第3章

机器学习算法基础

本书前两章分别对推荐系统的一些基本概念和生产环境下的推荐系统进行了介绍。1.2.3节里，利用基于内容的推荐算法作为引例，介绍了推荐系统的原理，其实个性化推荐系统的核心是其背后的推荐算法，无论是学术界还是工业界，推荐算法的发展都是推荐系统应用不断发展的基石。从1997年“推荐系统”这一名词第一次被提出，到目前为止经过了二十多年的发展，推荐算法也从基于统计的方法逐渐过渡为基于机器学习、深度学习的方法。现在的研究人员对推荐算法的创新，最主要的思路也是以机器学习作为主体，通过数据驱动模型的方式解决问题，许多推荐算法中都可以发现机器学习的影子，而机器学习中的很多理论也指导着推荐算法的发展。学习经典机器学习方法以及思考如何将机器学习运用于解决实际问题的这种思路，对于学习和掌握推荐系统领域内的先进算法是重要且必需的一步，因此，本章会从一些传统机器学习方法例如线性回归算法出发，逐步递进到更加复杂但效果更好、更贴近前沿研究的方法，例如深度神经网络等，为读者后续学习推荐算法打下理论基础。

3.1 机器学习算法概述

为了让计算机解决问题，通常需要一个算法，将输入变换到指定的输出，实现特定的任务。例如，排序任务中，输入为一组数字，计算机执行排序算法输出这组数字的有序排列。然而，对于某些复杂的任务，无法直接通过编程来实现，尽管可以通过程序一步一步的if-else来设置所有的逻辑判断，当面对海量的条件时，强行实现算法难度太高且通用性不够。如果能让机器像人一样自我学习，对各种条件有自己的判断，让机器能够学习识别图片，学习识别语音等，就能够轻松应对这些复杂的任务。

对于排序任务，不需要排序数字就可以设计出能够排序的算法，然而很多应用程序，没有一个算法，只有已知输入和输出的数据实例，能否让计算机自动提取这些任务的算法呢？机器学习算法应运而生。机器学习是一类学习型算法，就是让计算机对一部分数据进行学习，对另外一部分数据进行预测与判断。本节首先对机器学习的工作过程做详细的介绍，随后介绍各个类型的机器学习算法的特点。

3.1.1 机器学习算法基本过程

机器学习是一类算法的总称，这些算法的作用是从大量历史数据中挖掘出其中隐含的规律，并利用这种规律对新出现的数据进行预测。不要小看“预测”两个字，它的作用太大

了，几乎所有的问题都可以归结为预测，例如，图片识别任务中，机器学习可以预测未分类图片所属的类别；在语音识别任务中，机器学习可以预测未知内容音频所要表达的信息；机器学习甚至可以根据历史的股票信息预测未来股价的走势。实际上，机器学习过程与人类的学习过程十分类似，人类的学习是一个人根据过往的经验，对同一类问题进行总结或归纳，得出一定的规律——也就是我们所说的知识，然后利用这些知识来对新的问题做出判断（也就是所谓的预测）的过程。下面以小学生学习新知识的过程为例，来类比讲解机器学习中的有监督学习过程，如图 3-1 所示。

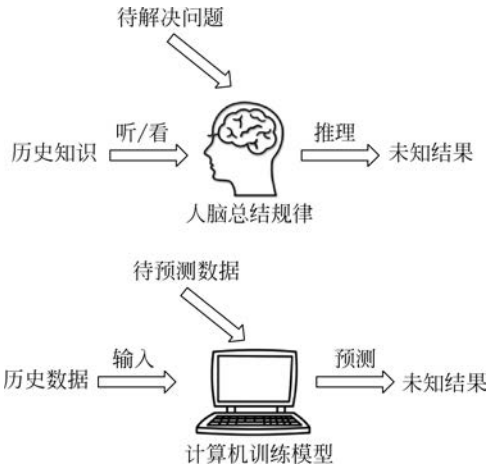

图 3-1　人脑学习与计算机学习对比

（1）准备好将要学习的知识相关的习题册。

（2）做习题，并通过“对答案”来纠正自己做错的习题。

（3）经过多轮习题练习，此时的小学生已经掌握了一定的知识。

（4）根据自己所学习到的知识，参加这部分知识相关的考试。

由上述过程可以看出，小学生通过做题学习新知识的过程很好地对应了机器学习中最主体的“有监督学习”的过程，具体对应关系如表 3-1 所示。

表 3-1　有监督机器学习与小学生学习过程对比

机器学习（有监督）	小学生学习
训练样本（对应大数据）	习题
样本标签	习题答案
训练过程（对应算法）	做习题过程（学习过程）
模型（训练的输出）	知识（学习的结果）
预测（代替人类做出判断）	考试（掌握知识加以运用）

由上述分析可见，经过一个过程的学习（训练）之后，模型对于新样本的预测能力相当于小学生在考试中的表现，在机器学习中称为泛化（Generalization）能力。泛化能力受诸多因素的影响，其中最为典型的是过拟合（Over-fitting）与欠拟合（Under-fitting）。过拟合与欠拟合也可以在小学生学习过程中找到对应物：①当小学生习题练习得不够充分时，在考试中就会遇到很多不会做的题型，导致考试成绩不理想——这种情况对应的是机器学习中的欠拟合问题；②而当小学生做了很多题，过分依赖所练习的习题（死做题），失去了对新题的

应变能力，即使同样的题目，只是换了数字，却也无从下笔，这也导致了考试的失利——这种情况则对应机器学习中的过拟合问题——过分追求逼近训练样本。

对于机器学习中过拟合的现象，通俗地说就是模型太过于追求拟合所有的输入数据，以至于把噪声数据的特征也纳入到模型的学习中，从而使学习到的知识过于“僵化”，即从噪声中学习到“伪知识”，这样在对新数据进行预测时表现出较差的泛化能力。在机器学习过程中，噪声和伪知识不可避免，我们能做的是尽量降低噪声以及其对应的伪知识的占比，从而提高模型的泛化准确性。因此，机器学习中解决过拟合的办法大体分为以下三种。

（1）优化输入数据。重新清理数据，去除噪声数据，或者增加数据量，使得训练数据中噪声占比尽量小，有效数据占比尽量大。

（2）调整模型损失函数。通过模型设计人员的经验和观察，在损失函数之后加入正则化项，例如 L1 正则化和 L2 正则化。其中，L1 正则具有对特征进行筛选的作用，去除噪声特征，L2 正则对大数值的权重向量进行严厉惩罚，使得网络更倾向于使用所有输入特征，而不是过度依赖于输入特征中的小部分特征。

（3）修改神经网络结构。例如，在神经网络中通过 dropout 方法，使得神经元在训练的时候以一定的概率不工作，从而减少隐藏层节点之间的相互作用，从而达到防治过拟合的效果。

机器学习中出现欠拟合的问题常常是因为模型学习能力太弱，难以刻画数据样本背后隐藏的相对复杂的问题，无法学习到数据集中的“一般规律”——可以认为是“想得太简单了”。因此，解决欠拟合的主要办法就是要提高模型的学习能力，一种解决办法是选择学习能力更强的非线性模型（例如决策树、深度学习等）；另一种办法是通过在现有模型上添加其他特征项，例如，考虑加入组合特征、更高次的特征等来增大假设空间等。

从数学的角度来说，可以将机器学习看作寻找一个映射函数（一般是非线性的），输入是历史数据，输出是预测的结果。通常这个映射函数会很复杂，甚至很难用具体化、形式化的公式来表达。机器学习的目标是使得学到的函数不仅在历史数据（训练集）上表现很好，更重要的是它同样能够适用于新数据（测试集）。

通常，让计算机来学习一个好的映射函数，要经过以下四个步骤。

（1）使用特征工程处理原始数据。特征工程是机器学习工作流程的最初阶段，指的是把原始数据转变为模型的机器学习算法可训练的数据过程，其目的是能够获取更好的训练数据特征，是机器学习最重要的一方面。如果特征做得不好，即使再好的算法也难以获得满意的结果，例如，在猫狗分类任务中，使用毛色作为区分特征的效果远不如使用外形轮廓作为区分特征的效果，因为猫和狗的毛色种类较多且接近，而二者的外形轮廓存在着更为明显的差异。

特征工程一般包含两方面：特征抽取、特征选择。特征抽取是将原始数据转换成算法可以理解和使用的数据，经过特征抽取后得到的新特征是原来特征的一个映射，例如，先将图片每个像素点转换为一个 0～255 的灰度值，将这些点放入一个向量中；然后再将这个向量喂给算法进行训练。经过不断的训练，算法就能够根据这些向量学习出图片中是否有某一类别的对象。然而有时候，经过抽取后的特征维度很高，需要花费很大的算力，而且这些特征并不一定都是有用的，因此需要一些特征选择算法，对这些特征进行评估，筛选出最重要的特征，经过特征选择后得到的特征是原来特征的一个子集，例如，预测一个西瓜是否是好瓜时，原始的特征包含瓜藤、瓜蒂、颜色、花纹、运输方式等众多特征，经过特征选择后，将运输方式等无用的特征丢弃以减少不必要的计算。

（2）选择合适的机器学习算法。机器学习的目标就是通过训练样本得到稳定的、可以预测新数据的模型。这里的模型可以认为是两个方面的组合物：①某一种模型结构（如逻辑回归、朴素贝叶斯、神经网络等），相当于一种映射函数；②经过“附着”在这个模型结构（映射函数）之上的参数（这些参数就是模型训练之后得到的输出）。选择什么样的训练算法需要依据实际的问题而定，针对不同的问题，算法的设计和选择也有很大差异。因此，对于一个算法工程师来说，对各类机器学习模型的用途、优缺点的了解，以及对实际任务的分析并做出正确的模型选型进行训练尝试十分重要，也是算法工程师是否合格、优秀的主要衡量要点。

（3）定义损失函数。判断一个模型的好坏，需要确定一个衡量标准，即损失函数。通俗地说，损失函数是用来描述模型预测值与真实标签值之间的差距，而计算差距的方式则需要依据具体问题而定，例如，在分类算法中较为常用的是交叉熵损失函数，在回归算法中一般采用欧氏距离损失函数等。

（4）选择优化方法。定义好损失函数只是确定了优化的目标，损失函数的值越小代表模型对数据的拟合越好，如何通过大量的输入数据寻找到一组参数，使得这个损失函数达到最小，这就需要选择一种数学上的优化求解方法。对于使用机器学习算法解决应用型的任务（如推荐系统）来说，一般尝试使用一些通用型的模型优化算法，例如，梯度下降算法、Adam 优化算法、牛顿法等。

3.1.2 机器学习算法的分类

机器学习的算法有很多，在实际应用中，需要根据不同的应用场景选取合适的机器学习算法，才能发挥各个算法的作用，切实解决实际场景的智能化任务。机器学习算法根据不同的分类标准有多种不同的类别。

首先，根据需要解决问题的类型不同，可以将机器学习划分为：分类问题、回归问题、聚类问题。回归与分类问题都是研究一组随机变量（自变量）与另一组随机变量（因变量）之间的关系。用中学数学函数中的 $y=f(x)$ 为例来说，因变量 y 是指被影响、决定的变量，本身不参与运算，而自变量 x 则是指自身发生变化、改变并参与运算，最终影响因变量的变量。不同之处在于：分类问题所预测的数据对象是离散值，即有限的类别（如判断电子邮件是/否为垃圾邮件，肿瘤是恶性/良性等）；回归问题预测的数据对象则是数值型连续随机的变量，使用场景如房价预测、股票走势等连续变化的案例；聚类问题则用于在数据中寻找隐藏的模式或分组，将数据集中的样本划分成若干个通常不相交的子集，每一类中的数据具有更高的相似度，一般可应用于新闻聚类、文章推荐等场景。

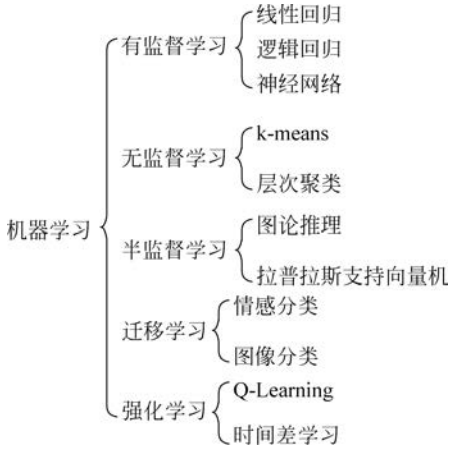

图 3-2 机器学习分类图

其次，根据输入数据类型的不同，对于同一个问题的建模也有多种方式，在机器学习和人工智能领域，通常会考虑算法的学习方式，这样可以在建模和算法选择的时候能根据输入数据来选择最合适的算法以获得最好的结果。在机器学习领域主要有以下五种重要的学习方式，如图 3-2 所示。

（1）有监督学习。在有监督学习中，输入数

据被称为训练数据，且每个训练数据都有一个对应的真实标签，例如，图片分类系统中一个图片对应一个类别标签，邮件分类系统中每个邮件对应“垃圾邮件”或“非垃圾邮件”标签等。有监督学习的学习过程，是将模型的预测结果与训练数据的实际标签进行比较，不断调整预测模型的参数，使得模型的预测能力向着更准确的方向改进，以达到令人满意的准确率。有监督学习算法包含对回归以及分类问题的处理，常见的算法有线性回归、逻辑回归、支持向量机、神经网络等算法。

(2) 无监督学习。与有监督学习最大的不同，无监督学习的训练样本是没有标签的，机器从无标签的数据中探索并推断出潜在的联系，为进一步数据分析提供基础。此类学习方式主要用于两种任务：①聚类，在聚类任务中由于事先不知道数据的类别，因此只能通过分析数据样本在特征空间中的分布，从而将数据集中的样本划分为若干个不相交的子集，把相似的数据聚为一类，常见的无监督聚类算法有 k-means 与层次聚类等算法；②降维，即减少数据变量中的维度，很多时候，输入数据都是非常高维度的特征，高维数据不仅会给计算带来麻烦，而且这些特征中包含大量冗余的特征，这些冗余的特征会妨碍模型查找规律，降维就是在保留数据结构和有用性的同时对数据进行压缩。常用的无监督降维算法有主成分分析(Principal Component Analysis)与奇异值分解(Singular Value Decomposition)算法等。

(3) 半监督式学习。在很多实际问题中，只有少量的带有标签的数据，而对数据进行标注的代价十分昂贵。例如，医学图像中的标注需要结合特别的设备以及专家的指导，因此提出半监督学习，一种将有监督学习和无监督学习相互结合的学习方式。在此学习方式下，输入数据包含大量的未标记数据和少量标签数据，这些算法首先对不包含标签的数据进行建模，在此基础上再对标识的数据进行预测。半监督学习的核心思想就是利用未标记数据帮助模型定义同类样本的边界，再借助少量有标签的样本为各个类别提供标签信息，从而实现以无监督学习来增强有监督学习的分类效果。半监督学习中常用的算法有图论推理算法(Graph Inference)和拉普拉斯支持向量机(Laplacian SVM)等。

(4) 迁移学习。无监督学习利用大量的无标签数据，解决了标注困难的问题，然而也有很多问题领域，无法得到足够的数据。例如，语音识别任务中，对于普通话有着足够多的数据，然而对于那些只有少数人说的方言，所拥有的数据就不够庞大。为此，希望通过一种方式能将从普通话中学习得到的东西，用于数据量不多的方言的识别，这种方式正是迁移学习。通俗来说，迁移学习就是运用已有知识来学习新的知识，核心在于找到已有知识与新知识之间的相似性。在迁移学习中，称已有的知识为“源域”，要学习的新知识叫“目标域”，在迁移学习中，源域和目标域虽然不同，但一般有一定的关联，需要尽量减小源域和目标域的分布差异，进行知识迁移，从而实现跨域数据标定。这类学习方式常用于情感分类、图像分类等任务中，对现有知识数据域进行复用，让已有的大量工作不至于完全丢弃。

(5) 强化学习。在有监督学习方法中，输入数据仅被用于检查模型的对错，而在强化学习中，输入数据直接反馈到模型，模型必须对这种反馈立刻做出调整。强化学习的灵感来源于心理学中的行为主义理论，即有机体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为。用一个通俗的例子来说明：将小白鼠放入迷宫中寻找出口，如果前进方向正确，就会给它正反馈(奖励食物)，否则给出负反馈(电击惩罚)，则当小白鼠走完迷宫所有道路后，无论将它置于何处，它都能通过之前的学习找到通往出口的正确道路。强化学习最为典型的应用就是谷歌 AlphaGo 的升级品——AlphaGo

Zero，无须人为设计特征，将棋子在棋盘上的摆放情况作为模型的输入，机器通过自我博弈的方式，不断提升自己从而完成出色的下棋任务。强化学习常见的算法包括 Q-Learning 和时间差学习（Temporal Difference Learning）等。

深度学习能够捕捉非线性的用户-项目关系，能够处理图像、文本等各种类型的数据源，因此基于深度学习的推荐系统得到了越来越多的应用。但是现有的模型大多数建立在静态图的基础上，实现了系统的短期预测结果的最优，而忽略了推荐是一个动态的顺序决策过程，长期的推荐目标没有被明确地解决。

强化学习的本质是让初始化的智能体在环境中探索，通过环境的反馈来不断纠正自己的行动策略，以期得到最大的奖励。在推荐系统中，用户的需求会随时间动态变化，强化学习智能体不断探索的特性正好符合了推荐系统对动态性的要求，并且在不断尝试建模更长期的回报。因此，越来越多的研究者将强化学习应用在推荐系统中。

3.2 线性回归算法

线性回归可以说是机器学习中最基本的问题类型，包含以下两个方面的字眼。

（1）线性。线性的含义是因变量与自变量之间按比例、成直线的关系，在数学上可以理解为一阶导数为常数的函数。相对的，“非线性”则是指不按比例、不成直线的关系，一阶导数不为常数。可以看出，变量之间的关系有千万种，线性关系是一种特例，也是最简单的关系。对于模型而言，线性和非线性的区别在于前者可以用“直线”将样本划分开。

（2）回归。关于回归的概念，在之前已经介绍过，回归的本质与分类相同，都是研究一组变量与另一组变量之间的联系，不同之处在于回归预测的是一组连续值，例如，明天的天气温度、股票的走势等。

线性回归的概念最初来源于统计学，如今被广泛用于机器学习中。如果两个及以上的变量之间存在“线性关系”，即已知模型结构了，剩下来需要做的即是利用历史数据估计线性模型结构中的参数，也就是因变量前面的系数（也包括截距），建立一个完整（模型结构＋模型参数）、有效的模型来预测新给出的变量所对应的结果。

线性回归看似简单，然而许多先进的机器学习方法都可以看作线性回归的扩展，本节将对线性回归的原理和算法做一个详细的介绍。

3.2.1 线性回归模型

对于单特征（自变量）场景来说，线性可以理解为一条直线，函数定义形如 $y=ax+b$。线性回归模型可以理解为用一条直线较为精准地描述一个特征与结果之间的映射关系，扩展到多特征时，例如在预测房子价格时，房子的位置、占地面积、年龄等作为一组特征，房价作为需要预测的结果，那么可以将训练样本 j 的 n 个特征视为一个向量 $(x_1^j, x_2^j, x_3^j, \cdots, x_n^j)$，对应的房价标记为 y_j，则线性回归模型的函数关系式可以用如下形式表示。

$$H(x_1, x_2, \cdots, x_n) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + b \tag{3.1}$$

其中，x_i 为自变量，对应实际任务中的不同维度的特征，θ_i 表示对应的权重系数，控制着特征 i 对结果的影响程度，是模型要学习的参数。b 表示偏置项，在线性模型中也称为函数的截距。为了方便形式化表达，又将 b 写作 θ_0，并添加 $x_0=1$ 项，则式（3.1）中的偏置项

可以写作 $\theta_0 x_0$，可以用矩阵的形式将式(3.1)简写为如下：

$$H(x_1, x_2, \cdots, x_n) = \sum_{i=0}^{n} \theta_i x_i = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{X} \tag{3.2}$$

其中，$\boldsymbol{\theta}^{\mathrm{T}}$ 表示权重系数向量(未知，待学习)，$\boldsymbol{X}$ 表示特征向量(已知)。当只有一个变量时，模型是二维平面中的一条直线，此时称为一元线性回归，当有两个变量时，模型是三维空间中的一个平面，有更多变量时，模型将是更高维度的，此时称作多元线性回归。线性回归就是要找到一条直线(平面，或更高维度的超平面)，并且让这条直线尽可能拟合所有输入的数据点。图 3-3 为线性回归模型图。

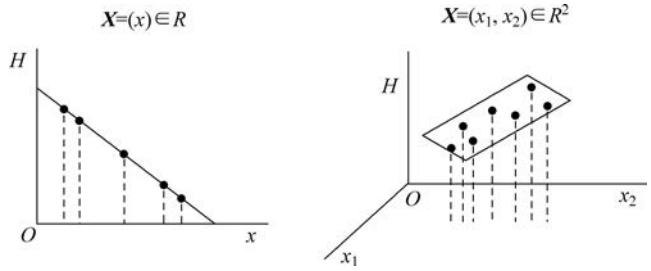

图 3-3　线性回归模型图

3.2.2　线性回归模型的损失函数

对于上述线性回归模型的公式，其中，$\boldsymbol{X}$ 是已知的特征向量$(x_1, x_2, \cdots, x_n)$，然而$\boldsymbol{\theta}=(\theta_1, \theta_2, \cdots, \theta_n)$的值是未知的，如图 3-3 中的二维平面中，不同的$\boldsymbol{\theta}$ 值对应不同的直线，多条直线都满足线性回归模型的公式，但显然这些直线对于数据点的拟合能力各不相同。图 3-4 为不同的线性回归曲线对比。为了找到最佳的一条直线，使得模型的预测值 $H(\boldsymbol{X})$尽可能接近样本 $\boldsymbol{X}$ 对应的真实结果 $\boldsymbol{Y}$，就需要依据训练数据计算出最优的 θ 的值，于是定义一种用于描述 $H(\boldsymbol{X})$和真实值 y_i 之间的差距的函数，称为损失函数。

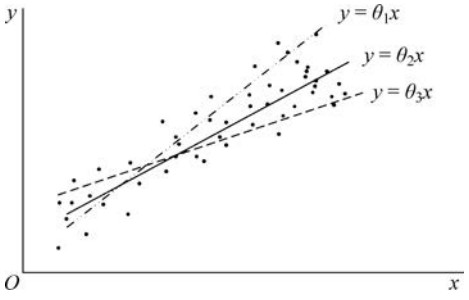

图 3-4　不同的线性回归曲线对比图

线性回归中，常用均方误差损失函数来计算预测值和真实值之间的差值(误差)，即线性回归图像中数据点到拟合直线之间的垂直距离。由于误差计算之后存在正负号，因此使用误差的平方来避免正负误差互相抵消效应，再用误差平方和除以数据样本总量 n 即得到均方误差，用公式写作：

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - H(x_i))^2 \tag{3.3}$$

其中，y_i 为真实标签值，$H(x_i)$为模型的预测值。要使所有点到直线的距离之和最小，就是要最小化均方误差。可见，损失函数就是衡量回归模型误差的函数，也就是用于拟合数据点的“直线”的评价标准，损失函数值越小，直线越能拟合数据。

3.2.3 梯度下降求解线性回归模型参数的最优值

定义好损失函数之后，只是有了一个对模型表现的评价标准，还是无法得到最佳的模型参数集合，因此还需要一种求解让损失函数最小的参数组合的方法。梯度下降（Gradient Descent）正是求解机器学习模型参数最常用的方法之一。

1. 什么是梯度和梯度下降

要理解梯度下降为什么能够让模型的损失函数最小，或者局部最小，首先需要理解梯度这个概念。

梯度是微积分中一个重要的概念，给定一个可微的函数，它可以对应到物理空间中的一个曲线或曲面（一元函数对应二维空间中的曲线，二元函数对应三维空间中的曲面，多元函数对应高维空间中的超曲面），梯度表示的是函数对于所有自变量求偏导数，并由全部偏导数组成的向量，梯度的方向与函数在当前点变化最快的方向一致。

在优化线性回归模型的损失函数时，我们希望找到能使损失函数值最小的参数组合，这个参数组合被称为损失函数的全局最优解。那么应该如何找到一个最优的参数使得损失函数达到最小呢？

以二元函数的优化求解为例（之所以选择二元函数为例是因为一元函数太特殊，只有一个自变量，不利于以此类推到更多元；而对于二元函数来说，如果理解了梯度下降方法的优化过程，对于更高维度的函数来说就比较容易理解了。因为，$n=2$ 与 $n=4,5,6,\cdots,n$ 没有太大区别，容易以此类推），参考一个游客下山（三维空间，对应二元函数）的过程：一个游客被困在山上，需要从山上下来（找到山的最低点）。但此时山上的浓雾很大，导致能见度很低，只能看到身边 1m 的地形。因此，想要顺利下山，只能利用自己可以看见的周围 1m 的信息，一步一步地找到下山的路。具体过程：①首先，以他当前所处的位置为基准，寻找这个位置下山最陡峭的地方，然后朝着下降方向走一步；②然后，又继续以当前位置为基准，再找最陡峭的地方走，直到最后到达最低处。梯度下降示意图如图 3-5 所示。

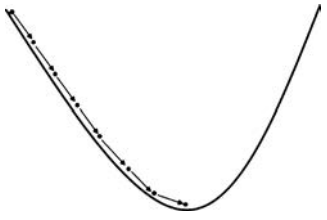

图 3-5 梯度下降示意图

上述方法正是利用了梯度下降的基本思想，梯度在数学上被定义为函数增长最快的方向，让损失函数的自变量每次都沿着梯度相反的方向变化，从而在多轮迭代后取得全局最优解。同理，如果是上山也可以有类似的思路，对应到机器学习中就变成梯度上升算法了。

体会了梯度下降的基本思想，那么梯度又如何计算呢？实际上，如果只是想求出梯度是十分简单的，这里先直接给出梯度的定义和求解方式。对于一元函数而言，梯度是一个标量数值，其实就是函数的微分，代表了这个函数在某个给定点的切线斜率。具体来说，就是将一元函数对其自变量求导，当前自变量位置的导数值即表示了函数的梯度值；而在多元函数中，只能求得函数对每一个自变量的偏导数，梯度向量的计算方法是将当前位置每一个偏导数值求出，再组合成向量，该向量的方向即代表了多元函数生成的曲面在当前点的梯度，并且是变化最快的方向（至于为什么，将在后面进行解释推导）。在实际进行机器学习模型训练时，由于事先不知道损失函数的最小值取于何处，可以先对参数进行一个随机选择，然后计算出损失函数对参数的导数，在训练的每一轮迭代中让参数沿着梯度方向变化一定的

值，下一轮迭代前再重新计算梯度值，从而不断地逼近最优解。

梯度下降法是一种常用的迭代方法，其目的是让输入特征向量找到一个合适的迭代方向，逐步使得误差函数值能达到局部最小值。在拟合线性回归方程时，我们把损失函数视为以参数向量为输入的函数，找到其梯度下降的方向并进行迭代，就能找到误差函数最小时对应的最优参数值。

2. 梯度下降求解线性回归模型最优参数

为便于读者理解，这里以最简单的一元线性回归为例，说明梯度下降求解参数的过程。

(1) 首先，将损失函数写作如下形式：

$$J(\theta)=\frac{1}{2n}\sum_{i=1}^{n}(y_i-(\theta_1 x_1^i+\theta_0))^2 \tag{3.4}$$

式(3.4)中，θ_0 即为前面所说的偏置项，x_1^i 表示第 i 个样本的第 1 个特征值，多元线性回归与一元线性回归的区别在于特征值(自变量)的数量不止一个。

(2) 对损失函数求偏导得到：

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{\partial}{\partial\theta_j}\frac{1}{2n}\sum_{i=1}^{n}(y_i-(\theta_1 x_1^i+\theta_0)) \tag{3.5}$$

(3) 对于式(3.5)，需要求解的参数只有 θ_1 和 θ_0 两个，分别对这两个参数求导。

当 $j=0$ 时：

$$\frac{\partial}{\partial\theta_0}J(\theta)=\frac{1}{n}\sum_{i=1}^{n}((\theta_1 x_1^i+\theta_0)-y_i) \tag{3.6}$$

当 $j=1$ 时：

$$\frac{\partial}{\partial\theta_1}J(\theta)=\frac{1}{n}\sum_{i=1}^{n}(((\theta_1 x_1^i+\theta_0)-y_i)\cdot x_1^i) \tag{3.7}$$

由于 $x_0=1$，上面两个式子可以统一为式(3.8)：

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{1}{n}\sum_{i=1}^{n}(((\theta_1 x_1^i+\theta_0)-y_i)\cdot x_j^i) \tag{3.8}$$

我们知道，函数对某个自变量 θ_j 的偏导数的意义为函数在这个自变量方向 θ_j 上对应这一点上的切线的斜率。因此，θ 减去对应的偏导数，就等于函数朝着最小值的方向移动了一步，这一步的大小可以由一个参数 α 来控制，即机器学习所说的学习率(Learning Rate)。学习率设置过小，参数更新得缓慢，会导致模型收敛过程十分缓慢；学习率设置过大，梯度则会在最小值附近来回震荡，甚至会导致模型无法收敛，因此需要一个适当的学习率约束着每次下降的距离不会太多也不会太少。设置学习率后的参数优化公式可写作：

$$\theta_j:=\theta_j-\alpha\frac{1}{n}\sum_{i=1}^{n}((H(x_j^i)-y_i)\cdot x_j^i) \tag{3.9}$$

其中，j 为线性回归模型中的参数个数，当 $j=0$ 时表示偏置项 b，此时对应的 x_0^i 值为固定值 1。且式(3.9)为只取一个样本时的数学表达，如果取所有的样本则称为批量梯度下降(Batch Gradient Descent，BGD)。当训练数据集太大时，通常会随机选取其中一个样本，称为随机梯度下降法(Stochastic Gradient Descent，SGD)。

3. 为什么梯度反方向是函数下降最快的方向

本节前面内容介绍了什么是梯度以及梯度下降，并在线性回归模型中实际运用了梯度

下降算法，讲解了如何对模型参数进行更新。实际上，很多机器学习算法，都可以使用梯度下降进行模型优化，这主要是因为机器学习算法在训练时都需要不断最小化模型的损失函数，而梯度的方向是函数增长最快的方向，通过迭代地对参数值以梯度负方向（最小化损失函数为目标）给定一小步的改变，达到最快达到最小值的目的，进而训练出参数最优解。然而，读者可能会产生一些疑问，为什么梯度的方向就是函数增长最快的方向呢？这是理解梯度下降优化算法的关键，本节下面的内容将对该点进行阐述。

关于梯度向量存在两个“硬性的定义”：①梯度是函数值上升最快的方向；②高维空间中，梯度是每个自变量的偏导组成的向量的方向。然而，读者可能会产生一些疑问——为什么这两个“硬性定义”的方向向量刚好就一致了？这个是理解梯度下降优化算法的关键，本节下面的内容将对该点进行阐述。

在讲解梯度之前，先对导数进行一个复习。

（1）首先，理解一下一元函数的导数的意义。从数学上说，如果一个函数 $y=f(x)$ 的自变量 x 在某一点 x_0 处产生了一个增量 Δx，当这个增量无限趋近于 0 时，函数值的增量 Δy 与自变量增量的比值存在极限，便称这个极限为 $f(x)$ 在 x_0 处的导数，记作 $f'(x_0)$。一个一元函数对应了二维空间中的一条曲线，导数的几何意义在于：函数 $f(x)$ 在某个点 x_0 处的导数值等于函数曲线在该点 x_0 切线的斜率，其物理意义在于导数值表示函数在某点的瞬时变化率。

（2）接着，扩展到二元函数的偏导数的理解。在一元函数中，只存在一个自变量，因此函数的导数自然是唯一的自变量 x 方向的变化率方向，而在多元函数中，由于涉及两个以上的自变量，按照数学上求极限的方法，对某一个自变量求导时，需要将其他变量进行固定处理，此时导数转变为偏导数。二元函数的图像对应的是三维空间中的一个曲面，不同于曲线在某一点的切线，一个曲面上的一点可以衍生出无数个方向的切线，此时二元函数的偏导数实际上是对应了函数沿着各个坐标轴的变化率。以二元函数 $z=f(x,y)$ 为例，z 对 x 的偏导数 $f_x(x,y)$ 表示固定 y 值时函数沿着 x 轴方向的变化率，当确定了 y 的值时，$f_x(x,y_0)$ 可以视为函数的曲面被平面 $y=y_0$ 所截得的曲线在某一点的切线斜率，函数对变量 y 的偏导数意义相似。

（3）再者，看一下方向导数的概念。从上述二元函数的偏导数含义可以看出，单个偏导数的值只能表示函数沿着相应坐标轴方向的变化率。因为曲面上的一个点有无数条不同方向的切线，因此不能保证函数沿着坐标轴方向的变化速度是所有方向上变化最快的，此时便需要引出方向导数的概念。

方向导数指的是：在函数定义域内的某个点，对某一方向（不一定是坐标轴方向）求导得到的导数，表示的是一个函数沿着某个指定方向的变化率。在一个平面中，给定不平行的两个基向量，可以表示出平面内任意一个向量，同理，给定了二元函数对 x 和 y 的偏导数，可以使用这两个偏导数来组合表示出曲面在任意方向上的切线斜率。对于上面的二元函数 $z=f(x,y)$，假设曲面上存在一个点 $z_0=(x_0,y_0)$，从 z_0 出发可以引出任意一个方向的向量，假设 $\boldsymbol{u}$ 表示任意方向的单位向量，θ 表示 $\boldsymbol{u}$ 与 x 轴正方向的夹角，那么应该如何求点 z_0 在 $\boldsymbol{u}$ 方向的导数（变化率）呢？在普通高等数学课本中基本都讲解过方向导数的计算方法：

$$\frac{\partial f}{\partial \boldsymbol{u}}\Big|_{z_0} = f_x(x_0, y_0)\cos\theta + f_y(x_0, y_0)\sin\theta \tag{3.10}$$

此时，将$(f_x(x_0, y_0), f_y(x_0, y_0))$定义为向量$\boldsymbol{A}$，将$(\cos\theta, \sin\theta)$定义为向量$\boldsymbol{I}$，可以得到：

$$\frac{\partial f}{\partial \boldsymbol{u}}\Big|_{z_0} = \boldsymbol{A} \cdot \boldsymbol{I} = |\boldsymbol{A}| \times |\boldsymbol{I}| \times \cos\alpha \tag{3.11}$$

α为两个向量之间的夹角。可见，如果想要式(3.11)方向导数取最大值，有$\cos\alpha = 1$，即$\alpha = 0$，也就是说，$\boldsymbol{I}$与$\boldsymbol{A}$平行(重合)时，函数增长的速度是最快的。最终可以给这个二元函数(三维曲面)在某个点增长速度最快的方向起一个名字，叫作梯度，它的计算方法就是对于其二元自变量x，y分别求偏导数，再将这些偏导数组成一个向量，便得到梯度。

综上，以二元函数(三维曲面)为例解释了为什么曲面上的某一点的梯度(人为定义为函数在两个坐标轴方向的偏导数组成的向量)是从该点可以引出的众多方向中上升最快的方向。梯度的这个性质可以很容易地推广到更高维度的超曲面中，即n元函数的梯度方向就是其在各个自变量坐标轴的偏导数组成的向量的方向。

3.2.4　线性回归算法正则化

正则化的本质是对模型权重系数大小的约束。某个特征的权重系数越小，该特征对于最后的结果带来的影响就越小，这一无关紧要的特征就只能对模型预测结果进行微调，扰动较小，可以让模型专注于有决定性的那些特征。

在梯度下降优化线性回归模型的参数过程中，我们希望通过对权重系数进行约束，抑制相关度小的特征对预测结果的扰动，以此来减小过拟合。试想，在判断一个西瓜是否好坏的过程中，本身结果与某一特征相关性很小(例如西瓜的运输方式)，然而恰好在两个样本中，这一特征不同，结果也不同，一旦模型误将这一特征看作重要的分类依据，就会出现过拟合的现象，因此使用正则化对权重系数进行约束后，能够有效避免模型的过拟合。一般常用的有L1正则化和L2正则化。其中，线性回归的L1正则化有个特殊的名字叫Lasso回归，加入L1正则化后的线性回归损失函数：

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - (H(X_i))^2 + \lambda\sum_{j=1}^{n}\|\theta_j\| \tag{3.12}$$

L1正则化中L1指的是L1范数，它指的是向量中各个元素绝对值之和。线性回归的损失函数中参数的L1范数即为$\sum_{i=1}^{n}\|\theta_j\|$，系数$\lambda$用于调节损失函数中均方差项、正则化项的权重。然而Lasso回归的求解比较复杂，通常线性回归算法中采用L2正则化，又称Ridge回归，即在损失函数中加入L2范数，其表达式如下。

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - (H(X_i))^2 + \frac{1}{2}\lambda\sum_{j=1}^{n}\theta_j^2 \tag{3.13}$$

L2范数指的是向量各元素的平方和，即式(3.13)中的$\sum_{j=1}^{n}\theta_j^2$，这里采用最小二乘法来求解Ridge回归，首先将式(3.13)写成矩阵的形式：

$$J(\theta) = \frac{1}{2}(\boldsymbol{X}\theta - \boldsymbol{Y})^{\mathrm{T}}(\boldsymbol{X}\theta - \boldsymbol{Y}) + \frac{1}{2}\lambda\sum_{j=1}^{n}\theta_j^2 \tag{3.14}$$

式中 $\boldsymbol{X}=\{x_0,x_1,\cdots,x_n\}$，$x_0$ 为常数 1，表示偏置项 θ_0 的系数。令 $J(\theta)$关于 θ 的导数为 0，则有：

$$\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\theta-\boldsymbol{Y})+\lambda\theta=0 \tag{3.15}$$

整理后得到能使损失函数值最小的 θ：

$$\theta=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{E})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} \tag{3.16}$$

3.2.5 实验

数据准备：这里以房屋价格的预测为例，根据房屋的面积来预测房屋的价格，输入数据如表 3-2 所示。

表 3-2 房屋价格预测输入数据

编号	面积	价格	编号	面积	价格
1	100	6400	5	300	10 450
2	150	7500	6	350	11 370
3	200	8130	7	500	16 400
4	250	9600			

使用 sklearn 机器学习框架提供的函数建立一个线性回归模型：$\boldsymbol{H}(x_1)=ax_1+b$。其中，$a$ 和 b 是要学习的参数，模型拟合输入的数据，利用梯度下降算法不断朝着损失函数减小的方向迭代更新参数。代码如图 3-6 所示。

```
import pandas as pd
from io import StringIO
from sklearn import linear_model
import matplotlib.pyplot as plt

#房屋面积与价格历史数据
csv_data = 'size,price\n100,6400\n150,7500\n200,8130\n250,9600\n300,10450\n350,11370\n500,16400\n'
#读入dataframe
df = pd.read_csv(StringIO(csv_data))
print(df)

#利用sklearn库建立线性回归模型
regr = linear_model.LinearRegression()
#拟合输入数据
regr.fit(df['size'].values.reshape(-1, 1), df['price'])
#得到直线的斜率、截距
a, b = regr.coef_, regr.intercept_
#给出待预测面积
area = 287
#根据直线方程预测的价格
print(a * area + b)
```

图 3-6 线性回归核心代码

实验结果：线性回归算法拟合数据的效果如图 3-7 所示。由图可见，历史数据点均分布在直线的附近，当给定一个新的面积值，将面积值代入模型学习到的直线方程就可以预测出该面积值对应的房价（所预测的房价值可能是真实值，也可能是一个和真实值较为接近的数值）。

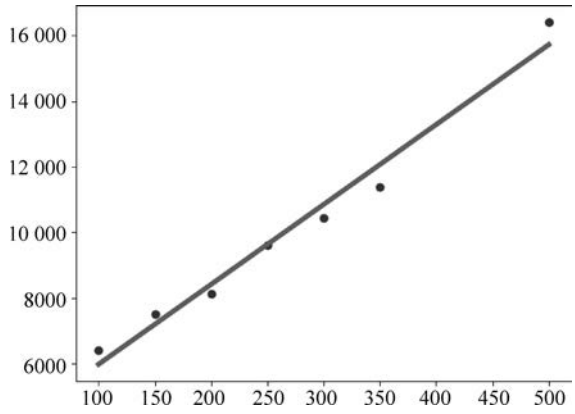

图 3-7　线性回归实验效果图

3.2.6　线性回归算法特点

线性回归是回归问题中最简单的一种，线性回归假设预测值与特征变量之间线性相关，即满足一个多元一次方程。在线性回归中，一般使用均方误差构建损失函数，使用梯度下降来求解损失函数。

线性回归算法的优点如下。

（1）由于线性回归算法模型简单，因此建模速度快，计算复杂度低，使得其在数据量大的情况下依然保持很快的运行速度。

（2）线性回归算法的模型十分容易理解，可以从权重系数直接看出每个特征对结果的影响程度，结果可解释性强，根据权重系数就可以对每个变量进行理解和解释，有利于决策分析。

（3）当预先定义好一些变量时，只需提供一个简单的线性回归模型，就能很好地拟合数据。

线性回归算法的缺点如下。

（1）线性回归算法要求变量之间必须呈线性关系，因此不能很好地拟合非线性数据。

（2）线性回归算法分离信号与噪声的效果不理想，容易出现过拟合现象，使用前需要手工去除不相关的特征。

（3）线性回归算法无法学习到数据集中的特征之间的交互关系，不能很好地表达高度复杂的数据，因此对于复杂问题线性回归算法表现的效果较差。

3.3　逻辑回归算法

线性回归模型是为了寻找输入样本数据 X 与输出结果 Y 之间的线性关系，即当因变量 Y 与自变量 X 之间满足 $Y=\theta X$ 的线性关系时，求解特征的系数向量 θ。如果 Y 是类似于天气温度、股票数值等连续值，因此称为回归模型。如果 Y 是邮件类别（非垃圾邮件、垃圾邮件）等离散值时，那么该如何处理呢？一种简单的实现是对回归模型求得的连续值 Y 进行

一次函数转换，记为 $g(Y)$，函数 g 的作用是将 Y 的连续值划分为离散区间，每个区间对应一个类别。例如，回归模型预测的是每日最高温度，而我们需要知道的是今日的气温情况，如果今日最高气温值大于 30℃时划分为类别“高温”，最高气温值为 20～30℃划分为类别“正常”，气温值为 5～20℃划分为类别“凉爽”，气温值为(－∞，5℃)划分为类别“寒冷”，可通过一个阶跃函数完成上述回归问题到分类问题的转换，y 为回归模型预测后的连续气温值，则 $g(y)$为如图 3-8 所示的函数。

$$g(y)=\begin{cases}0, & 30\leqslant y\\ 1, & 20\leqslant y<30\\ 2, & 5\leqslant y<20\\ 3, & y<5\end{cases}$$

图 3-8　阶跃函数示意图

其中，数值 0，1，2，3 分别对应高温、正常、凉爽、寒冷四种分类，如果结果类别只有两种，就是最常见的“二分类”问题。逻辑回归算法正是基于这个出发点被提出的，本节以二元逻辑回归为例介绍逻辑回归算法，多元逻辑回归的损失函数推导以及优化方法与二元逻辑回归类似。

3.3.1　逻辑回归模型

前面提到从线性回归到逻辑回归的转变，主要在于中间加了一个转换函数 $g(y)$，即

$$Y=g(\theta X)$$

对于二分类任务，可以首先将线性回归算法的输出的连续数值映射到[0，1]区间，然后选择一个阈值，例如 0.5，将预测值大于 0.5 的结果归为 1，反之归为 0(0 与 1 代表了两种分类)。在逻辑回归中，通常选用 Sigmoid 函数作为从连续值到离散类别之间的转换函数，其公式如下：

$$g(z)=\frac{1}{1+\mathrm{e}^{-z}} \tag{3.17}$$

Sigmoid 函数也称 Logistic 函数，其对应的函数图像如图 3-9 所示。

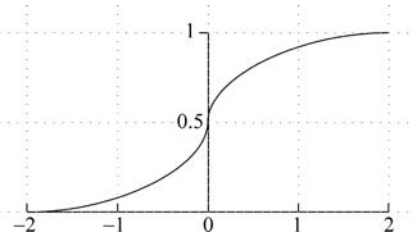

图 3-9　Sigmoid 函数

在逻辑回归算法中之所以使用 Sigmoid 函数作为转换函数，主要有以下两种原因。

(1) 首先，当 $z=0$ 时，$g(z)$的值为 0.5；当 $z>0$ 时，$g(z)$的值大于 0.5，且随着 z 值的增加，$g(z)$无限趋近于 1；当 $z<0$ 时，$g(z)$的值小于 0.5，且随着 z 值的减小，$g(z)$无限趋近于 0。

(2) 此外，Sigmoid 函数还有一个优秀的导数性质，即可以由其自身来表达、计算：

$$g'(z)=g(z)(1-g(z)) \tag{3.18}$$

这一导数性质使得在之后的计算中，对 Sigmoid 函数求导很容易得到。将 z 写作之前线性回归中的输入变量形式，即得到二元逻辑回归模型的一般形式：

$$H(x)=\frac{1}{1+\mathrm{e}^{-x\theta}} \tag{3.19}$$

3.3.2　逻辑回归损失函数

1. 为什么不能使用均方误差

为了描述 $H(x)$和真实值之间的差距，线性回归算法中采用均方误差作为损失函数可

以得到较好的拟合效果，而在逻辑回归中，如果也使用均方误差，其误差函数形式如下。

$$J(\theta)=\frac{1}{n}\sum_{i=1}^{n}(y_i-(H(X_i))^2 \tag{3.20}$$

$$H(x)=\frac{1}{1+e^{-x\theta}} \tag{3.21}$$

此时，如果将 Sigmoid 函数 $H(x)$带入到均方差损失函数中，所得到的是一个非凸函数，其函数图像可能存在多个极小值，如图 3-10 所示。此时，如果采用梯度下降来求解损失函数的最优参数，很大概率会陷入局部最优解中，因此就需要从另外的角度来设计逻辑回归算法的损失函数。

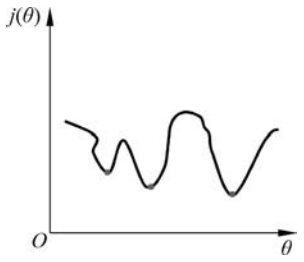

图 3-10　使用均方差误差函数的损失函数图像

2. 逻辑回归中的损失函数

在前面分析了由于使用均方误差作为逻辑回归算法的损失函数会使模型很容易陷入局部最优解中，因此在逻辑回归算法中通常使用一种称为交叉熵的损失函数。

假定输入样本 x，$\hat{y}$ 表示输入样本为 x 时预测结果 $y=1$ 的概率，则 $1-\hat{y}$ 表示输入样本为 x 时预测结果 $y=0$ 的概率，则有：

$$\begin{cases}\text{if } y=1\text{：}p(y\mid x)=\hat{y}\\ \text{if } y=0\text{：}p(y\mid x)=1-\hat{y}\end{cases} \tag{3.22}$$

通过指数形式，可以将两个式子合并如下（之所以能够这么合并，是因为 $y=0$ 和 $y=1$ 作为指数的独特计算结果）：

$$p(y\mid x)=\hat{y}(1-\hat{y})^{(1-y)} \tag{3.23}$$

为了方便求解，对等式两边同时取对数（由于对数函数是严格递增的函数，最大化取对数后的函数等价于最大化原函数），对式(3.23)等式两边取对数后可得：

$$\log p(y\mid x)=y\log(\hat{y})+(1-y)\log(1-\hat{y}) \tag{3.24}$$

将式(3.24)取反后作为损失函数，因此，最小化该损失函数等同于最大化上述对数函数。对于包含 n 个样本的训练集，服从独立同分布的样本的联合概率即为每个样本 i 的概率的乘积，因此有：

$$J(\theta)=-\log\prod_{i=1}^{n}p(y^i\mid x^i)=-\sum_{i=1}^{n}y^i\log(H(x^i))+(1-y^i)\log(1-H(x^i)) \tag{3.25}$$

至此，就得到了逻辑回归的损失函数。

3.3.3　梯度下降求解最优值

对于逻辑回归的损失函数 $J(\theta)$，仍然可以使用梯度下降法来求解参数 θ 的迭代公式（下面公式推导相对比较复杂，对于只想应用一些编程工具中现成的梯度下降方法已实现模型的求解，其实可以跳过不看）。首先，将逻辑回归模型函数看作如下函数：

$$H_\theta(x)=g(\theta^{\mathrm{T}}X)\ \frac{1}{1+e^{-\theta^{\mathrm{T}}X}} \tag{3.26}$$

通过梯度下降方式更新参数 θ：

$$\theta_j = \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}, \quad (j = 0, 1, \cdots, n) \tag{3.27}$$

使用链式求导的方式求解 $J(\theta)$对 θ 的偏导：

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial J(\theta)}{\partial g(\theta^{\mathrm{T}} X)} * \frac{\partial g(\theta^{\mathrm{T}} X)}{\partial \theta^{\mathrm{T}} X} * \frac{\partial \theta^{\mathrm{T}} X}{\partial \theta_j} \tag{3.28}$$

其中第一项：

$$\frac{\partial J(\theta)}{\partial g(\theta^{\mathrm{T}} X)} = y * \frac{1}{g(\theta^{\mathrm{T}} X)} + (y-1) * \frac{1}{1 - g(\theta^{\mathrm{T}} X)} \tag{3.29}$$

再根据 Sigmoid 函数的导数性质 $g'(z) = g(z)(1 - g(z))$可得第二项为：

$$\frac{\partial g(\theta^{\mathrm{T}} X)}{\partial \theta^{\mathrm{T}} X} = g(\theta^{\mathrm{T}} X)(1 - g(\theta^{\mathrm{T}} X)) \tag{3.30}$$

最后剩下第三项部分：

$$\frac{\partial \theta^{\mathrm{T}} X}{\partial \theta_j} = \frac{\partial J(\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n)}{\partial \theta_j} \tag{3.31}$$

将上述三项代入损失函数得：

$$\frac{\partial}{\partial \theta_j} J(\theta) = -(y - H_\theta(X)) x_j \tag{3.32}$$

由此可得，梯度上升公式为：

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} (y^i - H_\theta(x^i)) x_j^i \tag{3.33}$$

3.3.4 逻辑回归算法的正则化

与很多机器学习算法一样，逻辑回归算法也会面临过拟合的问题，因此在建立逻辑回归模型时也会考虑加入正则化项。逻辑回归算法的正则化方法与线性回归基本相同，即在损失函数后加入 L1 范数或 L2 范数，以及对范数添加一个系数 λ，来调节正则化项在损失函数中的影响力大小。

加入 L1 正则化后的逻辑回归损失函数如下。

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} [-y^i \log(H(x^i)) - (1 - y^i) \log(1 - H(x^i)) + \lambda \sum_{j=1}^{n} \| \theta_j \|] \tag{3.34}$$

加入 L2 正则化后的逻辑回归损失函数如下。

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} [-y^i \log(H(x^i)) - (1 - y^i) \log(1 - H(x^i)) + \lambda \sum_{j=1}^{n} \theta_j^2] \tag{3.35}$$

正则化后的逻辑回归损失函数对权重系数进行了约束，能够有效避免模型的过拟合，其求解方式与线性回归相同，这里不再赘述。

3.3.5 实验

数据准备：假设一个二分类的问题，根据两种特征来判断样本所属的类别，有如表 3-3 所示的输入数据。

表 3-3　逻辑回归实验输入数据表

编号	特征 1	特征 2	类别
1	−0.017 612	14.053 064	0
2	−1.395 634	4.662 541	1
3	−0.752 157	6.538 620	0
4	−1.322 371	7.152 853	0
5	0.423 363	11.054 677	0
…	…	…	…
100	0.317 029	14.739 025	0

主要代码如图 3-11 所示，首先定义一个线性模型函数 $H(x)=w_0+w_1x_1+w_2x_2$，其中，x_1 和 x_2 对应两种特征。线性模型函数的预测值 $H(x)$被 Sigmoid 函数映射到(0,1)区间，$H(x)$的值也反映了对于输入样本 x 预测结果为 1 的概率，假设阈值设定为 0.5，若 $H(x)$经 Sigmoid 函数处理后的值大于 0.5，将被分类为第一类，反之则被分类为另一类。

```
from numpy import *
filename='test.txt' #存取数据的文件名
def loadDataSet():   #读取数据
    datas = []
    labels = []
    fr = open(filename)
    for line in fr.readlines():
        lineArr = line.strip().split()
        datas.append([1.0, float(lineArr[0]), float(lineArr[1])])   #因为有两个特征，模型函数定义为W0+W1*X1+W2*X2
        labels.append(int(lineArr[2]))
    return datas,labels

def sigmoid(X):  #Sigmoid函数
    return 1.0/(1+exp(-X))

def gradAscent(datas, labels): #梯度上升求最优参数
    dataMatrix=mat(datas)
    classLabels=mat(labels).transpose()
    m,n = shape(dataMatrix)
    alpha = 0.001  #梯度下降的学习率
    maxCycles = 500 #设置迭代的次数
    weights = ones((n,1)) #初始化权重系数以及偏置
    for k in range(maxCycles):
        h = sigmoid(dataMatrix*weights)
        error = (classLabels - h)
        weights = weights + alpha * dataMatrix.transpose()* error #迭代更新权重系数
    return weights
```

图 3-11　逻辑回归算法代码

实验结果：逻辑回归模型拟合数据的效果如图 3-12 所示。由图可见，逻辑回归算法实际上是一个分类算法，而且是线性分类算法。本示例中，两种特征构成了二维平面，逻辑回归算法拟合的决策边界是一条直线，将平面划分为两个部分，每个部分对应一种类别。

3.3.6　逻辑回归算法特点

逻辑回归是在线性回归的基础上加了一个 Sigmoid 函数(非线性)映射，使得逻辑回归成为一个优秀的非线性分类算法，下面总结一下逻辑回归算法的一些特点。

逻辑回归算法的优点如下。

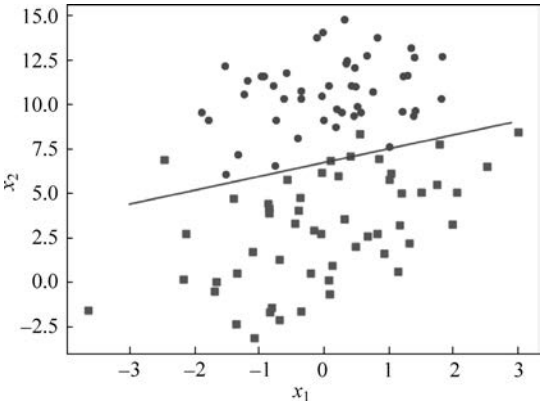

图 3-12　逻辑回归实验效果图

(1) 逻辑回归算法与线性回归类似,便于理解和实现,且广泛地应用于工业问题上。

(2) 由于分类时计算量非常小,逻辑回归算法通常速度非常快,需要存储的资源非常少。

(3) 对于每个样本,逻辑回归算法不仅可以预测出对应的类别,更是给出了样本分类的概率值。

(4) 通过正则化手段,可以有效地避免回归问题中常出现的多重共线性问题。

逻辑回归算法的缺点如下。

(1) 特征空间中的特征是对原始数据更高维的抽象,当特征空间很大,即特征数量多导致维度较高时,逻辑回归的性能并不理想。

(2) 由于逻辑回归模型结构简单,假设数据服从伯努利分布,很难拟合复杂数据的真实分布,因此容易出现欠拟合的情况,通常准确度不高。

(3) 尽管逻辑回归算法中引入了 Sigmoid 非线性变换,但是模型函数 $H_0(X)$并不是一个自变量 X 关于因变量 $H_\theta(X)$的函数,而是 $H_\theta(X)$关于 X 的后验概率(已知观测到的 X,预测其类别 $H_\theta(X)$),判断一个模型是否是线性,是通过决策边界是否是线性来判断的,以二分类为例,决策边界上的样本被划分为正负样本的概率相等,即通过令 $P(y=1|x,w)=P(y=0|x,w)$,可以求解得到决策边界方程为 $\theta^{\mathrm{T}}X+b=0$,因此逻辑回归算法本质上是一个线性的分类器,所以仍然不能很好地处理特征之间的相关性。

3.4　决策树

有监督学习方法(Supervised Learning)使用有输入和输出标签的数据训练模型,是在机器学习算法中应用最广泛、最有效的一种。换言之,在有监督学习方法中,我们知道样本数据的真实类别或者标签值,如果模型预测错误,这些值可以告诉模型正确的结果是什么,相当于起到了监督和纠正的作用。决策树算法(Decision Tree),又名判定树或分类树,由于该算法具备较高的预测准确度,同时又能给预测判定过程提供一定可解释性,因此在有监督学习算法中扮演着重要的角色。具体而言,决策树算法整体是一种根据数据特征采用树状结构构建的决策模型,根部节点是一条决策规则,后续的决策规则从根部向下面扩散,关于这里涉及的“后续决策规则”可以仍然是一条决策规则,也可以是不需

要再做决策的节点，即叶子节点，当决策树的所有分支都成为叶子节点时决策树的构造就完成了。决策树本质上是为了挖掘数据集中潜在的知识、规律、模式，其具体的挖掘过程即决策树的构建过程，当然，几乎所有的机器学习方法都可以看成是这样的数据挖掘或者知识学习的过程。

3.4.1　决策树的结构

在学习具体的决策树算法之前，首先了解一下决策树中的整体结构和涉及的必要概念。顾名思义，决策树是一种树状数据结构（如二叉树或者多叉树），主要由节点和边构成，其中节点又可以进一步分为如下三种类型。

（1）**根节点**（**root node**）：决策树根部的节点，最初的特征分裂点，也可以理解为第一个节点分裂规则。

（2）**内部节点**（**inner node**）：由根节点开始，向下延伸出的分叉节点，在功能上和根节点类似，属于一种节点分裂规则，内部节点继续向下分裂延伸又可以得到新的内部节点或者叶子节点。根据节点中特征属性不同而延伸出的边表示不同值域上的输出。

（3）**叶子节点**（**leaf node**）：当决策树构建完成后，处于分裂的最底层的节点是叶子节点，该类节点无法再选择特征（所有特征都分裂过）或者没有必要继续选择特征（节点内的所有样本都已经被划分到同一类别）进行分裂。

决策树的决策过程都是从根节点启动，每次选择一个特征进行分裂，不断生成子节点，直到当前叶子节点已经可以实现分类决策，如图 3-13 所示。

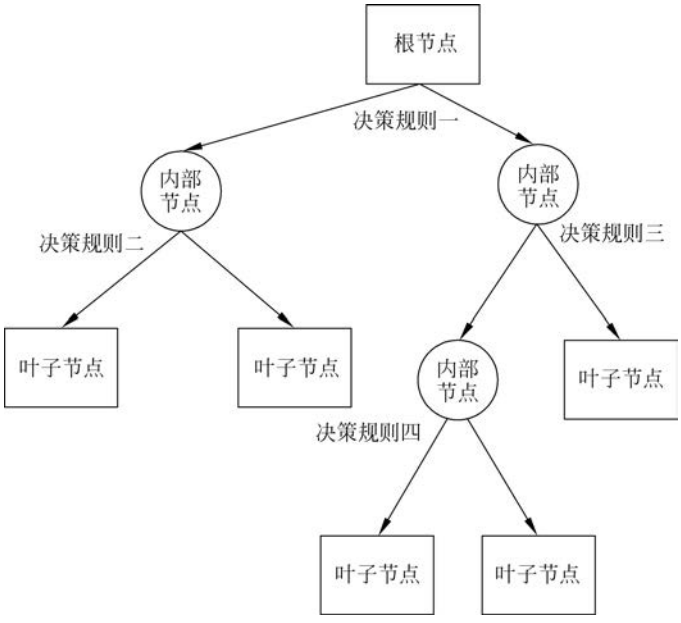

图 3-13　决策树算法模型图

决策树算法根据在树的构建过程中度量数据纯度的指标不同，主要可以分为以下三类：ID3、C4.5、CART，下面分别从分裂策略、剪枝策略、优缺点来详细介绍这三种决策树算法。

3.4.2 决策树算法

在决策树算法中，特征选择和子节点分支又称为决策过程(Decision)，是构建树的关键。信息论中有一句话——“**信息熵**(Information Entropy)越低，数据纯度越高”，生成决策树就是在使用某个特征对数据集进行划分后，使得信息熵较划分前降低，即数据集的纯度在划分之后变得更高。而度量数据信息熵(或者数据纯度)的指标有：**信息增益**(Information Gain)、**信息增益率**(Information Gain Ratio)和**基尼系数**(Gini Index)。使用这三种度量指标的算法分别就是 ID3、C4.5 和 CART(Classification And Regression Tree)这三种决策树算法。简单来说，决策树的构建过程即为：**根据数据集度量指标，迭代地选择划分标准(即什么特征，如果是连续性特征还需要考虑特征阈值)来对数据集进行划分，直到样本都能被划分到特定类别**。然而，当按照上述过程构造完决策树之后，经常会遇到过拟合的问题，即决策树在训练集上表现较好，但是在测试集上表现不佳。这个决策树的过拟合问题，一般可以通过对决策树进行适当的剪枝来缓解，剪枝后的决策树被称为子树。下面重点介绍不同的决策树划分标准和剪枝策略。

1. 信息增益(ID3 算法)

在介绍信息增益这个划分标准之前，需要先了解**信息熵**的概念。在信息论中，信息熵表示数据的“有序化”程度，为所有可能事件 x_i 发生概率的数学期望，则信息熵的计算公式定义如下。

$$H(x)=-\sum_{i=1}^{n}p(x_i)\log p(x_i) \tag{3.36}$$

其中，$p(x_i)$表示可能事件 x_i 发生的概率，信息熵 $H(x)$的值越小，则系统的纯度越高，即无序化程度越低。

前面已经说明过，决策树的生成过程就是使用某种特征对数据集进行划分，使得划分后的数据集的纯度比划分前的纯度更高。举一个很简单的例子，判断一个人是否是计算机专业的，这时可能会一头雾水不知道如何分辨，我们认为此时的系统很混乱、纯度低且信息熵高，但是如果给出一种特征划分依据——我们悄悄地跟这个人说：“计算机专业不就是修计算机的嘛”，然后观察这个人是否有点不悦，如果急了就说明此人大概率是计算机专业的，反之则不是，这样就有很大的把握来做出判断，一切都变得清晰明了了，此时我们认为系统不再混乱、纯度更高了且信息熵更低了。如何计算系统在特征划分前后的信息熵变化将是下面需要说明的问题。

在决策树划分过程中使用式(3.36)计算信息熵，划分前后的信息熵差值即为**信息增益**。那么选择不同的特征进行划分，会计算得到不同的信息增益，而使得信息增益最大的特征理论上是决策效果最优的，例如上面的小例子中的“激将法”，当然还可以选择其他的“激将法”——告诉他“计算机专业的尽头是秃头”，然后观察反应，究竟计算机专业的同学对哪种激将法更加敏感，使用信息增益就可以很好地度量。**ID3 算法**就是以“信息增益最大化”为目标来选择特征对数据集进行划分。我们使用如下公式来表达信息增益：

$$\text{Gain}(D,A)=H(D)-H(D\mid A) \tag{3.37}$$

其中，A 表示针对数据集 D 的一种特征划分方案，$H(D)$表示数据集 D 在进行特征划分之前的信息熵，$H(D|A)$表示这种特征划分后系统的信息熵，$\text{Gain}(D,A)$即该特征划分后的

降低的信息增益差值。让我们再回到上面的小例子中，A 其实就是我们实施的“激将法”，在实施 A 之前，无法确定目标人物 D 是否是计算机专业，因此 $H(D)$ 很大，但是通过“激将法”之后再观察他的反应就可以对他进行大致判定，在这样的条件 A 之下的系统信息熵 $H(D|A)$ 显然变小了很多，前后信息熵的差值就是实施“激将法 A”（某种特征划分）之后对目标人物的专业判定的自信程度。

2. 信息增益率（C4.5 算法）

将信息增益作为划分标准可以有效降低系统的信息熵，但是却更倾向于选择大量取值的特征作为划分标准，也就是说，如果数据集中某个特征的取值（或者是分段区间）非常多，即使该特征的决策能力很弱，如果将信息增益作为划分标准，该特征也可能会被作为最优的划分节点。举一个极端的例子：该特征下的每一个取值都对应了不同的类别，这种情况下的信息增益最大，然而决策树的预测精度将会很低，还是以 ID3 中判断某人是否是计算机专业为例，激将法“计算机专业的尽头是秃头”就类似于大量取值的特征，因为数学专业和物理专业的同学会对此产生严重质疑，同时少林寺的小师傅们也可能对此存在些许不满，这说明这样的特征划分策略没有很好的区分效果，但是使用信息增益作为划分依据的 ID3 很容易将这样的“激将法”作为特征划分依据，这显然是不合理的。为了解决 ID3 存在的问题，“信息增益率”被提出作为一种改进的特征选择标准，公式表达如下。

$$\text{GainRatio}(D,A)=\frac{\text{Gain}(D,A)}{H(A)} \tag{3.38}$$

其中，$\text{Gain}(D,A)$ 是 ID3 算法中使用的信息增益，$H(A)$ 表示特征 A 划分下的信息熵，如果特征 A 划分下的具体取值越多，$H(A)$ 的数值越大。可见，信息增益率 $\text{GainRatio}(D,A)$ 将信息增益 $\text{Gain}(D,A)$ 除以特征 A 划分下的信息熵 $H(A)$，这样就避免了直接选择取值可能多的特征作为划分节点了。使用信息增益率作为划分标准的决策树算法，被称为 **C4.5**。然而，C4.5 也存在缺陷，和 ID3 相反，C4.5 更倾向于选择取值较少的特征作为划分节点，解决该问题的一种常见做法是：先找出信息增益高于平均的特征，然后从这些特征中找出信息增益率最高的特征，这是一种启发式的（Heuristic）解决方案。

3. 基尼指数（CART 算法）

上面提到的 ID3 算法和 C4.5 算法分别以信息增益和信息增益率为特征划分依据，这两者都是基于信息熵模型的，在计算的过程中都涉及取对数（log），该操作相比于一般的四则运算计算复杂度更高，而本节即将介绍的 CART 算法使用的特征划分依据——基尼指数，虽然本质上也可以看成是对信息熵的计算，但是其计算过程回避了取对数操作，降低了计算负担。下面介绍基尼指数以及计算过程。

基尼指数（Gini Index）又名基尼不纯度（Gini Impurity），**CART 决策树生成算法**（Classification and Regression Tree）使用基尼指数作为划分标准，基尼不纯度的公式如下。

$$\text{Gini}(D)=\sum_{i=1}^{m}p_i(1-p_i)=\sum_{i=1}^{m}1-p_i^2 \tag{3.39}$$

其中，p_i 表示样本被分类正确的概率，基尼指数越小则表示样本被分类错误的概率越小，数据集的纯度越高。CART 选择基尼指数最小的特征作为最优的划分特征。

值得注意的是，和 ID3 和 C4.5 相比，CART 除了划分标准不同之外，本质上是二叉树，

而 ID3 和 C4.5 属于多叉树，分叉数对应了特征的取值个数，而这个构造上的区别也导致了 CART 树更加容易陷入过拟合问题，因为 ID3 和 C4.5 属于多叉树，要求所有特征最多只能被用来分裂一次，而 CART 是二叉树，对特征分裂次数没有限制，因此如果不对 CART 树进行剪枝操作，它就会进行无限分裂和过度膨胀，最后陷入过拟合。此外，CART 决策树从名字上就可以猜到它不仅能解决分类问题，还适用于回归问题，虽然事实确实如此，但是当需要解决一个回归问题时，很少会选择决策树类的算法，因为解决回归问题要求机器学习模型具有良好的泛化能力，而决策树类的算法优势在于对数据特征的记忆能力，其泛化能力较弱。

4. 决策树剪枝

根据特征选择准则对数据集进行划分构造决策树，使得决策树的每个叶子节点都对应一种类别，这样的决策树在训练集上表现很好，但在测试集上的效果往往较差，即模型的泛化能力不好，这种现象也是一种**过拟合**。为了避免过拟合，需要对决策树进行剪枝操作，基本的剪枝策略有：**预剪枝**(Pre-pruning)和**后剪枝**(Post-pruning)。

(1) **预剪枝**：预剪枝的方法有很多，例如，当决策树的高度达到预先设定的阈值时就停止树的进一步分裂；内部节点中样本数量小于某个预先设定的阈值也可以停止树的分裂；应用最广泛的剪枝方法是计算节点的进一步分裂给系统在判定精度上带来的增益，如果增益小于某个预先设置的阈值就停止该节点的分裂，将其设定为叶子节点。由于预剪枝策略是在决策树的生成过程中进行，没有必要等待完整决策树的生成，并算法简单高效，适合大规模数据集，然而预剪枝是一种基于“贪心策略”的剪枝方法，只考虑当前的模型效果，而不在乎后续分支是否可以进一步提升模型表现，使得算法过早地停止决策树的构造，因此预剪枝策略存在着“欠拟合”的问题。

(2) **后剪枝**：不同于预剪枝，后剪枝则是在决策树构建完成后，开始自底向上地对完整决策树中的非叶子节点进行考察，即验证将该节点对应的子树转换为叶子节点是否能够提升模型的判定精度，若这样的转换有效，则将该子树替换为叶子节点。相比于预剪枝，后剪枝可以实现更高的预测精度，然而后剪枝必须等待决策树的完整构建，因此算法效率较低。

3.4.3 决策树算法总结

决策树算法的优点如下。

(1) 决策树算法既可以用来解决分类问题(ID3，C4.5，CART)，也可以解决回归问题(CART)。

(2) 决策树算法的结构可以很自然地实现多分类问题，而大部分的方法都可以通过多次二分类来实现多分类任务。

(3) 输入到决策树算法的数据不需要额外的预处理，既可以处理离散值，也可以处理缺失值，即使数据中存在缺失值也不会影响决策树的生成和分类。

(4) 决策树结构原理简单直观，算法效率高，并且相比较于其他的机器学习方法，尤其是深度学习方法，决策树算法具有更好的可解释性。

决策树算法的缺点如下。

(1) 决策树算法容易陷入过拟合，并且模型泛化能力有限。

（2）决策树算法对数据敏感，当数据发生轻微变动时，可能会导致决策树结构的剧烈变动。

（3）决策树算法的预测结果更加倾向于数据集中分布比例更大的特征，针对该问题的一种可行解决方案是根据样本中特征分布比例分配对应的权重来平衡这个问题。

（4）决策树算法的泛化能力有限，难以捕获到一些复杂的关系或者模式，例如异或问题。

3.4.4　基于 sklearn 的决策树实验

1. 数据集介绍

为了便于演示，我们采用 Scikit-Learn 自带的鸢尾花数据集，并通过一个简单的实验来加深读者的认识，数据集的每个样本有 4 个属性：sepal length（萼片长度）、sepal width（萼片宽度）、petal length（花瓣长度）、petal width（花瓣宽度），单位都是厘米（cm）。样本标签包括 3 个品种类别：Setosa、Versicolour、Virginica，样本数量 150 个，每类 50 个，如表 3-4 所示。

表 3-4　鸢尾花数据集

	sepal length	sepal width	petal length	petal width	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
…	…	…	…	…	…
50	6.4	3.5	4.5	1.2	1
…	…	…	…	…	…

2. 实验部分

这里采用 load_iris()函数加载 Scikit-Learn 自带的 iris 数据集，然后直接调用 sklearn 包里面的 DecisionTreeClassifier 对数据集实现多分类，训练后使用 graphviz 库对决策树进行可视化，代码如图 3-14 所示。

decision-tree.py > ...

```
from sklearn.datasets import load_iris
import sklearn.tree as tree
import graphviz

iris = load_iris()
X, y = iris.data, iris.target
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("iris")

```

图 3-14　决策树分类代码

模型训练结束后的分类可视化结果如图 3-15 所示。

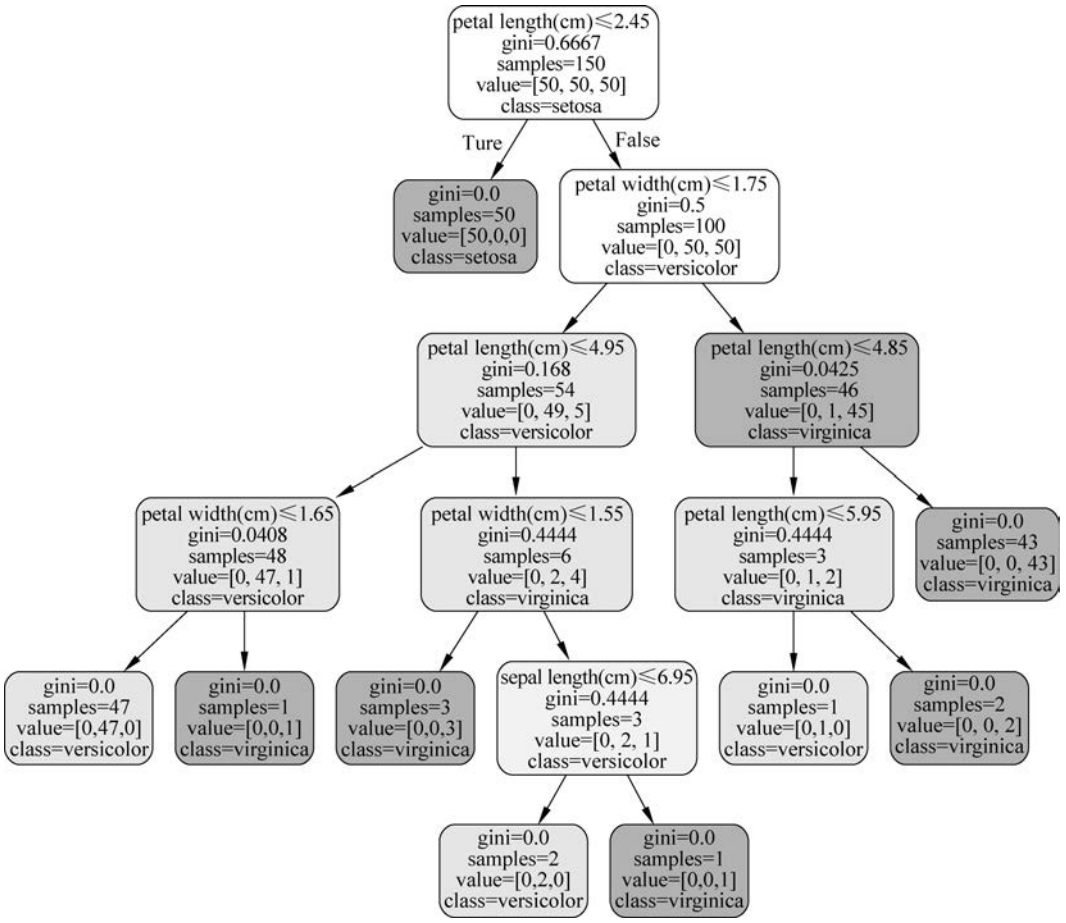

图 3-15　鸢尾花分类决策树效果图

3.5　朴素贝叶斯

和 3.4 节介绍的决策树算法不同，**朴素贝叶斯分类器**(Naive Bayes Classifier)主要依赖于统计学理论，有着坚实的数学理论支撑，从而保证了朴素贝叶斯模型具备较为稳定的分类精度。同时，朴素贝叶斯模型可以看成是一种统计模型，没有需要学习优化的参数，同时对于数据缺失问题并不敏感，算法稳定鲁棒性强，并且简单易用。

理论上来说，朴素贝叶斯模型相比于其他的分类方法应该具有更高的精度，但是实际应用中并非总是如此，主要原因是朴素贝叶斯模型为了简化模型计算提出了一个理想化的假设，即**数据集中各个属性(特征)之间是相互独立的**，不存在相关性。显然，这样的假设过于理想，在实际情况中是不成立的，因此降低了贝叶斯分类器的预测精度。由此，我们也可以联想到：朴素贝叶斯分类器的预测精度强烈依赖数据集中属性之间的相关性，若属性之间的联系较弱，分类器的预测表现就会好，反之，表现很差。

接下来在具体介绍朴素贝叶斯算法之前，先回顾一下和朴素贝叶斯算法相关的统计学

知识，此外，为了帮助读者更好地理解本节的内容，这里使用高校科研课题组中一位同学的失眠状况作为简单的案例，如表 3-5 所示。

表 3-5　实验室某同学连续 8 天失眠状况统计表

科研是否顺利-X_1	食堂饭菜是否好吃-X_2	社交是否愉快-X_3	是否颈椎酸疼-X_4	睡前是否有饥饿感-X_5	睡前是否思考人生-X_6	是否失眠-Y
0(否)	1(是)	1(是)	0(否)	1(是)	1(是)	1(是)
1(是)	0(否)	1(是)	1(是)	0(否)	0(否)	1(是)
1(是)	1(是)	1(是)	1(是)	0(否)	0(否)	0(否)
1(是)	0(否)	0(否)	1(是)	0(否)	1(是)	1(是)
0(否)	0(否)	0(否)	0(否)	0(否)	0(否)	1(是)
1(是)	1(是)	1(是)	0(否)	0(否)	0(否)	0(否)
0(否)	0(否)	0(否)	0(否)	1(是)	1(是)	0(否)
1(是)	0(否)	1(是)	1(是)	0(否)	0(否)	?

这里，$X_1 \sim X_6$ 是这位同学在一天中发生的 6 种情况，即模型的输入数据，这位同学晚上是否失眠 Y 就是每一条实例对应的结果或者标签(Label)。

3.5.1　朴素贝叶斯相关的统计学知识

朴素贝叶斯模型源自统计学中的贝叶斯学派，贝叶斯学派的思想可以简单表示为：先验概率＋数据≈后验概率。先验概率，即根据前人的或者自己日常积累的经验和分析得到的概率值(统计值)。通俗来说，先验就是人的常识，例如，掷骰子掷到 6 的概率是 1/6，这是一个常识，也是我们可以脱口而出的值；对于后验概率，即结果已经发生，求这个结果由某个因素引起的概率大小，简单来说，这个过程就是执果索因。例如，在上面给出的案例中，这位同学失眠了($Y=1$)是已经发生的结果，那么导致这个结果发生的因素可能是当天科研不顺利($x_1=0$)，也可能是因为颈椎疼($x_2=1$)，这两个因素分别导致失眠的概率 $P(X_1=0|Y=1)$ 和 $P(X_2=1|Y=1)$ 都是后验概率。

在实际的分类任务中，往往通过计算某个问题的后验概率来得到分类的概率，根据贝叶斯学派的假设，只要通过计算先验概率和充分利用已有数据就可以得到。严格来说，做到这一点并不简单，虽然数据很容易获得，但是如何确定某个问题的先验概率却是一个难题，因为先验概率往往是一种数据所在领域的经验知识，难以得到具体的量化。于是贝叶斯学派简化了先验概率的获取过程，直接假设模型服从某个特定的分布，显然这样的做法并不那么严谨，但是在大量的实际应用中，这样似乎还存在瑕疵的贝叶斯理论却展示出了很好的效果。下面介绍贝叶斯理论的相关知识。

条件独立公式，这里假设事件 X_1 和 X_2 是相互独立的，则有：

$$P(X_1,X_2)=P(X_1)P(X_2) \tag{3.40}$$

其中，$P(X_1)$ 和 $P(X_2)$ 分别表示 X_1 和 X_2 独立发生时的概率，$P(X_1,X_2)$ 表示 X_1 和 X_2 同时发生时的概率，即联合概率。这个公式的含义是：当 X_1 和 X_2 相互独立时，两者同时发生的概率等于两者各自发生的概率相乘。在上面表格的案例中，X_1 和 X_2 直观上没有什么关联，可以认为它们之间是条件独立的，但是，X_1 和 X_6 之间似乎存在较强的相关性，因为如果科研不顺利，睡前有较大可能会思考人生或者带有少些焦虑。

条件概率公式，这里不设置事件 X 和 Y 独立的假设，$P(X|Y)$表示先发生事件 Y，然后发生事件 X 的条件概率。由于 $P(X,Y)=P(X)P(Y|X)$，同理，$P(Y,X)=P(Y)P(X|Y)$，因为 $P(X,Y)=P(Y,X)$，最终可以得到如下贝叶斯定理。

$$P(Y \mid X)=\frac{P(X \mid Y)P(Y)}{P(X)} \tag{3.41}$$

将这个公式带入到上面的案例中，$P(Y|X)$是需要计算的概率，即在知道了这位同学的某种属性状态 X（假设此处仅考虑 X_1-科研是否顺利这一种情况）之后，失眠（$Y=1$）的概率。这个概率值可通过 $P(X|Y)P(Y)/P(X)$计算得到，其中，$P(X|Y)$就是前面提到的后验概率，表示发生了 Y（失眠）的条件下，由特征 X（科研不顺利）导致这个结果发生的概率，$P(Y)$表示所有观测样本中发生失眠的概率，$P(X)$表示所有观测样本中科研不顺利的概率。

全概率公式，如下：

$$P(X)=\sum_{m}\{P(X \mid Y_m)P(Y_m)\} \tag{3.42}$$

其中，X 表示观测数据（特征），在上面的案例中 X 可以进一步分为 6 个特征 $X_1 \sim X_6$，Y_m 表示观测样本对应的具体类别或者标签，在上面的案例中，$m=2$，即 Y_0 表示失眠，Y_1 表示不失眠，并且 $\sum_{m}P(Y_m)=1$，即所有类别各自发生的概率之和为 1。

根据上述一系列的公式，得到**贝叶斯公式（就是一个分类公式）**如下。

$$P(Y_m \mid X)=\frac{P(X \mid Y_m)P(Y_m)}{\sum_{m}\{P(X \mid Y_m)P(Y_m)\}} \tag{3.43}$$

根据上面的贝叶斯公式，可以很容易计算出在给定观测样本数据 X 的情况下，其所属类别为 Y_m 的概率值了。

3.5.2 朴素贝叶斯模型

朴素贝叶斯模型正是通过上述的贝叶斯公式来实现分类任务，只是朴素贝叶斯增加了一个并不严谨但是可以简化模型计算的假设，即各个样本数据的特征之间相互独立，也正是因为这个假设，所以该模型才被称为“朴素”贝叶斯。根据这样的假设，朴素贝叶斯的公式可以简单表示如下。

$$P(Y_m \mid X)=\frac{P(X \mid Y_m)P(Y_m)}{P(X)}=P(Y_m)\prod_{i=1}^{I}P(X_i \mid Y_m)/P(X) \tag{3.44}$$

其中，$P(X)$表示数据样本中某种特征存在的概率（也就是某种特征在所有统计样本中出现的频次，如科研顺利的频次、心情愉快的频次）。对于已知某个样本特征，计算其分别属于哪个类别（Y_0 还是 Y_1）时，本质是计算其属于哪个类别的概率值更大，而对于不同类别的 Y_m，$P(X)$的值在计算时是一样的，因此在具体的实现中，可以约去，这样的话，朴素贝叶斯分类器可以简化为简单的公式表示：

$$P(Y_m \mid X) \cong P(Y_m)\prod_{i=1}^{N}P(X_i \mid Y_m) \tag{3.45}$$

假设输入数据 X 中包含 N 个特征，直接用 X_i 表示样本 X 的第 i 个特征的某个特定取值。这样，只要统计出类别 Y_m 的概率 $P(Y_m)$，以及当样本的类别对应为 Y_m 时，各种特征

取值出现的概率，就可以计算出当前的观测样本属于类别 Y_m 的概率大小，以此类推，也可以统计出该观测样本属于其他类别的概率值，取最大的概率对应的类别即为样本的分类预测结果。

接下来，利用该朴素贝叶斯公式来预测上面提到的失眠案例，以验证上面的数学公式推导。

(1) 计算训练样本中每个类别的频率，得到：$P(Y_1)=4/6=2/3$；$P(Y_0)=1/3$。

(2) 计算每个类别条件下，各个特征属性划分的频率。

$P(X_1=1|Y_1)=1/2$；$P(X_1=0|Y_1)=1/2$；$P(X_1=1|Y_0)=2/3$；$P(X_1=0|Y_0)=1/3$；

$P(X_2=1|Y_1)=1/4$；$P(X_2=0|Y_1)=3/4$；$P(X_2=1|Y_0)=2/3$；$P(X_2=0|Y_0)=1/3$；

$P(X_3=1|Y_1)=1/2$；$P(X_3=0|Y_1)=1/2$；$P(X_3=1|Y_0)=2/3$；$P(X_3=0|Y_0)=1/3$；

$P(X_4=1|Y_1)=1/2$；$P(X_4=0|Y_1)=1/2$；$P(X_4=1|Y_0)=1/2$；$P(X_4=0|Y_0)=1/2$；

$P(X_5=1|Y_1)=1/4$；$P(X_5=0|Y_1)=3/4$；$P(X_5=1|Y_0)=1/3$；$P(X_5=0|Y_0)=2/3$；

$P(X_6=1|Y_1)=1/2$；$P(X_6=0|Y_1)=1/2$；$P(X_6=1|Y_0)=1/3$；$P(X_6=0|Y_0)=2/3$

(3) 预测，使用上面训练得到朴素贝叶斯分类器鉴别第 8 天该同学是否会失眠。该同学第 7 天的状态是：$X_1=1, X_2=0, X_3=1, X_4=1, X_5=0, X_6=0$：

$$P(Y_1|X)=P(Y_1)P(X_1=1|Y_1)P(X_2=0|Y_1)P(X_3=1|Y_1)P(X_4=1|Y_1)P(X_5=0|Y_1)P(X_6=0|Y_1)=2/3\times1/2\times3/4\times1/2\times1/2\times3/4\times1/2=0.0234$$

$$P(Y_0|X)=P(Y_0)P(X_1=1|Y_0)P(X_2=0|Y_0)P(X_3=1|Y_0)P(X_4=1|Y_0)P(X_5=0|Y_0)P(X_6=0|Y_0)=1/3\times2/3\times1/3\times2/3\times1/2\times2/3\times2/3=0.0110$$

显然，$P(Y_1|X)>P(Y_0|X)$，所以该朴素贝叶斯模型判定该同学第 8 天晚上又要失眠了，当然，为了方便读者理解，上面的案例数据较少，实际应用中应该收集足够多的样本，使得使用统计频率来代表先验概率更站得住脚。

值得注意的是，正是由于朴素贝叶斯假设各个特征之间独立，$P(X|Y_m)=\prod_{i=1}^{I}P(X_i|Y_m)$ 才得以成立，如果特征之间不独立，是存在相互关系的，那么想要计算某个类别下某个特征为特定值的概率就没有那么容易了，因为还需要考虑到其他所有的特征。因此可以发现，这个独立性假设虽然过于理想化，但是却能极大简化模型，提升了模型计算效率。

3.5.3　总结

朴素贝叶斯算法的优点如下。

(1) 朴素贝叶斯算法有着比较实在的统计学理论基础，也比较容易理解。

(2) 和很多深度学习方法需要大量的训练数据来进行优化不同，朴素贝叶斯可以在小规模数据集表现出不俗的分类精度。

(3) 和决策树不同，当数据发生轻微变动，朴素贝叶斯模型不会发生太大的变化，具有更好的稳定性。

(4) 朴素贝叶斯算法对数据缺失问题并不敏感，同时算法简单高效易用，在文本分类中使用广泛。

朴素贝叶斯算法的缺点如下。

从数学角度来说，如果使用贝叶斯公式来实现分类任务，其精度相比较于其他方法应当

具有最小的误差，但是由于朴素贝叶斯假设特征间相互独立，导致了在某些特征之间相关性高的场景下，模型表现不佳；相反，如果特征之间相关性很弱，朴素贝叶斯将具备很好的分类效果。

3.5.4 基于 sklearn 的 Naive-Bayes 实验

这里选用 Gaussian 朴素贝叶斯来作为模型，即假设每个特征在每个类别下的条件概率满足正态分布：

$$P(x_i \mid y)=\frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{3.46}$$

其中，μ_y 表示在样本类别 y 中，所有 x_i 的平均值；σ_y 表示在样本类别 y 中，所有 x_i 的方差。采用的估计方法是极大似然估计。

和决策树类似，我们选取鸢尾花数据集作为实验对象，先将数据集平均划分为训练集和测试集，然后用高斯朴素贝叶斯训练，测试后输出预测结果的正确数量，代码如图 3-16 所示。

gaussian-bayes.py > ...

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
clf = GaussianNB()
y_pred = clf.fit(X_train, y_train).predict(X_test)

correct = (y_test == y_pred).sum()
total = X_test.shape[0]
print("The correct of predictions: {}/{}".format(correct, total))

```

图 3-16 朴素贝叶斯分类代码

运行输出结果如下。

```
The correct of predictions: 71/75
```

3.6 神经网络

神经网络是一种在很久之前就开始被研究的算法模型，虽然早期取得了一定的进展，但是却因为计算能力和计算条件有限，陷入了一段很长时间的低谷。后来，随着硬件计算能力的快速提升，以及在深度学习研究上取得的进展，神经网络再次受到研究者们的关注，不同的神经网络变体也相继面世并且在各个机器学习任务上展示出优秀的表现。本节主要从思想和算法两个角度来介绍全连接神经网络及其变体卷积神经网络和循环神经网络。

3.6.1 神经元模型

神经元，作为神经网络中最基本的网络结构，其灵感主要来自生物学中的神经系统，在动物的神经系统中，神经元有两种状态：兴奋状态和抑制状态（其实相当于激活函数的作

用)。如果神经元受到刺激,就会从抑制状态转变为兴奋状态,该过程被称为“激活”,当这个神经元处于兴奋状态时,它就会向周围其他的神经元传递使自己兴奋的“消息”。图 3-17 是一个神经元的图示,左侧的须状物质是信号输入部位,这里的信号源可以是外界信息,也可能是其他神经元输出的信号(对应神经网络的特征输入,或者上一层网络提供给当前神经元的输入)。中间的白色管道状物质是信号处理部分,负责将输入的信号汇总并执行相关的处理。右侧的须状物质是信号输出部分,处理完的信号会通过该模块输出到其他的神经元。其实对于神经元模型,先要理解它是由数据输入、数据处理、数据输出三个部分组成。

图 3-17　生物学神经元示意图

下面将图 3-17 的神经元简化为模型图的形式,如图 3-18 所示,来详细讲解信号从输入到输出的过程。

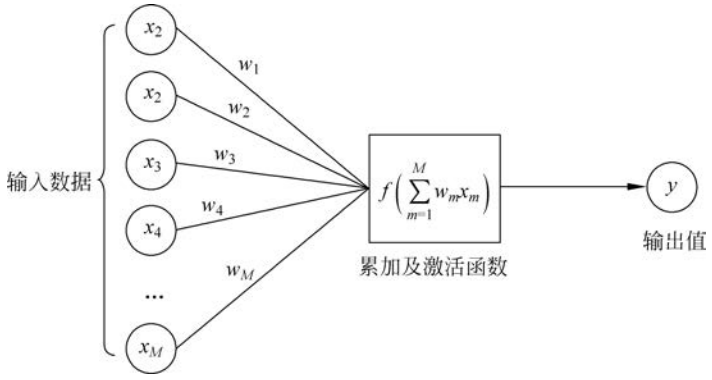

图 3-18　计算机科学中神经元模型图

首先,假设输入数据的特征有 M 维,即有 M 个不同的特征,表示为 $\{x_m \mid 1 \leqslant m \leqslant M\}$,对应于神经元模型的输入部分;接着将这些特征汇总,直观上来看,人对于不同的特征数据具有不同的感兴趣程度,所以在特征汇总的过程中,要为每一个特征加权,汇总后的特征会经过一个非线性的激活函数 $f(\cdot)$来学习这些特征中隐含的非线性关系,一般激活函数可以使用阶跃函数表示——大于阈值激活,反之处于抑制状态。但是,阶跃函数不连续不可导,在后续计算时不是很方便,因此一般使用 Sigmoid 函数 $\sigma(\cdot)$作为激活函数,其计算公式如下。

$$\sigma(\cdot)=\frac{1}{1+\mathrm{e}^{-x}} \tag{3.47}$$

Sigmoid 激活函数的图形其实很像一个阶跃函数,但是连续可导,如图 3-19 所示。

经过激活函数处理后的结果又作为其他神经元的输入数据传递到其他的神经元。神经元处理的过程可以使用如下公式简单表示:

$$\boldsymbol{Y}=\sigma(\boldsymbol{WX}) \tag{3.48}$$

其中,$\boldsymbol{X}$ 是输入数据的矩阵形式,$\boldsymbol{W}$ 是权重矩阵,$\boldsymbol{Y}$ 是输出数据的矩阵形式。

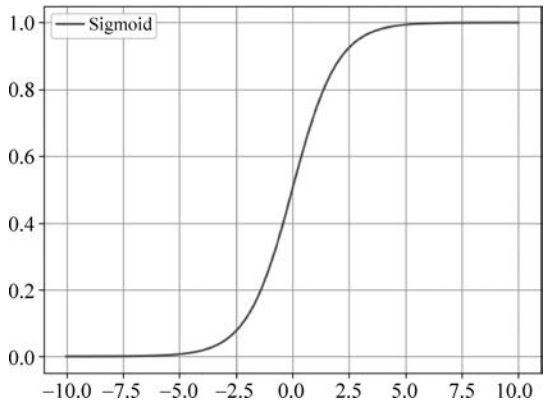

图 3-19 Sigmoid 函数图像

其实，除了 Sigmoid 函数之外，常用的激活函数还有很多，例如双曲正切函数 tanh，修正线性单元 ReLU 等，它们的公式如下。

$$\begin{cases}\tanh(x)=\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\\ \mathrm{ReLU}(x)=\max(0,x)\end{cases} \tag{3.49}$$

双曲正切函数的图像如图 3-20 所示。

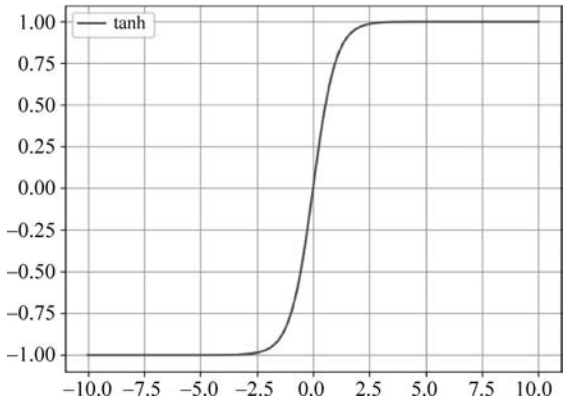

图 3-20 tanh 函数图像

修正线性单元的图像如图 3-21 所示。

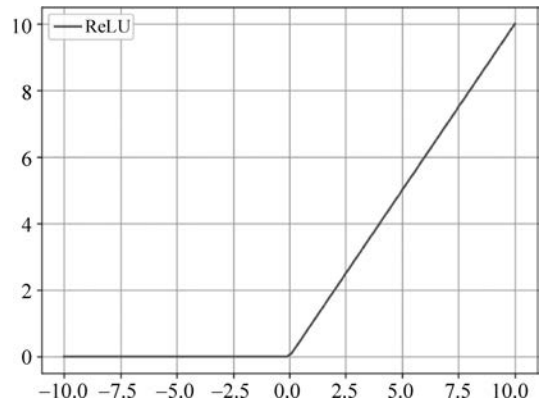

图 3-21 ReLU 函数图像

但是，我们注意到神经元的激活函数几乎都是非线性的，其实这并不难理解——使用非线性激活函数是为了增加神经网络模型的非线性因素，使得网络的能力更加强大，从而能够学习更复杂的数据。最终使得神经网络可以拟合任意非线性函数，关于这个结论的数学证明就不在这里赘述了，如果读者感兴趣，可以阅读相关论文[1]。

下面从数学角度简单解释一下，在神经网络中，为什么不用线性激活函数。假设某神经网络如图 3-22 所示。

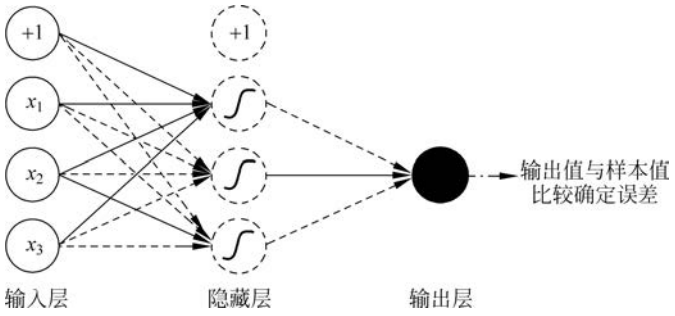

图 3-22　仅有 1 个隐藏层的神经网络

设输入 $x=[x_1,x_2,x_3]^{\mathrm{T}}$，输出为 $\hat{y}$，第 i 层的权重矩阵和偏差分别为 $\boldsymbol{W}^{[i]}$，$b^{[i]}$，激活函数为 $g^{[i]}$，权重计算值和激活函数值为 $z^{[i]}$，$a^{[i]}$。

则隐藏层满足：

$$\begin{cases} z^{[1]}=\boldsymbol{W}^{[1]}x+b^{[1]} \\ a^{[1]}=g^{[1]}(z^{[1]}) \end{cases} \tag{3.50}$$

输出层满足：

$$\begin{cases} z^{[2]}=\boldsymbol{W}^{[2]}a^{[1]}+b^{[2]} \\ a^{[2]}=g^{[2]}(z^{[2]}) \end{cases} \tag{3.51}$$

如果该网络的激活函数均为 $g(x)=x$，则 $a^{[1]}=z^{[1]}$，$a^{[2]}=z^{[2]}$，易得：

$$\begin{aligned} a^{[2]}&=\boldsymbol{W}^{[2]}(\boldsymbol{W}^{[1]}x+b^{[1]})+b^{[2]} \\ &=\boldsymbol{W}^{[2]}\boldsymbol{W}^{[1]}x+\boldsymbol{W}^{[2]}b^{[1]}+b^{[2]} \end{aligned} \tag{3.52}$$

显然，如果在**隐藏层**使用线性激活函数或者不用激活函数，那么神经网络只是把输入线性组合再输出。这样，无论神经网络有多少层，只是一直在重复线性计算，所以还不如直接去掉全部隐藏层。

3.6.2　全连接神经网络

1. 网络结构

神经元模型本质上就是一个**单层感知器**(Single-Layer Perceptron)，是最简单的人工神经网络。由于其输入层和输出层直接相连，只能用于处理线性问题，不能处理非线性问题。

多层感知器(Multi-Layer Perceptron，MLP)本质上就是在输入层与输出层之间添加了若干个隐藏层，且每一层之间都是全连接的(Fully-Connected)，即前一层的每个神经元会都与下一层的全部神经元连接。

相对于单层感知器，多层感知器输出端从一个变成了多个。此外，除了输入层，其余所

有层的节点，都是使用非线性激活函数的神经元，这样就克服了单层感知器针对非线性分类问题的弱点。

一个简单的 MLP 模型如图 3-23 所示。

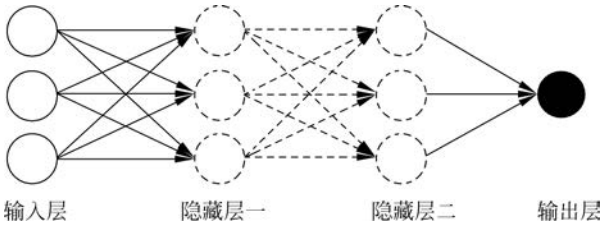

图 3-23　多层感知器模型

2. 网络优化——反向传播算法

神经网络中的参数主要包括：连接权重、网络层数、神经元个数、学习率 α 等。其中，只有神经元之间的连接权重能够通过网络的不断训练得到优化，也就是神经网络模型训练的最终目标；而其他的参数只能通过人工设置，称为超参数（Hyper Parameter），通过调整超参，能够改变模型训练的过程和最终收敛的效果。

在构建完神经网络之后，需要通过优化算法来更新神经网络中的参数，参数对应于网络中各层神经元之间的权重矩阵。因为在训练的每次迭代中，通过正向传播得到输出值之后，需要从输出层开始，反向逐层向输入层逼近，用链式法则求出每一层的梯度，所以该优化算法被称为反向传播算法（Backward Propagation）。反向传播算法是在神经网络中应用最广泛的优化算法，其主要流程如图 3-24 所示。

Algorithm 1: Back Propagation

Data: Training Set, Learning Rate α
Result: Network with weights updated
1 **foreach** *X in Training Set* **do**
2 　compute the output Y of the network whose input is X;
3 　compute the loss J;
4 　compute the gradients of hidden layer using **chain rule**;
5 　update weights matrix;
6 **end**

图 3-24　反向传播算法流程

3. 总结

全连接神经网络具有如下优点。

（1）神经网络具有较强的泛化能力，可以捕获到特征之间重要的非线性关系，这种非线性特征提取能力在现实问题中非常重要，因为大部分特征之间的相互关系都是高阶非线性的。

（2）具备自学习能力，在构建完神经网络和模型特征之后就可以自动进行模型的优化，不需要额外的基于领域知识的预处理操作。

全连接神经网络的缺点如下。

（1）神经网络的可解释性较差，整个网络的中间处理是一个类似于“黑盒”的过程，因此在一些需要可解释性的任务中无法使用神经网络模型，例如医疗预测等。

（2）神经网络的计算负担很大，通常来说，基于神经网络的算法都会比基于传统机器学

习的算法的计算复杂度更高。神经网络的计算负担往往取决于数据的大小，以及网络的深度和宽度。

(3) 神经网络的学习能力往往需要大量的训练数据支撑，因此对于只有很小规模数据量的数据集，朴素贝叶斯等算法往往可以表现出更优的分类效果。

3.6.3　卷积神经网络

在介绍卷积神经网络(Convolution Neural Network)之前，先了解一下一般的人类视觉原理，人眼首先获取外部输入信号(视觉数据，如图片)，接着对输入信号做出初步处理(学习一些边缘和方向特征)，然后对这些特征进行抽象(学习输入信号中的物体的大致形状，圆形或者方形)，接着进一步抽象(根据上一步抽象的结果做出判定，例如眼前的物品是猫或者狗)。图 3-25 是人脑进行对人脸、汽车、大象、椅子识别的一个示例[2]。

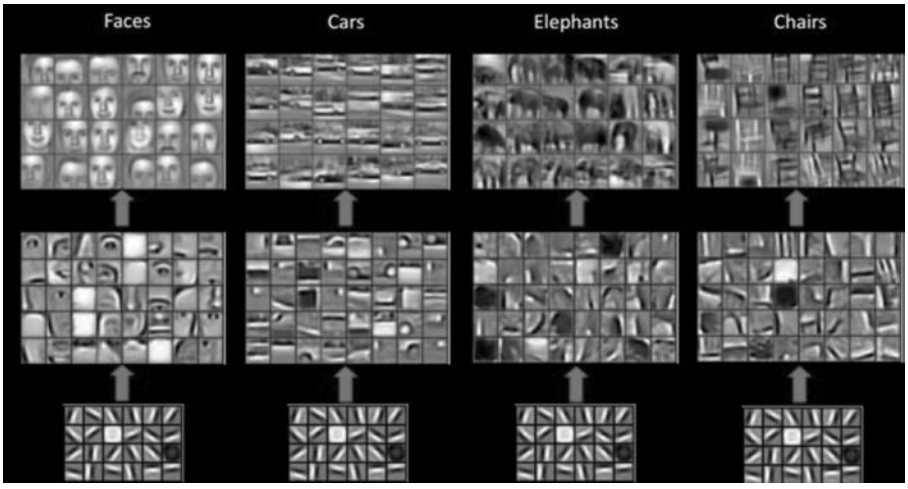

图 3-25　人脑对人脸、汽车、大象、椅子的分级识别过程示意图

可以发现，即使是针对不同的物品进行识别，人类的视觉原理也是通过这样的步骤来分级认知的。使用图 3-26 来表示人类识别大象的具体流程，具体来说，在最底层处理得到的特征(各种边缘和方向特征)基本上是类似的，接下来的特征处理，可以提取出此类物体的形状和局部特征(如大象鼻子、耳朵等)，到最后的处理步骤，上一步得到的各种局部特征通过组合形成相应的图像，从而实现准确的判定。

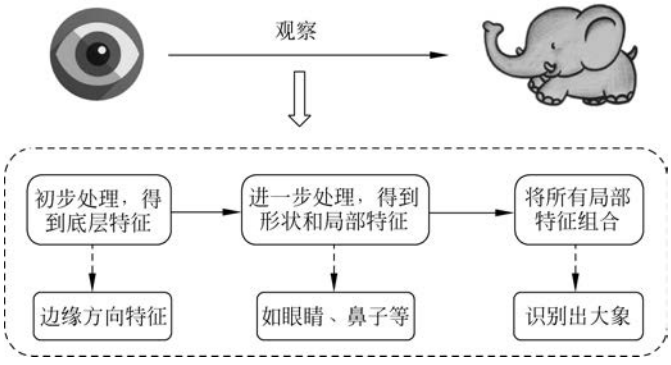

图 3-26　人脑对大象的识别过程的简易流程图

由上述人类视觉原理，计算机视觉科学家受到启发：可以让计算机来模仿人类大脑的视觉处理过程，从而构造出对应神经网络，在这样的网络结构中，底层网络部分实现初级图像特征的处理和提取，网络更高层部分则依赖于底层网络构造的初级特征来提取高级特征，最终通过不同类型特征的组合，在网络的最后一层做出判定和分类。事实上，这样的想法是可行的，并且卷积神经网络就是基于这样的灵感而被提出和设计的，下面将详细介绍卷积神经网络的模型结构。

1. 卷积神经网络的模型结构

卷积神经网络和全连接神经网络一样，是一种多层神经网络，但是由于其能够捕获空间相关性特征信息，更加擅长于处理视觉相关的分类问题。卷积神经网络通过一些有效的策略，成功地将图像识别中特征维度进行降维，解决了“维度爆炸”问题，从而实现有效的模型训练和预测。下面以早期最经典的卷积神经网络 LeNet 为例来讲解卷积神经网络的一般结构，如图 3-27 所示。

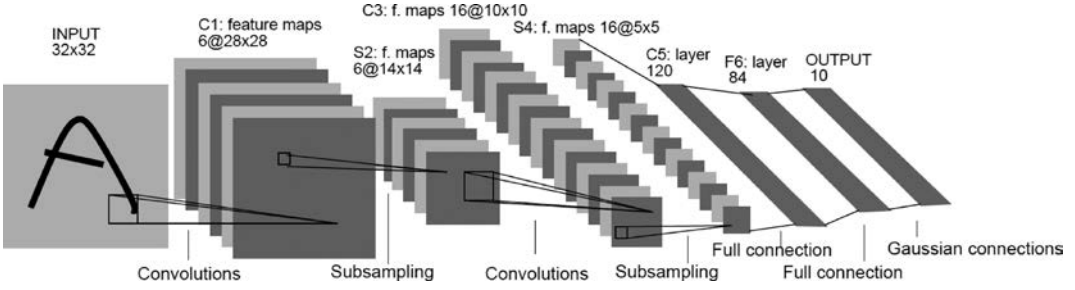

图 3-27 LeNet-5 卷积神经网络框架图[3]

由图 3-27 可以看出，卷积神经网络整体可以看成是由卷积层、池化层、全连接层这三部分组成。其中，卷积层与池化层实现特征的逐层学习和提取，卷积层是为了捕获图像局部特征，池化层主要是为了降低特征的维度，最后通过一个全连接神经网络实现分类任务。下面具体介绍卷积、池化等操作。

根据卷积神经网络的命名可以知道，卷积是卷积神经网络的核心操作。首先，需要了解卷积操作中的相关概念。

(1) 卷积核：是卷积神经网络中除了全连接神经网络之外，唯一需要学习和更新的参数，可以理解为一个滑动窗口，在特征图(Feature Map，也可以理解为卷积层的输入)上按照固定步长进行滑动，将特征图中相应的数值和卷积核对应的特征值做逐元素相乘并求和，权值即卷积核中每个位置上的数值。

(2) 步长：卷积核每次滑动的间距。

(3) 填充(Padding)：顾名思义，填充就是往输入特征矩阵中填充一些“假”特征，一般在特征矩阵的边缘填充 0，这样做的原因主要是防止不断的卷积操作使得图像特征图越来越小；其次，是为了让边缘的特征和中间的特征可以被卷积核捕获的频率保持一致。

(4) 池化：为了减少特征处理的计算量，同时缓解过拟合问题，对卷积后得到的特征进行裁剪。

假如给定一个 $m \times n$ 的卷积核 $\boldsymbol{W}$ 如图 3-28 所示。

$$\boldsymbol{W}=\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ & \cdots & & \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

图 3-28 $m \times n$ 卷积核示意图

使用该卷积核对输入图像 X 进行卷积操作，其过程可以

使用公式描述如下。

$$Z = w_1 x_1 + w_2 x_2 + \cdots + w_{mn} x_{mn} = \sum_{k=1}^{mn} w_k x_k = \boldsymbol{W}^{\mathrm{T}} X \tag{3.53}$$

卷积核中的每一个权值分别和图像 X 中对应位置上的像素值相乘后求和，就相当于一个加权求和的过程。然而这只是一次卷积运算，实际上，卷积神经网络对图像需要进行多次这样的卷积操作：卷积核按照步长在输入图像 X 上进行滑动，并且对覆盖的局部区域进行上述卷积运算，直到卷积核遍历完整个图像 X，计算得到的结果即为此次卷积过程的特征输出。一个标准的卷积过程如图 3-29 所示。

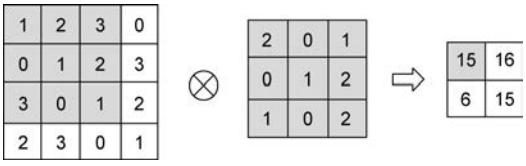

图 3-29　卷积过程示例

在图 3-29 中，卷积核大小为 3×3，每次会覆盖 3×3 的区域，将步长设置为 1，则卷积核总共会滑动 4 次，最后会得到 2×2 的输出特征。值得注意的是，神经网络在卷积层中设置的卷积核可能有很多个，每个卷积核都会输出这样的 2×2 的特征，即卷积核的数量对应了输出特征图的深度。

为了更好地理解卷积神经网络中的卷积操作，下面详细介绍卷积核的维度和网络输入输出大小的关系。首先，卷积核的输出特征和输入特征的尺寸关系表述如下。

$$\mathrm{output}_{\mathrm{size}} = \frac{\mathrm{input}_{\mathrm{size}} - \mathrm{kernel}_{\mathrm{size}} + 2 \times \mathrm{padding}}{\mathrm{stride}} + 1 \tag{3.54}$$

其中，output_size 和 input_size 分别对应输出特征和输入特征的尺寸，kernel_size 是卷积核的尺寸，stride 是步长，padding 表示需要在输入特征周围填充。卷积核的参数量也是和输入输出特征的通道数直接相关的，每个卷积核都和输入特征具有相同的通道数，每个通道分别进行卷积计算并求和，由此可见，卷积核的参数量为 out_channels(输出通道数)×in_channels(输入通道数)×kernel_size(卷积核尺寸)×kernel_size(卷积核尺寸)，相应地，卷积核的维度为(out_channels,in_channels,kernel_size_h,kernel_size_w)。

一般来说，在通过卷积层提取到特征表示之后，我们就迫不及待地利用特征来实现分类。虽然这样做理论上是可行的，但是巨大的计算量是不可避免的，同时模型容易陷入过拟合的问题，回想前面介绍的决策树算法也存在过拟合的问题，并且可以使用剪枝的策略将缓解过拟合，那么在卷积神经网络中是否也可以使用类似于剪枝的策略来解决过拟合的问题呢？池化操作很好地解决了这个问题，池化可以理解成是对输入特征中的局部区域的不同位置上的特征进行聚合统计，这里的聚合统计可以是统计出该区域中所有值的最大值，即最大值池化(Max Pooling)，也可以是统计出平均值，即平均值池化(Mean Pooling)，目前在深度学习的研究中，最大值池化是使用更为广泛且效果更好的池化方法。如图 3-30 所示是最大值池化的过程，平均值池化也是类似的过程。

其实池化过程和卷积过程有一点很相似，即都是一种窗口滑动的过程，然而这两个过程还是有着本质的区别。首先，池化过程是没有参数需要学习和更新的，其次，池化输出的特

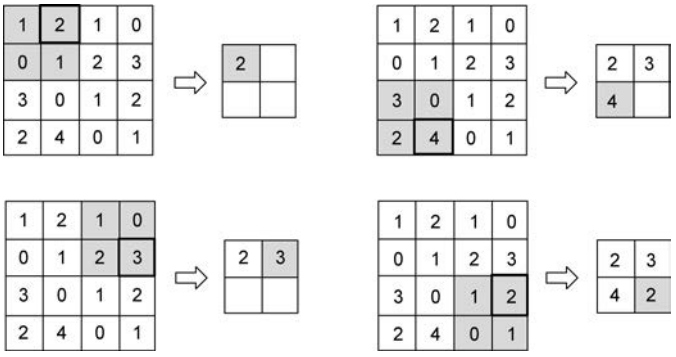

图 3-30 最大池化示例图

征虽然宽度和长度都降低了，但是深度没有改变，而卷积输出的特征图的深度和卷积核的深度是一致的。

上述过程通过卷积、池化后得到了初步的输出特征，但是这个过程还只是一种线性的过程，需要设置非线性激活函数来提高模型的非线性泛化能力，在卷积神经网络中，这个道理和全连接神经网络是一样的。不过，在卷积神经网络中一般使用 ReLU(Rectified Linear Unit)激活函数，相比较于全连接神经网络中用得比较多的 Sigmoid 激活函数在网络过深时容易发生梯度消失的问题，当输入大于 0 时，ReLU 激活函数的导数为 1，可以有效避免梯度无限接近零的梯度消失现象，同时，ReLU 激活函数还会使一部分神经元的输出为 0，提升了网络的稀疏性，减少了网络参数之间的依赖性，一定程度缓解了过拟合问题。

在通过上述多次卷积部分的特征提取(卷积＋池化)之后，卷积神经网络使用全连接神经网络来处理这些特征并实现具体的分类任务，一般在全连接的最后一层使用 Softmax 函数计算当前样本在所有分类上的概率分布。

2. 卷积神经网络总结

卷积神经网络的优点如下。

(1) 对于图像处理而言，卷积神经网络的模型结构相比于全连接神经网络更加合理，特征共享机制提升了模型的特征提取能力，从而实现了更高的分类精度。

(2) 和全连接神经网络相比，卷积神经网络需要学习更新的参数更少，一个完整的卷积神经网络大部分的参数都集中在卷积之后的全连接层中。因此，卷积神经网络可以设计得更深。

卷积神经网络的缺点如下。

(1) 和全连接神经网络一样，需要大量的训练数据才能保证模型的泛化能力。

(2) 可解释性差，整体模型的计算过程是一个“黑盒”。

3.6.4 循环神经网络

前面介绍的机器学习方法都是单独地去处理一个一个的输入数据，并且认为这些输入之间是独立的。但是实际上，在某些机器学习场景下，捕获不同输入之间的序列关系是非常必要的。例如，在自然语言理解研究领域，需要理解一句话的意思，显然单词之间的前后顺序也是非常重要的特征，因为有些时候如果句子中单词的顺序改变了，句子的语义也就发生了颠覆性的转变。例如，“先有鸡再有蛋”改变单词顺序后就变成了“先有蛋再有鸡”，显然两

个句子的含义是相反的。所以，为了解决类似这样的问题，捕获不同输入之间可能存在的顺序关系，或者为了更好地处理序列输入，循环神经网络（Recurrent Neural Network）应运而生。

1. 传统的循环神经网络结构

这一部分先介绍循环神经网络的整体模型结构，接着详细说明循环神经网络中数据处理过程。图 3-31 是循环神经网络的简单示意图。

在图 3-31 中，如果不看循环层 W，则网络的结构就和全连接神经网络完全一致。从图可见，指向 W 和从 W 发出的箭头形成了一个闭环，显然这是一个循环的过程，循环神经网络正是由此结构得以命名。将这样的闭环结构按照时间线展开后可以得到如图 3-32 所示的网络结构。当然，图中仅展示了 3 个时刻（$t-1$，t，$t+1$）输入的局部模型，根据这个局部模型，3 个时刻的输入分别是 x_{t-1}，x_t，x_{t+1}，每个输入都会通过中间的隐藏层并得到各自对应的输出 o_{t-1}，o_t，o_{t+1}，这样看起来就像是 3 个并行的全连接神经网络，但是这里将相邻的全连接神经网络进行关联，关联规则是将前一时刻的隐藏层提取的特征也作为下一时刻全连接神经网络的输入，这样，除了第一个时刻和最后一个时刻的全连接神经网络，其他的子网络都有两个输入：上一时刻的隐藏层，此刻的数据输入。

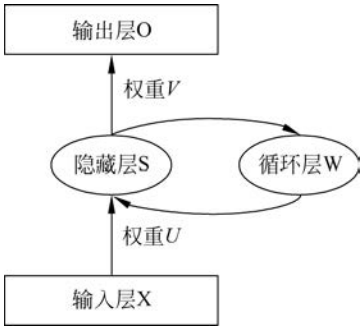

图 3-31　循环神经网络示意图

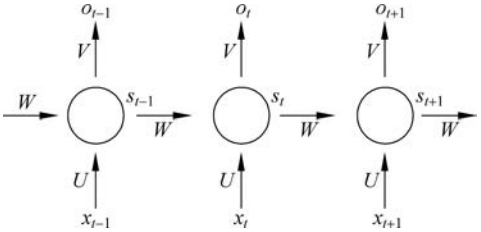

图 3-32　时间线展开后的循环神经网络结构图

接下来，使用公式来描述整体的循环神经网络的数据处理过程，具体如下。

$$\begin{cases} S_t = f(\boldsymbol{U} \cdot X_t + \boldsymbol{W} \cdot S_{t-1}) \\ O_t = g(\boldsymbol{V} \cdot S_t) \end{cases} \tag{3.55}$$

其中，S_t 表示在 t 时刻的隐藏层提取的特征，O_t 是在 t 时刻的输出，$\boldsymbol{U}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$ 是权重转换矩阵，是模型中需要学习和更新的网络参数。值得注意的是，在循环神经网络中，即使是在不同时刻的子网络中，$\boldsymbol{U}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$ 都是权值共享的，$f(\cdot)$ 是非线性激活函数，一般使用 tanh 激活函数，$g(\cdot)$ 是输出函数，一般使用 softmax 函数来计算类别概率。

2. 长短期记忆

尽管传统循环神经网络可以捕获序列数据中的时序特征，但是仍然存在着长期相关性的局限性，又被称为长期依赖问题。具体来说，循环神经网络中的节点经过若干阶段的计算之后，之前较早时刻处理的特征已经退化，或者说被覆盖，即随着序列数据时间片的推移，循环神经网络会逐渐丧失学习长期记忆的能力。然而，在很多机器学习的任务中，长期记忆和短期记忆分别发挥着不同的作用，缺一不可。基于此，一种循环神经网络的变体，长短期记忆网络（Long Short-Term Memory，LSTM）被提出。

为了更好地介绍 LSTM 的内部结构，可以通过和传统循环神经网络的对比来学习 LSTM 的模型细节。图 3-33 是传统循环神经网络的模型简图[①]。

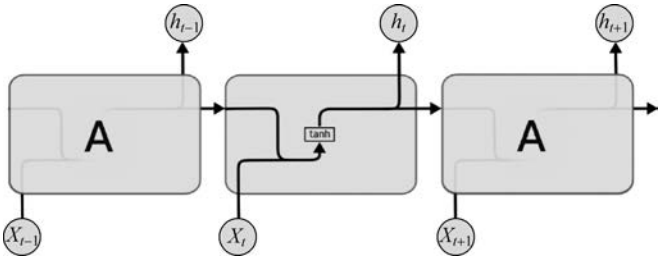

图 3-33　传统循环神经网络模型简图

显然，传统循环神经网络作为一个链式结构，对于每个时间段上的处理都非常简单，同时，在这样的处理单元中，上一时刻 $t-1$ 的输出和当前时间段的输入可以看成等效作为输入数据，并且二者一起经过当前单元中非线性激活函数处理后作为下一时刻 $t+1$ 的隐层输入特征。简单来说，上一时刻的记忆都会通过非线性的处理过程，随着网络的深度传播，这些网络处理单元堆叠了多次非线性操作，早期的记忆（早期的隐藏层的信息）被不断淡化，以至于网络会失去学习长期记忆的能力。

为了解决这样的问题，LSTM 专门设计了“遗忘门”机制，遗忘门会**线性保留**上一个时刻的记忆。具体来说，该模块会利用当前时刻的输入数据和上一时刻的隐层信息来计算一个概率值，来作为上一时刻记忆的保留比例，接着将保留下来的记忆和当前时刻产生的记忆合并，进入到下一时刻的处理中。遗忘门的处理如图 3-34 所示。

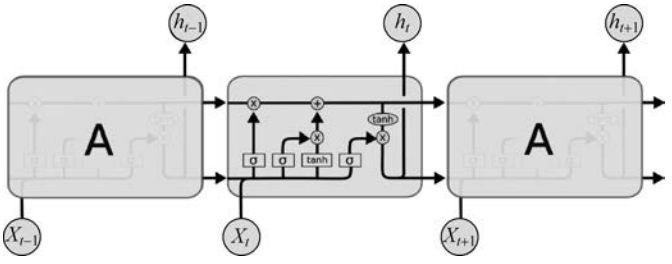

图 3-34　遗忘门处理示意图

其中，当前时刻的输入为 X_t，上一时刻的隐层信息和记忆信息分别为 h_{t-1} 和 C_{t-1}。注意，很多读者容易将隐层信息 h 和记忆信息 C 混淆，其实可以将记忆信息理解成是隐层信息的改进版本，前面也提到了，隐层信息无法保存长期的记忆，而记忆单元却可以，遗忘门的处理过程如下。

$$p_1 = \text{Sigmoid}(\boldsymbol{W}_1(h_{t-1} \oplus X_t) + b_1) \tag{3.56}$$

其中，$\boldsymbol{W}_1$ 和 b_1 表示处理数据的权重转换矩阵和偏置项，$\oplus$ 代表拼接操作，p_1 是一个介于 0 和 1 的概率值，p_1 的值决定了上一次记忆的保留比例，除此之外，还需要计算一个概率值 p_2 决定当前时刻输入数据在形成记忆时的保留比例。下面的公式表示，当前时刻完整记忆的形成过程：

① 图片来源于 https://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$\begin{cases} p_2 = \text{Sigmoid}(\boldsymbol{W}_2(h_{t-1} \oplus X_t) + b_2) \\ C_t = p_1 \cdot C_{t-1} + p_2 \cdot \tanh(\boldsymbol{W}_3(h_{t-1} \oplus X_t) + b_3) \end{cases} \tag{3.57}$$

在这个公式中，前一部分显然是对上一时刻记忆的保留过程，而后一部分 $p_2 \cdot \tanh(\boldsymbol{W}_2(h_{t-1} \oplus X_t) + b_2)$ 则表示当前时刻的数据 $(h_{t-1} \oplus X_t)$ 所形成的记忆信息。两个部分相加便是此刻全部的记忆数据。与上一时刻的处理过程一样，当前时刻的记忆数据还会继续向后一个时刻传播，并且迭代地进行上述过程。

在 LSTM 中，还需要考虑不同时刻的隐藏单元信息的生成，那 LSTM 如何生成当前时刻的隐藏层信息 h_t 的呢？h_t 表示当前时刻 t 的全部输出信息，如图 3-35 所示是隐藏层信息的生成过程，可以发现，h_t 的生成几乎用到了前面提及的所有数据，包括上一时刻的隐藏信息 h_{t-1}、此刻输入数据 X_t、上一时刻的记忆数据 C_{t-1}。

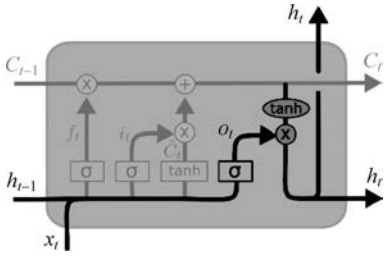

图 3-35　隐藏层信息生成过程

根据这张图，可以很直观地理解到一个事实：当前时刻单元隐藏层的生成主要依靠的是①当前时刻的记忆信息 C_t；②当前时刻的输入数据；③上一时刻的隐藏层信息，这些数据仅仅被用来计算一个概率值，这个概率值主要用于保留 C_t 经过非线性处理后的部分信息作为当前时刻的隐藏层信息，这个套路还是上面提及的遗忘门机制。我们使用下面的公式来表示这个过程。

$$\begin{cases} p_3 = \text{Sigmoid}(\boldsymbol{W}_4(h_{t-1} \oplus X_t) + b_4) \\ h_t = p_3 \cdot \tanh(C_t) \end{cases} \tag{3.58}$$

注意，时刻 t 的隐藏层信息 h_t 和记忆单元 C_t 有着本质的区别，C_t 表示前面所有时刻的记忆总和，通过前面的公式可以发现，C_t 的计算可以看成是前面所有时刻的记忆单元的线性的加权累加，因此其对于长期记忆的保存效果较好，而隐藏层信息 h_t 也可以被理解成记忆信息，然而却只是当前时刻的记忆，从公式 $h_t = p_3 \cdot \tanh(C_t)$ 可以看出，h_t 本质上是先将 C_t 通过 tanh 激活函数压缩到 $(-1,1)$，然后计算一个概率值 p_3 来过滤出 C_t 中和当前时刻相关的上下文信息，也就是说，当前时刻的隐藏层信息 h_t 是全局记忆信息 C_t 的一部分信息。

3. 门循环单元

门循环单元(Gate Recurrent Unit，GRU)可以理解为一种轻量级的 LSTM，因为 GRU 有着和 LSTM 非常接近的模型表现，但是模型却比 LSTM 更加简洁。GRU 的结构很大程度上基于 LSTM，是 LSTM 的一种变体，它将 LSTM 的记忆单元 C_t 和隐藏单元 h_t 合并成一个单元，称为“更新门”，当然也有一些其他的改进，鉴于其和 LSTM 的相似性，我们简要介绍 GRU 模型。

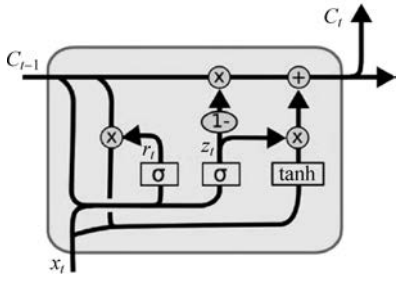

图 3-36　门循环单元模型图

下面配合简单的 GRU 示意图和对应的公式来说明 GRU 的模型细节。GRU 在当前时刻的模型图如图 3-36 所示。

GRU 将上一时刻的隐藏层(或者上一次时刻的输出信息)信息和需要线性保存的记忆合并在了一个新

的单元中，原因在于既然记忆单元中已经包含隐藏层信息（LSTM 小结的末尾提供了相关说明），那为什么还要保留隐藏层信息呢？于是在 GRU 的计算过程中就移除了隐藏层单元 h（这个陈述要给出一定的原因介绍，要不太生硬。这个“由于”是怎么来的?），为了方便理解，仍然使用 C 来表示这个合并的单元。首先，学习记忆的线性保留过程：

$$\begin{cases} p_1 = \text{Sigmoid}(\boldsymbol{W}_1(C_{t-1} \oplus X_t)) \\ p_2 = \text{Sigmoid}(\boldsymbol{W}_2(C_{t-1} \oplus X_t)) \\ C_t = (1-p_1) \cdot C_{t-1} + p_1 \tanh(\boldsymbol{W}_3(p_2 \times C_{t-1} \oplus X_t)) \end{cases} \tag{3.59}$$

显然，公式中的 $\tanh(\boldsymbol{W}_3(p_2 \times C_{t-1} \oplus X_t))$ 便是此刻利用输入数据和之前所有时刻记忆信息生成的此刻的记忆信息，由于 C_{t-1} 包含 $t-1$ 时刻及其之前时刻所有的记忆信息，因此使用 C_{t-1} 和 X_t 来生成一个概率值 p_2，用于保留 $t-1$ 时刻及其之前时刻的记忆 C_{t-1} 中的信息，并将该信息融入当前时刻记忆信息的生成中，即将 $p_2 \times C_{t-1}$ 和 X_t 拼接后通过非线性激活函数 $\tanh(\cdot)$ 来生成当前时刻的记忆信息，最后，将前一时刻线性保留下来的记忆信息 C_{t-1} 和当前时刻生成的记忆信息相加，得到了此刻完整的记忆信息。值得注意的是，C_{t-1} 的线性保留比例是 $(1-p_1)$，当前时刻记忆信息的保留比例为 p_1（LSTM 利用前一时刻隐藏层和当前输入计算得到的概率值作为保留比例），这是因为 GRU 认为，之前所有时刻的记忆和当前时刻生成的记忆应该是此消彼长的关系，因此直接使用 $(1-p_1)$ 作为 C_{t-1} 的保留概率值。

4. 总结

神经网络除了全连接神经网络、卷积神经网络、循环神经网络，还有很多其他的变体，本节主要介绍最常用的三种神经网络，分别适用于不同的场景，全连接神经网络具有较强的学习能力和泛化能力，可以看成是其他神经网络的基础模块，卷积神经网络适用于处理结构化的数据（图像），循环神经网络适用于处理时序数据，值得注意的是，在卷积神经网络和循环神经网络中，都用到了全连接神经网络作为数据处理的一个环节。尽管这些神经网络模型都具备强大的特征捕获能力，但是它们都存在一个不可避免的缺陷，即可解释性差，整体模型的计算和推理过程仍然是一个“黑盒”，因此对于需要较高可解释性的任务就需要对这些变体做进一步改造。

3.6.5 图神经网络

图神经网络[4]（Graph Neural Network，GNN）专门用于处理图结构数据，例如，来自社交网络、知识图谱等数据。推荐系统中大部分的信息具有图结构，例如，社交图、知识图谱、用户-项目的交互图、序列中的项目转移图等，使用图神经网络能够通过迭代传播来捕获高阶的交互信号，并且能够有效地整合社交关系、属性等辅助信息，因此可以将推荐问题转换为图中的链接预测问题。近两年来，随着图神经网络的发展，越来越多的研究将推荐问题转换成为图结构来解决。本节将重点介绍一下 GNN 的基本原理。

图神经网络可以通过节点间的信息传播捕获图结构的依赖，其主要的思想为：迭代地聚合邻域信息，并整合聚合后的邻域信息和自身节点的特征更新节点表征。本节将从时空领域解释说明图神经网络原理。对于一个给定无向图 $G=(V,E)$，其中，V 为顶点集合，E 为边集合，e_{uv} 表示 u 节点和 v 节点的边信息，定义 h_u 为 u 节点隐含特征表示，如图 3-37 所示。

图结构特征提取是直接使用图的拓扑结构，根据图的邻接信息进行信息收集。该方法需要设计聚合函数和更新函数，聚合函数用来聚合来自邻居的信息，更新函数用来融合邻居节点和中心节点。

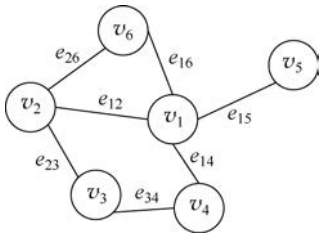

图 3-37　无向图示例

聚合函数：对于一条边 e_{uv} 包含两个端点 u 节点和 v 节点，同时 u 节点可以称为 v 节点的邻居，聚合函数把中心节点的所有邻居上一层特征进行聚合操作（如求和、加权平均等操作），定义第 k 层节点 v 的聚合函数为：

$$h_v^{(k)} = \text{AGGREGATE}^{(k)}(\{(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}) : u \in N(v)\})$$

其中，$N(v)$ 表示 v 节点的所有邻居集合，$h_v^{(k)}$ 表示 v 节点的邻居所包含的信息，$\text{AGGREGATE}^{(k)}(\cdot)$ 表示第 k 层的聚合操作（如求和、加权平均等）。

更新函数：通过聚合函数可以得到中心节点来自邻居的信息表征，我们通过更新函数来更新中心节点的最终表征，定义为：

$$h_v^K = \text{COMBINE}(h_v^{K-1}, h_v^{(k)})$$

其中，h_v^K 表示 v 节点在第 k 层最终表征，$\text{COMBINE}^{(k)}(\cdot)$ 表示第 k 层的更新函数（如拼接向量、加权平均等操作）。通过以上两个步骤可以得到节点 v 通过 K 层图神经网络之后的最后表征，然后应用到下游任务例如节点分类、链接预测等。

3.6.6　实验评估

1. 测试全连接神经网络、卷积神经网络在手写数字识别任务上的表现

为了评估全连接神经网络、卷积神经网络的模型表现，我们选择 MNIST 手写数字识别数据集进行实验。MNIST 数据集中有样本数为 60 000 的训练集和样本数为 10 000 的测试集，主要包括图片和标签，图片是 28×28 的像素矩阵，标签是 0～9 的数字。我们使用百度的 PaddlePaddle 框架分别实现全连接神经网络和卷积神经网络，通过全连接神经网络和卷积神经网络在 MNIST 数据集上展现出了准确率作为评估指标。

首先，实现全连接神经网络模型主要代码如图 3-38 所示，具体来说，我们构造了一个有 3 个隐藏层的网络，神经元个数分别为 300、100、10，前两层使用 ReLU 作为激活函数，最后一层使用 softmax 函数输出分类结果。

接着，实现卷积神经网络模型主要代码如图 3-39 所示，其中设置了两层卷积＋池化操作，最后使用 3 层全连接层来输出分类结果，输出层使用 softmax 激活函数，其余中间层使用 ReLU 激活函数。

最后，使用 MNIST 数据训练和测试这两个模型，具体代码如图 3-40 和图 3-41 所示，这里主要是对卷积神经网络进行训练和测试，将 model＝CNN()替换为 model＝MLP()时就可以对全连接神经网络模型进行训练和测试。

通过训练和测试，得到全连接神经网络和卷积神经网络在 MNIST 数据集上的评估结果，准确率分别为 0.9653 和 0.9835，同时，CNN 网络中的模型参数相比于全连接神经网络要少很多，却可以实现更高的图像分类精度，这表明 CNN 更加适合处理图像分类任务。

```
class MLP(paddle.nn.Layer):
    def __init__(self):
        super(MLP, self).__init__()
        #第一层全连接层，神经元个数为300
        self.linear1 = paddle.nn.Linear(in_features=28*28, out_features=300)
        #第二层全连接层，神经元个数为100
        self.linear2 = paddle.nn.Linear(in_features=300, out_features=100)
        #第三层全连接层，神经元个数为10
        self.linear3 = paddle.nn.Linear(in_features=100, out_features=10)

    def forward(self, x):
        #将图像输入数据展开
        x = paddle.flatten(x, start_axis=1, stop_axis=-1)
        #将展开后的数据输入到第一层全连接层并经过ReLU激活函数
        x = self.linear1(x)
        x = F.relu(x)
        #经过第二层全连接层和ReLU激活函数
        x = self.linear2(x)
        x = F.relu(x)
        #输出层使用softmax函数计算该数据在十个类别上的预测概率
        x = self.linear3(x)
        x = F.softmax(x)
        return x
```

图 3-38　Paddle 实现的全连接神经网络模型代码

```
class CNN(paddle.nn.Layer):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = paddle.nn.Conv2D(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2)
        self.max_pool1 = paddle.nn.MaxPool2D(kernel_size=2,  stride=2)
        self.conv2 = paddle.nn.Conv2D(in_channels=6, out_channels=16, kernel_size=5, stride=1)
        self.max_pool2 = paddle.nn.MaxPool2D(kernel_size=2, stride=2)
        self.linear1 = paddle.nn.Linear(in_features=16*5*5, out_features=120)
        self.linear2 = paddle.nn.Linear(in_features=120, out_features=84)
        self.linear3 = paddle.nn.Linear(in_features=84, out_features=10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = self.max_pool2(x)
        x = paddle.flatten(x, start_axis=1,stop_axis=-1)
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        x = self.linear3(x)
        x = F.softmax(x)
        return x
```

图 3-39　Paddle 实现的卷积神经网络模型代码

2. 测试 LSTM 和 GRU 在 ChnSentiCorp 情感分类任务上的表现

为了评估 LSTM 和 GRU 的模型表现，我们在 ChnSentiCorp 情感分类数据集上（PaddlePaddle 框架直接提供该数据集）进行对比实验，该数据集是一个短文本情感二分类数据集，包括训练集和测试集，训练集包括 9600 条评论，测试集包括 1200 条评论。同样地，我们使用百度开源的代码实现 LSTM 和 GRU 模型，以分类准确率作为评估指标。

```
#用 DataLoader 实现数据加载
train_loader = paddle.io.DataLoader(train_dataset, batch_size=64, shuffle=True)

model = CNN()

model.train()

#设置迭代次数
epochs = 5

#设置优化器
optim = paddle.optimizer.Adam(parameters=model.parameters())
#设置损失函数
loss_fn = paddle.nn.CrossEntropyLoss()

#迭代训练
for epoch in range(epochs):
    for batch_id, data in enumerate(train_loader()):
        x_data = data[0]            #训练数据
        y_data = data[1]            #训练数据标签
        predicts = model(x_data)    #预测结果
        #计算损失
        loss = loss_fn(predicts, y_data)
        #计算准确率
        acc = paddle.metric.accuracy(predicts, y_data)

        #反向传播
        loss.backward()

        #可视化打印
        if (batch_id+1) % 900 == 0:
            print("epoch: {}, batch_id: {}, loss is: {}, acc is: {}".format(epoch, batch_id+1, loss.numpy(), acc.numpy()))

        #更新参数
        optim.step()

        #梯度清零
        optim.clear_grad()
```

图 3-40　模型的训练代码

```
#加载测试数据集
test_loader = paddle.io.DataLoader(test_dataset, batch_size=64, drop_last=True)
#设置损失
loss_fn = paddle.nn.CrossEntropyLoss()

model.eval()

for batch_id, data in enumerate(test_loader()):

    x_data = data[0]            #测试数据
    y_data = data[1]            #测试数据标签
    predicts = model(x_data)    #预测结果

    #计算损失
    loss = loss_fn(predicts, y_data)

    #计算精度
    acc = paddle.metric.accuracy(predicts, y_data)

    #打印信息
    if (batch_id+1) % 30 == 0:
        print("batch_id: {}, loss is: {}, acc is: {}".format(batch_id+1, loss.numpy(), acc.numpy()))
```

图 3-41　模型的测试代码

首先，实现 LSTM 模型主要代码如图 3-42 所示。

接着，实现 GRU 模型的主要代码如图 3-43 所示。

最后，利用 ChnSentiCorp 情感分类数据集来训练和测试 LSTM 和 GRU 模型。LSTM 和 GRU 模型在 ChnSentiCorp 数据集上呈现出的最优分类准确率分别为 0.8967 和 0.8917，两者在情感分类任务上的模型表现较为接近，值得一提的是，相比较于 LSTM，由于 GRU 将隐藏层和记忆单元合并从而简化了模型结构，其计算负担更小。

```
class LSTMModel(nn.Layer):
    def __init__(self,
                 vocab_size,
                 num_classes,
                 emb_dim=128,
                 padding_idx=0,
                 lstm_hidden_size=198,
                 direction='forward',
                 lstm_layers=1,
                 dropout_rate=0.0,
                 pooling_type=None,
                 fc_hidden_size=96):
        super().__init__()
        self.embedder = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=emb_dim,
            padding_idx=padding_idx)
        self.lstm_encoder = nlp.seq2vec.LSTMEncoder(
            emb_dim,
            lstm_hidden_size,
            num_layers=lstm_layers,
            direction=direction,
            dropout=dropout_rate,
            pooling_type=pooling_type)
        self.fc = nn.Linear(self.lstm_encoder.get_output_dim(), fc_hidden_size)
        self.output_layer = nn.Linear(fc_hidden_size, num_classes)

    def forward(self, text, seq_len):
        embedded_text = self.embedder(text)
        text_repr = self.lstm_encoder(embedded_text, sequence_length=seq_len)
        fc_out = paddle.tanh(self.fc(text_repr))
        logits = self.output_layer(fc_out)
        return logits
```

图 3-42　Paddle 实现的 LSTM 模型代码

```
class GRUModel(nn.Layer):
    def __init__(self,
                 vocab_size,
                 num_classes,
                 emb_dim=128,
                 padding_idx=0,
                 gru_hidden_size=198,
                 direction='forward',
                 gru_layers=1,
                 dropout_rate=0.0,
                 pooling_type=None,
                 fc_hidden_size=96):
        super().__init__()
        self.embedder = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=emb_dim,
            padding_idx=padding_idx)
        self.gru_encoder = nlp.seq2vec.GRUEncoder(
            emb_dim,
            gru_hidden_size,
            num_layers=gru_layers,
            direction=direction,
            dropout=dropout_rate,
            pooling_type=pooling_type)
        self.fc = nn.Linear(self.gru_encoder.get_output_dim(), fc_hidden_size)
        self.output_layer = nn.Linear(fc_hidden_size, num_classes)

    def forward(self, text, seq_len):
        embedded_text = self.embedder(text)
        text_repr = self.gru_encoder(embedded_text, sequence_length=seq_len)
        fc_out = paddle.tanh(self.fc(text_repr))
        logits = self.output_layer(fc_out)
        return logits
```

图 3-43　Paddle 实现的 GRU 模型代码

参考文献

[1] Hornik, Kurt, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators [J]. Neural networks. 1989: 359-366.

[2] Zhuo L, Zhu Z, Li J, et al. Feature extraction using lightweight convolutional network for vehicle classification[J]. Journal of Electronic Imaging, 2018, 27(5): 051222.

[3] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE. 1998, 86(11): 2278-2324.

[4] Zhou J, Gui G, Hu S, et al. Graph neural networks: A review of methods and applications[J]. AI Open, 2020, 1: 57-81.