数据维度较高或样本量不足时，计算效率低且协方差矩阵可能不可逆，限制其在大规模数据场景的应用。

对数据分布假设严格：马氏距离隐含假设数据服从多元正态分布，若实际数据分布偏离该假设，可能导致聚类结果出现偏差。

3）相似系数

相似系数常用来度量指标之间的相似性。性质越相近的样本，它们的相似系数绝对值越接近 1，彼此无关的样本之间的相似系数接近 0。相似系数主要有两种。

（1）夹角余弦。

夹角余弦用于度量数据间的相似性，它更直观地呈现数据向量间的角度关系。在计算时，只关注数据向量的方向，不考虑向量的长度（即数据的绝对值大小）。定义如下：

$$< \boldsymbol{z}_u, \boldsymbol{z}_w >= \frac{\sum_{j=1}^{N_d} z_{u,j}, z_{w,j}}{\| \boldsymbol{z}_u \| \cdot \| \boldsymbol{z}_w \|}$$

由线性代数的柯西不等式可知，$< \boldsymbol{z}_u, \boldsymbol{z}_w > \in [-1, 1]$。夹角余弦通过测量两个向量内积空间的夹角的余弦值来度量它们之间的相似性。

（2）相关系数。

与夹角余弦不同，相关系数在计算时，不仅考虑向量方向，还会对数据的均值和标准差进行标准化处理，重点关注数据的线性关系紧密程度。相关系数通常使用皮尔逊相关系数（Pearson Correlation Coefficient）计算，公式如下：

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

其中，n 为数据点的数量；x_i 和 y_i 分别是变量 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 的第 i 个观测值，$\bar{x}$ 和 $\bar{y}$ 分别是变量 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 的均值；相关系数 r_{xy} 的取值范围为[−1, 1]，绝对值越接近 1，表示线性关系越强，0 表示无线性关系，正数表示正相关，负数表示负相关。

2. 聚类分析与机器学习分类的区别

聚类分析和机器学习分类都是数据分析领域的重要方法，但它们在原理、数据要求、应用场景和结果特点等方面存在明显差异。

1）聚类分析

聚类分析属于无监督学习算法，它不需要预先设定分类标准和数据标签。该算法通过分析数据对象自身的特征，比如数值大小、属性特征等，将相似的数据自动归为一类。

聚类分析主要用于发现数据中的潜在规律和结构，适用于探索未知的数据分析场景。

聚类分析的结果不唯一，受算法和参数设置影响，可能产生不同结果，且类别数量也不确定。

2）机器学习分类

机器学习分类是监督学习算法，在模型训练前，需要准备大量带有明确标签（分类结果）的数据。模型通过学习这些数据的特征和对应标签，掌握分类规则，进而对新数据进行分类预测。

机器学习分类更侧重根据已有规律进行精准预测。

机器学习分类在模型训练完成后，对于给定的输入数据，输出的类别是确定的。

3.4.2 K-Means 聚类

K-Means 聚类的核心目标，是将给定的数据集划分成 K 个不同的簇，让每个簇内的数据点尽可能紧密地聚集在各自的“中心”（称为质心）周围，而不同簇之间的数据点则尽量远离。

1. K-Means 算法

K-Means 聚类算法通过不断调整质心的位置，计算数据点到质心的距离，并将数据点分配到距离最近的质心所属的簇，反复迭代，最终使得簇内数据点到质心的距离之和最小，从而完成聚类任务。

具体步骤如下。

（1）随机初始化质心：从数据集中随机选择 K 个点作为初始质心。这一步是 K-Means 算法的起点，初始质心的选择对最终聚类结果有一定影响。为减少随机性带来的不稳定性，可进行多次随机初始化并选择最优结果，来更合理地选择初始质心。

（2）分配样本：计算每个数据点到各个质心的距离，通常使用欧氏距离计算：

$$d(\boldsymbol{x},\boldsymbol{y})=\sqrt{\sum_{i=1}^{n}(\boldsymbol{x}_i-\boldsymbol{y}_i)^2}$$

其中，$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别表示数据点和质心的向量形式，n 为特征维度。通过以上距离公式比较数据点与所有质心的距离，将每个数据点分配到距离最近的质心所在的簇。

（3）更新质心：重新计算每个簇的质心，即簇内所有样本的平均位置。对于具有 m 个数据点的簇 C_j，其新质心 $\boldsymbol{\mu}_j$ 的计算公式为：

$$\boldsymbol{\mu}_j=\frac{1}{m}\sum_{\boldsymbol{x}\in C_j}\boldsymbol{x}$$

这一过程相当于找到簇的“几何中心”，使得簇内数据点尽可能靠近该点。

例如，当一个簇内有三个数据点 1、3、5 时，更新后的质心为(1+3+5)/3=3。

（4）迭代终止判断：重复步骤（2）和（3），直到质心位置不再发生变化，或者达到设定的最大迭代次数为止。“质心位置不再发生变化”是指前后两次迭代中，所有质心的坐标差异都小于某个极小阈值（如 10^{-6}）。设置最大迭代次数可以避免算法因数据特性或初始质心选择不当而陷入无限循环。

例如，当最大迭代次数设为 100 次时，即使质心未完全收敛，算法也会在达到该次数后停止迭代，输出当前聚类结果。

2. 示例

已知 5 个样品的观测值为：3，6，8，10，15。试用 K-Means 法将其分为两类，初始质心分别取 3，15。

解：初始分类：

将观测值视为 5 个单独的类，根据初始质心 3 和 15，计算其余点到这两个质心的距离，距离 3 近的归为一类，距离 15 近的归为另一类。根据欧氏距离，3 距离自身为 0，6 距离 3 为 3，8 距离 3 为 5，10 距离 3 为 7，15 距离 3 为 12；而 3 距离 15 为 12，6 距离 15 为 9，8 距离 15 为 7，10 距离 15 为 5，15 距离自身为 0。所以初始分类为第一类{3，6，8}，第二类

{10，15}。

计算新质心：

第一类{3，6，8}的质心为(3 + 6 + 8)/3 ≈ 5.67，

第二类{10，15}的质心为(10 + 15)/2 = 12.5。

重新分配数据点：

再次计算各点到新质心 5.67 和 12.5 的距离：

3 距离 5.67 为 2.67，6 距离 5.67 为 0.33，8 距离 5.67 为 2.33，10 距离 5.67 为 4.33，15 距离 5.67 为 9.33；而 3 距离 12.5 为 9.5，6 距离 12.5 为 6.5，8 距离 12.5 为 4.5，10 距离 12.5 为 2.5，15 距离 12.5 为 2.5。

样本分类结果仍为第一类{3，6，8}，第二类{10，15}，质心不再变化，聚类结束。

3．常用函数汇总

- random.sample()：从给定的数据序列中随机抽取指定数量且不重复的元素，可用于选取初始质心，保证随机性和多样性。
- scipy.spatial.distance.euclidean()：专门用于计算两个向量之间的欧氏距离，帮助判断数据点与质心的远近。
- np.mean()：计算数组在指定轴上的平均值，用于更新簇的质心。

通过对比前后两次质心的差异，判断质心是否稳定，从而决定是否停止迭代。

3.4.3 层次聚类

层次聚类是一种基于簇间相似度构建树形聚类结构的无监督学习算法。

1．层次聚类算法与迭代步骤

层次聚类的基本思想是通过计算类间距离，将相似的数据逐步聚合，直至满足终止条件。阈值是用于控制层次聚类终止条件的一个关键参数。当类间的距离大于设定的阈值时，聚类过程停止。通过调整阈值，用户可以灵活控制最终聚类后类的数量：较小的阈值会生成更多、更精细的类；较大的阈值则会产生更少、更宽泛的类。

层次聚类的具体步骤如下。

（1）初始化：将每个数据点视为一个单独的类。此时，类的数量等于数据点的总数。

（2）计算初始距离矩阵：计算每两个类之间的距离，常见的类间距离度量方法有单链接（最近邻）、全链接（最远邻）、组平均等。

（3）合并聚类：找出距离最近的两个类，并将它们合并为一个新的类。

（4）更新距离矩阵：重新计算新类与其他类之间的距离。

（5）重复步骤：重复步骤（2）～（4），直到满足终止条件，如达到预设的类数量，或者所有类间的距离都大于阈值。

2．示例

已知 5 个样品的观测值为：A(1)，B(3)，C(6)，D(8)，E(12)，给定阈值 3 和 5，分别用最近邻方法、最远邻方法、组平均法聚类。

解：根据欧氏距离公式计算任意两个样品间的距离，得到初始距离矩阵：

	A	B	C	D	E
A	0	2	5	7	11
B	2	0	3	5	9
C	5	3	0	2	6
D	7	5	2	0	4
E	11	9	6	4	0

合并聚类与更新距离矩阵：

（1）最近邻方法。

第一轮：距离最小为 2（C-D 和 A-B），合并{C}和{D}得到[{A}, {B}, {C, D}, {E}]。

第二轮：计算新类与其他类的最小距离，$d(\{C, D\}, \{B\}) = 3$ 最小，合并{B}与{C, D}得到[{A}, {B, C, D}, {E}]。

若阈值是 3：由于 $d(\{B, C, D\}, \{A\}) = 2 < 3$，继续合并得到[{A, B, C, D}, {E}]；此时 $d(\{A, B, C, D\}, \{E\}) = 4 > 3$，聚类停止。若阈值是 5：继续合并[{A, B, C, D}, {E}]，因为 $d(\{A, B, C, D\}, \{E\}) = 4 < 5$，最终合并为[{A, B, C, D, E}]。

（2）最远邻方法。

第一轮：合并{C}和{D}得到[{A}, {B}, {C, D}, {E}]。

第二轮：计算新类与其他类的最大距离，其中 $d(\{C, D\}, \{B\}) = 5$ 最小，合并{B}与{C, D}得到[{A}, {B, C, D}, {E}]。

若阈值是 3：由于 $d(\{B, C, D\}, \{A\}) = 5 > 3$，聚类停止，结果为[{A}, {B, C, D}, {E}]。若阈值是 5：继续合并[{A}, {B, C, D}, {E}]，因为 $d(\{A\}, \{B, C, D\}) = 5$，$d(\{B, C, D\}, \{E\}) = 9 > 5$，最终结果为[{A, B, C, D}, {E}]。

（3）组平均方法。

第一轮：合并{C}和{D}，新类{C, D}的质心为$(6 + 8) / 2 = 7$，得到[{A}, {B}, {C, D}, {E}]。

第二轮：计算质心距离，$d(\{C, D\}, \{B\}) = |7 - 3| = 4$ 最小，合并{B}与{C, D}，新质心为$(3 + 6 + 8) / 3 \approx 5.67$，得到[{A}, {B, C, D}, {E}]。

若阈值是 3：$d(\{B, C, D\}, \{A\}) = |5.67 - 1| = 4.67 > 3$，聚类停止，结果为[{A}, {B, C, D}, {E}]。若阈值是 5：继续合并[{A}, {B, C, D}, {E}]，$d(\{B,C,D\}, \{A\}) = 4.67 < 5$，合并后质心为$(1+3+6+8) / 4 = 4.5$，此时 $d(\{A, B, C, D\}, \{E\}) = |12-4.5| = 7.5 > 5$，最终结果为[{A, B, C, D}, {E}]。

综上，不同方法与阈值下的聚类结果如表 3-6 所示。

表 3-6　不同方法与阈值下的聚类结果

方法	阈值 3	阈值 5
最近邻方法	[{A, B, C, D}, {E}]	[{A, B, C, D, E}]
最远邻方法	[{A}, {B, C, D}, {E}]	[{A, B, C, D}, {E}]
组平均方法	[{A}, {B, C, D}, {E}]	[{A, B, C, D}, {E}]

3．常用函数

- scipy.spatial.distance.pdist()：用于计算数据点间的距离矩阵，可选择不同距离度量方

法，如'euclidean'（欧氏距离）。

- scipy.cluster.hierarchy.linkage()：执行凝聚式层次聚类，method 参数设置链接方式，如'single'（最近邻）、'complete'（最远邻）、'average'（组平均）。
- scipy.cluster.hierarchy.dendrogram()：绘制树状图，直观展示层次聚类每次合并过程。
- scipy.cluster.hierarchy.fcluster()：根据聚类结果和设定阈值，将数据点划分到不同簇。

3.4.4 模型评估指标

1. 评估指标

将数据通过聚类分析方法分组后，结果好坏需要我们通过评估来判断，主要评估任务与指标如表 3-7 所示。

表 3-7 聚类分析主要评估任务与指标

评估类型	常用指标	评估目标	取值理想范围
内部评估	轮廓系数(Silhouette Score)	簇内紧密且簇间分离	接近 1
	CH(Calinski-Harabasz)分数	簇间方差/簇内方差比值	越大越好
外部评估	调整兰德指数(ARI)	对比聚类结果与真实标签	1（完全匹配）
	调整互信息(AMI)	衡量信息重叠程度	1（完全相关）

1）内部评估（无真实标签时）

评估任务：衡量聚类结果的“自洽性”，即数据点是否在簇内紧密、簇间分离。应用于用户分群、文档聚类等无标注数据场景。

内部评估指标：

（1）轮廓系数（Silhouette Score）。

原理：计算每个点到所属簇的距离 a 和到最近簇的距离 b。值越接近 1，表示聚类效果越好。

公式：$(b - a) / \max(a, b)$，a 越小（簇内紧密）、b 越大（簇间分离），值越大。

（2）Calinski-Harabasz 分数（CH 分数）。

原理：计算簇内方差和簇间方差的比值，值越大表示簇间分离度越高。

2）外部评估（有真实标签时）

评估任务：对比聚类结果与已知真实分组的一致性。

外部评估指标：

（1）调整兰德指数（Adjusted Rand Index，ARI）。

原理：对比聚类结果与真实标签的匹配程度，修正随机匹配的概率。

范围：[−1, 1]，1 表示完全匹配，0 表示随机匹配。

（2）互信息（Mutual Information，MI）。

原理：衡量聚类结果与真实标签的信息重叠程度，值越大相关性越强。

变种：调整互信息（Adjusted MI）可修正随机因素的影响。

2. 聚类分析评估的 Python 命令

以下是实现聚类分析评估任务的代码（以轮廓系数和调整兰德指数为例）：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score, adjusted_rand_score
# 1. 生成测试数据（4 个簇，300 个样本）
X, true_labels = make_blobs(n_samples=300, centers=4, random_state=42)  # 2. 执行 KMeans 聚类
kmeans = KMeans(n_clusters=4, random_state=42)
pred_labels = kmeans.fit_predict(X)
# 3. 计算评估指标
# 内部评估：轮廓系数（越接近 1 越好）
silhouette = silhouette_score(X, pred_labels)
# 外部评估：调整兰德指数（越接近 1 越匹配真实标签）
ari = adjusted_rand_score(true_labels, pred_labels)
# 4. 打印结果
print(f"轮廓系数: {silhouette:.3f}")
print(f"调整兰德指数: {ari:.3f}")
```

3.5 实践案例

3.5.1 特征工程

特征工程是机器学习中至关重要的环节，直接影响模型性能。通常包含以下步骤。

（1）数据加载与预处理：缺失值和异常值处理直接影响模型基础质量。

（2）特征衍生：结合业务知识构造有意义的新特征（如单价、到市中心距离）。

（3）特征转换：标准化和编码，这是数值与类别特征的必要处理。

（4）特征选择：通过统计检验或模型重要性筛选核心特征，减少冗余。

以房价预测为例，根据房屋属性预测房价。数据集中包含以下特征。

（1）基础信息：面积、房龄、卧室数量、浴室数量。

（2）位置信息：经纬度、学区评级。

（3）时间信息：建造年代、上次装修时间、装修情况。

（4）其他特征：车库容量、绿化率、物业费。

对于房价预测，完整的特征工程代码如下。

1. 数据加载

定义一个名为 load_data()的函数，其功能是从指定路径中加载 CSV 文件并返回一个 DataFrame 对象。

```
def load_data(file_path):
    try:
        df = pd.read_csv(file_path, encoding='gbk')
        print(f"数据加载成功，共{df.shape[0]}行，{df.shape[1]}列")
        return df
    except Exception as e:
        print(f"数据加载失败: {e}")
        return None
```