机器学习

主编：严晓东
副主编: 陈华 王国长 赵旋

编　　者

- 第一章：赵旋
- 第二章：胡琴琴
- 第三章：张梅
- 第四章：国忠金
- 第五章：王重仁
- 第六章：王国长
- 第七章：周峰
- 第八——十二章：严晓东
- 第十三章：句媛媛
- 第十四章：严晓东
- 第十五章：陈华
- 编辑秘书：呼亚楠；崔文海；吉晓婷；陈熙；连蓓蓓；封文丽；吴淑琦

序　言

随着新一代科学技术的发展，人工智能（Artificial Intelligence，AI）技术已经在各领域，包括科学、社会、经济、管理等方面，体现出解决各类复杂问题的优势，得到广泛认可，已经成为创新驱动发展的核心驱动力之一。大部分人工智能技术在应用落地方面得到很好的应用，尤其是 AI 中的机器学习日益成为一门通用的技能。最近有一些统计学家逐渐从大数据统计方法研究转入到前沿机器学习的科学问题的挖掘，例如基本原理、数学理论等存在的诸多瓶颈，影响着“可解释、可通用”的下一代机器学习的开发。华为创始人兼 CEO 任正非在接受央视《面对面》采访时特别强调，“人工智能就是统计学，计算机与统计学就是人工智能”，也间接强调了统计学在研究机器学习技术原理与理论上的重要地位。每一个学习者不能因为原理不懂而放弃使用机器学习方法的机会，这也正是写作本书意义和出发点。尽量使用统计语言介绍机器学习与大数据分析方法，做到详尽的数学推导，直观的文字解释，便捷的程序实现，给读者愉悦的学习体验，迎合普通高等学校理工类以及经济管理类等高年级本科生和研究生的基础知识储备。

本书的主要应用价值体现在机器学习方法与程序操作相结合。学习机器学习的最主要目的之一就是应用，应用离不开软件操作。同时，软件操作又有助于对机器学习方法的理解。R 语言为统计学家的母语，擅长于统计计算，Python 语言更偏于计算机及工程类专业使用的软件，因此 R 与 Python 语言是机器学习的两大常用语言。为此，本书提供“双语”服务，在介绍每个机器学习方法之后，最后章节详细介绍相应的 R 与 Python 语言实操。

目前市场上已出版了不少机器学习的书籍，再增加这一本书主要原因在于，计算机科学家的机器学习教材强调算法，使用“伪代码”(pseudo codes) 来介绍方法，从实际工程 (engineering) 问题的角度讲解方法的使用。统计学家的机器学习教材一般缺少大数据统计方法的介绍，例如解决超高维数据的特征筛选、流数据的在线学习以及模型平均的集成学习等方法讲解。另外，本教材在一些概念上进行了混用，比如统计学里的变量、响应变量、协变量、离散响应变量等分别对应机器学习里的特征、因变量、自变量、标签等，但是并不影响我们的理解。

本教材依据“数据中国”的国家战略需求，针对“统计学”，“数据科学与大数据技术”等专业对《机器学习》教学资源的需求，依靠山东省大数据研究会大数据专业建设委员会搭建了大数据教材编写团队，形成了《机器学习》的编写队伍。山东国家应用数学中心、山东大学金融研究院等十多个教学科研平台也成为本教材编写的重要支撑。

严晓东

2022 年 06 月 06 日

内容提要

本书主要围绕机器学习以及大数据分析的核心方法，进行深入而详细的讲解，特别关注方法理论推导以及程序实现，特色在于机器学习和大数据分析方法的统计语言介绍，并配有较多的插图、例子来直观地解释机器学习的原理。同时，结合流行的 R 与 Python 语言，介绍方法的程序实现，为读者提供“双语”服务。本书的机器学习和大数据分析方法配有详尽的数学符号与公式，并辅以直观的文字解释、细致的数学推导解读复杂的机器学习原理，做到删繁就简，娓娓道来，逻辑清晰。本书适合普通高等学校理工类以及经济管理类等高年级本科生和研究生使用。先修课包括数学分析、高等数学、线性代数、概率论与数理统计、统计学等以及 R 语言与 Python 语言的基本操作。

本书主要介绍常用机器学习方法，具体包括无监督学习、监督学习、集成学习、在线学习、机器学习预测以及大数据分析。

无监督学习

包括第一章与第二章，主要介绍两种无监督机器学习算法：聚类分析与主成分分析。

第一章：聚类分析。详细介绍了几种较为常用的聚类算法及其 python 语言与 R 语言代码。

第二章：主成分分析。介绍了线性主成分分析算法与非线性主成分分析算法，并利用 python 语言与 R 语言进行主成分分析实践。

监督学习

包括第三章，第四章，第五章与第六章。此部分主要介绍了监督学习中常见算法：回归分析，支持向量机，决策树算法与保形预测。

第三章：回归分析。此章节主要介绍单响应变量的线性回归模型、单响应变量的广义线性模型 (GLM) 和多元响应变量协方差广义线性模型 (McGLM)。并给出了基于 Python 与 R 语言代码的相应案例实践。

第四章：支持向量机。本章分别介绍了线性可分 SVM、线性 SVM、非线性 SVM 处理分类问题以及回归问题的相关理论与方法。最后给出了基于 Python 语言与 R 语言的实现代码。

第五章：决策树。介绍的内容主要包括：决策树的基本概念、回归树和分类树的建模过程、防止决策树过拟合的方法及实现决策树相关的 Python 和 R 语言程序。

第六章：保形预测。该章节主要介绍了保形预测的基本思想及保形预测的几类常见算法，并给出了基于 Python 语言和 R 语言的实现代码。

稀疏学习

包括第七章与第八章。此部分主要介绍了稀疏学习中常见算法：模型选择与特征筛选。

第七章：模型选择。主要介绍了两种常用的模型选择的方法：子集选择法和压缩估计法及基于 Python 语言与 R 语言的实现代码。

第八章：特征筛选。此章节主要介绍了针对超高维数据进行特征筛选的几种常用方法，并给出了基于 Python 语言与 R 语言的实现代码。

深度学习

包括第九章与第十章。主要介绍了深度学习中基本概念以及较为常见的模型。

第九章：人工神经网络。此章主要讨论了前馈神经网络的数学模型、正向和反向计算方法以及 RBF 神经网络的数学模型和计算方法，对于其它神经网络形式做了简单介绍。同时也给出了基于 R 语言与 Python 语言的人工神经网络模型实践分析。

第十章：深度学习。此章节针对图像数据，讨论了卷积神经网络、循环神经网络、长短时记忆网络、自编码器等网络的结构和计算过程及其基于 Python 语言与 R 语言的实现代码。

集成学习

包括第十一章，第十二章与第十三章。这一部分主要介绍了集成学习的三种不同方法：随机森林，Boosting 方法与模型平均。

第十一章：随机森林。主要介绍了基本概念，详细介绍了随机森林方法，并给出了基于 Python 语言与 R 语言的实现代码。

第十二章：Boosting 方法。本章主要介绍 Boosting 方法及其代表性算法 AdaBoost，并给出了程序实现。

第十三章：模型平均。本章节主要介绍了频率和贝叶斯模型平均方法及其理论依据，最后也给出了模型平均方法的程序实现。

增量学习

包括第十四章与第十五章。这部分主要介绍了增量学习的两种常见算法：在线学习与并行计算。

第十四章：在线学习。此章节主要介绍均值模型、线性模型、岭回归以及高维模型的在线学习算法和程序算法。

第十五章：并行计算。此章节主要基于 Python 和 R 编程环境，给出了基于 CPU 和基于 GPU 的并行计算的相关概念、原理和计算流程。

第十六章：迁移学习。此章节主要基于 Python 和 R 编程环境，给出了统计特征变换方法及几何特征变换方法等迁移学习常用的算法。

生成学习

包括第十七章。这部分主要介绍了生成式机器学习的常见算法。

第十七章：生成式机器学习。此章节主要深入探讨生成式机器学习的核心概念和方法，以及它在机器学习和数据科学中的广泛应用。

强化学习

包括第十八章与第十九章。这部分主要介绍了强化学习学习的常见算法：动态规划算法，Sarsa 算法与 Q-learning 算法。

第十八章：强化学习概述。此章节主要介绍强化学习的基本概念、基本方法，并介绍了经典的强化学习算法：动态规划算法、蒙特卡罗方法。

第十九章：无模型强化学习。此章节主要介绍强化学习的重要方法及机制：基于时序差分的无模型强化学习算法，基于资格迹的强化学习机制，以及引入近似值函数的强化学习求解方法。

目　录

第一部分

无监督学习

第一章　聚类分析

无监督学习（Unsupervised Learning）是机器学习中的一种重要方法。与监督学习不同，无监督学习的目标是从数据中发现隐藏的关系、结构和规律，而不是根据已知的标签进行分类或预测。通常用于数据探索、降维、聚类、特征学习和生成等任务，还有助于发现数据中的新见解、处理大规模数据集和减少人工干预。

无监督学习在许多领域都有应用，包括数据挖掘、自然语言处理、计算生物学、图像处理和推荐系统等。它是机器学习中的重要分支，有助于揭示数据中的潜在模式，从而提供有关数据的深入理解。

本章及下一章将介绍两种主要的无监督学习方法：聚类分析（Cluster Analysis）和主成分分析（Principal Component Analysis，PCA）。

1.1　简介

当数据集有明确的标签时，可利用标签信息进行监督学习，发现数据内在特性，然而，在一些实际问题中，标签数据有时较难获取或者无法获取，此时，通常使用无监督学习对数据进行贴签，聚类分析是无监督学习的典型代表。

所谓聚类是指根据某种规则，将样本集中具有相似特征的样本进行分组的过程，每个分组组成一个“簇”，不同簇之间互不相交，所有簇的并集构成整个样本集，以簇中心表示簇内所有样本的特性。聚类过程使得同一簇内样本的相似度尽可能高，不同簇间样本的相似度尽可能低，从而达到“物以类聚”的目的。

目前，聚类算法被广泛应用于实际问题中，例如：企业对客户的价值分析，由于数据量较大，通常很难采用人工方式定义某一客户是重点客户或者一般客户，此时可通过聚类算法将客户分为不同簇，每个簇表示一个类别，然后通过领域专家判断每个类别的特性。

本章将介绍几种较为常用的聚类算法，在此之前，首先介绍聚类算法中的重要概念—相似度。

1.2 相似度

相似度在聚类算法中至关重要，是聚类算法的核心问题，具体表现在两个层面：（1）不同的聚类算法其相似度计算方法也不同；（2）同一聚类算法采用不同的相似度计算将得到不同的聚类效果。本节主要介绍数据对象间相似度和簇间相似度。

1.2.1 数据对象间相似度

聚类是对所有数据对象或样本进行一定的处理，达到“物以类聚”的目标。此过程涉及不同样本之间相似度的刻画，选取不同的相似度将产生不同的聚类结果，本节介绍几种样本之间相似度的度量方法。

1、闵可夫斯基距离（Minkowski distance）

每一个样本可以看作是一个向量，所有样本则是向量空间中点的集合，可以采用向量之间的距离度量刻画样本之间的相似度。假设 $\boldsymbol{X}=(X_1,\cdots,X_p)^{\mathrm{T}}$、$\boldsymbol{Y}=(Y_1,\cdots,Y_p)^{\mathrm{T}}$、$\boldsymbol{Z}=(Z_1,\cdots,Z_p)^{\mathrm{T}}$ 表示三个样本，其中 p 表示样本特征的维数，给定函数 $\mathrm{d}(\cdot,\cdot)$，如果其满足下面四个性质，则称其为距离度量：

（1）非负性：$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})\geq 0$；

（2）同一性：$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=0$ 当且仅当 $\boldsymbol{X}$=$\boldsymbol{Y}$；

（3）对称性：$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\mathrm{d}(\boldsymbol{Y},\boldsymbol{X})$；

（4）直递性：$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})\leq \mathrm{d}(\boldsymbol{X},\boldsymbol{Z})+\mathrm{d}(\boldsymbol{Z},\boldsymbol{Y})$。

闵可夫斯基距离是常用的距离度量，其计算公式如下：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\left(\sum_{k=1}^{p}|X_k-Y_k|^t\right)^{\frac{1}{t}} \tag{1.2.1}$$

上式中 $t\geq 1$。下面介绍闵可夫斯基距离的三种特殊形式。

当 $t=1$ 时，称为曼哈顿距离，定义为：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\sum_{k=1}^{p}|X_k-Y_k| \tag{1.2.2}$$

当 $t=2$ 时，称为欧氏距离，定义为：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\left(\sum_{k=1}^{p}|X_k-Y_k|^2\right)^{\frac{1}{2}} \tag{1.2.3}$$

当 $t=\infty$ 时，称为切比雪夫距离，定义为：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\max_{1\leq k\leq p}|X_k-Y_k| \tag{1.2.4}$$

2、马氏距离（Mahalanobis distance）

马氏距离可视为欧式距离的一种泛化形式，一定程度上可以克服欧氏距离中由于不同特征量纲不一致导致的影响距离计算的问题。

若样本 $\boldsymbol{X}=(X_1,\cdots,X_p)^{\mathrm{T}}$、$\boldsymbol{Y}=(Y_1,\cdots,Y_p)^{\mathrm{T}}$ 服从同一类型分布且具有相同的协方差阵，其协方差矩阵为 $\boldsymbol{\Sigma}$，则马氏距离可以表示为：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\sqrt{(\boldsymbol{X}-\boldsymbol{Y})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{Y})} \tag{1.2.5}$$

上式中，当协方差矩阵 $\boldsymbol{\Sigma}$ 是单位矩阵时，即各个维度是独立同分布的，则马氏距离等同于欧氏距离。

3、夹角余弦

两个向量之间的夹角余弦可用于衡量两者之间的相似性，当把样本看作向量时，若两者之间的夹角余弦是 0，即角度为 90 度，那么说明两者之间是独立的；若夹角余弦是 1，说明两者指向相同方向；若夹角余弦是-1，说明两者指向相反方向。夹角余弦越靠近 1，表明样本之间的相似度越高。

因此 $\boldsymbol{X}$ 与 $\boldsymbol{Y}$ 的夹角余弦定义为：

$$\mathrm{d}(\boldsymbol{X},\boldsymbol{Y})=\frac{\sum_{k=1}^{p}X_kY_k}{\left[\left(\sum_{k=1}^{p}X_k^2\right)\left(\sum_{k=1}^{p}Y_k^2\right)\right]^{\frac{1}{2}}} \tag{1.2.6}$$

4、相关系数

相关系数介于 [0,1] 之间，可以衡量变量之间线性相关的程度，在聚类算法中可用于样本间相似度的度量，其绝对值越接近 1，样本之间的相似度越高；越接近 0，样本之间相似度越低。从两类样本集 $\mathbf{X}=(\boldsymbol{X}_1,\cdots,\boldsymbol{X}_n)^{\mathrm{T}}$ 和 $\mathbf{Y}=(\boldsymbol{Y}_1,\cdots,\boldsymbol{Y}_n)^{\mathrm{T}}$ 中任取两个样本 $\boldsymbol{X}_i=(X_{i1},\cdots,X_{ip})^{\mathrm{T}}$、$\boldsymbol{Y}_j=(Y_{j1},\cdots,Y_{jp})^{\mathrm{T}}$，两者之间的相关系数定义为：

$$\mathrm{d}(\boldsymbol{X}_i,\boldsymbol{Y}_j)=\frac{\left(\boldsymbol{X}_i-\bar{\boldsymbol{X}}\right)^{\mathrm{T}}\left(\boldsymbol{Y}_j-\bar{\boldsymbol{Y}}\right)}{\left\|\boldsymbol{X}_i-\bar{\boldsymbol{X}}\right\|_2\left\|\boldsymbol{Y}_j-\bar{\boldsymbol{Y}}\right\|_2} \tag{1.2.7}$$

其中 $\|*\|_2$ 表示 L_2 范数（欧几里得范数），$\bar{\boldsymbol{X}}$ 和 $\bar{\boldsymbol{Y}}$ 分别表示样本集 $\mathbf{X}$ 和 $\mathbf{Y}$ 的样本均值，

$$\bar{\boldsymbol{X}}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{X}_i,\quad \bar{\boldsymbol{Y}}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{Y}_i$$

1.2.2 簇间相似度

聚类过程将样本集划分为互不相交的簇，簇间相似度可通过计算簇间距离得到，簇间距离可以通过样本间的距离定义。假设 G_1、G_2 是两个不同的簇，两者之间的距离可通过以下方式定义。

1、最小距离

最小距离由簇间距离最近的两个样本决定，定义为：

$$D_{12} = \min_{\boldsymbol{X}_i \in G_1, \boldsymbol{X}_j \in G_2} d_{ij} \tag{1.2.8}$$

其中 d_{ij} 表示样本 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 之间的距离。

2、最大距离

最大距离由簇间距离最远的两个样本决定，定义为：

$$D_{12} = \max_{\boldsymbol{X}_i \in G_1, \boldsymbol{X}_j \in G_2} d_{ij} \tag{1.2.9}$$

3、平均距离

平均距离等于两簇之间所有样本间的距离的平均值，定义为：

$$D_{12} = \frac{1}{|G_1|\,|G_2|} \sum_{\boldsymbol{X}_i \in G_1, \boldsymbol{X}_j \in G_2} d_{ij} \tag{1.2.10}$$

其中 $|G_1|$ 表示簇 G_1 中包含的样本个数，$|G_2|$ 表示簇 G_2 中包含的样本个数。

4、中心距离

两个不同簇中心之间距离定义为中心距离，假设 $\bar{\boldsymbol{X}}_1$ 表示簇 G_1 的中心，$\bar{\boldsymbol{X}}_2$ 表示簇 G_2 的中心，则中心距离定义为：

$$D_{12} = d_{\bar{\boldsymbol{X}}_1 \bar{\boldsymbol{X}}_2} \tag{1.2.11}$$

其中，

$$\bar{\boldsymbol{X}}_1 = \frac{1}{|G_1|} \sum_{\boldsymbol{X}_i \in G_1} \boldsymbol{X}_i,$$

$$\bar{\boldsymbol{X}}_2 = \frac{1}{|G_2|} \sum_{\boldsymbol{X}_j \in G_2} \boldsymbol{X}_j$$

1.3 K 均值聚类

1.3.1 原理

K 均值也称为 K-means，是一种原理简单、应用广泛的聚类算法。其目标是将包含 n 个样本的数据集，按照某种规则划分到 K 个互不相交的子集中，称为 K 个簇，相同簇中样本的相似度较高，不同簇之间样本的相似度较低。

假设样本集合为 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，其中 $\boldsymbol{X}_i$ 表示第 i 个样本。令 $\boldsymbol{\mu}_1$，$\boldsymbol{\mu}_2$，$\cdots, \boldsymbol{\mu}_K$ 是 K 个向量，表示 K 个簇中心，其中 $\boldsymbol{\mu}_k$ 代表第 k 个簇的中心。K 均值

聚类采用欧氏距离计算样本与簇中心之间的距离，然后将样本置入距离最近的簇中心所代表的簇中，这样形成 K 个互不相交的集合（簇）$G_1, G_2, \cdots, G_K$，当 $j \neq k$ 时满足 $G_j \cap G_k = \emptyset$，并且 $\cup_{k=1}^{K} G_k = \mathbf{X}$。

基于欧式距离，K 均值聚类算法的目标函数为：

$$L(\Theta) = \sum_{k=1}^{K} \sum_{\boldsymbol{X}_i \in G_k} \|\boldsymbol{X}_i - \boldsymbol{\mu}_k\|^2 \tag{1.3.1}$$

其中，$\boldsymbol{\mu}_k = \dfrac{1}{|G_k|} \sum_{\boldsymbol{X}_i \in G_k} \boldsymbol{X}_i$。

计算上述目标函数的最优解并非易事，K 均值聚类算法采用迭代更新策略得到优化问题的近似解。首先选取初始簇中心 $\boldsymbol{\mu}_k$，$k = 1, 2, \cdots, K$，然后按照欧氏距离最小原则将样本划入不同簇中，最后更新簇中心，以便得到更小的 $L(\Theta)$ 值。将此过程不断迭代下去，直到聚类结果基本保持不变，停止计算。K 均值聚类算法的流程如下：

1、从样本集 $\mathbf{X}$ 中随机选取 K 个样本向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$ 作为初始簇中心。

2、计算每一个样本到 K 个簇中心的距离，并将样本划入到距离最小的簇中心所代表的簇中。

3、重新计算每个簇的中心向量 $\{\boldsymbol{\mu}'_1, \boldsymbol{\mu}'_2, \cdots, \boldsymbol{\mu}'_K\}$。如果 $\boldsymbol{\mu}'_k \neq \boldsymbol{\mu}_k$，则将 $\boldsymbol{\mu}_k$ 更新为 $\boldsymbol{\mu}'_k$。

4、不断迭代第 2 步和第 3 步，直到所有的簇中心不再更新为止。

5、算法结束，输出划分结果。

如图 1.1，展示了 K 均值聚类的整个过程，每张图代表每一次的聚类结果。首先从原始数据集中随机选取三个样本作为初始聚类中心，分别用红色、绿色和黑色的三角形表示，如图 1.1(a) 所示。之后计算每个样本与三个聚类中心的距离，根据距离最近原则将样本划分为颜色各异的三个簇，如图 1.1(b)。随后，重新计算每个簇的样本均值，作为新的三个聚类中心，如图 1.1(c)，此时簇中心已经发生改变，计算样本到新的簇中心之间的距离，得到新的簇划分。不断迭代此过程，直至聚类中心不再变化。

1.3.2 特点

K 均值聚类是机器学习中的常用无监督学习算法，其优点是易于实现、拥有较好的性能，并且相比于其他机器学习算法具有较少的超参数。然而，K 的取值以及初始簇中心的选择对算法有较大的影响。K 的取值关乎聚类输出的效果以及可解释性，应当选取合适的 K 值作为聚类簇数。初始簇中心的选择影响算法的迭代效果，好的初始簇中心能够加快算法的收敛，反之将影响收敛速度。本节将从这两个方面对 K 均值算法进行分析。

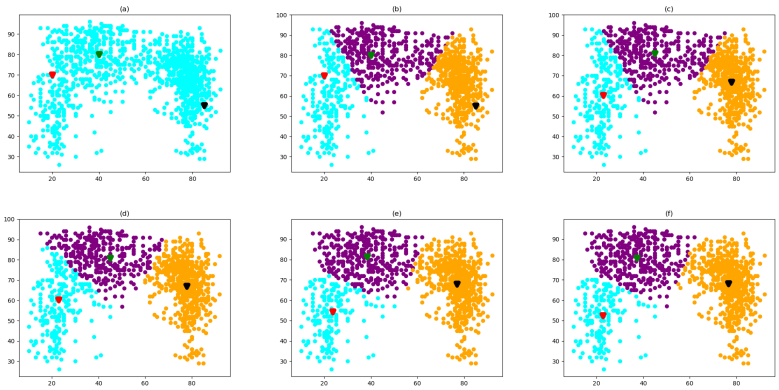


图 1.1 K 均值聚类过程

1、K 值的确定

在 K 均值算法中，簇个数 K 的选择对算法最终的输出结果影响较大，不同的 K 值将产生不同的聚类效果，在实际应用中，K 的最优值通常较难获取，本节介绍三种 K 值确定方法。

（1）根据数据的先验知识确定 K 值，或者可通过简单数据分析得到 K 的可能取值。

（2）指示函数确定法：也可称为肘部确定法。定义一个指示函数，如图 1.2，横轴表示 K 的取值，纵轴表示指示函数的取值。函数值随着 K 值的变化而变化，对于选定的某个临界值，如果 K 大于该值时，函数的变化趋于平缓，反之小于该值时，函数值的变化较为剧烈，则可选择该值作为最优 K 值。其中指示函数可以选择误差平方和、平均直径、平均半径等。

（3）平均轮廓系数法：轮廓系数的计算公式如下：

$$R\left(\boldsymbol{X}_i\right)=\frac{B\left(\boldsymbol{X}_i\right)-I(\boldsymbol{X}_i)}{\max[B\left(\boldsymbol{X}_i\right),I(\boldsymbol{X}_i)]} \tag{1.3.2}$$

上式中，$\boldsymbol{X}_i$ 表示第 i 个样本向量，$I(\boldsymbol{X}_i)$ 表示样本 $\boldsymbol{X}_i$ 与其所在簇中其他样本之间的

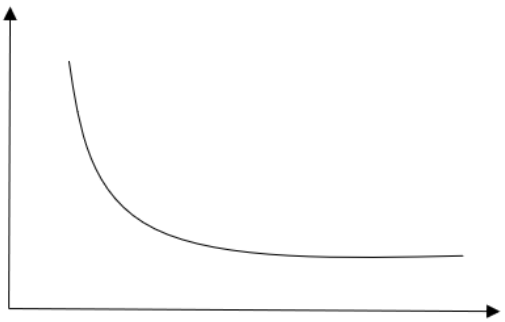


图 1.2 指示函数与 K 值关系图

平均距离，$B(\boldsymbol{X}_i)$ 表示样本 $\boldsymbol{X}_i$ 与其他簇中所有样本之间的平均距离。由式 (1.3.2) 可计算出每个样本的轮廓系数，然后再求平均值，得到平均轮廓系数。一般情形下，平均轮廓系数越接近 1 聚类效果越好。

2、初始簇中心的选择

实际问题中，合适的初始簇中心对算法至关重要。本部分介绍四种选择初始簇中心的方法。

（1）根据对数据集的先验知识，由领域专家设定初始簇中心。

（2）采用不同的簇中心，多次运行算法，选取最优的初始簇中心。

（3）选择尽可能远离的 K 个点：首先从样本集合中选择一个样本，然后选择距离此样本最远的样本作为第二个簇中心，依次进行下去，直到选出 K 个初始簇中心。

（4）采用层次聚类等算法进行初始聚类，利用这些类的中心作为 K 均值算法的初始簇中心。

1.4 模糊 C 均值聚类

K 均值聚类算法将样本划分到不同簇，其结果具有非黑即白的性质，即非 0 即 1，所有样本对象均被硬性划分到某一簇类别，在很多实际问题中可以达到较好效果，如是否下雨、是否旅行等；然而，这种硬聚类的方式不适用于类别界限不明确的问题，如健康程度、冷热程度等。模糊 C 均值聚类(Fuzzy C-means，FCM) 采用软聚类思想，用隶属度描述样本与簇之间的关系，隶属度表示样本属于某一簇的确定程度。

与 K 均值聚类相似，FCM 算法也是通过最小化目标函数，不断迭代更新聚类簇中心，当目标函数收敛时，算法停止迭代；不同的是，FCM 算法在 K 均值聚类算法目标函数的基础上，增加了隶属度度量，其表达式为：

$$L(\Theta)=\sum_{i=1}^{n}\sum_{j=1}^{C}\omega_{i,j}^{m}\left\|\boldsymbol{X}_i-\boldsymbol{\mu}_j\right\|^2,\quad 1\leq m<\infty \tag{1.4.1}$$

其中，n 为样本总个数，C 为聚类中心数，m 为隶属度参数，$\boldsymbol{X}_i$ 表示第 i 个样本，$\boldsymbol{\mu}_j$ 表示第 j 个聚类中心，$\omega_{i,j}^{m}$ 用于刻画隶属度并且满足，

$$\sum_{j=1}^{C}\omega_{i,j}=1$$

通过拉格朗日乘子法可以得到：

$$\boldsymbol{\mu}_j=\frac{\sum_{i=1}^{n}\omega_{i,j}^{m}\boldsymbol{X}_i}{\sum_{i=1}^{n}\omega_{i,j}^{m}}$$

$$\omega_{i,j}=\frac{1}{\sum_{k=1}^{C}\frac{\left\|\boldsymbol{X}_i-\boldsymbol{\mu}_j\right\|^{\frac{2}{m-1}}}{\left\|\boldsymbol{X}_i-\boldsymbol{\mu}_k\right\|^{\frac{2}{m-1}}}}$$

FCM 算法通过不断迭代更新 $\omega_{i,j}^m$ 与 $\boldsymbol{\mu}_j$ 的取值，最终得到聚类结果，迭代终止条件可设置为：

$$\max_{i,j}\{|\omega_{i,j}^{m,s+1} - \omega_{i,j}^{m,s}|\} < \epsilon$$

其中，s 表示迭代次数，ϵ 为给定的误差阈值。其含义为，当两次迭代之间的最大隶属度变化量小于给定阈值时，说明继续迭代隶属度的变化量也会非常小，此时停止迭代，得到最终的聚类结果。

FCM 算法的算法流程为：

1、超参数确定：选择合适的聚类簇数 C、隶属度参数 m 以及误差阈值 ϵ；

2、初始化隶属度矩阵 $\boldsymbol{\omega}^{m,0}$；

3、计算并更新聚类中心 $\boldsymbol{\mu}_j$；

4、计算并更新隶属度矩阵 $\boldsymbol{\omega}^{m,s}$；

5、比较更新前后隶属度矩阵的最大变化量，如果小于 ϵ，算法停止，否则返回第 3 步。

1.5 高斯混合聚类

高斯混合聚类是在概率框架下实施聚类过程，采用软聚类的思想计算样本被划分到不同簇的概率，选取概率最大的簇作为最终的划分结果。

在阐述高斯混合聚类算法之前，首先给出多变量高斯分布概率密度函数的定义：如果样本空间中的 p 维随机向量 $\boldsymbol{X} = (X_1, \cdots, X_p)^{\mathrm{T}}$ 服从高斯分布，则其概率密度函数为

$$\Pr(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{p}{2}}\,|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})}. \tag{1.5.1}$$

其中，$\boldsymbol{\mu}$ 表示均值向量，$\boldsymbol{\Sigma}$ 表示协方差矩阵，显然其概率密度函数由 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$ 唯一确定。下面以二维高斯分布为例，给出其概率密度函数图像（如图 1.3）。

高斯混合聚类模型的算法可分为三个环节：

（1）假设样本集是由多个高斯分布生成，每个分布的概率密度函数为式 (1.5.1)，不同高斯分布的 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$ 值也不同；

（2）采用 EM 算法不断迭代更新 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$ 的值拟合样本数据，直到算法收敛或者达到最大迭代次数为止；

（3）利用所计算出的概率密度函数对样本集进行聚类。其中第（2）步是整个算法的核心，下面将详细推导参数的更新策略。

其中，EM 算法（Expectation-Maximization）是一种用于估计含有隐变量（latent variables）的概率模型参数的迭代优化算法，广泛应用于 GMM 的参数估计。

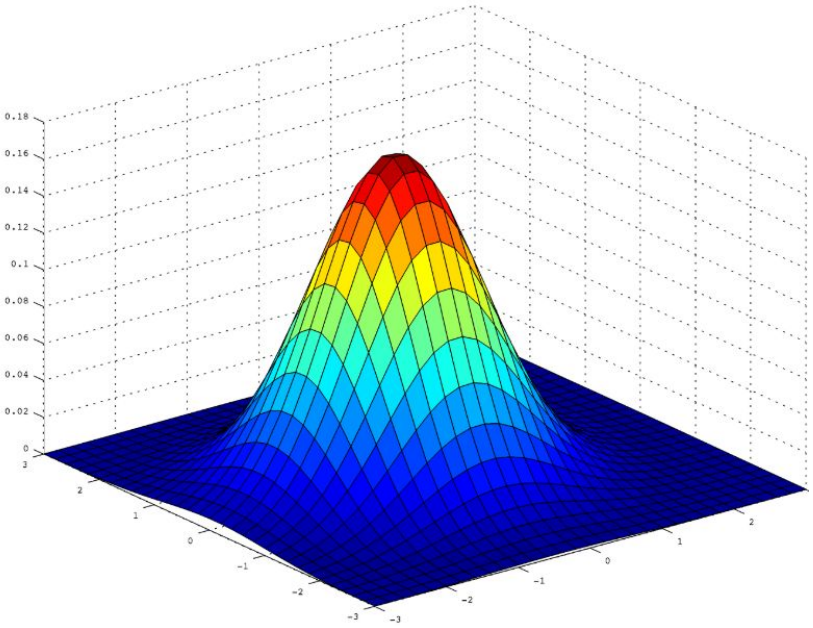


图 1.3 二维高斯分布概率密度函数

EM 算法是一种迭代优化算法，用于估计概率模型参数，特别适用于包含隐变量的模型。其中，E 步骤（Expectation）：使用当前参数估计计算隐变量的后验概率（在给定观测数据的条件下隐变量的概率分布）；M 步骤（Maximization）：最大化完全数据（包括观测数据和隐变量）的对数似然函数，更新模型参数。

GMM 和 EM 算法的结合允许数据以不同的概率分布组合来表示，更灵活地适应各种数据分布。总体来说，EM 算法在高斯混合模型中的应用允许通过迭代优化来估计模型参数，从而实现对复杂数据分布的建模。EM 算法的有效性在于其对包含隐变量的概率模型参数估计的鲁棒性和收敛性。

假设高斯混合聚类模型由 K 个高斯分布组成，每个高斯分布称为一个成分，每个成分对应一个簇划分，对高斯分布进行线性组合得到高斯混合聚类的概率密度函数：

$$\Pr(\boldsymbol{X}) = \sum_{k=1}^{K} \lambda_k \Pr(\boldsymbol{X}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.5.2}$$

其中，$\boldsymbol{\mu_k}$ 和 $\boldsymbol{\Sigma_k}$ 分别表示第 k 个混合成分的均值向量与协方差矩阵。λ_k 是第 k 个混合成分系数（权重），并且满足

$$\sum_{k=1}^{K} \lambda_k = 1$$

可以采用极大似然估计求解模型参数 λ_k、$\boldsymbol{\mu_k}$ 以及 $\boldsymbol{\Sigma_k}$。假设样本集 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，对数似然函数可以定义为：

$$
\begin{aligned}
L(\Theta) &= \ln\left(\prod_{i=1}^{n}\Pr(\boldsymbol{X}_i)\right) \\
&= \sum_{i=1}^{n}\ln\left(\sum_{k=1}^{K}\lambda_k\Pr(\boldsymbol{X}_i|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right)
\end{aligned}
\tag{1.5.3}
$$

此时的目标转化为：求使得式 (1.5.3) 最大化的 λ_k、$\boldsymbol{\mu}_k$ 和 $\boldsymbol{\Sigma}_k$，其中 $1 \leq k \leq K$。对于 $\boldsymbol{\mu}_k$ 和 $\boldsymbol{\Sigma}_k$ 的求解，可分别对其求导，并令导数等于 0，即

$$
\frac{\partial L(\Theta)}{\partial \boldsymbol{\mu}_k} = 0 \tag{1.5.4}
$$

$$
\frac{\partial L(\Theta)}{\partial \boldsymbol{\Sigma}_k} = 0 \tag{1.5.5}
$$

求解式 (1.5.4)，我们可以得到，

$$
\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{n}\theta_{ik}\boldsymbol{X}_i}{\sum_{i=1}^{n}\theta_{ik}} \tag{1.5.6}
$$

其中 θ_{ik} 定义为，

$$
\theta_{ik} = \frac{\lambda_k\Pr(\boldsymbol{X}_i|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{k=1}^{K}\lambda_k\Pr(\boldsymbol{X}_i|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}
$$

θ_{ik} 表示第 i 个样本 $\boldsymbol{X}_i$ 被划入第 k 个混合成分的概率，高斯混合聚类选取 $\max(\theta_{ik}), k = 1, 2, \cdots, K$ 作为 $\boldsymbol{X}_i$ 的簇划分。同理求解式 (1.5.5)，我们可以得到

$$
\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{n}\theta_{ik}(\boldsymbol{X}_i - \boldsymbol{\mu}_k)(\boldsymbol{X}_i - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_{i=1}^{n}\theta_{ik}} \tag{1.5.7}
$$

下面我们考虑 λ_k 的求解方法。由于 λ_k 满足条件

$$
\sum_{k=1}^{K}\lambda_k = 1, \lambda_k \geq 0
$$

因此需求解带约束的极值优化问题，采用拉格朗日乘子法容易得到:

$$
\lambda_k = \frac{1}{n}\sum_{i=1}^{n}\theta_{ik} \tag{1.5.8}
$$

通过上述推导过程可以看出，高斯混合聚类可以采用 EM 算法对参数进行更新，具体可分为两个步骤：（1）根据前一个迭代步（或初始值）的参数值计算 θ_{ik}；（2）利用第（1）步求得的 θ_{ik} 更新 λ_k、$\boldsymbol{\mu}_k$ 和 $\boldsymbol{\Sigma}_k$。

高斯混合聚类算法的流程可以概括为：

1、参数初始化，包括 λ_k、$\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$ 以及混合成分个数 K。

2、根据前一个迭代步（或初始值）的参数值计算 θ_{ik}。

3、利用式 (1.5.6)、(1.5.7)、(1.5.8) 更新模型参数 $\boldsymbol{\mu}_k$、$\boldsymbol{\Sigma}_k$、λ_k。如果满足停止条件，则停止迭代；否则返回第 2 步，继续迭代计算。

4、根据所求得到 θ_{ik}，将 $\boldsymbol{X}_i$ 划入到相应簇划分 C_k，$C_k = C_k \cup \boldsymbol{X}_i$。

5、得到最终的簇划分 $C = \{C_1, C_2, \cdots, C_K\}$，算法结束。

1.6 层次聚类

层次聚类顾名思义是一个逐层进行聚类的过程，通过计算相似度形成一种树形结构。按照形成树形结构方式的不同可以分为聚合聚类和分裂聚类。

聚合聚类是自下而上进行聚类，初始状态将每个样本当作一个簇，然后按照簇间距离最近原则合并两个簇，组成一个新簇，不断迭代此过程，直到满足条件为止；分裂聚类是自上而下进行聚类，初始状态是将样本集看作一个簇，然后按照簇间距离最远的原则将样本划分到两个新的簇，不断迭代此过程，直到满足条件为止。

本节以聚合聚类为例，介绍层次聚类的算法思想。假设样本集为 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，样本个数为 n，特征维数为 p，将样本集划分为 K 个簇，则聚合聚类的算法流程为：

1、将 n 个样本分为 n 个簇，即每个簇中只包含一个样本。

2、对于给定数据集，通过计算各簇之间的距离得到距离矩阵 $D = (d_{ij})_{n\times n}$。其中 d_{ij} 表示第 i 个簇与第 j 个簇之间的距离。

3、将距离最近的两个簇合并为一个新簇。

4、计算新簇与其他簇之间的距离，并更新距离矩阵。

5、不断迭代 3、4 两个过程，直到满足条件停止。

在聚合聚类算法中，样本之间的距离（相似度）计算可以采用本章1.2.1节中的方法，簇间距离的计算可采用1.2.2节中的方法。通常根据距离最小的原则将相似簇进行合并，算法的停止条件可以是达到设定的簇的个数 K 或者达到某个指示函数的阈值。

下面通过例题1.1阐述聚合聚类的算法流程。

例 1.1 假设某数据集中有 4 个样本，通过计算样本之间的欧氏距离得到如下距离矩阵 D，请利用聚合聚类算法将 4 个样本进行聚类。

$$D = \begin{pmatrix} 0 & 2 & 5 & 4 \\ 2 & 0 & 3 & 5 \\ 5 & 3 & 0 & 6 \\ 4 & 5 & 6 & 0 \end{pmatrix}$$

解：（1）将 4 个样本分为 4 个簇 G_1, G_2, G_3, G_4，每个簇中只有一个样本，则簇间距离矩阵为 D。

（2）根据 D，挑选出距离最近的两个簇 G_1, G_2，将这两个簇合并为一个新簇 $G_5 = \{G_1, G_2\}$，此时已划分为 3 个簇 G_3, G_4, G_5。

（3）分别计算新簇 G_5 与 G_3、G_4 之间的最短距离，得到最短距离大小是 $D_{35} = 3$，$D_{45} = 4$。

（4）可以看出，距离最近的 2 个簇是 G_3, G_5，合并这两个簇形成一个新簇 $G_6 = \{G_3, G_5\} = \{G_1, G_2, G_3\}$。此时已划分为 2 个簇 G_4, G_6。

（5）将 G_4, G_6 合并为新簇 $G_7 = \{G_4, G_6\} = \{G_1, G_2, G_3, G_4\}$，算法结束。聚类过程可以表示为图 1.4。

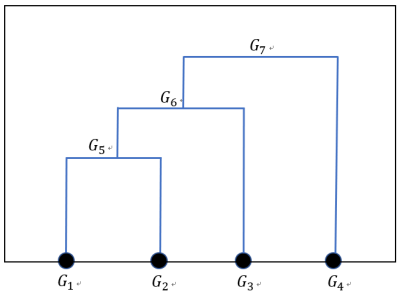


图 1.4 层次聚类过程图

层次聚类可以达到“分层”的效果，通过选取不同的聚类簇数，得到不用层次的聚类结果。然而算法实现过程的复杂度较高，计算量通常高于 K 均值算法。

1.7 DBSCAN 聚类

通常情形下，K 均值聚类一般适用于样本集具有凸形簇状结构（形似“椭球”的簇结构），对非凸样本集效果不够理想。本节介绍一种既可用于凸样本集，也可用于非凸样本集的DBSCAN（Density-Based Spatial Clustering of Applications with Noise）方法。

DBSCAN 是一种基于密度的聚类方法，算法假设样本的类别划分可由样本之间分布的紧密程度决定，紧密程度较高的样本被划分到相同簇，反之，被划分到不同簇，最终，通过样本之间的紧密程度得到聚类结果。由此可见，紧密程度的刻画是 DBSCAN 算法的核心。

DBSCAN 算法采用邻域描述样本集分布的紧密程度，邻域是由 (ϵ, M) 表示，其含义是某一样本 ϵ 邻域内的样本个数不超过 M。下面给出与描述样本集 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$ 紧密程度相关的一些概念：

ϵ-邻域: 对于 $\mathbf{X}$ 中的任一样本 $\boldsymbol{X}_j$，其 ϵ 邻域 $N_\epsilon(\boldsymbol{X}_j)$ 表示与该样本距离不大于 ϵ 的所有样本组成的集合，即 $N_\epsilon(\boldsymbol{X}_j) = \{\boldsymbol{X}_i \in \mathbf{X} | \mathrm{d}(\boldsymbol{X}_i, \boldsymbol{X}_j) \leq \epsilon\}$，其中 $\mathrm{d}(\boldsymbol{X}_i, \boldsymbol{X}_j)$ 表示 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 之间的距离，例如闵可夫斯基距离。

核心对象: 对于 $\mathbf{X}$ 中的任一样本 $\boldsymbol{X}_j$，若其所对应的 $N_\epsilon(\boldsymbol{X}_j)$ 中样本个数大于等于 M，则称 $\boldsymbol{X}_j$ 是核心对象。

密度直达: 若 $\boldsymbol{X}_i \in N_\epsilon(\boldsymbol{X}_j)$，并且 $\boldsymbol{X}_j$ 是一个核心对象，则称样本 $\boldsymbol{X}_i$ 由样本 $\boldsymbol{X}_j$ 密度直达。

密度可达: 对于 $\mathbf{X}$ 中的任意两个样本 $\boldsymbol{X}_i$ 和 $\boldsymbol{X}_j$，如果存在样本序列 $q_1, q_2, \cdots, q_t$ 满足 $q_1 = \boldsymbol{X}_j$, $q_t = \boldsymbol{X}_i$，并且 q_{l+1} 由 q_l 密度直达, 其中 $1 \leq l \leq t-1$，则称样本 $\boldsymbol{X}_i$ 由样本 $\boldsymbol{X}_j$ 密度可达。

密度相连: 对于 $\mathbf{X}$ 中的任意两个样本 $\boldsymbol{X}_i$ 和 $\boldsymbol{X}_j$，如果存在一个样本 $\boldsymbol{X}_l$ 使得 $\boldsymbol{X}_i$ 和 $\boldsymbol{X}_j$ 均可由 $\boldsymbol{X}_l$ 密度可达，则称 $\boldsymbol{X}_i$ 和 $\boldsymbol{X}_j$ 密度相连。

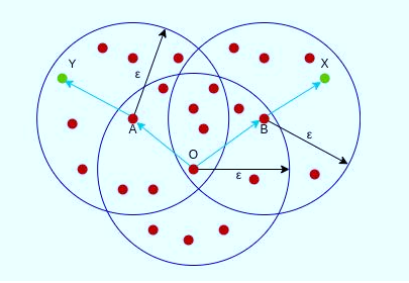


图 1.5 基本概念示意图

上图1.5描述了上述基本概念，如图所示，样本 A、B 和 O 均是核心对象，Y 由 A 密度直达，A 和 B 由 O 密度直达，X 由 B 密度直达；X 和 Y 均由 O 密度可达；X 与 Y 密度相连。

基于上述基本概念，可以给出 DBSCAN 算法中簇的定义: **由密度可达关系导出的最大密度相连的样本集合，即为聚类结果的一个簇**。

DBSCAN 算法的目标是从样本集中找出所有满足簇定义的簇划分，由此可见 DBSCAN 不需要预先给定簇划分的个数。在寻找簇划分时，DBSCAN 从样本集中选择一个没有被划分的核心对象作为 “种子”，寻找由该核心对象密度可达的所有样本构成一个样本子集，即为一个簇划分；然后选择另外一个没有被划分的核心对象作为” 种子”，依次进行下去，直到所有核心对象均被划分为止。

给定样本集 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$, DBSCAN 算法的流程可以描述为:

1、初始化 ϵ、M、核心对象集 $\Omega = \varnothing$、聚类簇数 k=0 以及未被访问的样本集 $\Phi = \mathbf{X}$;

2、遍历样本集中的所有样本，找出核心对象，将其添加到 Ω 中，此时 Ω 存储了所有的核心对象;

3、若 $\Omega = \varnothing$，则算法结束，否则进行第 4 步;

4、记录当前未被访问的样本集合: $\Phi^* = \Phi$，在 Ω 中随机挑选一个核心对象 o，初始化队列 $\Omega^* = <o>$，更新未被访问的样本集 $\Phi = \Phi - \{o\}$；

5、在队列 Ω^* 中随机取出一个核心对象 o，此时 $\Omega^* = \Omega^* - <o>$，并找出其 ϵ-邻域 $N_\epsilon(o)$，令 $\Delta = N_\epsilon(o) \cap \Phi$，更新 $\Phi = \Phi - \Delta$，更新 $\Omega^* = \Omega^* \cup (\Delta \cap \Omega)$；

6、如果 $\Omega^* = \varnothing$，进行第 7 步，否则返回第 5 步；

7、令 $k = k + 1$，更新当前簇 $C_k = \Phi^* - \Phi$，更新 $\Omega = \Omega - C_k$，返回第 3 步；

8、输出最终的簇划分 $C = \{C_1, C_2, \cdots, C_k\}$。

注: 若队列 Ω^* 中的元素在此前循环中被访问过，那么 Ω^* 中将不再包含此元素。

DBSCAN 算法从样本集紧密程度的角度进行簇划分，与 K 均值、高斯混合聚类等算法相比，DBSCAN 算法不需要提前确定聚类簇数，并且可应用于非凸数据集的聚类，在得到聚类结果的同时还可以识别出异常点。然而，当样本集规模较大时，算法收敛时间较长；并且算法调优涉及 ϵ 和 M 两个参数，不同参数组合对聚类的结果影响较大，调参工作量较为复杂。

1.8 其他类型聚类方法

聚类方法多种多样，针对不同的应用场景可能会产生比传统聚类方法更好的算法，因此在使用聚类方法解决实际问题时，应当根据问题本身的特点选择合适的聚类方法。本节将介绍其他几种常用的聚类方法。

1.8.1 混合型数据聚类方法

在实际应用中，数据集通常是由数值型和分类型特征组成，尤其是当数据集是由多源数据集成而得时，如医疗数据（包括病人个人信息：姓名、学历、职业、收入情况等；医疗监测数据：血常规、血脂等）、客户数据（包括客户的年龄、性别、种族、消费记录、消费频次、消费金额等）等。针对这种混合型数据进行聚类分析时需进行特殊处理，本节介绍一种常用的混合型数据聚类方法：K-prototypes。

K-prototypes 是基于 K 均值方法的一种可以处理混合数据的聚类方法。与 K 均值聚类方法相比，主要存在两点不同：(1) 簇中心的选取；(2) 距离的计算。下面针对这两点进行详细阐述。

在 K 均值聚类中，每个簇选取簇内样本的均值作为当前簇中心，这种计算方式仅适用于特征都是数值型数据，当某些特征是分类型时将无法计算。因此 K-prototypes 方法在计算簇中心时将所有特征分为两部分：数值型、分类型。对于数值型特征依然计算均值，对于分类型特征选择众数，然后将两部分合并起来组成当前簇的中心。令样本集为 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，不失一般性，假设任意一个样本 $\boldsymbol{X}_i =$

$(X_{i1},\cdots,X_{ip})^{\mathrm{T}}$ 包含 p 个特征，其中 q 个数值型特征、p-q 个分类型特征。令 $\{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\cdots,\boldsymbol{\mu}_K\}$ 表示 K 个簇中心。

K 均值常采用欧氏距离计算样本与簇中心之间的距离，在 K-prototypes 中数值型特征依然采用欧氏距离，分类型特征采用汉明距离，然后得到样本点与簇中心的距离。K-prototypes 的算法流程可以描述为：

1、初始化聚类簇数 K 以及簇中心 $\{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\cdots,\boldsymbol{\mu}_K\}$；

2、计算数据集中样本点 $\boldsymbol{X}_i$ 到 K 个簇中心的距离，将所有样本划入距离最近的簇中；

3、重新计算每个簇的中心；

4、重复第 2 步、第 3 步，直到满足停止条件（簇中心变化很小或者达到迭代的最大次数）；

注：对于两个数字来说，汉明距离就是转成二进制后，对应的位置值不相同的个数。例如，假设有两个十进制数 a =93 和 b=73，如果将这两个数用二进制表示的话，有 a =1011101、b=1001001，可以看出，二者的从右往左数的第 3 位、第 5 位不同（从 1 开始数），因此，a 和 b 的汉明距离是 2。

1.8.2 双向聚类方法

传统的聚类方法可大致分为两种模式：一种是对样本进行聚类（如根据销售数据对客户类型聚类分析）；另一种是对特征或者变量进行聚类（如分析影响青少年成长的因素，分为积极因素、消极因素等）。这两种模式均是对单一方向（样本或者特征）进行聚类，通常被称为单向聚类。

尽管单向聚类可以解决多数实际问题，仍然存在一些特殊场景并不适合采用单向聚类进行分析，例如利用患某种疾病的病人基因数据对病人进行划分，通常只有少量基因对该病产生影响，并且不同的基因组合对疾病的影响也不尽相同，由于需要同时考虑样本和特征之间的关系，捕捉局部性质，单向聚类在利用所有基因进行聚类分析时效果不佳，因此利用双向聚类方法解决此类问题得到较多的研究。近年来双向聚类方法被广泛应用于基因表达分析、文本数据挖掘等领域，下面介绍一种双向聚类方法——稀疏双向 K 均值聚类。

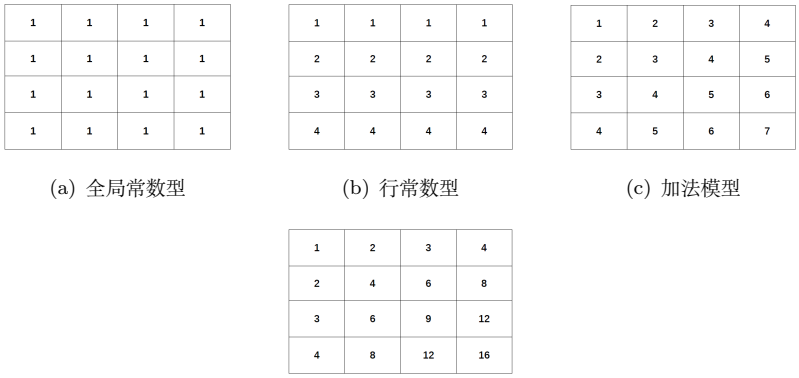
稀疏双向聚类的目标是从原始数据集中根据矩阵元素的相似性提取具有特定结构的子矩阵，下面介绍几种常见的类型：

全局常数型：子矩阵中的所有元素是同一常数，即对于子矩阵 $B=(b_{tr})$，其中 $1\le t\le T, 1\le r\le R$, 满足 $B_{tr}=c$, 其中 c 是常数，如图1.6(a)。实际场景中由于噪声的存在，通常要求子矩阵在阈值 ϵ 范围内满足元素为常数。

行 (列) 常数型：子矩阵的每一行 (或列) 的元素为常数。以行常数为例，对于子矩阵 $B=(b_{tr})$，其中 $1\le t\le T, 1\le r\le R$, 满足 $b_{t1}=b_{t2}=\cdots=b_{tR}=c_t$, 其中 c_t

是常数，如图1.6(b)。

线性关系型：子矩阵的每行 (列) 的元素是其他任意一行 (列) 的线性组合。以行满足线性关系为例，对于子矩阵 $B=(b_{tr})$，其中 $1\leq t\leq T, 1\leq r\leq R$, 满足 $B_{t_1}=c_{t_2}\times B_{t_2}+\theta_{t_2}$ ，其中 c_{t_2},θ_{t_2} 是常数，B_{t_1} 表示矩阵第 t_1 行。列满足线性关系同理可得。当 $c_{t_2}=1$ 时，被称为加法模型，如图1.6(c)；当 $\theta_{t_2}=0$ 时，被称为乘法模型，如图1.6(d)。



(a) 全局常数型 (b) 行常数型 (c) 加法模型

(d) 乘法模型

图 1.6 双向聚类结构类型.

稀疏双向 K 均值聚类是将 K 均值聚类拓展到稀疏双向聚类中的一种算法，求解过程与 K 均值算法类似。现给定原始数据集矩阵为 $\mathbf{X}=(\boldsymbol{X}_1,\cdots,\boldsymbol{X}_n)^{\mathrm{T}}$，其中 n 表示样本数，p 表示特征数且 $\boldsymbol{X}_i=(X_{i1},\cdots,X_{ip})^{\mathrm{T}}$。假设 n 个样本可以划分为 K_1 个类别，记为 $G_1,G_2,\cdots,G_{K_1}$，p 个特征可以划分为 K_2 个类别，记为 $G_1^*,G_2^*,\cdots,G_{K_2}^*$。并且假设样本 $\boldsymbol{X}_i$ 中的元素 X_{ij} 的期望 $\mathrm{E}(X_{ij})=\mu_{ls}$，其中 μ_{ls} 表示 X_{ij} 属于第 l 个样本类别、第 s 个特征类别所对应子矩阵中的元素均值。双向聚类的目标函数为最小化下式：

$$\min\sum_{l=1}^{K_1}\sum_{s=1}^{K_2}\sum_{i\in G_l}\sum_{j\in G_s}(X_{ij}-\mu_{ls})^2 \tag{1.8.1}$$

由式 (1.8.1) 可以看出，当 $K_1=1$ 时，转化为关于特征的 K 均值聚类，当 $K_2=1$ 时，转化为关于样本的 K 均值聚类。

为了增强聚类结果的可解释性，并且减少类别方差，可对式 (1.8.1) 中的 μ_{ls} 施加 Lasso 惩罚，此时目标函数可变为：

$$\min\left\{\frac{1}{2}\sum_{l=1}^{K_1}\sum_{s=1}^{K_2}\sum_{i\in G_l}\sum_{j\in G_s}(X_{ij}-\mu_{ls})^2+\lambda\sum_{l=1}^{K_1}\sum_{s=1}^{K_2}|\mu_{ls}|\right\}. \tag{1.8.2}$$

其中 λ 是非负参数。

显然稀疏双向 K 均值聚类是 K 均值聚类的一种泛化形式，在利用稀疏双向 K 均值聚类算法时，通常先将原始数据集矩阵 $\mathbf{X}$ 进行中心化处理，然后再进入算法流程。算法参数的求解过程与 K 均值算法相似，可以利用迭代求解的思想不断更新参数的取值，直到参数在某个阈值 ϵ 范围内不再变化或者达到最大迭代次数时停止计算。

1.9 聚类实践

1.9.1 R 语言实践

K 均值聚类算法

R 语言中有 K 均值聚类算法库 kmeans，本小节将基于该库函数，选取莺尾花数据集中的 4 个属性为原始数据，阐述 K 均值聚类算法的 R 语言实现过程。

莺尾花数据集共包含 5 个属性，最后一个属性为类别，本实验不予采用，仅使用前 4 个属性进行 K 均值聚类分析，属性含义分别为萼片长度（Sepal.Length）、萼片宽度（Sepal.Width）、花瓣长度（Petal.Length）、花瓣宽度（Petal.Width）。实验代码如下：

```
df<-iris[,1:4]
#调用kmeans函数，centers参数为簇数量；nstart参数表示随机起始分区数
KMmodel<-kmeans(df,centers=3,nstart=8)
KMmodel$centers#输出簇中心
KMmodel$cluster#输出每个样本所属的簇划分
```

模糊 C 聚类算法

本小节将基于 R 语言 e1071 包中的 cmeans 函数实现 FCM 算法。本实验将随机生成 50 个均值为 0、标准差为 0.2 以及 50 个均值为 1、标准差为 0.3 的数据，构成包含 50 个样本的二维数据集合作为样本集。采用 cmeans 方法对该样本集进行聚类分析，代码如下：

```
set.seed(5)
cdata<-rbind(matrix(rnorm(50,mean=0,sd=0.2),ncol=2),
matrix(rnorm(50,mean=1,sd=0.3),ncol=2))#生成样本集
library(e1071)
#iter.max参数为最大迭代次数；method参数默认为cmeans，表示模糊C均值聚类
CMmodel<-cmeans(cdata,centers=2,iter.max=30,verbose=TRUE,method="cmeans")
CMmodel$centers#输出聚类中心
CMmodel$size#每个簇中数据点的个数
```

高斯混合聚类算法

R 语言 mclust 包中的 Mclust 函数可以实现高斯混合聚类算法，本节将随机生成样本集，调用 Mclust 函数阐述高斯混合聚类算法的实现过程。

使用上一部分生成的样本集，并且调用 summary 函数、plot 函数等输出聚类结果，效果图如图1.7。

```
library(mclust)
Gmodel<-Mclust(cdata)
summary(Gmodel)
plot(Gmodel,what="classification")
```

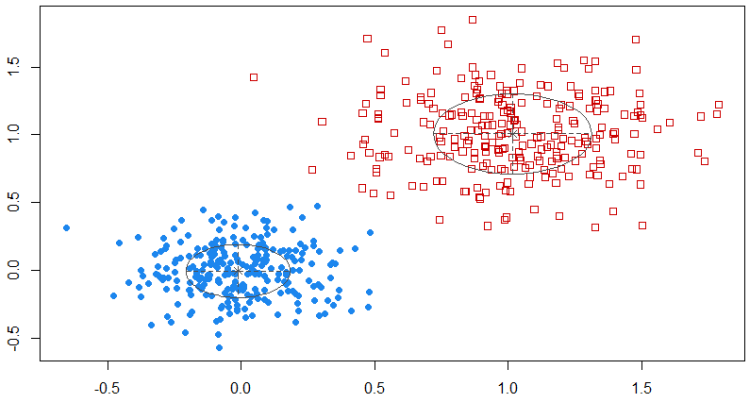


图 1.7 聚类效果图

层次聚类算法

R 语言中 hclust 函数可以实现层次聚类算法，本小节将随机生成样本集，调用 hclust 函数阐述层次聚类算法的实现过程。

```
set.seed(5)
Hdata<-rbind(matrix(rnorm(10,mean=0,sd=0.2),ncol=2),matrix(rnorm(10,mean=1,sd=0.3),
ncol=2),matrix(rnorm(10,mean=2,sd=0.3),ncol=2))#生成样本集
#调用dist函数计算样本之间的欧氏距离（或者其他形式的距离），得到距离矩阵
HDdist<-dist(Hdata,method="euclidean")
Hmodel=hclust(HDdist,method="single")#调用hclust函数采用最短距离法进行聚类分析
plot(Hmodel)#调用plot函数画出层次聚类效果图
rect.hclust(Hmodel,k=3)#调用rect.hclust函数用矩形框出
```

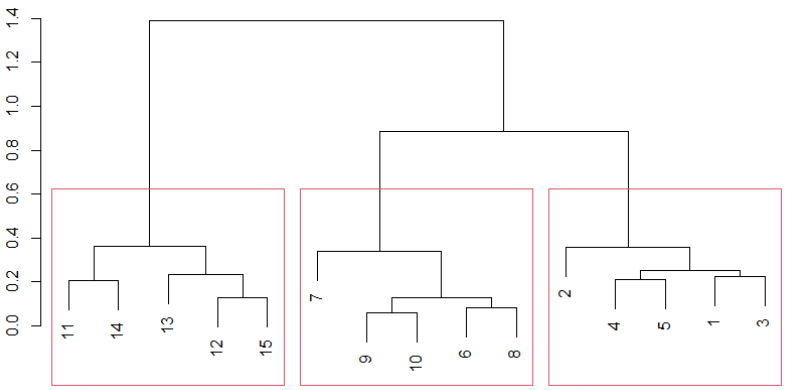


图 1.8 层次聚类效果图

DBSCAN 聚类算法

R 语言中的 fpc 库和 dbscan 库均可实现 DBSCAN 算法，由于 dbscan 库中的 dbscan 函数具有更快的计算效率并且可以处理更大规模的数据集，因此本节将基于 dbscan 函数实现。此次实验同样选取莺尾花数据集的前 4 个属性作为实验数据。

```
DBdata<-iris[,1:4]
library(dbscan)
DBmodel<-dbscan(DBdata,eps=0.7,MinPts=5)#调用dbscan函数：选取eps为0.7，MinPts为5
print(DBmodel$cluster)#结果输出
plot(DBdata[,1:2],col=DBmodel$cluster)
```

双向聚类算法

本小节以 Bimax 算法为例，利用 biclust 库中的函数实现算法整体流程，实现代码如下：

```
library(biclust)
#生成一个由 5000 个满足正态分布的随机数构成的 100×50 的矩阵
test <- matrix(rnorm(5000), 100, 50)
#生成由均值是 3、标准差是 0.1 的 100个随机数构成的 10×10 矩阵
test[11:20,11:20] <- rnorm(100, 3, 0.1)
loma <- binarize(test,2)#进行预处理
#使用bcbimax 函数进行双聚类并将结果保存在 res 中
res <- biclust(x=loma, method=BCBimax(), minr=4, minc=4, number=10)
```

输出结果如下：

```
An object of class Biclust
call:
  biclust(x = loma, method = BCBimax(), minr = 4, minc = 4, number = 10)
There was one cluster found with
  10 Rows and  10 columns#输出结果显示有一个 10×10 的簇
```

1.9.2 Python 语言实践

K 均值聚类算法

K 均值聚类算法原理简单、容易实现，并且有较多资料库可以直接调用。本小节将基于 sklearn 库实现 K 均值算法。代码实现如下：

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
#调用sklearn库中的make_blobs函数建立数据集，特征维数设置为2，样本数设置为200
X, y = make_blobs(n_features=2, n_samples=200, centers=4,
cluster_std=0.80, random_state=0)
#利用KMeans.fit函数对构建的数据集进行K均值聚类，选取聚类簇数K=4
kmeans = KMeans(n_clusters=4).fit(X)
y_pre = kmeans.predict(X)#predict函数返回每个样本所属的簇
para = kmeans.get_params()#get_params函数返回模型参数
plt.scatter(X[:, 0], X[:, 1], c=y_pre)
centers = kmeans.cluster_centers_#cluster_centers_函数返回聚类算法的四个簇中心
plt.scatter(centers[:,0], centers[:, 1], c='black', s=100, marker='+')
plt.show()#绘制聚类效果图
```

聚类效果图如图 1.9 所示，不同颜色表示样本属于不同簇。

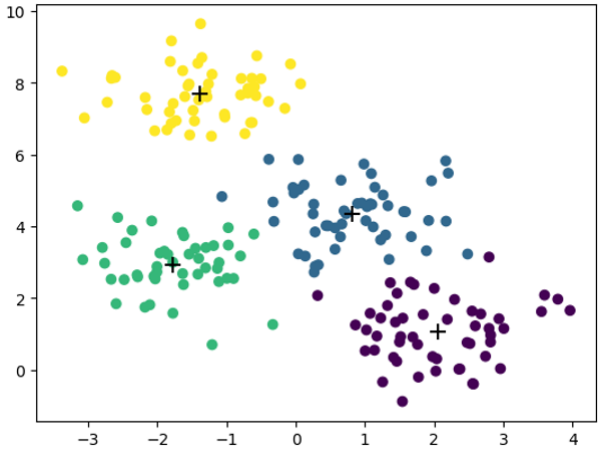


图 1.9 聚类效果图

模糊 C 聚类算法

本小节调用 skfuzzy.cluster 库中的 cmeans 函数实现 FCM 算法，代码实现如下：

```
import numpy as np
```

```
from skfuzzy.cluster import cmeans
data = np.random.uniform(1, 100, (2, 100))#在1到100中随机生成包含2个特征的100个样本
#调用cmeans函数实现FCM聚类，参数c为聚类的簇数；error为误差阈；maxiter为最大迭代次数
center, U, U0, dist, mj, knum, evalue = cmeans(data, m=2, c=3,
error=0.001, maxiter=10000)
```

其中 cmeans 函数输出参数：center 为聚类中心；U 为最终隶属矩阵；U0 为初始隶属矩阵；dist 为样本点到聚类中心的距离矩阵；mj 为目标函数值；knum 为迭代次数；evalue 为评价指标，越接近 1 效果越好。

```
import matplotlib.pyplot as plt
for i in U:
    label = np.argmax(U, axis=0)  #根据输出结果，将样本点划分到不同簇
for i in range(np.shape(data)[1]):
    if label[i] == 0:
        plt.scatter(data[0][i], data[1][i], c='b')
    elif label[i] == 1:
        plt.scatter(data[0][i], data[1][i], c='g')
    elif label[i] == 2:
         plt.scatter(data[0][i], data[1][i], c='r')
plt.show()   #绘制聚类结果图
```

FCM 聚类结果图，如图 1.10 所示。

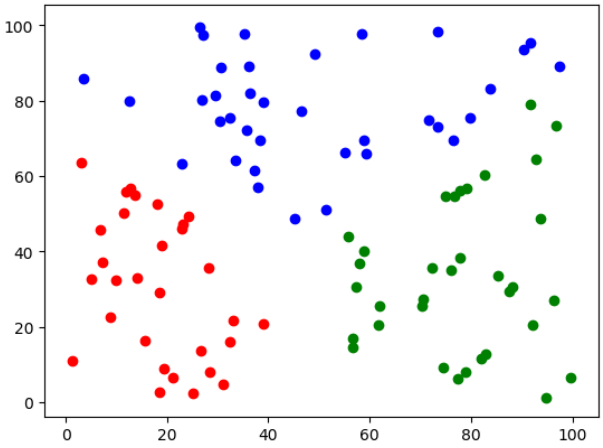


图 1.10 FCM 聚类结果图

高斯混合聚类算法

本小节将利用 sklearn 实现高斯混合聚类算法，实现代码如下：

```
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture
```

```
from matplotlib import pyplot  as plt
#调用sklearn库中的make_blobs函数建立数据集。特征维数设置为2，样本数设置为200
X, y = make_blobs(n_features=2, n_samples=200, centers=4,
cluster_std=0.80, random_state=0)
#GaussianMixture函数实现高斯混合聚类算法。混合成分k取2，采用fit方法作用于原始样本集
gmm = GaussianMixture(n_components=2, random_state=0).fit(X)
para = gmm.get_params()
y_pre = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pre)
plt.show() #绘制聚类散点图
```

高斯混合聚类效果图，如图 1.11 所示。

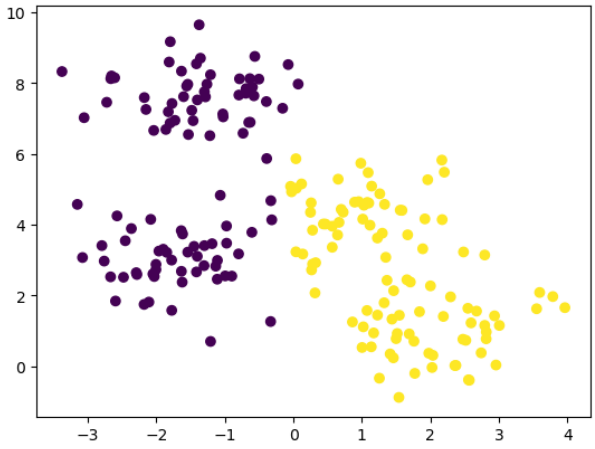


图 1.11 聚类效果图

层次聚类算法

本小节以聚合聚类算法为例，利用 sklearn 库中 AgglomerativeClustering 函数实现算法整体流程，实现代码如下：首先导入相应模块，建立数据集

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot  as plt
from sklearn.cluster import AgglomerativeClustering as Agc
#基于make_blobs函数建立数据集。设置特征维数为2，数据集中样本数为200
X, y = make_blobs(n_features=2, n_samples=200, centers=5,
cluster_std=1, random_state=0)
```

再采用 AgglomerativeClustering.fit 函数进行聚合聚类

```
#聚类簇数设置为5
model = Agc(n_clusters=5).fit(X)
para = model.get_params()#返回模型参数
y_pred = model.labels_#返回样本所属的簇
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()#绘制聚合聚类效果图
```

聚合聚类效果图，如图 1.12 所示，不同颜色表示样本属于不同簇。

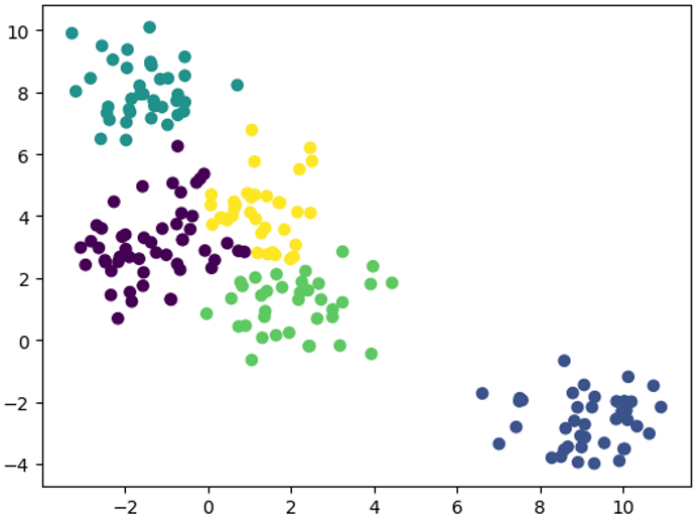


图 1.12 聚合聚类效果图

DBSCAN 聚类算法

本节我们采用 sklearn.cluster 中的 DBSCAN 方法，展示 DBSCAN 算法的实现过程，并与 K 均值算法进行对比。实现过程如下：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles,make_blobs
#调用make_circles方法生成包含1000个点的环形数据子集
X1, y1 = make_circles(n_samples=1000, factor=0.5, noise=0.03)
#调用make_blobs方法生成包含1000个点并以(1.3, 1.3)为中心的数据子集
X2, y2 = make_blobs(n_samples=1000, centers=[[1.3,1.3]],
cluster_std=[[.04]], random_state=6)
X = np.concatenate((X1, X2))#将两个数据子集进行拼接得到我们要分析的数据集合
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()#绘制数据集散点图
```

数据集散点图如图1.13所示。

首先采用 K 均值算法对上述数据集进行聚类分析，得到聚类结果如图1.14，可以看出 K 均值聚类不能较好地捕捉此数据集的内在特征。

```
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
y_pred = KMeans(n_clusters=3).fit_predict(X)#聚类簇数k取3
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
```

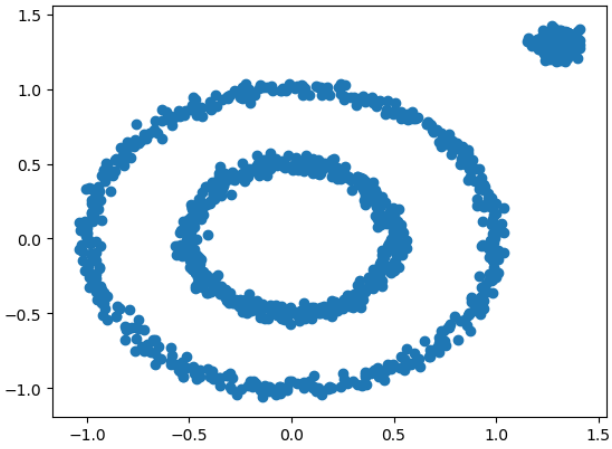


图 1.13 数据集散点图

```
plt.show()
```

再采用 DBSCAN 算法对上述数据集进行聚类，得到聚类结果图如图1.15，可以看出 DBSCAN 算法将数据集分为三个簇，较好地捕捉到数据的特征。

```
y_pred = DBSCAN(eps = 0.1, min_samples = 10).fit_predict(X)#选取eps为0.1，PNum为10
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

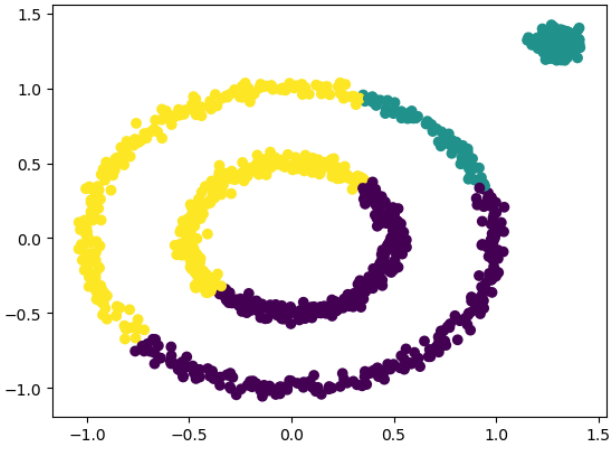


图 1.14 K 均值聚类结果图

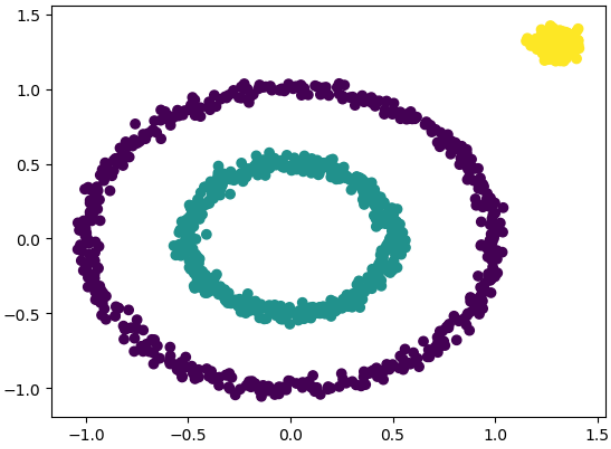


图 1.15 DBSCAN 聚类结果图

双向聚类算法

本小节采用 sklearn.cluster 中的 SpectralCoclustering 方法，展示双向聚类算法的实现过程。首先建立原始数据集，实现代码如下：

```
from matplotlib import pyplot as plt
from sklearn.datasets import make_biclusters
from sklearn.cluster import SpectralCoclustering
from sklearn.metrics import consensus_score
data, rows, columns = make_biclusters(shape=(300, 300), n_clusters=5,
noise=5, shuffle=False, random_state=0)
plt.matshow(data, cmap=plt.cm.Blues)#构建原始数据集
plt.title("Original dataset")#绘制原始数据集
```

绘制出原始数据集如图1.16所示。

然后将原始数据集打乱：

```
rng = np.random.RandomState(0)
row_idx = rng.permutation(data.shape[0])
col_idx = rng.permutation(data.shape[1])
data = data[row_idx][:, col_idx]#打乱原始数据集
plt.matshow(data, cmap=plt.cm.Blues)
plt.title("Shuffled dataset")#绘制打乱后的数据集
```

打乱后的数据集图像1.17所示。下一步利用双向聚类算法对打乱后的数据集重新排列：

```
model = SpectralCoclustering(n_clusters=5, random_state=0).fit(data)
#调用SpectralCoclustering函数重新排列打乱后的数据集
score = consensus_score(model.biclusters_ , (rows[:, row_idx], columns[:, col_idx]))
#计算前后两种双向聚类的相似性
```

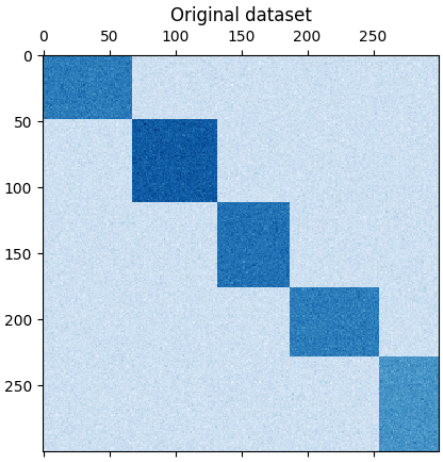


图 1.16 原始数据集

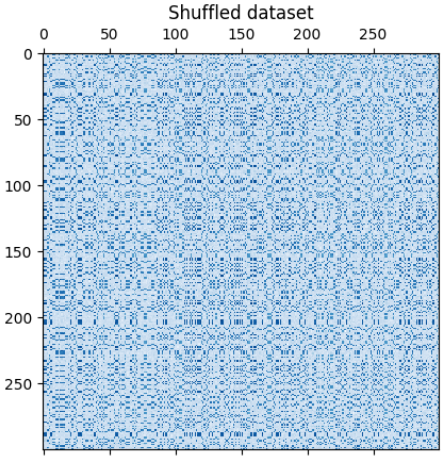


图 1.17 打乱后的数据集

```
print("consensus score: {:.3f}".format(score))#输 出 结 果
fit_data = data[np.argsort(model.row_labels_), :]
fit_data = fit_data[:, np.argsort(model.column_labels_)]
plt.matshow(fit_data , cmap=plt.cm.Blues)
plt.title("After biclustering; rearranged to show biclusters")
plt.show()#绘 制 利 用 双 向 聚 类 算 法 后 的 数 据 集
```

重排后的数据集如图1.18所示。可以看出双向聚类算法在本例中有较好的应用效果。

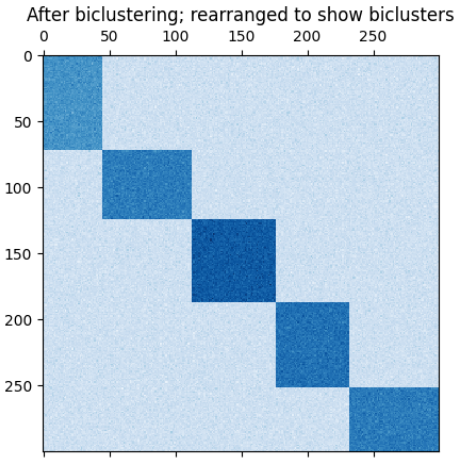


图 1.18 双向聚类算法效果

总结

本章介绍了多种常用的聚类算法。K 均值聚类算法采用距离最近原则将样本集划分到不同簇中，并计算每个簇中样本的均值作为簇中心，基于 K 均值的思想，发展了较多变种算法，例如：处理分类属性的 K-modes 算法 [1]、可加快迭代过程的 X-Means 算法 [2] 等。

相比于 K 均值聚类非黑即白的硬聚类过程，FCM 和高斯混合聚类算法采用软聚类的思想，通过计算样本属于某一簇划分的可能性，并且根据可能性大小进行簇划分。这类聚类算法在一些场景中得到较好的聚类效果，被广泛应用于实际问题。

层次聚类算法通过对簇进行分裂或者合并，达到聚类的目标，本章主要介绍了聚合聚类的算法流程，分裂聚类与之相反，算法是由一个簇生成多个簇的过程。相关改进算法有：BIRCH [3] 等。

DBSCAN 算法从样本集分布紧密程度的角度出发进行簇划分，分布越紧密越有可能被划入同一簇。此方法可应用于非凸样本的聚类分析。更多基于密度的聚类可参考：ST-DBSCAN [4]、OPTICS [5]。

可处理混合型数据的聚类方法，除本章介绍的 K-prototypes 之外，其他方法可参考：ClustMD [6]、KAMILA [7] 等；更多双向聚类方法可参考：谱双向聚类 [8]、信息双向聚类 [9]、凸双向聚类 [10] 等。

1.10 习题

1、证明：当 $t < 1$ 时，式 (1.2.1) 不是距离度量。

2、在 K 均值聚类方法中，可采用指示函数法确定最优 K 值，请写出误差平方和、平均直径、平均半径的指示函数形式。

3、证明 K 均值聚类算法的收敛性。

4、试给出 FCM 聚类算法步骤的证明过程。

5、在求解高斯混合模型参数时，我们用到了 EM 算法。下面请叙述 EM 算法一般化步骤并给出相关证明。

6、参考聚合聚类算法，试写出分裂聚类的算法流程，并采用分裂聚类对例题1.1进行分析。

7、分别采用 K 均值聚类、模糊 C 均值聚类、高斯混合聚类、层次聚类、DBSCAN 聚类分析 2019 年中国主要城市平均气温，并对结果进行分析与对比。（数据来源：http://www.stats.gov.cn/tjsj/ndsj/2020/indexch.htm。数据表名称：8-5 主要城市平均气温（2019 年））。

8、编写程序实现 K-prototypes 算法，作用于 UCI 数据集 German Credit Data（https://archive.ics.uci.edu/ml/datasets/Statlog+（German+Credit+Data）），并对结果进行分析。（注意：数据集最后一列为类别标签，聚类过程不予采用）

第二章　主成分分析

主成分分析和聚类分析都是对数据进行降维的无监督学习方法，聚类的目标是发现数据中的潜在群组，使得同一组内的数据相似度较高，不同组之间的相似度较低。它通常用于发现数据中的内在结构和模式，而不关心特征之间的关系。而主成分分析（PCA）目标是找到数据中的主要成分，通过线性变换将数据投影到一个新的坐标系中，找到数据中的主要方差方向，以便用较少的维度来表示数据。

2.1　简介

在实际问题的研究中，往往涉及多变量，且不同变量之间有一定的相关性。当变量较多时，将增加分析问题的复杂性。由于变量较多且变量之间存在相关性，使得观测到的数据在一定程度上有所重叠。因此利用变量间信息的重叠，通过变量的降维，可以使得复杂问题得到简化。

主成分分析（principal component analysis，PCA）是利用降维的思想，在尽量减少信息损失的前提下，将多变量转化为少数几个综合变量的一种机器学习方法。例如，在商品经济中，用主成分分析可以将复杂的经济数据综合成几个商业指数，如物价指数、生活费用指数以及商业活动指数等。主成分分析是由 Pearson [11] 提出，后来被 Hotelling [12] 发展起来的。通常将生成的综合变量称为主成分，这些主成分保留原始变量的绝大部分信息，都是原始变量的线性组合，而且各个主成分之间互不相关。在研究复杂问题时，通过主成分分析，可以从事物错综复杂的关系中找出一些主要成分，揭示事物内部变量之间的规律，简化问题，提高分析效率。

2.2　总体的主成分

2.2.1　总体主成分的定义

主成分分析的目标是找到原始变量的一个能够按照“重要性”排序并且信息不重复的线性组合。具体地，假设 $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)^{\mathrm{T}}$ 为 p 维随机向量，其均

值为 $\boldsymbol{\mu}$，协方差矩阵为 $\boldsymbol{\Sigma}$。对 $\boldsymbol{X}$ 进行线性变换，可以形成新的综合变量，记为 $\boldsymbol{Y}=(Y_1,Y_2,\cdots,Y_p)^{\mathrm{T}}$，即

$$\begin{cases} Y_1 &= a_{11}X_1+a_{21}X_2+\cdots+a_{p1}X_p=\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{X} \\ Y_2 &= a_{12}X_1+a_{22}X_2+\cdots+a_{p2}X_p=\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{X} \\ &\vdots \\ Y_p &= a_{1p}X_1+a_{2p}X_2+\cdots+a_{pp}X_p=\boldsymbol{a}_p\boldsymbol{X} \end{cases}$$

首先用一个综合变量 Y_1 来替代原始的 p 个变量，为了使得 Y_1 在 $X_1,X_2,\cdots,X_p$ 的所有线性组合中最具代表性，应使其方差最大化，以最大程度地保留原始变量的方差和协方差结构信息。由于

$$\mathrm{var}(Y_1)=\mathrm{var}(\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{X})=\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_1$$

若对 $\boldsymbol{a}_1$ 不加以约束，可使得 Y_1 的方差任意增大，那么方差最大化就变得没有意义。因此主成分分析限制 $\boldsymbol{a}_1$ 为单位向量，即 $\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1=1$，寻求向量 $\boldsymbol{a}_1$，使得 $\mathrm{var}(Y_1)=\mathrm{var}(\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{X})$ 达到最大，Y_1 就称为第一主成分。如果第一主成分所含信息不够多，不足以代表原始数据的 p 个变量，则需要考虑 Y_2，为了使得 Y_2 中所含信息与 Y_1 不重叠，则应该要求

$$\mathrm{cov}(Y_1,Y_2)=0$$

即 Y_1 和 Y_2 不相关。因此主成分分析在上述约束和条件 $\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_2=1$ 的条件下寻求 $\boldsymbol{a}_2$，使得 $\mathrm{var}(Y_2)=\mathrm{var}(\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{X})$ 达到最大，所求的 Y_2 称为第二主成分。类似地，可以定义第三主成分，$\cdots$，第 p 主成分。各主成分在总方差中所占比重依次递减。实际应用中，通常只需挑选前几个主成分，达到简化问题，抓住问题本质的目的。

2.2.2 总体主成分的求法

本节将阐述求解 $\boldsymbol{X}$ 主成分的计算过程。假设 $\boldsymbol{\Sigma}$ 是 $\boldsymbol{X}=(X_1,X_2,\cdots,X_p)^{\mathrm{T}}$ 的协方差矩阵，$\boldsymbol{\Sigma}$ 的特征值及其相应的正交单位化特征向量分别为 $\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_r>\lambda_{r+1}=\cdots=\lambda_p=0$ 及 $\boldsymbol{a}_1,\cdots,\boldsymbol{a}_p$，其中 $r=\mathrm{rank}(\boldsymbol{\Sigma})$。首先求出 $\boldsymbol{X}$ 的第一个主成分 $Y_1=\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{X}$。由于第一主成分的系数 $\boldsymbol{a}_1$ 应在条件 $\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1=1$ 下，使得 $\boldsymbol{X}$ 的所有线性变换中方差

$$\mathrm{var}(\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{X})=\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_1$$

最大化。因此，求第一主成分就转换为求解以下约束最优化问题：

$$\max_{\boldsymbol{a}_1}\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_1$$

$$\text{s.t.}\qquad \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1=1$$

根据拉格朗日乘子法，定义拉格朗日函数

$$L(\boldsymbol{a}_1, \lambda) = \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_1 - \lambda(\boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{a}_1 - 1)$$

其中 λ 为拉格朗日乘子。将拉格朗日函数 $L(\boldsymbol{a}_1, \lambda)$ 分别对参数 $\boldsymbol{a}_1, \lambda$ 求导，令其为 0，即得：

$$\begin{cases} \boldsymbol{\Sigma} \boldsymbol{a}_1 - \lambda \boldsymbol{a}_1 = 0 \\ \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{a}_1 = 1 \end{cases}$$

因此，λ 是协方差矩阵 $\boldsymbol{\Sigma}$ 的特征值，$\boldsymbol{a}_1$ 是其对应的单位特征向量。可得目标函数

$$\boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_1 = \boldsymbol{a}_1^{\mathrm{T}} \lambda \boldsymbol{a}_1 = \lambda \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{a}_1 = \lambda$$

因此如果 $\boldsymbol{a}_1$ 是 $\boldsymbol{\Sigma}$ 的最大特征值 λ_1 对应的单位特征向量，则 $\boldsymbol{a}_1$ 与 λ_1 是上述最优化问题的解。即可得第一个主成分 $Y_1 = \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{X}$，其方差为协方差矩阵 $\boldsymbol{\Sigma}$ 的最大特征值 λ_1，其系数 $\boldsymbol{a}_1$ 是 λ_1 对应的单位特征向量。

下面求解 $\boldsymbol{X}$ 的第二个主成分 $Y_2 = \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{X}$。由于第二个主成分的系数 $\boldsymbol{a}_2$ 应满足以下条件：单位向量 $\boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_2 = 1$，且 $Y_2 = \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{X}$ 与 $Y_1 = \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{X}$ 不相关，并使得 $\boldsymbol{X}$ 的所有线性变换中方差

$$\mathrm{var}(\boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{X}) = \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_2$$

达到最大。因此，求第二主成分就转换为求解以下约束最优化问题：

$$\max_{\boldsymbol{a}_2} \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_2$$

$$\text{s.t.} \qquad \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_2 = 1, \quad \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_2 = 0,$$

由于

$$\boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_2 = \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_1 = \boldsymbol{a}_2^{\mathrm{T}} \lambda_1 \boldsymbol{a}_1 = \lambda_1 \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_1 = \lambda_1 \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{a}_2$$

则有

$$\boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_1 = 0, \quad \boldsymbol{a}_1^{\mathrm{T}} \boldsymbol{a}_2 = 0$$

定义拉格朗日函数

$$L(\boldsymbol{a}_2, \lambda, \phi) = \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{a}_2 - \lambda(\boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_2 - 1) - \phi \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_1$$

其中 λ, ϕ 为拉格朗日乘子。将拉格朗日函数 $L(\boldsymbol{a}_2, \lambda, \phi)$ 分别对参数 $\boldsymbol{a}_2, \lambda, \phi$ 求导，令其为 0，即得：

$$\begin{cases} 2\boldsymbol{\Sigma} \boldsymbol{a}_2 - 2\lambda \boldsymbol{a}_2 - \phi \boldsymbol{a}_1 = \boldsymbol{0} \\ \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_2 = 1 \\ \boldsymbol{a}_2^{\mathrm{T}} \boldsymbol{a}_1 = 0 \end{cases}$$

由于

$$2\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_2 - 2\lambda\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_2 - \phi\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1 = 0$$

则 $\phi = 0$，进而可以得到

$$\boldsymbol{\Sigma}\boldsymbol{a}_2 - \lambda\boldsymbol{a}_2 = 0$$

显然，λ 是协方差矩阵 $\boldsymbol{\Sigma}$ 的特征值，$\boldsymbol{a}_2$ 是其对应的单位特征向量。此时，目标函数可表示为

$$\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_2 = \boldsymbol{a}_2^{\mathrm{T}}\lambda\boldsymbol{a}_2 = \lambda\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_2 = \lambda$$

因此如果 $\boldsymbol{a}_2$ 是 $\boldsymbol{\Sigma}$ 的第二大特征值 λ_2 对应的单位特征向量，则 $\boldsymbol{a}_2$ 与 λ_2 是上述最优化问题的解。即可得第二个主成分 $Y_2 = \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{X}$，其方差为协方差矩阵 $\boldsymbol{\Sigma}$ 的第二大特征值 λ_2，系数向量 $\boldsymbol{a}_2$ 是 λ_2 对应的单位特征向量。

以此类推，可知第 k 个主成分 $Y_k = \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{X}$，其方差为协方差矩阵 $\boldsymbol{\Sigma}$ 的第 k 大特征值 λ_k，系数向量 $\boldsymbol{a}_k$ 是 λ_k 对应的单位特征向量。

因此，假设 $\boldsymbol{X}$ 的第 k 个主成分为

$$Y_k = \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{X} = a_{k1}X_1 + a_{k2}X_2 + \cdots + a_{kp}X_p$$

其中 $\boldsymbol{a}_k = (a_{k1}, \cdots, a_{kp})^{\mathrm{T}}$。显然有：

$$\begin{cases} \mathrm{var}(Y_k) = \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_k = \lambda_k\boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{a}_k = \lambda_k, k = 1, \cdots, p \\ \mathrm{cov}(Y_k, Y_j) = \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{a}_j = \lambda_k\boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{a}_j = 0, k \neq j \end{cases}$$

即令 $\boldsymbol{A} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_p)$，则 $\boldsymbol{A}$ 是一个正交矩阵，且 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{A} = \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_p)$，其中 $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_p)$ 表示对角矩阵。因此，求主成分问题就转化成了求协方差矩阵的特征值和特征向量。

2.2.3 总体主成分的性质

1、主成分的协方差矩阵 $\mathrm{var}(\boldsymbol{Y}) = \boldsymbol{\Lambda}$。即 $\mathrm{var}(Y_j) = \lambda_j, j = 1, \cdots, p$ 且 $Y_1, Y_2, \cdots, Y_p$ 互不相关。

2、假设 $\boldsymbol{\Sigma} = (\sigma_{jk})_{p\times p}$ 表示 $\boldsymbol{X}$ 的协方差矩阵，则总体主成分的方差之和可表示为：

$$\sum_{j=1}^{p}\lambda_j = \sum_{j=1}^{p}\sigma_{jj}$$

事实上，由于 $\boldsymbol{\Sigma} = \boldsymbol{A}\Lambda\boldsymbol{A}^{\mathrm{T}}$，则

$$\sum_{j=1}^{p}\lambda_j = \mathrm{tr}(\boldsymbol{\Lambda}) = \mathrm{tr}(\boldsymbol{\Lambda}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) = \mathrm{tr}(\boldsymbol{A}\boldsymbol{\Lambda}\boldsymbol{A}^{\mathrm{T}}) = \mathrm{tr}(\boldsymbol{\Sigma}) = \sum_{j=1}^{p}\sigma_{jj}$$

由此可知，主成分分析是把 p 个随机变量 $X_1, X_2, \cdots, X_p$ 的总方差分解为 p 个不相关的随机变量 $Y_1, Y_2, \cdots, Y_p$ 的方差之和。

在主成分分析中，令 η_k 表示第 k 个主成分的方差贡献率，定义为：

$$\eta_k = \frac{\lambda_k}{\sum_{j=1}^{p} \lambda_j}, \quad k = 1, \cdots, p$$

其含义是第 k 个主成分 Y_k 所提取的信息占总信息的比例。根据主成分分析的算法原理，第一主成分的贡献率最大，意味着 Y_1 综合原始变量 $X_1, X_2, \cdots, X_p$ 所含的信息能力最强，而 $Y_2, Y_3, \cdots, Y_p$ 的综合能力依次减弱。

前 m 个主成分 $Y_1, \cdots, Y_m$ 的方差贡献率之和定义为：

$$\sum_{j=1}^{m} \eta_j = \frac{\sum_{j=1}^{m} \lambda_j}{\sum_{j=1}^{p} \lambda_j}$$

表示主成分 $Y_1, \cdots, Y_m$ 的累积贡献率，其含义是前 m 个主成分综合提供原始变量信息的能力。在实际应用中，通常选取 $m < p$，使得前 m 个主成分的累积贡献率达到较高的比例（例如，大于 85%）。此时，用 $Y_1, \cdots, Y_m$ 代替原始随机变量 $X_1, X_2, \cdots, X_p$ 不但使得变量的维数降低，而且也不损失太多的信息。

3、设矩阵 $\boldsymbol{A}$ 的第 k 行第 j 列元素为 $\boldsymbol{A}_{kj}$。由于 $\boldsymbol{Y} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}$，则有 $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Y}$，故而有 $X_j = \boldsymbol{A}_{j1}Y_1 + \boldsymbol{A}_{j2}Y_2 + \cdots + \boldsymbol{A}_{jp}Y_p$，$\mathrm{cov}(Y_k, X_j) = \lambda_k \boldsymbol{A}_{jk}$，则可得主成分 Y_k 与原始变量 X_j 的相关系数为

$$\rho_{Y_k, X_j} = \frac{\mathrm{cov}(Y_k, X_j)}{\sqrt{\mathrm{var}(Y_k)}\sqrt{\mathrm{var}(X_j)}} = \frac{\lambda_k \boldsymbol{A}_{jk}}{\sqrt{\lambda_k}\sqrt{\sigma_{jj}}} = \frac{\sqrt{\lambda_k}}{\sqrt{\sigma_{jj}}}\boldsymbol{A}_{jk}$$

它给出了主成分 Y_k 与原始变量 X_j 的线性关联性的度量，也称为因子负荷量或因子载荷量。

4、之前所提到的累积贡献率度量了前 m 个主成分 $Y_1, \cdots, Y_m$ 综合提供原始变量 $X_1, X_2, \cdots, X_p$ 所含的信息能力，那么前 m 个主成分中包含原始变量 X_j 有多少信息应该如何度量呢？这个指标为前 m 个主成分 $Y_1, \cdots, Y_m$ 与原始变量 X_j 的相关系数的平方和，我们称之为 $Y_1, \cdots, Y_m$ 对原始变量 X_j 的贡献率。

下面我们通过一个例子阐述总体主成分的计算方法。

例 2.1 设随机变量 $\boldsymbol{X} = (X_1, X_2, X_3)^{\mathrm{T}}$ 的协方差矩阵为

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & -2 & 0 \\ -2 & 5 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

则知 $\boldsymbol{\Sigma}$ 的特征值为 $\lambda_1 = 3 + \sqrt{8}, \lambda_2 = 2, \lambda_3 = 3 - \sqrt{8}$，相应的单位正交特征向量为

$$\boldsymbol{a}_1 = \begin{bmatrix} 0.383 \\ -0.924 \\ 0.000 \end{bmatrix}, \quad \boldsymbol{a}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{a}_3 = \begin{bmatrix} 0.924 \\ 0.383 \\ 0.000 \end{bmatrix}$$

因此，主成分为

$$Y_1 = 0.383X_1 - 0.924X_2$$

$$Y_2 = X_3$$

$$Y_3 = 0.924X_1 + 0.383X_2$$

取 $m = 1$ 时，Y_1 的累积贡献率为 $\dfrac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3} = \dfrac{3+\sqrt{8}}{8} = 72.8\%$；取 $m = 2$ 时，Y_1, Y_2 的累积贡献率为 97.85%。下表列出 m 个主成分对变量 X_j 的贡献率。

j	ρ_{Y_1,X_j}	$\rho^2_{Y_1,X_j}$	ρ_{Y_2,X_j}	$\rho^2_{Y_2,X_j}$
1	0.925	0.855	0.000	0.000
2	−0.998	0.996	0.000	0.000
3	0.000	0.000	1.000	1.000

由此可见，当 $m = 1$ 时，Y_1 的累积贡献率已达 72.8%，但是 Y_1 对 X_3 的贡献率为 0，这是因为在 Y_1 中没有包含 X_3 的任何信息，因此仅取 $m = 1$ 不够，故而取 $m = 2$，这时 Y_1, Y_2 的累积贡献率为 97.85%，且 Y_1, Y_2 对 $X_j (j = 1, 2, 3)$ 的贡献率也比较高。

2.2.4 标准化变量的主成分

在实际问题中，通常有两种情形不适合直接从协方差矩阵 $\boldsymbol{\Sigma}$ 出发求主成分。一种是各变量的单位不全相同，对同样的变量使用不同的单位直接进行主成分分析，其结果一般是不一样的，甚至差异较大，这样做出来的分析也没有意义。另一种是各变量的单位虽相同，但是其变量方差的差异较大，以至于主成分分析的结果往往倾向于方差大的变量，而方差小的变量几乎被忽略。因此，对这两种情形，通常先将各原始变量做标准化处理，然后从标准化变量的协方差矩阵出发求主成分。常用的标准化变换为：

$$X_j^* = \frac{X_j - \mu_j}{\sqrt{\sigma_{jj}}}, i = 1, \cdots, p$$

其中 $\mu_j = E(X_j), \sigma_{jj} = \mathrm{var}(X_j)$。此时 $\boldsymbol{X}^* = (X_1^*, X_2^*, \cdots, X_p^*)^{\mathrm{T}}$ 的协方差矩阵为原始变量 $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)^{\mathrm{T}}$ 的相关系数矩阵 $\boldsymbol{\rho} = (\rho_{kj})_{p\times p}$，其中

$$\rho_{kj} = \frac{\mathrm{cov}(X_k, X_j)}{\sqrt{\mathrm{var}(X_k)}\sqrt{\mathrm{var}(X_j)}}$$

因此只需直接从相关系数矩阵 $\boldsymbol{\rho}$ 出发求主成分，此时的主成分分析将均等地对待每一个原始变量。从相关系数矩阵出发求的主成分与从协方差矩阵出发是完全类似的，并且主成分的一些性质具有更简单的数学形式。

设 $\boldsymbol{\rho}$ 的特征值 $\lambda_1^* \geq \lambda_2^* \geq \cdots \geq \lambda_r^* > \lambda_{r+1}^* = \cdots = \lambda_p^* = 0$，其中 $r = \text{rank}(\boldsymbol{\rho})$，$\boldsymbol{\rho}$ 的 p 个单位特征向量为 $\boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_p^*$，且相互正交，则 p 个主成分为：$Y_1^* = \boldsymbol{a}_1^{*\mathrm{T}}\boldsymbol{X}^*, Y_2^* = \boldsymbol{a}_2^{*\mathrm{T}}\boldsymbol{X}^*, \cdots, Y_p^* = \boldsymbol{a}_p^{*\mathrm{T}}\boldsymbol{X}^*$。记 $\boldsymbol{Y}^* = (Y_1^*, Y_2^*, \cdots, Y_p^*)^{\mathrm{T}}$，$\boldsymbol{A}^* = (\boldsymbol{a}_1^*, \boldsymbol{a}_2^*, \cdots, \boldsymbol{a}_p^*)$，则有 $\boldsymbol{Y}^* = \boldsymbol{A}^{*\mathrm{T}}\boldsymbol{X}^*$。上述主成分具有的性质可以概括如下：

1、$E(\boldsymbol{Y}^*) = \boldsymbol{0}, \text{var}(\boldsymbol{Y}^*) = \boldsymbol{\Lambda}^* = \text{diag}(\lambda_1^*, \cdots, \lambda_p^*)$。

2、$\sum_{j=1}^{p} \text{var}(Y_j^*) = \sum_{j=1}^{p} \lambda_j^* = \sum_{j=1}^{p} \text{var}(X_j^*) = p$。

3、第 k 个主成分 Y_k^* 的贡献率为 $\dfrac{\lambda_k^*}{p}$，前 m 个主成分 $Y_1^*, \cdots, Y_m^*$ 的累积贡献率为 $\dfrac{\sum_{j=1}^{m} \lambda_j^*}{p}$。

4、主成分 Y_k^* 与 X_j^* 的相关系数为 $\rho_{Y_k^*, X_j^*} = \sqrt{\lambda_k^*}\boldsymbol{A}_{jk}^*$。

下面通过一个例子说明分别从协方差矩阵和相关系数矩阵出发求主成分的差异。

例 2.2 随机变量 $\boldsymbol{X} = (X_1, X_2, X_3)^{\mathrm{T}}$ 的协方差矩阵为

$$\boldsymbol{\Sigma} = \begin{bmatrix} 16 & 2 & 30 \\ 2 & 1 & 4 \\ 30 & 4 & 100 \end{bmatrix}$$

其相关系数矩阵为

$$\boldsymbol{\rho} = \begin{bmatrix} 1.00 & 0.50 & 0.75 \\ 0.50 & 1.00 & 0.40 \\ 0.75 & 0.40 & 1.00 \end{bmatrix}$$

经计算可知 $\boldsymbol{\Sigma}$ 的特征值为 $\lambda_1 = 109.793, \lambda_2 = 6.469, \lambda_3 = 0.738$，相应的单位正交特征向量为

$$\boldsymbol{a}_1 = \begin{bmatrix} 0.305 \\ 0.041 \\ 0.951 \end{bmatrix}, \quad \boldsymbol{a}_2 = \begin{bmatrix} 0.944 \\ 0.120 \\ -0.308 \end{bmatrix}, \quad \boldsymbol{a}_3 = \begin{bmatrix} -0.127 \\ 0.992 \\ -0.002 \end{bmatrix}$$

因此，主成分为

$$Y_1 = 0.305X_1 + 0.041X_2 + 0.951X_3$$

$$Y_2 = 0.944X_1 + 0.120X_2 - 0.308X_3$$

$$Y_3 = -0.127X_1 + 0.992X_2 - 0.002X_3$$

Y_1 的贡献率为 $\dfrac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} = 0.938$。如此高的贡献率归因于 X_1 的大方差，以及 X_1, X_2, X_3 之间存在一定的相关性。

相关系数矩阵 $\boldsymbol{\rho}$ 的特征值为 $\lambda_1^*=2.114,\lambda_2^*=0.646,\lambda_3^*=0.240$，相应的单位正交特征向量为

$$\boldsymbol{a}_1^*=\begin{bmatrix}0.627\\0.497\\0.600\end{bmatrix},\quad \boldsymbol{a}_2^*=\begin{bmatrix}-0.241\\0.856\\-0.457\end{bmatrix},\quad \boldsymbol{a}_3^*=\begin{bmatrix}-0.741\\0.142\\0.656\end{bmatrix}$$

因此，主成分为

$$\begin{aligned}Y_1^*&=0.627X_1^*+0.497X_2^*+0.600X_3^*\\Y_2^*&=-0.241X_1^*+0.856X_2^*-0.457X_3^*\\Y_3^*&=-0.741X_1^*+0.142X_2+0.656X_3^*\end{aligned}$$

Y_1^* 的贡献率为 $\dfrac{\lambda_1^*}{3}=0.705$，$Y_1^*$ 和 Y_2^* 的累积贡献率为 $\dfrac{\lambda_1^*+\lambda_2^*}{3}=0.920$。比较从 $\boldsymbol{\Sigma}$ 出发和从 $\boldsymbol{\rho}$ 出发的主成分分析结果。可知从 $\boldsymbol{\rho}$ 出发的 Y_1^* 的贡献率 0.705 明显小于从 $\boldsymbol{\Sigma}$ 出发的 Y_1 的贡献率 0.938，事实上，原始变量方差之间的差异越大，这一点往往越明显。此例也说明标准化后的结论可能会发生很大的变化，因此标准化并不是无关紧要的。

2.3 样本主成分

在上一节，我们可以从协方差矩阵 $\boldsymbol{\Sigma}$ 或相关系数矩阵 $\boldsymbol{\rho}$ 出发求主成分。但是在实际问题中，$\boldsymbol{\Sigma}$ 和 $\boldsymbol{\rho}$ 一般都是未知的，需要通过样本来进行估计得到。设

$$\boldsymbol{X}_i=(X_{i1},\cdots,X_{ip})^{\mathrm{T}},i=1,\cdots,n$$

为取自样本数据矩阵 $\mathbf{X}=(\boldsymbol{X}_1,\cdots,\boldsymbol{X}_n)^{\mathrm{T}}$ 的一个简单随机样本，其中 p 表示特征维数，n 表示样本数。样本的协方差矩阵和相关系数矩阵分别为

$$\begin{aligned}\boldsymbol{S}&=(s_{kj})_{p\times p}=\frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{X}_i-\bar{\boldsymbol{X}})(\boldsymbol{X}_i-\bar{\boldsymbol{X}})^{\mathrm{T}}\\\boldsymbol{R}&=(r_{kj})_{p\times p}=\frac{s_{kj}}{\sqrt{s_{kk}s_{jj}}}\end{aligned}$$

其中

$$\bar{\boldsymbol{X}}=(\bar{X}_1,\cdots,\bar{X}_p)^{\mathrm{T}},\quad \bar{X}_j=\frac{1}{n}\sum_{i=1}^{n}X_{ij},\quad j=1,\cdots,p,$$

$$s_{kj}=\frac{1}{n-1}\sum_{i=1}^{n}(X_{ik}-\bar{X}_k)(X_{ij}-\bar{X}_j)\quad k,j=1,\cdots,p$$

分别以 $\boldsymbol{S},\boldsymbol{R}$ 作为 $\boldsymbol{\Sigma},\boldsymbol{\rho}$ 的估计，再按照总体主成分的方法求得的主成分称之为样本主成分。

设 $\widehat{\lambda}_1 \geq \widehat{\lambda}_2 \geq \cdots \geq \widehat{\lambda}_r > \widehat{\lambda}_{r+1} = \cdots = \widehat{\lambda}_p = 0$ 为样本协方差矩阵 $\boldsymbol{S}$ 的特征值，$\widehat{\boldsymbol{a}}_1,\cdots,\widehat{\boldsymbol{a}}_p$ 为对应的正交单位化特征向量，其中 $\widehat{\boldsymbol{a}}_i = (\widehat{a}_{1i},\cdots,\widehat{a}_{pi})^{\mathrm{T}}$，则第 m 个样本的第 j 个主成分可表示为 $Y_m j = \widehat{\boldsymbol{a}}_j^{\mathrm{T}}\boldsymbol{X}_m$。此时，可以得到

1、Y_j 的样本方差为

$$\mathrm{var}(Y_j) = \widehat{\boldsymbol{a}}_j^{\mathrm{T}}\boldsymbol{S}\widehat{\boldsymbol{a}}_j = \widehat{\lambda}_j, \quad j = 1,\cdots,p$$

2、Y_k 与 Y_j 的样本协方差为

$$\mathrm{cov}(Y_k, Y_j) = \widehat{\boldsymbol{a}}_k^{\mathrm{T}}\boldsymbol{S}\widehat{\boldsymbol{a}}_j = 0$$

3、样本总方差为

$$\sum_{j=1}^{p} s_{jj} = \sum_{j=1}^{p} \widehat{\lambda}_j$$

第 j 个主成分的贡献率为

$$\widehat{\eta}_j = \frac{\widehat{\lambda}_j}{\sum_{j=1}^{p}\widehat{\lambda}_j}, \quad j = 1,\cdots,p$$

前 k 个主成分累积贡献率为

$$\sum_{j=1}^{k}\widehat{\eta}_j = \frac{\sum_{j=1}^{k}\widehat{\lambda}_j}{\sum_{j=1}^{p}\widehat{\lambda}_j}$$

类似的，为了避免单位不统一或者变量之间差异性较大产生的影响，可以对样本进行标准化处理，即令

$$\boldsymbol{X}_i^* = \left(\frac{X_{i1}-\bar{X}_1}{\sqrt{s_{11}}}, \frac{X_{i2}-\bar{X}_2}{\sqrt{s_{22}}}, \cdots, \frac{X_{ip}-\bar{X}_p}{\sqrt{s_{pp}}}\right)^{\mathrm{T}}, i = 1,\cdots,n$$

标准化后数据的样本协方差矩阵即为原始数据的样本相关系数矩阵 $\boldsymbol{R}$。由 $\boldsymbol{R}$ 出发所求的样本主成分称为标准化样本主成分。计算 $\boldsymbol{R}$ 的特征值及相应的正交单位化特征向量即可求得标准化样本主成分。此时，标准化的样本总方差为 p.

选取前 m 个样本主成分，使其累积贡献率达到一定的要求（例如：大于 85%），用这 m 个样本主成分代替原始数据进行分析，可以达到在保留大部分信息的前提下，降低原始数据维数的目的。

2.4 非线性主成分分析

传统主成分分析一般适应于线性降维，其对于非线性数据往往不能达到较好的效果，例如，不同人之间的人脸图像存在非线性关系，用传统的线性主成分分析结果不尽人意。下面介绍几种非线性主成分分析算法。

2.4.1 核主成分分析

核主成分分析（Kernel Principal Component Analysis, KPCA）是对传统主成分分析（PCA）算法的非线性拓展。简单地说，通过将非线性不可分问题映射到维度更高的特征空间，使其在新的特征空间上线性可分。设样本数据矩阵 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，其中 p 表示特征维数，n 表示样本数。$\boldsymbol{X} \in \mathcal{R}^p$ 为取自 $\mathbf{X}$ 的一个简单随机样本，为了将其映射到维度更高的 k 维子空间，定义如下非线性映射函数 ϕ:

$$\phi : \mathcal{R}^p \to \mathcal{R}^k (p \ll k)$$

换句话说，利用 KPCA，可以通过非线性变换将数据映射到一个高维空间，然后在此高维空间中使用标准 PCA 将其映射到另外一个低维空间中，并通过线性分类器进行划分。但是，由于协方差矩阵每个元素都是向量的内积，因此映射到高维度空间后，向量维度增加导致计算量大幅度增大。故而，可以利用核函数忽略映射函数的具体形式，直接得到低维数据映射到高维后的内积。

假设 $\mathbf{X}$ 表示标准化后的样本数据矩阵，则 $\phi(\mathbf{X})$ 是一个映射后的矩阵，维数是 $n \times k$，可以计算得到协方差矩阵为

$$\boldsymbol{\Sigma} = \frac{1}{n} \phi(\mathbf{X})^{\mathrm{T}} \phi(\mathbf{X})$$

特征向量可表示为 $\boldsymbol{\nu} = \phi(\mathbf{X})^{\mathrm{T}} \boldsymbol{a}$，代入 $\boldsymbol{\Sigma\nu} = \lambda\boldsymbol{\nu}$，则可得

$$\frac{1}{n} \phi(\mathbf{X})^{\mathrm{T}} \phi(\mathbf{X}) \phi(\mathbf{X})^{\mathrm{T}} \boldsymbol{a} = \lambda \phi(\mathbf{X})^{\mathrm{T}} \boldsymbol{a}$$

两边乘以 $\phi(\mathbf{X})$，则可得

$$\frac{1}{n} \phi(\mathbf{X}) \phi(\mathbf{X})^{\mathrm{T}} \boldsymbol{a} = \lambda \boldsymbol{a}$$

即

$$\frac{1}{n} \boldsymbol{K} bola = \lambda \boldsymbol{a} \tag{2.4.1}$$

其中 $\boldsymbol{K}$ 为核矩阵：$\boldsymbol{K} = \phi(\mathbf{X}) \phi(\mathbf{X})^{\mathrm{T}}$。显然，$\boldsymbol{K}$ 由高维空间中的内积决定，为了避免由此带来的复杂计算，可通过定义核函数表示样本点在高维空间中的内积，令 $k(\boldsymbol{X}, \boldsymbol{Y})$ 表示核函数，$\boldsymbol{X} = (X_1, \cdots, X_p)^{\mathrm{T}}$、$\boldsymbol{Y} = (Y_1, \cdots, Y_p)^{\mathrm{T}}$ 表示样本。此时矩

阵 $\boldsymbol{K}$ 的第 i 行，第 j 列元素 $\boldsymbol{K}_{ij}=k(\boldsymbol{X_i},\boldsymbol{Y_j})$。由单位特征向量的假定知 $\boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{\nu}=1$，推出 $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{a}=1$，因此得到条件

$$n\lambda\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}=1 \tag{2.4.2}$$

利用（2.4.1）和条件（2.4.2）可以求解出未知向量 $\boldsymbol{a}$，以及对应的特征值和特征向量。接下来，对于一个新的样本 $\boldsymbol{X}$，我们可以得到他的第一主成分是

$$\phi(\boldsymbol{X})\boldsymbol{\nu}_1=\sum_{i=1}^{n}\boldsymbol{a}_i k(\boldsymbol{X},\boldsymbol{X_i})$$

其中 $\boldsymbol{\nu}_1$ 是最大特征值对应的特征向量。

常用的核函数有：

1、线性核函数：

$$k(\boldsymbol{X},\boldsymbol{Y})=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}+c$$

其中 c 为参数。

2、多项式核函数：

$$k(\boldsymbol{X},\boldsymbol{Y})=(a\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}+c)^d$$

其中 a,b,c 为参数。

3、高斯核函数：

$$k(\boldsymbol{X},\boldsymbol{Y})=\exp(-\frac{\|\boldsymbol{X}-\boldsymbol{Y}\|^2}{2\sigma^2})$$

其中 σ 为参数，高斯核函数是径向基函数核的一个典型代表。

4、指数核函数：

$$k(\boldsymbol{X},\boldsymbol{Y})=\exp(-\frac{\|\boldsymbol{X}-\boldsymbol{Y}\|}{2\sigma^2})$$

其中 σ 为参数。指数核函数也是径向基函数核代表，与高斯核函数很像，只是将 L_2 范数变成 L_1 范数。

5、拉普拉斯核函数：

$$k(\boldsymbol{X},\boldsymbol{Y})=\exp(-\frac{\|\boldsymbol{X}-\boldsymbol{Y}\|}{\sigma})$$

拉普拉斯核函数完全等价于指数核，区别在于前者对参数的敏感性降低，也是一种径向基函数核。

值得注意的是，上述理论均是在 $\phi(\mathbf{X})$ 已经中心化的前提下完成的。在实际应用中，应首先将矩阵 $\phi(\mathbf{X})$ 中心化，即

$$\widetilde{\phi}(\mathbf{X})=\phi(\mathbf{X})-\mathbf{1}_n\cdot\phi(\mathbf{X})$$

其中 $\mathbf{1}_n=\frac{1}{n}\mathbf{1}_{n\times1}\mathbf{1}_{n\times1}^{\mathrm{T}}$，为 $n\times n$ 矩阵，其每个元素为 $\frac{1}{n}$。由上述可知，不需要显示计算 $\widetilde{\phi}(\mathbf{X})$，只需得到中心化后的核矩阵即可：

$$\widetilde{\boldsymbol{K}}=\widetilde{\phi}(\mathbf{X})\widetilde{\phi}(\mathbf{X})^{\mathrm{T}}=\boldsymbol{K}-\boldsymbol{K}\cdot\mathbf{1}_n-\mathbf{1}_n\cdot\boldsymbol{K}+\mathbf{1}_n\cdot\boldsymbol{K}\cdot\mathbf{1}_n$$

因此上面介绍 KPCA 使用的 $\phi(\mathbf{X})$ 和 $\boldsymbol{K}$ 本质上就是这里的 $\widetilde{\phi}(\mathbf{X})$ 和 $\widetilde{\boldsymbol{K}}$。

2.4.2 t-SNE 非线性降维算法

t 分布随机邻域嵌入（t-distributed stochastic neighbor embedding t-SNE）是一种针对高维数据的非线性降维算法，在 2008 年由 Laurens van der Maaten 和 Geoffrey Hinton [13] 提出。传统主成分分析是一种线性算法，不能解释特征之间的复杂多项式关系。而 t-SNE 是基于邻域图上随机游走的概率分布寻找数据内的结构，将数据点之间的相似度转化为条件概率，原始空间中数据点的相似度由正态分布表示，嵌入空间中数据点的相似度由 t 分布表示。通过原始空间和嵌入空间的联合概率分布的KL 散度（用于评估两个分布的相似度的指标）来评估嵌入效果的好坏。

1、SNE 算法

t-SNE 算法是从 SNE 改进而来，所以先介绍 SNE 。给定一组高维数据 $\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$，$\boldsymbol{X}_i = (X_{i1}, \cdots, X_{ip})^{\mathrm{T}}$。目标是将这组数据降维到二维，SNE 的基本思想是若两个数据在高维空间中是相似的，那么降维到二维空间时距离应当较近。

随机邻近嵌入（SNE）首先通过将数据点之间的高维欧几里得距离转换为相似性的条件概率来描述两个数据之间的相似性。

假设高维空间中的两个点 $\boldsymbol{X}_i, \boldsymbol{X}_j$，以点 $\boldsymbol{X}_i$ 为中心构建方差为 σ_i 的高斯分布。用 $P_{j|i}$ 表示 $\boldsymbol{X}_j$ 在 $\boldsymbol{X}_i$ 邻域的概率，若 $\boldsymbol{X}_j$ 与 $\boldsymbol{X}_i$ 相距很近，那么 $P_{j|i}$ 很大；反之，$P_{j|i}$ 很小，$P_{j|i}$ 可以表示为

$$P_{j|i} = \frac{\exp(-\left\|\boldsymbol{X}_j - \boldsymbol{X}_i\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\left\|\boldsymbol{X}_k - \boldsymbol{X}_i\right\|^2 / 2\sigma_i^2)}, \quad P_{i|i} = 0,$$

其中 $\left\|\boldsymbol{X}_i - \boldsymbol{X}_j\right\|$ 表示点 $\boldsymbol{X}_i, \boldsymbol{X}_j$ 的欧式距离，这里只关注不同数据点间的距离所以我们设置 $P_{i|i} = 0$。高斯核的带宽 σ_i 是条件概率中所涉及到的范围。有些特征点是稀疏的，有些比较紧密，因此带宽大小也是不同的。一般来说，数据密度高的区域带宽要小于数据密度低的区域。每个数据点的高斯核带宽的最优值可以通过简单的二进制搜索 [14] 等得到。

当把数据映射到低维空间后，高维数据点之间的相似性也应该在低维空间的数据点上体现出来。假设 $\boldsymbol{X}_i, \boldsymbol{X}_j$ 映射到低维空间后对应 $\boldsymbol{Y}_i, \boldsymbol{Y}_j$ ，那么 $\boldsymbol{Y}_i, \boldsymbol{Y}_j$ 邻域的条件概率为 $Q_{j|i}$:

$$Q_{j|i} = \frac{\exp\left(-\left\|\boldsymbol{Y}_j - \boldsymbol{Y}_i\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\boldsymbol{Y}_k - \boldsymbol{Y}_i\right\|^2\right)}$$

低维空间中的方差直接设置为 $\sigma_i = \dfrac{1}{\sqrt{2}}$ 。同样 $Q_{i|i} = 0$ 。

若条件概率 $Q_{j|i}$ 反映了高维数据点 $\boldsymbol{X}_i, \boldsymbol{X}_j$ 之间的关系，那么我们希望条件概率 $P_{j|i}$ 与 $Q_{j|i}$ 应该完全相等。若给定 $\boldsymbol{X}_i$ 与其他所有点之间的条件概率，则可构成一个条件概率分布 $\mathcal{P}_i$ 。同理在低维空间存在一个条件概率分布 $\mathcal{Q}_i$ 与之对应，那么我们希望条件概率分布 $\mathcal{Q}_i$ 与 $\mathcal{P}_i$ 完全一样。为了衡量两个分布之间的相似性，采用 Kullback-Leibler (KL) 散度最小化低维与高维下两个分布条件概率的差异，SNE 最终目标就是对所有数据点最小化 KL 距离，可以使用梯度下降算法最小化如下代价函数:

$$C = \sum_i \mathrm{KL}\left(P_i \| \mathcal{Q}_i\right) = \sum_i \sum_j P_{j|i} \log \frac{P_{j|i}}{Q_{j|i}}$$

但由于 KL 距离是一个非对称的度量。这意味着当 $P_{j|i}$ 较大，$Q_{j|i}$ 较小时，代价较高；而 $P_{j|i}$ 较小，$Q_{j|i}$ 较大时，代价较低。即高维空间中两个数据点距离较近时，若映射到低维空间后距离较远，那么将得到一个很高的惩罚，这符合我们的初衷。反之，高维空间中两个数据点距离较远时，若映射到低维空间距离较近，将得到一个很低的惩罚值，我们的初衷是这里也应得到一个较高的惩罚。即 SNE 的代价函数更关注局部结构，而忽视了全局结构。

2、*t*-SNE 算法

在 SNE 中，高维空间中条件概率 $P_{j|i}$ 不等于 $P_{i|j}$ ，低维空间中 $Q_{j|i}$ 不等于 $Q_{i|j}$ ，于是为简化计算提出对称 SNE，采用联合概率分布代替原始的条件概率，使得 $P_{ij} = P_{ji}$，$Q_{ij} = Q_{ji}$ 优化 $P_{i|j}$ 和 $Q_{i|j}$ 的 KL 散度的一种替换思路是，使用联合概率分布来替换条件概率分布，即 $\mathcal{P}$ 和 $\mathcal{Q}$ 分别是高维空间和低维空间里各个点的联合概率分布，此时目标函数为:

$$C = \mathrm{KL}(\mathcal{P} \| \mathcal{Q}) = \sum_i \sum_j P_{ij} \log \frac{P_{ij}}{Q_{ij}}$$

其中，在高维空间中定义 P_{ij} :

$$P_{ij} = \frac{P_{i|j} + P_{j|i}}{2}$$

为了使得高维度下的中高等距离在映射后距离更大，以减轻拥挤问题。在低维空间中使用更加偏重长尾分布的方式将距离转换为概率分布。因此，对于高维数据点 $\boldsymbol{X}_i$ 和 $\boldsymbol{X}_j$ 的低维对应点 $\boldsymbol{Y}_i$ 和 $\boldsymbol{Y}_j$，采用自由度为 1 的归一化的 t 核:

$$Q_{ij} = \frac{(1 + \|\boldsymbol{Y}_i - \boldsymbol{Y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\boldsymbol{Y}_k - \boldsymbol{Y}_l\|^2)^{-1}}, \quad Q_{ii} = 0$$

因此 t-SNE 是通过仿射变换将数据点映射到概率分布，将两个数据点之间的距离转换为以一个点为中心一定范围内另一个点出现的条件概率。由于在嵌入空间中的目标函数是非凸的，因此，可以用梯度下降最小化目标函数，此时

$$\frac{\partial C}{\partial \boldsymbol{Y}_i} = 4 \sum_{i \neq j} (P_{ij} - Q_{ij}) Q_{ij} Z (\boldsymbol{Y}_i - \boldsymbol{Y}_j)$$

其中 $Z = \sum_{k \neq l} (1 + \|\boldsymbol{Y}_k - \boldsymbol{Y}_l\|^2)^{-1}$。

因此，t-SNE 的主要优势在于通过 t 分布与正态分布的差异，解决了样本分布拥挤、边界不明显的问题，使得相似的样本能够聚集在一起，而差异大的样本能够有效地分开。但它同时也有计算复杂度高，目标函数非凸，容易得到局部最优解；对参数敏感、结果具有随机性等缺陷。

t-SNE 非线性降维算法在实际问题中可用于处理高维数据集，并应用于自然语言处理、图像处理、基因组数据和语音处理等。

2.5 主成分分析实践

2.5.1 R 语言实践

主成分分析

R 语言中可用于进行主成分分析的函数有：prcomp() 和 princomp()，以及 psych 包对数据进行主成分分析，其中，principal 函数进行主成分分析，fa.parallel 函数生成碎石图等。也可根据 PCA 的原理，利用自编函数的方法。下面将利用 R 中自带的范例数据集 iris 进行示范。

1、标准化数据：为了统一数据的量纲并对数据进行中心化，在主成分分析之前往往需要对原始数据进行标准化。

```
data<-iris
irisda<-as.matrix(scale(data[,1:4]))#对原数据进行z-score归一化
head(irisda)
```

2、计算相关系数（协方差）矩阵并求解特征值和相应的特征向量，提取各主成分方差，计算方差贡献率和累积贡献率，利用碎石函数 screeplot() 绘制碎石图。碎石图是可以用于确定主成分合适个数的有用的可视化工具。其将特征值从大到小排列，选取一个拐点对应的序号，序号后的特征值全部较小且大小差不多，因此此值可作为主成分的个数。

```
covm<-cor(irisda)#归一化后相关系数等于协方差
eign<-eigen(covm)
val <- eign$values#提取结果中的特征值，即各主成分的方差
(Standard_deviation <- sqrt(val))#换算成标准差(Standard deviation);
(Proportion_of_variance <- val/sum(val))
(Cumulative_Proportion <- cumsum(Proportion_of_variance))#计算方差贡献率和累积贡献率
par(mar=c(6,6,2,2))
plot(eign$values,type="b", cex=2, cex.lab=2, cex.axis=2, lty=2, lwd=2,
xlab = "主成分编号", ylab="特征值")#碎石图绘制
```

3、计算主成分得分

```
(U<-as.matrix(eign$vectors))#提取结果中的特征向量
PC <-irisda %*% U#进行矩阵乘法，获得PC score;
colnames(PC) <- c("PC1","PC2","PC3","PC4")
head(PC)
```

4、绘制主成分散点图

```
df<-data.frame(PC,iris$Species)#与iris数据集的第5列数据合并
head(df)
library(ggplot2)
#提取主成分的方差贡献率，生成坐标轴标题
xlab<-paste0("PC1(",round(Proportion_of_variance[1]*100,2),"%)")
ylab<-paste0("PC2(",round(Proportion_of_variance[2]*100,2),"%)")
p1<-ggplot(data = df,aes(x=PC1,y=PC2,color=iris.Species))+
stat_ellipse(aes(fill=iris.Species),
type ="norm", geom ="polygon",alpha=0.2,color=NA)+
geom_point()+labs(x=xlab,y=ylab,color="")+
guides(fill="none")#绘制散点图并添加置信椭圆
p1
```

本节只展示直观性强的碎石图2.1和主成分散点图2.2。

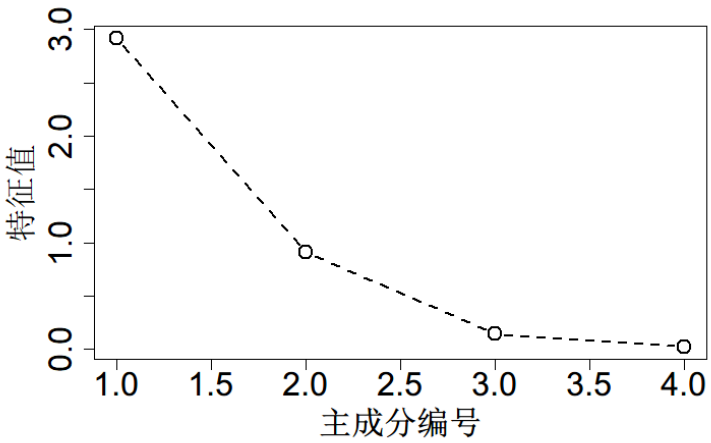


图 2.1 screeplot

在理解了主成分分析的具体步骤后，也可用 R 语言中常见的 PCA 函数验证以上分析过程的正确性。在实际操作中，可以根据具体情况，按需选择函数。

（1）prcomp() 函数

```
pca1 <- prcomp(data[,1:4], center = TRUE,scale. = TRUE)
summary(pca1)
```

（2）princomp() 函数：如果使用 princomp() 函数，需要先做归一化，princomp() 函数并无数据标准化相关的参数，且默认使用 covariance matrix，得到的结果与使

图 2.2 PCA

用相关性矩阵有细微差异（如下）。原因是：根据相关系数公式可知，归一化后的相关性系数近乎等于协方差。

```
pca2 <- princomp(irisda,cor = T)
summary(pca2)
pca3 <- princomp(irisda)
summary(pca3)
```

通过比较分析得到 Standard deviation、Proportion of variance、Cumulative Proportion 发现，3 种主成分分析的方法得到的结果完全一致。

（3）principal() 函数：

```
library(psych)
PC <- principal(irisda , nfactors=2, rotate ="none")
pc <- data.frame(PC$scores)
p  <- ggplot(pc, aes(x=PC1, y=PC2, color="iris$Species" )) +
      geom_point(size=4,alpha=0.5)+
      theme(axis.text= element_text(size=20))+
      theme(panel.grid.major =element_blank(),
            panel.grid.minor = element_line(color="steelblue"),
            panel.background = element_blank(),
            axis.line = element_line(colour = "black"))+
      stat_ellipse(lwd=1,level = 0.8)
p
```

其中 nfactors: 指定主成分个数，rotate: 指定模型旋转的方法，默认为最大方差法。最终可得主成分散点图2.3。

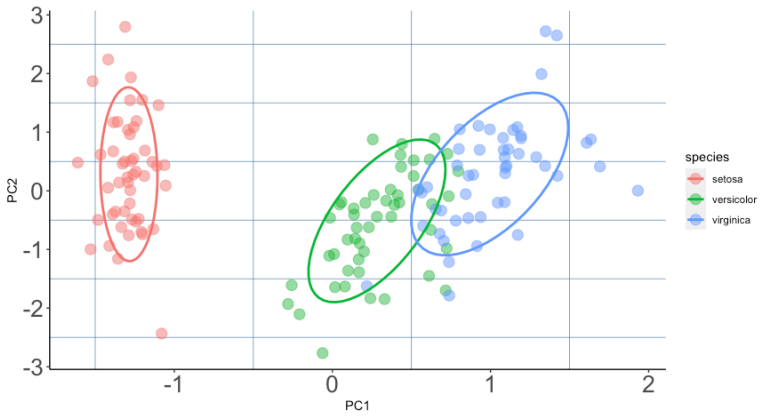


图 2.3 PCA-principal()

非线性主成分分析

（1）核主成分分析（KPCA）：在 R 中可以通过 kernlab 包中函数 kpca() 实现基于核分析的主成分分析。以 R 中自带的范例数据集 iris 进行示范。

```
data(iris)
library(kernlab)
test <- sample(1:150,20)
kpc  <- kpca(~.,data=iris[-test,-5],kernel="rbfdot",kpar=list(sigma=0.2),features=2)
pcv(kpc)
#绘制数据图
plot(rotated(kpc),col=as.integer(iris[-test,5]),
xlab="1st-Principal-Component",ylab="2nd-Principal-Component")
emb <- predict(kpc,iris[test,-5])#嵌入余下的点
points(emb,col=as.integer(iris[test,5]))
```

其可视化图为2.4。

（2）t-SNE 非线性降维算法：R 中可以通过 Rtsne 包可以实现 t-SNE 分析，所使用的函数为 Rtsne($\mathbf{X}$,...)，其中 X 为数据矩阵。以 R 中自带的范例数据集 iris 进行示范。

```
library(Rtsne)
data(iris)
iris.uni <- unique(iris)#去除重复数据
data <- iris.uni[,1:4]#获取性状信息
species <- iris.uni[,5]#获取品种信息
set.seed(321)
tsne_out <- Rtsne(data, dims = 2, pca = T, max_iter = 1000,
theta = 0.4, perplexity = 20, verbose = F)
library(ggplot2)
tsne_result <- as.data.frame(tsne_out$Y)
colnames(tsne_result) <- c("tSNE1","tSNE2")
ggplot(tsne_result,aes(tSNE1,tSNE2,color=species)) + geom_point()
```

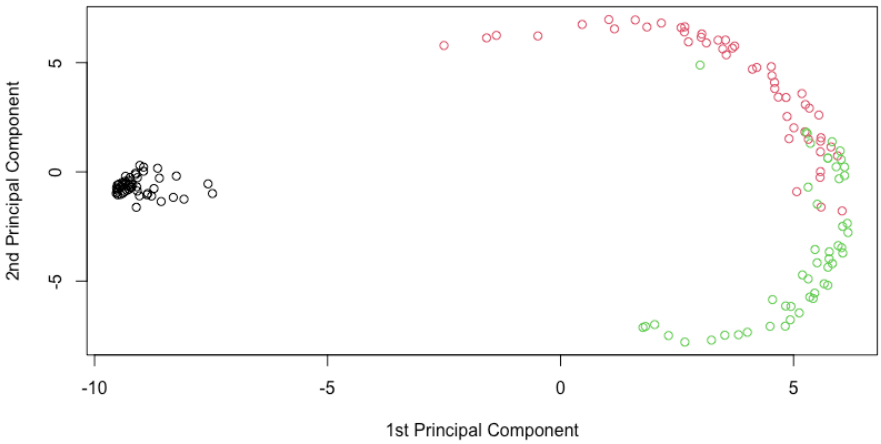


图 2.4 KPCA

其可视化图为2.5。

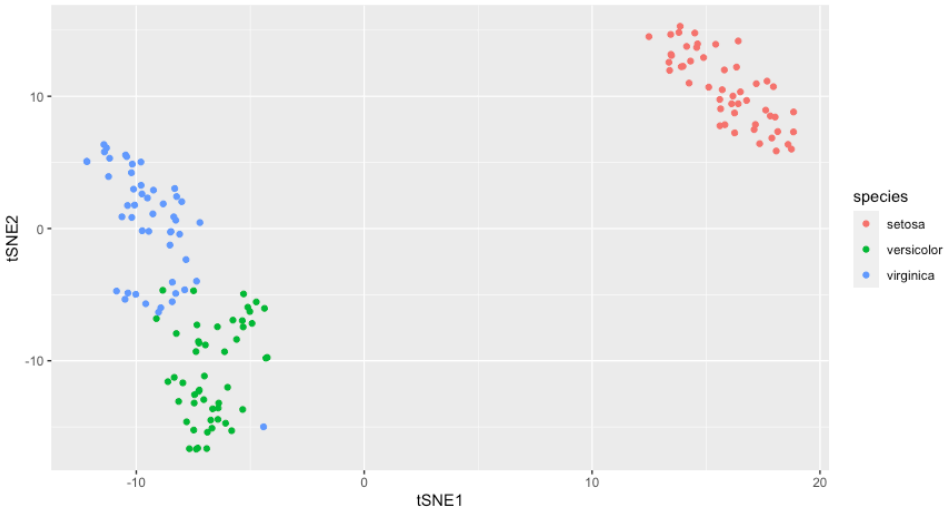


图 2.5 *t*-SNE

2.5.2 Python 语言实践

主成分分析

Python 语言中的 sklearn 库可实现主成分分析方法，本节将采用 PCA 对莺尾花数据集的特征进行分析，首先加载相应模块：

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
import numpy as np
```

再导入鸢尾花数据集，利用 PCA 类进行分析

```
iris = load_iris()
X = iris.data
Pca_Model = PCA()#导入PCA方法
Pca_Model = Pca_Model.fit(X)#建立PCA模型
Pca_Result = Pca_Model.transform(X)
print(Pca_Result)
Ratio = Pca_Model.explained_variance_ratio_
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.plot([1, 2, 3, 4], np.cumsum(Ratio))
plt.xticks([1, 2, 3, 4])
plt.title('累积贡献率')
plt.show()#基于各成分的贡献率，绘制累积贡献率折线图
```

累积贡献率折线图如图 2.6 所示，明显可以看出前两个成分的累积贡献率已经超过 97%，可选为主成分。

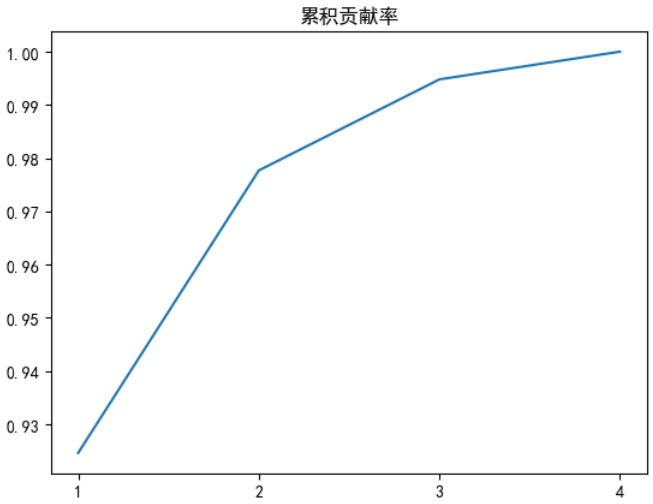


图 2.6 各成分累积贡献率

总结

主成分分析通过对原始高维特征进行映射变换得到少数几个主成分，得到的主成分能够最大程度地表达原始数据，是一种处理高维数据的有效降维方法。本章主要介绍了总体主成分、样本主成分以及非线性主成分分析的算法理论，并通过实验

案例阐述算法在实际问题中的应用。除本章介绍的几种主成分分析方法之外，更多方法可参考：稀疏主成分分析（SPCA）[15]，该方法不仅可以达到降维的效果，还能够增强算法可解释性；张量主成分分析（TPCA）[16] [17]，其将张量与数据处理相结合，在高维信息压缩等领域得到广泛应用；鲁棒/稳健主成分分析（RPCA）[18]可提高算法对异常值、噪声的鲁棒性。

主成分分析作为非常重要的降维机器学习方法被广泛认可，它是从最大投影方差的角度介绍了降维的过程。另外基于最小投影距离以及与之相应的奇异值分解过程也是非常重要的降维机器学习方法；现代非线性降维分析机器学习方法，例如流形学习和变分自编码器等，由于涉及到神经网络及贝叶斯方法，相关内容建议同学自行学习。

2.6 习题

1、设总体 $\boldsymbol{X}=(X_1,X_2)^{\mathrm{T}}$ 的协方差矩阵为

$$\boldsymbol{\Sigma}=\begin{bmatrix}5 & 2\\ 2 & 2\end{bmatrix}$$

求 $\boldsymbol{X}$ 的主成分 Y_1,Y_2，并计算第一主成分 Y_1 的贡献率。

2、总体 $\boldsymbol{X}=(X_1,\cdots,X_p)^{\mathrm{T}}$ 的协方差矩阵为

$$\boldsymbol{\Sigma}=\mathrm{diag}(\sigma_{11},\sigma_{22},\cdots,\sigma_{pp})$$

其中 $\sigma_{11}\geq\sigma_{22}\geq\cdots\geq\sigma_{pp}$，试求 $\boldsymbol{X}$ 的主成分及其具有的特征值。

3、试证明多元正态变量的主成分仍为正态变量且相互独立。

4、设总体 $\boldsymbol{X}=(X_1,X_2,X_3)^{\mathrm{T}}$ 的相关系数矩阵为

$$\boldsymbol{\Sigma}=\begin{bmatrix}1 & \rho & \rho\\ \rho & 1 & \rho\\ \rho & \rho & 1\end{bmatrix},(-1<\rho<1)$$

（1）求 $\boldsymbol{X}$ 的标准化变量的主成分及其主成分的贡献率和累积贡献率

（2）将上述结果推广到 p 维情形。

5. 运用主成分分析对我国 2019 年分地区居民人均消费支出倾向进行分析。（数据来源：http://www.stats.gov.cn/tjsj/ndsj/2020/indexch.htm。数据表名称：6-20 分地区居民人均消费支出（2019 年））。

第二部分

监督学习

第三章　回归分析

监督学习（Supervised Learning）也是一种机器学习的重要方法，算法学习的目标是预测或分类。在监督学习中，算法接收输入数据和相应的输出标签，通过学习这些它们之间的映射关系，对新的未标记数据进行预测。具有高准确性、可解释性强和增强决策等优势。广泛应用于分类、回归、目标检测和推荐系统中。

无监督学习算法接收没有标签的训练数据，模型在学习过程中不知道样本的正确标签，从数据中找到模式或结构，而无需指导。而监督学习是算法接收有标签的训练数据，每个训练样本都与一个明确的目标输出或标签相关联，模型的任务是学习如何从输入数据映射到这些标签。

本章及第四章、第五章、第六章将分别介绍四种主要的监督学习方法：回归分析（Regression Analysis）、支持向量机（Support Vector Machine, SVM）、决策树（Decision Tree）和保形预测（Conformal Prediction）。

3.1　简介

回归（Regression）这一概念最早由英国生物统计学家高尔顿（F.Galton）和皮尔逊（K.Pearson）在研究父母和子女的身高遗传特性时提出。回归分析（Regression Analysis）是处理多个变量间相关关系的一种数学方法，是机器学习中的监督学习方法。

回归模型是一种统计预测模型，需要预测的变量叫做因变量（或响应变量（response variable）），用来解释因变量变化的变量叫做自变量（或协变量（covariate）、特征（feature））。回归模型通过若干个自变量来预测响应变量。

从响应变量个数来看，回归模型中可以有一个响应变量，也可以有多个响应变量。从响应变量取值来看，回归模型中的响应变量可以是连续变量，也可以是离散变量。

一般的回归模型适用于单个响应变量、取值为连续型且一般服从正态分布的情形。广义线性模型（generalized linear models, GLM）适用于单个响应变量、取值是离散型且分布服从指数分布族的情形。多元响应变量协方差广义线性模型（multivariate covariance generalized linear model, McGLM）是广义线性模型的扩展，适用于服从非正态分布且不独立的多个响应变量的情形。

本章主要介绍单响应变量的线性回归模型、单响应变量的广义线性模型（GLM）和多元响应变量协方差广义线性模型（McGLM）。

3.2 单响应变量的线性回归模型

本节主要介绍单响应变量的线性回归模型的原理、自变量的选择、多重共线性和岭回归。

3.2.1 线性回归模型的原理

下面介绍线性回归模型的一般形式、基本假设、未知参数的估计、显著性检验、预测等。

一、线性回归模型的一般形式

定义 3.1 p 个自变量 $X_1, X_2, \cdots, X_p$ 对单个响应变量 Y 的线性回归模型为

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \tag{3.2.1}$$

式中，$\beta_0, \beta_1, \beta_2, \cdots, \beta_p$ 是 $p+1$ 个未知参数。β_0 称为回归常数，$\beta_1, \beta_2, \cdots, \beta_p$ 为回归系数。当 $p=1$ 时，式 (3.2.1) 称为一元线性回归模型；当 $p \geq 2$ 时，式 (3.2.1) 称为 p 元线性回归模型。式中 ε 是随机误差，常假定 $\mathrm{E}(\varepsilon) = 0, \mathrm{var}(\varepsilon) = \sigma^2$。

若对单个响应变量的线性回归模型来说，观测到样本数据为 $\{(X_{i1}, X_{i2}, \cdots, X_{ip}; Y_i), i = 1, \cdots, n\}$，则基于 n 个样本的线性回归模型可表示为

$$\begin{cases} Y_1 = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12} + \cdots + \beta_p X_{1p} + \varepsilon_1 \\ Y_2 = \beta_0 + \beta_1 X_{21} + \beta_2 X_{22} + \cdots + \beta_p X_{2p} + \varepsilon_2 \\ \vdots \\ Y_n = \beta_0 + \beta_1 X_{n1} + \beta_2 X_{n2} + \cdots + \beta_p X_{np} + \varepsilon_n \end{cases}$$

写成矩阵形式为

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{3.2.2}$$

式 (3.2.2) 中 $\mathbf{X}$ 为设计矩阵，

$$\boldsymbol{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1p} \\ 1 & X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

二、线性回归模型的基本假设

为了方便地进行模型的参数估计及检验，对线性回归模型进行如下基本假设：

1. 解释变量 $X_1, X_2, \cdots, X_p$ 是确定性变量，而不是随机变量。且 $\mathrm{rank}(\mathbf{X}) = p+1 < n$，即设计矩阵 $\mathbf{X}$ 是列满秩矩阵，其列之间不线性相关。

2. 随机误差项满足高斯-马尔柯夫条件，即在给定自变量取值的情况下，随机误差项满足零均值、同方差及序列不相关。

$$\begin{aligned} &\mathrm{E}(\varepsilon_i) = 0, \quad i = 1, 2, \cdots, n \\ &\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = \begin{cases} \sigma^2, & i = j \\ 0, & i \neq j \end{cases} \quad i, j = 1, 2, \cdots, n \end{aligned} \tag{3.2.3}$$

3. 假定误差满足正态分布假定，即 $\varepsilon_i \sim N(0, \sigma^2), \quad i = 1, 2, \cdots, n$，且 $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n$ 相互独立。矩阵形式为

$$\boldsymbol{\varepsilon} \sim N_n(\mathbf{0}, \sigma^2 \mathbf{I}_n) \tag{3.2.4}$$

其中 $\mathbf{I}_n$ 是 $n \times n$ 单位阵。

其中条件 1 和 2 是为了最小二乘估计，条件 3 是为了假设检验过程中研究估计量的分布，当然条件 3 也可以用于参数的极大似然估计。

三、参数的估计

下面利用最小二乘法求线性回归模型 $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ 中回归参数 $\boldsymbol{\beta}$ 的最小二乘估计量。

最小二乘法(Ordinary Least Square Method,OLS)就是寻找未知参数 $\beta_0, \beta_1, \beta_2, \cdots, \beta_p$ 的估计值 $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2, \cdots, \widehat{\beta}_p$，使得响应变量的离差平方和

$$Q(\beta_0, \beta_1, \beta_2, \cdots, \beta_p) = \sum_{i=1}^{n} [Y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip})]^2$$

达到最小，即满足

$$\begin{aligned} &Q(\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2, \cdots, \widehat{\beta}_p) = \sum_{i=1}^{n} [Y_i - (\widehat{\beta}_0 + \widehat{\beta}_1 X_{i1} + \widehat{\beta}_2 X_{i2} + \cdots + \widehat{\beta}_p X_{ip})]^2 \\ &= \min_{\beta_0, \beta_1, \beta_2, \cdots, \beta_p} \sum_{i=1}^{n} [Y_i - (\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip})]^2 \end{aligned}$$

根据微积分求极值的原理，只需求 $Q(\beta_0, \beta_1, \beta_2, \cdots, \beta_p)$ 的最小值点。将 Q 关于 $\beta_0, \beta_1, \beta_2, \cdots, \beta_p$ 分别求偏导数，并令其为零，得到正规方程组

$$\begin{cases} \left.\dfrac{\partial Q}{\partial \beta_0}\right|_{\beta_0=\widehat{\beta}_0} = -2\sum\limits_{i=1}^{n}[Y_i - (\widehat{\beta}_0 + \widehat{\beta}_1 X_{i1} + \widehat{\beta}_2 X_{i2} + \cdots + \widehat{\beta}_p X_{ip})] = 0 \\ \left.\dfrac{\partial Q}{\partial \beta_1}\right|_{\beta_1=\widehat{\beta}_1} = -2\sum\limits_{i=1}^{n}[Y_i - (\widehat{\beta}_0 + \widehat{\beta}_1 X_{i1} + \widehat{\beta}_2 X_{i2} + \cdots + \widehat{\beta}_p X_{ip})]X_{i1} = 0 \\ \cdots \\ \left.\dfrac{\partial Q}{\partial \beta_p}\right|_{\beta_p=\widehat{\beta}_p} = -2\sum\limits_{i=1}^{n}[Y_i - (\widehat{\beta}_0 + \widehat{\beta}_1 X_{i1} + \widehat{\beta}_2 X_{i2} + \cdots + \widehat{\beta}_p X_{ip})]X_{ip} = 0 \end{cases}$$

整理得矩阵形式的正规方程组

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})\widehat{\boldsymbol{\beta}} = \mathbf{X}^{\mathrm{T}}\boldsymbol{Y}$$

由 $\mathbf{X}$ 的列满秩性可知, $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 为满秩对称阵, 从而回归参数 $\boldsymbol{\beta}$ 的最小二乘估计为

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{Y} \tag{3.2.5}$$

根据唐年胜，李会琼 [31]，可以证明，在高斯-马尔柯夫假定 (3.2.3) 下最小二乘估计 $\widehat{\boldsymbol{\beta}}$ 是 $\boldsymbol{\beta}$ 的最小方差线性无偏估计。且在假定 (3.2.4) 下，我们可以推导出

$$\widehat{\boldsymbol{\beta}} \sim N_{p+1}(\boldsymbol{\beta}, \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}) \tag{3.2.6}$$

证明可详见唐年胜，李会琼 [31]。

定义 3.2 对于线性回归模型 $Y = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon$，称

$$\widehat{Y} = \widehat{\beta}_0 + \widehat{\beta}_1 X_1 + \widehat{\beta}_2 X_2 + \cdots + \widehat{\beta}_p X_p = \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}$$

为样本回归方程，称 $\widehat{Y}$ 为响应变量的拟合值，称 $e = Y - \widehat{Y}$ 为回归残差。

定义 3.3 最小二乘法没有给出未知参数 σ^2 的估计。利用最大似然比原理可得 $\boldsymbol{\beta}$ 极大似然估计仍为 $\widehat{\boldsymbol{\beta}}$，同时给出 σ^2 的极大似然估计为

$$\frac{(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})}{n} = \frac{1}{n}\sum_{i=1}^{n} e_i^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2 \tag{3.2.7}$$

因为上述估计有偏，通常取

$$\widehat{\sigma}^2 = \frac{(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})}{n-p-1} = \frac{1}{n-p-1}\sum_{i=1}^{n} e_i^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2 \tag{3.2.8}$$

作为 σ^2 的无偏估计，称 $\widehat{\sigma}$ 为回归标准差。

四、显著性检验

下面介绍回归方程显著性的 F 检验、回归参数显著性的 t 检验以及衡量回归方程拟合程度的拟合优度检验。

1、回归方程的显著性检验：F 检验

（1）平方和分解式

下面基于方差分析的思想，从数据出发来研究因变量数据变异的原因。

数据 $Y_1, Y_2, \cdots, Y_n$ 的波动大小可用总离差平方和 $\mathrm{SST} = \sum_{i=1}^{n} (Y_i - \bar{Y})^2$ 来度量，其中 $\bar{Y} = \sum_{i=1}^{n} Y_i/n$，它反映响应变量 Y 的波动程度或不确定性。

称 $\mathrm{SSR} = \sum_{i=1}^{n} (\widehat{Y}_i - \bar{Y})^2$ 为回归平方和，它是由回归方程确定的，是由自变量 $\boldsymbol{Y}$ 的波动引起的，反映了自变量解释因变量波动的贡献。

称 $\mathrm{SSE} = \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2$ 为残差平方和，它是不能由自变量解释的波动，是由自变量之外的因素引起的。

易证 $\mathrm{SST} = \mathrm{SSR} + \mathrm{SSE}$，称此式为平方和分解式。

平方和分解式表明，总离差平方和 SST 中能够由自变量解释的部分为 SSR，不能由自变量解释的部分为 SSE。因此，回归平方和越大，回归的效果就越好。

（2）F 检验统计量

在正态分布假设 (3.2.4) 下，需要检验自变量 $X_1, X_2, \cdots, X_p$ 从整体上对因变量 Y 是否有显著的影响，为此提出：

原假设

$$\mathrm{H}_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

备择假设

$$\mathrm{H}_1 : \beta_1, \beta_2, \cdots, \beta_p \text{至少有一个不为零}$$

在原假设 H_0 下，构造检验统计量

$$F = \frac{\mathrm{SSR}/p}{\mathrm{SSE}/(n-p-1)} \sim F(p, n-p-1) \tag{3.2.9}$$

给定显著性水平 α $(0 < \alpha < 1)$，右侧 F 检验的临界值为 $F_\alpha(p, n-p-1)$。当检验统计量 F 的观测值落在拒绝域，即 $F > F_\alpha(p, n-p-1)$ 时，拒绝原假设，说明回归方程显著，$\boldsymbol{X}$ 与 Y 有显著的线性关系；否则，只能接受原假设，认为回归方程不显著。

也可以利用 P 值法进行判断，其中 $P = \Pr(Z > F)$，$Z \sim F(p, n-p-1)$，当 $P < \alpha$ 时，拒绝原假设；否则，只能接受原假设。

通常利用方差分析表3.1进行 F 检验。

表 3.1 方差分析表

方差来源	自由度	平方和	均方	F 值	$\Pr(Z>F)$
回归	p	SSR	SSR/p	$\dfrac{\mathrm{SSR}/p}{\mathrm{SSE}/(n-p-1)}$	
残差	$n-p-1$	SSE	$\mathrm{SSE}/(n-p-1)$		
总和	$n-1$	SST			

2、回归系数的显著性检验：t 检验

在多元线性回归分析中，回归方程显著并不意味着每个自变量对 Y 的影响都显著，所以需要对每个自变量进行显著性检验。为此，提出

原假设

$$\mathrm{H}_{0j}:\beta_j=0,\quad j=1,\cdots,p$$

备择假设

$$\mathrm{H}_{1j}:\beta_j\neq 0,\quad j=1,\cdots,p$$

因 $\widehat{\boldsymbol{\beta}}=(\widehat{\beta}_0,\widehat{\beta}_1,\widehat{\beta}_2,\cdots,\widehat{\beta}_p)^{\mathrm{T}}\sim N_{p+1}(\boldsymbol{\beta},\sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1})$，若记

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}=(c_{ij})\overset{\Delta}{=}C,\quad i,j=0,1,2,\cdots,p \tag{3.2.10}$$

则

$$\widehat{\beta}_j\sim N(\beta_j,c_{jj}\sigma^2),\quad j=1,2,\cdots,p$$

构造 t 统计量

$$t_j=\frac{\widehat{\beta}_j}{\sqrt{c_{jj}}\widehat{\sigma}}\sim t(n-p-1) \tag{3.2.11}$$

上式中，$\widehat{\sigma}^2=\dfrac{(\boldsymbol{Y}-\mathbf{X}\widehat{\boldsymbol{\beta}})^{\mathrm{T}}(\boldsymbol{Y}-\mathbf{X}\widehat{\boldsymbol{\beta}})}{n-p-1}$。

给定显著性水平 α $(0<\alpha<1)$，双侧 t 检验的临界值为 $t_{\alpha/2}(n-p-1)$。当检验统计量的观测值落在拒绝域，即 $|t_j|>t_{\alpha/2}(n-p-1)$ 时，拒绝原假设，认为 β_j 显著不为 0，X_j 对 Y 影响显著；否则，只能接受原假设。

还可以利用 P 值法进行判断，其中 $P=\Pr(|Z|>|t_j|)$, $Z\sim t(n-p-1)$。当 $P<\alpha$ 时，拒绝原假设；否则，只能接受原假设。

3、拟合优度检验

拟合优度检验用于检验回归方程对样本观测值的拟合程度。

定义 3.4 样本决定系数为

$$\mathrm{R}^2=\frac{\mathrm{SSR}}{\mathrm{SST}}=1-\frac{\mathrm{SSE}}{\mathrm{SST}}$$

样本决定系数 R^2 是一个回归直线与样本观测值拟合优度的相对指标，反映了因变量的波动能被自变量解释的比例。其取值在 0 与 1 之间。R^2 越接近 1，拟合优度越好。

在应用过程中，残差平方和往往随着解释变量个数的增加而减少。如果在模型中增加一个解释变量，R^2 往往增大。而由增加解释变量个数引起的 R^2 的增大，往往与拟合好坏无关，因此样本决定系数需要调整。在样本容量一定的情况下，增加解释变量必定使得自由度减少。故可将残差平方和与总离差平方和分别除以各自的自由度，以剔除变量个数对拟合优度的影响。

定义 3.5 记 $\bar{\mathrm{R}}^2$ 为修正的决定系数，则

$$\bar{\mathrm{R}}^2 = 1 - \frac{\mathrm{SSE}/(n-p-1)}{\mathrm{SST}/(n-1)} \tag{3.2.12}$$

$\bar{\mathrm{R}}^2$ 越大，对应的回归方程拟合程度越好。

五、预测

若基于观测数据 $\{(X_{i1}, X_{i2}, \cdots, X_{ip}; Y_i), i = 1, \cdots, n\}$ 建立的回归方程

$$\widehat{Y} = \widehat{\beta}_0 + \widehat{\beta}_1 X_1 + \cdots + \widehat{\beta}_p X_p = \boldsymbol{X}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}$$

通过了回归方程的显著性检验和回归系数的显著性检验，则对给定的新样本观测点 $\boldsymbol{X}_0 = (1, X_{01}, X_{02}, \cdots, X_{0p})^{\mathrm{T}}$，可以用 $\widehat{Y}_0 = \boldsymbol{X}_0{}^{\mathrm{T}}\widehat{\boldsymbol{\beta}} = \widehat{\beta}_0 + \widehat{\beta}_1 X_{01} + \cdots + \widehat{\beta}_p X_{0p}$ 作为 $Y_0 = \beta_0 + \beta_1 X_{01} + \cdots + \beta_p X_{0p} + \varepsilon_0$ 的点预测。

因 $\widehat{\boldsymbol{\beta}}$ 与 ε_0 独立，$\widehat{\boldsymbol{\beta}}$ 与 Y_0 独立，从而$\widehat{Y}_0$ 与 Y_0 独立。下面证明可详见唐年胜，李会琼 [31]，

$$\widehat{Y}_0 - Y_0 \sim N(0, \sigma^2(1 + \boldsymbol{X}_0{}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\boldsymbol{X}_0)$$

从而

$$\frac{\widehat{Y}_0 - Y_0}{\widehat{\sigma}\sqrt{1 + \boldsymbol{X}_0{}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\boldsymbol{X}_0}} \sim t(n-p-1)$$

由此得，Y_0 的置信度 $1-\alpha$ 的置信区间为

$$\left(\widehat{Y}_0 - t_{\alpha/2}(n-p-1)\widehat{\sigma}\sqrt{1 + \boldsymbol{X}_0{}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\boldsymbol{X}_0}, \widehat{Y}_0 + t_{\alpha/2}(n-p-1)\widehat{\sigma}\sqrt{1 + \boldsymbol{X}_0{}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\boldsymbol{X}_0}\right)$$

3.2.2 多重共线性

一般情况下，当回归方程的自变量之间存在很强的线性关系，回归方程的检验高度显著时，有些回归系数却不能通过显著性检验，甚至出现有的回归系数所带符号与实际经济意义不符，此时可认为变量间存在多重共线性。

我们已经知道多元线性回归模型 (3.2.2) 的未知参数 $\boldsymbol{\beta}$ 的最小二乘估计为

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

这就要求 $\mathbf{X}$ 列满秩，即 $\mathrm{rank}(\mathbf{X}) = p+1$. 也就是要求 $\mathbf{X}$ 的列向量之间线性无关。然而，往往 $\mathrm{rank}(\mathbf{X}) < p+1$，此时 $|\mathbf{X}^{\mathrm{T}}\mathbf{X}| = 0$，从而 $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ 不存在，也就无法得到参数的估计量。

在实际问题的研究中，经常见到近似多重共线性的情形。此时，$\mathrm{rank}(\mathbf{X}) = p+1$，但 $|\mathbf{X}^{\mathrm{T}}\mathbf{X}| \approx 0$，$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ 的对角线元素很大，从而 $\mathrm{var}(\widehat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ 的对角线元素很大，使得参数的估计精度很低。

多重共线性可以用方差膨胀因子进行判断。

定义 3.6 称式 (3.2.10) 中矩阵 $C = (c_{ij}) = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ 中对角线元素 c_{jj} 为自变量 X_j 的方差膨胀因子（variance inflation factor，VIF），记作 VIF_j。

易证

$$c_{jj} = \frac{1}{1-\mathrm{R}_j^2}$$

其中，R_j^2 是以 X_j 为因变量对其余 $p-1$ 个自变量进行线性回归得到的样本决定系数，它度量了自变量 X_j 与其余 $p-1$ 个自变量的线性相关程度。这种相关程度越强，说明自变量之间的多重共线性越严重。此时，R_j^2 越接近 1，VIF_j 越大。VIF_j 的大小反映了自变量之间存在多重共线性的强弱。当 $\mathrm{VIF}_j \geq 10$ 时，说明自变量 X_j 与其余自变量之间存在严重的多重共线性。

3.2.3 岭回归

针对自变量之间出现多重共线性时，普通最小二乘法效果明显变差的问题，霍尔（Hoerl）[33] 于 1962 年提出一种改进最小二乘估计的方法。

当变量 X_j 之间相互独立时，使用最小二乘法估计的参数 $\widehat{\boldsymbol{\beta}}$ 是真实值的无偏估计，即满足 $E(\widehat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$，并且在所有的无偏估计中具有最小的方差。从求解公式上看，若要保证该回归系数有解，必须确保 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 矩阵是满秩的，即 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 可逆，但在实际中，若是 $\boldsymbol{X}_j$ 间存在高度多重共线性，或者自变量个数大于等于观测个数，不论哪种情况，最终算出来的行列式都等于 0 或者是近似为 0，此时 $\widehat{\boldsymbol{\beta}}$ 的值表现不稳定，即此时 $\widehat{\boldsymbol{\beta}}$ 的方差很大（虽然其在所有无偏估计类中最小，但其本身仍很大），导致其均方误差 MSE（$\mathrm{MSE}(\widehat{\boldsymbol{\beta}}) = \mathrm{var}(\widehat{\boldsymbol{\beta}}) + \left[\mathrm{bias}(\widehat{\boldsymbol{\beta}})\right]^2$）很大。

当自变量之间存在多重共线性，即 $|\mathbf{X}^{\mathrm{T}}\mathbf{X}| \approx 0$ 时，如果给 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 加上一个正的常数矩阵 $\lambda\mathbf{I}$ $(\lambda > 0)$，那么 $\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I}$ 接近奇异的程度会比 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 接近奇异的程度小得多，从而使得行列式不再为零，可以求逆。即得到岭估计：

$$\widehat{\boldsymbol{\beta}}(\lambda) \equiv (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{Y}$$

容易看出，岭估计使用了单位矩阵乘以常数 λ，$\lambda\mathbf{I}$ 矩阵在 0 构成的平面上有一条 1 组成的“岭”，这便是岭回归名称的由来。

定义 3.7 称 $\widehat{\boldsymbol{\beta}}(\lambda)$ 为参数 $\boldsymbol{\beta}$ 的岭估计。其中，λ 称为岭参数。由参数 $\boldsymbol{\beta}$ 的岭估计所建立的回归方程称为岭回归方程。

易证，$\widehat{\boldsymbol{\beta}}(\lambda)$ 是回归参数 $\boldsymbol{\beta}$ 的有偏估计。且 $||\widehat{\boldsymbol{\beta}}(\lambda)|| < ||\widehat{\boldsymbol{\beta}}||$，这表明 $\widehat{\boldsymbol{\beta}}(\lambda)$ 可看成由 $\widehat{\boldsymbol{\beta}}$ 进行某种向原点的压缩。在均方误差意义下，$\widehat{\boldsymbol{\beta}}(\lambda)$ 优于最小二乘估计 $\widehat{\boldsymbol{\beta}}$。当 $\lambda = 0$ 时，岭估计 $\widehat{\boldsymbol{\beta}}(0)$ 即为普通最小二乘估计 $\widehat{\boldsymbol{\beta}}$。证明可详见唐年胜，李会琼 [31]。

因为岭参数 λ 不是唯一确定的，故岭回归估计 $\widehat{\boldsymbol{\beta}}(\lambda)$ 是回归参数 $\boldsymbol{\beta}$ 的一个估计族。当岭参数 λ 在 $(0, \infty)$ 内变化时，$\widehat{\beta}_j(\lambda)$ 是 λ 的函数。在平面坐标系中，把函数 $\widehat{\beta}_j(\lambda)$ 随 $\lambda > 0$ 变化所描绘出来的曲线，称为岭迹。

选择岭参数 λ 的值一般满足以下原则：

1、各回归系数的岭估计基本稳定；

2、用最小二乘估计所得符号不合理的回归系数，其岭估计的符号变得合理；

3、回归系数没有不合乎经济意义的绝对值；

4、残差平方和增加不太多。

在实际应用中，可以利用 $\widehat{\boldsymbol{\beta}}(\lambda)$ 的每一分量 $\widehat{\beta}_j(\lambda)$ 在同一图形上的岭迹的变化形状来确定适当的 λ。但是，岭迹法存在一定的主观性。此外，还可以用广义交叉验证（generalized cross validation，GCV）来确定岭参数。通常选取使得 GCV 最小的 λ 值。具体可详见 [34]。

3.3 广义线性模型

在本章的前面章节介绍了多元线性模型以及其假设，一般的线性模型主要适用于响应变量是连续型随机变量，并且其一般服从正态分布。但是我们会发现在实际应用中有些响应变量是离散的，而且并不服从正态分布。针对这种情况，Nelder 和 Wedderburn 将传统线性模型进行推广到广义线性模型（generalized linear models）。广义线性模型将因变量的分布推广到指数分布族，从连续型变量拓展到离散型变量，通过连接函数将服从指数族分布的响应变量的期望与自变量的线性函数连接起来，下面开始介绍广义线性模型。在给出广义线性模型的定义前先介绍指数族分布与连接函数。

3.3.1 指数族分布

指数族分布是以指数分布表示的一族分布，很多常见的分布如正态分布、Poisson 分布、二项分布等属于指数族分布。下面给出指数族分布的定义。

定义 3.8 若随机变量 Y 的概率质量（离散型）或概率密度（连续型）具有如下形式：

$$f(y;\theta,\phi)=\exp\left(\frac{y\theta-b(\theta)}{a(\phi)}+c(y,\phi)\right) \tag{3.3.1}$$

则称 y 服从指数族分布。其中 $a(\cdot)$，$b(\cdot)$，$c(\cdot,\cdot)$ 为已知连续函数，θ 和 ϕ 为未知参数。通常，θ 与 $\mathrm{E}(Y)$ 有关，是我们所感兴趣的参数，而 ϕ 与 $\mathrm{var}(Y)$ 有关，称为多余参数 （nuisance parameter）。

下面以正态分布、泊松分布和二项分布为例说明这两种分布的概率密度或概率质量具有 (3.3.1) 式的形式，并求出对应的 $a(\cdot),b(\cdot),c(\cdot,\cdot)$ 以及未知参数。

例 3.1 设 $Y\sim N\left(\mu,\sigma^2\right)$，则 Y 的概率密度可表为；

$$\begin{aligned}f(y;\theta,\phi)&=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)\\&=\exp\left(\frac{y\mu-\frac{1}{2}\mu^2}{\sigma^2}-\frac{1}{2}\left(\frac{y^2}{\sigma^2}+\ln\left(2\pi\sigma^2\right)\right)\right)\end{aligned}$$

对照 (3.3.1) 式有

$$\theta=\mu,\phi=\sigma^2,a(\phi)=\phi$$

$$b(\theta)=\frac{1}{2}\mu^2=\frac{1}{2}\theta^2,c(y,\phi)=-\frac{1}{2}\left(\frac{y^2}{\phi}+\ln(2\pi\phi)\right)$$

例 3.2 设 $Y\sim P(\mu)$（参数为 μ 的 Poisson 分布），则 Y 的概率分布可表示为

$$P(Y=y)=\frac{\mu^y}{y!}\exp(-\mu)=\exp(y\ln\mu-\mu-\ln(y!))\quad y=0,1,2,\cdots$$

对照 (3.3.1) 式有

$$\theta=\ln\mu,\phi=1,a(\phi)=\phi,b(\theta)=\mu=\exp(\theta),c(y,\phi)=-\ln(y!)$$

例 3.3 设 $X\sim B(m,\mu)$（二项分布），其中 $0<\mu<1,m$ 为正整数，且二者均为未知参数。令 $Y=X/m$，则 Y 服从指数族分布。事实上，Y 的所有可能的取值为 $y=0,1/m,2/m,\cdots,1$，且

$$\begin{aligned}P(Y=y)&=P(X=my)\\&=C_m^{my}\mu^{my}(1-\mu)^{m-my}\\&=\exp\left(\frac{y\ln\left(\frac{\mu}{1-\mu}\right)+\ln(1-\mu)}{\frac{1}{m}}+\ln\left(C_m^{my}\right)\right)\end{aligned}$$

对照 (3.3.1) 式有

$$\theta = \ln\left(\frac{\mu}{1-\mu}\right), \phi = \frac{1}{m}, a(\phi) = \phi$$

$$b(\theta) = -\ln(1-\mu) = \ln(1+\exp(\theta)), c(y,\phi) = \ln\left(C_m^{my}\right)$$

3.3.2 连接函数

通过前面的学习，我们知“回归”一般是用于预测样本的值，这个值通常是连续的。在分类问题的应用上效果往往不理想。为了保留线性回归“简单、效果佳”的特点，又想让它能够进行分类，需要对预测值再做一次处理。这个多出来的处理过程，就是 GLM 所做的最主要的事。而处理这个过程的函数，我们把它叫做连接函数。

连接函数 $g(\cdot)$ 是将自变量第 i 组观测的线性组合与第 i 个观测值 Y_i 的期望连接起来的函数，即

$$g\left(\mathrm{E}(Y_i)\right) = g\left(\mu_i\right) = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j \quad i = 1, \cdots, n \tag{3.3.2}$$

在线性回归模型中，由于假定 $Y_i \sim N\left(\mu_i, \sigma^2\right)$，而 $\mathrm{E}(Y_i) \in \mathcal{R}$，$\beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j$ 一般也在 $\mathcal{R}$ 中取值，因此通常取 $g(\mathrm{E}(Y_i)) = \mathrm{E}(Y_i) = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j$ $(i = 1, \cdots, n)$。但是取 $g(\mathrm{E}(Y_i)) = \mathrm{E}(Y_i)$ 并不总是合适的，例如：当 Y_i 服从泊松分布即 $Y_i \sim P\left(\mu_i\right)$ 时，则 $E\left(Y_i\right) = \mu_i > 0$ 而 $\beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j$ 有可能取负值。因此直接建立模型

$$g(\mathrm{E}(Y_i)) = \mathrm{E}(Y_i) = \mu_i = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j$$

是不合理的。

表3.2中介绍几种常见的连接函数。

连接函数有很多种，一般在广义线性模型的建模中我们会使用正则连接函数，即

$$g(\mathrm{E}(Y)) = g(\mu) = \theta$$

3.3.3 广义线性模型

定义 3.9 设 $(Y_i; X_{i1}, X_{i2}, \cdots, X_{ip})$ $(i = 1, 2, \cdots, n)$ 为因变量 Y 和自变量 $X_1, X_2, \cdots, X_p$ 的观测值，若

表 3.2 常用的连接函数

连接函数名称	link function	连接函数公式
恒等函数	identity	$g(\mu)=\mu$
Logit 函数	logit	$g(\mu)=\ln(\mu/(1-\mu))$
Probit 函数	probit	$g(\mu)=\pi^{-1}(\mu)$
对数函数	log	$g(\mu)=\ln(\mu)$
逆函数	inverse	$g(\mu)=1/\mu$
平方根函数	sqrt	$g(\mu)=\sqrt{\mu}$
逆高斯分布	$1/\mathrm{mu}^2$	$g(\mu)=1/\mu^2$
重对数函数	loglog	$g(\mu)=\ln(-\ln(\mu))$
互补重对数函数	cloglog	$g(\mu)=\ln(-\ln(1-\mu))$
柯西函数	cauchit	$g(\mu)=\tan(\pi(\mu-0.5))$

(1) $Y_1,Y_2,\cdots,Y_n$ 相互独立，且对每个 i,Y_i 服从指数族分布，即

$$Y_i\sim f\left(y_i;\theta_i,\phi_i\right)=\exp\left(\frac{y_i\theta_i-b\left(\theta_i\right)}{a\left(\phi_i\right)}+c\left(y_i,\phi_i\right)\right)$$

(2) $g\left(\mu_i\right)=\beta_0+\sum_{j=1}^{p}X_{ij}\beta_j$，其中 $\mu_i=E\left(Y_i\right)(i=1,2,\cdots,n)$

则称 Y 与 $X_1,X_2,\cdots,X_p$ 服从广义线性模型。

表 3.3 常用的广义线性模型

分布	函数	模型
正态 (Gaussian)	$\mathrm{E}(Y)=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}$	普通线性模型
二项 (Binomial)	$\mathrm{E}(Y)=\dfrac{\exp\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}\right)}{1+\exp\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}\right)}$	Logistic 模型
泊松 (Poission)	$\mathrm{E}(Y)=\exp\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\beta}\right)$	对数线性模型

表3.4是三种常见的广义线性模型。由于每一个模型都有相应的指数分布，因此我们可以采用极大似然方法进行参数估计。

例 3.4 逻辑回归是一种广泛用于分类问题的机器学习算法。为了对数据进行分类，它使用了一个 S 形函数（sigmoid 函数）将线性模型的输出转换为 0 和 1 之间的概率值。极大似然估计是一种常用的方法来估计逻辑回归模型的参数。其基本思想是找到一组参数，使得在这组参数下，模型对训练数据的拟合最好。以下是计算逻辑回归参数的一般步骤：

1. 定义模型

首先需要定义逻辑回归模型。逻辑回归模型可以表示为：

$$P(Y_i = 1|X_i; \boldsymbol{\beta}) = h_{\boldsymbol{\beta}}(X_i) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T X_i}}$$

其中 Y_i 表示第 i 个样本的类别（0 或 1），$\boldsymbol{\beta}$ 是参数，$h_{\boldsymbol{\beta}}(X_i)$ 是模型预测的 X_i 属于类别 1 的概率。

2. 定义似然函数

假设有 n 个训练样本，则整个数据集的似然函数可以表示为：

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} h_{\boldsymbol{\beta}}(X_i)^{Y_i}(1 - h_{\boldsymbol{\beta}}(X_i))^{1-Y_i}$$

该式子可以理解为，对于每个样本，我们计算模型预测其类别的概率，并将其与其真实类别的概率相乘。由于样本之间是独立同分布的，因此我们将所有样本的似然函数相乘，从而得到整个数据集的似然函数。

3. 计算对数似然函数

对数似然函数可以方便地计算和优化，而不会影响最优解。因此，我们可以将似然函数取对数得到：

$$l(\boldsymbol{\beta}) = \log L(\boldsymbol{\beta}) = \sum_{i=1}^{n} [Y_i \log(h_{\boldsymbol{\beta}}(X_i)) + (1 - Y_i) \log(1 - h_{\boldsymbol{\beta}}(X_i))]$$

我们可以由对数似然函数构造损失函数，用梯度下降法求出使得损失最小对应的参数 $\boldsymbol{\beta}$，下面是逻辑回归中的损失函数：

$$J(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} [Y_i \log(h_{\boldsymbol{\beta}}(X_i)) + (1 - Y_i) \log(1 - h_{\boldsymbol{\beta}}(X_i))]$$

这个目标函数也叫做交叉熵损失函数。

4. 梯度下降求解最优参数

我们的目标是最大化对数似然函数 $l(\boldsymbol{\beta})$。使用梯度下降法，我们可以找到最优的 $\boldsymbol{\beta}$ 值。具体来说，我们首先计算对数似然函数的梯度：

$$\frac{\partial l(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{n} (h_{\boldsymbol{\beta}}(X_i) - Y_i) X_{ij}$$

然后，我们通过迭代更新 $\boldsymbol{\beta}$ 的值，直到达到最大化 $l(\boldsymbol{\beta})$ 的目标。

3.4 多元响应变量协方差广义线性模型 McGLM

传统的线性模型 LM（Linear Model）适用于处理单个响应变量的情形。广义线性模型 GLM（Generalized Linear Model，GLM）扩展了线性模型，可用于处理独

立非正态数据和单个响应变量情形，并且假设方差函数是已知的。Anderson [38] 和 Pourahmadi [39] 将 GLM 方法扩展到处理非独立非正态数据和单变量响应变量的情形，提出协方差广义线性模型（Covariance Generalized Linear Model，cGLM）。Wagner Hugo Bonat 和 Bent Jørgensen [40] 将 cGLM 扩展到处理非独立非正态数据和多个响应变量的情形，提出多元响应变量协方差广义线性模型（Multivariate Covariance Generalized Linear Model，McGLM），它是广义线性模型及协方差广义线性模型的扩展。

本节介绍多元响应变量协方差广义线性模型的原理、参数估计和模型选择等。

3.4.1 McGLM 模型的原理

McGLM 旨在处理具有时间和空间相关结构的多元响应变量，通过采用传统广义线性模型的方差函数方法来考虑非正态性，采用连接函数和线性预测器建模均值结构，采用协方差连接函数和矩阵线性预测器建模协方差结构，为正态和非正态多元数据分析提供通用的统计建模框架。

McGLM 的每一个响应变量均可由连接函数、方差函数、协方差函数三种函数及线性预测器和矩阵线性预测器来构成。

定义 3.10 McGLM

设样本容量 n，响应变量个数 R，响应变量矩阵 $\mathbf{Y}_{n\times R}=(\boldsymbol{Y}_1,\cdots,\boldsymbol{Y}_R)$，响应变量 $\mathbf{Y}$ 的期望值矩阵为 $\mathbf{M}_{n\times R}=(\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_R)$。$n\times n$ 阶矩阵 $\boldsymbol{\Sigma}_r$ 表示每个响应变量 $\boldsymbol{Y}_r(r=1,\cdots,R)$ 的协方差矩阵，该矩阵描述了每个响应变量内的协方差结构。$R\times R$ 阶矩阵 $\boldsymbol{\Sigma}_b$ 表示响应变量 $Y_1,\cdots,Y_R$ 间的的相关系数矩阵。$\mathbf{X}_r$ 为设计矩阵的 $n\times k_r$ 阶子阵。回归参数向量 $\boldsymbol{\beta}_r$ 为 $k_r\times 1$ 阶的回归参数子向量。多元响应变量协方差广义线性模型（McGLM）满足：

$$\begin{aligned}&\mathrm{E}(\mathbf{Y})=\mathbf{M}=\{\mathrm{g}_1^{-1}(\mathbf{X}_1\boldsymbol{\beta}_1),\cdots,\mathrm{g}_R^{-1}(\mathbf{X}_R\boldsymbol{\beta}_R)\}\\&\mathrm{var}(\mathbf{Y})=\mathbf{C}=\boldsymbol{\Sigma}_R\overset{G}{\otimes}\boldsymbol{\Sigma}_b\end{aligned}\tag{3.4.1}$$

其中 $\boldsymbol{\Sigma}_R\overset{G}{\otimes}\boldsymbol{\Sigma}_b=\mathrm{Bdiag}(\widetilde{\boldsymbol{\Sigma}}_1,\cdots,\widetilde{\boldsymbol{\Sigma}}_R)(\boldsymbol{\Sigma}_b\otimes\mathbf{I}_n)\mathrm{Bdiag}(\widetilde{\boldsymbol{\Sigma}}_1^{\mathrm{T}},\cdots,\widetilde{\boldsymbol{\Sigma}}_R^{\mathrm{T}})$ 是 $\boldsymbol{\Sigma}_R$ 与 $\boldsymbol{\Sigma}_b$ 的广义克罗内克积 [41]。它是 $nR\times nR$ 阶矩阵，表示所有响应变量的联合协方差矩阵。$\widetilde{\boldsymbol{\Sigma}}_r(r=1,\cdots,R)$ 表示协方差矩阵 $\boldsymbol{\Sigma}_r$ 的 Cholesky 分解的下三角矩阵，是 $n\times n$ 阶矩阵。运算符 $\mathrm{Bdiag}(\cdot)$ 表示分块对角矩阵。$\mathbf{I}_n$ 表示 $n\times n$ 阶单位阵。$\otimes$ 表示克罗内克积。函数 $\mathrm{g}_r(\cdot)$ 为均值结构中的连接函数（link function）。

一、响应变量的协方差矩阵 $\boldsymbol{\Sigma}_r$

不同取值类型的响应变量 $\mathbf{Y}_r$ 可取不同的协方差矩阵：

1、若 $\mathbf{Y}_r$ 取值为连续型、二值型、二项式、比例或指数数据，其协方差矩阵 $\boldsymbol{\Sigma}_r$ 可定义为

$$\boldsymbol{\Sigma}_r = V(\boldsymbol{\mu}_r;\boldsymbol{p}_r)^{\frac{1}{2}}(\boldsymbol{\Omega}(\boldsymbol{\tau}_r))V(\boldsymbol{\mu}_r;\boldsymbol{p}_r)^{\frac{1}{2}} \tag{3.4.2}$$

2、若 $\mathbf{Y}_r$ 取值为计数数据，其协方差矩阵 $\boldsymbol{\Sigma}_r$ 可采用如下形式

$$\boldsymbol{\Sigma}_r = \mathrm{diag}(\boldsymbol{\mu}_r) + V(\boldsymbol{\mu}_r;\boldsymbol{p}_r)^{\frac{1}{2}}(\boldsymbol{\Omega}(\boldsymbol{\tau}_r))V(\boldsymbol{\mu}_r;\boldsymbol{p}_r)^{\frac{1}{2}} \tag{3.4.3}$$

式 (3.4.2) 与 (3.4.3) 中，$V(\boldsymbol{\mu}_r;\boldsymbol{p}_r) = \mathrm{diag}(\vartheta(\boldsymbol{\mu}_r;\boldsymbol{p}_r))$ 是对角矩阵，其主对角线元素由方差函数（variance function）$\vartheta(\cdot\,;\boldsymbol{p}_r)$ 按元素应用于向量 $\boldsymbol{\mu}_r$ 而给出。方差函数 $\vartheta(\cdot\,;\boldsymbol{p}_r)$ 中的参数 $\boldsymbol{p}_r$ 为幂参数（power parameter）。

$\boldsymbol{\Omega}(\boldsymbol{\tau}_r)$ 为散度矩阵（dispersion matrix），描述了响应变量的协方差矩阵 $\boldsymbol{\Sigma}_r$ 中不依赖于均值结构的协方差部分。$\boldsymbol{\tau}_r$ 为散度参数 (dispersion parameter)。

二、方差函数 $\vartheta(\cdot\,;\boldsymbol{p}_r)$

方差函数 $\vartheta(\cdot\,;\boldsymbol{p}_r)$ 在 McGLM 中起着重要作用。模型选择不同的方差函数，则对应不同的响应变量边际分布。下面介绍 McGLM 常用的方差函数。

1、对于连续响应变量，方差函数采用幂函数 $\vartheta(\mu_r;p_r) = {\mu_r}^{p_r}$ 来描述 Tweedie 分布族。Tweedie 分布族的特例有高斯分布（$p_r = 0$）、伽马分布（$p_r = 2$）和逆高斯分布（$p_r = 3$）。

2、对于取值为二值型、二项式或比例数据的响应变量，方差函数常采用扩展二项式函数 $\vartheta(\mu_r;\boldsymbol{p}_r) = {\mu_r}^{p_{r1}}(1-\mu_r)^{p_{r2}}$。

3、对于计数数据的离散型响应变量，方差函数采用函数 $\vartheta(\mu_r;p_r) = \mu_r + \tau_r{\mu_r}^{p_r}$，(其中 τ_r 是散度参数）来描述 Poisson-Tweedie 分布族。由于散度参数仅出现在第二项中，因此协方差矩阵 $\boldsymbol{\Sigma}_r$ 采用式 (3.4.3) 的形式。Poisson-Tweedie 分布族的特例有 Hermite 分布（$p_r = 0$）、Neyman 分布（$p_r = 1$）、负二项分布（$p_r = 2$）和泊松逆高斯（Poisson-inverse Gaussian）分布（$p_r = 3$）。

三、幂参数 p_r

在 McGLM 二阶矩假设下，模型估计幂参数 p_r 的信息来自均值和方差的关系，因此估计幂参数 p_r 时需要均值向量有足够的变化，即要求线性预测器中存在显著的协变量。对于 McGLM 中所有的方差函数，幂参数 p_r 是区分重要分布的指标。模型中的算法允许估计幂参数 p_r，该参数可用于自动选择分布。

四、协方差连接函数 $h(\cdot)$ 与矩阵线性预测器

Anderson [38] 和 Pourahmadi [39]，Bonat 和 Jørgensen [40] 提出使用矩阵线性预测器结合协方差联接函数来对散度矩阵 $\boldsymbol{\Omega}(\boldsymbol{\tau}_r)$ 进行建模，即根据已知矩阵的线性组合建立一个模型

$$h(\boldsymbol{\Omega}(\boldsymbol{\tau}_r)) = \tau_{r0}\mathbf{Z}_{r0} + \cdots + \tau_{rD}\mathbf{Z}_{rD} \tag{3.4.4}$$

这种结构与均值结构的线性预测器相似，称为矩阵线性预测器 (matrix linear predictor)。将矩阵线性预测器 (3.4.4) 代入式 (3.4.1) 中，即得到多元响应变量协方差广义线性模型。

式 (3.4.4) 中，$h(\cdot)$ 为协方差连接函数 (covariance link function)；$\mathbf{Z}_{rd}(d = 0, \cdots, D)$ 是反映响应变量 $\mathbf{Y}_r$ 内协方差结构的已知矩阵，$\boldsymbol{\tau}_r = (\tau_{r0}, \cdots, \tau_{rD})^{\mathrm{T}}$ 为 $D+1$ 维散度参数向量。

McGLM 中协方差连接函数常采用恒等函数 (identity)、逆函数 (inverse) 和指数矩阵函数 (exponential-matrix)。

实际上，很难定义散度参数向量 $\boldsymbol{\tau}_r$ 的参数空间。Bonat 和 Jørgensen 利用倒数似然算法，使用一个调整常数来控制算法的步长，并避免参数向量 $\boldsymbol{\tau}_r$ 的不合理值。

矩阵预测器中的协方差结构常采用协方差矩阵的线性模型，用于处理非高斯数据、纵向自相关数据 (如复合对称、移动平均和一阶自回归等)、空间数据及区域数据等。关于如何指定矩阵 $\mathbf{Z}_{rd}$，详见 [40]。

五、连接函数 $g(\cdot)$

McGLM 模型中，连接函数 $g(\cdot)$ 将响应变量的期望与协变量联系起来，反映了模型的均值结构。常采用的连接函数也如表3.2。

3.4.2 参数估计

设参数向量 $\boldsymbol{\theta} = (\boldsymbol{\beta}^{\mathrm{T}}, \boldsymbol{\lambda}^{\mathrm{T}})^{\mathrm{T}}$，其中回归参数向量 $\boldsymbol{\beta}_{K\times 1} = (\boldsymbol{\beta}_1^{\mathrm{T}}, \cdots, \boldsymbol{\beta}_R^{\mathrm{T}})^{\mathrm{T}}$，散度参数向量 $\boldsymbol{\lambda}_{Q\times 1} = (\rho_1, \cdots, \rho_{R(R-1)/2}, p_1, \cdots, p_R, \boldsymbol{\tau}_1^{\mathrm{T}}, \cdots, \boldsymbol{\tau}_R^{\mathrm{T}})^{\mathrm{T}}$。

McGLM 模型采用基于二阶矩假设的估计函数法进行拟合。模型基于拟得分 (quasi-score) 函数和皮尔逊估计 (Pearson estimating) 函数的有效牛顿评分算法 (Newton scoring algorithm)，针对回归参数和散度参数，利用修正的追赶算法 (modified chaseralgorithm) 以及改进的倒数似然算法 (reciprocal likelihood algorithm)，通过调节常数 α 来调节步长，从而进行参数估计及统计推断，进而拟合 McGLM 模型。

具体来讲，我们让 $\mathcal{Y} = \left(\boldsymbol{Y}_1^{\mathrm{T}}, \ldots, \boldsymbol{Y}_R^{\mathrm{T}}\right)^{\mathrm{T}}$ 和 $\mathcal{M} = \left(\boldsymbol{\mu}_1^{\mathrm{T}}, \ldots, \boldsymbol{\mu}_R^{\mathrm{T}}\right)^{\mathrm{T}}$ 表示 $nR \times 1$ 列向量，其中的元素分别由响应变量矩阵 $\mathbf{Y}_{n\times R}$ 和期望值矩阵 $\mathbf{M}_{n\times R}$ 按列排序得到。

为了进行参数估计，我们采取拟得分函数 [42]

$$\boldsymbol{\psi}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \mathbf{D}^{\mathrm{T}}\mathbf{C}^{-1}(\mathcal{Y} - \mathcal{M}), \tag{3.4.5}$$

其中 $\mathbf{D} = \nabla_{\beta}\mathcal{M}$ 是一个 $nR \times K$ 矩阵，并且 ∇_{β} 表示对相应参数求梯度。此外，可以得到 $\boldsymbol{\psi}_{\boldsymbol{\beta}}$ 的 $K \times K$ 维敏感性矩阵和特异性矩阵

$$\mathbf{S}_{\boldsymbol{\beta}} = \mathrm{E}\left(\nabla_{\boldsymbol{\beta}}\boldsymbol{\psi}_{\boldsymbol{\beta}}\right) = -\mathbf{D}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{D} \text{ 和 } \mathrm{V}_{\boldsymbol{\beta}} = \mathrm{var}\left(\boldsymbol{\psi}_{\boldsymbol{\beta}}\right) = \mathbf{D}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{D} \tag{3.4.6}$$

散度参数所采用的皮尔逊估计函数的各个部分定义如下

$$\boldsymbol{\psi}_{\lambda_i}(\boldsymbol{\beta},\boldsymbol{\lambda}) = \operatorname{tr}\left(W_{\lambda i}\left(\boldsymbol{r}^{\mathrm{T}}\boldsymbol{r}-\mathbf{C}\right)\right), \qquad i=1,\ldots,Q, \tag{3.4.7}$$

其中 $\mathbf{W}_{\lambda_i} = -\partial\mathbf{C}^{-1}/\partial\lambda_i$ 并且 $\boldsymbol{r} = \mathcal{Y} - \mathcal{M}$。

$\boldsymbol{\psi}_{\boldsymbol{\lambda}}$ 的 $Q\times Q$ 维度敏感性矩阵的第 i 行第 j 列元素定义如下：

$$\mathrm{S}_{\lambda_{ij}} = \mathrm{E}\left(\frac{\partial}{\partial\lambda_i}\psi_{\lambda_j}\right) = -\operatorname{tr}\left(\mathbf{W}_{\lambda_i}\mathbf{C}\mathbf{W}_{\lambda_j}\mathbf{C}\right). \tag{3.4.8}$$

$\boldsymbol{\psi}_{\boldsymbol{\lambda}}$ 的 $Q\times Q$ 维特异性矩阵的第 i 行第 j 列元素定义如下：

$$\mathrm{V}_{\lambda_{ij}} = \operatorname{cov}\left(\psi_{\lambda_i},\psi_{\lambda_j}\right) = 2\operatorname{tr}\left(\mathbf{W}_{\lambda_i}\mathbf{C}\mathbf{W}_{\lambda_j}\mathbf{C}\right) + \sum_{l=1}^{nR} k_l^{(4)}\left(\mathbf{W}_{\lambda_i}\right)_{ll}\left(\mathbf{W}_{\lambda_j}\right)_{ll}, \tag{3.4.9}$$

$k_l^{(4)}$ 是 $\mathcal{Y}_l$ 的第四阶累积量（the fourth cumulant）。2004 年，Jørgensen 和 Knudsen [43] 提出修正的追赶算法用来解 $\boldsymbol{\psi}_{\boldsymbol{\beta}} = 0$ 和 $\boldsymbol{\psi}_{\boldsymbol{\lambda}} = 0$。具体更新公式如下：

$$\begin{aligned}\boldsymbol{\beta}^{(i+1)} &= \boldsymbol{\beta}^{(i)} - \mathbf{S}_{\boldsymbol{\beta}}^{-1}\psi_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}^{(i)},\boldsymbol{\lambda}^{(i)}\right)\\ \boldsymbol{\lambda}^{(i+1)} &= \boldsymbol{\lambda}^{(i)} - \alpha\mathbf{S}_{\boldsymbol{\lambda}}^{-1}\boldsymbol{\psi}_{\boldsymbol{\lambda}}\left(\boldsymbol{\beta}^{(i+1)},\boldsymbol{\lambda}^{(i)}\right)\end{aligned}$$

2016 年，Bonat 和 Jørgensen [40] 提出倒数似然算法（reciprocal likelihood algorithm），通过调节常数 α 来调节步长，从而对参数 $\boldsymbol{\lambda}$ 估计。具体更新公式如下：

$$\boldsymbol{\lambda}^{(i+1)} = \boldsymbol{\lambda}^{(i)} - \left[\alpha\boldsymbol{\psi}_{\boldsymbol{\lambda}}\left(\boldsymbol{\beta}^{(i+1)},\boldsymbol{\lambda}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\psi}_{\boldsymbol{\lambda}}\left(\boldsymbol{\beta}^{(i+1)},\boldsymbol{\lambda}^{(i)}\right)\mathrm{V}_{\boldsymbol{\lambda}}^{-1}\,\mathbf{S}_{\boldsymbol{\lambda}} + \mathbf{S}_{\boldsymbol{\lambda}}\right]^{-1}\boldsymbol{\psi}_{\boldsymbol{\lambda}}\left(\boldsymbol{\beta}^{(i+1)},\boldsymbol{\lambda}^{(i)}\right).$$

McGLM 中回归参数估计量对协方差结构的形式依赖性相对较小，而回归参数估计量的标准误差则直接依赖于协方差结构的选择。McGLM 还可以进行回归参数的稳健和偏差校正标准误差、残差分析等。详见 [40]。

3.5 回归分析实践

3.5.1 R 语言实践

单响应变量的线性回归模型

（1）下面对影响城镇居民消费支出的影响因素进行回归分析。

用 R 语言对数据进行回归分析，代码如下：

```
data=read.csv("D:/data.csv",header = T)#读取数据
rownames(data)=data[,1]#用地区名称对行命名
data=data[,-1]#删掉地区所在列
#y是城镇居民年消费支出
#剩余的变量x1,...,x9是自变量
lm=lm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9,data=data)#建立回归方程
summary(lm)#输出回归结果及显著性检验结果
anova(lm)#方差分析表
```

R 输出结果此处省略。从回归参数估计结果可以看出，F 检验对应的 P 值 <0.05，即回归方程通过了显著性检验。拟合的回归方程为

$$\widehat{Y} = -7574 + 1.228X_1 + 1.838X_2 + 0.904X_3 + 0.988X_4 + 1.570X_5 + 0.021X_6$$

$$+0.008X_7 + 66.570X_8 - 29.490X_9$$

回归参数估计结果显示，自变量 X_1~X_6 对应 t 检验的 P 值 <0.05，其回归系数显著不为零。X_1~X_6 对 Y 影响显著，且回归系数为正。但回归常数和自变量 X_7~X_9 的显著性检验不能通过，还要进一步完善模型。

（2）利用显著性自变量，使用 lm 函数对影响城镇居民消费支出作第二次回归，从而最终回归方程为

$$\widehat{Y} = -736.4 + 1.23X_1 + 1.83X_2 + 0.99X_3 + 0.96X_4 + 1.6X_5 + 0.02X_6 \qquad (3.5.1)$$

若自变量的新样本观测数据为

$$\boldsymbol{X}_0 = (8000, 2000, 6000, 2500, 3000, 9000, 8000, 103, 2.5)$$

则可利用最终回归方程 (3.5.1) 进行预测，由输出结果可知，因变量的点预测值为 26100.17，置信水平 95% 的预测区间为 (24443.79,27756.55)。

```
newdata=data.frame(x1=8000,x2=2000,x3=6000,x4=2500, x5=3000,x6=9000,
x7=8000,x8=103,x9=2.5)
ypred=predict(lm_final,newdata,interval = "prediction",level = 0.95)
ypred#点预测及区间预测
    fit         lwr         upr
1   26100.17    24443.79    27756.55
```

（3）对影响消费支出的各种因素进行岭回归分析。计算代码及运行结果如下：

```
datascale=data.frame(scale(data))#对数据进行标准化并转换为数据框格式存储
datascale=datascale[,1:7]#提取标准化后的因变量y及自变量x1~x6
library(MASS)#加载MASS包
ridge=lm.ridge(y~.,data = datascale,lambda = seq(0,10,0.1))#岭回归
beta = ridge$coef#回归系数
lambda = ridge$lambda#岭参数
plot(lambda,lambda,type="n",xlab="岭参数",ylab="岭回归系数",ylim=c(0,0.6))
linetype=c(1:6); color=c(1:6)#创建没有点和线的图形区域
for(i in 1:6)
```

```
    lines(lambda,beta[i,],lty=linetype[i],cex=0.75,col=color[i])#绘制岭迹线
legend("topright",legend=c("x1","x2","x3","x4","x5","x6"),cex=0.8,
lty= linetype,col=color)#添加图例
select(ridge)#选择使得GCV达到最小值的岭参数0.1
modified HKB estimator is 0.0409
modified L-W estimator is 0.0151
smallest value of GCV  at 0.1
install.packages("ridge") ; library(ridge)#下载并安装ridge包
#对选择的岭参数进行岭回归
Ridge_select=linearRidge(y~.-1,data = datascale,lambda = 0.1)
summary(Ridge_select)#输出结果
```

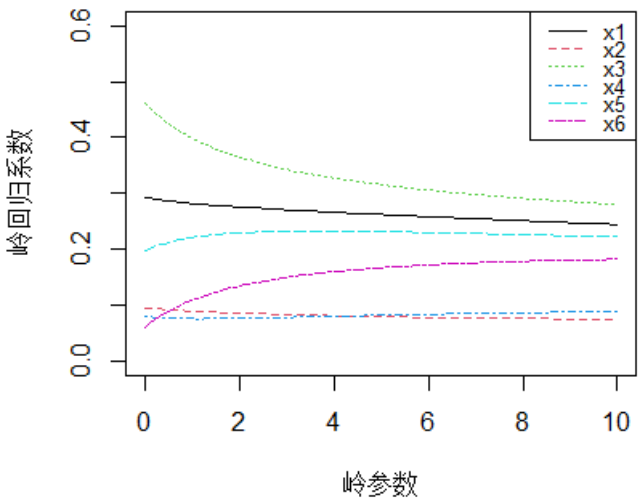


图 3.1 标准化岭迹图

岭迹图3.3显示，随着岭参数 λ 的增大，岭迹逐渐趋于稳定。当 $\lambda \geq 0.1$ 时，六条岭迹已较平稳了。使用 GCV 模型评估标准，可以选取 $\lambda = 0.1$ 来建立岭回归方程。由输出结果可知，自变量 $X_1 \sim X_6$ 均通过了显著性检验。

广义线性模型

下面使用 glm 函数对数据建立广义线性模型，其中 family 参数可以根据因变量的数据结构作出选择，比如 family 可以包括正态分布的线性模型，两点分布的 Logistic 模型以及 Poisson 的回归模型。下面只展示 Logistic 建模过程。这里我们选取了 45 名驾驶员的调查结果。如下表所示：

表 3.4 45 名驾驶员的调查结果

序号	y	x_1	x_2	x_3	序号	y	x_1	x_2	x_3	序号	y	x_1	x_2	x_3
1	1	1	17	1	16	0	1	68	1	31	0	0	17	0
2	0	1	44	0	17	0	1	18	1	32	1	0	45	0
3	0	1	48	1	18	0	1	68	0	33	1	0	44	0
4	0	1	55	0	19	1	1	48	1	34	0	0	67	0
5	1	1	75	1	20	0	1	17	0	35	1	0	55	0
6	1	0	35	0	21	1	1	70	1	36	0	1	61	1
7	1	0	42	1	22	0	1	72	1	37	0	1	19	1
8	0	0	57	0	23	1	1	35	0	38	0	1	69	0
9	1	0	28	0	24	0	1	19	1	39	1	1	23	1
10	1	0	20	0	25	0	1	62	1	40	0	1	19	0
11	0	0	38	1	26	1	0	39	1	41	1	1	72	1
12	1	0	45	0	27	1	0	40	1	42	0	1	74	1
13	1	0	47	1	28	0	0	55	0	43	1	1	31	0
14	0	0	52	0	29	1	0	68	0	44	0	1	16	1
15	1	0	55	0	30	0	0	25	1	45	0	1	61	1

R 代码如下所示：

```
#读取数据，数据是对45名驾驶员的调查结果
data=read.table("clipboard",header=T)
#x1视力状况（分类变量），x2年龄（数值型），x3驾车教育（分类变量）
#y去年是否出过事故（分类变量）
logit.glm<-glm(y~x1+x2+x3,family=binomial,data=data)#Logistic回归模型
summary(logit.glm)#Logistic回归模型结果
#输出结果如下：
             Estimate     Std. Error    t value  Pr(>|t|)
(Intercept)  0.597610     0.894831       0.668    0.5042
x1          -1.496084     0.704861      -2.123    0.0338*
x2          -0.001595     0.016758      -0.095    0.9242
x3           0.315865     0.701093       0.451    0.6523
```

由于 X_1 显著，重新运行仅仅基于 X_1 的 Logistic 模型，得到筛选后的的 Logistic 模型：

$$\Pr(Y=1 \mid X_1)=\frac{\exp\left(0.6190-1.3728X_1\right)}{1+\exp\left(0.6190-1.3728X_1\right)}$$

由结果可知，视力因素对事故率有影响。下面预测视力正常司机即 $X_1=1$ 及视力有问题司机，即 $X_1=0$ 发生事故的概率。R 代码如下：

```
pre1<-predict(logit.step,data.frame(x1=1))#预测视力正常司机Logistic回归结果
p1<-exp(pre1)/(1+exp(pre1))#预测视力正常司机发生事故概率
```

```
pre2<-predict(logit.step,data.frame(x1=0))#预测视力有问题的司机Logistic回归结果
p2<-exp(pre2)/(1+exp(pre2))#预测视力有问题的司机发生事故概率
c(p1,p2)#结果显示
1         1
0.32      0.65
```

McGLM 模型

mcglm 包是一个基于 Matrix 包的完整 R 实现。可以利用 mcglm 包的函数 mcglm() 来拟合 McGLM 模型。模型由一组列表指定，列表给出了线性和矩阵线性预测的符号描述。mcglm() 允许在连接、方差和协方差函数列表中进行选择，灵活地指定均值和协方差结构，并通过类似于 glm() 函数的公式界面来明确处理多元响应变量。mcglm() 将回归参数的拟得分函数和协方差参数的皮尔逊估计函数结合起来，采用估计函数法拟合 McGLM 模型。

本例采用澳大利亚健康调查数据集 ahs（Australian Health Survey），数据包括 1987-1988 年澳大利亚有关卫生系统准入措施的 5190 个样本数据，包含 5 个计数响应变量和有关社会条件的 9 个协变量。数据集 ahs 中主要变量包括：Ndoc，离散响应变量，离散变量，咨询医生或专家的次数；Nndoc，离散响应变量，与卫生专业人员咨询的次数；age，连续自变量，受访者的年龄（以年为单位）除以 100；income，连续自变量；受访者的年收入（以澳元计）除以 1000。利用数据集 ahs，建立响应变量 Ndoc、Nndoc 关于自变量 income、age 的二元响应变量协方差广义线性模型。

```
library(mcglm)#加载mcglm包
data(ahs)#数据集ahs
form1 = Ndoc ~ income+age#第一个响应变量Ndoc的线性预测公式
form2 = Nndoc ~ income+age#第二个响应变量Nndoc的线性预测公式
Z0 = mc_id(ahs)#构建矩阵线性预测器的第一个分量矩阵
options(digits = 3)#设定保留小数点后3位
#拟合McGLM模型,其中参数:linear_pred指定线性预测器;matrix_pred指定矩阵线性预测器;
#link指定连接函数;variance指定方差函数;data指定数据集
fit.ahs <- mcglm(linear_pred = c(form1, form2), matrix_pred = list(Z0, Z0),
                 link = c("log", "log"), variance = c("poisson_tweedie", "
                                                      poisson_tweedie"), data =
                                                       ahs)
summary(fit.ahs)
coef(fit.ahs)#参数估计值
gof(fit.ahs)#输出拟合优度
confint(fit.ahs)#模型参数的区间估计
anova(fit.ahs)#拟合模型的方差分析表
vcov(fit.ahs)#参数估计量的方差协方差阵
fitted(fit.ahs)#输出拟合值
residuals(fit.ahs)#输出皮尔逊、原始和标准化残差
plot(fit.ahs)
#输出各自变量的得分信息准则SIC，用于选择线性预测器的分量
mc_sic(fit.ahs,scope = c("income","age"), data = ahs,response = 1)
#输出每个分量矩阵的SIC协方差值、自由度等
```

```
mc_sic_covariance(fit.ahs,scope =Z0,idx = 1,response = 1)
```

对响应变量 Ndoc、Nndoc 分别选取线性预测器 form1、form2，矩阵线性预测器的第一个分量矩阵均为单位矩阵，连接函数均为对数函数，方差函数均为 poisson_tweedie，协方差连接函数均采用恒等变换。利用数据集 ahs 拟合 McGLM，运行结果表明，所有回归参数、散度参数及相关系数估计均显著。

3.5.2 Python 语言实践

单响应变量的线性回归模型

在 Python 中，常使用 statsmodels 和 sklearn 库来做线性回归。首先选择来自 sklearn 库的波士顿房价数据集建立回归模型。

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import datasets, linear_model
import statsmodels.api as sm
boston = datasets.load_boston() #导入数据集
X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.DataFrame(boston.target, columns=['MEDV'])
X = sm.add_constant(X) #由于此模型没有截距项，给训练集加上一列数值为1的特征
model = sm.OLS(y,X).fit()
print(model.summary())
```

输出结果显示，p 值大于 0.05 即认为与波士顿的房价波动变化关系不大，因此剔除 INDUS、AGE 选项，再次使用 statsmodel 中的最小二乘法：

```
X.drop(['AGE','INDUS'], axis=1, inplace=True)
model = sm.OLS(y,X).fit()
print(model.summary())
#拟合效果可视化
pre = model.predict(X.values)
plt.figure(figsize=(16,10))
plt.xticks(range(0, 506, 10), rotation=45)
plt.plot(range(len(X)), pre, 'b', label='predict')
plt.plot(range(len(y)), y.values, 'r', label='real')
plt.legend()
plt.show()
```

拟合效果如图3.2所示。

此外，用 sklearn 进行线性回归也是十分常见的作法，下面介绍用 Python 语言实现的，利用 sklearn 简单线性回归的通用方程拟合线性模型。这里我们将使用 SciKit-Learn 糖尿病数据集（https://scikit-learn.org/stable/datasets/index.html）。

该数据集包含 10 个特征（均值中心化和缩放后）和一个目标值：衡量基本线后一年的疾病进展情况。下面是代码：

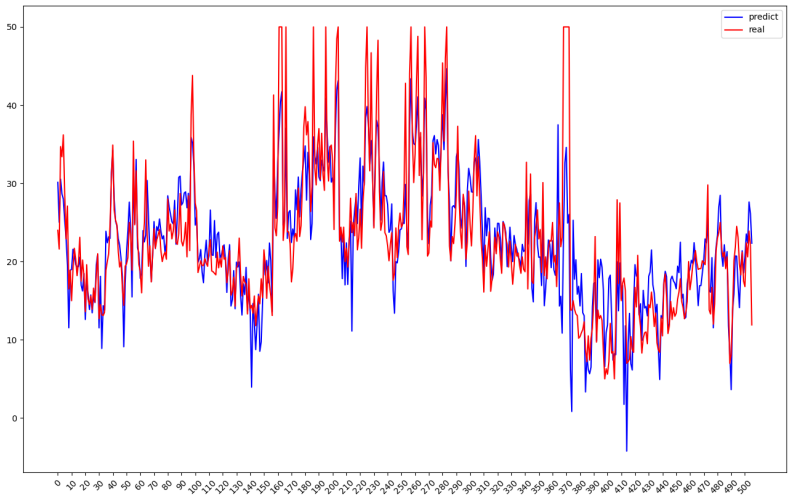


图 3.2 模型拟合效果

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
diabetes, target = load_diabetes(return_X_y=True) #载入数据集
diabetes = pd.DataFrame(diabetes) #准备模型数据
y = target #提取特征和目标
X = diabetes
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25,
random_state=27)#设置测试集和训练集
```

用 sklearn 进行线性回归, 得到 R^2 值（CV Mean）和标准偏差（STD）。

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
lr = LinearRegression().fit(X_train , y_train) #训 练 模 型
#得 到 交 叉 验 证 评 分
print('CV Mean: ', cross_val_score(lr, X_train , y_train,cv=5).mean())
print('STD: ', cross_val_score(lr, X_train , y_train,cv=5).std())
```

平均 R^2 值（CV Mean）为 0.48 和标准偏差（STD）为 0.14。这里 R^2 值较低表明模型不是很准确。标准偏差值（STD）为 0.14 表明可能对训练数据进行了过度拟合。解决过拟合的一个办法是简化模型。下面通过引入正则化来简化线性回归模型。

与其他线性模型一样，岭回归（Ridge）将在其拟合方法中接受数组 X, y，并将线性模型的系数 w 存储在其 $coefmember$ 中。在岭回归中用正则化参数 α 来控制模型的复杂性。α 的数值越高，就越能迫使系数向零移动，增加对模型的限制。这降低了训练性能，但也增加了模型的普适性。但把 α 设置得太高可能会导致模型过于简单，对数据的拟合不足。

下面使用 Scikit-Learn 的 Ridge 类来提高性能。

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1).fit(X_train , y_train)#选 取alpha=1训 练 模 型
print('CV Mean: ',cross_val_score(ridge, X_train , y_train , cv=5).mean())
print('STD: ',cross_val_score(ridge, X_train , y_train , cv=5).std())#得 到 交 叉 验
                                        证 评 分
```

以上结果，R^2 值（CV Mean）为 0.48 为 0.38，这意味着岭回归模型只能解释百分之 38 的方差，与上面的线性回归相比，没有改进。但标准差减少，这表明不太可能出现过拟合。

我们使用了上述正则化参数 α 的默认值，这可能不会带来最佳性能，下面通过调整正则化参数 α 来提高 R^2 值（CV Mean），下面使用网格搜索来找到一个最佳的 α 值。

```
from sklearn.model_selection import GridSearchCV
alpha = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
param_grid = dict(alpha=alpha)
#明 确 指 定 交 叉 验 证(cv)的 值 为5来 消 除FutureWarning提 示
grid = GridSearchCV(estimator=ridge, param_grid=param_grid, scoring='r2', verbose=1,
                                        n_jobs=-1, cv=5, iid=False)
grid_result = grid.fit(X_train, y_train)
print('Best Score:', grid_result.best_score_)
print('Best Params:', grid_result.best_params_)
```

广义线性模型

此小节我们学习一个利用 Python 语言实现的广义线性模型中对数线性泊松回归。下面这个例子说明了对数线性泊松回归在法国汽车第三方责任索赔数据集（$https://www.openml.org/d/41214$）上的使用，具体可参见：$A.Noll, R.Salzmanna ndM.V.Wuthrich, CaseStudy : FrenchMotorThird-PartyLiabilityClaims(Nove mber8, 2018).doi : 10.2139/ssrn.3164764$

首先，绘制数据集，在这个数据集中，每个样本对应一个保险单。可用的特征包括：司机年龄、车龄、车辆功率等。我们的目标是给定投保人群体的历史数据，预测一个新的投保人在发生车祸后的预期索赔频率。

从 OpenML 下载汽车索赔数据集：https://www.openml.org/d/41214

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import fetch_openml#这 一 模 块 主 要 包 含 现 实 生 活 中 的 大 型 的 数 据 集
df = fetch_openml(data_id=41214, as_frame=True).frame#导 入 汽 车 第 三 方 责 任 索 赔 数 据 集
```

索赔数量（ClaimNb）是一个正整数，可以被建模为泊松分布。然后假定它是在一个给定的时间间隔内以恒定的速率发生的离散事件的数量（曝光率，以年为单位）。

这里通过一个（按比例）泊松分布对 X 的频率 $y = ClaimNb/Exposure$ 进行建模，并使用 $Exposure$ 作为样本权重。

```
df["Frequency"] = df["ClaimNb"] / df["Exposure"]
#使用Exposure作为样本权重，得到平均频率
print("Average Frequency= {}".format(np.average(df["Frequency"],
weights=df["Exposure"])))
#输出零索赔的比例
print("Fraction of exposure with zero claims={0:.1%}".format(
df.loc[df["ClaimNb"] == 0, "Exposure"].sum() / df["Exposure"].sum()))
fig, (ax0, ax1, ax2) = plt.subplots(ncols=3, figsize=(16, 4))
#利用plt.subplots函数绘制并列的三个图
ax0.set_title("Number of claims")
ax0 = df["ClaimNb"].hist(bins=30, log=True, ax=ax0) #索赔数量
ax1.set_title("Exposure in years")
#风险指某一保单的承保期限，以年为单位
ax1 = df["Exposure"].hist(bins=30, log=True, ax=ax1)
ax2.set_title("Frequency (number of claims per year)")
ax2 = df["Frequency"].hist(bins=30, log=True, ax=ax2)#索赔频率（以每年索赔数量衡量）
```

运行产生以下结果和图像:

```
AVerage Frequency= 0.10070308464041304
Fraction of exposure with zero claims= 93.9%
```

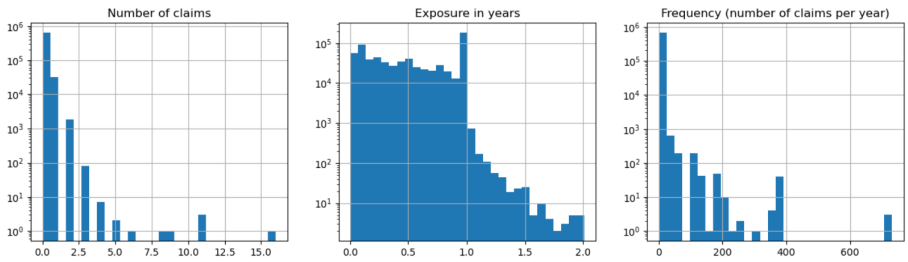


图 3.3 汽车索赔数据图

剩下的几栏可以用来预测索赔事件的频率。这些列是非常异质的，混合了不同尺度的分类和数字变量，可能分布非常不均匀。因此，为了用这些预测因子来拟合线性模型，可进行标准特征转换。

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer, OneHotEncoder
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
from sklearn.compose import ColumnTransformer
#导入pipeline类，可以将多个处理步骤合并为单个Scikit-Learn学习器
log_scale_transformer = make_pipeline(FunctionTransformer(np.log, validate=False),
StandardScaler())
#函数ColumnTransformer，可以选择地进行数据转换和预处理。
linear_model_preprocessor=ColumnTransformer([("passthrough_numeric", "passthrough",
["BonusMalus"]), ("binned_numeric", KBinsDiscretizer(n_bins=10, encode='onehot-dense'
```

```
), ["VehAge", "DrivAge"]), ("log_scaled_numeric", log_scale_transformer, ["Density"])
, ("onehot_categorical", OneHotEncoder(sparse=False), ["VehBrand", "VehPower",
"VehGas", "Region", "Area"])])
```

首先用（L2 惩罚）最小二乘法线性回归模型（即 Ridge 回归）对目标变量进行建模。预计这样的线性模型在这样一个大的数据集上会出现拟合不足，故使用较低的惩罚 α。

```
from sklearn.linear_model import Ridge #导入岭回归模型
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_poisson_deviance
#Pipeline把若干个命令串起来，前面命令的输出成为后面命令的输入
y, X = df.pop('Frequency'), df
y_train, y_test, X_train, X_test = train_test_split(y, X, random_state=0)
ridge_glm = Pipeline([("preprocessor", linear_model_preprocessor),
 ("regressor", Ridge(alpha=1e-6))]).fit(X_train, y_train, regressor__sample_weight=
 X_train["Exposure"])
#model.fit用来拟合模型参数
```

泊松偏差不能用模型所预测的非正数值来计算。对于确实返回一些非阳性预测的模型（如 Ridge），我们忽略了相应的样本，这意味着得到的泊松偏差是近似的。

```
pos = (y_test > 0) & (ridge_glm.predict(X_test) > 0)
print("Ridge evaluation: ")
print("MSE: ", mean_squared_error(y_test, ridge_glm.predict(X_test)))
print("MAE: ", mean_absolute_error(y_test, ridge_glm.predict(X_test)))
#输出岭回归评价MSE和MAE及平均泊松偏差
print("平均泊松偏差: ", mean_poisson_deviance(y_test[pos], ridge_glm.predict(X_test)
[pos]))
```

接下来在目标变量上拟合泊松回归。将正则化强度 α 设置为大约 1e-6，超过样本数（即 1e-12），以模仿 Ridge 回归，其 L2 惩罚项随样本数的变化而变化。由于泊松回归在内部对预期目标值的对数而不是直接对预期值进行建模（对数与恒等连接函数），X 和 y 之间的关系不再是完全线性的。因此，泊松回归被称为广义线性模型（GLM），而不是像 Ridge 回归那样的普通线性模型。

```
from sklearn.linear_model import PoissonRegressor #导入PoissonRegressor拟合泊松回归
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error,mean_absolute_error,
mean_poisson_deviance
n_samples = df_train.shape[0]
#Pipeline函数将多个任务抽象为一个包括多个步骤的流水线式工作
y, X = df.pop('Frequency'), df
y_train, y_test, X_train, X_test = train_test_split(y, X, random_state=0)
poisson_glm = Pipeline([("preprocessor", linear_model_preprocessor),
("regressor", PoissonRegressor(alpha=1e-12, max_iter=300))])
poisson_glm.fit(X_train, y_train, regressor__sample_weight=X_train["Exposure"])
```

同样地，输出输出泊松回归的 MSE 和 MAE 及平均泊松偏差

```
print("Ridge evaluation: ")
print("MSE: ", mean_squared_error(y_test, poisson_glm.predict(X_test)))
print("MAE: ", mean_absolute_error(y_test, poisson_glm.predict(X_test)))
print("平均泊松偏差: ", mean_poisson_deviance(y_test, poisson_glm.predict(X_test)))
```

McGLM 模型

McGLM 方法可以很方便的通过 R 中 mcglm 包中函数 mcglm() 实现。但是如果你想在 python 中使用上述方法，并且调用相关结果的话，可以参考下述步骤：

1、以上述 R 语言代码为例，首先将 McGLM 方法以 R 语言形式保存至桌面且命名为 “rcode.R”。

2、打开 python 编译器，并且安装 “rpy2” 库。

3、运行下述代码。

```
import rpy2.robjects as robjects #导入rpy2.robjects函数，用来调用R脚本。
robjects.r.source("C:/Users/Desktop/rcode.R")
#输入R脚本地址后，函数robjects.r.source会直接调用R脚本，并且输出结果。
```

3.6 习题

1、试证明平方和分解式：SST = SSR + SSE.

2、试证明，在 n 个样本误差服从独立同分布的正态分布假定下，因变量 $\boldsymbol{Y}$ 服从 n 维正态分布 $\boldsymbol{Y} \sim N_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_n)$。且未知参数 $\boldsymbol{\beta}$ 的极大似然估计量与最小二乘估计量等价，并求出 σ 的极大似然估计。

3、研究货运总量 Y（万吨）与工业总产值 X_1（亿元）、农业总产值 X_2（亿元）、居民非商品支出 X_3（亿元）的关系，数据见表3.5。

（1）计算 Y, X_1, X_2, X_3 的相关系数矩阵；

（2）求 Y 关于 X_1, X_2, X_3 的线性回归方程；

（3）对拟合的线性回归方程进行拟合优度检验、方程的显著性检验，对每个回归系数做显著性检验；

（4）若有回归系数未通过显著性检验，剔除，重新拟合回归方程并进行检验；

表 3.5 习题 1 数据表

序号	1	2	3	4	5	6	7	8	9	10
Y	160	260	210	265	240	220	275	160	275	250
X_1	70	75	65	74	72	68	78	66	70	65
X_2	35	40	40	42	38	45	42	36	44	42
X_3	1	2.4	2	3	1.2	1.5	4	2	3.2	3

（5）求回归系数的置信度 95% 的置信区间；

（6）求标准化回归方程；

（7）将 X_1, X_2, X_3 分别取 75，42，3.1，对因变量进行点预测及置信度 95% 的区间预测。

4、某种水泥在凝固时放出的热量 y（卡/克，cal/g）与水泥中的四种化学成分的含量 (%) 有关，这四种化学成分分别是 X_1 铝酸三钙（3CaO·Al2O3），X_2 硅酸三钙（3CaO·SiO2），X_3 铁铝酸四钙（4CaO·Al2O3·Fe2O3），X_4 硅酸二钙（2CaO·SiO2）。现观测到 13 组数据，数据见表3.6。用逐步回归法做变量选择，建立 Y 关于四种成分的线性回归方程。

表 3.6 习题 2 数据

序号	1	2	3	4	5	6	7	8	9	10	11	12	13
Y	78.5	74.3	104.3	87.6	95.9	109.2	102.7	72.5	93.1	115.9	83.8	113.3	109.4
X_1	7	1	11	11	7	11	3	1	2	21	1	11	10
X_2	26	29	56	31	52	55	71	31	54	47	40	66	68
X_3	6	15	8	8	6	9	17	22	18	4	23	9	8
X_4	60	52	20	47	33	22	6	44	22	26	34	12	12

5、一家大型商业银行有多家分行，近年来，该银行的贷款额平稳增长，但不良贷款额也有较大比例的提高。为弄清楚不良贷款形成的原因，下面利用银行业务的有关数据进行定量分析，找出控制不良贷款的办法。数据表3.7是该银行所属 25 家分行某年的有关业务数据。

（1）建立不良贷款与其余 4 个自变量的线性回归方程，对模型进行检验；

（2）分析回归模型的共线性；

（3）采取逐步回归法选择变量，对模型进行检验；

（4）建立不良贷款 y 对 4 个自变量的岭回归。

6、临床医学中为了研究麻醉剂用量与患者是否保持静止的关系，对 30 名患者在手术前 15 分钟给予一定浓度的麻醉剂后的情况进行了记录。记录数据来自于 R 软件 DAAG 包中自带的 anesthetic 数据集，见表3.8，其中，麻醉剂浓度为自变量

表 3.7 习题 3 数据

分行编号	1	2	3	4	5	6	7	8	9	10	11	12	13
y	0.9	1.1	4.8	3.2	7.8	2.7	1.6	12.5	1	2.6	0.3	4	0.8
x1	67.3	111.3	173	80.8	199.7	16.2	107.4	185.4	96.1	72.8	64.2	132.2	58.6
x2	6.8	19.8	7.7	7.2	16.5	2.2	10.7	27.1	1.7	9.1	2.1	11.2	6
x3	5	16	17	10	19	1	17	18	10	14	11	23	14
x4	51.9	90.9	73.7	14.5	63.2	2.2	20.2	43.8	55.9	64.3	42.7	76.7	22.8
分行编号	14	15	16	17	18	19	20	21	22	23	24	25	
y	3.5	10.2	3	0.2	0.4	1	6.8	11.6	1.6	1.2	7.2	3.2	
x1	174.6	263.5	79.3	14.8	73.5	24.7	139.4	368.2	95.7	109.6	196.2	102.2	
x2	12.7	15.6	8.9	0.6	5.9	5	7.2	16.8	3.8	10.3	15.8	12	
x3	26	34	15	2	11	4	28	32	10	14	16	10	
x4	117.1	146.7	29.9	42.1	25.3	13.4	64.3	163.9	44.5	67.9	39.7	97.1	

X，患者是否保持静止为因变量 Y，Y 取 1 时表示患者静止，Y 取 0 时表示患者有移动，试建立 Y 关于 X 的 Logistic 回归模型。

表 3.8 习题 4 数据

序号	1	2	3	4	5	6	7	8	9	10
麻醉剂浓度 X	1	1.2	1.4	1.4	1.2	2.5	1.6	0.8	1.6	1.4
患者是否保持静止 Y	1	0	1	0	0	1	1	0	1	0
序号	11	12	13	14	15	16	17	18	19	20
麻醉剂浓度 X	0.8	1.6	2.5	1.4	1.6	1.4	1.4	0.8	0.8	1.2
患者是否保持静止 Y	0	1	1	1	1	1	1	0	1	1
序号	21	22	23	24	25	26	27	28	29	30
麻醉剂浓度 X	0.8	0.8	1	0.8	1	1.2	1	1.2	1	1.2
患者是否保持静止 Y	0	0	0	0	0	1	0	1	0	1

7、R 中包 mcglm 自带数据集 NewBorn 是有关早产儿的呼吸物理治疗数据集，旨在评估呼吸物理疗法对出生体重低于 1500g 的早产儿通气后心肺功能的影响。NewBorn 数据集由护理团队 Waldemar Monastier hospital，Campo Largo，PR，Brazil 收集。样本容量 270 个，变量 21 个。数据集中主要变量含义解释见表3.9。利用响应变量 SPO2 关于自变量 Sex、APGAR1M、APGAR5M、PRE、HD，SUR 建立 McGLM 模型。

8、R 中包 mcglm 自带数据集 soya 中有关豌豆的样本数据。数据集有 7 个变量的 75 个观察值。数据集中的试验是在一个种植大豆的温室中进行的。此试验有两株按地块种植的植物。数据集 soya 中变量含义见表3.10。为研究豌豆的产量、每株豌豆的种子数量及可生长发育的豌豆比率问题，建立三元独立响应变量 grain、seeds、

表 3.9 习题 5 NewBorn 数据集的主要变量含义表

变量	描述
Sex	性别
APGAR1M	生命第一分钟的 APGAR 指数
APGAR5M	生命第五分钟的 APGAR 指数
PRE	因素，两个类别（过早：是；否）
HD	因子，两个水平（汉森病，是；否）
SUR	因子，两级（表面活性剂，是；否）
SPO2	氧饱和度（有界）

viablepeasP 关于协变量 pot、water、block 的 McGLM 模型并对模型拟合结果进行分析。

表 3.10 习题 6 soya 数据集中变量含义表

变量	描述
pot	自变量, 钾肥, 五水平因子（0,30,60,120,180）
water	自变量, 土壤中的水分, 三水平因子 (37.5,50,62.5)
block	自变量, 地块, 五水平因子 (I,II,III,IV,V)
grain	响应变量，连续型数据，单株粮食产量
seeds	响应变量，计数型数据，每株植物的种子数量
viablepeas	二项式型数据，-每株可存活豌豆的数量。
totalpeas	二项式型数据，-每株豌豆的总数量
viablepeasP	响应变量, 每株可存活豌豆比率, 值为 viablepeas/totalpeas

第四章　支持向量机

回归分析是一种监督学习任务，通常假设输入特征和输出目标之间存在线性或非线性的关系，通过统计方法，建立一个关于输入特征和输出目标之间的函数关系。而支持向量机通常将数据映射到高维空间中的超平面，通过找到一个最优的超平面，最大程度地将不同类别的样本分开。

4.1　简介

对给定样本数据集进行分类处理通常有多种办法，如前面章节中介绍的：GLM 或者 McGLM 等，而支持向量机（Support Vector Machine，SVM）作为众多分类器中的一种，却有着其他分类器没有的优点，下面我们就来介绍一下 SVM。

支持向量机（Support Vector Machine，SVM）是一种基于统计学习理论的有监督新型学习机，是由前苏联教授 Vladimir N.Vapnik 等 [58] 于 1963 年在统计学习理论基础上提出的。支持向量机理论最初用来解决两类线性可分问题，之后逐步推广到多分类、非线性等问题。与传统学习方法不同，支持向量机是结构风险最小化方法的近似实现，是借助最优化方法来解决机器学习分类问题的新工具。在处理线性可分问题时，其“对样本依赖小”等优点使其成为了众多分类器当中的佼佼者。

4.2　SVM 算法

4.2.1　SVM 的基本内容

最初 SVM 被提出时，是用来解决二分类问题的。其主要内容是：对于给定的包含两个类别的样本数据集 $\mathbf{Z}=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\ldots..(\boldsymbol{X}_n,Y_n)\},\boldsymbol{X}_i\in\mathcal{R}^p,Y_i\in\{-1,1\}$，确定一个超平面，对数据集进行二分类处理，如图4.1。

其分类判别式为：

$$f(\boldsymbol{X})=\operatorname{sign}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}+b\right) \tag{4.2.1}$$

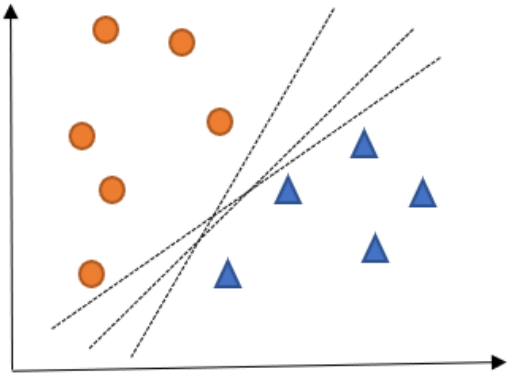


图 4.1 二分类超平面

可以看到，不止有一个超平面能够把样本数据正确的分为两类，SVM 算法是在这众多超平面中挑选出鲁棒性最强的超平面 $\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{X}+b^{*}$ 来作为最终的分类器。

SVM 可分为三种不同的类别：线性可分 SVM、线性 SVM、非线性 SVM。下面分别对这三种类型的 SVM 进行介绍。

4.2.2 线性可分 SVM

若至少存在一个超平面，可以将两类数据完全分开，此时的 SVM 被称为线性可分 SVM。线性可分 SVM 的原理就是要达到硬间隔最大化，又被称作最大间隔分类器。在样本点可以被超平面分为两类的基础上，线性可分 SVM 的目标是找到一个超平面，使得每个样本点到该超平面的距离都足够大。下面对算法理论进行详细阐述。对于给定样本数据集 $\mathbf{Z}=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\ldots..(\boldsymbol{X}_n,Y_n)\},\boldsymbol{X}_i\in\mathcal{R}^p,Y_i\in\{-1,1\}$。在样本空间中，将划分超平面用如下线性方程来描述：

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}+b=0$$

其中 $\boldsymbol{w}=(w_1,w_2,\ldots,w_p)^{\mathrm{T}}$ 为法向量，b 为位移项。显然，超平面可以被法向量 $\boldsymbol{w}$ 和位移 b 所确定，记为 $(\boldsymbol{w},b)$。对于 $(\boldsymbol{X}_i,Y_i)\in\mathbf{Z}$，其到超平面 $(\boldsymbol{w},b)$ 的距离可表示为：

$$r_i=\frac{\left|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b\right|}{\|\boldsymbol{w}\|} \tag{4.2.2}$$

所有 r_i $(i=1,\cdots,n)$ 的最小值称为间隔，可表示为式 (4.2.3)。而这些和超平面 $(\boldsymbol{w},b)$ 距离最小的样本点称作支持向量 （如图 4.2 所示）

$$\text{margin}=\min_{1\le i\le n} r_i \tag{4.2.3}$$

为了使这个超平面更具鲁棒性，SVM 算法的目标是通过调整参数 $\boldsymbol{w}$ 和 b，找到以最大间隔把两类样本分开的超平面，也称之为最大间隔超平面。首先，我们希望两类样本分别分割在该超平面的两侧；其次，两侧距离超平面最近的样本点到超平面的距离被最大化。即：

$$\max_{\boldsymbol{w},b} \min_{1\leq i\leq n} r_i \tag{4.2.4}$$

假设样本点可以被超平面分为两个类别，显然对于 $(\boldsymbol{X}_i, Y_i) \in \mathbf{Z}$，满足下面约束条件：

$$\begin{cases} \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b > 0, & Y_i = 1 \\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b < 0, & Y_i = -1 \end{cases}$$

可将其化简为：

$$Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) > 0 \tag{4.2.5}$$

这一约束条件意味着所有样本点被超平面正确分类。下面希望从能将样本数据集正确分类的超平面里选取特殊的一个超平面，使得两类样本数据集到这一超平面的最短距离最大化，即间隔最大化：

$$\begin{aligned} &\max_{\boldsymbol{w},b} \min_{1\leq i\leq n} r_i \\ &\text{s.t. } Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) > 0 \end{aligned} \tag{4.2.6}$$

在该约束条件下，可以将 r_i 转化为：

$$r_i = \frac{Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right)}{\|\boldsymbol{w}\|}$$

优化目标函数转化为：

$$\max_{\boldsymbol{w},b} \min_{1\leq i\leq n} \frac{Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right)}{\|\boldsymbol{w}\|}$$

为了便于化简计算，对超平面 $(\boldsymbol{w}, b)$ 进行放缩变换，使得 $\left|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right| \geq 1$，即 $Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) \geq 1$。

因此，优化问题 (4.2.6) 转换为下面形式：

$$\begin{aligned} &\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|} \\ &\text{s.t. } Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) \geq 1 \quad i = 1, 2, \cdots, n \end{aligned} \tag{4.2.7}$$

注意到目标函数中的 $\|\boldsymbol{w}\|$ 在分母上，所以可以将求最大值的约束优化问题转换成求最小值的凸二次规划问题：

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t. } Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) \geq 1 \quad i = 1, 2, \cdots, n \end{aligned} \tag{4.2.8}$$

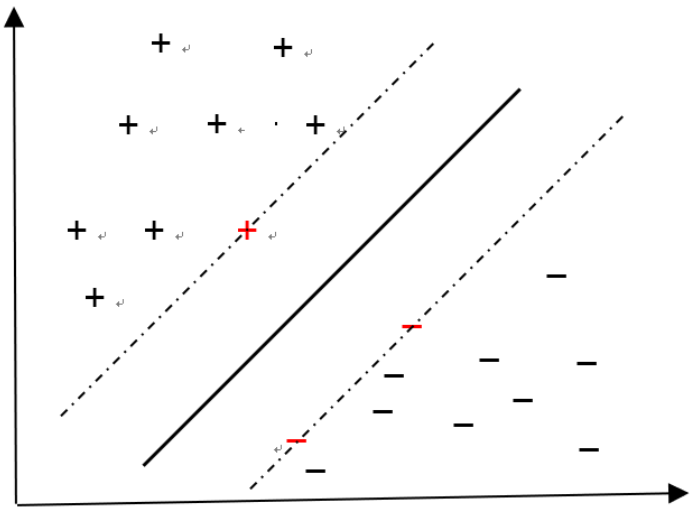


图 4.2 支持向量

(注：此处的 $\frac{1}{2}$ 并不影响最终的计算结果)

上式即为线性可分 SVM 的最优化问题。上述优化问题为不等式约束的优化问题，可利用拉格朗日乘子法将原问题转化为对偶问题。此外，这样做可以更自然的引入核函数，进而推广到非线性的分类问题。对于更一般化的约束优化问题来说，对偶问题可以将非凸问题转化为凸优化问题。

首先，引入拉格朗日乘子向量 $\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\cdots,\lambda_n)^{\mathrm{T}}$，并写出拉格朗日函数：

$$L(\boldsymbol{w},b,\lambda)=\frac{1}{2}\|\boldsymbol{w}\|^2-\sum_{i=1}^{n}\lambda_i\left(Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b\right)-1\right),\quad \lambda_i\geq 0 \tag{4.2.9}$$

此时，可以得到原问题 (4.2.8) 等价于一个极大极小化问题：

$$\min_{\boldsymbol{w},b}\max_{\lambda}L(\boldsymbol{w},b,\lambda) \tag{4.2.10}$$

进而，可定义原问题 (4.2.8) 的对偶问题：

$$\max_{\lambda}\min_{\boldsymbol{w},b}L(\boldsymbol{w},b,\lambda) \tag{4.2.11}$$

根据凸优化理论，可以知道，当原问题的目标函数和不等式约束函数是凸函数时，在不等式约束函数严格可行情况下，原问题最优解 $(\boldsymbol{w}^*,b^*)$ 与其对偶问题的最优解 λ^* 满足下面等式

$$L(\boldsymbol{w}^*,b^*,\lambda^*)=\min_{\boldsymbol{w},b}\max_{\lambda}L(\boldsymbol{w},b,\lambda)=\max_{\lambda}\min_{\boldsymbol{w},b}L(\boldsymbol{w},b,\lambda) \tag{4.2.12}$$

显然，这里目标函数 $\frac{1}{2}\|\boldsymbol{w}\|^2$ 及其约束函数 $1-Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b\right)\quad(i=1,2,\cdots,n)$ 是凸函数，那么上述等式在 SVM 中仍然成立。此时，可以对原问题的对偶问题求最优解，进而可以求出原问题的最优解。

根据 (4.2.11)，首先对 $\boldsymbol{w}$ 和 b 求极小值。令 $L(\boldsymbol{w}, b, \lambda)$ 对 $\boldsymbol{w}$ 和 b 的偏导数为 0，可以得到：

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \lambda_i Y_i = 0$$

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{i=1}^{n} \lambda_i Y_i \boldsymbol{X}_i$$

将所得的关系带入 $L(\boldsymbol{w}, b, \lambda)$，式 (4.2.10) 转化为

$$\begin{aligned} &\max_{\lambda} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j Y_i Y_j \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{X}_j \\ &\text{s.t.} \sum_{i=1}^{n} \lambda_i Y_i = 0, \quad \lambda_i \geq 0. \end{aligned} \tag{4.2.13}$$

式 (4.2.13) 是一个含等式约束的优化问题。我们将其转化为求最小值问题，上式等价于

$$\begin{aligned} &\min_{\lambda} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j Y_i Y_j \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{X}_j - \sum_{i=1}^{n} \lambda_i \\ &\text{s.t.} \sum_{i=1}^{n} \lambda_i Y_i = 0 \quad \lambda_i \geq 0 \end{aligned} \tag{4.2.14}$$

可以发现，上述对偶问题是一个二次规划问题，可以采用序列最小最优化（sequential minimal optimization, SMO）算法求解最优化问题 (4.2.14)。序列最小优化算法，其核心思想非常简单：每次只优化一个参数，其他参数先固定住，仅求当前这个优化参数的极值。

为提高求解效率，其求解过程每次选择两个变量进行优化，其他变量固定，具体而言，算法流程可描述为：

（1）选取两个待优化变量；

（2）固定其他变量的值，求解优化问题 (4.2.14)（直接令目标函数关于待优化变量梯度等于零即可）；

（3）不断重复第（1）、（2）两个步骤，直到算法收敛。

SMO 算法一般选择违反 KKT 条件最严重的样本所对应的变量作为第一个优化变量，第二个变量的选择应当使得目标函数有足够大的下降。一种启发式的选择方式为：首先寻找与第一个变量所对应样本间隔最大的样本，然后将该样本所对应的变量设置为第二个变量。

使用 SMO 算法可以求得对偶问题的最优解 $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \cdots, \lambda_n^*)$。根据式 (4.2.12)，对偶问题的最优解也是原问题的最优的拉格朗日乘子向量，接下来把最优的拉格朗日乘子向量带入原问题求解即可。

为了简化计算，我们引入 KKT 条件。已知 KKT 条件是判断等式约束和不等式约束的优化问题的必要条件。对于只含不等式约束的优化问题来说，如果目标函数和约束函数是凸函数，且不等式约束函数是可以满足的，那么 KKT 条件是这一不等式约束的优化问题的充要条件。在 SVM 中，原问题显然满足上述要求，那么我们可以得到原问题的充要条件（KKT 条件）即：

$$\begin{cases} \dfrac{\partial L}{\partial b} = 0 \Rightarrow \displaystyle\sum_{i=1}^{n} \lambda_i Y_i = 0 \\ \dfrac{\partial L}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \displaystyle\sum_{i=1}^{n} \lambda_i Y_i \boldsymbol{X}_i \\ \lambda_i \geq 0 \quad i = 1, \cdots, n \\ Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b \right) - 1 \geq 0 \quad i = 1, \cdots, n \\ \lambda_i \left(Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b \right) - 1 \right) = 0 \quad i = 1, \cdots, n \end{cases}$$

从而求解原问题等价于求解上述 KKT 条件，其中 λ_i 是拉格朗日乘数。

分析 KKT 条件我们还能发现更多关于样本的性质。对任意训练样本 $(\boldsymbol{X}_i, Y_i)$ 总有 $\lambda_i = 0$ 或 $Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b \right) - 1 = 0$。若 $\lambda_i = 0$，则所对应的样本点不会出现在系数 $\boldsymbol{w}$ 中，即该样本点不影响最终模型；若 $\lambda_i > 0$，则必有 $Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b \right) - 1 = 0$，则所对应的样本点位于最大间隔边界上，是一个支持向量。这显示出支持向量机的一个重要性质：训练完成后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关。

将对偶问题的最优解 $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \cdots, \lambda_n^*)$ 代入上述 KKT 条件等式，可以得到 $\boldsymbol{w}^*$，

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \lambda_i^* Y_i \boldsymbol{X}_i \tag{4.2.15}$$

为了得到最优决策平面，还需求解 b^*。注意到对任意支持向量 $(\boldsymbol{X}_k, Y_k)$，都有 $\lambda_k^* > 0$ 且 $Y_k \left(\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{X}_k + b \right) - 1 = 0$；对任意非支持向量 $(\boldsymbol{X}_l, Y_l)$，都有 $\lambda_l^* = 0$。

那么此时我们可以得到：

$$\boldsymbol{w}^* = \sum_{k \in \mathrm{K}} \lambda_k^* Y_k \boldsymbol{X}_k \tag{4.2.16}$$

K=$\{i \mid \lambda_i^* > 0, i = 1, \ldots, n\}$ 为所有支持向量的下标集。

将 $\boldsymbol{w}^*$ 带入 $Y_s \left(\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{X}_s + b \right) - 1 = 0$，其中 $(\boldsymbol{X}_s, Y_s)$ 是一支持向量，进而可以得到：

$$Y_s \left(\sum_{k \in \mathrm{K}} \lambda_k^* Y_k \boldsymbol{X}_k^{\mathrm{T}} \boldsymbol{X}_s + b \right) = 1$$

由于 $Y_s^2=1$，等式两边同时乘以 Y_s 可以解得 b^*。理论上可选取任意支持向量通过上式解得 b^*，但在现实任务中，常采用取所有支持向量的平均值的做法，即：

$$b^*=\frac{1}{|\mathrm{K}|}\sum_{s\in\mathrm{K}}\left(Y_s-\sum_{k\in\mathrm{K}}\lambda_k^*Y_k\boldsymbol{X}_k^{\mathrm{T}}\boldsymbol{X}_s\right)$$

最终得到最优决策平面：

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{X}+b^*=\sum_{i\in\mathrm{K}}\lambda_i^*Y_i\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}+\frac{1}{|\mathrm{K}|}\sum_{s\in\mathrm{K}}\left(Y_s-\sum_{k\in\mathrm{K}}\lambda_k^*Y_k\boldsymbol{X}_k^{\mathrm{T}}\boldsymbol{X}_s\right).$$

观察上述式子可以发现，最优决策平面并不需要用所有的样本来计算，而只需要用到支持向量。此外，最优决策平面与 $\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}$ 有关，这为将数据映射到高维空间，引入核函数做好了铺垫。

4.2.3 软间隔与线性 SVM

在实际问题当中常存在一些“噪声点”（如图4.3所示），采用线性可分 SVM 解决此类问题的过程中试图把这些“噪声点”也正确分类，往往影响模型的泛化能力，所以应当对模型放松要求，允许其在分类时出一点错误，这样会使模型的鲁棒性更好，这也是线性 SVM的基本出发点：即划分超平面虽然不能完全分开所有样本，但是可以使绝大多数样本正确被分类，其目标是软间隔最大化。

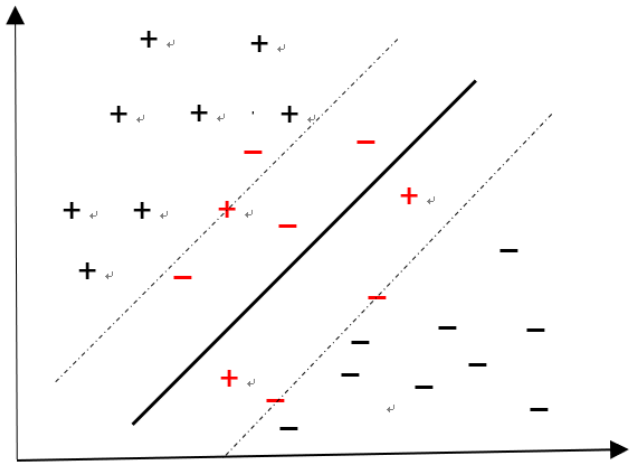


图 4.3 分类问题中的“噪声点”

假设划分超平面为：

$$f(\boldsymbol{X}) = w_1 X_1 + w_2 X_2 + \cdots + w_p X_p + b = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{X} + b = 0 \tag{4.2.17}$$

由于存在样本点不能满足式 (4.2.7) 中的约束条件，使得该约束不成立，因此在线性 SVM 中，对每一个样本引入松弛因子 ζ_i，此时约束条件被放宽为：

$$Y_i f\left(\boldsymbol{X}_i\right) = Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b\right) \geq 1 - \zeta_i, \quad (i = 1, \cdots, n)$$

上述约束条件允许样本位于间隔区域内，也允许出现错误分类。为了使在最大化间隔的同时不满足约束的点尽可能的少，线性 SVM 优化问题可写成如下形式：

$$\begin{aligned} \min_{\boldsymbol{w}, b, \zeta} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \zeta_i \\ \text{s.t.} \quad & Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b\right) \geq 1 - \zeta_i, \quad (i = 1, \cdots, n) \\ & \zeta_i \geq 0, \quad (i = 1, \cdots, n) \end{aligned} \tag{4.2.18}$$

其中 C 是可调节控制参数，在最小化目标函数处加了 $C \sum_{i=1}^{n} \zeta_i$，相当于施加了一个惩罚项。当 C 值越大时，对 ζ_i 惩罚越大；反之，C 值越小，对 ζ_i 惩罚越小。

最优化问题 (4.2.18) 是一个凸二次规划问题，可以通过拉格朗日乘子法进行求解，拉格朗日函数为：

$$L(\boldsymbol{w}, b, \boldsymbol{\lambda}, \boldsymbol{\beta}) = \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=1}^{n} \zeta_i - \sum_{i=1}^{n} \lambda_i \left(Y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{X}_i + b\right) - 1 + \zeta_i\right) - \sum_{i=1}^{n} \beta_i \zeta_i \tag{4.2.19}$$

其中 $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_n)^{\mathrm{T}}$，$\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_n)^{\mathrm{T}}$。分别对 $\boldsymbol{w}, b, \zeta_i$ 求导数，并令值为零，可以得到：

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{n} \lambda_i Y_i \boldsymbol{X}_i \\ \sum_{i=1}^{n} \lambda_i Y_i &= 0 \\ C - \lambda_i - \beta_i &= 0 \end{aligned} \tag{4.2.20}$$

将上面结果代入拉格朗日函数 (4.2.19)，可以得到如下对偶问题：

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j Y_i Y_j \boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{X}_j + \sum_{i=1}^{n} \lambda_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \lambda_i Y_i = 0 \\ & 0 \leq \lambda_i \leq C, \quad (i = 1, \cdots, n) \end{aligned} \tag{4.2.21}$$

对偶问题 (4.2.21) 等价于

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{i=1}^{n}\lambda_i\lambda_j Y_i Y_j \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_j - \sum_{i=1}^{n}\lambda_i \\ \text{s.t.} \quad & \sum_{i=1}^{n}\lambda_i Y_i = 0 \\ & 0 \le \lambda_i \le C, \quad (i = 1, \cdots, n) \end{aligned} \tag{4.2.22}$$

求解优化问题 (4.2.22)，得到最优解 $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \cdots, \lambda_n^*)^{\mathrm{T}}$，代入式 (4.2.20)，即可得到 $\boldsymbol{w}^*$。由于原始问题是凸二次规划问题，其解满足 KKT 条件，即

$$\beta_i \zeta_i = 0, \quad (i = 1, \cdots, n) \tag{4.2.23}$$

$$\lambda_i \left(Y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b\right) - 1 + \zeta_i\right) = 0, \quad (i = 1, \cdots, n) \tag{4.2.24}$$

在最优解 $\boldsymbol{\lambda}^*$ 中，取分量 λ_k^* 满足 $0 < \lambda_k^* < C$，由式 $C - \lambda_k^* - \beta_k^* = 0$ 得 $0 < \beta_k^* < C$，再由式 (4.2.23) 和 (4.2.24) 易计算得到 b^*，进而得到划分超平面。

由 $\boldsymbol{w}$ 的表达式可知，当 $\lambda_i^* = 0$ 时，该样本不会对最终的模型产生影响，当 $0 < \lambda_i^* \le C$ 时，该样本是支持向量，包括以下几种类型：

(1) $0 < \lambda_i^* < C$，则 $\beta_i > 0$，进而由式 (4.2.23) 可以得到 $\zeta_i = 0$，此时 $\boldsymbol{X}_i$ 位于最大间隔边界上；

(2) $\lambda_i^* = C$，$0 < \zeta_i < 1$，由式 (4.2.24) 知，$Y_i\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{X}_i + b^*\right) = 1 - \zeta_i > 0$。那么，此时 $\boldsymbol{X}_i$ 分类正确，并且位于划分超平面和间隔边界之间；

(3) $\lambda_i^* = C$，$\zeta_i = 1$，由式 (4.2.24) 知，$Y_i\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{X}_i + b^*\right) = 1 - \zeta_i = 0$。此时 $\boldsymbol{X}_i$ 位于划分超平面上；

(4) $\lambda_i^* = C$，$\zeta_i > 1$，由式 (4.2.24) 知，$Y_i\left(\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{X}_i + b^*\right) = 1 - \zeta_i < 0$。此时 $\boldsymbol{X}_i$ 被分类错误。

4.2.4 核函数与非线性 SVM

一般地，如果包含两个类别的样本集之间存在线性边界，那么建立线性 SVM 可以得到较好的分类效果。但是一些情况下，如果问题本身不是线性可分的，即边界是非线性的，那么线性 SVM 往往效果不佳，此时需建立非线性 SVM对样本数据进行非线性分类。

首先看一个简单的例子，如图4.4（左）所示，一维空间中的样本点属于两个类别，分别用红色和蓝色表示不同类别。在这种情况下，无法用一个点（即一维空间的

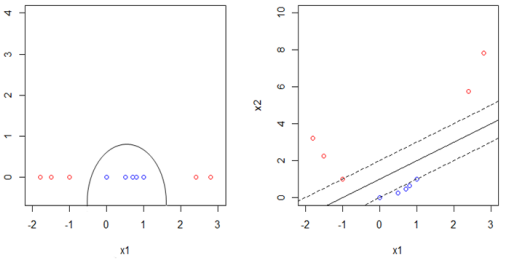


图 4.4 非线性 SVM 图例

超平面）来将不同类别的样本分开，但是可用一条复杂的曲线将它们分为两个类别。显然，分类边界不是线性的。

为了解决上述非线性可分问题，尝试将自变量的二次项添加到超平面中，即此时的划分面是：

$$\left\{X : f(X) = w_1 X + w_2 X^2 + b = 0\right\} \tag{4.2.25}$$

式 (4.2.25) 中，可以将 X 看作一个变量 X_1，X^2 看作另一个变量 X_2，此时变成二维空间的线性问题。如图4.4 （右），在构造出的二维空间中，样本点可以用一个线性超平面（即图中的黑色实线）分为两个类别。因此，在二维空间中，线性 SVM 是有效的。同理，可将上述问题扩展到 p 维空间中，此外，还可使用不同类型的多项式，如三次、四次甚至是更高阶，以及交叉项构造特征空间，进而在新构造的特征空间中建立线性超平面。

总的来说，对于线性不可分的数据（如图 4.5），可对样本进行变换，将其映射到高维特征空间，使得样本集在高维空间中线性可分，如图 4.6 所示。

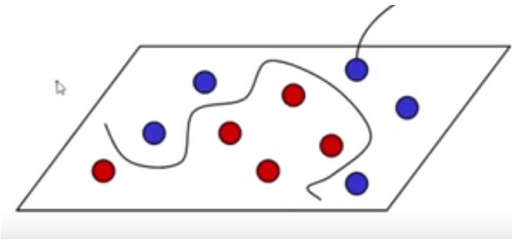


图 4.5 线性不可分的数据

如果可以找到合适的特征空间，便可以将原问题通过特征空间中的线性超平面进行划分。然而实际问题中，一方面，构造合适的特征空间是非常困难的；另一方面，特征空间的构造方法可能并不唯一，如果处理不当，将会得到维数较大的特征空间，此时的计算量将变得复杂。因此有必要寻找合适的构造特征空间的方法，使得在新的特征空间中能有效求解得到线性超平面。

核方法通过核函数（kernel）构造特征空间，使得在新的特征空间中能有效求解线性超平面。在介绍核函数之前，有必要先对内积的概念进行介绍。

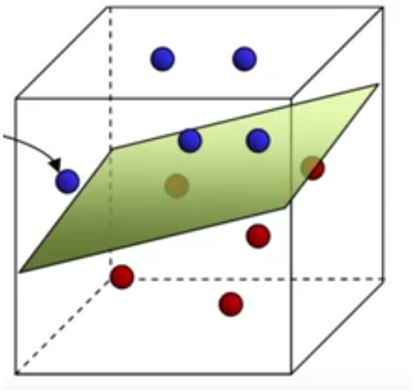


图 4.6 高维空间线性可分

两个样本 $\boldsymbol{X}_i = (X_{i1}, \cdots, X_{ip})^{\mathrm{T}}$ 和 $\boldsymbol{X}_k = (X_{k1}, \cdots, X_{kp})^{\mathrm{T}}$ 的内积定义为：

$$\langle \boldsymbol{X}_i, \boldsymbol{X}_k \rangle = \sum_{j=1}^{p} X_{ij} X_{kj} \tag{4.2.26}$$

通过前几小节的证明可以发现，线性支持向量机的对偶问题可以描述成内积的形式，即：

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j Y_i Y_j \langle \boldsymbol{X}_i, \boldsymbol{X}_j \rangle - \sum_{i=1}^{n} \lambda_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \lambda_i Y_i = 0 \\ & 0 \leq \lambda_i \leq C, \quad (i = 1, \cdots, n). \end{aligned} \tag{4.2.27}$$

通过计算可得到对偶问题的拉格朗日乘子 λ_i，根据 KKT 条件，可以计算最优决策平面且并将其描述为内积的形式：

$$f(\boldsymbol{X}) = \sum_{i=1}^{n} \lambda_i Y_i \langle \boldsymbol{X}, \boldsymbol{X}_i \rangle + b \tag{4.2.28}$$

上式中有 n 个参数 $\lambda_i, i = 1, \cdots, n$，每个训练样本对应一个参数。为了得到 $f(X)$，需要计算 $\boldsymbol{X}$ 与每个训练样本 $\boldsymbol{X}_i$ 之间的内积，但事实证明，有且仅有支持向量所对应的 λ_i 是非零的。若用 S 表示支持向量样本点指标的集合，那么式 (4.2.28) 可以改写成：

$$f(\boldsymbol{X}) = \sum_{i \in S} \lambda_i Y_i \langle \boldsymbol{X}, \boldsymbol{X}_i \rangle + b \tag{4.2.29}$$

式 (4.2.29) 的求和项比式 (4.2.28) 少得多。总而言之，我们仅需知道内积便可以计算 $f(\boldsymbol{X})$。

对于非线性可分问题，我们的初衷是想找到这样一个映射 ψ，将样本点 $\boldsymbol{X}_i$ 映射为新的高维特征空间的特征向量 $\psi(\boldsymbol{X}_i)$，进而通过其对偶问题求解相应拉格朗日乘子。最后，考虑原问题的 KKT 条件，求解最优决策平面。

在上述步骤中，可以发现在原空间内，对偶问题及原问题的 KKT 条件求解以内积的形式出现。自然的，在新的特征空间内，只需知道 $\langle\psi(\boldsymbol{X}_i),\psi(\boldsymbol{X}_j)\rangle$ 便可以在新的特征空间中求解原问题的对偶问题，然后得到最优决策平面。高维空间中内积的计算较为复杂，可通过核技巧直接定义核函数 $K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\langle\psi(\boldsymbol{X}_i),\psi(\boldsymbol{X}_j)\rangle$ 来解决复杂计算问题，通俗的来说，就是将求解映射 ψ 的问题转化为选择合适核函数 $K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)$ 的问题。

对于非线性可分问题，考虑核技巧后，此时 $f(\boldsymbol{X})$ 变为:

$$f(\boldsymbol{X})=\sum_{i\in S}\lambda_i Y_i K\left(\boldsymbol{X},\boldsymbol{X}_i\right)+b \tag{4.2.30}$$

其中 $K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)$ 被称为核函数。

非线性 SVM 的处理方法是构造核函数以代替高维空间中的内积计算，在高维特征空间中解决原始空间中线性不可分的问题。具体来说，在线性不可分的情况下，首先选择一个合适的核函数，核函数的作用是避免在高维空间中进行内积计算，然后在高维空间中执行线性可分的 SVM，最终计算出最优的划分超平面，从而达到把低维空间中线性不可分的数据集进行分类的目的。

核函数的选择至关重要，将影响 SVM 对数据集的分类效果。实际问题中，应当根据问题本身特征选择合适的核函数。表 4.1 给出了常用的几种核函数:

表 4.1 几种常用核函数

名称	表达式	参数
线性核	$K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_j$	
多项式核	$K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\left(\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_j\right)^d$	$d\geq 0$ 为多项式的次数
高斯核	$K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\exp\left(-\dfrac{\left\|\boldsymbol{X}_i-\boldsymbol{X}_j\right\|^2}{2\sigma^2}\right)$	$\sigma>0$ 为高斯的带宽（width）
拉普拉斯核	$K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\exp\left(-\dfrac{\left\|\boldsymbol{X}_i-\boldsymbol{X}_j\right\|}{\sigma}\right)$	$\sigma>0$
Sigmoid 核	$K\left(\boldsymbol{X}_i,\boldsymbol{X}_j\right)=\tanh\left(\beta\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_j+\theta\right)$	tanh 为双曲正切函数,$\beta>0,\theta>0$

注意，关于核函数的选择一直以来都是支持向量机研究的热点，通常情况下，高斯核函数是使用最多的。此外，采用核函数而不是直接将样本集映射到特征空间的优势在于，无需明确指明映射函数，并且可以避免计算高维空间中的内积，大大降低计算量。

4.3 SVM 与逻辑斯蒂回归的关系

本节讨论 SVM 与逻辑斯蒂回归的关系。首先, 为了建立支持向量机分类器

$$f(\boldsymbol{X}) = b + w_1 X_1 + \cdots + w_p X_p$$

最优化问题 (4.2.18) 采用软间隔最大化的学习策略，进而得到分隔超平面以及决策函数。若将松弛变量 ζ_i 取为如下形式:

$$\zeta_i = \begin{cases} 1 - Y_i f\left(\boldsymbol{X}_i\right), & Y_i f\left(\boldsymbol{X}_i\right) < 1 \\ 0, & Y_i f\left(\boldsymbol{X}_i\right) \geq 1 \end{cases}$$

即,

$$\zeta_i = \max\left(0, 1 - Y_i f\left(\boldsymbol{X}_i\right)\right) \tag{4.3.1}$$

则 (4.2.18) 等价于如下优化问题:

$$\min_{\boldsymbol{w}, b} \ \gamma\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} \max\left(0, 1 - Y_i f\left(\boldsymbol{X}_i\right)\right) \tag{4.3.2}$$

其中 γ 为调节控制参数，$\gamma\|\boldsymbol{w}\|^2$ 是岭回归的惩罚项，这一项需要根据偏差的关系来确定。下面阐述 (4.3.2) 与 (4.2.18) 的等价性:

由 ζ_i 的定义，显然问题 (4.2.18) 中的两个约束条件均成立。又因为 (4.3.2) 可写为:

$$\min_{\boldsymbol{w}, b} \ \gamma\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} \zeta_i \tag{4.3.3}$$

如果取 $\gamma = \dfrac{1}{2C}$, 那么可转化为

$$\min_{\boldsymbol{w}, b} \ \frac{1}{C}\left(\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \zeta_i\right)$$

问题 (4.3.2) 可转化为 (4.2.18)。

反之，当 (4.2.18) 中的 ζ_i 取为 (4.3.1) 时，可转化为 (4.3.2)。因此 (4.3.2) 与 (4.2.18) 是等价的。

令 $P(\boldsymbol{w}) = \|\boldsymbol{w}\|^2$，$L\left(Y_i f\left(\boldsymbol{X}_i\right)\right) = \max\left(0, 1 - Y_i f\left(\boldsymbol{X}_i\right)\right)$，式 (4.3.2) 可转化为如下的 “损失函数 + 惩罚” 的形式:

$$\min_{\boldsymbol{w}, b} \ \sum_{i=1}^{n} L\left(Y_i f\left(\boldsymbol{X}_i\right)\right) + \gamma P(\boldsymbol{w})$$

这里，把具有形式

$$L\left(Y f\left(\boldsymbol{X}\right)\right) = \max\left(0, 1 - Y f\left(\boldsymbol{X}\right)\right)$$

的损失函数称为hinge 损失函数（合页损失函数）。这类损失函数的特点是，对于边界外且分类正确的样本点，损失为零；对于边界上的样本点以及分类错误的样本点，损失是线性的。

对于逻辑斯蒂回归，加入 L2 正则化项目后，其优化函数变为：使用的损失函数为

$$L(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}[Y_i \log(h_{\boldsymbol{w}}(\boldsymbol{X}_i)) + (1-Y_i)\log(1-h_{\boldsymbol{w}}(\boldsymbol{X}_i))] + \gamma\|\boldsymbol{w}\|^2$$

其中，n 是样本数量，Y_i 是第 i 个样本的真实标签 (0 或 1)，$h_{\boldsymbol{w}}(\boldsymbol{X}_i) = \dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{X}_i}}$ 是第 i 个样本的预测概率 (由 Sigmoid 函数得到)。

SVM 与逻辑斯蒂回归的目标是都减少” 错误率 “。SVM 是寻找最优划分超平面降低错误率，其损失函数为 hinge 损失；逻辑斯蒂回归通过最大化样本属于其真实类别的概率来降低错误率，其损失函数为负对数似然。两者的正则化项都是 L2 正则。

因此，SVM 和 Logistic 回归的结果通常是非常接近的，对于一个给定的问题，该如何选择是使用 SVM 还是 Logistic 回归呢？这个问题更多时候需要根据实际数据选择合适的算法，当然也有一些特殊情形：（1）当类别区分度较高时，可以选择 SVM；（2）如果想要得到估计的概率，那么就需要选择 Logistic 回归；（3）对于决策边界是非线性的情况，核函数的 SVM 方法应用更加广泛。

4.4 支持向量回归

本节将介绍如何将支持向量机扩展到回归问题中，称为支持向量回归（Support Vector Regression，SVR），其思想与分类问题类似，不同之处在于支持向量回归的目的是要寻找一个超平面，在距离超平面 ϵ 范围内尽可能的包含最多的样本点。传统回归方法通过计算预测值与真实值差别计算损失，只要两者不一致，就会产生损失；而支持向量回归容许预测值与真实值之间存在 ϵ 的差别，仅当差别值大于 ϵ 时才会产生损失。

因此，可引入如下ϵ 不敏感损失函数：

$$l_\epsilon(r) = \begin{cases} 0, & |r| \geq \epsilon \\ |r| - \epsilon, & |r| > \epsilon \end{cases}$$

此时，SVR 问题的优化目标函数可表示为：

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} l_\epsilon\left(f\left(\boldsymbol{X}_i\right) - Y_i\right) \tag{4.4.1}$$

其中，$\|\boldsymbol{w}\|$ 表示权重的向量范数，C 是正则化参数，ϵ 是预定义间隔。

引入两个松弛变量 ζ_i 和 $\hat{\zeta}_i$，分别表示上下两侧的松弛程度，(4.4.1) 可变化为：

$$\begin{aligned}
&\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\left(\zeta_i + \hat{\zeta}_i\right) \\
&\text{s.t.} \quad f\left(\boldsymbol{X}_i\right) - Y_i \leq \epsilon + \zeta_i, \\
&\qquad Y_i - f\left(\boldsymbol{X}_i\right) \leq \epsilon + \hat{\zeta}_i, \\
&\qquad \zeta_i \geq 0, \quad \hat{\zeta}_i \geq 0, \quad i = 1, 2, 3, \cdots, n
\end{aligned} \tag{4.4.2}$$

利用拉格朗日乘子法，推导上述优化问题的对偶问题。首先写出拉格朗日函数，

$$\begin{aligned}
&L(\boldsymbol{w}, b, \boldsymbol{\xi}, \hat{\boldsymbol{\xi}}, \boldsymbol{\mu}, \hat{\boldsymbol{\mu}}, \boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) \\
&= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\left(\zeta_i + \hat{\zeta}_i\right) - \sum_{i=1}^{n}\left(\mu_i\zeta_i + \hat{\mu}_i\hat{\zeta}_i\right) \\
&+ \sum_{i=1}^{n}\alpha_i\left(f\left(\boldsymbol{X}_i\right) - Y_i - \varepsilon - \zeta_i\right) + \sum_{i=1}^{n}\hat{\alpha}_i\left(Y_i - f\left(\boldsymbol{X}_i\right) - \varepsilon - \hat{\zeta}_i\right)
\end{aligned} \tag{4.4.3}$$

其中 $\boldsymbol{\mu}, \hat{\boldsymbol{\mu}}, \boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}} \geq 0$。分别对 $\boldsymbol{\omega}$，b，ζ_i 和 $\hat{\zeta}_i$ 求偏导并令其为零得到，

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n}\left(\hat{\alpha}_i - \alpha_i\right)\boldsymbol{X}_i = 0 \tag{4.4.4}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n}\left(\hat{\alpha}_i - \alpha_i\right) = 0 \tag{4.4.5}$$

$$\frac{\partial L}{\partial \zeta_i} = C - \mu_i - \alpha_i = 0 \tag{4.4.6}$$

$$\frac{\partial L}{\partial \hat{\zeta}_i} = C - \hat{\mu}_i - \hat{\alpha}_i = 0 \tag{4.4.7}$$

将上述等式代入 (4.4.3) 可得，

$$\begin{aligned}
Q(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = &-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\hat{\alpha}_i - \alpha_i\right)\left(\hat{\alpha}_j - \alpha_j\right)\boldsymbol{X}_i^T\boldsymbol{X}_j \\
&- \varepsilon\sum_{i=1}^{n}\left(\alpha_i + \hat{\alpha}_i\right) + \sum_{i=1}^{n}Y_i\left(\hat{\alpha}_i - \alpha_i\right)
\end{aligned}$$

那么，可以得到对偶问题，

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}} \quad Q(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) \\
&\text{s.t.} \quad \sum_{i=1}^{n}\left(\hat{\alpha}_i - \alpha_i\right) = 0 \\
&\qquad \boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}} \in [0, C]
\end{aligned} \tag{4.4.8}$$

KKT 条件为：

$$\begin{aligned}
\alpha_i \left(f\left(\boldsymbol{X}_i\right) - Y_i - \varepsilon - \xi_i\right) &= 0 \\
\hat{\alpha}_i \left(Y_i - f\left(\boldsymbol{X}_i\right) - \varepsilon - \hat{\xi}_i\right) &= 0 \\
\left(C - \alpha_i\right) \zeta_i &= 0 \\
\left(C - \hat{\alpha}_i\right) \hat{\zeta}_i &= 0 \\
\alpha_i \hat{\alpha}_i &= 0 \\
\zeta_i \hat{\zeta}_i &= 0
\end{aligned} \tag{4.4.9}$$

接下来，利用 SMO 方法求解上述对偶问题，得到 α_i 和 $\hat{\alpha}_i$ 的值，并由等式 (4.4.4) 可得，SVR 的解为：

$$f(\boldsymbol{X}) = \sum_{i=1}^{n} \left(\hat{\alpha}_i - \alpha_i\right) \boldsymbol{X}_i^T \boldsymbol{X} + b \tag{4.4.10}$$

显然，SVR 的支持向量为所有使得 $\left(\hat{\alpha}_i - \alpha_i\right) \neq 0$ 的样本，这些样本位于间隔外面，故 SVR 解依然具有稀疏性。

那么，如何求解 b 呢？事实上，得到所有 α_i 的值后，任取 $0 < \alpha_j < C$，根据 KKT 条件必然有 $\zeta_j = 0$，此时可以得到，

$$b = Y_j + \epsilon - \sum_{i=1}^{n} \left(\hat{\alpha}_i - \alpha_i\right) \boldsymbol{X}_i^T \boldsymbol{X}_j \tag{4.4.11}$$

在实际求解中，一般选取多个满足 $0 < \alpha_j < C$ 的样本，求解得到多个 b 后求平均值，作为最终 b 的取值。

4.5 SVM 实践

4.5.1 R 语言实践

R 语言中的 e1071 包可以实现 SVM 算法，算法实现流程如下：

```
library(e1071)
data(iris)#加载iris数据集
attach(iris)
set.seed(1)
#将数据集分为两部分，训练集占2/3，测试集占1/3
index<-sample(1:nrow(iris),round(nrow(iris)*2/3))
train_data<-iris[index,]
test_data<-iris[-index,]
```

```
train_data_x<-train_data[,1:4]
train_data_y<-train_data[,5]
test_data_x<-test_data[,1:4]
test_data_y<-test_data[,5]
#构建SVM模型，type为C-classification，核函数为radial
svmmodel<-svm(train_data_x,train_data_y,type='C-classification',kernel = 'radial')
#检验模型在训练集上的拟合效果
train_data_pred<-predict(object=svmmodel,train_data_x)
train_data_result<-table(train_data_pred,train_data_y)
train_data_result
train_data_accuracy<-sum(diag(train_data_result))/sum(train_data_result)
train_data_accuracy
#评估模型的预测性能
test_data_pred<-predict(object=svmmodel,test_data_x)
test_data_result<-table(test_data_pred,test_data_y)
test_data_result
test_data_accuracy<-sum(diag(test_data_result))/sum(test_data_result)
test_data_accuracy
```

4.5.2 Python 语言实践

本节将基于莺尾花数据集，采用 sklearn.svm 中的 SVC 函数训练一个 SVM 模型, 算法实现流程如下：

```
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
iris = datasets.load_iris()#导入样本集
x = iris.data[:,0:4]#截取前四列作为特征
y = iris.target#取出标签
#调用MinMaxScaler函数将数据集映射到0-1范围内，避免特征取值相差较大影响模型
ScalerModel = MinMaxScaler(feature_range=(0,1))
x = ScalerModel.fit_transform(x)
print(x)
#调用train_test_split函数将原数据集划分为训练集和测试集，训练集占2/3，测试集占1/3
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=1/3,
random_state=0, stratify=y)
#构建非线性SVM模型，核函数选择高斯核，参数C我表示正则化参数，gamma为核函数的参数
model = svm.SVC(C=0.8, kernel='rbf', gamma=20, decision_function_shape='ovr')
model.fit(x_train, y_train)#拟合样本数据集
print('accuracy', model.score(x_train, y_train))#输出拟合评分
y_predict = model.predict(x_test)#将模型作用于测试集
ResultReport = classification_report(y_test, y_predict)
print(ResultReport)#输出评估结果
```

总结

支持向量机一直是机器学习领域的研究热点，本章分别介绍了线性可分 SVM、线性 SVM、非线性 SVM 处理分类问题以及支持向量回归处理回归问题的相关理论与方法。支持向量机以间隔为基本出发点，通常采用凸优化理论求解，借助 SMO 算法实现快速计算，利用核函数解决线性不可分问题，使得该方法较为广泛地适用于实际生产。

除本章介绍的几种基本方法之外，还有较多的扩展模型，如基于粒度划分方法的粒度支持向量机（Granular Support Vector Machines, GSVM）[59,60]；结合模糊数学与支持向量机的模糊支持向量机（Fuzzy Support Vector Machines，FSVM）[61]，此类方法一定程度上可以克服噪声点对算法的影响；基于排序学习的排序支持向量机（Ranking Support Vector machines, RSVM）[62]；孪生支持向量机（Twin Support Vector Machines，TWSVM）[63–65]，该方法提升了经典 SVM 的泛化性能以及计算效率，对大规模数据集具有较好的处理效果。

支持向量机相关算法库可参考：LIBSVM、LIBLINEAR、e1071 等。

4.6 习题

1、请简要分析优化问题（4.2.18）转化成（4.3.2）中定义的损失函数的思路。

2、支持向量回归使用的目标函数是如何得到的？请给出简要分析思路。

3、假设样本集 $\mathbf{X}$ 线性可分，采用线性可分 SVM 得到划分超平面 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}+b=0$，其中 $\boldsymbol{w}=(2,1,-1)^{\mathrm{T}}$，$b=2$。试求出间隔并判断下面三个样本是否为支持向量。

（1）$\boldsymbol{X}_1=(3,1,-6)^{\mathrm{T}}$

（2）$\boldsymbol{X}_2=(2,-2,1)^{\mathrm{T}}$

（3）$\boldsymbol{X}_3=(-2,3,2)^{\mathrm{T}}$

4、对于同一样本集 $\mathbf{X}$，通过训练得到 SVM 和 Logistic 回归两个分类器。此时，若在训练集中新加入一个远离 SVM 决策边界的样本点，重新训练得到的两个新分类器是否发生变化？请用数学语言解释原因。

5、自选 UCI 数据集（https://archive.ics.uci.edu/ml/datasets.php），采用不同的核函数训练 SVM，对得到的模型以及实验结果进行对比。

6、自选 UCI 数据集（https://archive.ics.uci.edu/ml/datasets.php），训练一个支持向量回归模型。

第五章　决策树

相较于前两种方法，决策树通过树状结构进行决策，每个节点表示对一个特征的测试，每个分支代表一个测试结果，每个叶节点存储一个输出值。决策树算法可以处理分类特征和数值特征，不需要对数据进行特殊的缩放，适用于需要可解释性的情况。

5.1　简介

决策树方法最早产生于 20 世纪 60 年代，其中 CART 算法是决策树最经典和最主要的算法。CART 算法（Classification and Regression Tree）是 Breiman 等 [67] 在 1984 年提出来的一种非参数方法，它可以用于解决分类问题（预测定性变量，或者说当因变量是离散变量时），又可以用于回归问题（预测定量变量，或者说当因变量是连续变量时），分别称为分类树（Classification tree）和回归树（Regression tree）。CART 算法的基本思想是一种二分递归分割方法，在计算过程中充分利用二叉树，在一定的分割规则下将当前样本集分割为两个子样本集，使得生成的决策树的每个非叶子节点都有两个分裂，这个过程又在子样本集上重复进行，直至无法再分成叶子节点为止。在本章中，我们介绍的内容主要包括：决策树的基本概念、回归树和分类树的建模过程、防止决策树过拟合的方法及实施决策树相关的 Python 和 R 语言程序。

5.2　决策树的基本原理

决策树(decision tree) 是一种常见的机器学习方法，决策树模型呈树形结构，采用自顶向下递归的方法。决策树包含三种节点，分别是：根节点、内部节点和叶子节点。刚开始构建模型时，全部样本组成的节点，没有入边，只有出边，称为根节点(root node)；不再继续分裂的节点称为树的叶子节点（leaf node），没有出边，只有入边，叶子节点的个数决定了决策树的规模和复杂程度；根节点和叶子节点之外的节点都称作内部节点（internal node），内部节点既有入边，又有出边。

决策树模型结构如图5.1所示，一棵决策树一般包含一个根节点（黄色圆形图标）、若干个内部节点（蓝色圆形图标）和若干个叶子节点（方形图标）。叶子节点对应于

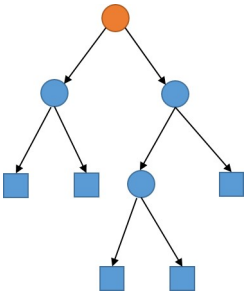


图 5.1 决策树模型结构

决策结果，每个内部节点对应于一个自变量，每个内部节点包含的样本集合根据自变量测试的结果被划分到它的子节点中，根节点包含全部训练样本。根节点在一定的分割规则下被分割成两个子节点，这个过程在子节点上重复进行，直至无法再分为叶子节点为止。从根节点到每个叶子节点的路径对应了一个判定测试序列。

决策树算法遵循自顶向下、分而治之的策略，**具体步骤**为：

（1）选择最好的自变量作为测试自变量并创建树的根节点；

（2）为测试自变量每个可能的取值产生一个分支；

（3）将训练样本划分到适当的分支形成子节点；

（4）对每个子节点，重复上面的过程，直到所有的节点都是叶子节点。

若考虑自变量只有两维的情况，将决策树分类过程在二维的坐标轴中画出，如图5.2所示。则可以发现，决策树实际上就是对自变量空间的划分，且划分区域的数目就是叶节点的数目，如图5.2中的红线和绿线。

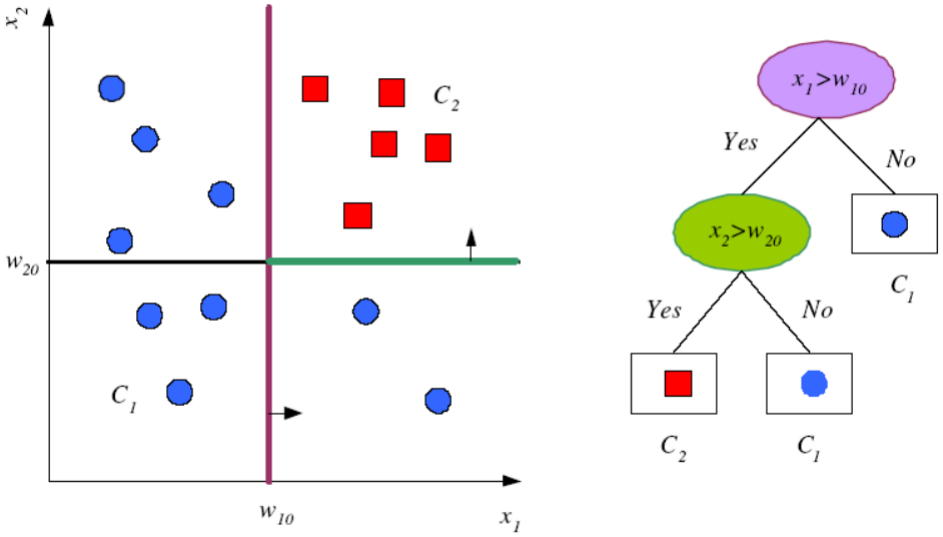


图 5.2 二维坐标轴中的决策树分类过程

决策树的建立过程可以概括为以下两个关键环节：

（1）将自变量空间（即 $\boldsymbol{X}=(X_1,X_2,...,X_p)^{\mathrm{T}}$ 的所有可能取值构成的集合）分割成 T 个互不重叠的叶子节点 $R_1,...,R_T$；

（2）对落入节点 R_t 的每个观测点，其预测值为 R_t 节点中训练集的因变量值的众数（分类树），或者平均数（回归树）。

决策树生长是一个递归的过程，因此在算法中需要包含跳出递归的条件。直观的看，决策树停止生长的条件有以下三种情况，满足以下三个条件之一就能使决策树停止生长：

（1）当前节点包含的样本全部属于同一类别；

（2）当前自变量集为空，或所有样本在所有自变量上取值相同；

（3）当前节点包含的样本集合为空。

决策树算法有以下几条**优点**：

（1）计算量相对小，训练速度快；

（2）易理解、解释性强；

（3）不需要任何先验假设；

（4）可以处理连续变量和离散变量（性别）；

（5）可以处理缺失值和具有尺度不变性，即对自变量做一个单调变换，不改变树的生成结果。

决策树算法包括如下**缺点**：

（1）决策树算法可以创建很复杂的树，但容易导致过拟合，可以通过**剪枝**等手段缓解；

（2）决策树算法训练结果方差大、不稳定，数据很小的扰动可能得到完全不同的分裂结果，有可能是完全不同的决策树，这个缺点可以通过**集成学习**来解决。

过拟合（Overfitting）是指机器学习模型在训练数据上表现得过于优越，以至于在未见过的新数据上表现不佳的现象。简而言之，过拟合发生时，模型过度适应了训练数据的噪声和细节，而没有正确地捕捉到数据中的真实模式和普遍规律。

具体而言，过拟合通常表现为以下一些特征：

(1) 训练数据拟合良好：模型在训练数据上表现良好，准确度高，损失函数低；

(2) 测试数据表现差：在未见过的测试数据上，模型的性能较差，预测结果不够准确；

(3) 模型过于复杂：模型可能过度灵活，以至于能够适应训练数据中的每一个细节，包括噪声和异常值；

(4) 泛化能力下降：模型在新数据上的泛化能力减弱，不能很好地适应不同的数据分布。

防止过拟合也有一些常用的方法，比如：

(1) 训练数据增强：增加训练数据量，有助于提高模型的泛化能力；

(2) 交叉验证：使用交叉验证来评估模型在不同数据集上的性能，有助于发现过拟合问题;

(3) 正则化：引入正则化项，如 L1 正则化或 L2 正则化，以限制模型参数的大小;

(4) 特征选择：选择最重要的特征，避免使用过多不相关或冗余的特征;

(5) 模型简化：选择较简单的模型结构，避免使用过于复杂的模型;

(6) 早停：在训练过程中监测模型性能，在验证数据上性能不再提升时停止训练，防止过拟合。

过拟合是机器学习中需要注意的重要问题，合适的防范手段能够提高模型的泛化性能，使其更好地适应新的未见数据。

5.3 分类树与回归树

对于决策树而言，根据预测变量的类型不同，可分为不同的决策树。若目标变量是离散的，则为分类树，若目标变量是连续的，则为回归树。

5.3.1 分类树

我们首先介绍分类树。当因变量为定性变量即离散型变量时，可建立分类树模型。分类树是一种特殊的分类模型，是一种直接以树的形式表征的非循环图。它的建模过程就是自动选择分裂变量，以及根据这个变量进行分裂的条件。

假设数据 $X_i = (X_{i1}, \ldots, X_{ip})^{\mathrm{T}}$ 包含 p 个输入变量和一个离散型的因变量 $Y \in (1, , ..., K)$，样本量为 n。现在我们想把数据分成 T 个区域（或称为节点），第 t 个节点 R_t 的样本量为 $n_t (t = 1, , ..., T)$。

则建立分类树的过程可以用如下**算法**表示：

1、**分支**。采用递归二叉分裂法在训练集中生成一棵分类树。递归二叉分裂法是指是从树顶端开始依次分裂自变量空间，每个分裂点都产生两个新的分裂，并且每次分裂选取最优分裂方案。最优的分裂方案是使得 T 个节点的不纯度减少到最小，衡量节点不纯度的指标在第三节详细介绍。

2、**剪枝**。对生成的回归树进行剪枝，得到一系列最优子树，剪枝在第四节详细介绍。

3、**预测**。令 $\widehat{P}_{tk} = \dfrac{1}{n_t} \sum_{i \in R_t} I(Y_i = k)$ 表示在叶子节点 t 中第 k 类样本点的比例，则预测叶子节点 t 的类别为：

$$\widehat{t}_m = \mathrm{argmax}_k \widehat{P}_{tk}$$

即叶子节点 t 中类别最多的一类。

5.3.2 回归树

当因变量为连续型变量时，可建立回归树模型。回归树和分类树的思想类似，首先是在分支准则上有差异，即衡量节点不纯度的指标不一样，其次是预测准则上存在差异，对于分类树，将落在该叶节点的观测点的最大比例类别作为该叶节点预测值，而对于回归树，则是将落在该叶节点的观测点的平均值作为该叶节点预测值。

回归树的数据结构与分类树类似，唯一区别是回归树解决的是连续型的因变量 Y，并且回归树算法过程跟分类树类似：

1、**分支**。采用递归二叉分裂法在训练集中生成一棵回归树。最优的分裂方案是使得 T 个节点的不纯度减少到最小，其中，衡量节点不纯度的指标为样本均方误差。

2、**剪枝**。对生成的分类树进行剪枝，得到一系列最优子树，剪枝在第四节详细介绍。

3、**预测**。节点得到后，可以确定某一给定预测数据所属的节点，回归树用这一节点的训练集的因变量的平均值作为预测值：

$$\widehat{Y}_t = \frac{1}{n_t} \sum_{i \in R_t} Y_i.$$

5.4 分支条件

我们现在再来讨论该如何构建节点，即对决策树进行分支。理论上，我们可以将节点所对应的区域形状作任意分割，但出于模型的简化和可解释性考虑，一般只将区域划分为高维矩形。不过，若要将自变量空间划分为矩形区域的所有可能性都进行考虑，这在计算上是不可行的。所以，我们对区域的划分一般采用一种自上而下（top-down）、贪婪（greedy）的方法：递归二叉分裂（recursive binary splitting）。自上而下指的是它从树顶端开始依次分裂自变量空间，每个分裂点都产生两个新的分裂。贪婪指在建立树的每一步中，最优分裂确定仅限于某一步进程，而不是针对全局，选择那些能够在未来进程中构建出更好的树的分裂点。对每一个节点重复以上过程，寻找继续分割数据集的最优自变量和最优分裂点。此时被分割的不再是整个自变量空间，而是之前确定的两个区域之一。这一过程不断持续，直到符合某个分裂准则再停止，譬如，当叶节点包含的观测值个数低于某个最小值时，分裂停止。

如何确定最优的分裂方案？我们基于不纯度的减少来作为分裂准则，即通过最小化节点不纯度来确定最优分裂变量和最优分裂点。对于分类树和回归树有不同的衡量节点不纯度的指标，我们将在后面的分类树和回归树部分进行具体的描述。

决策树算法的关键是划分自变量的选择，决策树自变量划分方法很多，比较经典有：适用于分类树的信息增益、增益率和基尼指数，以及适用于回归树的均方误差。

5.4.1 信息熵

在介绍信息增益之前，我们先引入信息量和信息熵的概念。

假设随机变量 $\boldsymbol{X}$ 的概率分布为 $\Pr(\boldsymbol{x})$，则任意一事件 $\boldsymbol{X}=\boldsymbol{x}$ 的信息量可以表示为 $h(\boldsymbol{x})=-\log_2\Pr(\boldsymbol{X}=\boldsymbol{x})$，其中 $\Pr(\boldsymbol{X}=\boldsymbol{x})$ 表示该事件发生的概率。某事件发生的概率越小，该事件的信息量越大。

信息熵是度量样本集合纯度最常用的一种指标，也是一种不确定性的度量。1948 年，香农（Shannon）[68] 在他著名的《通信的数学原理》论文中指出："信息是用来消除随机不确定性的东西"，并提出了"信息熵"的概念（借用了热力学中熵的概念），来解决信息的度量问题。

对于随机变量 $\boldsymbol{X}$ 而言，信息熵可表示为：$H(\boldsymbol{X})=-\sum\limits_{\boldsymbol{X}_i\in\mathcal{X}}\Pr(\boldsymbol{X}=\boldsymbol{X}_i)\ln\Pr(\boldsymbol{X}=\boldsymbol{X}_i)$，其中 $\mathcal{X}$ 表示所有可测事件的集合。特殊的，若假设两点分布中某一点发生的概率是 θ，则其信息熵：

$$H(\boldsymbol{X})=-\sum_{\boldsymbol{X}_i\in\mathcal{X}}\Pr(\boldsymbol{X}=\boldsymbol{X}_i)\ln\Pr(\boldsymbol{X}=\boldsymbol{X}_i)=-\theta\ln\theta-(1-\theta)\ln(1-\theta)$$

若信息熵退化为定值则熵最小为 0；若随机变量是连续型随机变量且取值在有限区间内（即随机变量密度函数在有限区间内大于零），则均匀分布的熵最大；随机变量是连续型随机变量且取值在无限区间内，通过拉格朗日乘子法可以推导正态分布的熵最大。

下面考虑响应变量是离散随机变量的情况介绍信息熵。即 $Y\in\{1,\cdots,K\}$ 的 K 分类问题。

对于离散响应变量集合 $\boldsymbol{Y}=(Y_1,\cdots,Y_n)^{\mathrm{T}}$ 来说，当前离散响应变量集合中第 k 类样本所占的比例为 $\widehat{P}_k=\dfrac{1}{n}\sum\limits_{i=1}^{n}I(Y_i=k)$，则信息熵的计算公式为：

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\widehat{P}_k\cdot\log_2\widehat{P}_k \tag{5.4.1}$$

信息熵越大，意味着不确定性越大；信息熵越小，则不确定性越小。例如，对于一盏灯有两种可能的状态：开和关，假设样本集合中一半是开的样本，一半是关的样本，那么信息熵为：

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\widehat{P}_k\cdot\log_2\widehat{P}_k=-(\frac{1}{2}\cdot\log_2\frac{1}{2}+\frac{1}{2}\cdot\log_2\frac{1}{2})=1.$$

在此情况下，信息熵很大，说明灯的两种状态不确定性很大。

假如这盏灯出现故障，无法正常打开，那么此时信息熵为：

$$H(\boldsymbol{Y}) = -\sum_{k=1}^{K} \widehat{P}_k \log_2 \widehat{P}_k = -(\frac{1}{1} \cdot \log_2 \frac{1}{1}) = 0.$$

在此情况下，灯的状态是确定的，因此信息熵很小。

5.4.2 信息增益

有了信息熵的概念，我们引入信息增益，信息增益是针对某一种自变量而言的，假设为 X（一维）。

假设使用自变量 X 的样本集 $\boldsymbol{X} = (X_1, \cdots, X_n)^{\mathrm{T}}$ 对分类树划分出 M 个区域 $R_1 \cdots, R_M$（在二叉分裂时，$M = 2$），其中叶子节点 R_m 中含有样本数为 n_m，简记为 $|R_m| = n_m$。假设节点 R_m 中类 k 的观测比例为：$\widehat{P}_{mk} = \dfrac{1}{n_m} \sum\limits_{i \in R_m} I(Y_i = k)$。

假设自变量是离散变量，M 个节点是由自变量 X 的 M 个可能的取值划分出来的。若自变量是连续变量，我们可以从连续变量取值范围上取几个点对样本进行划分，此时被划分的样本集可等价理解成离散变量样本。

接下来，计算出用离散变量样本集 $\boldsymbol{X}$ 对样本集 $\boldsymbol{Y}$ 进行再次划分所获得的信息增益，假设 $\boldsymbol{X}$ 取之于集合 $\{1, \cdots, M\}$，$\boldsymbol{Y}$ 取之于集合 $\{1, \cdots, K\}$。

信息增益（information gain）表示自变量 X 使目标不确定性减少的程度。信息增益计算公式如下：

$$\triangle = H(\boldsymbol{Y}) - \sum_{m=1}^{M} \frac{n_m}{n} H(\boldsymbol{Y} \mid R_m) \tag{5.4.2}$$

其中 $H(\boldsymbol{Y})$ 此时称为父节点的信息熵以及

$$H(\boldsymbol{Y} \mid R_m) = -\sum_{k=1}^{K} \widehat{P}_{mk} \log_2 \widehat{P}_{mk} \tag{5.4.3}$$

信息增益是自变量划分前的信息熵与自变量划分后加权信息熵的差值，即**信息增益 = 信息熵 - 条件熵**。

信息增益越大，则意味着使用自变量来进行划分所获得的“纯度提升”越大。下面以打网球实例来演示信息增益的具体计算过程，该实例是根据天气状况来决定是否打网球，影响打网球的自变量包括天气、温度、是否有风，数据集如表5.1所示，数据集中一共包括 10 个样本，构建决策树算法对是否适合打网球进行预测。首先计算划分前根节点的信息熵，根节点是否打网球两类的样本所占比例分别为 $\dfrac{6}{10}$ 和 $\dfrac{4}{10}$，

信息熵计算公式为：

$$H(\text{parent}) = -\sum_{k=1}^{K} \widehat{P}_k \cdot \log_2 \widehat{P}_k = -(\frac{6}{10} \cdot \log_2 \frac{6}{10} + \frac{4}{10} \cdot \log_2 \frac{4}{10}) = 0.971.$$

假设因变量 Y “是否打球” 分别用 1 和 2 表示，Y=1 表示打球，Y=2 表示不打球。

表 5.1 打网球数据集

序号	weather	temperature	windy	play_tennis
1	晴	热	否	否
2	晴	热	是	否
3	阴	热	否	是
4	雨	温	否	是
5	雨	凉	是	否
6	雨	凉	否	是
7	阴	凉	是	是
8	晴	温	否	否
9	晴	凉	否	是
10	雨	温	否	是

在此基础上，首先计算天气的信息增益，对于天气，自变量取值包括：晴、阴、雨，分别对应节点 R_1、R_2、R_3。其中对于 “晴” 取值，共有 4 个样本，即 $n_1 = 4$，打球数量占比为 $\widehat{P}_{11} = \frac{1}{n_1} \sum_{i \in R_1} I(Y_i = 1) = \frac{1}{4}$，不打球数量占比为 $\widehat{P}_{12} = \frac{1}{n_1} \sum_{i \in R_1} I(Y_i = 2) = \frac{3}{4}$，即天气的其他取值采用同样的计算方式。首先根据公式(5.4.3)，用 1、2、3 分别代表晴天、阴天和雨天，分别计算不同自变量取值的信息熵：

$$\begin{aligned}
H(\boldsymbol{Y} \mid R_1) &= -(\frac{1}{4}\log_2\frac{1}{4} + \frac{3}{4}\log_2\frac{3}{4}) = 0.811, \\
H(\boldsymbol{Y} \mid R_2) &= -(\frac{2}{2}\log_2\frac{2}{2}) = 0, \\
H(\boldsymbol{Y} \mid R_3) &= -(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}) = 0.811.
\end{aligned}$$

信息增益：

$$\begin{aligned}
\triangle_{\text{weather}} &= H(\text{parent}) - \sum_{m=1}^{M} \frac{n_m}{n} H(\boldsymbol{Y} \mid R_m), \\
&= 0.971 - ((4/10) * 0.811 + (2/10) * 0 + (4/10) * 0.811) = 0.971 - 0.649 = 0.322.
\end{aligned}$$

同样可以求出温度和是否有风两个自变量的信息增益：

$$\triangle_{\text{temperature}} = 0.971 - ((3/10) * 0.918 + (4/10) * 0.811 + (3/10) * 0.918) = 0.971 - 0.875 = 0.05$$

$$\triangle_{\text{windy}} = 0.971 - ((3/10) * 0.918 + (7/10) * 0.863) = 0.971 - 0.880 = 0.09.$$

由信息增益的计算结果可以看出，天气的信息增益最大，因此首先选择天气作为划分自变量。

5.4.3 增益率

不同的离散自变量的离散值的取值个数是不同的，即自变量不同 M 不同。如果选取的某一自变量取值个数较多，即 M 较大，那么这一划分的信息增益越大。特别的，对于某一自变量，如果该自变量的每个取值内只含一个样本点，那么选取该自变量划分后的加权信息熵是 0，进而此划分的信息增益最大。因此，在以信息增益作为划分训练数据集的特征时，我们会偏向于选择取值较多的自变量。故而，我们引入信息增益率是为了避免自变量值个数对信息增益的影响。

信息增益率（information gain ratio）在信息增益的基础上增加了惩罚项，惩罚项是特征的固有值，是避免上述情况而设计的。信息增益率的定义如下：

$$\text{GainR} = \frac{\triangle}{\text{IV}},$$

其中

$$\text{IV} = -\sum_{m=1}^{M} \frac{n_m}{n} \log_2 \frac{n_m}{n}$$

一般的话，某个自变量的取值数目越多，则 IV 值越大。

以上面打网球的例子，计算各自变量增益率。首先计算各自变量的 IV 值：

$$\text{IV}_{\text{weather}} = -\frac{4}{10} * \log_2 \frac{4}{10} - \frac{2}{10} * \log_2 \frac{2}{10} - \frac{4}{10} * \log_2 \frac{4}{10} = 1.52,$$

$$\text{IV}_{\text{temperature}} = -\frac{3}{10} * \log_2 \frac{3}{10} - \frac{3}{10} * \log_2 \frac{3}{10} - \frac{4}{10} * \log_2 \frac{4}{10} = 1.57,$$

$$\text{IV}_{\text{windy}} = -\frac{3}{10} * \log_2 \frac{3}{10} - \frac{7}{10} * \log_2 \frac{7}{10} = 0.88.$$

得到三个自变量的 IV 值，由结果可知，天气和温度的取值个数比是否有风多，则天气和温度的 IV 值高。接下来计算各个自变量的信息增益率：

$$\text{GainR}_{\text{weather}} = \frac{\triangle_{\text{weather}}}{\text{IV}_{\text{weather}}} = \frac{0.322}{1.52} = 0.212$$

$$\text{GainR}_{\text{temperature}} = \frac{\triangle_{\text{temperature}}}{\text{IV}_{\text{temperature}}} = \frac{0.05}{1.57} = 0.032$$

$$\text{GainR}_{\text{windy}} = \frac{\triangle_{\text{windy}}}{\text{IV}_{\text{windy}}} = \frac{0.09}{0.88} = 0.102$$

求得三个自变量的增益率，发现天气的增益率较高，因此首先选择天气作为划分自变量。

5.4.4 基尼指数

对于分类因变量样本集合 $\boldsymbol{Y}$ 来说，假定当前样本集合 $\boldsymbol{Y}$ 中第 k 类样本所占的比例为 $\widehat{P}_k(k=1,...,K)$，数据集的纯度可用基尼值（Gini）来度量：

$$\text{Gini}(\boldsymbol{Y}) = 1 - \sum_{k=1}^{K} \widehat{P}_k^2. \tag{5.4.4}$$

$\text{Gini}(\boldsymbol{Y})$ 反映了从数据集 $\boldsymbol{Y}$ 中随机抽取两个样本，其类别标记不一致的概率。$\text{Gini}(\boldsymbol{Y})$ 越小，则数据集 $\boldsymbol{Y}$ 的纯度越高。基于某个离散自变量划分出的第 m 区域上的基尼值表示为 $\text{Gini}(\boldsymbol{Y} \mid R_m) = 1 - \sum_{k=1}^{K} \widehat{P}_{mk}^2$.

基于分类因变量的基尼指数（Gini Index）定义为：

$$\text{GiniI} = \sum_{m=1}^{M} \frac{n_m}{n} \text{Gini}(\boldsymbol{Y} \mid R_m) \tag{5.4.5}$$

基尼指数越小，则不纯度越低，自变量越好。

5.4.5 分类误差

令节点 R_m 中类 k 的观测比例为 $\widehat{P}_{mk} = \dfrac{1}{n_m} \sum_{i \in R_m} I(Y_i = k)$，并且我们得到节点 R_m 中的所有样本的预测类别为：

$$\widehat{k}_m = \text{argmax}_k \widehat{P}_{mk}$$

它是节点 R_m 上样本数最多的类。节点 R_m 上的分类误差表示为：

$$\text{CE}(\boldsymbol{Y} \mid R_m) = \frac{1}{n_m} \sum_{i \in R_m} I\left(Y_i \neq \widehat{k}_m\right) = 1 - \widehat{P}_{m\widehat{k}_m}$$

最终也可以采用加权的思想，得到基于 M 个节点的分类误差（CE）：

$$\text{CE} = \sum_{m=1}^{M} \frac{n_m}{n} \text{CE}(\boldsymbol{Y} \mid R_m) \tag{5.4.6}$$

5.4.6 均方误差

如果分裂变量 X_j 是连续的自变量以及因变量 Y 也是连续变量，我们使用分裂点 t 去分裂自变量 X_j 的样本集 $\boldsymbol{X}_j = (X_{1j}, \cdots, X_{nj})^{\mathrm{T}}$。若考虑二分裂，使用分裂变量 X_j 和 t 可以定义两个子区域：

$$R_1(j,t) = \{i \mid X_{ij} < t\} \quad \text{和} \quad R_2(j,t) = \{i \mid X_{ij} \geq t\} \tag{5.4.7}$$

其中 X_{ij} 表示样本点 $\boldsymbol{X}_i$ 的第 j 个特征。

对应分裂变量 X_j 和分裂点 t 的均方误差定义为：

$$\mathrm{MSE}(j,t) = \frac{1}{n_1} \sum_{i \in R_1(j,t)} (Y_i - \widehat{c}_1)^2 + \frac{1}{n_2} \sum_{i' \in R_2(j,t)} (Y_{i'} - \widehat{c}_2)^2 \tag{5.4.8}$$

其中 $n_1 = |R_1(j,t)|$，$n_2 = |R_2(j,t)|$ 分别表示落入第一个区域和第二个区域的训练样本的数量，且

$$\widehat{c}_1 = \frac{1}{n_1} \sum_{i \in R_1(j,t)} Y_i \quad \text{和} \quad \widehat{c}_2 = \frac{1}{n_2} \sum_{i' \in R_2(j,t)} Y_{i'},$$

5.4.7 算法总结

下面介绍三种决策树算法，其中 ID3 与 C4.5 算法主要用于分类树，而 CART 算法既可以用于分类树也可以是回归树。对决策树 $\mathcal{T}$ 的优劣衡量我们可以用以下代价函数。

代价函数为 $C(\mathcal{T}) = \sum_{t=1}^{T} n_t H(t)$，$T$ 表示所有叶子节点的个数，n_t 是每个叶子节点中样本的个数，在这个公式中相当于权重，因为各叶节点包含的样本数目不同，可使用样本数加权求熵和。$H(t)$ 是每个叶子节点的损失度量准则，上面章节中已介绍。代价函数越小越好。对所有叶节点的熵求和，该值越小说明对样本的分类或者回归越精确。

ID3 算法（**信息增益**）是一种贪心算法，构造的决策树用于分类。于 1986 年由 Quinlan [69] 提出的 ID3 决策树学习算法就以信息增益为准则来选择划分自变量。ID3 算法起源于概念学习系统（CLS），以信息熵的下降速度为选取自变量的标准，即在每个节点选取还尚未被用来划分的具有最高信息增益的自变量作为划分标准，然后继续这个过程，直到生成的决策树能完美分类训练样例。

上述信息增益的定义是基于离散自变量，对于连续自变量，我们可以采用 (5.4.7) 给出的策略，将连续自变量进行离散化，即使用分裂点对连续自变量进行分裂产生区域（即节点）。此时使用的信息增益准则，不仅要选择自变量还要估计出分裂点。

C4.5 算法（**信息增益率**）采用信息增益率来选择最优划分自变量。不是直接选择信息增益率最大的候选划分自变量，而是先从候选划分自变量中找出信息增益高于平均水平的自变量，再从中选择信息增益率最高的。采用了信息增益率来选择特征，以减少信息增益容易选择分类取值多的自变量的问题。

同样的，对于连续自变量，我们可以采用（5.4.7）给出的策略使用分裂点对连续自变量进行分裂，然后使用上述的信息增益率准则，选择自变量以及最优分裂点。

CART 算法（**回归：均方误差；分类：基尼指数**）包括 CART 分类树和 CART 回归树。

CART 回归树

回归树是决策树模型的一种，专门针对因变量是连续型随机变量或向量，自变量是实值随机向量的树的模型。为了便于理解，我们首先考虑一维连续随机变量 Y，二元自变量 X_1,X_2 的回归问题，并以图5.3为例来阐述基于树的回归方法。

首先，按 $X_2 \geq t_1$ 和 $X_2 < t_1$ 把整个样本空间 N_0 划分为两个子区域 R_1 和 N_1，为了得到更加精确的预测，我们可以继续对 N_1 子区域进行划分，得到两个子区域 R_2 和 R_3 。接下来我们形象地定义一些树中的概念。为了记号方便，我们称图5.3用于分裂的变量 X_2 和 X_1 为分裂变量，t_1 和 t_2 为分裂点，N_0 是根节点（包含所有样本），N_1 是中间节点（也是 R_2 和 R_3 的父节点），R_1，R_2 和 R_3 为叶子节点。回归树需要解决以下 4 个关键问题：

1、如何选取分裂自变量？图5.3第一次选择的分裂特征为 X_2，第二次选择的分裂特征为 X_1 ；

2、如何确定分裂点的值？如5.3中的 t_1, t_2；

3、如何确定停止分裂的准则，即为什么叶子节点 R_1, R_2, R_3 不再分裂？

由问题 1-3 可知，基于回归树模型的核心就是要依次找出分裂变量和分裂点，制定停止规则和对每一个叶子节点的样本进行预测。

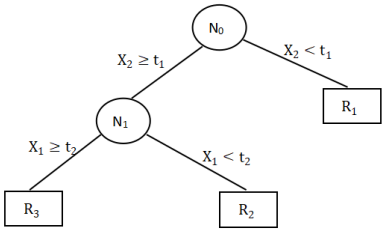


图 5.3 基于树模型定义的预测区域

接下来，我们将介绍如何逐步生成回归树。不失一般性，我们假设含有 n 个观测数据的数据集为：$N_0 = \{(\boldsymbol{X}_1, Y_1), \cdots\cdots, (\boldsymbol{X}_n, Y_n)\}$，其中，$\boldsymbol{X}_i = (X_{i1}, \cdots, X_{ip})^{\mathrm{T}}$ 是 p 维实值随机向量，Y_i 是一维实值随机变量。

具体的做法是：从所有数据出发，考虑分裂变量 X_j 和划分点 t 以及对应的划

分成的两个子区域 $R_1(j,t)$ 和 $R_2(j,t)$，根据表达式（5.4.8），通过最小化 $R_1(j,t)$ 和 $R_2(j,t)$ 的均方误差之和

$$\min_{j,t} \mathrm{MSE}(j,t)$$

得到所对应的划分变量 j 和划分点 t. 即通过搜索所有的自变量 X_j, $j=1,\cdots,p$, 以及对照分裂点 t 搜索所有的观测值 $\{X_{1j},\cdots,X_{nj}\}$，可以确定最好的 (j,t)，从而可以把整个样本分到两个区域。对每一个区域重复以上过程进行分裂。最后对所有的区域重复这一过程，直到达到最终的停止条件，关于停止条件及树修剪等方面详细知识可以参考 [70] [71] [67]。

CART 分类树类似回归树，我们只须把上式中的度量标准 $\mathrm{MSE}(j,t)$ 替换成基尼指数，序贯的选择最优变量去分裂样本，从而产生节点，最终生成分类树。

5.5 剪枝

决策树对训练数据有很好的分类和回归能力，虽然能在训练集中取得良好的预测效果，但对未知的测试数据未必有好的预测能力，即泛化能力弱，可能发生过拟合现象，导致在测试集上效果不佳。所以，为了防止决策树过度生长、出现过拟合现象，我们的解决方案是对决策树进行剪枝或构建随机森林。在此处我们主要介绍剪枝，随机森林在后续章节进行介绍。

剪枝是决策树算法缓解过拟合的一种重要方法。可通过剪枝在一定程度上避免因决策分支过多，以致于把训练集自身的一些特点当做所有数据都具有的一般性质而导致的过拟合。剪枝主要分为两种：预剪枝（pre-prune）和后剪枝（post-prune）。

5.5.1 预剪枝

预剪枝在决策树构造时就进行剪枝。方法是在构造的过程中对节点进行评估，包括以下几种策略。

（1）、如果对某个节点进行划分，在验证集中不能带来准确性的提升，那么对这个节点进行划分就没有意义，这时就会把当前节点作为叶节点，不对其进行划分。

（2）、或者更为直接，预先设置每一个节点所包含的最小样本数目，例如 10，若该节点总样本数小于 10 时，则不再分；

（3）、或者预先指定树的高度或者深度，例如树的最大深度为 4；或者指定节点的熵小于某个值时就不再继续划分。

预剪枝的优点是可以降低训练模型的资源开销，缺点是存在欠拟合风险。这是因为有些分支的当前划分虽然不能提升性能，但在其基础上进行的后续划分却有可能导致性能显著提高。预剪枝提前把这些分支剪掉了，可能带来欠拟合风险。

5.5.2 后剪枝

预剪枝方法在建树过程中要求每个节点的分裂使得不纯度下降超过一定阈值。这种方法具有一定的短视，因为很有可能某一节点分裂不纯度的下降没超过阈值，但是其在后续节点分裂时不纯度会下降很多，而预剪枝法则在前一节点就已经停止分裂了。

后剪枝在生成决策树之后再进行剪枝，通常会从决策树的叶节点开始，逐层向上对每个节点进行评估。如果剪掉这个节点子树，与保留该节点子树在分类准确性上差别不大，或者剪掉该节点子树，能在验证集中带来准确性的提升，那么就可以把该节点子树进行剪枝。

后剪枝的优点是比预剪枝保留了更多的分支，欠拟合风险小，泛化性能往往优于预剪枝决策树，缺点是模型训练时会占用较多的资源。

在实际应用中，更多的是使用后剪枝法，其中的代价复杂性剪枝（cost complexity pruning）是最常用的方法。这种方法是先让树尽情生长，得到 $\mathcal{T}_0$，然后再在 $\mathcal{T}_0$ 基础上进行修剪。设 T 表示子树 $\mathcal{T}$ 的叶节点数目，n_t 表示叶节点 t 的样本量，则代价复杂性的剪枝法的损失函数为：

$$\mathcal{C}_\alpha(\mathcal{T}) = \sum_{t=1}^{T} n_t H(t) + \alpha T \tag{5.5.1}$$

其中 $\sum_{t=1}^{T} n_t H(t)$ 是代价函数，α 是参数。$\mathcal{T}$ 是任意一棵子树，它通过对 $\mathcal{T}_0$ 进行剪枝得到，也就是减去 $\mathcal{T}_0$ 某个中间节点的所有子节点，使其成为 $\mathcal{T}$ 的叶节点。对于固定的 α，一定存在这样一棵子树 $\mathcal{T}_\alpha$ 使得损失函数 $\mathcal{C}_\alpha(\mathcal{T})$ 达到最小。

调整参数 α 控制着模型对数据的拟合与模型的复杂度（树的大小）之间的平衡。当 $\alpha = 0$ 时，最优子树 $\mathcal{T}_\alpha$ 等于原树 $\mathcal{T}_0$。随着 α 取值增大，损失函数对叶节点数目 T 惩罚增大，那么此时我们偏向于选择叶节点数目少一点的树。

后剪枝步骤：从 $\mathcal{T}_0$ 开始，自下而上地考虑每个内部节点 t ，考虑两种情况：

（1）、以 t 为根节点的子树 $\mathcal{T}_t$ ，其损失为：$\mathcal{C}_\alpha(\mathcal{T}_t) = \mathcal{C}(\mathcal{T}_t) + \alpha T_t$；

（2）、对 t 进行剪枝，即将 $\mathcal{T}_t$ 作为叶子节点，其损失为：$\mathcal{C}_\alpha(t) = \mathcal{C}(t) + \alpha$；

我们通过比较 $\mathcal{C}_\alpha(\mathcal{T}_t)$ 与 $\mathcal{C}_\alpha(t)$ 大小来判断是否要对 t 进行剪枝：

当 α 较小时，可以容忍模型有较高的复杂度，所以 $\mathcal{C}_\alpha(\mathcal{T}_t) < \mathcal{C}_\alpha(t)$ ，即这个时候不需要剪枝；

当 α 逐渐增大到某一阈值时，这个时候需要考虑在训练数据上 $\mathcal{T}_t$ 的损失增大，所以有 $\mathcal{C}_\alpha(\mathcal{T}_t) = \mathcal{C}_\alpha(t)$ ，即这个时候剪不剪枝都可以；

当 α 继续增大，以至于大于上述阈值时，有 $\mathcal{C}_\alpha(\mathcal{T}_t) > \mathcal{C}_\alpha(t)$ ，这个时候剪枝带来的收益大于作为一棵子树 $\mathcal{T}_t$ 所带来的收益，所以要剪枝。

从以上过程我们可以看出，对于树中的每个内部节点 t，都有一个特定的阈值 $g(t)$。当 $\alpha = g(t)$ 时，$C_\alpha(T_t) = C_\alpha(t)$。故而 $g(t)$ 可以决定是否需要对其进行剪枝，且该阈值等于

$$g(t) = \frac{C(t) - C(T_t)}{T_t - 1}.$$

上式 $g(t)$ 可由等式 $C_\alpha(T_t), C_\alpha(t)$ 解出。

1、因此，在生成子树 T_1 时，我们可以计算 T_0 的每个内部节点的阈值 $g(t)$，选择其中最小的记为 $g(t_1)$。当 α 大于该阈值 $g(t_1)$ 时，这意味我们要剪掉该节点 t_1 对应的子树 T_t 得到新的树 T_1 收益较大。当 $g(t_1) \leq \alpha < g(t_2)$ 时，T_1 使得损失函数最小。同理当 $g(t_2) < \alpha$ 时，剪掉节点 t_2 的子树，使 t_2 为叶节点，得到子树 T_2。同理子树 T_2 在区间 $g(t_2) \leq \alpha < g(t_3)$ 上使得损失函数最小。接着在 T_2 的基础上持续剪枝，就可以得到最终的子树序列。

2、交叉验证在生成子树序列 $T_1, T_2, T_3, \cdots$ 后，使用验证数据集，测试子树序列中每棵子树的损失，选择最小的子树作为剪枝后的决策树，这个时候也对应了一个 α_k 。

5.6 决策树实践

5.6.1 R 语言实践

R 语言中常用的决策树算法工具是 R 程序包 rpart（Recursive Partitioning and Regression Trees）里的 rpart 函数，本小节以鸢尾花数据集为例，用决策树进行分类。具体实现过程代码如下：

```
library(rpart)
library(rpart.plot)
#表达式子Species~.表示Species因变量，其余都是自变量
#决策树类型为“class”分类树；使用信息增益为划分标准
fit <- rpart(Species~., data=iris,method="class", parms=list(split="information"))
rpart.plot(fit,sub="Classification tree")
```

决策树输出结果：

5.6.2 Python 语言实践

决策树算法常用的 Python 工具是 scikit-learn（简称 sklearn），sklearn 是一个应用很广泛的 Python 机器学习库。sklearn 库中自带了决策树算法。

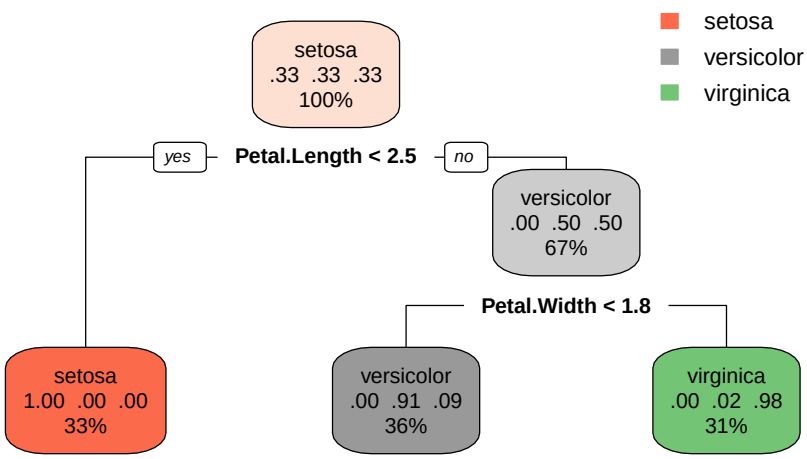


图 5.4 分类树树结构

sklearn 库中的 DecisionTreeClassifier 函数可以实现分类树，DecisionTreeClassifier 将两个数组作为输入，第一个数组是用于保存训练样本的自变量数组 X，第二个数组是用于保存训练样本的类标签的整数值数组 Y。

如以下程序示例，从 sklearn 模块中导入分类树的函数，对数组 X 和 Y 赋值后进行拟合：

```
from sklearn import tree
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = tree.DecisionTreeClassifier()#建立分类树
clf = clf.fit(X, Y)#基于数据进行分类树拟合
clf.predict([[2., 2.]])#对新输入的数据进行分类预测
clf.predict_proba([[2., 2.]])#预测信样本属于每个类的概率
```

使用 DecisionTreeRegressor 类，决策树也可以应用于回归问题。此时因变量 Y 具有浮点值而不是整数值。下面代码演示 DecisionTreeRegressor 类的使用：

```
from sklearn import tree
X = [[0, 0], [2, 2]]
Y = [0.5, 2.5]
clf = tree.DecisionTreeRegressor()#建立回归树
clf = clf.fit(X, Y)#基于数据进行回归树拟合
clf.predict([[1, 1]])#对新输入的数据进行回归预测
```

信息增益与增益率

本小节以前面介绍的预测是否打网球的实例，运用 Python 软件，演示信息增益和增益率的计算过程。

1、导入 math 库，自定义计算信息熵的函数并计算根节点的信息熵。

```
import math
def info(x,y):#定义计算特征熵的函数
    if x != y and x != 0:
    return -(x/y)*math.log2(x/y)-((y-x)/y)*math.log2((y-x)/y)#计算熵
    if x == y or x == 0:
    return 0#纯度最大，熵值为0
info_parent = info(6,10)#自变量划分前信息熵
print (info_parent)#计算根节点的信息熵
```

2、计算信息增益。首先计算天气（weather）、温度（temperature）以及是否有风自变量（windy）的信息增益。代码如下：

```
#分别计算出三种天气情况下的信息熵，再结合根节点的信息熵计算出该自变量的信息增益
weather_sunny_entropy = info(3,4)
weather_cloudy_entropy = info(2,2)
weather_rain_entropy = info(1,4)
weather_entropy = (4/10) * weather_sunny_entropy + (2/10) * weather_cloudy_entropy +
(4/10)  * weather_rain_entropy#计算天气特征信息熵
weather_info_gain = info_parent-weather_entropy#计算天气的信息增益
# 计算温度每种情况的熵
temperature_hot_entropy = info(1, 3)
temperature_warm_entropy = info(2, 3)
temperature_cold_entropy = info(2, 4)
#计算温度特征信息熵
temperature_entropy = (3/10) * temperature_hot_entropy + (3/10) *
temperature_warm_entropy + (4/10) * temperature_cold_entropy
temperature_info_gain = info_parent - temperature_entropy#计算温度信息增益
# 计算是否有风每种情况的熵
windy_yes_entropy = info(1,3)
windy_no_entropy = info(5,7)
#计算是否有风特征信息熵
windy_entropy = (3/10) * windy_yes_entropy + (7/10) * windy_no_entropy
windY_info_gain = info_parent - windy_entropy#计算是否有风信息增益
```

以是否有风自变量的信息熵与信息增益为例，将计算结果输出出来：

```
print ('windy_yes_entropy:', round(windy_yes_entropy, 4))
print ('windy_no_entropy:', round(windy_no_entropy, 4))
print('windy_entropy:', round(windy_entropy, 4))
print ('windY_info_gain:', round(windY_info_gain, 4))
```

3、接下来展示计算三种自变量的信息增益率。

```
#根据公式分别计算三种自变量对应的IV值：
iv_weather = -(4/10) * math.log2(4/10) - (2/10) * math.log2(2/10) - (4/10) *
```

```
math.log2(4/10)
iv_temperature = -(3/10) * math.log2(3/10) - (3/10) * math.log2(3/10) - (4/10) *
math.log2(4/10)
iv_windy = -(3/10) * math.log2(3/10) - (7/10) * math.log2(7/10)
#用各自变量的信息增益值除以对应的IV值得到三个自变量的信息增益率:
gain_ratio_weather = weather_info_gain / iv_weather
gain_ratio_temperature = temperature_info_gain / iv_temperature
gain_ratio_windy = windY_info_gain / iv_windy
print('gain_ratio_weather:', round(gain_ratio_weather, 4))
print('gain_ratio_temperature:', round(gain_ratio_temperature, 4))
print('gain_ratio_windy:', round(gain_ratio_windy, 4))
```

决策树算法

以下过程借助 sklearn 库完成，sklearn 库中自带了决策树算法，下面以打网球数据集，来演示决策树算法的应用。

1、导入 Pandas 库和 Sklearn 库，读入数据与数据预处理。

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics#导入相关函数
dataset=[ ['晴', '热', '否', '否'],['晴', '热', '是', '否'],['阴', '热', '否', '是'],
['雨', '温', '否', '是'],['雨', '凉爽', '否', '是'],['雨', '凉爽', '是', '否'],
['阴', '凉爽', '是', '是'],['晴', '温', '否', '否'],['晴', '凉爽', '否', '是'],
['雨', '温', '否', '是'] ]
#读入数据并命名
data = pd.DataFrame(dataset, columns=['weather', 'temperature', 'windy',
'play_tennis'])
#数据集中的四列数据都是字符串类型，因此字符串进行数值化，按类别进行赋值
labels = data['weather'].unique().tolist()
data['weather'] = data['weather'].apply(lambda n: labels. index(n))
labels = data['temperature'].unique().tolist()
data['temperature'] = data['temperature'].apply(lambda n: labels.index(n))
labels = data['windy'].unique().tolist()
data['windy'] = data['windy'].apply(lambda n: labels.index(n))
labels = data['play_tennis'].unique().tolist()
data['play_tennis'] = data['play_tennis'].apply(lambda n:labels.index(n))
```

2、决策树模型训练、预测与评估。首先对数据集进行拆分，前三列是特征自变量，最后一列是类别标签，分别记作 x 和 y。然后选择分类标准为基尼指数，对数据进行模型拟合。

```
x = data.drop(['play _ tennis'],axis=1)#所有样本特征
y = data['play_tennis']#类别标签
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(x,y)#模型训练
DecisionTreeClassifier(criterion='entropy')#输出结果
y_pred = clf.predict(x) #模型预测
print('Accuracy:%.4f'%(metrics.accuracy_score(y, y_pred)))#模型评估
```

3、绘制决策树。导入绘图和可视化树形结构的库和函数，绘制树形图，保存图像并展示。结果如图5.5所示。

```
from IPython.display import Image
from sklearn import tree
import pydotplus
dot_data = tree.export_graphviz(clf, out_file=None,feature_names=data.columns[0:3],
class_names=data.columns[3], filled=True, rounded=True, special_characters=True)
dot_data=dot_data.replace('\n','')
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png('example.png')#保存图像
Image(graph.create_png())
```

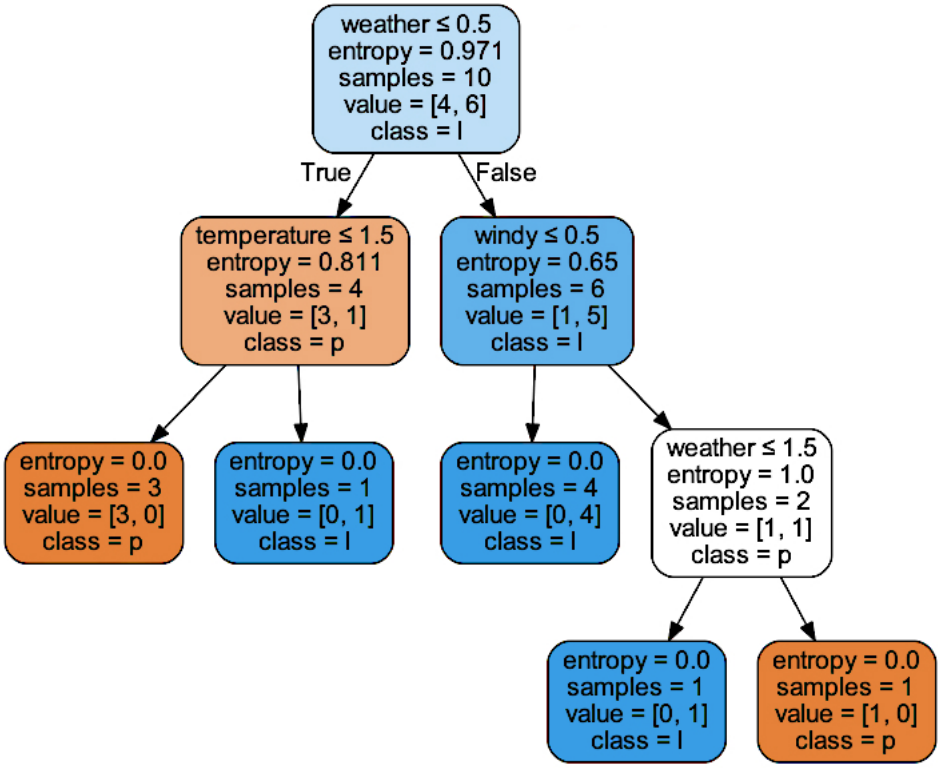


图 5.5 决策树树结构

总结

本章主要讨论了决策树中回归树和分类树的基本方法和算法，对于分类树主要采用投票准则来对新样本进行分类，而对于回归树则建议采用叶子节点里所有样本的样本均值来预测。本章介绍的基于树的方法具有以下优点：

（1）可以简单而自然的处理分类和有序变量；

（2）自动的逐步变量选择和降低复杂度；

（3）对任一单个有序变量的单调变换保持不变；

（4）对训练集中的异常点稳健；

（5）易于处理缺失值。

为了得到更加精确的预测，在叶子节点中样本量足够的情况下，也可以对叶子节点里的样本建立不同的预测模型而非简单的投票准则的分类树方法或者基于样本均值的回归树方法，不同的预测模型可以考虑线性或非线性模型等。其他关于基于回归树的集成算法包括 Bagged 树、Random Forests 和 Boosting 等集成算法和相关程序实现可以参考 Kuhn 和 Johnson 的书籍第八章（回归）和第十四章（分类）[71]。

5.7 习题

1、生成一模拟数据，来考察树方法在不同层次叶子节点的偏差。

2、Friedman [72] 介绍了如下的回归模型

$$Y = 10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \varepsilon$$

其中，$X_1, X_2, \ldots, X_5 \sim U(0,1)$，$\varepsilon \sim N(0, \sigma^2)$，我们也生成 $X_6, \ldots, X_{10}$ 5 个不相干的变量。

（1）设定样本量为 200，应用回归树方法拟合这一模拟数据；

（2）选取 $X_1, \ldots, X_5$ 中的一个进行单调变换，再次应用回归树模型拟合这一数据，与（1）中结果进行对比；

（3）将第一个样本扩大为原始值的 10 倍，应用回归树模型拟合这一数据，与（1）的结果进行对比。

（4）删除 $X_1, \ldots, X_5$ 中某一变量的部分观测值，然后应用回归树模型拟合这一数据，与（1）（2）的结果再次进行对比。

3、从鸢尾花数据集中任取 2 个自变量，应用分类树模型分析这一数据，并与书中例子进行对比。

4、思考一下对于回归树和分类树怎么评价变量的重要程度。

5、回归树和分类树都是对单一的变量进行划分，建立树模型，请考虑一种现代降维方法，先对自变量进行降维，然后再建立回归树，并应用习题 2 的模拟数据进行对比。

6、CART 分类树建模中，如果自变量是连续变量，如何结合基尼指数这一损失度量去选择最优变量，请论述。

第六章　保形预测

保形预测提供了一种量化机器学习模型预测的不确定性的方法。它不是像决策树、回归分析或支持向量机 (SVM) 那样的特定算法，而是一个可以与各种机器学习模型结合使用的框架。保形预测输出的不仅仅是预测值，还包括一个可信区间或概率置信度，表示预测的不确定性。

6.1　简介

在进行预测的时候，也许我们会思考这样的问题：我们的预测有多好？如果我们对一个新对象的类别进行预测，我们有多少信心认为我们的预测结果 $\widehat{Y}$ 确实等于这个未知的真实的类别 Y 呢？如果类别是一个数字，我们得到的 $\widehat{Y}$ 与真实的类别 Y 有多接近？在机器学习中，这些问题通常以过去的经验相当粗略地回答。

因此，在使用机器学习进行预测时，需要注意到预测值也是随机变量，存在不确定性。例如，使用一个股票自动交易系统中的机器学习方法预测股票价格。由于股票市场的高度不确定性，机器学习的点预测可能与实际值有很大不同。人工智能系统如果以高概率估计出覆盖目标真实值的范围，交易系统就可以计算出最好和最差的情况，并做出更明智的决定。所以，对实际值预测一个可信范围会更有意义。

保形预测（Conformal Prediction）是目前机器学习中热门的且非常灵活的预测技术。即当我们在处理回归或分类问题时，给定输入，保形预测可以输出一个预测区间或者一个预测集。

关于保形预测的理解，跟传统的区间估计是类似的。给定一种回归问题的预测方法，保形预测产生一个 95% 的预测区间 $\Gamma^{0.05}$，即该区间包含 Y 的概率至少为 95%，通常 $\Gamma^{0.05}$ 也包含预测值 $\widehat{Y}$。我们称 $\widehat{Y}$ 为点预测，称 $\Gamma^{0.05}$ 为区间预测。在分类的情况下，类别 Y 有有限个可能值，$\Gamma^{0.05}$ 可能由这些值中的几个值组成，或者在理想的情况下 $\Gamma^{0.05}$ 只是这些值中的一个值。

与传统区间估计所使用的方法不同的是，保形预测并没有研究预测值的分布或渐进分布，这也体现了保形预测的优势，可以适用于任何模型，因为研究渐进分布往往需要模型的假定。保形预测在处理连续因变量的回归预测问题时，对于单因变量的预测输出的是一个预测区间，对于多因变量的预测输出的是一个预测域。图6.1展示的是对单个连续因变量的区间预测。

保形预测在处理离散因变量的分类预测问题时，输出的是预测集。集合中的元

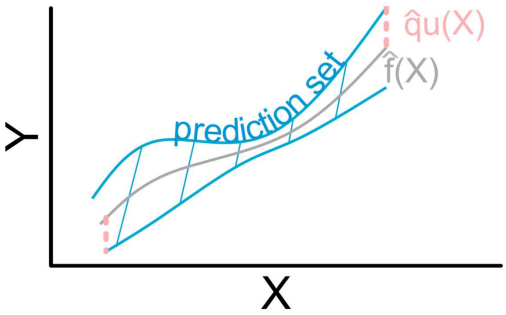


图 6.1 保形预测的预测区间

图 6.2 保形预测的预测集

素是由预测类别及对应的概率构成。图6.2 中我们展示了三种不同的输入值下，对松鼠类型预测的保形预测集。上述两图表明，预测区间和预测集都可以保证以较高的概率覆盖真实值。

6.1.1 简单运用

在介绍保形预测的流程之前，我们要先知道两个定义：一个是随机变量序列的可交换性，另一个是不符合度量。在后面章节中，保形预测的预测集既可代表回归问题的预测区间也可表示分类问题的预测集。

可交换性 (Exchangeability)

一个有限的随机变量序列是可交换的 (exchangeable)，是指随机变量的联合概率分布对随机变量的排列不变。

$$\Pr\left(\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n\right) = \Pr\left(\boldsymbol{X}_{\pi(1)}, \boldsymbol{X}_{\pi(2)}, \ldots, \boldsymbol{X}_{\pi(n)}\right)$$

这里 $\pi(1), \pi(2), \ldots, \pi(n)$ 代表自然数 $1, 2, \ldots, n$ 的任意一个排列。一个无限的随机变量序列是无限可交换 (infinitely exchangeable) 的，是指它的任意一个有限子序列都是可交换的。

如果一个无限随机变量序列 $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n, \ldots$ 是独立同分布的，那么它们是无限可交换的。反之不然。

不符合度量 (Nonconformity measure)

保形预测使用的不符合度量，度量了一个新例子与旧例子的不同程度，也可以说是度量预测值与真实值之间的不符合程度。因此，我们需要寻找一种方法来度量预测值与真实值之间的距离。

保形预测要求我们首先选择一个不符合度度量，而度量预测值和真实值之间距离的方法与数据类型和预测器的选择相关：在回归问题中，最常用的不符合度量是残差的绝对值：$R = |Y - \widehat{Y}|$。在分类问题中，最常用的不符合度量是：$H = 1 - \widehat{Y}$ 。

给定一个不符合度量，保形算法对每一个误覆盖水平 α 产生一个预测集 $\widehat{C}_n$。$\widehat{C}_n$ 为 $(1-\alpha)$ 预测集; 它有至少 $(1-\alpha)$ 的概率包含真实值 Y。

不同 α 得到的预测区间是嵌套的。因此，当 $\alpha_1 \geq \alpha_2$ 时，置信度 $(1-\alpha_1)$ 低于 $(1-\alpha_2)$，我们有 $\widehat{C}_{n,\alpha_1} \subseteq \widehat{C}_{n,\alpha_2}$。

在明确了以上两个概念之后，我们介绍运用保形预测来构造有效的预测集的几个基本步骤。假设我们有一个独立同分布的数据集：

$$\{(\boldsymbol{X}_1^{\mathrm{T}}, Y_1)^{\mathrm{T}}, \ldots, (\boldsymbol{X}_n^{\mathrm{T}}, Y_n)^{\mathrm{T}}\}$$

其中，$\boldsymbol{X}$ 是输入的特征，Y 是类别，n 是数据点的数量。我们假设一个机器学习模型：f：$\boldsymbol{X} \to Y$ 已经被训练。这个模型可以是一个经典的机器学习模型例如线性回归、支持向量机，或者深度学习技术例如全连接或卷积网络。目标是去估计模型输出的预测集。下面我们将描述保形预测的步骤：

保形预测的基本流程

a. 选择合适的不符合度量，计算不符合分数

选择一个适当的不符合度量 $S(\boldsymbol{X}, Y) \in \mathcal{R}$ 来测量模型输出 $\widehat{Y}$ 与类别 Y 之间的差异，亦称评分函数，得到值成为不符合分数。这个不符合度量非常重要，因为它决定了我们能得到什么样的预测集。例如，在回归问题中，我们可以用 $|\widehat{Y}_i - Y_i|$ 作为评分函数。通过这种方式，所得到的保形预测集在预测值 $\widehat{Y}_i$ 周围的 L_1 范式球内；在分类问题中，我们可以用 $(1 - \widehat{Y}_i)$ 作为评分函数，其中 $\widehat{Y}_i$ 是真实类别对应的预测值。

b. 计算不符合分数的 $(1-\alpha)$ 分位数

不符合分数的 $(1-\alpha)$ 分位数 $\widehat{q}$ 是通过计算 n 个不符合分数 $S_1 = S(\boldsymbol{X}_1, Y_1), \ldots, S_n = S(\boldsymbol{X}_n, Y_n)$ 的 $(1-\alpha)$ 分位数得到的。在完全保形预测方法中，我们需要训练 m 个模型来计算评分并构造预测集，其中 m 为可能的取值的个数，这无疑在计算上非常昂贵。为了降低计算复杂度，可以采用分裂保形预测的方法，分裂保形预测将整个

训练集分割成合适的训练集和校准集 (calibration set)，然后，只对训练集进行训练，在校准集上计算不符合分数。

c. 使用模型预测和不符合分数的 $(1-\alpha)$ 分位数构造预测集

接下来，我们使用模型预测值和不符合分数的 $(1-\alpha)$ 分位数构造保形预测集。假设新的输入为 $\boldsymbol{X}_{n+1}$，针对分类问题，得到的保形预测集可以表示为：$\tau(\boldsymbol{X}_{n+1}) = \{Y : S(\boldsymbol{X}_{n+1}, Y) \leq \widehat{q}\}$。例如，在使用保形预测进行回归分析时，不符合度量 $S(\boldsymbol{X}_{n+1}, Y) = |Y - \widehat{f}(\boldsymbol{X}_{n+1})|$，其中 $\widehat{f}$ 是用训练数据集得到预测器。则此时得到的 $1-\alpha$ 预测区间可以表示为：$\widehat{C}_n = [\widehat{f}(\boldsymbol{X}_{n+1}) - \widehat{q}, \widehat{f}(\boldsymbol{X}_{n+1}) + \widehat{q}]$。

6.1.2 预测集的有效性

我们的保形预测总是有效的。Glenn Shafer 和 Vladimir Vovk 着眼于可交换性和独立性之间的关系，得出了可交换序列的大数定律，这为我们认为 95% 的预测区域在 95% 的时间内是正确的提供了信心基础。保形预测最重要的新颖之处在于其连续误差是概率独立的。这使得我们能够以一种不同寻常的直接方式来解释 “95% 的正确率”。

对于保形预测的有效性的精确讨论实际上需要我们区分两种有效性: 保守的和精确的。一般来说，保形预测是保守有效的: 当它输出一个 $(1-\alpha)$ 集 (也就是说, 一组预测集置信水平为 $(1-\alpha)$ 时错误的概率不大于 α，并且在预测连续的例子时，它所犯的错误之间几乎没有相关性。这意味着，根据大数定律，在置信水平 $(1-\alpha)$ 上的长期错误频率约为 α 或更小。在实践中，保守性往往不是很大，尤其是当 n 很大时，经验结果显示，长期误差的频率与 α 非常接近。然而，从理论的角度来看，为了获得准确的有效性，我们必须在预测过程中引入一个随机误差 α，其中 $(1-\alpha)$ 集出现错误的概率正好是 α，错误在不同的试验中是独立产生的，而长期错误的频率收敛到 α。

保形预测可以提供数学上严格的保证。设 Y_{n+1} 为真值。Y 可以是分类问题中的一个类别, 也可以是回归问题中的一个实值。设 $\tau(\boldsymbol{X}_{n+1})$ 是一个预测集 (或区间)。如果 Y_{n+1} 在 $\tau(\boldsymbol{X}_{n+1})$ 内，我们将其定义为 $\tau(\boldsymbol{X}_{n+1})$ 覆盖 Y_{n+1}，即：$Y_{n+1} \in \tau(\boldsymbol{X}_{n+1})$。然后，给定一组同独立分布样本 $\{(\boldsymbol{X}_1^{\mathrm{T}}, Y_1)^{\mathrm{T}}, \ldots, (\boldsymbol{X}_n^{\mathrm{T}}, Y_n)^{\mathrm{T}}\}$，保形预测集满足以下覆盖保证：$\Pr(Y_{n+1} \in \tau(\boldsymbol{X}_{n+1})) \geq 1-\alpha$。

基于可交换性 (exchangeability) 假设的覆盖保证的证明可以在文章 A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification 的附录中找到。以上覆盖率保证在有限样本的情况下也是成立的，且与普通的统计推断通常都对变量的分布有更严格的假定 (比如 Gaussianity) 不同，该保证仅需要非常弱的条件 (可交换性),。

6.1.3 效率

针对分类问题，保形预测得到的预测集可能包含一个类别，也可能包含多个类别。在处理回归问题时，预测集通常是包含预测值的一个区间。在保证一定置信度的情况下，保形预测集越小，则效率越高，即分类预测集的元素越少或者回归预测区间越短，则越好。这就是我们通常所说的高效率保形预测集。

6.2 保形回归

保形预测可以用于回归，也可以用于分类，接下来我们首先以回归为例，解释保形预测的思想和原理。本质上，保形预测理论背后的基本思想与样本分位数有关。

6.2.1 保形预测基本思想

假设有独立同分布的随机变量样本 $U_1,\ldots,U_n$（事实上，此处独立同分布的假设可以被更弱的假设—可交换性替代）。对于一个给定的误覆盖水平 $\alpha\in(0,1)$ 和另一个独立同分布的样本 U_{n+1}，当我们基于 $U_1,\ldots,U_n$ 定义样本分位数 $\widehat{q}_{1-\alpha}$ 为：

$$\widehat{q}_{1-\alpha}=\begin{cases}U_{(\lceil(n+1)(1-\alpha)\rceil)} & \text{if } \lceil(n+1)(1-\alpha)\rceil\leq n\\ \infty & \text{otherwise}\end{cases},$$

可以得到：

$$\Pr\left(U_{n+1}\leq\widehat{q}_{1-\alpha}\right)\geq 1-\alpha,$$

其中，$U_{(1)}\leq\cdots\leq U_{(n)}$ 表示 $U_1,\ldots,U_n$ 的次序统计量。通过可交换性，U_{n+1} 在 $U_1,\ldots,U_n,U_{n+1}$ 中的排序是在集合 $\{1,\ldots,n+1\}$ 上均匀分布的，因此以上有限样本覆盖性质是很容易被验证的。

在回归问题中，我们观测到独立同分布的样本 $\boldsymbol{Z}_i=\left(\boldsymbol{X}_i^{\mathrm{T}},Y_i\right)^{\mathrm{T}}\sim\Pr(\boldsymbol{z}),i=1,\ldots,n$，我们可以考虑以下方法来构造 Y_{n+1} 在新特征值 $\boldsymbol{X}_{n+1}$ 处的预测区间，其中 $(\boldsymbol{X}_{n+1}^{\mathrm{T}},Y_{n+1})^{\mathrm{T}}$ 是分布 $\Pr(\boldsymbol{z})$ 中一个独立的随机变量。按照上述思想，我们可以构造以下预测区间：

$$\widehat{C}_{\text{naive}}\left(\boldsymbol{X}_{n+1}\right)=\left[\widehat{f}\left(\boldsymbol{X}_{n+1}\right)-\widehat{F}_n^{-1}(1-\alpha),\widehat{f}\left(\boldsymbol{X}_{n+1}\right)+\widehat{F}_n^{-1}(1-\alpha)\right],$$

其中，$\widehat{f}$ 是估计的回归函数预测器，$\widehat{F}_n$ 是拟合残差 $\left|Y_i-\widehat{f}\left(\boldsymbol{X}_i\right)\right|,\quad i=1,\ldots,n$ 的经验分布，$\widehat{F}_n^{-1}(1-\alpha)$ 是 $\widehat{F}_n$ 的 $(1-\alpha)$ 分位数。假设估计的回归函数 $\widehat{f}$ 是准确的，

则该预测区间在大样本情况下是有效的 (即估计的拟合残差分布的 $(1-\alpha)$ 分位数 $\widehat{F}_n^{-1}(1-\alpha)$ 足够接近总体残差 $|Y_i - f(\boldsymbol{X}_i)|, \quad i=1,\ldots,n$ 的 $(1-\alpha)$ 分位数)。保证 $\widehat{f}$ 的准确性通常需要数据分布 $\Pr(\boldsymbol{z})$ 和回归预测器 $\widehat{f}$ 本身均满足一定的条件, 例如适当地选择的预测模型和调优参数。

6.2.2 完全保形预测

一般来说，上述方法得到的预测区间可以粗略地覆盖真实值，因为拟合残差分布往往是向下倾斜的。保形预测区间 [73] [74] [75] [76] 克服了上述原始方法预测区间的缺陷，并且，某种程度上值得注意的是，保形预测可以保证提供适当的有限样本覆盖，而无需对 $\Pr(\boldsymbol{z})$ 和回归预测器 $\widehat{f}$ 进行任何假设（除了 f 是数据点的对称函数)。

具体算法如下: 对于一个新的输入值 $\boldsymbol{X}_{n+1}$，其对应的 Y_{n+1} 是未知的。因此我们为 Y_{n+1} 考虑一个实验集合 $\mathbf{Y}_{\text{trial}}$。从实验集合中，选取某一值 $Y \in \mathbf{Y}_{\text{trial}}$，我们基于增广数据集 $\boldsymbol{Z}_1 \ldots, \boldsymbol{Z}_n, (\boldsymbol{X}_{n+1}^{\mathrm{T}}, Y)^{\mathrm{T}}$ 上进行训练，构造一个增广回归预测器 $\widehat{f}_Y$。接下来我们计算不符合分数，即残差的绝对值。

$$
\begin{aligned}
R_{Y,i} &= \left|Y_i - \widehat{f}_Y(\boldsymbol{X}_i)\right|, \quad i=1,\ldots,n \quad \text{和} \\
R_{Y,n+1} &= \left|Y - \widehat{f}_Y(\boldsymbol{X}_{n+1})\right|,
\end{aligned}
$$

并且我们基于 $n+1$ 个不符合分数对 $R_{Y,n+1}$ 进行排序，得到

$$
\begin{aligned}
\pi(Y) &= \frac{1}{n+1}\sum_{i=1}^{n+1} I\{R_{Y,i} \le R_{Y,n+1}\} \\
&= \frac{1}{n+1} + \frac{1}{n+1}\sum_{i=1}^{n} I\{R_{Y,i} \le R_{Y,n+1}\},
\end{aligned}
$$

即 $\pi(Y)$ 是增广样本中残差绝对值小于最后一个样本点的残差绝对值 $R_{Y,n+1}$ 的样本所占的比例，其中 $I\{\cdot\}$ 表示示性函数。由于数据点的可交换性和 $\widehat{f}_Y$ 的对称性，当我们在估计 Y_{n+1} 时，我们发现构造的次序统计量 $\pi(Y_{n+1})$ 在集合 $\{1/(n+1), 2/(n+1), \ldots, 1\}$ 上是均匀分布的, 这意味着:

$$
\Pr\{(n+1)\pi(Y_{n+1}) \le \lceil (1-\alpha)(n+1)\rceil\} \ge 1-\alpha.
$$

上述推理可以通过如下假设检验过程理解，即对应原假设 $\mathbf{H}_0 : Y_{n+1} = Y$。在给定显著性水平 α 下，我们有如下结论:

(1) 如果事件 $\{(n+1)\pi(Y) > \lceil (1-\alpha)(n+1)\rceil\}$ 成立，则拒绝原假设;

(2) $1-\pi(Y)$ 相当于 P 值，如果 $1-\pi(Y) < \alpha$，则拒绝原假设。

通过在 $\mathbf{Y}_{\text{trial}}$ 遍历所有 Y 的取值，我们可以得到在 $\boldsymbol{X}_{n+1}$ 处的保形预测区间，即:

$$\widehat{C}_{\text{conf}}\left(\boldsymbol{X}_{n+1}\right)=\{Y\in\mathbf{Y}_{\text{trial}}:(n+1)\pi(Y)\leq\lceil(1-\alpha)(n+1)\rceil\}.$$

针对每一个新的自变量，我们都必须重复以上步骤产生一个预测区间。为了完整起见，我们总结为如下**算法 1**。

表 6.1 完全保形预测算法

算法 1：完全保形预测

输入： 数据 $\left(\boldsymbol{X}_i^{\mathrm{T}},Y_i\right)^{\mathrm{T}},i=1,\ldots,n$, 显著性水平 $\alpha\in(0,1)$, 回归算法 $\mathcal{A}$, 用于构造预测区间的新的点 $\mathbf{X}_{\text{new}}=\{\boldsymbol{X}_{n+1},\boldsymbol{X}_{n+2},\ldots\}$, 和作为试验集合 $\mathbf{Y}_{\text{trial}}=\{Y_1,Y_2,\ldots\}$

输出： $\mathbf{X}_{\text{new}}$ 中每一个元素对应的预测区间

for $\boldsymbol{x}\in\mathbf{X}_{\text{new}}$ **do**

  for $Y\in\mathbf{Y}_{\text{trial}}$ **do**

$$\widehat{f}_Y=\mathcal{A}\left(\left\{\left(\boldsymbol{X}_1,Y_1\right),\ldots,\left(\boldsymbol{X}_n,Y_n\right),(\boldsymbol{x},Y)\right\}\right)$$

$$R_{Y,i}=\left|Y_i-\widehat{f}_Y\left(\boldsymbol{X}_i\right)\right|,\quad i=1,\ldots,n,$$

以及 $R_{Y,n+1}=\left|Y-\widehat{f}_Y\left(\boldsymbol{x}\right)\right|$

$$\pi(Y)=\left(1+\sum_{i=1}^{n}I\left\{R_{Y,i}\leq R_{Y,n+1}\right\}\right)/(n+1)$$

  end for

$$\widehat{C}_{\text{conf}}\left(\boldsymbol{x}\right)=\{Y\in\mathbf{Y}_{\text{trial}}:(n+1)\pi(Y)\leq\lceil(1-\alpha)(n+1)\rceil\}$$

end for

返回 $\widehat{C}_{\text{conf}}\left(\boldsymbol{x}\right)$, for each $\boldsymbol{x}\in\mathbf{X}_{\text{new}}$

上述保形预测得到的预测区间具有有效的有限样本覆盖，并且这个区间是精确的，意味着它没有过多地覆盖，以下我们对此进行说明:

如果 $\left(\boldsymbol{X}_i^{\mathrm{T}},Y_i\right)^{\mathrm{T}},i=1,\ldots,n$ 为独立同分布的，那么对于一个新的独立同分布的点 $\left(\boldsymbol{X}_{n+1}^{\mathrm{T}},Y_{n+1}\right)^{\mathrm{T}}$，保形预测建立的预测区间 $\widehat{C}_{\text{conf}}\left(\boldsymbol{X}_{n+1}\right)$ 有以下覆盖保证:

$$\Pr\left(Y_{n+1}\in\widehat{C}_{\text{conf}}\left(\boldsymbol{X}_{n+1}\right)\right)\geq1-\alpha,$$

如果我们另外假设对所有 $Y\in\mathcal{R}$，拟合的绝对残差 $R_{Y,i}=\left|Y_i-\widehat{f}_Y\left(\boldsymbol{X}_i\right)\right|,i=1,\ldots,n$ 有一个连续的联合分布，那么下式也成立:

$$\Pr\left(Y_{n+1}\in\widehat{C}_{\text{conf}}\left(\boldsymbol{X}_{n+1}\right)\right)\leq1-\alpha+\frac{1}{n+1}.$$

该结论的证明详见文章 [77]。

6.3 保形方法

6.3.1 分裂保形预测

上一节中讲述的完全保形预测方法计算量较大，因为它要遍历所有 Y 的实验集。即对于任何 $\boldsymbol{X}_{n+1}$ 和未知的 Y，为了辨别某一个给定的 Y 是否包含在 $\widehat{C}_{\text{conf}}(\boldsymbol{X}_{n+1})$ 中，我们在增广数据集（其中包括新的点 $(\boldsymbol{X}_{n+1}, Y)$）上重新训练模型，并重新计算和排序绝对残差。这个步骤需要在 $\mathbf{Y}_{\text{trial}}$ 集合里面的所有元素上，重复进行，计算成本很高。在非线性回归问题中，核密度估计或样条方法去训练模型的较高复杂度都会影响完全保形预测。特别是在高维回归中，我们可能会使用相对复杂的回归预测器，如 Lasso 回归或深度学习等，模型的训练仍会花费大量时间，因此执行有效的完全保形预测仍然是一个有待解决的问题。

幸运的是，有一种完全通用的方法，我们称之为分裂保形预测，其计算成本大大低于完全保形预测方法。分裂保形方法采用样本分割将拟合步骤和排序步骤分离，其计算代价与拟合步骤相对简单。**算法 2** 总结了分裂保形预测算法，其他细节请参考 [78]。在这里，以及以后讨论分裂保形预测时，为了简单起见，我们假设样本容量 n 是偶数，当 n 是奇数时只需要非常小的改变。

表 6.2 分裂保形预测算法

算法 2：分裂保形预测

输入： 数据 $(\boldsymbol{X}_i^{\mathrm{T}}, Y_i)^{\mathrm{T}}, i = 1, \ldots, n$, 显著性水平 $\alpha \in (0, 1)$, 回归算法 $\mathcal{A}$,

输出： $\boldsymbol{x} \in \mathcal{R}^p$ 上的预测区间

将 $\{1, \ldots, n\}$ 随机分裂为两个大小相等的数据集 $\mathcal{I}_1, \mathcal{I}_2$

$\widehat{f} = \mathcal{A}\left(\{(\boldsymbol{X}_i^{\mathrm{T}}, Y_i)^{\mathrm{T}} : i \in \mathcal{I}_1\}\right)$

$R_i = \left|Y_i - \widehat{f}\left(\boldsymbol{X}_i\right)\right|, i \in \mathcal{I}_2$

$d = \{R_i : i \in \mathcal{I}_2\}$ 中第 k 小的值，其中 $k = \lceil (1-\alpha)(\frac{n}{2}+1) \rceil$

返回 $\widehat{C}_{\text{split}}(\boldsymbol{x}) = [\widehat{f}(\boldsymbol{x}) - d, \widehat{f}(\boldsymbol{x}) + d]$, 对于所有 $\boldsymbol{x} \in \mathcal{R}^p$

如果 $(\boldsymbol{X}_i^{\mathrm{T}}, Y_i)^{\mathrm{T}}, i = 1, 2, \ldots, n$ 是独立同分布的，对于一个新的数据点 $(\boldsymbol{X}_{n+1}^{\mathrm{T}}, Y_{n+1})^{\mathrm{T}}$，由**算法 2** 建立的分裂保形预测区间 $\widehat{C}_{\text{split}}\left(\boldsymbol{X}_{n+1}\right)$，满足：

$$\Pr\left(Y_{n+1} \in \widehat{C}_{\text{split}}\ \left(\boldsymbol{X}_{n+1}\right)\right) \geq 1 - \alpha.$$

另外，如果我们假设残差 $R_i, i \in \mathcal{I}_2$ 具有连续的联合分布，那么

$$\Pr\left(Y_{n+1} \in \widehat{C}_{\text{split}}\ \left(\boldsymbol{X}_{n+1}\right)\right) \leq 1 - \alpha + \frac{2}{n+2}.$$

与完全保形预测方法相比，分裂保形预测除了具有极高的效率外，在内存需求方面也具有优势。例如，如果回归算法 $\mathcal{A}$ 涉及变量选择，如 Lasso 或双向逐步回归等，当我们在估计新的点 $\boldsymbol{X}_i, i \in \mathcal{I}_2$ 时，我们只需运用基于样本 $\mathcal{I}_1$ 选择的变量，来拟合模型计算基于样本 $\mathcal{I}_2$ 的残差。因此算法 2 可以大大节省内存。

分裂保形预测也可以使用不均衡分裂实现。采用 $\rho \in (0,1)$，令 $|\mathcal{I}_1| = \rho n$ 则 $|\mathcal{I}_2| = (1-\rho)n$。例如，当回归算法很复杂的时候，可以选择 $\rho > 0.5$ 使训练得到的回归预测器 $\widehat{f}$ 更准确。

6.3.2 Jackknife 保形预测

Jackknife 保形预测是一种计算复杂性介于完全保形法和分裂保形法之间的保形预测方法。该方法利用留一残差的分位数来定义预测区间，具体内容如**算法 3** 所示。

表 6.3 Jackknife 保形预测算法

算法 3：Jackknife 保形预测

输入： 数据 $(\boldsymbol{X}_i^{\mathrm{T}}, Y_i)^{\mathrm{T}}, i = 1, \ldots, n$, 显著性水平 $\alpha \in (0,1)$, 回归算法 $\mathcal{A}$,

输出： $\boldsymbol{x} \in \mathcal{R}^p$ 上的预测区间

for $i \in \{1, \ldots, n\}$ **do**

$\widehat{f}^{(-i)} = \mathcal{A}(\{(\boldsymbol{X}_\ell, Y_\ell) : \ell \neq i\})$

$R_i = \left| Y_i - \widehat{f}^{(-i)}(\boldsymbol{X}_i) \right|$

end for

$d = \{R_i : i \in \{1, \ldots, n\}\}$ 中第 k 小的值，其中 $k = \lceil n(1-\alpha) \rceil$

返回 $\widehat{C}_{\mathrm{Jack}}(\boldsymbol{x}) = [\widehat{f}(\boldsymbol{x}) - d, \widehat{f}(\boldsymbol{x}) + d]$, 对于所有 $\boldsymbol{x} \in \mathcal{R}^p$

与分裂保形法相比，Jackknife 法的一个优点是它在构造绝对残差时使用更多的训练数据，随后再构造分位数。这意味着它通常可以产生更短的间隔长度。我们注意到，由于对称性，Jackknife 法具有有限样本内覆盖性质：

$$\Pr\left(Y_i \in \widehat{C}_{\mathrm{Jack}}(\boldsymbol{X}_i)\right) \geq 1 - \alpha, \quad \text{对于所有 } i = 1, \ldots, n$$

但是就样本外覆盖而言（真正的预测推断），它的属性比较脆弱。

接下来看几篇经典的 Jackknife 估计方法文章。Butler 和 Rothman [79] 表明，在低维线性回归中，Jackknife 法在足够强的正则条件下产生渐近有效区间，也意味着需要保证线性回归估计量的一致性。最近，Steinberger 和 Leeb [80] 在高维回归中建立了 Jackknife 区间的渐近有效性，他们虽然不需要基本估计量 $\widehat{f}$ 的一致性，但需要 $\widehat{f}$ 的一致渐近均方误差有界的条件。此外，Butler 和 Rothman [79] 和 Steinberger 和 Leeb [80] 的回归分析均基于线性模型，即回归函数是特征的线性函数，特征独立

于误差，误差是同方差的。但是这里介绍的 Jackknife 保形预测方法并不需要这些条件。

6.3.3 局部加权保形预测

当噪声是异方差的时候，我们可以用局部加权的思路替换完全保形预测方法或分裂保形方法中的绝对残差，即使用

$$V_i = \frac{|Y_i - \widehat{f}(\boldsymbol{X}_i)|}{\widehat{\sigma}(\boldsymbol{X}_i)}$$

替代

$$R_i = |Y_i - \widehat{f}(\boldsymbol{X}_i)|$$

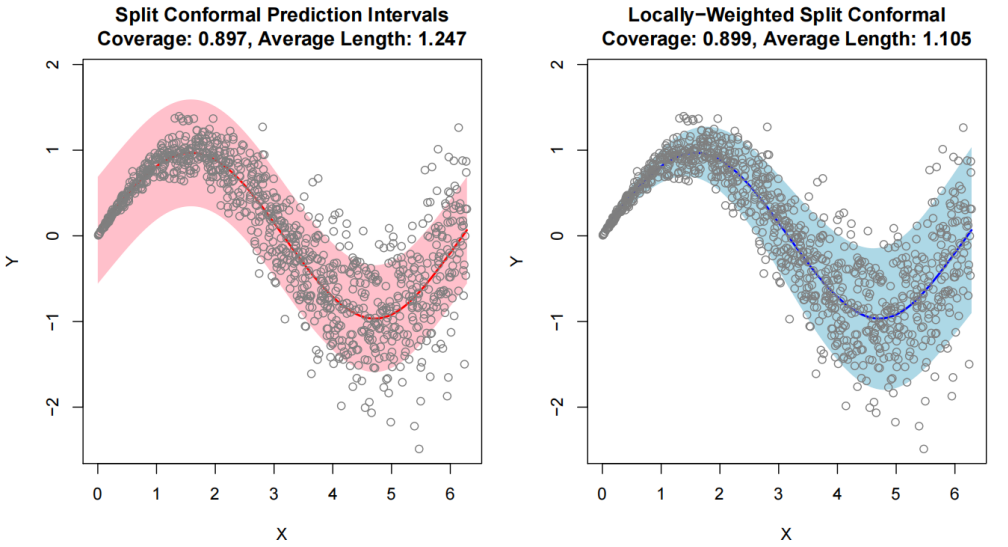
其中 $\widehat{\sigma}^2(\boldsymbol{x})$ 是绝对残差 $\mathrm{var}(|Y - \widehat{f}(\boldsymbol{X})| \mid \boldsymbol{X} = \boldsymbol{x})$ 的方差函数的一个估计。(注意：$\widehat{f}$ 和 $\widehat{\sigma}$ 可以联合计算，也可以单独计算。)

局部加权对预测波段的局部宽度可能有很大变化。以下是在分裂保形预测区间形式：

$$\widetilde{C}_{\mathrm{local}}(\boldsymbol{x}) = [\widehat{f}(\boldsymbol{x}) - \widehat{\sigma}(\boldsymbol{x})\widetilde{q}, \widehat{f}(\boldsymbol{x}) + \widehat{\sigma}(\boldsymbol{x})\widetilde{q}]$$

其中 $\widetilde{q} = \mathrm{Quantile}\left(1-\alpha; \{V_i\}_{i\in\mathcal{I}_2}\right)$ 表示 V_i 的 $1-\alpha$ 分位数。

异方差噪声示例



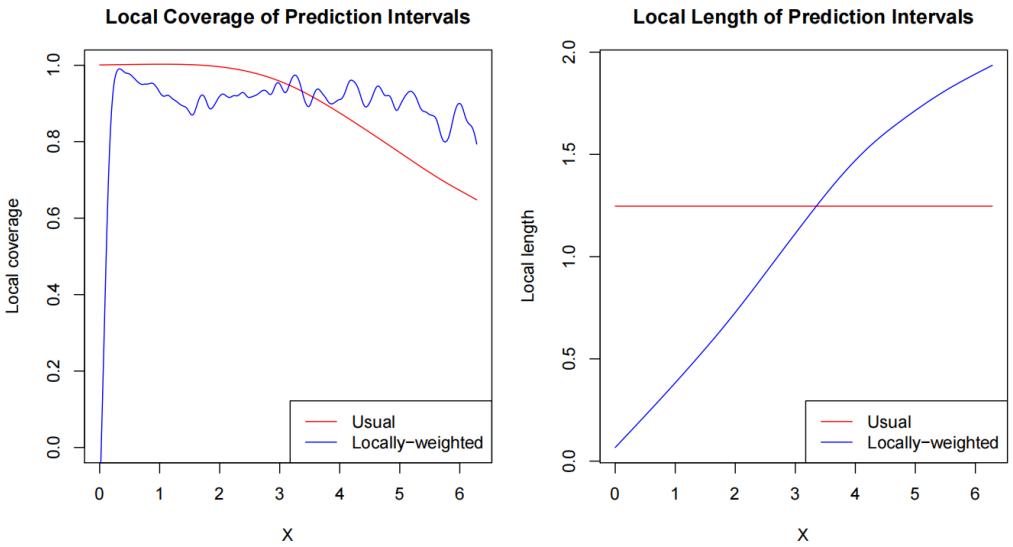


图 6.3 局部加权保形预测区间，其中 n=100, p=1。

从图6.3 可以看出，通过引入局部权重，我们可以获得能适应数据异质性的预测区域，在保证和之前的方法具有相同的有效性的同时，它具有更小的平均长度，局部的覆盖率也约等于我们想要的值（比如说 0.9）。

6.4 保形分类

选择合适的不符合度量是进行保形预测的重要步骤，我们在使用保形预测法进行分类时，应该根据数据类型和预测方法来选择不符合度量。下面我们将介绍两种不同的不符合度量下保形预测在分类中的应用。

6.4.1 Softmax 法

在机器学习尤其是深度学习中，Softmax 是个常用而且比较重要的函数，尤其在多分类的场景中使用广泛。假设预测目标是 K 个类别，则通过神经网络多层变换后得到一个 K 维的输入 $\boldsymbol{z} = (z_1, \cdots, z_K)$，接下采用 Softmax 函数将 $\boldsymbol{z}$ 映射为 K 个 0-1 之间的实数，并且归一化保证和为 1，因此多分类的概率之和也刚好为 1。Softmax 函数形式如下：

$$\text{Softmax}(\boldsymbol{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \quad k = 1, \ldots, K.$$

在分类保形预测中，我们可以将 Softmax 函数作为我们的预测函数，Softmax 方法经常用于图片分类，下面我们简单介绍 Softmax 方法下的保形预测。

假设我们的预测任务是判断一个 p 维图像 $\boldsymbol{X}$ 属于哪一个类别 $Y \in \{1, \ldots, K\}$。首先我们使用训练数据和 Softmax 函数得到预测模型，也可称其为分类预测器 $\widehat{f}$。分类预测器可以对于每个输入 $\boldsymbol{x}$ 输出对应每个类别的 Softmax 分数：$\widehat{f}(\boldsymbol{x}) = \text{Softmax}(\boldsymbol{z}) \in [0,1]^K$，其中 $\widehat{f}(\boldsymbol{x})_k = \text{Softmax}(\boldsymbol{z})_k$。然后，我们取适当数量的未用于训练的新数据点 $(\boldsymbol{X}_1^{\mathrm{T}}, Y_1)^{\mathrm{T}}, \ldots, (\boldsymbol{X}_n^{\mathrm{T}}, Y_n)^{\mathrm{T}}$ 作为校准数据集。利用 $\widehat{f}$ 和校准数据，我们试图构建一个可能的类别的置信集 $\mathcal{T}(\boldsymbol{x}) \subset \{1, \ldots, K\}$，且这个置信集在下面意义下是有效的：

$$1 - \alpha \leq \Pr\left(Y_{n+1} \in \mathcal{T}(\boldsymbol{X}_{n+1})\right) \leq 1 - \alpha + \frac{1}{n+1}.$$

其中，$(\boldsymbol{X}_{n+1}^{\mathrm{T}}, Y_{n+1})^{\mathrm{T}}$ 是来自同一分布的新测试点。换句话说，置信集包含正确类别的概率几乎正好是 $(1-\alpha)$，我们称这种性质为边际覆盖。为了使用 $\widehat{f}$ 和校准数据构造 $\mathcal{T}$，我们将执行一个简单的校准步骤。首先，我们设置不符合分数为 1 减去正确类别的 Softmax 分数：

$$S_i = S(\boldsymbol{X}_i, Y_i) = 1 - \widehat{f}(\boldsymbol{X}_i)_{Y_i},$$

如果预测器不好，S_i 有可能会很大。接下来定义 $\widehat{q}$ 为 $S_1, \ldots, S_n$ 的 $\lceil (1-\alpha)(n+1) \rceil$ 分位数，这实际上是经过小小修正的 n 个样本的 $(1-\alpha)$ 分位数。对于一个新的输入 $\boldsymbol{X}_{n+1}$（Y_{n+1} 未知），产生一个保形预测集：

$$\mathcal{T}(\boldsymbol{X}_{n+1}) = \{Y : S(\boldsymbol{X}_{n+1}, Y) < \widehat{q}\} = \left\{Y : \widehat{f}(\boldsymbol{X}_{n+1})_Y > 1 - \widehat{q}\right\}$$

该预测集包含那些对应 Softmax 分数足够大的类别，如图6.4 所示。值得注意的是，不论我们应用什么样的预测模型，选择任何分布的数据集，这个算法得到的预测集都是具有覆盖保证的。

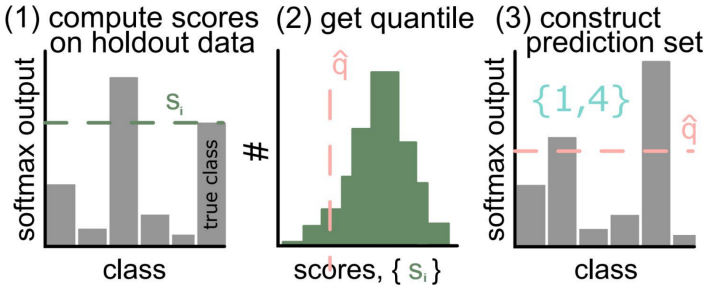


图 6.4 保形分类——Softmax 法

上述校准步骤的 PyTorch 代码如下：

```
#First, get conformal scores. n = calib_y.shape[0]
scores = 1-model(calib_x).softmax(dim=1)[:,calib_y]
```

```
# Second,get the adjusted quantile
qhat = torch.quantile(scores,np.ceil((n+1)(1-alpha))/n)
#Finally, deploy!
prediction_sets = (model(test_x) > (1-qhat)).nonzero()
```

6.4.2 最近邻法

最近邻法也是我们进行分类常用的方法之一。在保形预测中，我们使用最近邻法分类时，分类预测器将新的样本分类为与其距离最近的样本所对应的类别。

假设有 n 个样本 $\boldsymbol{Z}_1 = (\boldsymbol{X}_1^{\mathrm{T}}, Y_1)^{\mathrm{T}}, \boldsymbol{Z}_2 = (\boldsymbol{X}_2^{\mathrm{T}}, Y_2)^{\mathrm{T}}, ..., \boldsymbol{Z}_n = (\boldsymbol{X}_n^{\mathrm{T}}, Y_n)^{\mathrm{T}}$，每个 $\boldsymbol{Z}_i$ 包含特征向量 $\boldsymbol{X}_i$ 和类别 $Y_i \in \{1, \cdots, K\}$。然后我们对于一个新的样本点 $\boldsymbol{Z}_{n+1} = (\boldsymbol{X}_{n+1}^{\mathrm{T}}, Y_{n+1})^{\mathrm{T}}$，我们只能观察到 $\boldsymbol{X}_{n+1}$ 而不知道 Y_{n+1} 。最近邻法寻找距离 $\boldsymbol{X}_{n+1}$ 最近的 $\boldsymbol{X}_i$，并使用它的类别 Y_i 作为 Y_{n+1} 的预测值。如果没有合适的预测器，我们很难度量这个预测的正确性。但是我们可以通过比较 $\boldsymbol{X}$ 到与其有相同类别的旧例子的距离和 $\boldsymbol{X}$ 到与其有不同类别的旧例子的距离，来得到不符合度量，因此我们用下式表示不符合分数：

$$S_i = S(\boldsymbol{X}_i, Y_i) = \frac{\min\{\|\boldsymbol{X}_j - \boldsymbol{X}_i\| : j \neq i, Y_j = Y_i\}}{\min\{\|\boldsymbol{X}_j - \boldsymbol{X}_i\| : j \neq i, Y_j \neq Y_i\}}$$

不符合分数 S_i 大小，也可以代表分类是否正确。接下来定义 $\widehat{q}$ 为 $S_1, \ldots, S_n$ 的 $\lceil (1-\alpha)(n+1) \rceil$ 分位数。对于一个新的输入 $\boldsymbol{X}_{n+1}$，我们得到保形预测集：

$$\mathcal{T}(\boldsymbol{X}_{n+1}) = \{Y : S(\boldsymbol{X}_{n+1}, Y) < \widehat{q}\}$$

6.5 保形预测实践

6.5.1 R 语言实践

保形预测涉及的 R 包，可在该网址查阅：

$$https://github.com/ryantibs/conformal/$$

- 保形方法：全保形预测，分裂保形预测，Jackknife 保形预测，留一保形预测，局部加权法
- 基本的估计方法：线性回归，岭回归，套索回归，逐步回归，随机森林，或自定义
- 简单地运行模拟结果的框架在 [77] 中可以看到

分裂保形预测的简单代码

```
conformal.pred.split = function(x, y, x0, train.fun, predict.fun,alpha=0.1,rho=0.5,
w=NULL,mad.train.fun=NULL,mad.predict.fun=NULL,split=NULL,seed=NULL,verbose=FALSE) {
  #设置数据
  x = as.matrix(x)
  y = as.numeric(y)
  n = nrow(x)
  p = ncol(x)
  x0 = matrix(x0,ncol=p)
  n0 = nrow(x0)
  #检查输入参数
  check.args(x=x,y=y,x0=x0,alpha=alpha,train.fun=train.fun,predict.fun=predict.fun,
  mad.train.fun=mad.train.fun,mad.predict.fun=mad.predict.fun)
  #如果rho是空值，或者长度不等于一，或者不是数值，或者rho小于等于0或大于等于1，则停止
  if (is.null(rho) || length(rho)!= 1 || !is.numeric(rho) || rho<=0 || rho>=1)
    stop("rho must be a number in between 0 and 1")#rho必须是0和1之间的数字
  if (is.null(w)) w = rep(1,n+n0)
  #用户可以用一个字符串作为verbose参数
  if (verbose == TRUE) txt = ""
  if (verbose != TRUE && verbose != FALSE) {
    txt = verbose
    verbose = TRUE
  }
  #如果用户为了分割传递索引,则
  if (!is.null(split)) i1 = split
  #否则进行随机分割
  else {
    if (!is.null(seed)) set.seed(seed)
    i1 = sample(1:n,floor(n*rho))
  #Floor接受一个数字参数x，并返回一个包含不大于x中相应元素的最大整数的数字向量
  #sample(x, size, replace = FALSE, prob = NULL)
  #Sample从x的元素中提取指定大小的样本,使用替换或不替换
  }
  #从样本1到n中提取(rho*n)个样本，如果(rho*n)不是整数，则取小于它的最大整数
  i2 = (1:n)[-i1]#i2表示1到n去掉i1已经提取过的值
  n1 = length(i1)
  n2 = length(i2)
  if (verbose) {
    cat(sprintf("%sSplitting data into parts of size %i and %i ...\n",txt,n1,n2))
    cat(sprintf("%sTraining on first part ...\n",txt))
  }
  #用第一部分的数据i1进行训练
  out = train.fun(x[i1,,drop=F],y[i1])#输入的是x的i1行，和Y的i1行，得到训练模型out
  #使用训练好的模型out和点X，得到X的预测值Y
  #并将其排列为n行的矩阵拟合，可以看到和真实值的差别
  fit = matrix(predict.fun(out,x),nrow=n)
  #使用训练好的模型out和新的点X0，得到新的X的预测值Y，并将其排列为n0行的矩阵
  pred = matrix(predict.fun(out,x0),nrow=n0)
  m = ncol(pred)#令m等于新预测值Y的列数
  if (verbose) {
    cat(sprintf("%sComputing residuals and quantiles on second part ...\n",txt))
```

```
  }
  #使用第二部分的数据i2计算残差和分位数
  #计算校准集残差绝对值，得到的结果是一个n2*1的矩阵
  res = abs(y[i2] - matrix(predict.fun(out,x[i2,,drop=F]),nrow=n2))
  lo = up = matrix(0,n0,m)#暂且定义low和up均为n0*m的矩阵，用0填充
  for (l in 1:m) {
    #局部评分
    if (!is.null(mad.train.fun) && !is.null(mad.predict.fun)) {
      res.train = abs(y[i1] - fit[i1,l])
      mad.out = mad.train.fun(x[i1,,drop=F],res.train)
      mad.x2 = mad.predict.fun(mad.out,x[i2,,drop=F])
      mad.x0 = mad.predict.fun(mad.out,x0)
      res[,l] = res[,l] / mad.x2
    }
    else {
      mad.x0 = rep(1,n0)#不加权时，mad.x0是n0维的向量，以1填充
    }
    #order将残差绝对值按升序排列,得到的是残差的位置序号
    o = order(res[,l]); r = res[o,l]; ww = w[i2][o]#r是将残差按升序排列
    for (i in 1:n0) {
      q = weighted.quantile(c(r,Inf),1-alpha,w=c(ww,w[n+i]),sorted=TRUE)#取分位数
      lo[i,l] = pred[i,l] - q * mad.x0[i]#置信区间的下界是perd-q
      up[i,l] = pred[i,l] + q * mad.x0[i]#置信区间的上界是perd+q
    }
  }
  return(list(pred=pred,lo=lo,up=up,fit=fit,split=i1))
}
```

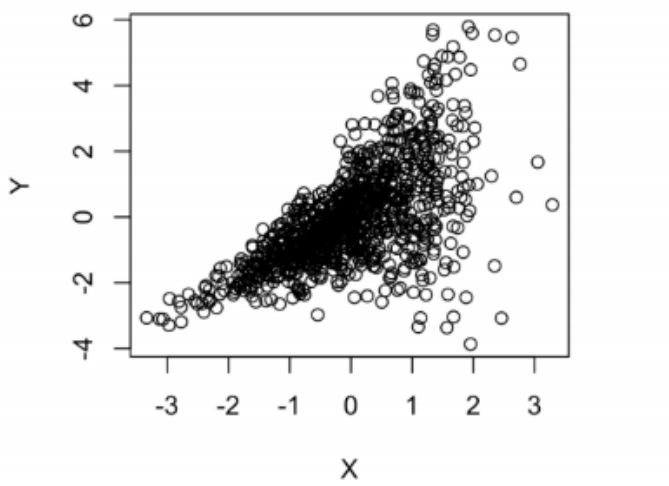
使用上述程序所构造的函数时，需要输入以下代码片段，定义预测函数和初始数据：

```
funs = lm.funs()# Or, e.g., lasso.funs(), rf.funs()
obj = conformal.pred(x, y, x0, alpha=0.1,
train.fun=funs$train, predict.fun=funs$predict)
```

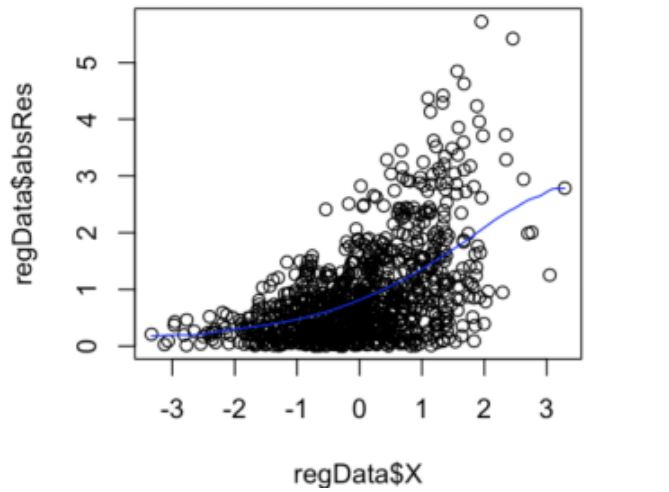
局部加权保形预测的简单代码

局部加权保形带允许带的长度在 X 上以一种解释异方差的方式变化。其思想是对条件扩散 (conditional spread) 进行建模，并利用对条件扩散的估计来缩放波段。我们使用局部 (核加权) 多项式回归对绝对残差的条件扩展进行建模，然后使用它来缩放波段。我们将修改这个例子，使异方差确实存在：

```
set.seed(1)
n <- 1000
X <- rnorm(n)#1000个服从标准正态分布的数据
Uhet <- rnorm(n, sd=exp(.5*X))#1000个服从正态分布，标准差为exp(.5*X)的数据
Y <- X + Uhet#线性回归模型存在异方差性，即随机误差项具有不同的方差
regData <- data.frame(X,Y)
plot(X, Y)
```



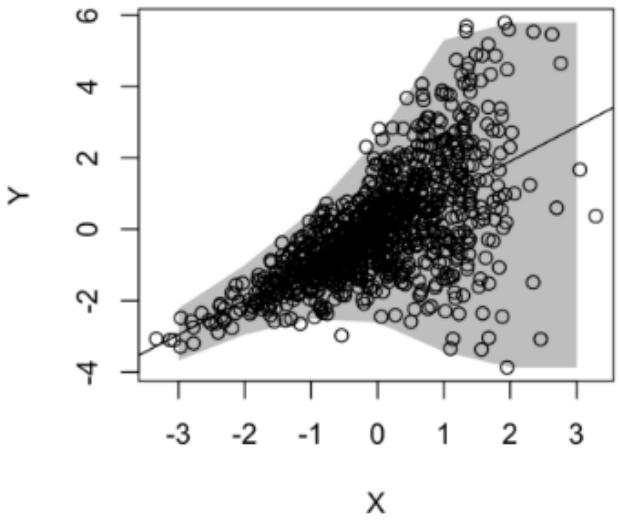

```
fitlm.het <- lm(Y~X, data=regData)#对X和Y做线性回归
regData$absRes <- abs(fitlm.het$residuals)#残差绝对值
library(locpol)#locpol(): 局部多项式估计，可以用来做非参数回归
#对于每个X，以该点为中心，按照预定宽度构造一个区间,以x为中心，根据残差给带宽加权
locFit <- locpol(absRes~X,data=regData,bw=2)#bw 平滑参数，带宽
plot(regData$X, regData$absRes)
#展示了X和线性回归残差的关系，随着x的增大，残差也越来越大
points(locFit$lpFit[,locFit$X],locFit$lpFit[,locFit$Y],type="l",col="blue")
```



现在我们应用局部加权保形预测：

```
nEval <- 200
yCand <- seq(from=min(Y), to=max(Y), length=nEval)
#定义局部加权保形预测函数
confPredict.lw <- function(y, Xin, bwArg){#bwArg带宽参数  #Xin是一个新的点
  nData <- nrow(regData)#原数据行数
  regData.a <- rbind(regData,c(Xin, y))#将原数据和c(Xin, y)按行合并
```

```
  fitlm.a <- lm(Y~X, data=regData.a)#用加了一行的新数据做线性回归
  regData.a$resOut.sc <- abs(fitlm.a$residuals)#加一列新的线性回归的残差绝对值
  xevalVec <- c(rev(seq(from=Xin, to=min(X), by= -0.25)),seq(from=Xin+.25, to=max(X),
  by=.25))#产生新的预测点,选取新的预测点Xin,以其为原点,生成部分数据,按从小到大排列
  #用加了一个新预测点的数据,以X为中心, 在给定带宽的情况下, 根据残差绝对值对带宽加权
  #xeval: 预测点的向量 预测多个新的点
  locFit <- locpol(resOut.sc~X,data=regData.a,bw=bwArg,xeval=xevalVec)
  resScale <- locFit$lpFit[,2][locFit$lpFit[,1]==Xin]#新加的点在locpol中对应的残差值
  #新的点在局部加权中对应的残差乘以 所有的点在1001个数据线性回归后对应的残差
  resOut <- regData.a$resOut.sc*resScale
  #新的点在新的回归中对应的残差乘以新的点在局部加权中对应的残差
  resOut_new <- regData.a$resOut.sc[length(resOut)]
  #找分位数点 #在resout中找小于resOut_new的点, 并求其均值
  pi.y <- mean(apply(as.matrix(resOut), 1,function(x){x<=resOut_new}))
  #ceiling取不小于 () 的最小整数
  testResult <- pi.y*(nData+1) <= ceiling(.975*(nData+1))
  return(list(testResult,resScale))
}
aug.over.X.lw <- function(Xval){
  #输出预测区间的最小值和最大值
  Cout.lw <- range(yCand[unlist(sapply(yCand,confPredict.lw,Xin=Xval, bwArg=2)[1,])])
  return(Cout.lw)
}
Xvals <- -3:3
#lapply把指定的待应用的函数应用于列表的每一个元素, 并返回列表结构的输出
#输入Xvals, 输出预测区间的最大值和最小值, 最大值一列, 最小值一列
augBands.lw <- matrix(unlist(lapply(Xvals, aug.over.X.lw)),ncol=2, byrow=T)
plot(X,Y, type="n")
polygon(c(Xvals, rev(Xvals)),
c(augBands.lw[,1],rev(augBands.lw[,2])),#包含多边形顶点坐标的向量
border=F,col="gray")#绘制多边形
points(X,Y)
abline(fitlm.het)
```



以上是我们运用 R 语言来实现保形预测的简单操作,下面我们也可以通过 python 语言实现保形预测。

6.5.2 Python 语言实践

完全保形预测的简单代码

我们可以根据完全保形预测的原理，通过构造不符合分数，确定分位点，得到保形预测区间。首先导入必要的包，并定义相应函数

```
import random
import sys
import statistics
eps = 0.20 #显著性水平
M = 1000
#定义不符合分数
def A(B, z):
    z0 = statistics.mean(B) #z0为预测值
    ans = abs(z0 - z) #取预测值减真实值的绝对值为不符合分数
    return ans
#完全保形预测 判断分位数点
def pz(z, zn):
    an = A(z, zn)    #an = z - zn
    z.append(zn)    #z后加上zn
    cnt = 0
    n = len(z)    #z的长度为n
    for i in range(n):
        z0 = z[:i] + z[i + 1:]    #z0等于z去掉zi
```

```
        a = A(z0, z[i])
        if a >= an:
            cnt += 1
    z.pop()   #删除z中最后一个值，即加进去的zn
    return cnt / n   #返回分位数点
def G(z):
    ans = []
    for zn in range(M):
        p = pz(z, zn)   #样本分位数点
        #如果样本分位数点大于显著性水平，则预测值zn包含在eps置信水平下的预测中
        if p > eps:
            ans.append(zn)
    return ans
#定义gen为符合期望为mu，标准差为sigma的正态分布的随机数
def gen():
    return int(random.gauss(mu, sigma))
```

将数据代入，得到保形预测区间

```
Z = [17, 20, 10, 17, 12, 15, 19, 22, 17, 19, 14, 22, 18, 17, 13, 12, 18, 15, 17]
mu = 500
sigma = 100
n = 2000
z = []
z.append(gen())
correct = total = 0
for i in range(n - 1):
    x = gen()
    print("%03d-%03d" % (min(G(z)), max(G(z))))
    if x in G(z):
        correct += 1
    total += 1
    print("%03d/%03d = %.0f" % (correct, total, correct / total * 100))
    z.append(x)
print(correct / total)
```

保形分类和保形回归的简单代码

另外我们也可以使用 python 调用 nonconformist 包，直接进行保形预测。下面是我们用该方法分别进行保形分类和保形回归的代码：

首先是保形分类，在这里，我们以使用支持向量机法对鸢尾花数据集进行分类为例：

```
from sklearn.datasets import load_iris
import numpy as np
from sklearn.svm import SVC
from nonconformist.cp import IcpClassifier
from nonconformist.nc import NcFactory
iris = load_iris()
idx = np.random.permutation(iris.target.size)   #对鸢尾花数据数量进行随机排列
```

```
# 将数据分为适当的训练集、校准集和测试集
idx_train, idx_cal, idx_test = idx[:50], idx[50:100], idx[100:]
model = SVC(probability=True)  # 建立预测模型
nc = NcFactory.create_nc(model)  # 建立不符合度量
icp = IcpClassifier(nc)  # 建立一个归纳保形分类预测器
# 用训练集拟合分类预测器
icp.fit(iris.data[idx_train, :], iris.target[idx_train])
# 用校准集校准分类预测器
icp.calibrate(iris.data[idx_cal, :], iris.target[idx_cal])
# 用测试集生成在95%置信度下的预测区间
prediction = icp.predict(iris.data[idx_test, :], significance=0.05)
# 打印预测结果
print(prediction[:5, :])
```

该预测结果是一个布尔型 numpy 数组，行数是测试集的数量，列数是类别数，每一行是一个代表在特定显著性水平下的类别是否包含在预测区间中的布尔向量。在这个例子中，对于给定的测试对象，我们也许会得到一个 $[True \quad True \quad False]$ 的向量，这意味着有 95% 的可能，True 对应的两个类别中有一个是正确的。

其次是保形回归，我们以使用随机森林法对波士顿数据集进行回归为例：

```
from sklearn.datasets import load_boston
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from nonconformist.cp import IcpRegressor
from nonconformist.nc import NcFactory
boston = load_boston()
idx = np.random.permutation(boston.target.size)
# 将数据分为适当的训练集、校准集和测试集
idx_train, idx_cal, idx_test = idx[:300], idx[300:399], idx[399:]
model = RandomForestRegressor()  # 建立预测模型
nc = NcFactory.create_nc(model)  # 建立不符合度量
icp = IcpRegressor(nc)  # 建立一个归纳保形回归器
# 用训练集拟合回归模型
icp.fit(boston.data[idx_train, :], boston.target[idx_train])
# 用校准集校准模型
icp.calibrate(boston.data[idx_cal, :], boston.target[idx_cal])
# 用测试集生成在95%置信度下的预测区间
prediction = icp.predict(boston.data[idx_test, :], significance=0.05)
# 打印预测结果
print(prediction[:5, :])
```

该预测结果是一个数值型 numpy 数组，行数是测试集的数量，列数是 2，每一行是一个代表区间下界和上界的向量，表示在特定显著性水平下的预测区间。在这个例子中，对于给定的测试对象，我们也许会得到一个数值向量 [26.47　39.45]，这意味着在 95% 置信度下得到的预测区间为 [26.47,　39.45]。

6.6 习题

1. 请分别列出完全保形预测和分裂保形预测方法的局限性。

2. 尝试编写加权保形预测的 Python 代码。

3. 编写一篇基于深度学习的保形预测程序。

4. 请使用 R 语言自带数据包中的 women 数据，编写使用分裂保形预测法对体重进行回归预测的 R 语言代码。要求：每次取出一组“身高-体重”数据作为测试集，其他数据作为训练集，对测试集数据中的体重进行预测。

5. 假设坐标轴中的点可分为两类，分别为：正样本点和负样本点。现存在训练集 (0, 3)，(2, 2)，(3, 3)，(-1, 1)，(-1, -1)，(0, 1)，其中 (0, 3)，(2, 2)，(3, 3) 为正样本点，(-1, 1)，(-1, -1)，(0, 1) 为负样本点。对于一个新的测试点 (0, 0)，请使用保形预测中的最近邻法判断该测试点属于哪一类别?

第三部分

稀疏学习

第七章　模型选择

稀疏学习（Sparse Learning）关注的是学习稀疏表示，即利用输入数据的稀疏性质来表示数据，以便更有效地表示和处理高维数据。稀疏学习通过发现和利用高维数据的冗余性提高模型的效率和性能，降低数据过拟合的风险。稀疏学习在图像处理、信号处理、自然语言处理和生物信息学等领域都有广泛应用。

监督学习和无监督学习直接对输入的训练数据进行操作，当训练数据维数较高，具有大量特征时，先进行稀疏学习，再进行监督学习和无监督学习可以带来更好的性能和更有效的特征表示。

本章和下一章将介绍两种主要的稀疏学习方法：模型选择（Model selection）和特征筛选（Feature Screening）。

7.1　简介

当我们考虑用模型来拟合数据，刻画数据的结构预测变量的时候，需要考虑两个基本问题：一是采用什么样的模型；二是采用哪些变量，即判断哪些变量是重要的，哪些变量是不重要的。由于统计推断和预测是在选定模型之后进行的，模型选择的好坏将直接影响到统计推断和预测的效果，因此模型选择问题得到了极大的重视和广泛的研究。

回归模型包含的预测变量并不是越多越好。首先，收集更多变量的数据会浪费时间和精力；其次，使用全部的预测变量会出现多重共线性的问题；最后，有些预测变量对响应变量并无影响，增加不必要的变量会带来估计噪声。因此我们要选择简单的且“足够好”的模型。

我们先介绍一个概念：方差—偏差平衡。当模型越来越复杂的时候，模型的预测能力也会越好。但问题是，模型只是在训练集上表现不错，而在测试集上表现比较差。具体来说：模型的偏差虽然小了，但是方差很大，即模型对“新”数据表现出“不稳定性”。这就是过拟合（over-fitting）了。模型过于复杂，连“噪声”也不放过，导致训练误差（training error）非常小，测试误差（test error）非常大。

如果一个模型对未知数据（通常是测试集的数据）的预测能力很好，我们就称它的泛化能力（generalization ability）很好，显然过拟合意味着模型的泛化能力很差。

如图7.1所示，当模型复杂度（degree of complexity）太低，训练误差和测试误差都很高，这个时候模型是欠拟合（under-fitting）的。随着模型复杂度的提升，测

试误差先减小后增大，但是训练误差不断地减小，最后过拟合。因此，控制模型复杂度可以实现方差—偏差的平衡。而控制模型复杂度可以转为选择最优模型。模型选择（model selection）就是从多个候选模型中，基于某评价准则和训练数据选出最优的模型。它的主要思想是通过训练数据来估计期望的测试误差，用数学语言描述模型选择就是：给定数据集 D，依据某个模型评价准则，从候选模型集合 S_m 中选出最优模型 S^*，即

$$S^* = \text{argmin}_m(\text{crit}(S_m; D))$$

其中 crit 表示模型评价准则。不同的评价准则对应不同的模型选择方法。

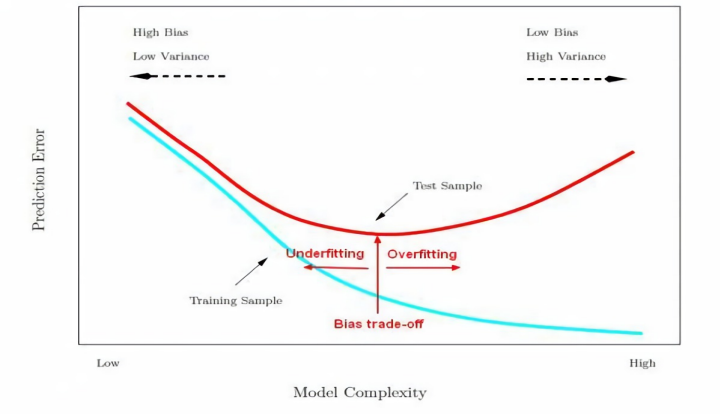


图 7.1 模型复杂度图

接下来介绍两种常用的模型选择方法：子集选择法和压缩估计法。在子集选择法中我们将介绍基于检验的模型选择方法和基于准则的模型选择方法，在压缩估计法中介绍 Lasso、非凸惩罚函数、群组变量选择方法和双层变量选择方法。

7.2 基于准则的方法

基于准则的方法将介绍 C_p 准则、AIC 准则、BIC 准则与交叉验证。

7.2.1 各种准则

通常训练集的均方误差比测试集的均方误差要低，而我们在选择模型时是希望得到一个具有最小测试误差的模型。并且训练误差随着更多变量加入模型中将会逐渐降低。因此，残差平方和 RSS 并不适用于对包含不同预测变量的模型进行模型选择。

为了基于测试误差选择最优模型，那么就需要去估计测试误差，通常有两种方法估计测试误差：

1、间接估计测试误差：根据过拟合导致的偏差对训练误差进行调整。

2、直接估计测试误差：通过交叉验证方法直接估计测试误差。

接下来就介绍几种适用于对包含不同预测变量的模型进行模型选择的方法，是间接估计测试误差的方法，包括 C_p 准则、AIC 准则、BIC 准则。

C_p 统计量

C_p 统计量的思想是通过在训练集 RSS 的基础上增加惩罚项，来调整训练误差倾向于低估测试误差这一现象。具体来说就是训练集 RSS 随着模型中预测变量的增加而降低，但是此时测试集的 RSS 要比训练集的 RSS 要高，所以就在训练集 RSS 的基础上增加一项，使得该增加项的大小随着模型中预测变量个数的增加而增加。采用最小二乘法拟合一个包括 k 个预测变量的模型，C_p 值计算如下：

$$C_p = \frac{1}{n}\left(\mathrm{RSS} + 2k\hat{\sigma}^2\right)$$

其中，$\hat{\sigma}$ 是响应变量观测误差的方差的估计值，k 是模型中变量的个数。C_p 统计量在训练集 RSS 的基础上增加惩罚项 $2k\hat{\sigma}^2$，这可以看作是在考虑拟合残差同时，依变量个数施加的惩罚。显然，惩罚项随模型中变量个数的增加而增大，用于调整由于变量个数增加而不断降低的训练集 RSS。如果 $\hat{\sigma}$ 是观测误差方差的无偏估计，则 C_p 是测试均方误差的无偏估计。C_p 值越小，表示模型的准确性越高。

AIC 准则

AIC 准则是衡量统计模型拟合优良性的一种准则，由日本统计学家赤池弘次 [81] 在 1974 年提出，因而又叫赤池信息准则。它是权衡估计模型复杂度和拟合数据优良性的准则。AIC 准则的定义公式为:

$$\mathrm{AIC} = 2k - 2\ln L$$

其中 k 是模型参数个数，L 是极大似然函数。由于似然函数的值越大得到的估计量就越好，因此 $-2\ln(L)$ 越小越好，该项刻画了模型的精度或拟合程度，又考虑到模型包含的预测变量的个数不能太多，故加入 $2k$ 对模型参数个数进行惩罚。即一方面要使似然函数值较大，另一方面通过惩罚部分限制模型的复杂度。因此，AIC 是寻找可以最好地解释数据但包含最少自由参数的模型。

假设模型的随机误差服从独立正态分布。此时 $-2\ln(L) = n\ln\left(\frac{\mathrm{RSS}}{n}\right) +$ 常数，那么 AIC 变为:

$$\mathrm{AIC} = 2k + n\ln\left(\frac{\mathrm{RSS}}{n}\right)$$

其中 n 为样本量，k 为参数个数，RSS 为残差平方和。

BIC 准则

Schwarz [82] 为了能够克服在大样本数据情形下 AIC 准则容易失效的不足，提出了贝叶斯信息准则。其定义公式为：

$$\mathrm{BIC} = k\ln(n) - 2\ln L$$

假设条件是模型的误差服从独立正态分布，BIC 准则的公式为：

$$\mathrm{BIC} = k\ln(n) + n\ln\left(\frac{\mathrm{RSS}}{n}\right)$$

其中，n 为样本容量。当样本容量很大时，在 AIC 准则中的残差平方和的所在项会受到样本容量影响而放大，而参数个数的惩罚因子却未随样本容量的变化而变化。因此当样本容量很大时，使用 AIC 准则选择的模型不收敛到真实的最优模型。事实上，它通常比真实模型所含的未知参数个数要多。为引入对预测变量过多的惩罚，BIC 准则把把 AIC 中的 2 换成了 $\ln(n)$。所以，当 $n > e^2$ 时，BIC 统计量相比于 AIC 统计量给包含多个变量的模型施以较重的惩罚，所以与 AIC 统计量相比，BIC 得到的模型规模更小。如果真实模型是有限维参数，BIC 准则会表现得更好。

AIC 和 BIC 比较

AIC 和 BIC 的原理是不同的，AIC 是从预测角度，选择一个好的模型用来预测，BIC 是从拟合角度，选择一个对现有数据拟合最好的模型，从贝叶斯因子的解释来讲，就是边际似然最大的那个模型。

相同点：构造这些统计量所遵循的统计思想是一致的，就是在考虑拟合残差的同时，依预测变量个数施加“惩罚”。

不同点：BIC 的惩罚项比 AIC 大，考虑了样本个数，可以防止模型精度过高造成的模型复杂度过高。AIC 和 BIC 前半部分是惩罚项，当 $n \geq 8$ 时，$k\ln(n) \geq 2k$，所以，BIC 相比 AIC 在大数据量时对模型参数惩罚得更多，导致 BIC 更倾向于选择参数少的简单模型。

7.2.2 交叉验证

交叉验证（cross-validation）它是一种没有任何前提假定直接估计泛化误差的模型选择方法。由于没有任何假定，它可以应用于各种模型选择中。它的基本思想是将数据分为两部分，一部分数据用来进行模型的训练，通常我们叫做训练集，另一部分数据用来测试训练生成模型的误差，我们叫做测试集。由于两部分数据的不同，

泛化误差的估计是在新的数据上进行，这样的泛化误差的估计可以更接近真实的泛化误差。在数据足够的情况下，我们可以很好估计出真实的泛化误差，但是在实际应用中，往往只有有限的数据可用，我们必须对数据进行重用，即对数据进行多次切分来得到好的估计，自从交叉验证提出以后，人们提出了不同的数据切分方式，因此产生了多种形式的交叉验证方法，下面我们介绍一些主要的交叉验证方法。

验证集方法

用给定的观测集估计使用某种模型拟合所产生的测试误差，一种最简单、直接的方法是验证集方法。首先，我们将给定的样本观测集数据随机地分为不重复的两部分，然后用训练集来训练模型，在测试集上验证模型及参数。接着，我们再把样本打乱，重新选择训练集和测试集，继续训练数据和检验模型。最后我们选择损失函数评估最优的模型和参数。这就是验证集方法的基本原理。下面以 5×2 交叉验证为例介绍训练集和测试集的选择方法。

5×2 交叉验证法的主要思想是将数据集 V 平均分为两部分 $V_1^{(1)}$ 和 $V_1^{(2)}$，首先用 $V_1^{(1)}$ 作为训练集，$V_1^{(2)}$ 作为验证集，然后互换角色，用 $V_1^{(2)}$ 作为训练集，$V_1^{(1)}$ 作为验证集，这样就得到了第一折，即第一次对折。为了得到第二折，我们将数据集重新打乱并划分为新的两个等份 $V_2^{(1)}$ 和 $V_2^{(2)}$。将 $V_2^{(1)}$ 作为训练集，$V_2^{(2)}$ 作为验证集，然后对调两者的角色，得到第二折，重复以上作法五次，会得到十个训练集和验证集。当然我们可以进行超过五次的对折得到更多的训练集和验证集，但在五次对折之后，各个集合共享了许多样本，使得计算出来的泛化误差估计变得相互依赖无法增加新的信息。

可以看出验证集方法的优点：处理简单，只需随机把原始数据分为两组即可；并且测试集和训练集是分开的，避免了过拟合的现象。但是其缺点也有两个部分：

1、由于是随机的将原始数据分组，所以最后验证集的分类准确率与原始数据的划分有很大的关系，最终模型的确定将强烈地依赖于训练集和验证集的划分方式，测试误差会根据划分方式的不同而变化。

2、由于只使用部分数据进行训练，得到的结果并不具有说服性。在实际应用中，当用于模型训练的观测数据越多时，模型的效果就会越理想。而验证集方法无法充分利用所有的观测数据，对模型的效果会产生影响。

接下来介绍交叉验证法，针对验证集方法的上述问题进行了改进。

留一交叉验证法

留一法（Leave-One-OutCross-Validation，LOOCV）的基本思想是每次从个数为 n 样本集中取出一个样本作为验证集，剩下的 $n-1$ 个样本作为训练集，重复进行多次，依次取遍所有 n 个数据作为验证集。也就是就是每次只留下一个样本做测试集，其它样本做训练集。操作步骤如下：

1、如果设原始数据有 n 个样本，即 $\{(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_i, Y_i), \ldots (\boldsymbol{X}_n, Y_n)\}$，那么以每个样本单独作为验证集，其余的 $n-1$ 个样本作为训练集进行训练，我们会得到 n 个模型。

2、对于每个在训练集上拟合的模型而言，我们考虑其相应的验证集进而得到该模型的均方误差。

（1）回归问题：$\text{MSE}_i = (Y_i - \hat{Y}_i)^2$ (1.1.8)

（2）分类问题：$\text{Err}_i = I(Y_i \neq \hat{Y}_i)$ (1.1.9)

3、将 n 个模型的 n 个均方误差取均值后，我们得到测试均方误差的 LOOCV 估计：

（1）回归问题：$\text{CV}_{(n)} = \dfrac{1}{n}\sum_{i=1}^{n}\text{MSE}_i$ (1.1.10)

（2）分类问题:$\text{CV}_{(n)} = \dfrac{1}{n}\sum_{i=1}^{n}\text{Err}_i$ (1.1.11)

LOOCV 是交叉验证方法中最常见的一种方法，相比于验证集方法，它有如下特点：

优点：每一回合中几乎所有的样本都用于训练模型，这样评估所得的结果比较可靠；实验中没有随机因素，整个过程是可重复的。

缺点：计算成本高，当 n 非常大时，计算耗时。

于是就有了一种折中的方法——K 折交叉验证法（K-fold CV）。

K 折交叉验证法

K 折交叉验证法（K-fold Cross Validation, K-CV）的原理:

1、将原始数据分成 K 组（一般是均分），或者说折（fold）；然后，让每个子集数据分别做一次验证集，其对应的剩余 $K-1$ 组子集数据作为训练集，这样会得到 K 个模型。

2、同 LOOCV 方法一样，基于 K-CV 的实验建立的 K 个模型，分别在训练集上拟合模型，再将拟合的模型用于保留的验证集上计算均方误差。

（1）回归问题:$\text{MSE}_k = \dfrac{1}{|G_k|}\sum_{i\in G_k}\left(Y_i - \widehat{Y}_i\right)^2$ (1.1.12)

（2）分类问题:$\text{Err}_i = I(Y_i \neq \hat{Y}_i)$ (1.1.13)

3、用这 K 个模型最终的验证集的均方误差的平均数作为此 K-CV 的估计。

（1）回归问题：$\text{CV}_{(K)} = \dfrac{1}{K}\sum_{k=1}^{K}\text{MSE}_k$ (1.1.14)

（2）分类问题：$\text{CV}_{(K)} = \dfrac{1}{K}\sum_{k=1}^{K}\text{Err}_k$ (1.1.15)

在实作上，K 要够大才能使各回合中的训练样本数够多。根据经验一般选取 $K=5$ 或 $K=10$，这两个取值会使测试误差的估计不会有过大的偏差或方差，$K=$

10 是相当足够了。

K 折交叉验证法有如下特点：

与 LOOCV 方法的关系： L0OCV 方法就是 K 折交叉验证法当 $K = n$ 时的一个特例。由于 $K < n$，K 折交叉验证法拟合模型次数 K 便小于 LOOCV 方法，这样就降低了计算成本。

优点： K-CV 使得每一个样本数据都既被用作训练数据，也被用作测试数据，可以有效的避免过拟合以及欠拟合的发生，最后得到的结果也比较具有说服性。

缺点： K 折交叉验证法的缺点在于 K 的确定。一方面，K 越大——样本划分的组多——训练集包含的观测数据多——拟合的偏差越小。特例是当 $K = n$ 时也就是 LOOCV, 我们能够得到一个近似无偏的测试误差估计。另一方面，K 越大就意味着每次用于拟合模型的训练集的观测数据重合度高，模型也就更相似。还是考虑特殊情况 LOOCV，每一次训练的观测数据几乎是相同的，拟合的结果之间是高度（正）相关的，由于许多高度相关的量的均值要比相关性相对较小的量的均值具有更高的波动性——方差也将更大。

与 C_p 准则、AIC 准则、BIC 准则、调整 $\bar{R}^2$ 相比，这些方法的优势在于直接给出了测试误差的直接估计，符合我们最初选择测试误差最小的模型的目的。

7.3 基于检验的方法

在基于检验的方法中介绍两种最常见的方法：最优子集法和逐步选择法。

7.3.1 最优子集法

对于含有 p 个预测变量组成的集合来说，最优子集选择即对 p 个预测变量的所有可能组合分别使用最小二乘回归进行拟合，即对含有一个预测变量的模型，拟合 p 个模型；对含有两个预测变量的模型，拟合 $p(p-1)/2$ 个模型，依此类推最后我们一共拟合了 $C(p,0) + C(p,1) + \cdots\cdots + C(p,p-1) + C(p,p)$ 即 2^p 个模型，再根据指标从所有可能模型中选取一个最优的模型。最优子集法具体过程可以概括为：

1、记不包含预测变量的零模型为 M_0。

2、对 $k = 1, \cdots, p$

（1）从 p 个预测变量中任意选择 k 个，拟合 $C(p,k)$ 个模型。

（2）在 $C(p,k)$ 个模型中选择最优的一个（RSS 最小或 $\bar{R}^2$ 最大），记为 M_k。

3、根据交叉验证预测误差、C_p、AIC、BIC 或者修正的决定系数 $\bar{R}^2$ 等指标，从 $M_0, \cdots, M_p$ 个模型中选择一个最优模型。

步骤 2 先在相同数量预测变量的模型中选择一个最优模型（内循环）；步骤 3 在不同数量预测变量的模型中进行模型选择（外循环）。通过外循环选择最优模型，需要改变准则。因为随着模型中预测变量数目增加，这 $p+1$ 个模型的 RSS 会下降。如果我们仅依据 RSS 进行模型选择的话，最后选出来的最优模型将包含所有变量。因此对于变量个数相同的模型，我们可通过 RSS 来选择最优模型；对于变量个数不同的模型，我们使用交叉验证预测误差、C_p、AIC、BIC 或者 $\bar{R}^2$ 等指标进行模型选择。例如，根据修正决定系数 $\bar{R}^2$ 的定义：

$$\bar{R}^2 = 1 - \frac{SSE/(n-k-1)}{SST/(n-1)}. \tag{7.3.1}$$

我们知道，$\bar{R}^2$ 越大，模型拟合程度越好。这里引入了预测变量数量 k 从而对预测变量增加进行了惩罚，即模型中包含的预测变量 k 增加时会抑制 $\bar{R}^2$ 的增加，使选出来的模型具有更小的测试误差。通俗的来讲，修正决定系数并不会“偏爱”预测变量较多的模型。

显然，最优子集法运用了穷举的思想。其优势很明显：简单直接，遍历所有可能的情况，最后的选择一定是最优的；劣势也很明显：当 p 越大时，可选模型数量在迅速增加，计算量明显增大，降低计算效率。因此，最优子集法只适用于 p 较小的情况。一般来说，当 $p<10$ 时，我们可以考虑最优子集法。

7.3.2 逐步选择法

最优子集法在 p 很大时运算效率低，并且随着搜索空间的增大找到的模型泛化能力差。因此，本章引入逐步选择法避免上述缺点。

在逐步选择法中，模型会在原有变量的基础上一次增加一个显著的预测变量或剔除一个不显著的预测变量，直到既没有显著的预测变量选入回归方程，或也没有不显著的预测变量从回归方程中剔除为止。最优子集法与逐步选择法的区别在于，最优子集法可选择任意 k 个变量进行建模，而逐步选择法只能在之前所选的 k 个变量的基础上建模。常用的逐步选择法包括向前逐步选择、向后逐步选择、双向选择。

向前逐步选择

向前逐步回归的思想是由少到多，逐个引入预测变量。我们首先考虑一个不包含任何预测变量的零模型，从零模型开始，依次引入一个预测变量，直至没有可引入的变量为止。所谓没有可引入的变量是指要添加的任何新变量都不会使模型有所改进。

具体来说就是从零号模型 M_0 开始，这个模型只有截距项而没有任何预测变量。然后，将 p 个预测变量依次加入模型中（每次只加入一个预测变量，考虑 p 次，得到 $p+1$ 个模型），保留 RSS 最小或 $\bar{R}^2$ 最大的那个预测变量，此时模型含有一个预

测变量，记为 M_1。然后此的基础上，将剩余的 $p-1$ 个预测变量依次分别加入，仍然保留 RSS 最小或 $\bar{\mathrm{R}}^2$ 最大的那个预测变量，此时得到的模型含有 2 个预测变量记为 M_2。这样重复操作，直至包含 p 个预测变量的模型 M_p。最后根据交叉验证预测误差、C_p、AIC、BIC 或者修正的 $\bar{\mathrm{R}}^2$ 等指标，从 $p+1$ 个最优模型中选择一个最优模型。

向前逐步回归的算法过程如下：

1、记不包含任何预测变量的零模型为 M_0。

2、对 k=0,1,2,$\cdots$,$p-1$：

（1）基于上一步选取的最优模型 M_k 的 k 个预测变量，将剩余的 $p-k$ 个预测变量分别加入，这样就得到 $p-k$ 个模型；

（2）在上述 $p-k$ 个模型中选择 RSS 或 $\bar{\mathrm{R}}^2$ 最高的模型作为最优模型，记为 M_{k+1}。

3、进一步使用交叉验证误差、C_p、AIC、BIC 或者修正的决定系数 $\bar{\mathrm{R}}^2$ 等指标，从 $p+1$ 个最优模型中选择一个最优模型。

与最优子集选择法要对 2^p 个模型进行拟合不同，向前回归只需要对零模型以及第 k 次迭代所包含的 $p-k$ 个模型进行拟合，其中 $k=0,1,2,\cdots,p-1$。相当于拟合 $\sum_{k=0}^{p-1}(p-k)=p(p+1)/2$ 个模型，特别的，当 $p=30$ 时，最优子集需要拟合 1073741824 个模型，而向前逐步回归只需要拟合 465 个模型。

但是我们需要注意的是向前逐步回归无法保证最后得到的模型是 2^p 个模型中最优的。例如在给定的包含三个变量 X_1、X_2、X_3 的数据集中，最优的单变量模型只包含 X_2，最优的双变量模型只包含 X_1、X_3，且通过一些指标得出最优双变量模型在 $2^3=8$ 个模型中是最优的。则通过向前逐步回归无法得到该最优模型，因为 M_1 包含变量 X_2，而 M_2 只能包含 X_2、X_3 或者 X_2、X_1，而无法得到包含 X_1、X_3 的双变量模型。

下面我们用一个例子说明向前回归的具体过程。

假设 $\{X_1,X_2,\cdots,X_p\}$ 为待选择的预测变量的集合，首先可以将每一个单一变量看作是一个子集，构成 p 个模型，在这 p 个模型中选择 RSS 最小或 $\bar{\mathrm{R}}^2$ 最大的为最优模型，在此不妨假设 $\{X_1\}$ 是最优的单一变量模型；接下来我们在上一步的基础上添加一个变量，得到 $p-1$ 个包含两个预测变量的模型，再基于上述准则选出最优模型，不妨假设 $\{X_1,X_2\}$ 为最优，且优于 $\{X_1\}$，则第二步选择结束；依次类推，直到第 $k+1$ 轮选择的最优模型拟合效果不如第 k 轮选出最优模型，则整个子集选择过程结束，并将第 k 轮选出的模型作为最后的最优模型。

向后逐步选择

与向前逐步回归法的选择方向相反，向后逐步回归法以一个包含全部预测变量的全模型为起点，逐次迭代，每次剔除一个对模型效果最不利的冗余变量，直到继

续剔除变量却不能使模型质量有所改进为止。具体来说就是从包含全部 p 个预测变量的模型 M_p 开始，然后一个个地移除 p 个预测变量，保留 RSS 最小或 $\bar{\mathrm{R}}^2$ 最大的那个模型，保留下来的模型是包含 $p-1$ 个预测变量的模型，记为 M_{p-1}，基于该模型继续移除剩余的 $p-1$ 个预测变量。这样重复操作，直至包含 0 个预测变量的模型 M_0。然后根据交叉验证预测误差、C_p、AIC、BIC 或者修正的 $\bar{\mathrm{R}}^2$ 等指标，从 $p+1$ 个最优模型中选择一个最优模型。

向后逐步回归的算法过程如下：

1、记不包所有预测变量的全模型为 M_p。

2、对 k=$p, p-1, \cdots, 1$：

（1）基于上一步选取的最优模型的 k 个预测变量，依次逐个剔除，这样就可以构建 k 个模型；

（2）在上述 k 个模型中选择最优模型，记为 M_{k-1}。

3、进一步使用交叉验证预测误差、C_p、AIC、BIC 或者修正的决定系数 $\bar{\mathrm{R}}^2$ 等指标，从 $p+1$ 个最优模型中选择一个最优模型，为最后得到的最优模型。

同样，向后逐步回归共需对 $1+p(p+1)/2$ 个模型进行搜索，比最优子集选择更高效。但是，向后逐步回归无法保证得到的模型是包含 p 个预测变量子集的最优模型。这是因为向后逐步回归同样都依赖于上一步的迭代结果。此处需要注意，因为向后逐步回归法是从全模型开始的，如果使用最小二乘法进行拟合，需要满足 $n > p$ 的条件。而向前逐步回归法则不需要满足这一条件。当预测变量向量的维度 p 比较高时，应优先选择向前逐步回归法。

向前逐步回归和向后逐步回归都属于贪心算法，能够局部达到最优，但是从全局来看不一定是最优的。向前逐步回归虽然每一次都能选取最显著的一个预测变量，但在实际情况下，很可能有的预测变量在开始时是显著的，但是在其余预测变量添加进去之后，它就变得不显著了，但向前逐步回归此时不会剔除该变量。而向后逐步回归则很有可能会遗漏一些很重要的变量，比如刚开始变量 X_1 不显著，但是在剔除 m 个变量后其变得显著，此时向后逐步回归并不会重新加入这个变量。总的来说这些问题是由于向前逐步回归不会剔除已经加入进来的变量，向后逐步回归不会加入已经剔除的变量导致的，下面介绍的双向选择会解决这些问题。

双向选择

双向选择是向前和向后逐步选择法的结合。每次引入一个变量，但与此同时也会剔除对模型没有贡献的变量。引入一个变量后，我们首先通过 F 检验，验证新变量是否会使得模型发生显著性变化。若原有变量由于新变量的加入变得不再显著时，剔除此变量。若模型依旧显著，再对所有变量进行 t 检验，剔除不显著变量。直到既没有新的显著的预测变量加入回归方程，也没有原有的不显著的预测变量从回归方程中剔除为止，最终我们得到一个最优模型。在这个过程中，某一特定预测变量可能会被反复引入和删除。该方法在试图达到最优子集选择效果的同时也保留了向前

和向后逐步回归在计算效率上的优势。

7.4 正则化方法

子集选择方法通过保留预测变量的一个子集，并舍弃其它变量，产生一个简单的具有一定解释性的模型。然而子集选择法也有缺点，主要体现在两个方面：首先，在进行最优变量子集筛选的过程时离散，通常具有很高的变异性，不够稳定；其次，忽略了变量选择过程中的随机误差。

模型选择的另一个典型方法是正则化，正则化就是通过对模型参数进行调整（数量和大小），降低模型的复杂度，以达到可以避免过拟合的效果。具体的解决方法是在模型的损失函数中加入正则项。如果不加入正则项，我们的目标是最小化损失函数；加入正则项以后，变成最小化损失同时复杂度不要太高，这一方法称为结构风险最小化。通过惩罚函数约束模型的回归系数，同步实现变量选择和系数估计，模型估计是一个连续的过程，更正了子集选择法的缺陷。

在详细介绍这一方法之前，我们先介绍目标函数和范数的概念。

1、目标函数

求解优化问题的关键就是最小化目标函数。假设样本矩阵为 $\mathbf{X}=(\boldsymbol{X}_1,\cdots,\boldsymbol{X}_n)^{\mathrm{T}}$ 是 $n\times p$ 矩阵，因变量记为 $\boldsymbol{Y}=(Y_1,\cdots,Y_n)^{\mathrm{T}}$。我们考虑每组样本中的预测变量为 $\boldsymbol{X}_i$，响应变量为 Y_i，p 维参数为 $\boldsymbol{\beta}$。目标函数的一般形式为：

$$\min_{\boldsymbol{\beta}} L(\boldsymbol{\beta})=\min_{\boldsymbol{\beta}}\left\{Q(\boldsymbol{\beta}|\boldsymbol{Y},\mathbf{X})+\sum_{j=1}^{p}p_{\lambda}(|\beta_j|)\right\}$$

其中 $Q(\boldsymbol{\beta}|\boldsymbol{Y},\mathbf{X})$ 是损失函数，不同模型的损失函数形式不同，线性模型的损失函数为最小二乘函数、Logistic 回归的损失函数为极大似然函数的负向变换。

$p_{\lambda}(|\beta_j|)$ 为正则化项，所谓正则化项，也叫惩罚项，描述的其实是模型的复杂度，模型的复杂度越高，过拟合的风险也就越大。惩罚项的惩罚力度越大时，模型的复杂度越低。例如前面章节讲到的岭回归，是在极小化损失函数的基础上通过对系数添加 L_2 范数惩罚来进行系数的连续收缩的，它是 Hoerl 和 Kennard [83] 于 1970 年提出的，是统计学著名的系数收缩方法之一。

岭回归的目标函数是：

$$\min_{\boldsymbol{\beta}} L(\boldsymbol{\beta})=(\boldsymbol{Y}-\mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{Y}-\mathbf{X}\boldsymbol{\beta})+\lambda\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$$

其中第一项是残差平方和 RSS（线性模型的损失函数），第二项是 L_2 正则化项，λ 为调节参数，$\lambda\geq 0$，为非负数，其作用是控制损失函数和正则化项对回归系数估计的相对影响程度，$\lambda=0$ 时，岭回归估计值与最小二乘估计值相同，λ 越大，则为了使目标函数最小，回归系数 $\boldsymbol{\beta}$ 的估计值就越接近于 0。

2、范数

我们都知道，函数与几何图形往往是有对应的关系，这个很好想象，特别是在三维以下的空间内，函数是几何图像的数学概括，而几何图像是函数的高度形象化。但当函数与几何超出三维空间时，就难以获得较好的想象，于是就有了映射的概念，进而引入范数的概念。

当我们有了范数的概念后，我们就可以引出两个向量的距离的定义，这个向量可以是任意维数的。通过距离的定义，进而我们可以讨论逼近程度，从而讨论收敛性、求极限。

L_0-范数：即向量中非零元素的个数。其公式为：

$$\|\mathbf{X}\|_0 = \sum_i \mathbb{I}(X_j \neq 0)$$

L_1-范数：即向量各元素绝对值之和。其公式为：

$$\|\mathbf{X}\|_1 = |x_1| + |x_2| + \ldots + |x_n|$$

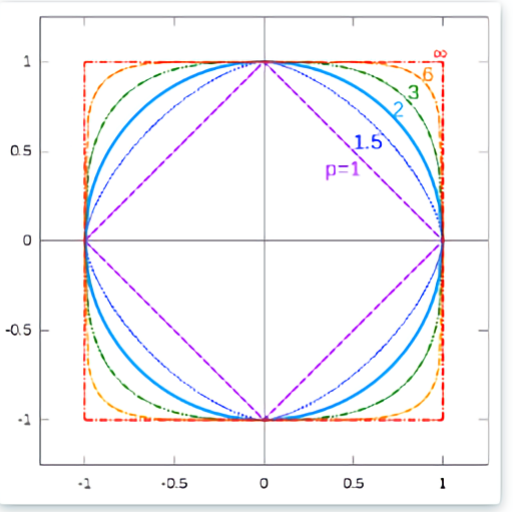
L_2-范数：欧几里得范数，常用计算向量长度，即向量元素绝对值的平方和再开方：

$$\|\mathbf{X}\|_2 = (|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2)^{1/2}$$

L_p-范数：即向量元素绝对值的 p 次方和的 $1/p$ 次幂：

$$\|\mathbf{X}\|_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{1/p}$$

下图展示了直角坐标系中 p 取不同的值时各个范数下单位向量终点的轨迹：



下图表示了 p 从无穷到 0 变化时，三维空间中到原点的距离（范数）为 1 的点构成的图形的变化情况，以常见比如 $p = 2$ 时为例，此时的范数也即欧氏距离，空间中到原点的欧氏距离为 1 的点构成了一个球面：

其中，L_1 范数正则化、L_2 范数正则化都有助于降低过拟合风险，我们在此主要运用 L_1 和 L_2 范数正则化，或者二者相结合的方法。对于线性回归模型，使用 L_1 正则化的模型叫做 Lasso 回归，使用 L_2 正则化的模型叫做 岭回归（Ridge 回归），两者相结合的方法叫做弹性网。

7.4.1 Lasso 回归

与岭回归类似，Lasso 回归是在极小化残差平方和的基础上通过对系数添加 L_1 范数惩罚来收缩系数的，1996 年由 Robert Tibshirani [130] 首次提出，全称为：Least absolute shrinkage and selection operator。该方法是一种压缩估计，当 λ 充分大时，可以把某些待估系数精确的收缩到零，因而特别适用于参数数目的缩减与参数的选择。

由于该方法以岭回归为基础，增加了稳定性。Lasso 的核心是稀疏性，稀疏性的优势在于它可以解释拟合模型，并且计算简单。

Lasso 回归的目标函数是：

$$\min_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) = (\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_1$$

岭回归和 Lasso 的比较

岭回归和 Lasso 回归的主要区别就是在正则化项，岭回归用的是 L_2 正则化，而 Lasso 回归用的是 L_1 正则化。L_2 正则能够有效的防止模型过拟合，解决非满秩下求逆困难的问题；L_1 正则化最大的特点是能稀疏矩阵，进行庞大预测变量数量下的预测变量选择。

如果把二者目标函数最小化看成是有约束的极值问题，那么岭回归可以表达为：

$$\min_{\boldsymbol{\beta}}(\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})$$

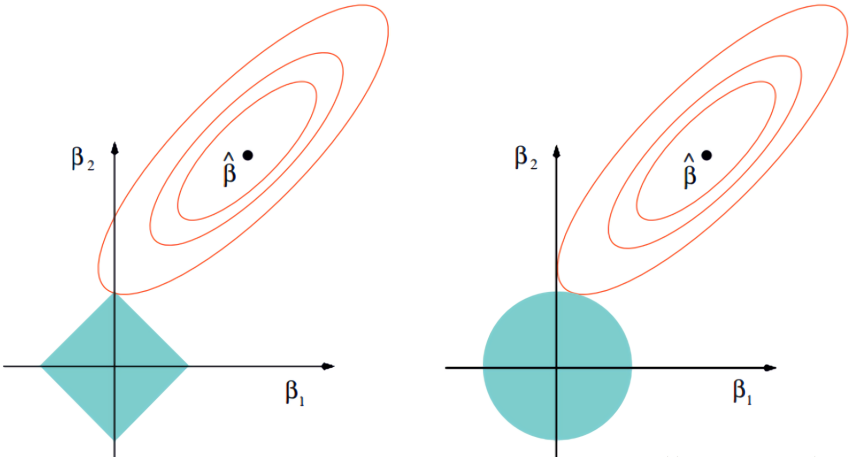
$$s.t. \quad \|\boldsymbol{\beta}\|_2^2 \leq t$$

Lasso 回归可以表达为：

$$\min_{\boldsymbol{\beta}}(\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta})$$

$$s.t.\ \ \|\boldsymbol{\beta}\|_1 \leq t$$

因此 Lasso 回归和岭回归可以看作是两个同样目标函数但在不同的约束区域下求解的问题。



如图，二维空间为例，不添加正则项的目标函数可以用一圈圈等值线表示，约束区域对应图中的蓝色区域。等值线和约束域的切点就是目标函数的最优解。岭方法对应的约束域是圆，其切点只会存在于圆周上，不会与坐标轴相切，则在任一维度上的取值都不为 0，因此没有稀疏；对于 Lasso 方法，其约束域是正方形，会存在与坐标轴的切点，使得部分回归系数变为零，因此很容易产生稀疏的结果。

所以，Lasso 方法可以达到变量选择的效果，将不显著的变量系数压缩至 0，在估计参数的同时实现了变量选择；而岭方法虽然也对原本的系数进行了一定程度的压缩，但是任一系数都不会压缩至 0，只有压缩功能，没有选择功能，最终模型保留了所有的变量。Lasso 回归的计算量将远远小于岭回归，但无论对于岭回归还是 Lasso 回归，它们的本质都是通过调节 λ 来实现模型偏差和方差的平衡调整。

7.4.2 非凸惩罚函数回归——SCAD 和 MCP

SCAD

Lasso 总是有偏估计，这是因为 Lasso 的调节参数 λ 对所有参数一视同仁，导致大的系数被过分压缩，带来较大的估计偏差，而且 Lasso 的结果具有不稳定性。由 Fan [84] 在 2001 年提出了SCAD（Smoothly Clipped Absolute Deviation Penalty）较好的弥补了 Lasso 的不足，更进一步地，该方法具有 Oracle 性质，使预测效果与真实模型别无二致。

Fan 和 Li 指出，一个好的惩罚函数应该使得最后得到的估计量满足三个性质：1. 无偏性。即当未知参数的真实值很大时，估计值应当近乎无偏。2. 稀疏性。即可

以自动地将很小的估计系数压缩为 0，降低模型的复杂度。3. 连续性。即估计的系数应当是连续的，保证模型预测的稳定性。

该方法也是在损失函数的基础上施加惩罚项，例如基于二次损失的惩罚最小二乘目标函数可以表示为：

$$L(\boldsymbol{\beta}) = \frac{1}{2n}\|\boldsymbol{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \sum_{j=1}^{p} p_\lambda\left(|\beta_j|\right) \tag{7.4.1}$$

SCAD 的惩罚函数为：

$$p_\lambda(|\beta_j|;\alpha) = \begin{cases} \lambda|\beta_j| & \text{若} \quad 0 \le |\beta_j| < \lambda \\ -\dfrac{|\beta_j|^2 - 2\alpha\lambda|\beta_j| + \lambda^2}{2(\alpha-1)} & \text{若} \quad \lambda \le |\beta_j| < \alpha\lambda \\ (\alpha+1)\lambda^2/2 & \text{其它} \end{cases}$$

其中，α 和 λ 是两个调节参数。

考虑 $\beta_j \in [0, +\infty)$ 的情况，SCAD 惩罚函数的导数为：

$$p'_\lambda(|\beta_j|;\alpha) = \begin{cases} \lambda, & \beta_j < \lambda \\ \dfrac{\alpha\lambda - \beta_j}{\alpha - 1}, & \lambda < \beta_j < \alpha\lambda \\ 0 & \beta_j > \alpha\lambda. \end{cases}$$

可以看出 SCAD 惩罚函数的导数在原点附近与 Lasso 相同，离原点越远导数值越小，直至为零，也就是说，SCAD 方法对较大的参数施加较少的惩罚，当参数大于 $\alpha\lambda$，不施加惩罚。因此显著的变量更容易被选入模型。

MCP

2010 年，Zhang [85] 提出的 MCP 方法保留了 SCAD 的渐进无偏优点，其惩罚函数同样是分段函数：

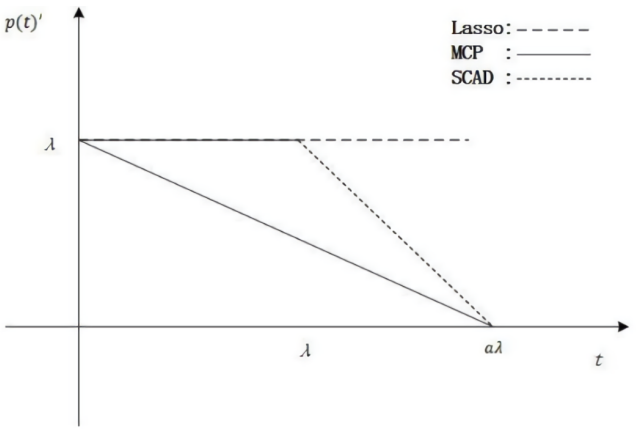
$$p_\lambda(|\beta_j|;\lambda,\alpha) = \begin{cases} \lambda|\beta_j| - \dfrac{|\beta_j|^2}{2\alpha}, & \text{若} \quad |\beta_j| \le \alpha\lambda \\ \dfrac{\alpha\lambda^2}{2}, & \text{若} \quad |\beta_j| > \alpha\lambda \end{cases}$$

其惩罚函数的导数为：

$$p'_\lambda(|\beta_j|;\lambda,\alpha) = \begin{cases} \lambda - \dfrac{|\beta_j|}{\alpha} & \text{若} \quad |\beta_j| \le \alpha\lambda \\ 0 & \text{若} \quad |\beta_j| > \alpha\lambda \end{cases}$$

可以看出，当 β_j 近似于零时，MCP 和 Lasso 的惩罚力度一致，而随 β_j 从零开始增大，MCP 的惩罚力度逐渐缩减为零。当 $|\beta| > \alpha\lambda$ 时，MCP 对大系数不施加惩罚，因此 MCP 同 SCAD 类似，也实现了回归系数的有差别惩罚，实现了更精确的估计。

下图展示了回归系数变化时 Lasso、SCAD 和 MCP 的惩罚力度的变化情况：



7.4.3 群组变量选择方法

随着科技发展，数据的种类与维数的不断增加，导致了回归建模时，预测变量的群组结构成为了一种普遍现象。比如，变量间的相关性无法忽略时，建模时应当处理共线性问题；或者在医学上研究基因对疾病的影响时，一般把同一基因下的变量作为一组变量来进行处理；或者在处理多分类变量时，经常使用的虚拟变量也是一类组变量，在进行组变量选择时它们要当成一个整体去对待。在这种情况下，相关的选择问题就变成选择组而不仅仅是单个变量，群组变量选择模型及其算法成为当下高维数据建模的主要研究方向，在此我们介绍两种代表性方法。

弹性网（Elastic Net）

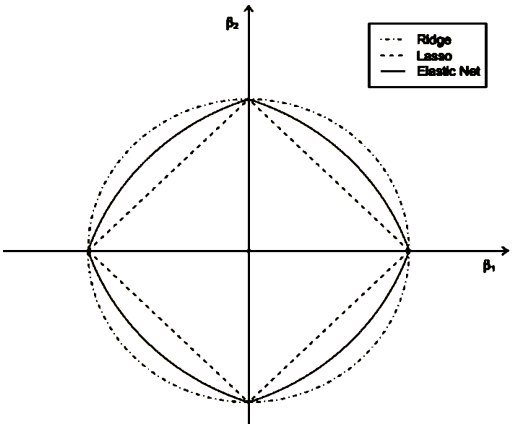
岭回归结果表明，岭回归虽然一定程度上可以拟合模型，但容易导致回归结果失真；Lasso 回归虽然能刻画模型代表的现实情况，但是模型过于简单，不符合实际，当多个变量和另一个变量高度相关的时候，Lasso 倾向于随机选择其中一个。但如果这几个变量都是对模型重要的变量，那么 Lasso 只能从多个相关强预测变量中选一个。

Zou 和 Hastie [86] 在 2005 年提出的弹性网结合了岭回归和 Lasso 算法，既能做到岭回归不能做的预测变量提取，又能实现 Lasso 不能做的预测变量分组，是最早解决共线性问题的变量选择方法，能够将显著的强相关变量同时选入模型。其目标函数为：

$$\min_{\boldsymbol{\beta}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\beta})+\lambda\left[\alpha\|\boldsymbol{\beta}\|_1+(1-\alpha)\|\boldsymbol{\beta}\|_2^2\right]$$

显然，这种方法是利用 L1 + L2 范数的凸组合来实现约束的，其中 L1 正则化用于变量选择，L2 正则化用于解决共线性问题。除了加入调节参数 λ，我们还引入参数 α 来调节 L1 范数和 L2 范数的平衡。当 $\alpha = 0$ 时，弹性网回归即为岭回归；当

$\alpha = 1$ 时，弹性网回归即为 Lasso 回归。当 $\alpha = 0.5$ 的时候，弹性网的几何结构如下图所示：



由图形可知，弹性网回归在坐标轴上有尖角，在各个象限内部呈弧形，因此既具备 Lasso 的变量选择功能，又具备 Ridge 的系数收缩功能。

Group Lasso

Yuan 和 Lin [87] 在 2006 年将 Lasso 方法推广到组群结构上面，诞生了 Group Lasso。我们可以将所有变量分组，然后在目标函数中加总每个组系数的 L2 范数，这样达到的效果就是可以将一整组的系数同时消成零，即抹掉一整组的变量，这种手法叫做Group Lasso 算法。其目标函数为：

$$\min_{\boldsymbol{\beta}} \left(\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{g=1}^{G} \|\sqrt{ql}\boldsymbol{\beta}_{\boldsymbol{I}_g}\|_2 \right)$$

其中，I_g 是 g 组的预测变量下标，$g = 1, 2, \ldots\ldots, G$，$G$ 代表组的个数，λ 是调整参数，$\sqrt{ql}$ 是每一组的加权，可按需调节。其惩罚函数中，在组内，是只具有压缩功能而没有选择功能的岭惩罚；在组间是具有变量选择功能的 Bridge 惩罚，因而整组变量会同时被选入或删除。我们在此提到的 Bridge 惩罚，即 $0 < p < 1$ 的 Lp 惩罚函数，从惩罚函数形式来看，Lasso 对系数 β_j 的惩罚强度是固定的，而 Bridge 对 β_j 的惩罚强度随 β_j 的增大而减小。当 β_j 趋于 0 时，Bridge 的惩罚强度趋于无穷大，使得其稀疏性较 Lasso 更加显著，而当 β_j 很大时，Bridge 的惩罚强度变得很小，使得其参数估计近似无偏。

由于 Group Lasso 是一种通过对惩罚项的修改将 Lasso 方法在群组结构下的拓展，因此与 Lasso 具备相同的缺点，惩罚率不随组系数的大小而变化，因此对较大的系数过分压缩，导致估计偏差。

7.4.4 双层变量选择方法

Group Lasso 方法由于只进行组变量的选择，不能选择组内重要的单变量，使得同组变量全被选择或全被删除，因此有些显著单变量在进行变量选择时会随着组变量被淘汰掉，从而导致模型误差过大，精确度降低。为了同时既考虑组的选择又考虑组内重要变量选择，我们引入双层变量选择方法。

Group Bridge

Huang 等 [88] 提出的Group Brige是一种可以满足组间、组内选择的双层变量选择方法，是最早实现双层变量选择的方法。其惩罚函数的形式为：

$$P(\boldsymbol{\beta};\lambda,\gamma)=\lambda\sum_{j=1}^{W}p_j^{\gamma}\|\boldsymbol{\beta}^{(j)}\|_1^{\gamma}$$

其中 $0<\gamma<1$,p_j 是第 j 组变量所包含个数，$\boldsymbol{\beta}^{(j)}=\left(\beta_1^{(j)},\ldots,\beta_{p_j}^{(j)}\right)$ 是第 j 组预测变量的回归系数，其中回归系数共分为 W 个组。其惩罚函数是由组内惩罚和组间惩罚函数复合而成，其中，组内选择的是 Lasso 惩罚函数，组间选择的是 Bridge 惩罚函数，从而实现了组间变量与组内变量的选择。

Composite MCP

Composite MCP是 Huang 等人 2009 年提出的另一种复合惩罚类的变量选择方法 [89]，其惩罚函数形式为：

$$P(\boldsymbol{\beta};\lambda,a,b)=\sum_{j=1}^{W}P_{\mathrm{MCP}}\left(\sum_{m=1}^{p_j}P_{\mathrm{MCP}}\left(|\beta_j^{(m)}|;\lambda,a\right);\lambda,b\right)$$

其组内和组间均使用 MCP 惩罚，均具备单变量选择的功能。

SGL（Sparse Group Penalty）

在双层变量选择方法中也存在另一种形式，它的惩罚函数不是由两个具有单变量选择的功能惩罚函数复合而成的，而是由单个变量惩罚和仅选择组变量惩罚的线性组合构成的函数。这种惩罚方法叫做稀疏组惩罚。

Simon 等人 2013 年提出的 SGL [90] 就属于此类方法，其惩罚函数形式为：

$$P_{\mathrm{SGL}}(\boldsymbol{\beta};\lambda_1,\lambda_2)=\lambda_1\sum_{j=1}^{W}\|\boldsymbol{\beta}^{(j)}\|_2+\lambda_2\|\boldsymbol{\beta}\|_1$$

其中第一项是选择重要组变量，选择的是 Group Lasso 方法的惩罚函数；第二个惩罚是选择单个变量，用的是 Lasso 方法的惩罚函数，其中 λ_1、λ_2 分别为作用在单个变量和组变量上的调整参数。

Adaptive SGL

SGL 相对复合函数型方法计算简单，但它对于所有系数及组系数的惩罚力度相同，过度压缩大系数，引起估计偏差。基于此 Fang 等 2014 年提出了更一般化

的Adaptive SGL [91]，其惩罚函数形式为：

$$P_{\text{adSGL}}(\boldsymbol{\beta};\lambda_1,\lambda_2)=\lambda_1\sum_{j=1}^{W}w_j\|\boldsymbol{\beta}^{(j)}\|_2+\lambda_2\sum_{j=1}^{W}\xi^{(j)T}|\boldsymbol{\beta}^{(j)}|$$

它通过引入权重 $\xi=(\xi^1,\ldots,\xi^W)$ 和 $w=(w_1,\ldots,w_W)$ 分别对单个系数和组系数施加不同程度的惩罚，权重依据样本数据而定，系数的真实值越大，给予权重越小，惩罚力度越小，增进了预测精度。

7.5 模型选择实践

7.5.1 R 语言实践

下面介绍怎么用 R 语言实现上面提到的模型选择方法。

子集选择法

本节使用 ISLR 包中的 College 数据集，该数据集包含了自 1995 年以来美国大学的 777 条数据，每条数据含有 18 个变量。其中将变量 “Apps” 作为因变量，其余 17 个变量作为预测变量进行下面的分析。具体实现过程如下：

1、**最优子集法**

```
library(ISLR)
#R语言的包leaps中的regsubsets()函数可用于筛选最优因变量子集
library(leaps)
data(College)
names(College)
dim(College)
#Apps为因变量、College为数据、nvmax表示所输出结果最优模型所包含的变量个数，默认值为8
subset.full<-regsubsets(Apps ~ .,College,nvmax = 17)
full.summary<-summary(subset.full)
#输出结果如下：
Selection Algorithm: exhaustive
         PrivateYes Accept Enroll Top10perc Top25perc F.Undergrad
1 (1)    " "        "*"    " "    " "        " "        " "
2 (1)    " "        "*"    " "    "*"        " "        " "
3 (1)    " "        "*"    " "    "*"        " "        " "
         P.Undergrad Outstate Room.Board Books Personal PhD Terminal
1 (1)    " "          " "      " "        " "   " "      " " " "
2 (1)    " "          " "      " "        " "   " "      " " " "
3 (1)    " "          " "      " "        " "   " "      " " " "
         S.F.Ratio perc.alumni Expend Grad.Rate
1 (1)    " "       " "         " "    " "
```

```
2 (1)    " "        " "          " "     " "
3 (1)    " "        " "          "*"     " "
#summary()函数的结果
names(full.summary)
full.summary$rsq
full.summary$cp
```

结果中的星号“*”表示该列对应的变量出现在该行对应的最优模型中，最前面的阿拉伯数字表示该模型中含有的变量数，如第三行结果表示该最优模型含有 3 个变量分别为 Accept、Expend、Top10perc。除此之外 summary() 函数还包含 $\bar{R}^2$、Rss、调整 $\bar{R}^2$、C_p、BIC 等结果。

从结果中可以看出随着模型中包含变量个数的增加，$\bar{R}^2$ 不断增大，C_p 值先不断地减小再增加，在变量数为 12 时达到最小。最优子集法选择模型时，根据不同的准则选择，例如在一定条件下选择 $\bar{R}^2$ 最大、C_p、AIC、BIC 值最小的模型。可以用图形表示出模型中变量个数与相应指标值的关系，从而更加方便明确的选择最优模型：

```
par(mfrow =c(1,3))
which.min(full.summary$cp)# CP
plot(full.summary$cp,xlab="Number of Variables",ylab="CP",type="b")
points(12,full.summary$cp[12],col="red",cex=2,pch=20)
which.min(full.summary$bic)# BIC
plot(full.summary$bic,xlab="Number of Variables",ylab="BIC",type="b")
points(10,full.summary$bic[10],col="red",cex=2,pch=20)
which.max(full.summary$adjr2)# Adjust Rsq
plot(full.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="b")
points(13,full.summary$adjr2[13],col="red",cex=2,pch=20)
#提取含有10个变量的最优模型
coef(subset.full,10)
```

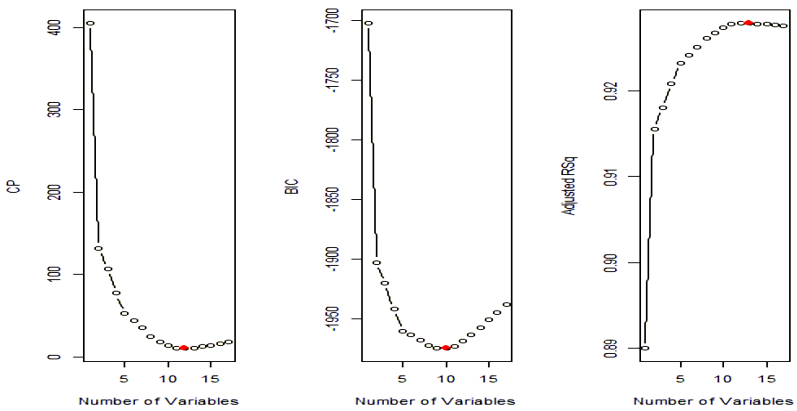


图 7.2 模型选取的变量个数与不同统计指标的关系

从图7.2中可以看出随着模型中变量个数的增加 C_p、BIC 的值是先下降后上升，

变量数分别为 12 和 10 时相应指标值达到最小，调整 $\bar{R}^2$ 的值是先上升后下降，在变量数为 13 时达到最大。

2、**向前逐步选择法与向后逐步选择法**

向前逐步选择法与向后逐步选择法仍是使用 regsubsets() 函数，分别将参数 method 设置为 forward 和 backward，并分别利用 coef() 函数提取取含有 8 个变量的最优模型

```
#向前逐步选择法
subset.fwd<-regsubsets(Apps ~ ., College, nvmax = 17, method = "forward")
summary(subset.fwd)
#向后逐步选择法
subset.bwd<-regsubsets(Apps ~ ., College, nvmax = 17, method = "backward")
summary(subset.bwd)
coef(subset.full,8)
coef(subset.fwd,8)
coef(subset.bwd,8)
```

惩罚方法

下面说明这三种方法分别在线性回归和 logistic 回归下的变量选择。

1、**线性回归**

首先产生样本量为 100 的样本，每个样本变量个数 p=20，各预测变量独立同分布于标准正态分布，由向量 beta 给出回归系数的真实值，误差项满足标准正态分布。在实现 LASSO 惩罚、MCP 惩罚以及 SCAND 惩罚这三种方法的变量选择时，使用 R 语言中的 glmnet 包和 ncvreg 包，其中 glmnet 包的 cv.glmnet() 函数默认使用 LASSO 方法选择变量，ncvreg 包中的 cv.ncvreg() 函数默认使用 MCP 方法选择变量。

（1）**LASSO 惩罚**

```
x<-matrix(rnorm(100*20), 100, 20)
#设定第7、8、9、10个变量对应的回归系数为零
beta<-c(seq(1, 2, length.out = 6), 0, 0, 0,  0, rep(1, 10))
y<-x %*% beta + rnorm(100)
library(glmnet)
fit1<-cv.glmnet(x, y, family = "gaussian")
beta.fit1<-coef(fit1)#提取参数的估计值
beta.fit1
resid1 <-(x %*% beta.fit1 [ -1 ] + beta.fit1 [ 1 ] - y)
MSE1<-sum(resid1 ^ 2)#计算残差平方和
MSE1
```

（2）**MCP 惩罚和 SCAD 惩罚**

```
library(ncvreg)
fit2<-cv.ncvreg(x, y, family = "gaussian")#MCP
fit3<-cv.ncvreg(x, y, family = "gaussian", penalty = "SCAD")
fit.mcp<-fit2$fit
```

```
fit.scad<-fit3$fit
beta.fit2<-fit.mcp$beta [, fit2$min ]#提取参数的估计值
beta.fit3<-fit.scad$beta [, fit2$min ]#提取参数的估计值
round(beta.fit2, 3)#保留三位小数
round(beta.fit3, 3)
resid2 <-(x %*% beta.fit2 [ -1 ] + beta.fit2 [ 1 ] - y)
resid3 <-(x %*% beta.fit3 [ -1 ] + beta.fit3 [ 1 ] - y)
MSE2<-sum(resid2 ^ 2)#计算残差平方和
MSE3<-sum(resid3 ^ 2)
```

2、**Logistic 回归**

在进行 Logistic 回归时使用的数据是 ncvreg 包中自带的 heart 数据集，该数据共 462 个样本，每个样本含有 10 个变量，以变量“chd”为因变量，其余 9 个变量作为预测变量。

（1）**LASSO 惩罚**

```
library(ncvreg)
data(heart)
x<-as.matrix(heart[,1:9])
y<-heart$chd
library(glmnet)
fit1<-cv.glmnet(x, y, family = "binomial")
beta.fit1<-coef(fit1)#提取参数的估计值
beta.fit1
round(beta.fit1, 3)
```

（2）**MCP 和 SCAD 惩罚**

```
fit2<-cv.ncvreg(x, y, family = "binomial")#MCP
fit3<-cv.ncvreg(x, y, family = "binomial", penalty="SCAD")#SCAD
fit.mcp<-fit2$fit
fit.scad<-fit3$fit
beta.fit2<-fit.mcp$beta [, fit2$min ]
beta.fit3<-fit.scad$beta [, fit3$min ]
round(beta.fit2, 3)
round(beta.fit3, 3)
```

组模型选择

1、**线性回归**

使用 R 语言中的 grpreg 包来处理 Group LASSO 问题，使用 glmnet 包处理弹性网问题，下面以 grpreg 包中的 birthwt.grpreg 数据为例进行分析。

（1）**Group LASSO**

```
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$bwt
colnames(X)#变量组结构,数字相同的为同一类变量
```

```
group<-c(1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 8)
cvfit<-cv.grpreg(X, y, group, penalty = "grLasso")
beta.cvfit<-coef(cvfit)#系数估计值
round(beta.cvfit,3)
```

（2）**弹性网**

```
library(glmnet)
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1:-2 ])
y<-birthwt.grpreg$bwt
fit.enet<-cv.glmnet(X, y, family = "gaussian", alpha = 0.2)
beta.enet<-coef(fit.enet)
round(beta.enet,3)
```

2、**Logistic 回归**

进行 Logistic 回归时仍使用 birthwt.grpreg 数据集，由于 cv.grpreg() 函数可以自动识别因变量的数据类型从而进行对应的回归，所以在使用 cv.grpreg() 时没有加 family 参数。

（1）**Group LASSO**

```
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$low
group<-c(1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 8)#变量的分组结构
cvfit<-cv.grpreg(X, y, group, penalty = "grLasso")
beta.cvfit<-coef(cvfit)
summary(cvfit)
plot(cvfit)
```

大家可以输出一下结果，可知有 6 个变量没有选入模型，且从 summary() 的结果可以看出，最优的 λ 值为 0.0379，在该最优 λ 值下模型的交叉验证误差达到最小为 0.2。图7.3展示了连续 λ 对数值下交叉验证误差的变化：

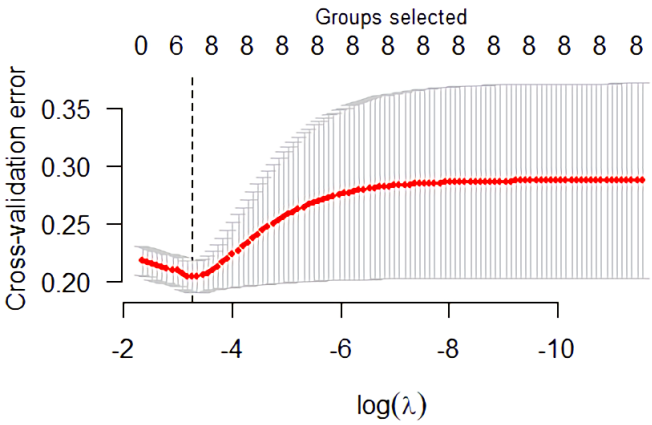


图 7.3 birthwt.grpreg 使用 Group LASSO 方法的交叉验证图

（2）**弹性网**

```
y<-birthwt.grpreg$low
fit.enet<-cv.glmnet(X, y, family = "binomial", alpha = 0.1)
beta.enet<-coef(fit.enet)
round(beta.enet,3)
```

双层模型选择

通过 R 语言中的 grpreg 包实现 Group Bridge 和 Composite MCP，通过 SGL 包实现 SGL 惩罚，下面仍然以 Birthwt.grpreg 为例说明这两种方法分别在线性回归和 Logistic 回归中的变量选择。

1、**线性回归**

（1）**复合函数双层选择：Group Bridge 和 Composite MCP**

```
#Group Bridge
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$bwt
group<-c(1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 8)#变量的分组结构
cvfit.b<-gBridge(X, y, group)#L1 group bridge
beta.cvfit.b<-select(cvfit.b)$beta
round(beta.cvfit.b,3)
cvfit.m<-cv.grpreg(X, y, group, penalty = "cMCP", gama = 2.5)#Composite MCP
coef(cvfit.m)
```

（2）**稀疏组惩罚：SGL**

```
library(SGL)
```

```
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$bwt
group<-c(1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 8)#变量的分组结构
data<-list(x = X, y = y)
cvFit<-cvSGL(data, group, type = "linear")#SGL
lambda.min<-which.min(cvFit$lldiff)
beta.cvFit<-cvFit$fit$beta [, lambda.min ]
round(beta.cvFit,3)
```

2、**Logistic 回归**

（1）**复合函数双层选择：Group Bridge 和 Composite MCP**

```
# Group Bridge
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$low
cvfit.b<-gBridge(X, y, group, family = "binomial")
beta.cvfit.b<-select(cvfit.b)$beta
round(beta.cvfit.b,3)
#Composite MCP
cvfit.m<-cv.grpreg(X, y, group, penalty = "cMCP", family = "binomial")
beta.cvfit.m<-coef(cvfit.m)
round(beta.cvfit.m,3)
```

（2）**稀疏组惩罚：SGL**

```
library(SGL)
library(grpreg)
data(birthwt.grpreg)
X<-as.matrix(birthwt.grpreg [, -1 : -2 ])
y<-birthwt.grpreg$low
group<-c(1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 8)#变量的分组结构
data<-list(x = X, y = y)
cvFit<-cvSGL(data, group, type = "logit")
lambda.min<-which.min(cvFit$lldiff)#最优值
beta.cvFit<-cvFit$fit$beta [, lambda.min ]#最优值时回归系数的估计结果
round(beta.cvFit,3)
cvFit$fit$intercepts [ lambda.min ]
```

7.5.2 Python 语言实践

本节使用 python 中 sklearn 模块中自带的数据集。sklearn 中含有很多类型的数据集，例如，“20newsgroups ”，“20newsgroups_vectorized” 用于文本分析；“california_housing ” 进行回归；“wine ” 进行分类。本书选择 “california_housing” 即加州房屋数据集作为代码部分的支撑。

下面先对该数据集进行简单的介绍：该数据集是基于 1990 年加州普查的数据，数据集中的每一个数据都代表着一块区域内房屋和人口的基本信息，总共包括 9 项：该地区中心的纬度（latitude）、该地区中心的经度（longitude）、区域内所有房屋屋龄的中位数（housingMedianAge）、区域内总房间数（totalRooms）、区域内总卧室数（totalBedrooms）、区域内总人口数（population）、区域内总家庭数（households）、区域内人均收入中位数（medianIncome）、该区域房价的中位数（medianHouseValue）。这里的地域单位是美国做人口普查的最小地域单位，平均一个地域单位中有 1400 多人。

本书主要以房屋价值中位数为标签，其余属性为预测变量，通过使用 python 编程，利用 pyhon 相关库，如 numpy、pandas、statsmodels 等库，对加州地区的数据进行预测变量选择。

逐步回归

首先导入相应模块，并定义用于逐步回归的 stepwise_select 函数

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols #加载ols模型
################################### 逐步回归
def stepwise_select(data, label, cols_all, method='forward'):
    '''
    data: 数据源(数据框) label: 标签(字符串) cols_all: 逐步回归的全部字段
    method: 方法(forward:向前, backward:向后, both:双向)
    函数返回值: select_col: 最终保留的字段列表(列表) summary: 模型参数
    AIC: 赤池信息量
    '''
    #1.前向回归
    #从一个变量都没有开始，将变量逐个加入到模型中，直至没有可以再加入的变量结束
    if method == 'forward':
        add_col = []
        AIC_None_value = np.inf
        while cols_all:
            # 单个变量加入，计算aic
            AIC = {}
            for col in cols_all:
                print(col)
                X_col = add_col.copy()
                X_col.append(col)
                X = sm.add_constant(house_data[X_col])
                y = house_data[label]
                # 将预测变量名连接起来
                formula = "{}~{}".format(label, "+".join(add_col + [col]))
                AIC[col] = ols(formula=formula, data=house_data).fit().aic
                AIC_min_value = min(AIC.values())
                AIC_min_key = min(AIC, key=AIC.get)
                # 如果最小的aic小于不加该变量时的aic，则加入变量，否则停止
```

```
            if AIC_min_value < AIC_None_value:
                cols_all.remove(AIC_min_key)
                add_col.append(AIC_min_key)
                AIC_None_value = AIC_min_value
            else:
                break
    select_col = add_col
#2.后向回归
# 从全部变量都在模型中开始，一个变量一个变量的删除，直至没有可以再删除的变量结束
elif method == 'backward':
    # 全部变量，一个都不剔除，计算初始aic
    X_col = cols_all.copy()
    X = sm.add_constant(house_data[X_col])
    y = house_data[label]
    formula = "{}~{}".format(label, "+".join(cols_all))  # 将预测变量名连接起来
    LR = ols(formula=formula, data=house_data).fit()
    AIC_None_value = LR.aic
    while p:
    # 删除一个字段提取aic最小的字段
        AIC = {}
        for col in cols_all:
            print(col)
            X_col = [i for i in cols_all if i != col]
            X = sm.add_constant(house_data[X_col])
            cols_all_remove = cols_all.copy()
            cols_all_remove.remove(col)
            # 将预测变量名连接起来
            formula = "{}~{}".format(label, "+".join(cols_all_remove))
            AIC[col] = ols(formula=formula, data=house_data).fit().aic
            AIC_min_value = min(AIC.values())
            AIC_min_key = min(AIC, key=AIC.get)
            # 如果最小的aic小于不删除该变量时的aic，则删除该变量，否则停止
            if AIC_min_value < AIC_None_value:
                cols_all.remove(AIC_min_key)
                AIC_None_value = AIC_min_value
                p = True
            else:
                break
    select_col = cols_all
        ######################### 3.双向选择
elif method == 'both':
    p = True
    add_col = []
    # 全部变量，一个都不剔除，计算初始aic
    X_col = cols_all.copy()
    X = sm.add_constant(house_data[X_col])
    y = house_data[label]
    formula = "{}~{}".format(label, "+".join(cols_all))  # 将预测变量名连接起来
    LR = ols(formula=formula, data=house_data).fit()
    AIC_None_value = LR.aic
    while p:
        # 删除一个字段提取aic最小的字段
```

```
        AIC = {}
        for col in cols_all:
            print(col)
            X_col = [i for i in cols_all if i != col]
            X = sm.add_constant(house_data[X_col])
            # 将预测变量名连接起来
            formula = "{}~{}".format(label, "+".join(add_col + [col]))
            LR = ols(formula=formula, data=house_data).fit()
            AIC[col] = LR.aic
            AIC_min_value = min(AIC.values())
            AIC_min_key = min(AIC, key=AIC.get)
            if len(add_col) == 0:  # 第一次只有删除操作，不循环加入变量
                if AIC_min_value < AIC_None_value:
                    cols_all.remove(AIC_min_key)
                    add_col.append(AIC_min_key)
                    AIC_None_value = AIC_min_value
                    p = True
                else:
                    break
            else:
            # 单个变量加入，计算aic
                for col in add_col:
                    print(col)
                    X_col = cols_all.copy()
                    X_col.append(col)
                    X = sm.add_constant(house_data[X_col])
                    formula = "{}~{}".format(label, "+".join(add_col + [col]))  #
                                                                    将预测变量名连接
                                                                    起来
                    LR = ols(formula=formula, data=house_data).fit()
                    AIC[col] = LR.aic
                    AIC_min_value = min(AIC.values())
                    AIC_min_key = min(AIC, key=AIC.get)
                    if AIC_min_value < AIC_None_value:
                    # 如果aic最小的字段在添加变量阶段产生，则加入该变量，如果aic
                    最小的字段在删除阶段产生,则删除该变量
                        if AIC_min_key in add_col:
                            cols_all.append(AIC_min_key)
                            add_col = list(set(add_col) - set(AIC_min_key))
                            p = True
                        else:
                            cols_all.remove(AIC_min_key)
                            add_col.append(AIC_min_key)
                            p = True
                            AIC_None_value = AIC_min_value
                    else:
                        break
    select_col = cols_all
#模型
X = sm.add_constant(house_data[select_col])
# 将预测变量名连接起来
formula = "{}~{}".format(label, "+".join(select_col))
```

```
    LR = ols(formula=formula, data=house_data).fit()
    summary = LR.summary()
    AIC = LR.aic
    return select_col, summary, AIC
```

再导入加利福尼亚房屋价值数据，调用上文定义的 stepwise_select 函数进行模型选择

```
#加载加利福尼亚房屋价值数据
from sklearn.datasets import fetch_california_housing as fch
data=fch() #导入数据
house_data=pd.DataFrame(data.data) #将预测变量转换成dataframe格式，便于查看
house_data.columns=data.feature_names  #命名预测变量
house_data.loc[:,"value"]=data.target #合并预测变量，因变量数据
house_data.shape #查看数据量
house_data.head(10) #查看前10行数据
cols_all=set(house_data.columns)  #将字段名转换成字典类型
cols_all.remove('value')
label='value'
print(stepwise_select(house_data, label, cols_all, method='both'))
```

岭回归与 Lasso 回归

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.datasets import fetch_california_housing as fch
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
iris = fch()
data = pd.DataFrame(iris.data)
print(data.corr())#看一下各变量之间的相关性
scaler = StandardScaler()
x = scaler.fit_transform(iris.data)
y = iris.target
lr = LinearRegression()
lr.fit(x,y)
print('lr.coef')
print(lr.coef_)
lasso = Lasso(alpha=0.1)
lasso.fit(x,y)
print('Lasso.coef')
print(lasso.coef_)
ridge = Ridge(alpha=10)
ridge.fit(x,y)
print('Ridge.coef')
print(ridge.coef_)
```

群组变量选择方法

首先是弹性网，代码如下

```
import numpy as np
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_california_housing as fch
import pandas as pd
iris = fch()
data = pd.DataFrame(iris.data)
print(data.corr())#看一下各变量之间的相关性
scaler = StandardScaler()
x = scaler.fit_transform(iris.data)
y = iris.target
elastic_net = ElasticNet(alpha=0.0001, l1_ratio=0.15)
print(elastic_net.fit(x, y).coef_)
```

然后是 GroupLasso

```
from groupLasso import GroupLassoRegressor
import numpy as np
np.random.seed(0)
#Create sample dataset:
X = np.random.randn(10, 3)
y = X[:, 0] + np.random.randn(10) * 0.1
#Set group_ids, which specify group membership:
# 0th feature and 1st feature are the same group.
group_ids = np.array([0, 0, 1])
model = GroupLassoRegressor(group_ids=group_ids, random_state=42, verbose=False,
alpha=1e-1)
model.fit(X, y)
print(model.coef_)
```

7.6 习题

1、试证明 $\widehat{\boldsymbol{\beta}} = (\widehat{\boldsymbol{\beta}}_1^{\mathrm{T}}, \mathbf{0}^{\mathrm{T}}) \in R^p$ 是基于惩罚最小二乘目标函数 (7.4.1) 的局部极小值的充分条件是：

$$n^{-1}\mathbf{X}_1^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}}) - p'_\lambda\left(\left|\widehat{\boldsymbol{\beta}}_1\right|\right)\operatorname{sgn}\left(\widehat{\boldsymbol{\beta}}_1\right) = \mathbf{0} \tag{7.6.1}$$

$$\left\|n^{-1}\mathbf{X}_2^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})\right\|_\infty \leq p'_\lambda(0+) \tag{7.6.2}$$

$$\lambda_{\min}\left(n^{-1}\mathbf{X}_1^{\mathrm{T}}\mathbf{X}_1\right) \geq \kappa\left(p_\lambda; \widehat{\boldsymbol{\beta}}_1\right) \tag{7.6.3}$$

其中设计矩阵 $\mathbf{X}_1$ 对应的变量与 $\widehat{\boldsymbol{\beta}}_1$ 对应的变量一致，$\mathbf{X}_2$ 是零系数估计量对应的变量。$\lambda_{\min}(\mathbf{A})$be 代表矩阵 $\mathbf{A}$ 的最小预测变量值，$\|\mathbf{a}\|_\infty = \max_j |a_j|$。$\kappa\left(p_\lambda; \mathbf{v}\right)$ 描述了 $p_\lambda(\cdot)$ 在 $\mathbf{v} = (v_1, \cdots, v_q)^{\mathrm{T}}$ 的局部凹性质，且定义为

$$\kappa\left(p_\lambda; \mathbf{v}\right) = \lim_{\epsilon \to 0+} \max_{1 \leq j \leq q} \max_{t_1 < t_2 \in (|v_j|-\epsilon, |v_j|+\epsilon)} -\frac{p'_\lambda\left(t_2\right) - p'_\lambda\left(t_1\right)}{t_2 - t_1}$$

根据凹函数性质，在 $[0, \infty)$，$\kappa\left(p_\lambda; \mathbf{v}\right) \geq 0$。Lasso 惩罚的 $\kappa\left(p_\lambda; \mathbf{v}\right) = 0$；对于 SCAD 惩罚，当 $|\mathbf{v}|$ 取值于 $[\lambda, a\lambda]$，$\kappa\left(p_\lambda; \mathbf{v}\right) = (a-1)^{-1}\lambda^{-1}$，其余地方 $\kappa\left(p_\lambda; \mathbf{v}\right) = 0$。

2、基于练习 1 的结论，证明

（1）Lasso 惩罚函数自动满足第三个 KKT 条件（7.6.3），且（7.6.1）-（7.6.2）转变为

$$n^{-1}\mathbf{X}_1^{\mathrm{T}}\left(\boldsymbol{Y} - \mathbf{X}_1\widehat{\boldsymbol{\beta}}_1\right) - \lambda \operatorname{sgn}\left(\widehat{\boldsymbol{\beta}}_1\right) = \mathbf{0}$$

$$\left\|(n\lambda)^{-1}\mathbf{X}_2^{\mathrm{T}}\left(\boldsymbol{Y} - \mathbf{X}_1\widehat{\boldsymbol{\beta}}_1\right)\right\|_\infty \leq 1$$

（2）$\left\|n^{-1}\mathbf{X}^{\mathrm{T}}(\boldsymbol{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}})\right\|_\infty \leq \lambda$

（3）当 $\lambda > \left\|n^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{Y}\right\|_\infty$，$\widehat{\boldsymbol{\beta}} = 0$。因此调谐参数 λ 必须在区间 $\left[0, \left\|n^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{Y}\right\|_\infty\right]$ 上取值。

3、考虑如下简单的惩罚最小二乘形式：

$$\hat{\theta}(z) = \operatorname{argmin}_\theta \left\{\frac{1}{2}(z-\theta)^2 + p_\lambda(|\theta|)\right\}.$$

（1）如果 $p_\lambda(|\theta|) = \dfrac{\lambda^2}{2} I(|\theta| \neq 0)$，即 L_0 惩罚，证明 $\widehat{\theta}_H(z \mid \lambda) = zI(|z| \geq \lambda)$。

（2）如果 $p_\lambda(\theta) = \dfrac{1}{2}\lambda^2 - \dfrac{1}{2}(\lambda - \theta)_+^2$，我们得到基于惩罚最小二乘硬阈估计量 $\widehat{\theta}_H(z) = zI(|z| > \lambda)$。并且解释硬阈惩罚函数在哪些方面优于 L_0 惩罚。

（3）若 $p_\lambda(\theta)$ 是 Lasso 函数，我们得到软阈估计量：

$$\widehat{\theta}_{\text{soft}}\left(z\right) = \operatorname{sgn}(z)(|z| - \lambda)_+$$

（4）若 $p_\lambda(\theta)$ 是 SCAD 函数，估计量是：

$$\widehat{\theta}_{\text{SCAD}}(z) = \begin{cases} \operatorname{sgn}(z)(|z| - \lambda)_+, & \text{when } |z| \leq 2\lambda \\ \operatorname{sgn}(z)[(\alpha-1)|z| - \alpha\lambda]/(\alpha - 2), & \text{when } 2\lambda < |z| \leq \alpha\lambda \\ z, & \text{when } |z| \geq \alpha\lambda \end{cases}$$

并且当 $\alpha = \infty$ 时，SCAD 估计量变成软阈估计量。

（5）若 $p_\lambda(\theta)$ 是 MCP 函数（$\alpha \geq 1$），估计量是：

$$\widehat{\theta}_{\mathrm{MCP}}(z) = \begin{cases} \mathrm{sgn}(z)(|z| - \lambda)_+/(1 - 1/\alpha), & \text{when } |z| < \alpha\lambda \\ z, & \text{when } |z| \geq \alpha\lambda \end{cases}$$

（6）计算基于 Elastic Net 惩罚函数 $p_\lambda(\theta) = \lambda\left\{(1-\alpha)\theta^2 + \alpha|\theta|\right\}$ 的估计量 $\hat{\theta}(z \mid \lambda)$。

4、根据 Lasso 惩罚最小二乘的第一个 KKT 条件，证明 Lasso 非零回归系数估计量是有偏的，并计算偏差。

第八章　特征筛选

模型选择和变量筛选的目标都是提高模型在新数据上的泛化性能，避免过拟合并提高模型的可解释性。模型选择侧重在一组不同的模型中选择一个最适合数据的模型，涉及选择不同的算法、模型结构或超参数。而变量筛选侧重在一个给定的模型中，选择对目标变量最具预测能力的特征或变量。

8.1　简介

尽管变量选择方法可以用来识别重要变量，但是在维度 p 非常高的情况下，即维度 p 随着样本数量 n 的增加呈现指数速率增长，$\log(p) = O(n^{\delta})$，$0 < \delta < 1$，例如基因数据，此时用于优化的算法仍然非常昂贵。在实践中，我们可以自然地考虑一个两阶段的方法：先特征筛选，然后变量选择。首先强调一下，在机器学习领域普遍使用特征表示统计学里的变量，两个概念本质上是等价的。具体地说，我们使用特征筛选方法将超高维 p 降低至中等尺度 $d \leq n$，一般情况下，维度 d 随着样本数量 n 的增加呈现幂增长，$d = O(n^{\zeta})$，$0 < \zeta < 1$。然后再用变量选择方法从剩下的变量中选择真实模型。如果在每一个降维阶段都保留了所有重要变量，那么这个两阶段方法要经济得多。接下来我们介绍诸多特征筛选方法，其目标是尽可能多地丢弃噪声特征，同时保留所有的重要特征。

在本章中，我们采用以下符号。设 Y 为响应变量，$\boldsymbol{X} = (X_1, \cdots, X_p)^{\mathrm{T}}$ 由 p 维预测变量组成，由此得到 n 个独立的随机样本 $\{\boldsymbol{X}_i, Y_i\}_{i=1}^{n}$。$\mathbf{Y} = (Y_1, \cdots, Y_n)^{\mathrm{T}}$，$\mathbf{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_n)^{\mathrm{T}}$ 是 $n \times p$ 的设计矩阵。设 $\mathbf{X}_j$ 为 $\mathbf{X}$ 的第 j 列，则 $\mathbf{X} = (\mathbf{X}_1, \cdots, \mathbf{X}_p)$。我们稍微滥用了符号 $\mathbf{X}$ 和 $\mathbf{X}_j$，但是在上下文中它们的含义是清楚的。令 ε 是一个一般随机误差，$\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^{\mathrm{T}}$。设 $\mathcal{M}_*$ 代表一个尺寸为 $s = |\mathcal{M}_*|$ 的真实模型，$\widehat{\mathcal{M}}$ 为尺寸为 $d = |\widehat{\mathcal{M}}|$ 的选择模型。对于不同的模型和背景，$\mathcal{M}_*$ 和 $\widehat{\mathcal{M}}$ 的定义可能有所不同。

8.2　基于边际模型的特征筛选

8.2.1 边际最小二乘

对于线性回归模型，其矩阵形式为

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{8.2.1}$$

当 $p \gg n$ 时，$\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 是奇异的，因此 $\boldsymbol{\beta}$ 的最小二乘估计没有很好的定义。在这种情况下，岭回归特别有用。模型（8.2.1）的岭回归估计量由下式给出

$$\widehat{\boldsymbol{\beta}}_{\lambda} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I}_p\right)^{-1} \mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

其中 λ 是一个岭参数。回归分析章节中介绍过岭回归解决了多重共线性的问题，从正则项看，岭回归估计量是线性模型的带 L_2 惩罚的惩罚最小二乘的解。当 $\lambda \to 0$ 且 $\mathbf{X}$ 为满秩时，$\widehat{\boldsymbol{\beta}}_{\lambda}$ 趋于最小二乘估计量，而当 $\lambda \to \infty$ 时，$\lambda\widehat{\boldsymbol{\beta}}_{\lambda}$ 趋于 $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$。这意味着当 $\lambda \to \infty$ 时，$\widehat{\boldsymbol{\beta}}_{\lambda} \propto \mathbf{X}^{\mathrm{T}}\mathbf{Y}$。

假设所有的协变量和响应变量标准化，它们的样本均值和方差分别为 0 和 1。因此 $\dfrac{1}{n}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$ 成为由响应变量和所有协变量之间的 Pearson 相关系数的样本形式组成的向量。这促使人们使用 Pearson 相关系数作为特征筛选的边际统计量。具体来说，首先我们标准化 $\mathbf{X}_j$ 和 $\mathbf{Y}$，然后计算

$$\omega_j = \frac{1}{n}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}, \quad j = 1, 2, \ldots, p \tag{8.2.2}$$

即第 j 个预测变量和响应变量之间的样本相关系数。

直观来说，X_j 和 Y 之间的相关性越高，说明 X_j 越重要。Fan 和 Lv [92] 提出根据 $|\omega_j|$ 对预测变量 X_j 的重要性进行排序 (SIS, Sure Independence Screening)，并开发了一种基于 Pearson 相关系数的特征筛选过程，也称为确定性筛选，如下所示。对于预先指定的比例 $\gamma \in (0, 1)$，选择排名靠前的 $\lceil \gamma n \rceil$ 个预测变量来获得子模型

$$\widehat{\mathcal{M}}_{\gamma} = \{1 \leq j \leq p : |\omega_j| \text{是排序在前} \lceil \gamma n \rceil \text{中大的}\}$$

其中 $\lceil \gamma n \rceil$ 表示 γn 的整数部分。它将超高维度降至一个相对适中的尺度 $\lceil \gamma n \rceil$，即 $\widehat{\mathcal{M}}_{\gamma}$ 的大小，然后对子模型 $\widehat{\mathcal{M}}_{\gamma}$ 再采用变量选择方法。我们需要设置 γ 值来进行筛选，一般地，$\lceil \gamma n \rceil$ 的值可以取为 $\lceil n/\log(n) \rceil$。

8.2.2 边际极大似然

假设 $\{\boldsymbol{X}_i, Y_i\}_{i=1}^n$ 是前面介绍的广义线性模型的随机样本。基于第 i 个样本 $\{Y_i, \boldsymbol{X}_i\}$，用下式

$$\ell\left(Y_i, \beta_0 + \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}\right) = Y_i\left(\beta_0 + \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}\right) - b\left(\beta_0 + \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta}\right)$$

表示使用正则连接函数 b 的似然函数的负对数（不失一般性，离散参数取 $\phi=1$）。注意，当 $p \gg n$ 时，负对数似然 $\sum_{i=1}^{n} \ell\left(Y_i, \beta_0+\boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{\beta}\right)$ 的最小值无法给出很好地定义。

接下来我们考虑边际极大似然方法筛选出重要预测变量。与线性回归模型的边际最小二乘估计相似，假设每个预测变量进行了标准化，均值为 0，标准差为 1，并定义第 j 个预测变量 X_j 的边际极大似然估计量（MMLE: Marginal Maximum Likelihood Estimator）$\widehat{\boldsymbol{\beta}}_j^M$ 为

$$\widehat{\boldsymbol{\beta}}_j^M=\left(\widehat{\beta}_{j0}^M, \widehat{\beta}_{j1}^M\right)=\arg \min_{\beta_{j0}, \beta_{j1}} \sum_{i=1}^{n} \ell\left(Y_i, \beta_{j0}+\beta_{j1} X_{ij}\right)$$

可以将 $\widehat{\beta}_{j1}^M$ 的大小作为边际筛选统计量，对 X_j 的重要性排序，并通过给定的阈值 κ_n 选择子模型，即

$$\widehat{\mathcal{M}}_{\kappa_n}=\left\{1 \leq j \leq p:\left|\widehat{\beta}_{j1}^M\right| \geq \kappa_n\right\}$$

阈值 κ_n 的作用与 $\lceil\gamma n\rceil$ 中的 γ 是一样的，都是选择一个合适的值确定子模型。

8.2.3 边际非参估计

假设我们有一个随机样本集 $\{(\boldsymbol{X}_i, Y_i)\}_{i=1}^n$，来自可加模型:

$$Y=\sum_{j=1}^{p} f_j(X_j)+\varepsilon \tag{8.2.3}$$

其中 $\boldsymbol{X}=(X_1, \ldots, X_p)^{\mathrm{T}}$，$\varepsilon$ 是条件均值为 0 的随机误差。为了快速识别（8.2.3）式中的重要变量，我们考虑以下 p 个边际非参数回归问题:

$$\min_{f_j} E\left[Y-f_j\left(X_j\right)\right]^2 \tag{8.2.4}$$

(8.2.4) 式中的最小值是 $f_j(X_j)=E\left(Y \mid X_j\right)$，即 Y 在 X_j 上的投影。我们根据 $E f_j^2\left(X_j\right)$ 对（8.2.3）式中协变量的统计量进行排序，并通过阈值选择一小组协变量。

为了获得边际非参数回归的样本形式，我们使用 B 样条基。令 $\mathcal{S}_n$ 为维度 $l \geq 1$ 的多项式样条的空间，$\{\Psi_{jk}, k=1, \ldots, d\}$ 表示具有 $\|\Psi_{jk}\|_{\infty} \leq 1$ 的 B 样条基，其中 $\|\cdot\|_{\infty}$ 是 sup 范数。在某些平滑条件下，非参数投影 $\{f_j\}_{j=1}^p$ 可以被 $\mathcal{S}_n$ 中的函数 f_{nj} 很好地近似计算，即 $f_j \approx f_{nj}$。并且对于任意的 $f_{nj} \in \mathcal{S}_n$，存在某些系数 $\{\beta_{jk}\}_{k=1}^{q_n}$，我们有

$$f_{nj}(x)=\sum_{k=1}^{q_n} \beta_{jk} \Psi_{jk}(x), \quad 1 \leq j \leq p$$

此时，边际回归问题的样本形式可以表示为

$$\min_{f_{nj}\in\mathcal{S}_n} E\left(Y-f_{nj}\left(X_j\right)\right)^2=\min_{\boldsymbol{\beta}_j\in\mathcal{R}^{q_n}} E\left(Y-\boldsymbol{\Psi}_j^{\mathrm{T}}\boldsymbol{\beta}_j\right)^2 \tag{8.2.5}$$

其中 $\boldsymbol{\Psi}_j\equiv\boldsymbol{\Psi}_j\left(X_j\right)=\left(\Psi_1\left(X_j\right),\ldots,\Psi_{q_n}\left(X_j\right)\right)^{\mathrm{T}}$ 表示 q_n 维基函数。基于上述最小化的目标函数，我们首先得到回归系数 $\boldsymbol{\beta}_j$ 的总体形式

$$\boldsymbol{\beta}_j=\left(E\boldsymbol{\Psi}_j\boldsymbol{\Psi}_j^{\mathrm{T}}\right)^{-1}E(\boldsymbol{\Psi}_jY)$$

其中 E 表示期望。进而可以 $f_{nj}\left(X_j\right)$ 的最小二乘解的总体形式表示出来，如下所示：

$$f_{nj}\left(X_j\right)=\boldsymbol{\Psi}_j^{\mathrm{T}}\left(E\boldsymbol{\Psi}_j\boldsymbol{\Psi}_j^{\mathrm{T}}\right)^{-1}E(\boldsymbol{\Psi}_jY),\quad j=1,\ldots,p$$

基于观测的随机样本集 $\{(\boldsymbol{X}_i,Y_i)\}_{i=1}^n$，利用矩估计方法，即 $E\boldsymbol{\Psi}_j\boldsymbol{\Psi}_j^{\mathrm{T}}$ 和 $E(\boldsymbol{\Psi}_jY)$ 用样本矩代替，我们就可以得到 $\boldsymbol{\beta}_j$ 的估计了。

$$\widehat{\boldsymbol{\beta}_j}=\left(\frac{1}{n}\sum_{i=1}^n\boldsymbol{\Psi}_j(X_{ij})\boldsymbol{\Psi}_j^{\mathrm{T}}(X_{ij})\right)^{-1}\frac{1}{n}\sum_{i=1}^n\boldsymbol{\Psi}_j(X_{ij})Y_i$$

接下来，计算目标 $Ef_j^2\left(X_j\right)$ 的统计量的表示形式。首先，$Ef_j^2\left(X_j\right)$ 用 $Ef_{nj}^2\left(X_j\right)$ 去代替，$Ef_{nj}^2\left(X_j\right)$ 用 $E(\boldsymbol{\Psi}_j^{\mathrm{T}}(X_j)\widehat{\boldsymbol{\beta}_j})^2$ 去近似。最后 $E\boldsymbol{\Psi}_j^{\mathrm{T}}(X_j)\widehat{\boldsymbol{\beta}_j}$ 所对应的估计量为 $\left\|\widehat{f}_{nj}\right\|_n^2$，具有如下形式：

$$\left\|\widehat{f}_{nj}\right\|_n^2=\frac{1}{n}\sum_{i=1}^n\widehat{f}_{nj}\left(X_{ij}\right)^2=\frac{1}{n}\sum_{i=1}^n\left\{\boldsymbol{\Psi}_j^{\mathrm{T}}(X_{ij})\widehat{\boldsymbol{\beta}_j}\right\}^2$$

接下来，引入阈值 κ_n，根据边际非参数估计量 $\left\|\widehat{f}_{nj}\right\|_n^2$ 对重要变量进行排序 (NIS, Nonparametric Independence Screening)。

$$\widehat{\mathcal{M}}_{\kappa_n}=\left\{1\leq j\leq p:\left\|\widehat{f}_{nj}\right\|_n^2\geq\kappa_n\right\}$$

8.3 基于边际相关系数的特征筛选

8.3.1 广义和秩相关系数

上一节中的 SIS 方法对于具有超高维预测变量的线性回归模型表现良好。众所周知，Pearson 相关系数是用来衡量线性相关性的。然而对于超高维数据要确定一个回归结构非常困难。如果线性模型指定错误，SIS 会失败，因为 Pearson 相关系数

只能捕获每个预测变量和响应变量之间的线性关系。因此，SIS 最有可能错过一些非线性的重要预测变量，NIS 方法也需要可加性的条件。如果存在更加复杂的非线性关系，例如协变量外面是更加复杂且未知的函数变换等，NIS 的特征筛选效果也会不好。因此，这促使我们需要将转换后的协变量和响应变量之间的 Pearson 相关系数计算出来，视为边际筛选统计量进行特征筛选。

为了捕获这种类型的非线性，Hall 和 Miller [93] 定义第 j 个预测变量 X_j 和 Y 的广义相关系数为：

$$\rho_g\left(X_j, Y\right)=\sup_{h \in \mathcal{H}} \frac{\operatorname{cov}\left\{h\left(X_j\right), Y\right\}}{\sqrt{\operatorname{var}\left\{h\left(X_j\right)\right\} \operatorname{var}(Y)}}, \quad j=1, \ldots, p$$

其中 $\mathcal{H}$ 是包含所有线性函数的一类函数，例如，它是一类给定次数的多项式函数。请注意，如果 $\mathcal{H}$ 是所有线性函数的类，则 $\rho_g\left(X_j, Y\right)$ 是 $h(X_j)$ 和 Y 之间 Pearson 相关系数。因此，$\rho_g\left(X, Y\right)$ 被认为是传统 Pearson 相关系数的推广。假设 $\left\{\left(X_{ij}, Y_i\right), i=1,2, \ldots, n\right\}$ 是总体 $\left(X_j, Y\right)$ 的随机样本。广义相关系数 $\rho_g\left(X_j, Y\right)$ 可以通过下式来估计。

$$\widehat{\rho}_g\left(X_j, Y\right)=\sup_{h \in \mathcal{H}} \frac{\sum_{i=1}^n\left\{h\left(X_{ij}\right)-\bar{h}_j\right\}\left(Y_i-\bar{Y}\right)}{\sqrt{\sum_{i=1}^n\left\{h\left(X_{ij}\right)-\bar{h}_j\right\}^2 \sum_{i=1}^n\left(Y_i-\bar{Y}\right)^2}}$$

其中，$\bar{h}_j=n^{-1} \sum_{i=1}^n h\left(X_{ij}\right)$，$\bar{Y}=n^{-1} \sum_{i=1}^n Y_i$。如果已知函数类 $\mathcal{H}$，我们可以使用 $\left|\widehat{\rho}_g\left(X_j, Y\right)\right|$ 筛选重要特征。

另外，我们也可以对响应变量进行函数转换，并定义转换后的响应变量和协变量之间的相关系数。一般地，转换回归模型被定义为

$$H\left(Y_i\right)=\boldsymbol{X}_i^{\mathrm{T}} \boldsymbol{\beta}+\varepsilon_i \tag{8.3.1}$$

Li 等 [94] 通过在模型（8.3.1）中假定 $H(\cdot)$ 严格单调，提出使用秩相关系数衡量每个预测变量的重要性。他们没有使用之前定义的样本 Pearson 相关系数，而是提出了边际秩相关系数

$$\widehat{\omega}_j=\frac{1}{n(n-1)} \sum_{i \neq l}^n I\left(X_{ij}<X_{lj}\right) I\left(Y_i<Y_l\right)-\frac{1}{4}$$

来衡量第 j 个预测变量 X_j 对响应变量 Y 的重要性，并将方法命名为 RCS（Rank Correlation Screening）。注意边际秩相关系数等于响应变量与第 j 个预测变量之间的*Kendall* τ 相关系数的四分之一。因此，我们可以基于秩相关系数 $\widehat{\omega}_j$ 的绝对值筛选重要预测变量，即

$$\widehat{\mathcal{M}}_{\kappa_n}=\left\{1 \leq j \leq p:\left|\widehat{\omega}_j\right|>\kappa_n\right\}$$

其中 κ_n 为预设定的阈值。

8.3.2 确定独立秩筛选

我们知道，X_j 和 Y 之间的 Pearson 相关系数仅仅刻画了 X_j 和 Y 之间的线性相关性。Zhu 等人 [95] 提出使用 X_j 和 $I(Y<y)$ 的 Pearson 相关系数来刻画 X_j 和 Y 之间的非线性相关性，因为示性函数具有单调变换不变性，即针对单调函数 $g(\cdot)$，有 $I(Y<y)=I(g(Y)<g(y))$。当 $j=1,\ldots,p$ 时，假设 $\mathrm{E}(X_j)=0$，$\mathrm{Var}(X_j)=1$，我们得到随机变量 X_j 和 $I(Y<y)$ 相关系数表示为

$$\Omega_j(y)=\mathrm{cov}\{X_j,I(Y<y)\}=\mathrm{E}[X_j\ \mathrm{E}\{I(Y<y)\mid X_j\}]$$

直观地说，如果 X_j 和 Y 是独立的，对于任意的 y，都有 $\Omega_j(y)=0$。另一方面，如果 X_j 和 Y 是相关的，则存在 y 使得 $\Omega_j(y)\neq 0$。他们提出使用

$$\omega_j=\mathrm{E}\left\{\Omega_j^2(Y)\right\},\quad j=1,\ldots,p$$

作为特征筛选的边际统计量。由于 ω_j 为正，可以使用 ω_j 对所有预测变量进行排序。

假设 $\{(\boldsymbol{X}_i,Y_i),i=1,\cdots,n\}$ 是来自 $\{\boldsymbol{X},Y\}$ 的随机样本。为了便于表示，我们假设样本预测变量都是标准化的。对于任意给定的 y，$\Omega_j(y)$ 的样本矩估计量为

$$\widehat{\Omega}_j(y)=n^{-1}\sum_{i=1}^{n}X_{ij}I\left(Y_i<y\right)$$

因此，ω_j 的估计量为

$$\widehat{\omega}_j=\frac{1}{n}\sum_{j=1}^{n}\widehat{\Omega}_j^2\left(Y_j\right)=\frac{1}{n}\sum_{j=1}^{n}\left\{\frac{1}{n}\sum_{i=1}^{n}X_{ij}I\left(Y_i<Y_j\right)\right\}^2,\quad j=1,\ldots,p$$

接下来，使用 $\widehat{\omega}_j$ 对所有候选预测变量 $X_j,j=1,\ldots,p$ 的重要性进行排序，然后选择最前面的几个作为重要预测变量，此过程被命名为确定独立秩筛选，简记为 SIRS (Sure Independent Ranking Screening)。

8.3.3 距离相关系数

这一章介绍另一种利用相关系数来有效地度量预测变量与响应变量之间的线性与非线性相关关系，来进行超高维数据的特征筛选。Li，Zhong 和 Zhu [96] 提出了一种基于距离相关系数筛选方法 (DCS, Distance Correlation Screening)。前面介绍的 Pearson 相关系数、秩相关系数和广义相关系数仅仅定义了两个随机变量的相关关系，而距离相关系数是定义了两个不同维度的随机向量的关系。

首先，定义两个随机向量 $\boldsymbol{U} \in R^{q_1}$ 和 $\boldsymbol{V} \in R^{q_2}$ 之间的距离协方差为

$$\mathrm{dcov}^2(\boldsymbol{U},\boldsymbol{V}) = \int_{R^{q_1+q_2}} \left\|\phi_{\boldsymbol{U},\boldsymbol{V}}(\boldsymbol{t},\boldsymbol{s}) - \phi_{\boldsymbol{U}}(\boldsymbol{t})\phi_{\boldsymbol{V}}(\boldsymbol{s})\right\|^2 w(\boldsymbol{t},\boldsymbol{s})d\boldsymbol{t}d\boldsymbol{s} \tag{8.3.2}$$

其中 $\phi_{\boldsymbol{U}}(\boldsymbol{t})$ 和 $\phi_{\boldsymbol{V}}(\boldsymbol{s})$ 是 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 的边际特征函数，$\phi_{\boldsymbol{U},\boldsymbol{V}}(\boldsymbol{t},\boldsymbol{s})$ 是 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 的联合特征函数，并且

$$w(\boldsymbol{t},\boldsymbol{s}) = \left\{c_{q_1}c_{q_2}\|\boldsymbol{t}\|_{q_1}^{1+q_1}\|\boldsymbol{s}\|_{q_2}^{1+q_2}\right\}^{-1}$$

$c_d = \pi^{(1+d)/2}/\Gamma\{(1+d)/2\}$（这个选择是为了方便微分的计算）。这里 $\|\phi\|^2 = \phi\bar{\phi}$, ϕ 表示复值函数，$\bar{\phi}$ 是 ϕ 的共轭。由定义（8.3.2）可知，当且仅当 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是独立时，$\mathrm{dcov}^2(\boldsymbol{U},\boldsymbol{V}) = 0$。Székely，Rizzo 和 Bakirov [97] 证明了

$$\mathrm{dcov}^2(\boldsymbol{U},\boldsymbol{V}) = S_1 + S_2 - 2S_3$$

其中

$$S_1 = \mathrm{E}(\|\boldsymbol{U}-\widetilde{\boldsymbol{U}}\|\|\boldsymbol{V}-\widetilde{\boldsymbol{V}}\|), \quad S_2 = \mathrm{E}(\|\boldsymbol{U}-\widetilde{\boldsymbol{V}}\|)\mathrm{E}(\|\boldsymbol{U}-\widetilde{\boldsymbol{V}}\|)$$

$$S_3 = \mathrm{E}\{\mathrm{E}(\|\boldsymbol{U}-\widetilde{\boldsymbol{U}}\| \mid \boldsymbol{U})\mathrm{E}(\|\boldsymbol{V}-\widetilde{\boldsymbol{V}}\| \mid \boldsymbol{V})\}$$

并且 $(\widetilde{\boldsymbol{U}},\widetilde{\boldsymbol{V}})$ 是 $(\boldsymbol{U},\boldsymbol{V})$ 的独立副本。因此，$\boldsymbol{U}$ 和 $\boldsymbol{V}$ 之间的距离协方差可以通过代入对应的样本来估计。具体来说，基于总体 $(\boldsymbol{U},\boldsymbol{V})$ 中的随机样本 $\{(\boldsymbol{U}_i,\boldsymbol{V}_i), i = 1,\cdots,n\}$，我们有

$$\widehat{\mathrm{dcov}}^2(\boldsymbol{U},\boldsymbol{V}) = \widehat{S}_1 + \widehat{S}_2 - 2\widehat{S}_3$$

其中，

$$\widehat{S}_1 = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\|\boldsymbol{U}_i-\boldsymbol{U}_j\|\,\|\boldsymbol{V}_i-\boldsymbol{V}_j\|$$

$$\widehat{S}_2 = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\|\boldsymbol{U}_i-\boldsymbol{U}_j\|\,\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\|\boldsymbol{V}_i-\boldsymbol{V}_j\|$$

$$\widehat{S}_3 = \frac{1}{n^3}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\|\boldsymbol{U}_i-\boldsymbol{U}_l\|\,\|\boldsymbol{V}_j-\boldsymbol{V}_l\|$$

根据 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 的距离相关系数定义

$$\mathrm{dcorr}(\boldsymbol{U},\boldsymbol{V}) = \mathrm{dcov}(\boldsymbol{U},\boldsymbol{V})/\sqrt{\mathrm{dcov}(\boldsymbol{U},\boldsymbol{U})\,\mathrm{dcov}(\boldsymbol{V},\boldsymbol{V})}$$

根据以上距离协方差阵的估计过程，可以得到距离相关系数估计量

$$\widehat{\mathrm{dcov}}(\boldsymbol{U},\boldsymbol{V})/\sqrt{\widehat{\mathrm{dcov}}(\boldsymbol{U},\boldsymbol{U})\widehat{\mathrm{dcov}}(\boldsymbol{V},\boldsymbol{V})}$$

假设响应变量 $\boldsymbol{Y}$ 是多维度的，如果从 p 个预测变量 X_j 中筛选出重要的预测变量，我们提出使用如下边际距离相关系数估计量:

$$\widehat{\omega}_j = \widehat{\mathrm{dcov}}(\boldsymbol{Y},X_j)/\sqrt{\widehat{\mathrm{dcov}}(\boldsymbol{Y},\boldsymbol{Y})\widehat{\mathrm{dcov}}(X_j,X_j)}$$

并且使用 $\widehat{\omega}_j^2$ 来对预测变量的重要性进行排序。

8.4 高维分类数据的特征筛选

8.4.1 Kolmogorov-Smirnov 统计量

Fan 和 Fan [98] 提出在高维二值分类中使用双样本 t 检验统计量作为特征筛选的边际统计量。尽管基于双样本 t 检验统计量 [98] 的特征筛选在高维分类问题中表现很好，但它可能会被重尾分布或者具有异常值的数据破坏，而且它是基于模型的，当数据结构不满足假定模型时，筛选结果也是失效的。

为了克服这些缺点，Mai 和 Zou [99] 提出了一种新的基 Kolmogorov-Smirnov 统计量的二值分类特征筛选方法。为了便于标记，重新标记 $Y = +1, -1$ 为类标签。令 $F_{1j}(x) = \Pr(X_j \leq x \mid Y = 1)$ 和 $F_{2j}(x) = \Pr(X_j \leq x \mid Y = -1)$ 分别为给定 $Y = 1, -1$ 时 X_j 的条件分布函数。因此，如果 X_j 和 Y 独立，则 $F_{1j}(x) \equiv F_{2j}(x)$。基于这一观察，上述条件分布函数的差可用于构建特征筛选的方法。

因此 Mai 和 Zou [99] 提出 Kolmogorov-Smirnov 边际效用：

$$\omega_j = \sup_{x \in \mathcal{R}} |F_{1j}(x) - F_{2j}(x)|$$

并且将

$$\widehat{\omega}_j = \sup_{x \in \mathcal{R}} \left| \widehat{F}_{1j}(x) - \widehat{F}_{2j}(x) \right|$$

作为特征筛选的边际统计量，Mai 和 Zou [99] 将这种特征筛选方法命名为Kolmogorov-Smirnov 统计量 (KS)，其中 $\widehat{F}_{1j}(x)$ 和 $\widehat{F}_{2j}(x)$ 为对应的经验条件分布函数，即

$$\widehat{F}_{1j}(x) = \frac{1}{n_1} \sum_{i:Y_i=1} I(X_{ij} \leq x), \quad \widehat{F}_{2j}(x) = \frac{1}{n_2} \sum_{i:Y_i=-1} I(X_{ij} \leq x)$$

其中 $n_1 = \sum_{i=1}^{n} I(Y_i = 1)$ 和 $n_2 = \sum_{i=1}^{n} I(Y_i = -1)$，最后使用 $\widehat{\omega}_j$ 来对预测变量的重要性进行排序。

8.4.2 均值-方差统计量

Cui，Li 和 Zhong [100] 提出了一种利用均值-方差指数进行超高维分类问题的确定独立筛选方法，形式如下：

$$\mathrm{MV}(X_j, Y) = \mathrm{E}_{X_j}\left[\mathrm{Var}_Y\left(F\left(X_j \mid Y\right)\right)\right] \tag{8.4.1}$$

它不仅保留了 Kolmogorov 滤波器的优点，而且允许分类响应变量具有 $O(n^{\kappa})$ 个发散的类别，其中 $\kappa \geq 0$。假设分类响应变量 Y 有 K 个类别 $\{y_1, \cdots, y_K\}$。令 $F_j(x) = \Pr(X_j \leq x)$ 表示第 j 个特征 X_j 的边际分布函数，$F_{jk}(x) = \Pr(X_j \leq x \mid Y = y_k)$ 表示给定 $Y = y_k$ 时 X_j 的条件分布函数。由于 Y 是离散响应变量，Cui, Li 和 Zhong [100] 推导出

$$\mathrm{MV}(X_j, Y) = \sum_{k=1}^{K} p_k \int [F_{jk}(x) - F_j(x)]^2 \, dF_j(x)$$

如果 X_j 和 Y 在统计上是独立的，那么对于任意的 k 和 x，理论上都有 $F_{jk}(x) = F_j(x)$。通过上式推导，均值-方差这是给定 $Y = y_k$ 时 X_j 的条件分布函数与 X_j 的无条件分布函数之间的 Cramér-von Mises 距离的加权平均，其中 $p_k = \Pr(Y = y_k)$。他们进一步表明当且仅当 X_j 和 Y 在统计上独立时，$\mathrm{MV}(X_j \mid Y) = 0$。

设 $\{(X_{ij}, Y_i) : 1 \leq i \leq n\}$ 是总体 (X_j, Y) 中大小为 n 的随机样本，则均值方差估计量是

$$\widehat{\mathrm{MV}}(X_j, Y) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i=1}^{n} \widehat{p}_k \left[\widehat{F}_{jk}(X_{ij}) - \widehat{F}_j(X_{ij}) \right]^2$$

其中 $\widehat{p}_k = n^{-1} \sum_{i=1}^{n} I\{Y_i = y_k\}$, $\widehat{F}_{jk}(x) = n^{-1} \sum_{i=1}^{n} I\{X_{ij} \leq x, Y_i = y_k\} / \widehat{p}_k$, $\widehat{F}_j(x) = n^{-1} \sum_{i=1}^{n} I\{X_{ij} \leq x\}$。因此我们可以使用 $\widehat{\mathrm{MV}}(X_j, Y)$ 筛选出重要的预测变量。这个过程被称为基于 MV 的确定独立筛选方法。

8.4.3 类别自适应筛选统计量

假设观察到具有 K $(K > 2)$ 类的分类响应变量 Y，即 $\{y_1, y_2, \ldots, y_K\}$，对于所有的 $k = 1, \ldots, K$ 都有 $p_k = \Pr(Y = y_k) > 0$。在大数据时代，不同分类变量响应变量的样本的来源可能不同，例如在不同的时间段或者在不同的实验方法下收集到。换句话说，具有分类响应变量的高维数据通常是异构的。因此，我们考虑

(i) (异构性) 一组重要的预测变量 $\mathcal{A}_k \equiv \{1 \leq j \leq p : \Pr(W_k \leq w_k \mid \boldsymbol{X})$ 依赖于 $X_j\}$，其中对于不同的 $k = 1, \ldots, K$，$W_k \equiv I(Y = y_k)$ 重要变量集合 $\mathcal{A}_k$ 可能不同；

(ii) (稀疏性) 对于某个常数 $\alpha > 0$ 的维度 $p = o\{\exp(n^{\alpha})\}$，$|\mathcal{A}_k| = s_k = o(n)$，其中 $|\mathcal{A}_k|$ 是 $\mathcal{A}_k$ 的基数，n 是样本大小。

为了寻找不同类别下的重要变量，Xie 等 [101] 考虑了给定 $I(Y = y_k)$ 时 X_j 的条件分布函数，即 $F_{jk}(x) = \Pr(X_j \leq x \mid Y = y_k) = \dfrac{\Pr(X_j \leq x, Y = y_k)}{\Pr(Y = y_k)}$，并提出如下边际筛选指数

$$\tau_{jk} = E_{X_j}\{F_{jk}(X_j)\} - \frac{1}{2}$$

很明显，如果 $I(Y=y_k)$ 与 X_j 独立，则 $\tau_{jk}=0$。令 $\{(\boldsymbol{X}_i,Y_i),i=1,\ldots,n\}$ 是独立同分布随机样本。定义 $\widehat{p}_k=\dfrac{1}{n}\sum_{i=1}^{n}I(Y_i=y_k),k=1,\ldots,K$。我们得到 $\tau_{jk},j=1,\ldots,p$ 的样本估计为

$$\widehat{\tau}_{jk}=\frac{1}{n+1}\sum_{l=1}^{n}\left\{\frac{1}{n}\sum_{i=1}^{n}\frac{I(X_{ij}\leq X_{lj},Y_i=y_k)}{\widehat{p}_k}\right\}-\frac{1}{2} \tag{8.4.2}$$

接下来，我们根据 $|\widehat{\tau}_{jk}|$ 值的大小，对所有候选预测变量 $X_j,j=1,\ldots,p$ 的重要性进行排序。$|\widehat{\tau}_{jk}|$ 也成为类别自适应筛选统计量，并简记为 CAS（Category Adaptive Screening）方法。

如果研究者对影响一个类集 $\mathcal{G}\subset\{1,2,\ldots,K\}$ 的重要变量感兴趣，可以采用如下筛选统计量

$$\widehat{\tau}_{j,\mathcal{G}}\equiv\sup_{k\in\mathcal{G}}|\widehat{\tau}_{jk}|$$

特别地，如果筛选出影响到所有类别的重要变量，我们采用

$$\widehat{\tau}_{j}\equiv\sup_{k\in\{1,2,\ldots,K\}}|\widehat{\tau}_{jk}|$$

因此，类别自适应筛选统计量方法非常灵活，既可以筛选出影响某一类的重要变量，也可以用于筛选影响某一类集以及所有类的重要变量。

8.5 特征筛选实践

8.5.1 R 语言实践

下面介绍怎么用 R 语言实现上面提到的特征筛选方法。

基于边际模型拟合的特征筛选

首先介绍如何使用 R 语言实现基于边际模型拟合的特征筛选方法。首先我们需要定义生成数据的函数。

```
###SIS 和 MMLE 生成数据的函数###
gdat_linear<- function(n,p,beta,rho){
  M1 <- matrix(rep(1:p,p),ncol=p,byrow=F)
  corM <- rho^(abs(M1-t(M1)))
  X<- mvrnorm(n,rep(0,p),corM)#生成多元正态数
  epsilon <- rnorm(n)#生成n个随机正态分布的数
  Y <- beta[1]*X[,1]+beta[2]*X[,2]+beta[3]*X[,3] + epsilon
```

```
  dat <- list(X = X,Y = Y)
  return(dat)
}
```

接下来我们定义确定独立筛选（SIS）[92] 的函数，即边际最小二乘估计。

```
SIS<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  s<-matrix(0,nrow=p,ncol=1)
  Y<-Y-mean(Y)#centered y
  X<-apply(X,2, function(x) (x-mean(x))/sd(x))#X每列标准化
  for(j in 1:p){
    x0<-X[,j]
    s[j,] <-abs(mean(Y*X[,j]))
  }
  list(rank=(p+1)-rank(s))
}
```

下面是求取边际极大似然估计量（MMLE）[102] 的函数。

```
MMLE<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  s<-c()
  Y<-Y-mean(Y)#centered y
  X<-apply(X,2, function(x) (x-mean(x))/sd(x))
  for(j in 1:p){
    s[j] <-glm(Y ~X[,j], family = gaussian())$coefficients[2]#广义线性模型
  }
  list(rank=(p+1)-rank(s))
}
```

定义了生成数据的函数、SIS 和 MMLE 函数，接下来的程序是调用函数。

```
library(MASS)
n = 50#样本量
p = 500#维数
rho<-0.5
beta <- c(1,1,0.8,rep(0,p-3))#参数真实值
d = floor(n/log(n))#前d个最重要的变量
###生成数据###
zdata <- gdat_linear(n,p,beta,rho)
Y <- zdata$Y
X <- zdata$X
SIS(X,Y)$rank[1:d]
MMLE(X,Y)$rank[1:d]
```

经过运行函数，我们筛选出前 $\lceil n/\log(n) \rceil$ 个重要特征：

```
###SIS###
[1]   3   1   2   7   9  10   4  87 245 459  69 452
###MMLE###
[1]   3   1   2   6   8   9   4  38 115 219 462 216
```

接下来再次定义生成数据的函数，以下程序是为非参数独立筛选（NIS）方法生成数据的函数。

```
gdat_nonlinear<- function(n,p,beta){
  X <- array(0,dim = c(n,p))
  for(s in 1:p){
    X[,s] = runif(n)#生成从0到1区间范围内的服从正态分布的随机数
  }
  epsilon <- rnorm(n)
  Y <- beta[1]*X[,1]+beta[2]*tan(X[,2])+beta[3]*X[,3]^3 + epsilon
  dat <- list(X = X,Y = Y)
  return(dat)
}
```

下面程序是非参数独立筛选（NIS）[103] 的函数。

```
NIS<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  s<-matrix(0,nrow=p,ncol=1)
  bfit<-matrix(0,nrow=n,ncol=p)
  Y<-Y-mean(Y)#centered y
  #X<-apply(X,2,function(x) (x-min(x))/(max(x)-min(x)))#X每列标准化
  for(j in 1:p){
    x0<-X[,j]
    knots=quantile(x0,c(1/4,2/4,3/4))#结点
    a=bs(x0, knots=knots,degree=1)#B样条
    b=lm(Y~a)#线性拟合
    s[j,] <-sum((b$fitted)^2)/n
    bfit[,j]<-b$fitted
  }
  list(rank=(p+1)-rank(s),fit=bfit,wk=s)
}
```

我们已经定义了生成数据的函数和非参数独立筛选（NIS）函数，下面程序是演示如何使用 NIS 筛选特征的过程。

```
###生成数据、运行定义筛选方法的函数###
library(splines)
library(MASS)
n = 200#样本量
p = 2000#维数
beta <- c(4,2,5,rep(0,p-3))#参数真实值
d = floor(n/(4*log(n)))#前d个最重要的变量
###生成数据###
zdata <- gdat_nonlinear(n,p,beta)
Y <- zdata$Y
X <- zdata$X
NIS(X,Y)$rank[1:d]
```

经过运行以上程序，我们筛选出前 $\lceil n/(4*\log(n))\rceil$ 个特征：

```
###NIS###
```

```
[1]    2    3    1 1046 1306  871 1905 1181 1167
```

基于边际相关系数的特征筛选

下面介绍如何 R 语言实现基于边际相关系数的特征筛选方法，我们需要定义生成数据的函数。以下程序为生成数据的函数。

```
####生成数据的函数#############
gdat<- function(n,p,beta,rho){
  M1 <- matrix(rep(1:p,p),ncol=p,byrow=F)
  corM <- rho^(abs(M1-t(M1)))
  X<- mvrnorm(n,rep(0,p),corM)
  epsilon <- rnorm(n)
  Y <- beta[1]*exp(X[,1])+beta[2]*tan(X[,2])+beta[3]*X[,3]^3 + epsilon
  dat <- list(X = X,Y = Y)
  return(dat)
}
```

接下来我们定义秩相关筛选（RCS）[94] 的函数。

```
RCS<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  w<-c()
  for(j in 1:p){
    a<-0
    for(i in 1:n){
      a<-a+sum((X[,j]<X[i,j])*(Y<Y[i]))/n#找到X[:,j]中小于X[i,j]和Y中小于Y[i]的做计算
    }
    w[j]<-abs(a/(n-1)-1/4)
  }
  list(rank=(p+1)-rank(w))
}
```

下面是确定独立秩筛选（SIRS）[95] 方法的函数。

```
SIRS<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  w<-matrix(0, nrow=p, ncol=1)
  for(k in 1:p){
    w.k.j<-NULL
    for(j in 1:n){
      s<-(t(X[,k])%*%(1*(Y<Y[j]))/n)^2#将X[:,k]的转置与（Y中小于Y[j]做计算
      w.k.j<-c(w.k.j,s)
    }
    w[k,]<-(n^3*(mean(w.k.j)))/(n*(n-1)*(n-2))
  }
  list(rank=(p+1)-rank(w))
}
```

下面程序定义了距离相关系数筛选（DCS）[96] 方法的函数。

```
DCS<-function(X,Y){
  p=dim(X)[2]
  n=length(Y)
  s<-matrix(0, nrow=p, ncol=1)
  for(j in 1:p){
    s[j]<-bcdcor(X[,j],Y)#距离相关系数
  }
  list(rank=(p+1)-rank(s))
}
```

我们已经定义了生成数据的函数、秩相关筛选（RCS）、确定独立秩筛选（SIRS）、距离相关系数筛选（DCS）方法的函数。下面程序演示了使用这三种方法进行特征筛选的过程。

```
###生成数据、运行定义筛选方法的函数###
library(energy)
library(MASS)
n = 200#样本量
p = 1000#维数
rho<-0.5
beta <- c(4,2,5,rep(0,p-3))#参数真实值
d = floor(n/(4*log(n)))#前d个最重要的变量
zdata <- gdat(n,p,beta,rho)
Y <- zdata$Y
X <- zdata$X
RCS(X,Y)$rank[1:d]
SIRS(X,Y)$rank[1:d]
DCS(X,Y)$rank[1:d]
```

经过运行以上程序，我们分别用三种方法筛选出了 $\lceil n/(4\log(n))\rceil$ 个特征：

```
###RCS###
[1]   2   3   1   4  36  25 493 402 131
###SIRS###
[1]   2   3   1  33 194 483 128 260 202
###DCS###
[1]   2   3   1   4  71  28 346 584 172
```

高维分类数据的特征筛选

下面介绍如何使用 R 语言实现高维分类数据的特征筛选方法。首先我们要定义生成数据的函数。

```
###生成数据的函数###
gdat=function(n,p,R){
  epsilon <- array(rnorm(n*p),dim = c(n,p))
  P <- c(50/100,50/100)
  #Y <- rbinom(n,1,P) + 1
  Y <- sample(1:R,n,replace =  T,prob=P)#在1:R中以P的概率随机重复抽样n个数
  X <- array(0,dim=c(n,p))
```

```
  u <- array(0,dim=c(R,p))
  diag(u) <- 1.5#对角线为1.5
  X <- u[Y,] + epsilon
  dat <- list(X=X,Y=Y)
  return(dat)
}
```

下面程序定义的是通过 Kolmogorov-Smirnov 统计量 [99] 进行特征筛选的方法。

```
KS<-function(X,Y){
  n<-nrow(X)
  p<-ncol(X)
  data_ks = t(cbind(Y,X))
  ks_index = order(data_ks[1,])#按从小到大的顺序排，返回索引
  data_m0 = data_ks[,ks_index]
  n_1 = length(which(Y==1))
  n_2 = n - n_1
  tau_ks<-c()
  for(j in 1:p){
    group1 = data_m0[(j+1),1:n_1]
    group2 = data_m0[(j+1),(n_1+1):n]
    tau_ks[j] = ks.test(group1,group2)$statistic#做Kolmogorov-Smirnov检验
  }
  list(rank=(p+1)-rank(tau_ks))
}
```

接下来我们定义均值方差统计量筛选 [100] 函数。

```
MV <- function(X,Y){
  n = nrow(X)
  p = ncol(X)
  pr = rep(0,R)
  n_1 = length(which(Y==1))
  n_2 = n - n_1
  pr[1] = n_1/n
  pr[2] = n_2/n
  mv<-c()
  for(j in 1:p){
    F1 = apply(outer(X[which(Y==1),j],t(X[,j]),"<"),3,t)*1
    F2 = apply(outer(X[which(Y==2),j],t(X[,j]),"<"),3,t)*1
    FF = apply(outer(X[,j],t(X[,j]),"<"),3,t)*1
    mv[j] = sum(pr[1]*(apply(F1,2,sum)/n/pr[1] - apply(FF,2,sum)/n)^2 +
    pr[2]*(apply(F2,2,sum)/n/pr[2] - apply(FF,2,sum)/n)^2)/n
  }
  list(rank=(p+1)-rank(mv))
}
```

下面程序定义的是类别自适应特征筛选 [101] 函数。

```
CAS<- function(X,Y){
  n = nrow(X)
  p = ncol(X)
  tau1 = 0
```

```
  tau2 = 0
  n_1 = length(which(Y==1))
  n_2 = n - n_1
  n = n_1 + n_2
  for(i in 1:n_1){
    for(j in 1:n){
      tau1 = tau1 + (X[i,] <= X[j,])
    }
  }
  for(i in 1:n_2){
    for(j in 1:n){
      tau2 = tau2 + (X[i+n_1,] <= X[j,])
    }
  }
  Tau1 = tau1/n/n_1 - 1/2
  Tau2 = tau2/n/n_2 - 1/2
  Tau = cbind(abs(Tau1), abs(Tau2))
  tau_sp_max = apply(Tau,1,max)#按行找出最大值
  list(rank=(p+1)-rank(tau_sp_max))
}
```

在前面我们分别定义了通过 Kolmogorov-Smirnov 统计量进行特征筛选、均值方差特征筛选、类别自适应特征筛选的函数，以及生成数据的函数。接下来我们通过示例来展示如何使用这些方法进行特征筛选。

```
###生成数据、运行定义筛选方法的函数###
n = 50#样本量
p = 500#维数
R = 2#二值响应变量
d = floor(n/log(n))#前d个最重要的变量
zdata <- gdat(n,p,R)
Y <- zdata$Y
X <- zdata$X
data_full = cbind(Y,X)
data_full1 = data_full[which(Y == 1),]
data_full2 = data_full[which(Y == 2),]
outdata = rbind(data_full1,data_full2)
Y_full = outdata[,1]
X_full = outdata[,2:(p+1)]
MV(X_full,Y_full)$rank[1:d]
KS(X_full,Y_full)$rank[1:d]
CAS(X_full,Y_full)$rank[1:d]
```

经过运行以上程序，我们分别用三种方法筛选出了 $\lceil n/\log(n) \rceil$ 个特征：

```
###MV###
[1]   1   2 416 343 198 357  76 474 166 396 115 175
###KS###
[1]   2.0   1.0 112.0 243.5 247.0 310.0 267.5 255.0 495.5 230.5 370.0 238.5
###CAS###
[1]   1.0   2.0 279.5 261.0 187.0 258.0  68.0 379.0 220.5 473.5 141.0 168.5
```

8.5.2 Python 语言实践

下面介绍怎么用 Python 语言实现上面提到的特征筛选方法。

基于边际模型拟合的特征筛选

首先介绍如何使用 Python 语言实现基于边际模型拟合的特征筛选方法。首先我们需要导入需要的库和函数。

```
import numpy as np
import math
import statsmodels.api as sm
from scipy.interpolate import BSpline
from sklearn import linear_model
```

接下来是定义生成数据的函数。

```
### SIS和MMLE生成数据的函数 ###
def gdat_linear(n,p,beta,rho):
    a = np.arange(1, p + 1).reshape(p, 1)
    M1 = np.tile(np.mat(a), (1, p))#将a在列上重复p次
    corM = np.float_power(rho*np.ones_like(M1), abs(M1 - M1.T))
    X = np.random.multivariate_normal(np.arange(1, p + 1), corM, size=n)
    #生成多元正态分布矩阵
    epsilon = np.random.normal(size=(1, n))#随机产生服从正态分布的数
    Y = beta[0] * X[:, 0] + beta[1] * X[:, 1] + beta[2] * X[:, 2] + epsilon#构成Y
    return {"X":X,"Y":Y}
```

下面程序是确定独立特征筛选（SIS）[92] 的函数，即边际最小二乘估计。

```
### Sure independence screening (SIS) by Fan and Lv (2008) ###
def SIS(X,Y):
    p = X.shape[1]#获取X矩阵的列数
    n = len(Y)#获取Y的长度
    s = []
    Y = Y - np.mean(Y) #centered y
    X = (X - np.mean(X, 0)) / np.std(X, 0)#标准化X
    for j in range(0,p):
        x0 = X[:, j]
        s.append(abs(np.mean(Y * x0)))#将Y与X中的每一列进行计算
    rank=(p - 1) - np.argsort(s)#rank为s中的元素从大到小排列其对应的index(索引)
    return rank
```

接下来是边际极大似然估计量（MMLE）[102] 的函数。

```
### Marginal Maximum Likelihood Estimator (MMLE) by Fan and Song (2010)###
def MMLE(X,Y):
    p = X.shape[1]#列数
    n = len(Y)
    s = []
```

```
    Y = Y-np.mean(Y) #centered y
    f = lambda x:(x-np.mean(x))/np.std(x)#创建一个标准化函数
    X = np.apply_along_axis(f,1,X)#在X的每一列上应用这个标准化函数
    for j in range(p):
        model = sm.GLM(Y[0],X[:,j]).fit()#默认是gauss分布,广义线性模型
        s.append(model.params[0]) #找到模型的回归系数
    rank = (p - 1) - np.argsort(s)
    return rank
```

定义了一系列生成数据的函数、SIS 和 MMLE 函数，下面是运行函数。

```
### 运行函数 ###
n = 50### 样本量
p = 500### 维数
rho = 0.5
d = math.floor(n / math.log(n))###前d个最重要的变量
beta = np.hstack((np.array([1, 1, 0.8]), np.zeros(p + 1 - 3)))
zdata = gdat_linear(n, p, beta, rho)#运行函数gdat_linear
X = zdata["X"]
Y = zdata["Y"]
print(SIS(X, Y)[0:d])#选取SIS方法中前d个变量
print(MMLE(X, Y)[0:d])#选取MMLE方法中前d个变量
```

我们得到了以下 $\lceil n/\log(n)\rceil$ 个特征。

```
###SIS###
[385  25 124  11 284 299 386 278 159 420 485 389]
###MMLE###
[250 246 256 257 240 255 243 239 232 244 271 233]
```

接下来是为非参数独立筛选（NIS）方法生成数据的函数。

```
### NIS生成数据的函数 ###
def gdat_nonlinear(n, p, beta):
    X = np.zeros((n, p))
    for s in range(p):
        X[:, s] = np.random.uniform(size=(1, n))#每一列生成n个从[0,1)中随机采样的数
    epsilon = np.random.normal(size=(1,n))#随机产生正态分布的数
    Y = beta[0]*X[:, 0] + beta[1]*np.tan(X[:, 1]) + beta[2]*X[:, 2]**3 + epsilon
    dat = {"X":X, "Y":Y}
    return dat
```

下面程序是非参数独立筛选（NIS）[103] 的函数。

```
###Nonparametric Independent Screening (NIS) by Fan et al.(2011)###
def NIS(X,Y):
    from sklearn.linear_model import LinearRegression
    p = X.shape[1]
    n = len(Y)
    s = []
    bfit = []
    Y = Y - np.mean(Y) #centered y
    lm = linear_model.LinearRegression()#线性模型
```

```
    for j in range(p):
        x0 = X[:, j]
        knots = np.percentile(x0, (25, 50, 75, 100)) #结点
        spl = BSpline(t=knots, c=x0, k=1) #B样条函数
        b = np.array([spl(0), spl(25), spl(50), spl(75), spl(100)]).reshape(-1, 1)
        model = lm.fit(b.T, Y)#线性模型拟合
        s.append((sum(model.intercept_)**2) / n)#将线性模型的回归系数计算
        bfit = model.intercept_
    rank = (p-1) - np.argsort(s)
    fit = bfit
    wk = s
    return rank
```

我们已经定义了生成数据的函数和非参数独立筛选（NIS）函数，下面展示运行程序。

```
### 运行函数 ###
n = 200### 样本量
p = 2000### 维数
beta = np.hstack((np.array([4,2,5]),np.zeros(p+1-3)))###参数真实值
d = math.floor(n / (4*math.log(n)))###前d个最重要的变量
###生成数据###
zdata = gdat_nonlinear(n,p,beta)#运行gdat_nonlinear函数
X = zdata["X"]
Y = zdata["Y"]
print(NIS(X,Y)[0:d])#选取NIS方法中前d个变量
```

经过运行以上函数，我们筛选出前 $\lceil n/(4*\log(n))\rceil$ 个特征。

```
[1999  658  659  660  661  662  663  664  665]
```

基于边际相关系数的特征筛选

下面介绍如何 Python 语言实现基于边际相关系数的特征筛选方法，我们需要定义生成数据的函数。以下为本节程序需要用到的库。

```
import numpy as np
import math
import dcor
```

下面程序是定义生成数据的函数。

```
### 生成数据的函数 ###
def gdat(n,p,beta,rho):
    a = np.arange(1, p+1).reshape(p, 1)
    M1 = np.tile(np.mat(a), (1, p))
    corM = np.float_power(rho*np.ones_like(M1), abs(M1-M1.T))
    X = np.random.multivariate_normal(np.arange(1, p+1), corM, size=n)
    #生成多元正态分布矩阵
    epsilon = np.random.normal(size=(1,n))#随机产生服从正态分布的数
    Y = beta[0,0]*X[:,0] + beta[0,1]*np.tan(X[:,1]) + beta[0,2]*X[:,2]**3 + epsilon
```

```
    dat = {"X":X,"Y":Y}
    return dat
```

接下来是定义秩相关筛选（RCS）[94] 的函数。

```
###Rank Correlation Screening (RCS) by Li et al.(2012)###
def RCS(X,Y):
    p = X.shape[1]
    n = len(Y)
    w = []
    for j in range(p):
        a = 0
        for i in range(n):
            a = a + sum(((X[:,j] < X[i,j])*(Y < Y[i])))/n
            #找到X[:,j]中小于X[i,j], 和Y中小于Y[i]的值做计算
        w.append(abs(a/(n-1) - 1/4))
    rank = (p-1) - np.argsort(w)
    return rank
```

下面程序是确定独立秩筛选（SIRS）[95] 方法的函数。

```
###Sure Independent Ranking and Screening (SIRS) by Zhu et al. (2011) ###
def SIRS(X, Y):
    p = X.shape[1]
    n = len(Y)
    w = []
    for k in range(p):
        w0 = []
        for j in range(n):
            s = np.dot(X[:, k].T,(1*(Y < Y[j]))/n)**2
            #将X[:,k]的转置 与 （Y中小于Y[j]）做计算
            w0.append(s)
        w.append(n**3 * (np.mean(w0)) / (n*(n-1)*(n-2)))
    rank = (p-1) - np.argsort(w)
    return rank
```

接下来定义的是距离相关系数筛选（DCS）[96] 方法。

```
###Distance Correlation Screening (DCS) by Li et al. (2012)###
def DCS(X, Y):
    p = X.shape[1]
    n = len(Y)
    s = []
    for j in range(p):
        s.append(dcor.distance_correlation(X[:,j ], Y))#获取X[:,j]与Y的距离相关系数
    rank = (p-1) - np.argsort(s)
    return rank
```

在上面我们定义了生成数据的函数、秩相关筛选、确定独立秩筛选、距离相关系数筛选方法的函数。下面程序展示三种方法的运行过程。

```
##运行函数
n = 200### 样本量
```

```
p = 1000### 维数
rho = 0.5
beta = np.hstack((np.matrix((4, 2, 5)),np.zeros((1, p+1-3))))###参数真实值
d = math.floor(n/(4*math.log(n)))###前d个最重要的变量

zdata = gdat(n,p,beta,rho)#运行gdat函数
Y = zdata["Y"]
X = zdata["X"]
datafull = np.hstack((X, Y.T))#水平方向合并矩阵X与Y
Y = datafull[:,p]#最后的第p列为Y
X = datafull[:,0:p]#前p-1列为X
print(RCS(X, Y)[0:d])#选取RCS方法中前d个变量
print(SIRS(X, Y)[0:d])#选取SIRS方法中前d个变量
print(DCS(X, Y)[0:d])#选取DCS方法中前d个变量
```

经过以上过程，我们筛选出以下 $\lceil n/(4*\log(n))\rceil$ 个特征。

```
###RCS###
[ 61 359  68 495 239  60 685 651 683]
###SIRS###
[999 998 997 996 995 994 993 992 991]
###DCS###
[662 405 969 962 165 680 464 280 283]
```

高维分类数据的特征筛选

下面介绍如何使用 Python 语言实现高维分类数据的特征筛选方法。首先我们要导入本节程序需要的库和函数。

```
import numpy as np
import itertools
from scipy.stats import ks_2samp
import math
import random
```

下面定义的是生成数据的函数。

```
### 生成数据的函数 ###
def gdat(n, p, R):
    epsilon = np.random.normal(size=(n, p))#随机产生服从正态分布的数
    P = np.array((50/100, 50/100))
    Y = np.random.choice(np.array((1, R)), n, p=P)
    X = np.zeros((n, p))
    u = np.zeros((R, p))
    if R<p:#将矩阵的对角线元素为1.5
        for i in range(R):
            u[i, i] = 1.5
    else:
        for i in range(p):
            u[i, i] = 1.5
```

```
    X = u[Y-1, :] + epsilon  #构成X
    dat = {"X":X,"Y":Y}
    return dat
```

接下来是通过 Kolmogorov-Smirnov 统计量 [99] 进行特征筛选的方法。

```
###Kolmogorov-Smirnov Statistics by Mai and Zhou (2013)###
def KS(X,Y):
    n = X.shape[0]#行数
    p = X.shape[1]#列数
    data_ks = np.hstack((np.mat(Y).T, X)).T#构成[Y.T,X]矩阵
    ks_index = np.lexsort(data_ks[0, :])#从小到大排序，返回数组
    data_m0 = data_ks[:, ks_index[0]]#按照索引ks_index构成data_m0
    n_1 = len(np.where(Y == 1)[0])#找到Y中等于1的数共有多少个
    n_2 = n - n_1
    tau_ks = []
    for j in range(p):
        group1 = data_m0[(j+1), 0:n_1]
        group2 = data_m0[(j+1), (n_1):n]
        group1 = list(itertools.chain(*group1.tolist()))
        group2 = list(itertools.chain(*group2.tolist()))
        tau_ks.append(ks_2samp(group1,group2)[0])#执行Kolmogorov-Smirnov检验
    rank = (p-1) - np.argsort(tau_ks)
    return rank
```

下面程序是均值方差统计量筛选 [100] 方法。

```
### Mean-Variance Screening by Cui et al., (2015)###
def MV(X,Y):
    n = X.shape[0]#行数
    p = X.shape[1]#列数
    pr = np.zeros(R)
    n_1 = len(np.where(Y == 1)[0])#找到Y中值为1的个数
    n_2 = n - n_1
    pr[0] = n_1 / n
    pr[1] = n_2 / n
    mv = []
    for j in range(p):
        #找到Y中值为1的相对应的X[:,j]列的值，与X[:,j]中每个值做比较
        F1 = [[1 if bb < aa else 0 for bb in X[np.where(Y == 1), j][0]]
        for aa in X[:, j].T]
        #找到Y中值为2的相对应的X[:,j]列的值，与X[:,j]中每个值做比较
        F2 = [[1 if bb < aa else 0 for bb in X[np.where(Y == 2), j][0]]
        for aa in X[:, j].T]
        #将X[:,j]中每个值互相做比较
        FF = [[1 if bb < aa else 0 for bb in X[:, j]] for aa in X[:,j].T]
        #每列应用求和函数
        F1_apply = np.apply_along_axis(lambda s:sum(s), 1, np.array(F1))/n/pr[0]
        FF_apply = np.apply_along_axis(lambda s:sum(s), 1, np.array(FF))
        F1_apply_sub_FF_apply = pr[0]*((F1_apply-FF_apply)**2)
        F2_apply = np.apply_along_axis(lambda s:sum(s), 1, np.array(F2))/n/pr[1]
        FF_apply = np.apply_along_axis(lambda s:sum(s), 1, np.array(FF))
        F2_apply_sub_FF_apply = pr[1]*((F2_apply-FF_apply)**2)
```

```
        mv.append(sum(F1_apply_sub_FF_apply+F2_apply_sub_FF_apply)/n)
    return (p-1) - np.argsort(mv)
```

接下来是类别自适应特征筛选 [101] 方法。

```
### Category Adaptive Screening by Xie et al., (2020)###
def CAS(X, Y):
    n = X.shape[0]#行数
    p = X.shape[1]#列数
    tau1 = 0
    tau2 = 0
    n_1 = len(np.where(Y == 1)[0])#找到Y中等于1的值共有几个
    n_2 = n - n_1
    n = n_1 + n_2
    for i in range(n_1):
        for j in range(n):
            tau1 = tau1 + (X[i, :] <= X[j, :])
            #找到X前n_1行中小于等于X[j,:]中的个数累计做和
    for i in range(n_2):
        for j in range(n):
            tau2 = tau2 + (X[i+n_1, :] <= X[j, :])
            #找到X从n_1+1行到最后一行中小于等于X[j,:]中的个数累计做和
    Tau1 = tau1/n/n_1 - 1/2
    Tau2 = tau2/n/n_2 - 1/2
    Tau = np.vstack((abs(Tau1), abs(Tau2)))
    tau_sp_max = Tau.max(axis=0)#选取每行的最大值
    rank = (p-1) - np.argsort(tau_sp_max)
    return rank
```

以上我们定义了通过 Kolmogorov-Smirnov 统计量进行特征筛选、均值方差特征筛选、类别自适应特征筛选的函数，以及为这些方法生成数据的函数。下面是运行函数的过程。

```
### 运行函数 ###
n = 50### 样本量
p = 500### 维数
R = 2###二值响应变量
d = math.floor(n/math.log(n))###前d个最重要的变量
zdata = gdat(n,p,R)#运行函数gdat
Y = zdata["Y"]
X = zdata["X"]
data_full = np.hstack((np.mat(Y).T, X))#水平方向合并矩阵Y.T与X
data_full1 = [i.tolist() for i in data_full.A if i[0] == 1]
#找到data_full中Y值为1的行
data_full2 = [i.tolist() for i in data_full.A if i[0] == 2]
#找到data_full中Y值为2的行
outdata = np.array(data_full1+data_full2)#按照行合并data_full1和data_full2
Y_full = outdata[:, 0]#第0列为Y
X_full = outdata[:, 1:(p+1)]#1到p列为X
print(KS(X_full,Y_full)[0:d])#选取KS方法中前d个变量
print(CAS(X_full,Y_full)[0:d])#选取CAS方法中前d个变量
print(MV(X_full,Y_full)[0:d])#选取MV方法中前d个变量
```

运行以上程序，我们筛选出 $\lceil n/\log(n)\rceil$ 个特征。

8.6 习题

1. 边际非参估计假设我们有一个随机样本 $\{(\boldsymbol{X}_i, Z_i, Y_i)\}_{i=1}^n$ 来自于部分线性模型

$$Y_i = \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{\beta} + m(Z_i) + \varepsilon_i,$$

其中 $\boldsymbol{X}_i$ 和 Z_i 分别是 p 维和 1 维的协变量，对响应变量分别产生线性和分线性影响。ε_i 是条件均值为 0 的随机误差。为了快速识别 $\boldsymbol{X}_i$ 中的重要变量，采用特征筛选方法。

（1）根据 10.2.1 和 10.2.3 节的内容，通过光滑样条逼近之后，考虑的 p 个边际回归模型对应的二次损失函数是什么?

（2）写出边际筛选统计量。

（3）R 与 Python 语言程序实现。

2. 基于边际相关系数的特征筛选方法大多采用了传统统计的相关系数，基于此，

（1）请写出 Spearman 相关系数，并解释其解决了何种统计模型的特征筛选问题。

（2）请列举其他可以用来进行特征筛选的相关系数。

3. 在 10.4.3 节的高维分类数据的类别自适应筛选统计量，考虑了协变量是连续的情形。而当协变量是有序分类变量时，写出相应的自适应筛选统计量形式。

4. 请使用心肌病微阵列数据集确定对小鼠 G 蛋白偶联受体（Ro1）过度表达最具影响力的基因。此数据集的样本量 $n = 30$，变量个数 $p = 6319$，属于超高维数据。基于 R 或 Python 语言，

(1)使用边际最小二乘、广义和秩相关系数以及距离相关系数筛选出前 $\lceil n/\log(n)\rceil$ 重要变量，并对比解释筛选结果。

（2）请使用一步变量选择方法 Lasso，SCAD，与 “特征筛选 + 变量选择” 两步模型拟合方法，即 SIS-Lasso，SIS-SCAD，对心肌病微阵列数据集进行线性回归分析。并使用样本内 MSE，即 $\dfrac{1}{n}\sum\limits_{i=1}^{n}(Y_i - \widehat{Y}_i)$ 对比模型预测结果。

5. 最近科研人员使用 mRNA 表达谱分析数据研究肺癌成因。本肺癌数据集包含 203 个快速冷冻肺肿瘤和正常肺的样本，mRNA 表达水平为 12600，即 $n = 203$，$p = 12600$。这里，响应变量表示肿瘤类型：肺腺癌（ADEN）139 例，肺鳞状细胞癌 21 例癌（SQUA），小细胞肺癌（SCLC）6 例，20 例肺类癌（COID），其余 17 例正常肺样本（正常）。基于 R 或 Python 语言，

（1）使用均值方差统计量与类别自适应筛选统计量筛选出前 $\lceil n/\log(n)\rceil$ 重要变量，并对比解释筛选结果。

（2）请使用“特征筛选 + 变量选择”两步模型拟合方法，即 CAS-Lasso，CAS-SCAD，对心肌病微阵列数据集进行线性 Logistic 回归分析。并使用样本内错分率，即 $\frac{1}{n}\sum_{i=1}^{n} I[Y_i \neq \widehat{Y}_i]$ 对比预测结果。

第四部分

深度学习

第九章　人工神经网络

深度学习（Deep Learning）是机器学习的一个子领域，其主要特点是使用深度神经网络来学习和表示数据，通过多层神经网络学习多层次的特征表示，可以学习到更抽象、更复杂的特征。

深度学习算法在处理复杂任务和深层次网络时，通常需要大量的标记数据，并且对计算资源的需求较高。由于它在大规模数据和复杂任务上出色的表现，深度学习在计算机视觉、自然语言处理、语音识别、医学图像分析等领域有广泛应用。深度学习只是机器学习更广泛领域中的一种方法，它可以与其他机器学习技术结合使用，以解决复杂问题。

本章及下一章将介绍两种主要的深度学习方法：人工神经网络（Artificial Neural Networks）和特征筛选（Deep Learning）。

9.1　简介

1943 年，心理学家 W · McCulloch 和数理逻辑学家 W · Pitts 在分析、总结神经元（Neuron）基本特性的基础上，首先提出了神经元的数学模型 [104]。到目前为止，人工神经网络（后面简称为神经网络）已经发展成了一个相当大的、多学科交叉的学科领域。对神经网络的定义已有很多，本书采用了 Kohonen 在 1988 年提出的定义，即“神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应”[105]。在机器学习中谈论神经网络时指的是“神经网络学习”，或者说，是机器学习与神经网络这两个学科领域的交叉部分。

9.2　人工神经元

神经网络起源于对生物神经元的研究，如图9.1所示生物神经元包括细胞体、树突、轴突等部分，其中树突是用于接受输入信息，输入信息经过突触处理，当达到一定条件时通过轴突传出，此时神经元处于激活状态；反之没有达到相应条件，则神经元处于抑制状态。

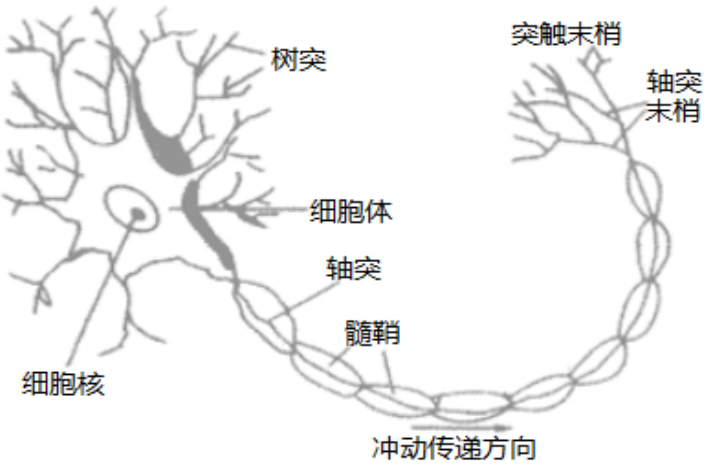


图 9.1 生物神经元

神经网络的基本节点是人工神经元（后面简称为神经元），其工作原理是仿照生物神经元提出的，如图9.2所示。输入值经过加权和偏置后，由激活函数处理后输出。

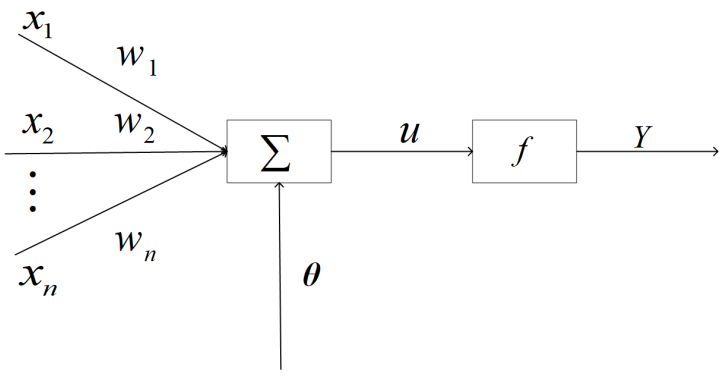


图 9.2 人工神经元模型

从图9.2所示的神经元示意图来看，对某一个神经元来说，它可以同时接受 n 个输入信号，分别用 $x_1, x_2, x_3, \cdots, x_n$ 来表示。每个输入端与神经元之间有联接权值 $w_1, w_2, w_3, \cdots, w_n$，神经元的总输入为对每个输入进行加权求和，同时加上偏置 θ，即：

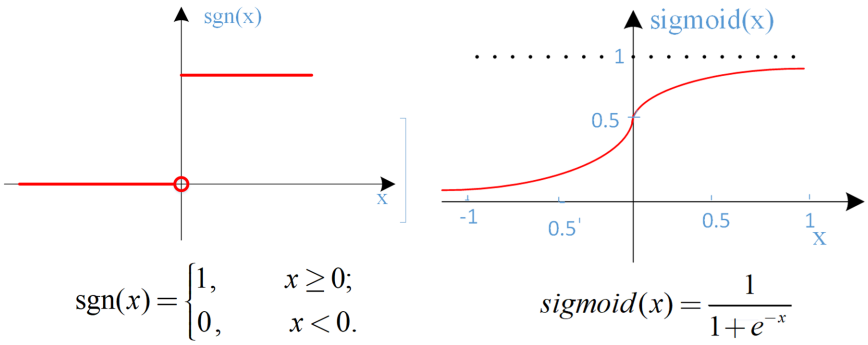
$$u = \sum_{i=1}^{n} w_i x_i + \theta \tag{9.2.1}$$

神经元的输出 Y 是对 u 的映射，

$$Y = f(u) = f\left(\sum_{i=1}^{n} w_i x_i + \theta\right) \tag{9.2.2}$$

其中，f 为激活函数。

激活函数能够给神经元引入非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到更多的非线性模型中。而理想的激活函数应该是像图9.3(a) 中 sgn(x) 函数那样的跃阶函数，可以将输入神经元的值映射为“抑制”或“兴奋”的输出，也就是 0 和 1 [106]。然而跃阶函数的性质比较差，它是不连续、不光滑的，因此实际常用 sigmoid 函数作为激活函数，如图9.3(b)，它把输入神经元的值映射到了 (0,1) 的范围内进行输出。



$$\mathrm{sgn}(x)=\begin{cases}1, & x\geq 0;\\ 0, & x<0.\end{cases}$$

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

(a)跃阶函数　　(b)sigmod函数

图 9.3　神经元激活函数

9.3　前馈神经网络

在实际问题中，不可能只使用一个神经元，会使用非常多的神经元进行计算。一般情况下，神经元分层排列，每个神经元只接受前一层的输入，并输出到下一层，网络没有下一层到上一层的回路信号，这种网络称为前馈神经网络（Feedforward Neural Networks）。前馈并不意味着网络中信号不能向后传，而是指网络拓扑结构上不存在环或回路。前馈神经网络的第一层为输入层（Input Layer），最后一级为输出层（Output Layer），输入层与输出层之间的各层统一称为隐含层（Hidden Layers）。隐含层和输出层神经元都是拥有激活函数的功能神经元。一个网络可以只包含一个隐含层，也可以包含多个隐含层。在这个意义下感知器可以理解为一种常见的前馈神经网络。

9.3.1 单层感知器

感知器，也称感知机，是 Frank Rosenblatt 提出的一种神经网络 [107]。如图9.4所示，这是一个简单的感知器逻辑图，该感知器具有两层，第一层为输入层，将输入的值传递给下一层；第二层为计算单元，并将结果输出。因此，该网络也被称为单层感知器，它能容易实现与、或、非等逻辑运算，也常用于线性分类问题。

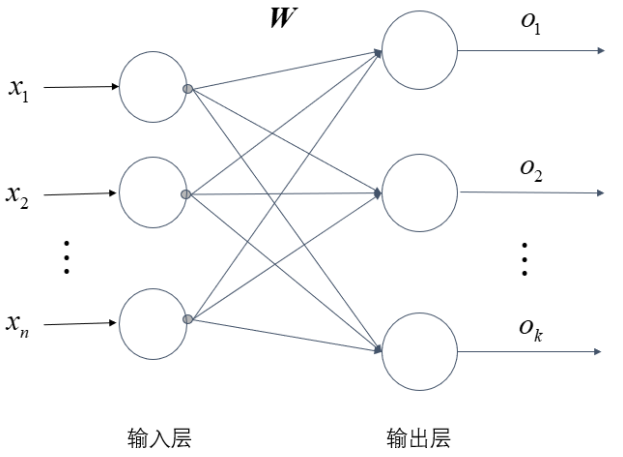


图 9.4 单层感知器模型

假设输入模式为 n 维特征向量 $\boldsymbol{X} = (x_1, \cdots, x_n)^{\mathrm{T}}$，则感知器的输入层应有 n 个神经元。若输出类别有 k 个，则输出层应包含 k 个神经元，即 $O = (o_1, o_2, \cdots, o_k)^{\mathrm{T}}$。输入层的第 j 个神经元与输出层的第 i 个神经元的连接权值为 w_{ji}，则第 i 个神经元的输出为：

$$o_i = f\left(\sum_{j=1}^{n}(w_{ji}x_j + \theta_i)\right) \tag{9.3.1}$$

如果感知器模型是一种针对二分类问题中的神经网络模型，其激活函数为符号函数 $\mathrm{sgn}(x)$，k 值为 2，感知机模型通过输入层接受输入信息 $\boldsymbol{X} = (x_1, \cdots, x_n)^{\mathrm{T}}$，其输出可以表示为：

$$o_i = sgn\left(\sum_{j=1}^{n}(w_{ji}x_j + \theta_i)\right) \tag{9.3.2}$$

一般地，给定训练数据集和感知器后，权重 $\boldsymbol{W}$ 以及阈值 θ 可通过学习得到。当把阈值 θ 可看作一个固定输入为 1.0 的“哑结点”（Dummy Node）并将所对应的连接权也记为 $\boldsymbol{W}$ 时，权重和阈值的学习就可统一为权重的学习。

9.3.2 多层感知器

单层感知器只能解决线性可分的问题，多层感知器可以突破这一局限，实现输入和输出之间的非线性映射。多层感知器的神经元层级之间采用全连接的方式，上层的神经元的输出作为输入推送给下一层的所有神经元。多层感知器中，除了输入输出层，中间可以有多个隐层。最简单的情况时只含一个输入层、一个隐含层和一个输出层，即三层感知器，也叫单隐层前馈神经网络，如图9.5所示。

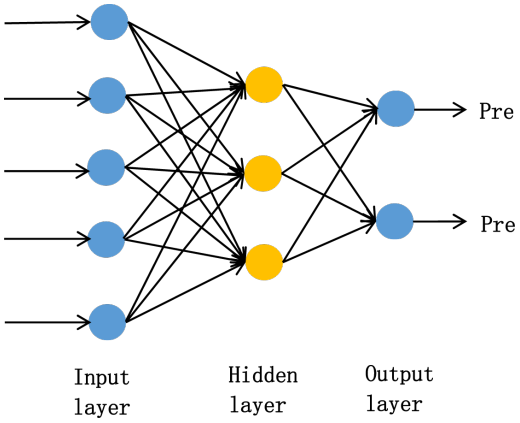


图 9.5 多层感知器示意图

若第 l 层的第 i 个神经元的输出为 $a_i^{(l)}$，其与上一层第 j 个神经元的连接权值为 $w_{ji}^{(l-1)}$，偏置为 $\theta_i^{(l)}$；与下一层的第 k 个神经元的连接权值为 $w_{ik}^{(l)}$，偏置为 $\theta_k^{(l+1)}$。则该神经元的输入为：$z_i^{(l)} = \sum_i w_{ji}^{(l-1)} a_j^{(l-1)} + \theta_j^{(l)}$，其中 $a_j^{(l-1)}$ 为上一层第 j 个神经元的输出，则第 l 层第 i 个神经元的输出为：

$$a_i^{(l)} = f\left(z_i^{(l)}\right) = f\left(\sum_j w_{ji}^{(l-1)} a_j^{(l-1)} + \theta_i^{(l)}\right) \tag{9.3.3}$$

其中，f 为激活函数。

从上面可以看出，人工神经网络的学习过程，就是根据训练数据来调整神经元之间的连接权值（Connection Weight）和每个功能神经元的偏置；换言之，神经网络学到的东西，蕴含在连接权值与偏置中。

9.4 神经网络的正向与反向传播算法

在前几节中，我们已经学习了神经网络的基本知识。由于神经网络的结构是多种多样的，在本节中将以单隐层神经网络和双隐层神经网络为例，介绍神经网络具

体的学习过程是怎样的。

9.4.1 神经网络的正向传播

先以简单的单隐层网络为例（图9.6 ），其中第一层是输入层单元，输入原始数据。第二层是隐藏层单元，对数据进行处理，然后呈递到下一层。最后是输出单元，计算后，输出计算结果。

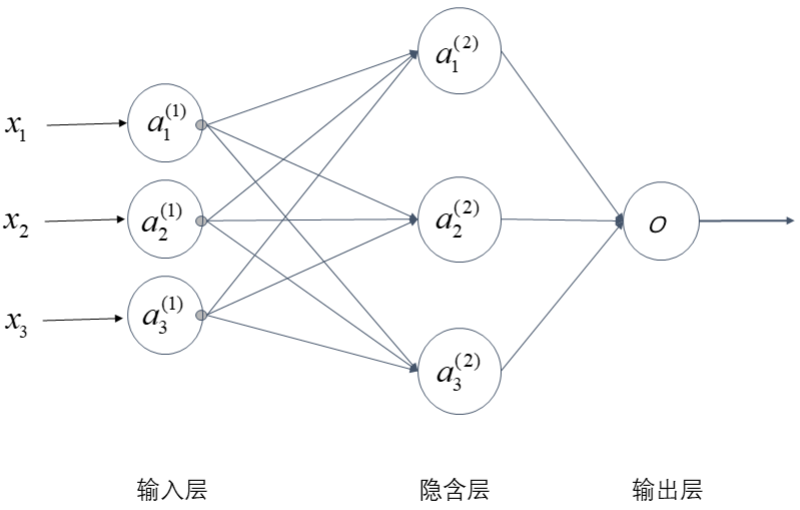


图 9.6 简单的单隐层神经网络

一般情况下，为了提高拟合能力，需要增加偏置项，如图9.7为一个 3 层的神经网络，其中第一层为输入层，最后一层为输出层，中间一层为带偏置的隐藏层。

下面引入一些标记法来帮助描述模型：

$a_i^{(j)}$ 代表第 j 层的第 i 个单元，$w^{(j)}$ 代表从第 j 层映射到第 j+1 层时的权重的矩阵，例如 $w^{(1)}$ 代表从第一层映射到第二层的权重的矩阵。其尺寸为：以第 j+1 层的单元数量为行数，以第 j 层的激活单元数加一为列数的矩阵。例如：图9.7所示的神经网络中如 $w^{(1)}$ 的尺寸为 $3\times 4=12$ 。

对于图9.7所示的模型，激活单元和输出分别表达为：

$$a_1^{(2)}=f\left(w_{10}^{(1)}x_0+w_{11}^{(1)}x_1+w_{12}^{(1)}x_2+w_{13}^{(1)}x_3\right) \tag{9.4.1}$$

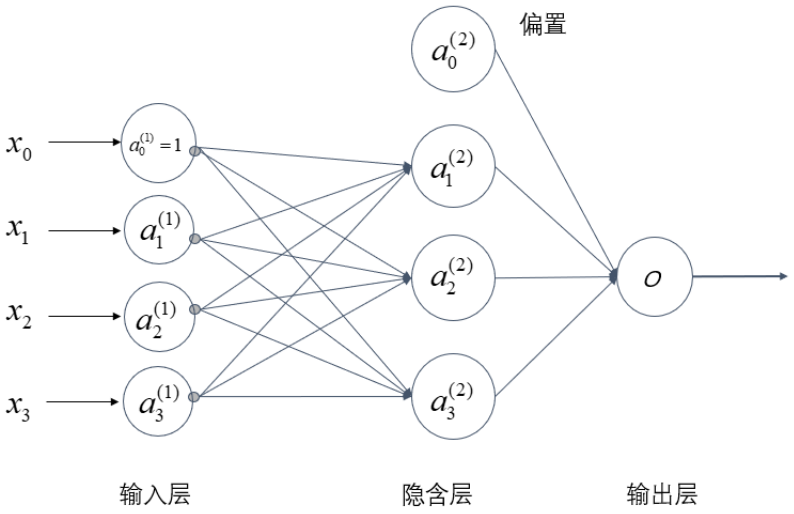


图 9.7 加入偏置的神经网络模型

$$a_2^{(2)} = f\left(w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3\right) \tag{9.4.2}$$

$$a_3^{(2)} = f\left(w_{30}^{(1)}x_0 + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3\right) \tag{9.4.3}$$

$$O = f\left(w_{10}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)}\right) \tag{9.4.4}$$

上述情况是一种两分类的神经网络模型，但当有更多种分类时，比如以下这种情况，该怎么办?

假如要训练一个神经网络算法来识别路人、汽车、摩托车和卡车，在输出层我们应该有 4 个值。例如，第一个值为 1 或 0 用于预测是否是行人，第二个值用于判断是否为汽车，以此类推。输入向量 x 有三个维度，两个中间层，输出层 4 个神经元分别用来表示 4 类，也就是每一个数据在输出层都会出现 $(a,b,c,d)^T$，且 $(a,b,c,d)^T$ 中仅有一个为 1，其余为 0 时，表示当前类。图9.8 是该神经网络的结构示例。

神经网络算法的输出结果为以下四种可能情形之一:

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

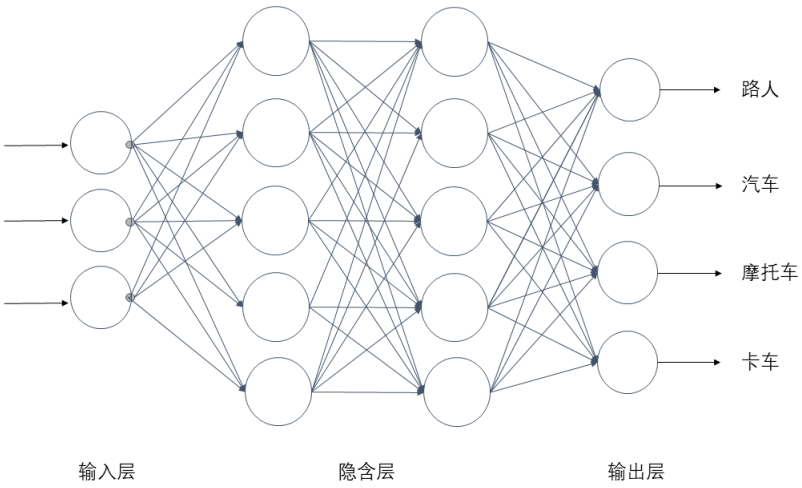


图 9.8 神经网络结构示意图

9.4.2 神经网络的损失函数

首先引入一些便于稍后讨论的符号：

假设神经网络的训练样本有 m 个，第 i 个样本的输入为 $\boldsymbol{X}_i = (x_{i0}, x_{i1}, \cdots, x_{in})^{\mathrm{T}}$，第 i 个样本的标签为 $\boldsymbol{Y}_i = (Y_{i1}, \cdots, Y_{ik})^{\mathrm{T}}$，其整体映射函数为 $G(\boldsymbol{X})$，若权重 W 为变量时，则整体映射函数记为 $G_W(\boldsymbol{X})$。L 表示神经网络层数，S_l 表示每层的神经元个数 $(l = 1, \cdots, L)$，则 S_L 代表最后一层中神经元的个数。若该神经网络是 k 分类问题，那么显然 $S_L = k$。

神经网络学习的过程，实际上就是最小化损失函数的过程。而损失函数是用来估计模型的预测值 $G(\boldsymbol{X})$ 与真实值 $\boldsymbol{Y}$ 的不一致程度，通常用误差函数进行衡量。对于连续函数，当 $G(\boldsymbol{X})$ 和 $\boldsymbol{Y}$ 为维度为一时，其损失函数常定义为：

$$J(W) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(G_W(\boldsymbol{X}_i) - \boldsymbol{Y}_i\right)^2\right] + \lambda\|W\|_2^2 \tag{9.4.5}$$

公式9.4.5右侧的前半部分为经验风险函数，后半部分为正则化项，λ 为正则化因子，由于是 2-范数的平方项，称为L_2 正则化项，也可以是其它形式的正则化项。

对于离散函数，比如，逻辑回归中常用的整体损失函数为：

$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m}\boldsymbol{Y}_i \log G_W(\boldsymbol{X}_i) + (1-\boldsymbol{Y}_i)\log\left(1 - G_W(\boldsymbol{X}_i)\right)\right] + \lambda\|W\|_2^2 \tag{9.4.6}$$

在逻辑回归中，只有一个输出变量，但是在神经网络中，可以有很多输出变量。如果 $G_W(\boldsymbol{X})$ 是一个维度为 k 的向量，$G_W(\boldsymbol{X})_j$ 表示 $G_W(\boldsymbol{X})$ 的第 j 个维度，并

且训练集中的因变量也是同样维度的一个向量，因此损失函数会比逻辑回归更加复杂一些，则损失函数为：

$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} Y_{ij}\log\left(G_W\left(\boldsymbol{X}_i\right)\right)_j + \left(1-Y_{ij}\right)\log\left(1-\left(G_W\left(\boldsymbol{X}_i\right)\right)_j\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}\left(w_{ij}^{(l)}\right)^2 \tag{9.4.7}$$

这个看起来复杂很多的损失函数，其背后的思想还是一样的，即希望通过损失函数来观察算法预测的结果与真实情况的误差有多大。唯一不同的是，对于每一行特征，都会给出 k 个预测，基本上可以利用循环，对每一行特征都预测 k 个不同结果，然后再利用循环在 k 个预测中选择可能性最高的一个，将其与 $\boldsymbol{Y}$ 中的实际数据进行比较。

在确定了模型与损失函数后，通过梯度下降算法就可以让神经网络进行学习了，但是，使用梯度下降算法有一个前提：该算法需要提前知道当前点的梯度，然而事实并非如此，如果仔细观察过此模型的损失函数之后，我们会发现这个函数相当的复杂，其导函数难以计算，而在更加复杂的神经网络模型中，对损失函数求导往往是不可能的，因此我们需要通过其他方法来进行实现。

9.4.3 反向传播算法

1974 年，Paul Werbos [108] 首次给出了训练神经网络的学习算法——反向传播算法（BP，Back Propagation），这个算法可以高效的计算每一次迭代过程中的梯度，它是迄今最成功的神经网络学习算法。在实际使用神经网络时，大多是在使用 BP 算法进行训练。值得指出的是，BP 算法不仅可用于多层前馈神经网络，还可用于其他类型的神经网络，例如训练递归神经网络 [109]。但通常说 “BP 网络” 时，一般是指用 BP 算法训练的多层前馈神经网络。

下面举一个简单的例子来说明 BP 算法是如何运作的：

假设训练集只有一个样本 $(\boldsymbol{X},\boldsymbol{Y})$，使用四层神经网络进行训练，其中 $k=4$，$S_L=4$（如图9.8 ），激活函数为 sigmoid 函数，损失函数为式9.4.5，该网络前向传播算法为：

一般从最后一层的误差开始计算，定义 $\delta^{(i)}=\dfrac{\partial J(W)}{\partial z^i}$ 来表示误差，则 $\delta^{(4)}=\dfrac{\partial J(W)}{\partial z^4}=(a^{(4)}-\boldsymbol{Y})*f'\left(z^{(4)}\right)$，在利用这个误差来计算前一层的误差 $\delta^{(3)}=\left(W^{(3)}\right)^{\mathrm{T}}\delta^{(4)}*f'\left(z^{(3)}\right)$，其中 * 表示矩阵对应元素相乘，$f'\left(z^{(3)}\right)$ 是激活函数的导数，$f'\left(z^{(3)}\right)=$

Algorithm 1

$$a^{(1)} = \boldsymbol{X} \tag{9.4.8}$$

$$z^{(2)} = W^{(1)} a^{(1)} \tag{9.4.9}$$

$$a^{(2)} = f\left(z^{(2)}\right)\left(add\ a_0^{(2)}\right) \tag{9.4.10}$$

$$z^{(3)} = W^{(2)} a^{(2)} \tag{9.4.11}$$

$$a^{(3)} = f\left(z^{(3)}\right)\left(add\ a_0^{(3)}\right) \tag{9.4.12}$$

$$z^{(4)} = W^{(3)} a^{(3)} \tag{9.4.13}$$

$$a^{(4)} = f\left(z^{(4)}\right) \tag{9.4.14}$$

$a^{(3)} * \left(1 - a^{(3)}\right)$。而 $\left(W^{(3)}\right)^{\mathrm{T}} \delta^{(4)}$ 则是权重导致的误差和。下一步是计算第二层的误差 $\delta^{(2)} = \left(W^{(2)}\right)^{\mathrm{T}} \delta^{(3)} * f'\left(z^{(2)}\right)$。第一层是输入变量，不存在误差。

有了所有误差的表达式后，便可以计算损失函数的偏导数了，假设 $\lambda = 0$，即不做任何正则化处理时有：

$$\frac{\partial J(W)}{\partial W_{ji}^{(l)}} = \frac{\partial J(W)}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial W_{ji}^{(l)}} = a_i^{(l)} \delta_i^{(l+1)}$$

其中，$a^l = (a_1^l, a_2^l, \cdots, a_k^l)$ 且 $\delta^l = (\delta_1^l, \delta_2^l, \cdots, \delta_k^l)$

BP 算法的学习过程实际上由正向传播过程和反向传播过程组成。在正向传播过程中，输入信息通过输入层经隐含层，逐层处理并传向输出层。如果在输出层得不到期望的输出值，则取输出与期望的误差的平方和作为目标函数，转入反向传播，逐层求出目标函数对各神经元权值的偏导数，构成目标函数对权值向量的梯量，作为修改权值的依据，网络的学习在权值修改过程中完成。误差达到所期望值时，网络学习结束。

9.4.4 全局最小与局部极小

若用 J 表示神经网络在训练集上的误差，则它显然是关于连接权值和偏置的函数。此时，神经网络的训练过程可看作一个参数寻优过程，即在参数空间中，寻找一

组最优参数使得 J 最小。

我们常会谈到两种“最优”:“局部极小”(Local Minimum)和“全局最小”(Global Minimum)。对 $\omega*$ 和 $\theta*$，若存在 $\epsilon > 0$ 使得

$$\forall (\omega;\theta) \in \{(\omega;\theta) \,|\, \|(\omega;\theta) - (\omega*;\theta*)\| \leqslant \epsilon\}$$

都有 $J(\omega;\theta) \geqslant J(\omega*;\theta*)$ 成立，则 $(\omega*;\theta*)$ 为局部极小解；若对参数空间中的任意 $(\omega;\theta)$ 都有 $J(\omega;\theta) \geqslant J(\omega*;\theta*)$，则 $(\omega*;\theta*)$ 为全局最小解。直观地看，局部极小解是参数空间中的某个点，其邻域点的误差函数值均不小于该点的函数值；全局最小解则是指参数空间中所有点的误差函数值均不小于该点的误差函数值。两者对应的分别称为误差函数的局部极小值和全局最小值。

显然，参数空间内梯度为零的点，只要其误差函数值小于邻点的误差函数值，就是局部极小点；可能存在多个局部极小值，但却只会有一个全局最小值。也就是说，“全局最小”一定是“局部极小”，反之则不成立。例如，图9.9中有两个局部极小，但只有其中之一是全局最小。显然，在参数寻优过程中是希望找到全局最小。

基于梯度的搜索是使用最为广泛的参数寻优方法。在此类方法中，我们从某些初始解出发，迭代寻找最优参数值. 每次迭代中，先计算误差函数在当前点的梯度，然后根据梯度确定搜索方向。例如，由于负梯度方向是函数值下降最快的方向，因此梯度下降法就是沿着负梯度方向搜索最优解。若误差函数在当前点的梯度为零，则已达到局部极小，更新量将为零，这意味着参数的迭代更新将在此停止。显然，如果误差函数仅有一个局部极小，那么此时找到的局部极小就是全局最小；然而，如果误差函数具有多个局部极小，则不能保证找到的解是全局最小。对后一种情形，称参数寻优陷入了局部极小，这显然不是我们所希望的。

在现实任务中，常采用以下策略来试图“跳出”局部极小，从而进一步接近全局最小:

1、以多组不同参数值初始化多个神经网络，按标准方法训练后，取其中误差最小的解作为最终参数。这相当于从多个不同的初始点开始搜索，这样就可能陷入不同的局部极小，从中进行选择有可能获得更接近全局最小的结果。

2、使用“模拟退火”（SA，Simulated Annealing）技术 [110]。模拟退火在每一步都以一定的概率接受比当前解更差的结果，从而有助于“跳出”局部极小。在每步迭代过程中，接受“次优解”的概率要随着时间的推移而逐渐降低，从而保证算法稳定。

3、使用随机梯度下降。与标准梯度下降法精确计算梯度不同，随机梯度下降法在计算梯度时加入了随机因素。于是，即便陷入局部极小点，它计算出的梯度仍可能不为零，这样就有机会跳出局部极小继续搜索。

此外，遗传算法（GA，Genetic Algorithms）[111] 也常用来训练神经网络以更好地逼近全局最小。需注意的是，上述用于跳出局部极小的技术大多是启发式智能算法，理论上尚缺乏保障。

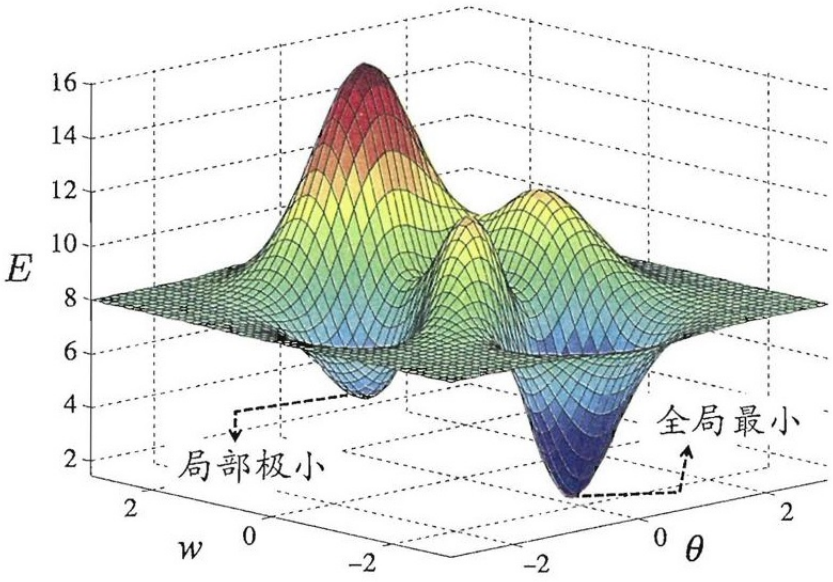


图 9.9 全局最小与局部极小

9.5 径向基网络

径向基函数神经网络（简称为径向基（RBF，Radial Basis Function）网络）是一种局部逼近型网络模型。所谓局部逼近型网络，是指网络输出仅与少数几个连接权重相关，对于每个参与模型训练的样本，通常仅有少数与其相关的权重需要进行更新。这种局部性的参数更新方式有利于加速模型的训练。

机器学习中回归任务的本质是根据已知离散数据集求解与之相符的连续函数，基本求解思路是对已知的离散数据进行拟合，使得拟合函数与已知离散数据的误差在某种度量意义下达到最小。RBF 网络对于此类问题的求解思路则是通过对已知离散数据进行插值的方式确定网络模型参数。

RBF 神经网络一共分为三层，如图9.10所示。第一层为输入层，由信号源节点组成；第二层为隐藏层，隐藏层中神经元的激活函数，即径向基函数是对中心点径向对称且衰减的非负线性函数，该函数是局部响应函数。局部响应函数一般要根据具体问题设置相应的隐藏层神经元个数；第三层为输出层，是对输入模式做出的响应，输出层根据线性权重进行调整，采用的是线性优化策略，因而学习速度相对较快。该网络将径向基函数作为隐单元的“基”构成隐含层空间，将输入直接映射到隐空间。

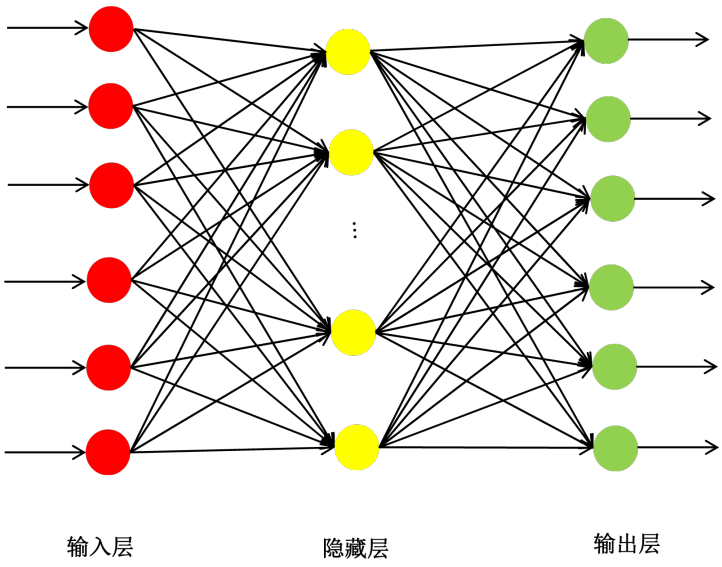


图 9.10 RBF 神经网络模型

RBF 网络的基本思想是：用 RBF 作为隐单元的“基”构成隐含层空间，这样就可以将输入直接映射到隐空间，而不需要通过“权”连接。当 RBF 的中心点确定以后，这种映射关系也就确定了。而隐含层空间到输出空间的映射是线性的，即网络的输出是隐单元输出的线性加权和，此处的权即为网络可调参数。隐含层的作用是把向量从低维度映射到高维度，这样低维度线性不可分的情况到高维度就可以变得线性可分了，这即是核函数的思想。这样网络由输入到输出的映射是非线性的，而网络输出对可调参数而言却又是线性的。网络的权就可由线性方程组直接解出，从而大大加快学习速度并避免局部极小问题。

对于给定训练样本集 $D=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\ldots,(\boldsymbol{X}_m,Y_m)\}$，其中

$$\boldsymbol{X}_i=(X_{i1},\cdots,X_{ip})^{\mathrm{T}}$$

为输入向量，Y_i 为 $\boldsymbol{X}_i$ 所对应的数值型真实值，插值的目标是找到某个函数 f 使得：

$$f(\boldsymbol{X}_i)=Y_i \quad i=1,\ldots,m \tag{9.5.1}$$

通常 f 为非线性函数，故其函数图像为一个插值曲面，该曲面经过数据集 D 中所有样本点。为求解函数 f，可考虑选择 m 个与样本点对应的基函数 Φ_1，Φ_2,$\cdots$,Φ_m，并将 f 进行近似表示为这组基函数的线性组合，即有：

$$f(\boldsymbol{X})=\sum_{i=1}^{m}w_i\Phi_i \tag{9.5.2}$$

只需确定上式中权重参数 w_1,w_2,$\cdots$,w_m 取值，就可确定插值函数 f 的具体形

式，进而将上述插值问题转换为如下非线性模型的参数问题：

$$\Phi_i = \Phi\left(\|\boldsymbol{X} - \boldsymbol{X}_i\|\right) \tag{9.5.3}$$

其中 Φ_i 为非线性函数，其函数自变量为输入数据 $\boldsymbol{X}$ 到中心点 $\boldsymbol{X}_i$ 的距离。

由于从中心点 $\boldsymbol{X}_i$ 到相同半径的球面上任意点的距离相等，即距离具有径向相同性，故通常将这种以距离作为自变量的基函数，称为径向基函数。将径向基函数的具体形式带入式中，则可得到插值函数 f 如下形式：

$$f(\boldsymbol{X}) = \sum_{i=1}^{m} w_i \Phi\left(\|\boldsymbol{X} - \boldsymbol{X}_i\|\right) \tag{9.5.4}$$

将数据集 D 中任意样本 $(\boldsymbol{X}_j, Y_j)$ 带入可得：

$$Y_j = \sum_{i=1}^{m} w_i \Phi\left(\|\boldsymbol{X}_j - \boldsymbol{X}_i\|\right) \tag{9.5.5}$$

将数据集 D 中数据均代入式9.5.5可得到由 m 个线性方程组成的线性方程组。记径向基函数的取值 $\Phi\left(\|\boldsymbol{X}_j - \boldsymbol{X}_i\|\right)$ 为 Φ_{ij}，则可将该线性方程组表示为如下矩阵形式：

$$\boldsymbol{\Phi w} = \boldsymbol{Y} \tag{9.5.6}$$

其中 $\boldsymbol{\Phi} = (\Phi_{ij})_{\boldsymbol{n}\times\boldsymbol{n}}$，$\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^{\mathrm{T}}$，$\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_n)^{\mathrm{T}}$。

径向基函数可以有多种选择，如高斯径向基函数、反演 S 型径向基函数等。

（1）高斯径向基函数：

$$G\left(\boldsymbol{X}, \boldsymbol{X}_i\right) = \exp\left(-\frac{1}{2\delta_i^2}||\boldsymbol{X} - \boldsymbol{X}_i\|^2\right) \tag{9.5.7}$$

（2）反演 S 型径向基函数：

$$R\left(\boldsymbol{X}, \boldsymbol{X}_i\right) = \frac{1}{1 + \exp\left(\frac{||\boldsymbol{X} - \boldsymbol{X}_i||^2}{\delta_i^2}\right)} \tag{9.5.8}$$

其中 δ_i 为扩展常数，其取值越大，数据分布范围越宽。

在实际应用中，高斯径向基函数表示为：

$$G\left(\boldsymbol{X}, c_i\right) = \exp\left(-\beta_i \left\|\boldsymbol{X} - c_i\right\|^2\right)$$

其中 c_i 称为中心。通常采用两步过程来训练该网络：第一步，确定神经元中心 c_i，常用的方式包括随机采样、聚类等；第二步，利用 BP 算法等来确定参数 w_i 和 β_i。

9.6 其它常见的神经网络

神经网络模型中包含了大量的不同模型，就算相同模型也可以通过改变网络的结构变得不同，因此本书不能详尽的列举所有的模型，所以选择了最常见的几种模型进行了简单的介绍。

ART 网络模型

竞争型学习（Competitive Learning）是神经网络中一种常用的无监督学习策略，在使用该策略时，网络的输出神经元相互竞争，每一时刻仅有一个竞争获胜的神经元被激活，其他神经元的状态被抑制。这种机制亦称“胜者通吃”（Winner-take-all）原则。

自适应谐振理论（ART，Adaptive Resonance Theory）网络 [112] 是竞争型学习的重要代表。该网络由比较层、识别层、识别阈值和重置模块构成。其中，比较层负责接收输入样本，并将其传递给识别层神经元。识别层每个神经元对应一个模式类，神经元数目可在训练过程中动态增长以增加新的模式类。

在接收到比较层的输入信号后，识别神经元之间相互竞争以产生获胜神经元。竞争的最简单方式是，计算输入向量与每个识别层神经元所对应的模式类的代表向量之间的距离，距离最小者胜。获胜神经元将向其他识别层神经元发送信号，抑制其激活。若输入向量与获胜神经元所对应的代表向量之间的相似度大于识别阈值，则当前输入样本将被归为该代表向量所属类别，同时，网络连接权将会更新，使得以后在接收到相似输入样本时该模式类会计算出更大的相似度，从而使该获胜神经元有更大可能获胜；若相似度不大于识别阈值，则重置模块将在识别层增设一个新的神经元，其代表向量就设置为当前输入向量。

ART 比较好地缓解了竞争型学习中的“可塑性-稳定性窘境”（Stability-plasticity Dilemma），可塑性是指神经网络要有学习新知识的能力，而稳定性则是指神经网络在学习新知识时要保持对旧知识的记忆。这就使得 ART 网络具有一个很重要的优点：可进行增量学习（Incremental Learning）或在线学习（Onlinelearning）。早期的 ART 网络只能处理布尔型输入数据，此后 ART 发展成了一个算法族，包括能处理包括实值输入的 ART2 网络、结合模糊处理的 FuzzyART 网络，以及可进行监督学习的 ARTMAP 网络等。

自组织映射网络模型

自组织映射（SOM，Self-Organizing Map）网络 [113] 是一种竞争学习型的无监督神经网络，它能将高维输入数据映射到低维空间（通常为二维），同时保持输入数据在高维空间的拓扑结构，即将高维空间中相似的样本点映射到网络输出层中的邻近神经元。

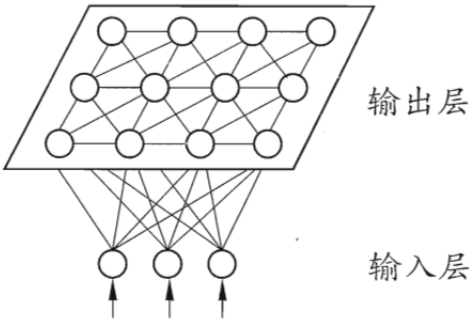


图 9.11 SOM 网络结构

如图9.11所示，SOM 网络中的输出层神经元以矩阵方式排列在二维空间中，每个神经元都拥有一个权向量，网络在接收输入向量后，将会确定输出层获胜神经元，它决定了该输入向量在低维空间中的位置。SOM 的训练目标就是为每个输出层神经元找到合适的权向量，以达到保持拓扑结构的目的。

SOM 的训练过程很简单：在接收到一个训练样本后，每个输出层神经元会计算该样本与自身携带的权向量之间的距离，距离最近的神经元成为竞争获胜者，称为最佳匹配单元（Best Matching Unit）。然后，最佳匹配单元及其邻近神经元的权向量将被调整，以使得这些权向量与当前输入样本的距离缩小。这个过程不断迭代，直至收敛。

级联相关网络

一般的神经网络模型通常假定网络结构是事先固定的，训练的目的是利用训练样本来确定合适的连接权、阈值等参数。与此不同，结构自适应网络则将网络结构也当作学习的目标之一，并希望能在训练过程中找到最符合数据特点的网络结构。级联相关（Cascade-Correlation）网络 [114] 是结构自适应网络的重要代表。

级联相关网络有两个主要成分：“级联”和“相关”。级联是指建立层次连接的层级结构。在开始训练时，网络只有输入层和输出层，处于最小拓扑结构；随着训练的进行，如图9.12所示，新的隐层神经元逐渐加入，从而创建起层级结构。当新的隐层神经元加入时，其输入端连接权值是冻结固定的。相关是指通过最大化新神经元的输出与网络误差之间的相关性（Correlation）来训练相关的参数。

与一般的前馈神经网络相比，级联相关网络无需设置网络层数、隐层神经元数目，且训练速度较快，但其在数据较小时易陷入过拟合。

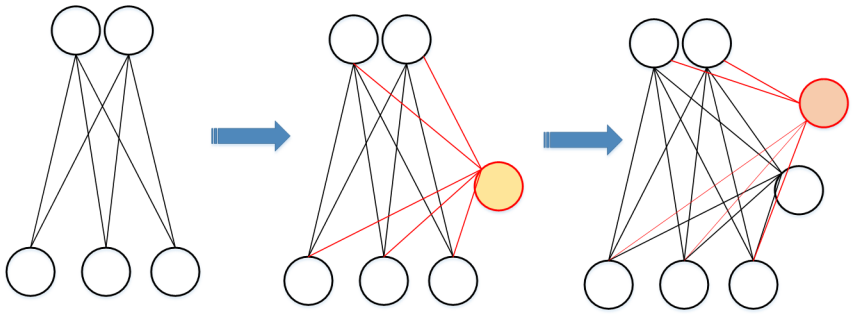


图 9.12 级联相关网络的训练过程

Elman 网络

与前馈神经网络不同，“递归神经网络”（Recurrent Neural Networks）允许网络中出现环形结构，从而可让一些神经元的输出反馈回来作为输入信号。这样的结构与信息反馈过程，使得网络在 t 时刻的输出状态不仅与 t 时刻的输入有关，还与 t-1 时刻的网络状态有关，从而能处理与时间有关的动态变化。

Elman 网络 [115] 是最常用的递归神经网络之一，其结构如图9.12所示，它的结构与多层前馈网络很相似，但隐层神经元的输出被反馈回来，与下一时刻输入层神经元提供的信号一起，作为隐层神经元在下一时刻的输入。隐层神经元通常采用 sigmoid 激活函数，而网络的训练则常通过推广的 BP 算法进行 [109]。

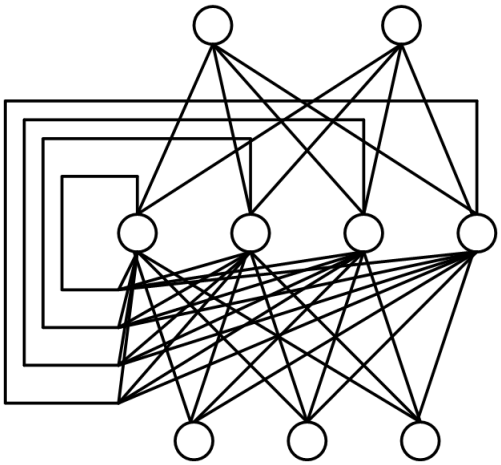


图 9.13 Elman 网络结构

9.7 神经网络实践

9.7.1 R 语言实践

本小节采用 Iris 数据集。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。为了更好显示计算结果，设置 2 个隐层，分别是 5 个单元和 4 个单元。基于 R 语言实现 BP 神经网络构建，具体实现代码如下：

```
library(neuralnet)
data("iris")
index <- sample(x = 2,size = nrow(iris),replace=TRUE,prob = c(0.7,0.3))
#按比例抽样形成训练集和测试集
trainset<-iris[index==1,]
testset<-iris[index==2,]
#为训练集新增versicolor, virginica, setosa数据列(类型编码100,010,001)
trainset$setosa<-trainset$Species=="setosa"
trainset$virginica<-trainset$Species=="virginica"
trainset$versicolor<-trainset$Species=="versicolor"
#构建网络结构
net_model=neuralnet(setosa+versicolor+virginica~
Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,#输入输出关系公式
data = trainset,#训练集
hidden=c(5,4))#隐藏层神经元数量
print(net_model)#输出模型
plot(net_model)
test_output<-compute(net_model,covariate=testset)#预测模型
result_output=as.data.frame(test_output$net.result)
cid=apply(result_output,1,which.max)#得到每行中最大值所在列
results=c("setosa","versicolor","virginica")[cid]#使用品种名替换最大值
table(results,testset$Species)#使用混淆矩阵判断结果
#输出结果为:
results      setosa versicolor virginica
setosa         17          0         0
versicolor      0         14         0
virginica       0          1        13
```

图9.14给出了神经网络的结构，包括权重。另外，从混淆矩阵结果上看，只有一个分类错误，正确率还是很高的。

9.7.2 Python 语言实践

多层感知器网络构建

本小节同样采用 Iris 数据集预测鸢尾花卉种类，首先进行数据导入和预处理。

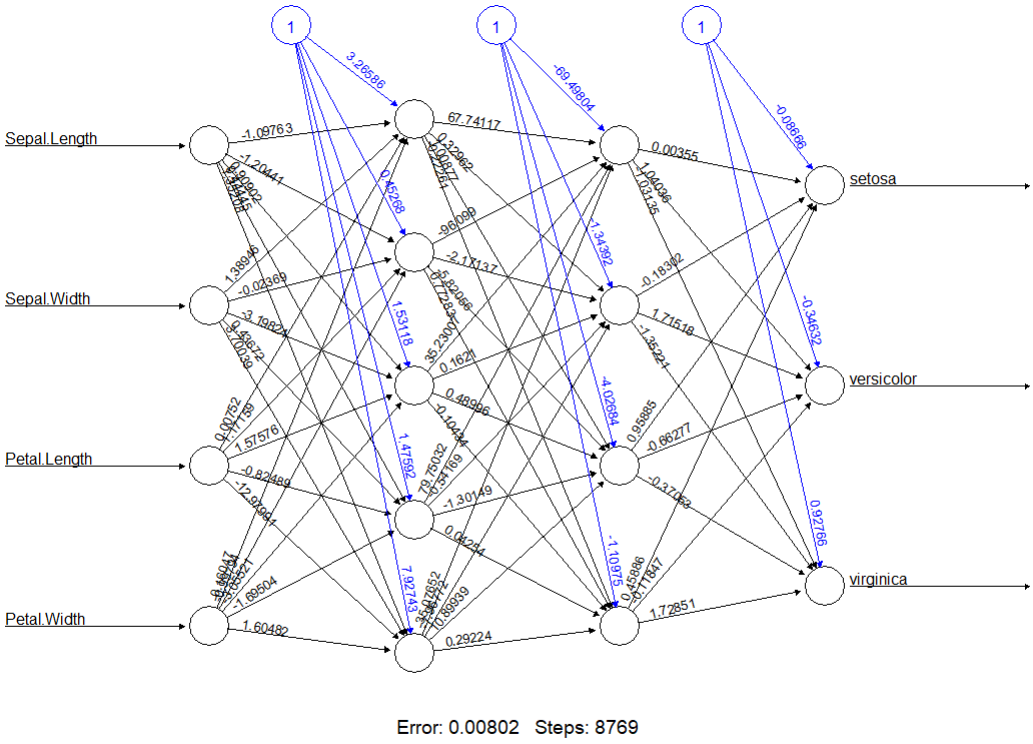


图 9.14 神经网络结构

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
df_Iris = pd.read_csv('data\iris.csv')
X = df_Iris[['sepal_length','sepal_width','petal_length','petal_width']];
y = df_Iris['species'];
#将数据按照8:2的比例随机分为训练集、测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

再建立多层感知器模型，设置隐含层为两层，第一层 5 个神经元，第二层 4 个神经元。

```
dt = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,4),
random_state=1)
#采用L-BFGS方法求解MLP；alpha为L2参数；隐含层有2层，第一层5个神经元，第二层4个神经元
dt.fit(X_train, y_train)#训练模型
cengindex = 0
for wi in dt.coefs_:
    cengindex += 1
    print('第%d层网络层:' % cengindex)
    print('权重矩阵维度:',wi.shape)
    print('系数矩阵:\n',wi)
```

```
dt.score(X_test, y_test)#用测试集评估模型的好坏
```

BP 神经网络构建

本小节以常见的 BP 神经网络为例，讲述神经网络的 python 实现方法。

我们使用的案例是 1974 个化合物的小肠上皮细胞渗透性数据，其自变量是 729 个分子描述符，因变量为小肠上皮细胞渗透性，属于分类问题，而我们要做的就是根据已有数据构建一个小肠上皮细胞渗透性预测模型。实现代码如下：首先导入相应包并设置随机种子

```
import pandas as pd
from keras.models import Model, Sequential
from keras.layers import Conv2D, Conv2DTranspose, Input, MaxPooling2D, add, Dropout,
core, Dense, Activation,BatchNormalization
from sklearn.model_selection import train_test_split
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import accuracy_score
import keras
import numpy as np
from numpy import random
random.seed(1)
```

再导入数据，进行预处理

```
xiangmu = 'Caco-2'
data1 = pd.read_csv("./data/1975-729.csv")#读取数据
data2 = pd.read_csv("./data/1975-5.csv")
data1 = data1.values
y_data = data2[xiangmu]
y_data = y_data.values
X = data1
Y = y_data
#将数据集划分为训练集与测试集,并对因变量进行Onehot编码
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.1, random_state=1, stratify =Y)
Y_train = keras.utils.to_categorical(y_train, 2)
Y_test = keras.utils.to_categorical(y_test, 2)
```

接下来搭建 BP 神经网络模型

```
model = Sequential()#逐层搭建神经网络模型
model.add(Dense(600, input_dim=729, init='uniform', activation='relu'))
model.add(BatchNormalization())
model.add(Dense(100,input_dim=600, init='uniform', activation='relu'))
model.add(BatchNormalization())
model.add(Dense(50, input_dim=100, init='uniform', activation='relu'))
model.add(BatchNormalization())
model.add(Dense(10,input_dim=50, init='uniform', activation='relu'))
model.add(BatchNormalization())
```

```
model.add(Dense(2,input_dim=10, activation='softmax'))
model.summary()#查看模型摘要
model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.1),
metrics=['acc'])#选择类别交叉熵作为损失函数，Adam作为训练的优化器，学习率为0.1
model_checkpoint = ModelCheckpoint('./data/bp-'+xiangmu+'-1.hdf5',
monitor='loss', verbose=1, save_best_only=True)#设定模型的ckpt储存方式
```

最后进行模型拟合并进行预测

```
model.fit(X_train, Y_train, epochs = 1000, batch_size = 1000, callbacks=[
                                        model_checkpoint])#训练模型
model.load_weights('./data/bp-'+xiangmu+'-1.hdf5')#读取ckpt文件
y_pred = model.predict(X_test)#进行预测
y_pred = np.argmax(y_pred, axis=1)
acc = accuracy_score(y_test, y_pred) * 100#对预测值的精度进行估计
print("\\nTesting Accuracy: {:.3f} %".format(acc))
```

总结

本章主要讨论了前馈神经网络的数学模型、正向和反向计算方法，RBF 神经网络的数学模型和计算方法等，并对其它神经网络形式做了简单介绍。但在理论和实践中是针对分类问题进行讲解和编程实现的。对于回归问题，限于篇幅，并未涉及，读者可参考马丁 T. 哈根的书《神经网络设计（原书第 2 版）》进行学习。

9.8 习题

1、请推导 BP 算法 2 的过程，并用 Python 或 R 语言编程实现。

2、根据 RBF 神经网络计算原理，请使用 Python 或 R 语言编写程序实现。

3、在基于 Python 中 sklearn 的多层感知器网络构建中，请对参数进行消融实验，探索参数的影响情况和最佳组合。若使用 R 语言，完成同样的工作。

4、在基于 Python 中 sklearn 的多层感知器网络构建中，请使用前面章节所讲过的决策树、贝叶斯、支持向量机等模型进行方法对比实验，比较各模型对于此问题解决的效果。若使用 R 语言，完成同样的工作。

第十章　深度学习

深度学习和人工神经网络之间存在密切的关系，人工神经网络是深度学习的基础，可以将人工神经网络视为深度学习的基本组成单元。深度学习通过使用深层次的神经网络（深度神经网络），强调通过层次化的方式学习数据的表示，每一层对数据进行不同层次的特征提取，能够学习更复杂的特征。

10.1　简介

理论上来说，参数越多的模型复杂度越高、容量（Capacity）越大，这意味着它能完成越复杂的学习任务。但一般情形下，复杂模型的训练效率低，易陷入过拟合，因此难以受到人们青睐。而随着云计算、大数据时代的到来，计算能力的大幅提高可缓解训练低效性，训练数据的大幅增加则可降低过拟合风险，因此，以深度学习（Deep Learning）为代表的复杂模型开始受到人们的关注。

典型的深度学习模型就是有许多层的神经网络。显然，对神经网络模型，提高容量的一个简单办法是增加隐层的数目。隐层多了，相应的神经元连接权、阈值等参数就会更多。模型复杂度也可通过单纯增加隐层神经元的数目来实现，从上一章的学习中可以了解道，单隐层的多层前馈网络已具有很强大的学习能力；但从增加模型复杂度的角度来看，增加隐层的数目显然比增加隐层神经元的数目更有效，因为增加隐层数不仅增加了拥有激活函数的神经元数目，还增加了激活函数嵌套的层数。然而，多隐层神经网络难以直接用经典算法（例如标准 BP 算法）进行训练，因为误差在多隐层内反向传播时，往往会发散而不能收敛到稳定状态。

无监督逐层训练（Unsupervised Layer-wise Training）是多隐层网络训练的有效手段，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，这称为预训练（Pre-training）；在预训练全部完成后，再对整个网络进行微调（Fine-tuning）训练。在使用无监督逐层训练时，首先训练第一层，这是关于训练样本的受限玻尔兹曼机（RBM，Restricted Boltzmann machine）模型，可按标准的 RBF 训练；然后，将第一层预训练好的隐结点视为第二层的输入结点，对第二层进行预训练；……。各层预训练完成后，再利用 BP 算法等对整个网络进行训练。

事实上，“预训练 + 微调”的做法可视为将大量参数分组，对每组先找到局部看来比较好的设置，然后再基于这些局部较优的结果联合起来进行全局寻优。这样就

在利用了模型大量参数所提供的自由度的同时，有效地节省了训练开销。

另一种节省训练开销的策略是权共享（Weight Sharing），即让一组神经元使用相同的连接权。这个策略在卷积神经网络（CNN，Convolutional NeuralNetwork）[116] [117] 中发挥了重要作用。

以 CNN 进行手写数字识别任务为例 [117]，如图10.1所示，网络输入是一个 32×32 的手写数字图像，输出是其识别结果，CNN 复合多个“卷积层”和“采样层”对输入信号进行加工，然后在连接层实现与输出目标之间的映射。每个卷积层都包含多个特征映射，每个特征映射（Feature Map）是一个由多个神经元构成的“平面”，通过一种卷积滤波器提取输入的一种特征。例如，图10.1中第一个卷积层由 6 个特征映射构成，每个特征映射是一个 28×28 的神经元阵列，其中每个神经元负责从 5×5 的区域通过卷积滤波器提取局部特征。采样层亦称为池化层（Pooling），其作用是基于局部相关性原理进行亚采样，从而在减少数据量的同时保留有用信息。例如图10.1中第一个采样层有 6 个 14×14 的特征映射，其中每个神经元与上一层中对应特征映射的 2×2 邻域相连，并据此计算输出。通过复合卷积层和采样层，图10.1中的 CNN 将原始图像映射成 120 维特征向量，最后通过一个由 84 个神经元构成的连接层和输出层连接完成识别任务。CNN 可用 BP 算法进行训练，但在训练中，无论是卷积层还是采样层，其每一组神经元（即图10.1中的每个“平面”）都是用相同的连接权，从而大幅减少了需要训练的参数数目。

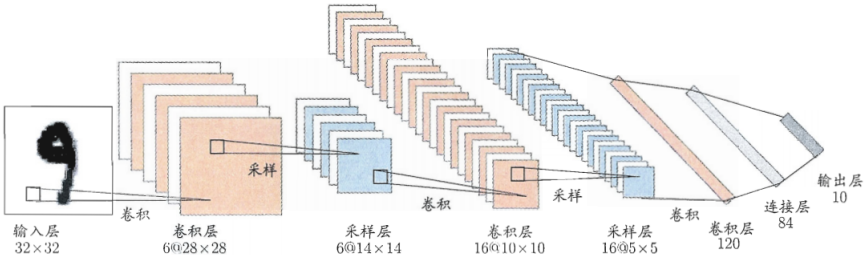


图 10.1 卷积神经网络用于手写数字识别 [117]

也可以从另一个角度来理解深度学习。无论是深度信念网络（DBN，Deep Belief Network）还是 CNN，其多隐层堆叠、每层对上一层的输出进行处理的机制，可看作是在对输入信号进行逐层加工，从而把初始的、与输出目标之间联系不太密切的输入表示，转化成与输出目标联系更密切的表示，使得原来仅基于最后一层输出映射难以完成的任务成为可能。换言之，通过多层处理，逐渐将初始的“低层”特征表示转化为“高层”特征表示后，用“简单模型”即可完成复杂的分类等学习任务。由此可将深度学习理解为进行特征学习（Feature Learning）或表示学习（Representation Learning）。

以往在机器学习用于现实任务时，描述样本的特征通常需由人类专家来设计，这

称为特征工程（Feature Engineering）。众所周知，特征的好坏对泛化性能有至关重要的影响，人类专家设计出好特征也并非易事；特征学习则通过机器学习技术自身来产生好特征，这使机器学习向“全自动数据分析”又前进了一步。

10.2 卷积神经网络

神经网络在得到广泛应用的同时，参数过多、容易发生过拟合和训练时间长等缺点也暴露出来。能否减少神经网络中参数的数目，并进一步提升神经网络的性能？CNN 应运而生。

卷积神经网络（CNN，Convolutional Neural Network）又称为卷积网络，是在图像处理和计算机视觉领域应用较为广泛的一种神经网络。与全连接神经网络同属于神经网络模型，相对于全连接神经网络而言，卷积神经网络的不同之处在于加入了卷积层（Convolution）和池化层（Pooling）两种结构，这两种结构是 CNN 必不可少的结构。前面已经介绍了它们基本的计算原理。

CNN 最早可以追溯到 1986 年 BP 算法的提出 [118]，1989 年 LeCun [119] 将 BP 算法用到多层神经网络中，1998 年 LeCun [117] 提出 LeNet-5 模型，卷积神经网络的雏形完成。在接下来近十年的时间里，卷积神经网络的相关研究趋于停滞，原因有两个：一是研究人员意识到多层神经网络在进行 BP 训练时的计算量极其之大，当时的硬件计算能力完全不可能实现；二是包括支持向量机在内的浅层机器学习算法也渐渐开始展露头角。直到 2006 年，Hinton [120] 在 Science 发文，指出“多隐层神经网络具有更为优异的特征学习能力，并且其在训练上的复杂度可以通过逐层初始化来有效缓解”，深度学习开始觉醒，并逐渐走入人们的视线。2009 年，李飞飞牵头建立了 ImageNet 数据集，该数据集包含图片的种类和数量远远超过以往所有的数据集。自 2010 年以来，每年都举行 ImageNet 大规模视觉识别挑战赛（ILSVRC），研究团队在给定的数据集上评估其算法，并在几项视觉识别任务中争夺更高的准确性。在刚开始的两年，冠军被支持向量机取得，而在 2012 年，AlexNet 取得了比赛的冠军，识别的正确率得到明显提升，这之后占据高位的一直是卷积神经网络，包括我们现在所熟知的 VGGNet，GooLeNet，ResNet 等。2016 年，AlphaGo 战胜了围棋世界冠军、职业九段棋手李在石，更是将深度学习和卷积神经网络推向了高潮。

图10.2展示了近些年来 CNN 的发展历程。

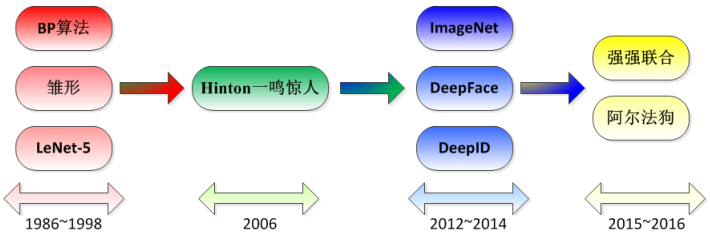


图 10.2 CNN 大事件

（一）卷积和卷积神经网络

卷积 （Convolution）运算有连续和离散两种定义，一维情形下有如下所示：

$$(f*g)(n)=\int_{-\infty}^{\infty} f(t)g(n-t)\,dt$$

$$(f*g)(n)=\sum_{t=-\infty}^{\infty} f(t)g(n-t)$$

从公式中可以发现，卷积运算先对 g 函数进行翻转，相当于在数轴上把 g 函数从右边折到左边去，也就是卷积中的“卷”。然后再把 g 函数平移到 n，在这个位置对两个函数的对应点相乘，然后相加，这个过程是卷积中的“积”。因此，所谓“卷积”，可直观理解为“卷”和“积”两个过程。

在卷积神经网络中“卷”和“积”如何体现呢？下面通过图10.3说明卷积神经网络的计算过程。

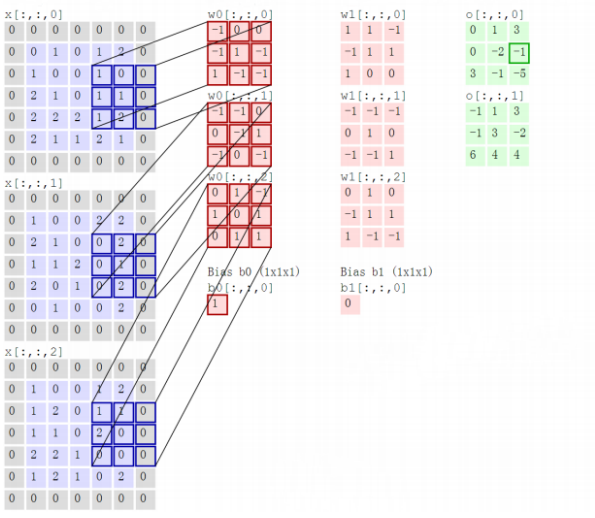


图 10.3 卷积运算

最左侧为输入，中间红色部分表示卷积核，最右侧为输出。卷积核移动到输入左上角开始计算，以此向右、向下滑动。例如，第一个卷积核移动到第一个输入的左上角时，此时计算过程为：$0\times(-1)+0\times0+0\times0+0\times(-1)+0\times1+1\times(-1)+0\times1+1\times(-1)+0\times(-1)=-2$。类似地，可以继续后面的运算。

通过计算的过程可以发现，“卷”并没有发挥作用，卷积神经网络仅仅应用了“积”。卷积神经网络具备两大特点：局部连接和权值共享。局部连接指的是卷积层的节点仅仅和其前一层的部分节点相连接；权值共享指的是同一张图使用相同的卷积核。局部连接和权值共享减少了参数数量，加快了神经网络的学习速率，同时也在一定程度上减少了过拟合的可能。

（二）基本参数

了解了卷积神经网络的基本过程后，接下来学习卷积神经网络中的几个基本参数。

1. **填充（Padding）**

在进行卷积操作的时候，6×6 的图像经过 3×3 的 filter 卷积得到 4×4 的卷积结果，如图10.4所示。用更加一般的形式表达：如果我们有一个 $n \times n$ 的图像，用 $f \times f$ 的过滤器做卷积，那么输出的维度就是 $(n-f+1) \times (n-f+1)$。在这个例子里是 6-3+1=4，因此得到了一个 4×4 的输出。

但是这样做存在明显缺点：每次卷积操作后图像尺寸都会缩小，同时角落边缘的像素点，在卷积计算的时候只被一个输出所触碰或者使用，但是中间的像素点会被多次卷积计算，所以那些在角落的像素点参与卷积计算较少，意味着丢掉一些图像边缘位置的信息。

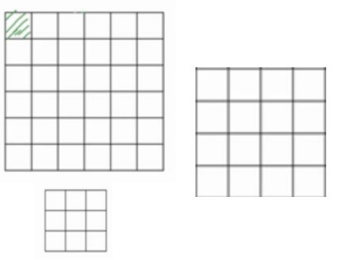


图 10.4 图像由 6×6 到 3×3

卷积操作中的Padding，可以自行指定 p 的值。一般有如下两种 Padding 方式。

（1）Valid：不填充，即 $p=0$；

（2）Same：填充后输出矩阵的大小与原矩阵保持一致，即 $p=\dfrac{f-1}{2}$，其中卷积核的大小为 $f \times f$。通常将 f 设置为奇数，这样我们方便我们对称填充。

2、步幅（Stride）

滑动卷积核时，我们会先从输入的左上角开始，每次往左滑动一列或者往下滑动一行逐一计算输出，我们将每次滑动的行数和列数称为Stride。

3、池化（Pooling）

Pooling是实现下采样的一种运算，能在提取图像关键特征的基础上减小图片尺寸，以减少训练中的参数数量，达到减少计算量、增大感受野并防止过拟合的目的。

如图10.5所示，Pooling 主要有最大值池化（Max-Pooling）和平均值池化（Average-Pooling）两种。最大值池化从所选区域的矩阵元素中取最大值，而平均值池化是将所选区域的矩阵元素求和后取平均值。

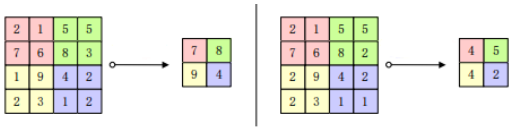


图 10.5 最大值池化和平均值池化示意图

4、欠拟合和过拟合

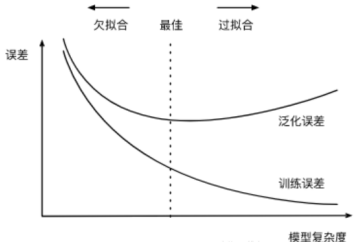


图 10.6 欠拟合和过拟合

（1）欠拟合

如图10.6，欠拟合是指模型拟合程度不高，数据距离拟合曲线较远，或指模型没有很好地捕捉到数据特征，不能够很好地拟合数据。解决办法是提高模型的复杂度，如增加神经网络的层数、宽度，减少正则化的参数等。

（2）过拟合

如图10.6，一个假设在训练数据上能够获得比其他假设更好的拟合，但是在训练数据外的数据集上却不能很好地拟合数据，表现为泛化能力差，称为过拟合。导致过拟合的原因有很多，主要包括以下几点：模型过于复杂、参数过多；数据太少；训练集和测试集的数据分布不同；样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系。解决办法包括 L_1、L_2 正则化，扩增数据，Dropout；Early Stopping 等。

L_1、L_2 正则化

在损失函数上添加正则化项，其中 L_1 正则化为参数 w 的绝对值（岭回归）、L_2 正则化为参数 w 的平方值（Lasso 回归）。通过对 w 值的修正，使其偏离不会太大，从而减少过拟合的产生。

扩增数据

更多的数据集，能够让搭建的网络在更多的数据中不断修正调整，进而训练出更好的模型。然而数据量都是有限的，所以可以从已有数据出发，对其进行调整以

得到更多的数据集。如图10.8，可以针对已有图像应用随机图像转换，如旋转、对称和放大等，来增加图像数量。

图 10.7 原图

图 10.8 翻转、镜面、缩放变换

Dropout

不同于 L_1、L_2 正则化通过修改代价函数防止过拟合，Dropout 修改的对象是神经网络。如图10.9所示，其思想主要是随机使得神经网络隐藏层中的一些神经元失活，在简化网络结构的同时防止了过拟合，而且随机失活迫使每一个神经元学习到有效的特征，增强了搭建网络的泛化能力。

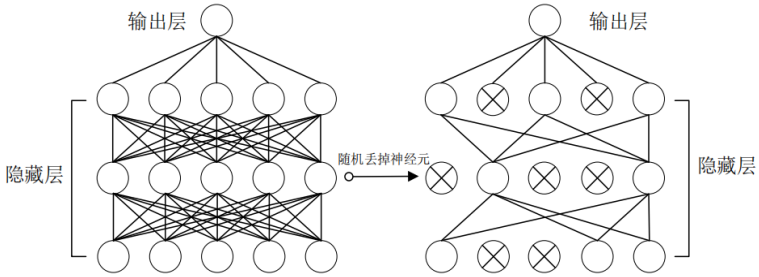


图 10.9 Dropout

在神经网络中使用 Dropout 时，一般以概率 0.5 选择神经元是否有效，即每层仅留下半数的神经元。因此使用 Dropout 相当于训练了多个只有半数隐层神经元的神经网络，每一个这样的网络，都可以给出一个分类结果，这些结果有对有错。随着训练的进行，大部分网络可以给出正确的分类结果，此时少数的错误分类结果不会对最终结果造成大的影响。

Early Stopping

当训练有足够的表示能力甚至会过拟合的大模型时，经常观察到训练误差会随着时间的推移逐渐降低但验证集的误差会再次上升。这意味着如果返回使验证集误差最低的参数设置，就可以获得更好的模型（有希望获得更好的测试误差）。在每次验证集误差有所改善后，我们存储模型参数的副本。当训练算法终止时，返回这些参数而不是最新的参数。当验证集上的误差在事先指定的循环次数内没有进一步改善时，算法就会终止。需要注意的是，通过提前终止自动选择超参数的显著代价是训练期间要定期评估验证集。

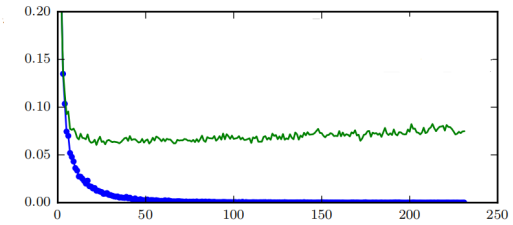


图 10.10 Early Stopping

图10.10中横轴为 epoch 次数，纵轴为损失。蓝线为训练集上的损失，随着 epoch 次数的不断增加明显减少，绿线为验证集上的误差。上述策略称为提前终止（Early Stopping）。

（三）卷积神经网络的一般结构

在卷积神经网络中，输入图像通过多个卷积层和池化层进行特征提取，逐步由低层特征变为高层特征；高层特征再经过全连接层和输出层进行特征分类，产生一维向量，表示当前输入图像的类别。因此，根据每层的功能，卷积神经网络可以划分为两个部分：由输入层、卷积层和池化层构成特征提取器，以及由全连接层和输出层构成分类器。

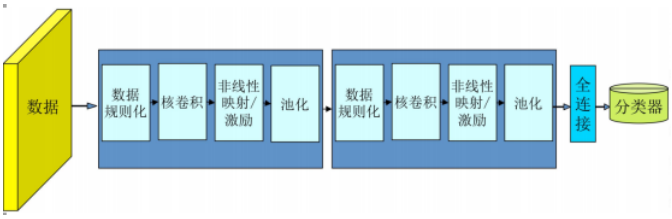


图 10.11 卷积神经网络架构

在实际操作中，卷积神经网络的架构可以如图10.11所示。

1、数据规则化：Batch Normalization

随着网络的深度增加，每层特征值分布会逐渐地向激活函数的输出区间的上下两端（激活函数饱和区间）靠近，这样下去会导致梯度消失。Batch Normalization（BN）通过将该层特征值分布重新拉回标准正态分布，特征值将落在激活函数对于输入较为敏感的区间，输入的小变化可导致损失函数较大的变化，使得梯度变大，避免梯度消失，同时也可加快收敛。

BN 在实际工程中被证明了能够缓解神经网络难以训练的问题。主要来说有如下优点：

（1）BN 使得神经网络中每层输入数据的分布相对稳定，加速了模型学习速度；

（2）BN 使得模型对网络中的参数不那么敏感，简化调参过程，使得网络学习更加稳定；

（3）BN 允许网络使用饱和性激活函数（例如 sigmoid，tanh 等），缓解梯度消失问题；

（4）BN 具有一定的正则化效果。

2、卷积池化层：特征提取

如图10.12，给定图片，如何确定图中包含哪些物体？可以检测图片中的边缘来达到识别的目的。比如说，图片中的栏杆和行人的轮廓线都可以看作是垂线，同样，栏杆就是很明显的水平线，它们也能被检测到。图片中的大部分物体都能用垂直和水平边缘线来刻画，那么如何在图像中检测这些边缘呢？下面通过一个简单例子来说明。

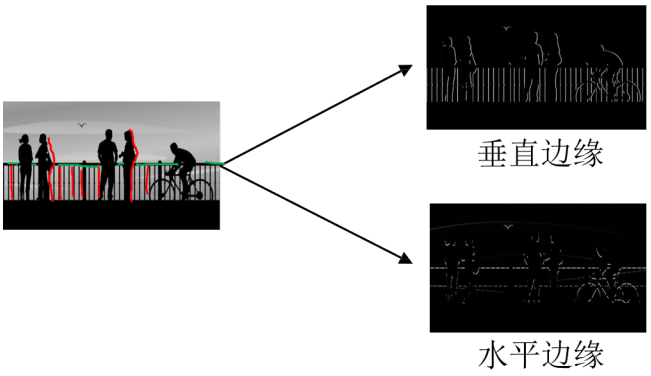


图 10.12 边缘检测

一个 6×6 的灰度图，可以表示为左亮右暗的形式，如图10.13左所示。很明显，中间存在明显的分界线，因此我们试图将该垂直边缘识别出来。选择卷积核为 3×3，相应的输出为 4×4。仍旧通过颜色的亮暗来表示卷积核和输出矩阵，如图10.13中和图10.13右所示。不难看出，输出矩阵对应的图中间有段发亮的区域，对应原图中的垂直边缘。

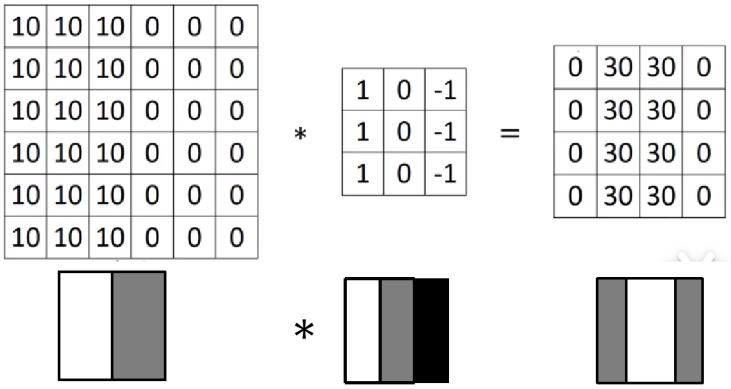


图 10.13 卷积提取特征示意图

显然这里检测到的边缘宽度远大于原图中的边缘宽度，这和原图像素大小有关，如果改用像素值为 500×500 的图像或者更大，检测效果将会有显著提升。垂直边缘

通过这样的方式可以很容易检测出，水平边缘的检测类似。推广到一般情况，卷积核就是通过这样的方式，提取到输入图像的特征。

3、全连接层

把分布式特征映射到样本标记空间，即将特征整合到一起输出为一个值，减少特征位置对分类带来的影响。

例如：假设你是一只小蚂蚁，你的任务是找小面包。你的视野比较窄，因此只能看到很小的一片区域。当你找到一片小面包之后，你不确定你找到的是不是全部的小面包，所以你和其余的蚂蚁一块儿开了个会，把所有的小面包都拿出来以确定是否已经找到所有的小面包。某种程度上可以认为，全连接层就是这个蚂蚁大会。需要注意的是，在数据输入到全连接层前还需要进行拉直操作，将所有数据拉成一维向量。或者可以认为是某种卷积操作。由于全连接层参数过多，正逐渐被全局平均池化（GAP，Global Average Pooling）的方法代替。

（四）常见的卷积神经网络

1、LeNet

LeNet由 LeCun [117] 在 1998 年提出，用于解决手写数字识别的视觉任务。

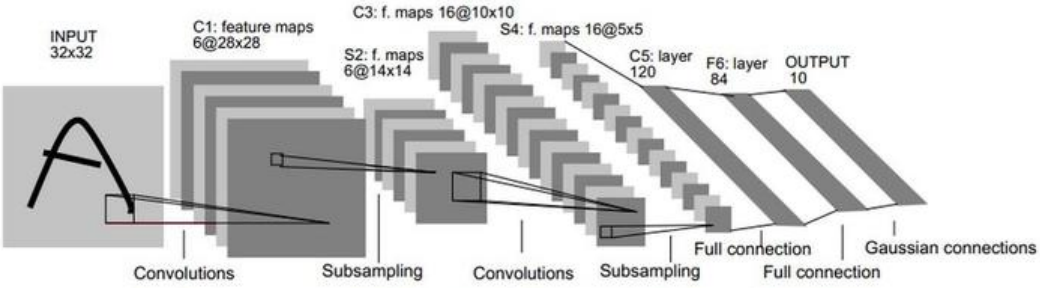


图 10.14 LeNet

图10.14中，Convolutions 表示卷积，Subsampling 表示下采样，即池化，Full Connection 表示全连接，Gaussian Connection 表示高斯连接。具体来看，LeNet 共有五层，包括 2 层卷积层和 3 层全连接层。

LeNet 中每个卷积层均使用尺寸为 5×5 的卷积核，步长为 1，并使用 sigmoid 激活函数。两个最池化层中池化核均为 2×2，且步长为 2，这里选择的是平均池化。由于池化核尺寸与步长相同，因此池化后在输出上每次滑动所覆盖的区域互不重叠，池化不改变通道数量。当卷积层块的输出传入全连接层块时，全连接层块会将小批量中每个样本变平。全连接层块含 3 个全连接层。它们的输出个数分别是 120、84 和 10，其中 10 为输出类别的个数。

2、AlexNet

AlexNet是 2012 年 ImageNet 竞赛的冠军，自此越来越多越来越复杂的神经网络被提出并得到广泛应用。

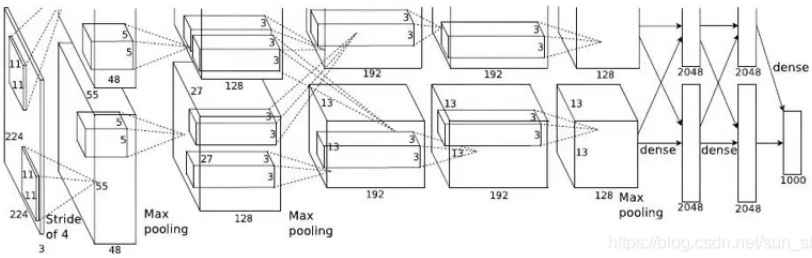


图 10.15 AlexNet

限于单个 GPU 的计算能力，AlexNet 在两个 GPU 上实现卷积神经网络的搭建和计算。图10.15中 Stride 表示步幅，Max Pooling 表示池化选择最大池化，Dense 为全连接。搭建和计算的大致如下：

AlexNet 共有八层，包括 5 层卷积层和 3 层全连接层。输入尺寸为 227× 227× 3 的彩色图片，使用 11× 11 的卷积核，步长选择为 4，输出图像的尺寸为 55× 55× 96，其中输出图像长（宽）度 $=\dfrac{\text{输入图像长（宽）度}+2\times\text{Padding}-\text{卷积核长（宽）度}}{\text{步幅}}+1$，因此 $\dfrac{227+2\times 0-11}{4}+1=55$。Pooling 选择步长为 2,3× 3 的池化核，输出图像的尺寸变为 27× 27× 96。

再次使用 5×5 的卷积核，步长选择为 1，并且使用 padding=2，输出得到 $\dfrac{27+2\times 2-5}{1}+1=27$，通道数为 256。Pooling 选择步长为 2,3×3 的池化核，输出图像的尺寸为 13× 13× 256。

第三、四、五层均选择 3× 3 的卷积核，且步幅均为 1。Pooling 仍旧选择步幅为 2,3×3 的池化核，输出图像的尺寸变为 6× 6× 256。

AlexNet 具有如下四个特点：

（1）使用ReLU 函数作为激活函数，很好地增强了网络的非线性表达能力。

（2）引入局部响应归一化（LRN,Local Response Normalization）。通过 ReLU 函数得到的值域没有固定区间，因此要对得到的结果进行归一化。具体来说放大对分类贡献较大的神经元，抑制对分类贡献较小的神经元。

（3）使用重叠的最大池化。即池化层中卷积核的尺寸大于步长，这样池化层的输出间出现重叠和覆盖，提升了特征的丰富性。

（4）通过 Dropout 和数据扩增等方式来防止神经网络出现过拟合现象。

3、ZFNet

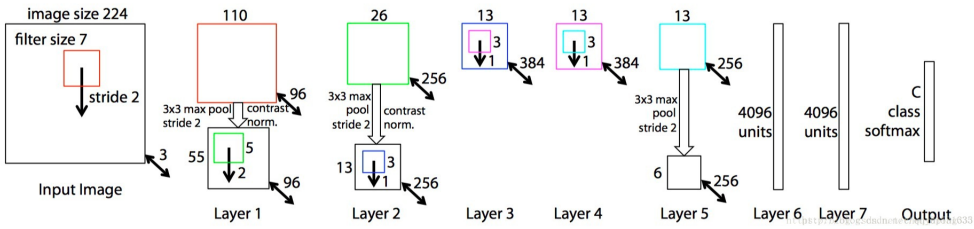


图 10.16 ZFNet

图10.16中 image size 为输出图片的尺寸，224×224，filter size 为卷积核的大小，stride 为步幅，max pool 表示池化选择最大池化，softmax 为最终用于分类的函数，Layer 标明了 ZFNet 的每一层。ZFNet是 AlexNet 的改进版本，共有八层，仍旧是 5 层卷积层和 3 层全连接层。改进主要有以下两点：

（1）使用一块 GPU 搭建稠密连接结构；（2）第一个卷积层中卷积核的尺寸从 11× 11 变为 7× 7，同时步长从 4 减小为 2；为了让后续输出图像的尺寸与 AlexNet 中的输出图像尺寸保持一致，第 2 个卷积层的步长从 1 变为 2。

整体来看，ZFNet 的改进似乎并不明显。那么为什么要做这样的修改？修改的动机又是什么呢？这是基于卷积神经网络深层特征的可视化提出的。可视化操作，针对的是已经训练好的网络，或者训练过程中的网络快照。可视化操作不会改变网络的权重，只是用于分析和理解在给定输入图像时网络观察到了什么样的特征，以及训练过程中特征发生了什么变化。

可视化主要有如下三个操作：

（1）Rectification：因为使用的是 ReLU 激活函数，前向传播时只将正值原封不动输出，负值置 0，“反激活”过程与激活过程没什么分别，直接将来自上层的输出再次输入到 ReLU 激活函数中即可。

（2）Unpooling：在前向传播时，记录相应 max pooling 层每个最大值来自的位置，在 Unpooling 时，根据来自上层的 map 直接填在相应位置上，其余位置为 0。

（3）Transposed Convolution：卷积操作输出图像的尺寸一般小于等于输入图像的尺寸，Transposed Convolution 可以将尺寸恢复到与输入相同，相当于上采样过程，该操作的做法是，与 Convolution 共享同样的卷积核，但需要将其左右上下翻转（即中心对称），然后作用在来自上层的输出图像进行卷积，结果继续向下传递。

不断进行上述操作，可以将特征映射回输入所在的像素空间，进而可以呈现出人眼可以理解的特征。给定不同的输入图像，看看每一层关注到最显著的特征是什么。

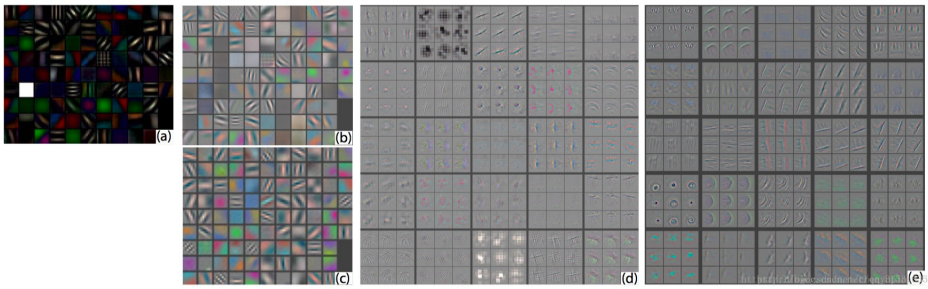


图 10.17 可视化特征图

图10.17（a）为没有经过裁剪的图片经过第一个卷积层后的特征可视化图，注意到有一个特征全白，（b）为 AlexNet 中第一个卷积层特征可视化图，（c）为 ZFNet 中第一个卷积层可视化图，可以看到相比前面有更多的独特的特征以及更少的无意义的特征，如第 3 列的第 3 到 6 行，（d）为 AlexNet 中第二个卷积层特征可视化图，（e）为 ZFNet 中的第二个卷积层特征可视化图，可以看到（e）中的特征更加干净，清晰，保留了更多的第一层和第二层中的信息。

通过对 AlexNet 的特征进行可视化，Zeiler 等人发现 AlexNet 第一层中有大量的高频和低频信息的混合，却几乎没有覆盖到中间的频率信息；且第二层中由于第一层卷积用的步长为 4 太大了，导致了有非常多的混叠情况。为了解决这个问题，Zeiler 等人提高采样频率，将步长从 4 调整为 2，与之相应的将卷积核尺寸也缩小（可以认为步长变小了，卷积核没有必要看那么大范围了），修改后第一层呈现了更多更具区分力的特征，第二层的特征也更加清晰 [121]。

4、VGGNet

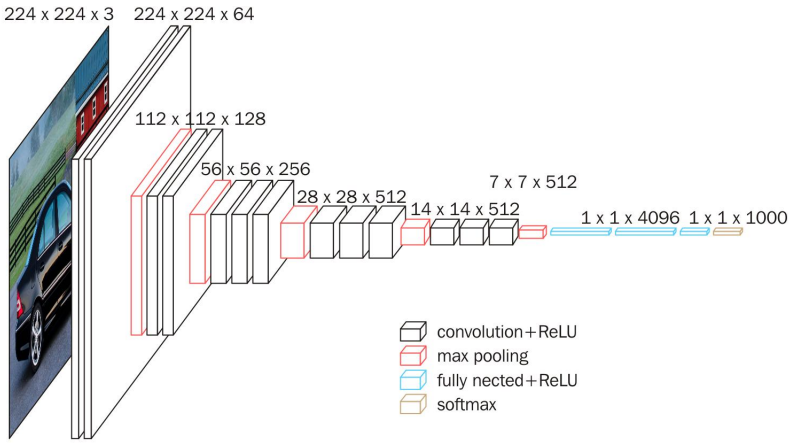


图 10.18 VGGNet

VGGNet（Visual Geometry Group Net）是由牛津大学计算机视觉组合和 Google DeepMind 公司研究员 [122] 一起研发的深度卷积神经网络。它探索了卷积神经网络的深度和其性能之间的关系，通过反复的堆叠 3×3 的卷积核和 2×2 的最大池化层，

成功地构建了 16 到 19 层的卷积神经网络。因此，VGGNet 是指一系列网络，下面以 VGG-16 为例进行介绍。

图10.18中 convolution+Relu 表示卷积后选择 Relu 激活函数，max pooling 表示池化层选择最大池化，full nected+Relu 表示全连接层后选择 Relu 激活函数，softmax 激活函数用于最后的分类。

VGGNet 是一种专注于构建卷积层的简单网络，相比 AlexNet 和 ZFNet 参数量大大减少，一个很重要的原因是用到了卷积核的堆叠。例如通过 2 个 3×3 的卷积核代替 1 个 5×5 的卷积核，3 个 3×3 的卷积核代替 1 个 7×7 的卷积核。通过这样的方式不仅节省了大量参数，还使得网络获得了更大的感受野，同时增强了卷积神经网络的非线性能力。

5、GoogLeNet

在通过卷积神经网络识别处理图像时，经常遇到图像突出部分的大小差别很大的情况。如图10.19和10.20，猫的图像可以是以下任意情况。每张图像中猫所占区域是不同的。

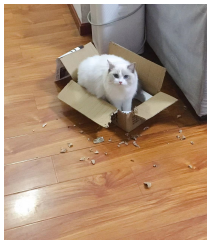


图 10.19 不同位置的猫 1

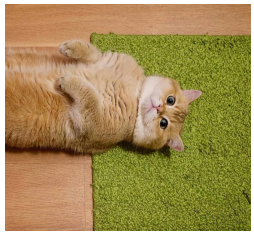


图 10.20 不同位置的猫 2

由于信息位置的巨大差异，为卷积操作选择合适的卷积核大小就比较困难。信息分布更全局性的图像偏好较大的卷积核，信息分布比较局部的图像偏好较小的卷积核。同时，非常深的网络更容易过拟合。将梯度更新传输到整个网络是很困难的。再加上简单地堆叠较大的卷积层非常消耗计算资源，因此考虑在同一层级上运行具备多个尺寸的卷积核呢？网络本质上会变得稍微宽一些，而不是更深。2015 年，串并联网络架构的 GoogLeNet 应运而生 [123]。GoogLeNet 最基本的网络块是 Inception，它是一个并联网络块，经过不断的迭代优化，发展出了 Inception-v1、Inception-v2、Inception-v3、Inception-v4、Inception-ResNet 共 5 个版本。Inception 家族的迭代逻辑是通过结构优化来提升模型泛化能力、降低模型参数。

Inception 就是把多个卷积或池化操作，放在一起组装成一个网络模块，设计神经网络时以模块为单位去组装整个网络结构。在未使用这种方式的网络里，我们一层往往只使用一种操作，比如卷积或者池化，而且卷积操作的卷积核尺寸也是固定大小的。但是，在实际情况下，在不同尺度的图片里，需要不同大小的卷积核，这样才能使性能最好，或者或，对于同一张图片，不同尺寸的卷积核的表现效果是不一

样的，因为他们的感受野不同。所以，我们希望让网络自己去选择，Inception 便能够满足这样的需求，一个 Inception 模块中并列提供多种卷积核的操作，网络在训练的过程中通过调节参数自己去选择使用，同时，由于网络中都需要池化操作，所以此处也把池化层并列加入网络中。

10.3 简单循环神经网络

循环神经网络（RNN，Recurrent Neural Network）是一类以序列（Sequence）数据为输入，在序列的演进方向进行递归（Recursion）且所有节点（循环单元）按链式连接的递归神经网络（Recursive Neural Network）。

在日常生活和学习中，有许多数据的不同样本之间是有关联的，比如气象数据、经济数据，而之前讲述的神经网络并不能很好的学习这一关联，而循环神经网络就能很好的解决这一问题。

要了解 RNN，首先从一个简单的循环神经网络开始（如图10.21）。其中，$\boldsymbol{X}$ 是一个向量，它表示输入层的值（图中并未画出表示神经元节点的圆圈）；$\boldsymbol{S}$ 是一个向量，它表示隐藏层的值；$\mathbf{U}$ 是输入层到隐藏层的权重矩阵；$\boldsymbol{O}$ 也是一个向量，它表示输出层的值；$\mathbf{V}$ 是隐藏层到输出层的权重矩阵；而权重矩 $\mathbf{W}$ 就是隐藏层上一时间的值对这一时间的输入的权重，它表示上一时间隐藏层的值对这一时间的影响。这一简单的循环神经网络与全连接神经网络的不同就在于他多出了权重矩阵 $\mathbf{W}$。

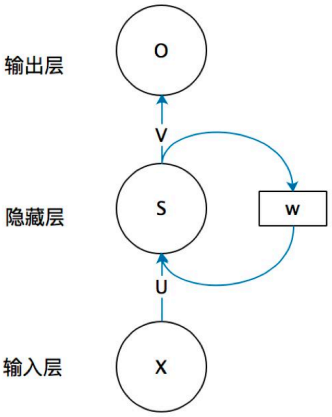


图 10.21 简单的循环神经网络

图10.22展示了这个神经网络展开的样子，可以发现权重矩阵 $\mathbf{W}$ 实际上就相当于 $t-1$ 时刻对 t 时刻的一个影响。

这个网络在 t 时刻接收到输入值为 $\boldsymbol{X}_t$，隐藏层的值是 $\boldsymbol{S}_t$ ，输出值是 $\boldsymbol{O}_t$ 。关键一点是，$\boldsymbol{S}_t$ 的值不仅仅取决于 $\boldsymbol{X}_t$，还取决于 $\boldsymbol{S}_{t-1}$。可以用下面的公式来表示循

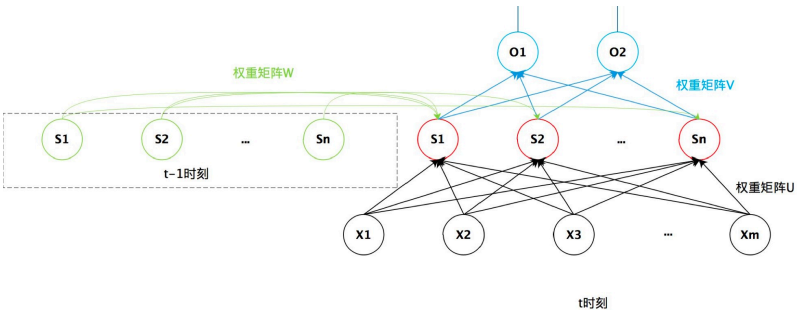


图 10.22 简单的循环神经网络

环神经网络的计算方法：

$$\boldsymbol{O}_t = g\left(\mathbf{V} \cdot \boldsymbol{S}_t\right)$$

$$\boldsymbol{S}_t = f\left(\mathbf{U} \cdot \boldsymbol{X}_t + \mathbf{W} \cdot \boldsymbol{S}_{t-1}\right)$$

循环神经网络的参数同样可以利用前文提到的反向传播算法进行求解。但同时，需要注意的是：反向传播算法按照时间逆序将信息一步步向前传递，如果输入序列较长，会存在梯度爆炸与梯度消失问题，也称为长程依赖问题，长程依赖问题的具体表现为：很久以前的输入，对当前时刻的网络影响较小，在反向传播时，梯度也很难影响到很久以前的输入。

10.4 长短时记忆神经网络

为了解决长程依赖问题，人们将基于图10.23中所展示神经网络进行了进一步改进，引入了门控机制（Gating Mechanism）。门（Gate）是一种让信息有选择地通过的方式，它由一个 sigmoid 函数和一个点乘运算组成。利用 sigmoid 函数返回 0 到 1 一个之间的数字的特性，令这个数字描述每个元素有多少信息可以通过，0 表示不通过任何信息，1 表示全部通过。

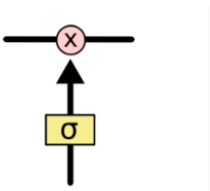


图 10.23 门

门机制控制了对于当前时刻的有哪些信息将被保留，有哪些信息将被遗忘。也将控制有哪些新信息将被加入记忆信息，有哪些信息又将从记忆信息中被提取出作为

当前时刻的输出。下面介绍一个加入了门控机制后的循环神经网络——长短时记忆神经网络（LSTM，Long Short-Term Memory Neural Network）。LSTM 的结构有很多种形式，但是都大同小异。这里介绍一种较为流行的结构 GRU（Gated Recurrent Unit）。结构图10.24如下：

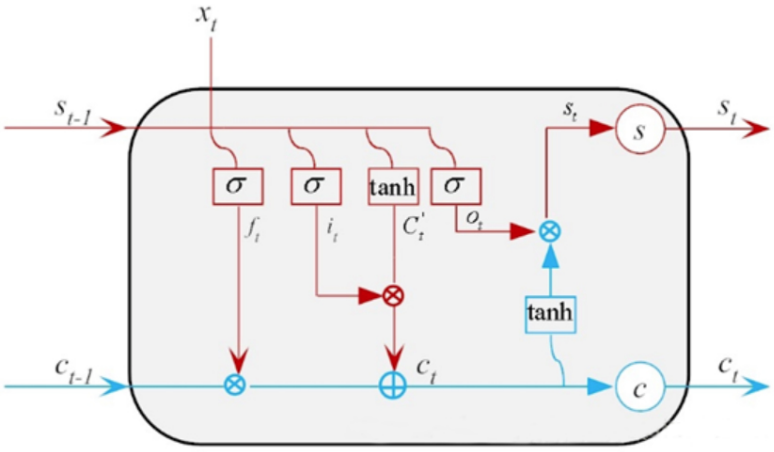


图 10.24 LSTM 完整结构逻辑图

LSTM 模型的关键是引入了一组记忆单元 c_t，以及三个控制“门”：输入门、输出门、控制门。它们允许网络可以学习何时遗忘历史信息，何时用新信息更新记忆单元。在时刻 t 时，记忆单元 c_t 记录了到当前时刻为止的所有历史信息，并受三个“门”控制，三个门中元素的数值取值范围是 $(0,1)$：

$$Input\ Gate: i_t = \sigma(U_i x_t + W_i s_{t-1})$$
$$Forget\ Gate: f_t = \sigma(U_f x_t + W_f s_{t-1})$$
$$Output\ Gate: o_t = \sigma(U_o x_t + W_o s_{t-1})$$

下面具体理解 LSTM 模型。首先了解 LSTM 模型中的核心机制：记忆细胞状态。

细胞状态就像一根传送带，直接在整个链上运行，运行过程中只有一些少量的线性交互，以保证信息在上面流传保持不变。这些信息将会随着网络一直传递下去，保证了在学习当前时刻输入信息的同时，不会遗忘掉之前学习到的信息。

在 LSTM 模型中引入的门控机制，目的就是为了控制记忆细胞状态，向其中增添与删除信息。首先是遗忘门 f_t 的更新机制：

$$f_t = \sigma(U_f x_t + W_f s_{t-1})$$

其中，W_f 是关于隐藏状态 s_t 的参数矩阵，U_f 是关于当前输入 x_t 的参数矩阵。由于 sigmoid 函数的存在，f_t 的取值范围在 $(0,1)$ 之间。

遗忘门 f_t 的具体功能是根据 x_t（当前输入）和 s_{t-1}（前一个隐藏状态），为记忆状态 C_{t-1}（t-1 时刻状态）中的每个元素输出 0 ~ 1 之间的数字，以决定每一个 C_{t-1} 元素保留多少信息。1 代表完全保留，0 代表彻底删除。

下一步是确定向记忆细胞中增添多少新信息。这一步骤分为两部分，首先需要确定当前输入信息，再决定输入信息的保留程度。对于当前输入信息 $\tilde{C}_t$ 的计算方式如下：

$$\tilde{C}_t = tanh(U_c x_t + W_c s_{t-1})$$

这里同样利用上一时刻隐藏层向量 s_{t-1} 与当前输入向量 x_t 分别乘上参数矩阵 W_c 与 U_c，进行一个线性变化。不同的是由于 $\tilde{C}_t$ 不是一个门，不需要其取值范围限定在 $(0,1)$ 之间，于是选择用 tanh() 函数代替 sigmoid 函数进行非线性激活。

得到 $\tilde{C}_i$ 后，使用输入门 i_t 来决定保留 $\tilde{C}_i$ 中的信息至记忆细胞状态中：

$$i_t = \sigma(U_i x_t + W_i s_{t-1})$$

与遗忘门相同，W_i 是关于前一个隐藏状态 s_{t-1} 的参数矩阵，U_i 是关于当前输入 x_t 的参数矩阵。同理，输入门 i_t 的取值范围在 $(0,1)$ 之间。

最后是输出门 o_t：基于当前的 c_t，输出那些作为当前时刻隐藏层的信息 h_t。o_t 的更新方法依旧类似于之前的遗忘门与输出门：

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$

其中，U_o 是关于前一个隐藏状态 h_{t-1} 的参数矩阵，W_0 是关于当前输入，o_t 的参数矩阵，b_0 是一个偏置向量。同理，由于 sigmoid 函数，输入门 o_tt 的取值范围也在 $(0,1)$ 之间。在获得 o_t 之后，利用之前更新好的当前时刻历史记忆信息 C_t，进行当前时刻隐藏层 h_t 信息的更新：

$$h_t = o_t(C_t)$$

在更新 h_t 的时候，首先用非线性函数 tanh() 对 C_t 进行激活，然后再与 o_t 进行点乘运算，保证向量维度保持不变。h_t 会作为下一个时刻的输入之一，再参与未来新的一系列更新。

最后对 LSTM 结构进行小结，但在此之前，有两点需要强调的是：

（1）尽管三个门，以及当前输入信息 $\tilde{C}_t$ 的更新都具有相同的形式，但是每各更新所用的参数都是独立不同的，这保证了反向传播时对三个门控单元即当前记忆信息独立地进行更新。

（2）以上公式中的所有参数都是可学习的参数。它们在最初是自动随机生成的随机数，但是随着模型的不断训练，这些参数会随着反向传播算法不断的进行更新，最终使得损失函数数值尽可能的小以提高模型的表达能力。

LSTM 结构回顾：

（1）LSTM 有单独的细胞状态，该状态贯穿整个 LSTM 网络。

（2）用遗忘门 $\tilde{C}_t$ 和输入门 i_t 决定历史记忆信息与新记忆信息的保留或放弃。

（3）当前输入信息 $\tilde{C}_t$ 来源于 s_{t-1} 和 x_t。

（4）当前记忆信息由 $f_{tt-1} + i_t \times \tilde{C}_t$ 计算得出。

（5）输出门控制细胞状态的输出 $s_t = o_t(C_t)$。

以上所介绍的循环神经网络主要大量应用在自然语言处理场合。自然语言处理（NLP）是指利用人类交流所使用的自然语言与机器进行交互通讯的技术。通过人为的对自然语言的处理，使得计算机对其能够可读并理解。在本章末尾，引入了一个在自然语言处理情境应用循环神经网络的代码示例，可供参考与学习。

10.5 自编码器

在机器学习任务中经常需要采用某种方式对数据进行有效编码，例如对原始数据进行特征提取便是几乎所有机器学习问题均需解决的编码任务。除此之外，对数据进行降维处理或稀疏编码也是常见的编码任务。通常对数据进行编码时需要按照编码要求将原始数据转化为特定形式的编码数据，并要求编码数据尽可能多地保留原始数据信息，对这样的编码数据进行分析处理不仅会更加方便，而且可保证分析处理的结果较为准确。对数据的编码过程事实上是一个数据映射过程。若将某类原始数据 $\boldsymbol{X}$ 编码为特定形式的数据 y，则需找到某个合适的映射 f 使得 $y = f(\boldsymbol{X})$。通常称映射 f 为编码器，并称将原始数据 $\boldsymbol{X}$ 映射为编码数据 y 的过程为编码过程。若编码数据 y 保留了大部分原始数据 $\boldsymbol{X}$ 中的信息，则从理论上说一定存在某种方式将数据 y 映射为与 $\boldsymbol{X}$ 相近的数据 $\boldsymbol{X}'$。假设存在映射 g 满足：

$$\boldsymbol{X}' = g(y) = g[f(\boldsymbol{X})] \tag{10.5.1}$$

则认为编码器 f 可对原始数据 $\boldsymbol{X}$ 进行有效编码，此时称映射 g 为解码器。

通常将用于实现自编码器的神经网络模型称为自编码器。由于神经网络输入层不具备数据处理能力，故自编码器中除输入层之外还包括两个分别用于实现编码器 f 和解码器 g 的模块，每个模块的神经元层数既可为单层也可为多层。一个单隐层的自编码器网络结构如图10.25所示：

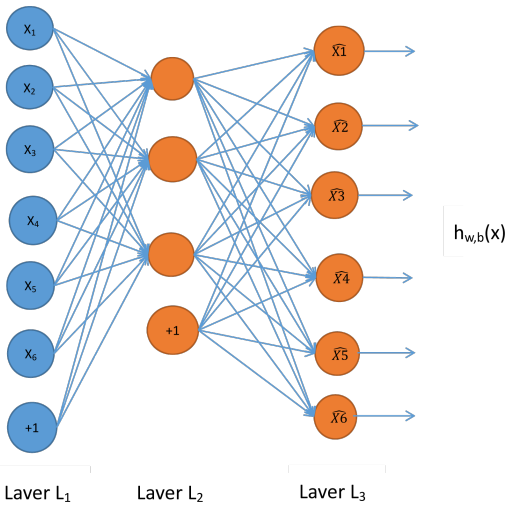


图 10.25 单隐层的自编码器

由图10.25可以看到，自编码器输出层的节点与输入层相等，训练这个网络以期望得到近似恒等函数。该模型首先使用编码器对输入数据进行编码，然后使用解码器尽量将编码数据还原为输入数据，即模型输出 $\boldsymbol{X}^{'}$ 尽量保持与模型输入 $\boldsymbol{X}$ 一致。显然，若编码器 f 和解码器 g 均采用恒等映射，即 $f(a)=a$ ，$g(a)=a$，则有如下关系：

$$y=f(\boldsymbol{X})=\boldsymbol{X}, \boldsymbol{X}'=g(y)=g[f(\boldsymbol{X})]=\boldsymbol{X} \tag{10.5.2}$$

此时恒有 $\boldsymbol{X}^{'}=\boldsymbol{X}$。但这样的自编码器显然毫无意义，因为自编码器的主要功能是对原始数据进行编码而非对编码数据进行还原，即更加注重自编码器中的编码器模块。

若自编码器的神经元均使用激活函数 σ,则对于输入数据 $\boldsymbol{X}=(X_1,X_2,\ldots,X_m)^{\mathrm{T}}$，由前向计算方法可知自编码器隐含层第 j 个神经元的输出应为：

$$f_j(\boldsymbol{X})=\sigma\left(\sum_{i=1}^{m} w_{ij}^{(1)} X_i+b_j^{(2)}\right) \tag{10.5.3}$$

由于隐含层数据处理结点个数为 s，故自编码器的隐含层可将 m 维数据输入转化为 s 维数据 $\boldsymbol{y}=(f_1(\boldsymbol{X}),f_2(\boldsymbol{X}),\ldots,f_s(\boldsymbol{X}))^T$ 若 $s<m$，则是对数据 $\boldsymbol{X}$ 进行降维；若 $s>m$，则是对数据 $\boldsymbol{X}$ 进行升维。自编码器隐含层到输出层的数据处理过程与此类似，最终输出 $\boldsymbol{X}^{'}$ 为：

$$\boldsymbol{X}'=(\mathrm{g}_1(\mathrm{y}),\mathrm{g}_2(\mathrm{y}),\ldots,\mathrm{g}_{\mathrm{m}-1}(\mathrm{y}))^{\mathrm{T}} \tag{10.5.4}$$

上式输出层第 j 个神经元输出 $g_j(y)$ 的具体取值为：

$$\mathrm{g_j(y)} = \sigma\left(\sum_{\mathrm{i=1}}^{\mathrm{s}} \mathrm{w_{ij}^{(2)}} \mathrm{f_i}(\boldsymbol{X}) + \mathrm{b_j^{(3)}}\right) \tag{10.5.5}$$

若已经完成模型训练过程，则 $\boldsymbol{X}^{'}$ 应与数据输入 $\boldsymbol{X}$ 差别不大。自编码器的模型训练通常使用不带标注信息的示例样本，故这是一种无监督学习方式。但由于要求自编码器的输入数据与输出数据尽可能接近，故对于训练样本 $\boldsymbol{X}_k$，若模型参数均已知，则可直接通过对比模型输入 $\boldsymbol{X}_k$ 与输出 $\boldsymbol{X}_k^{'}$ 的差异确定损失函数 $L(\boldsymbol{X}_k, \boldsymbol{X}_k^{'})$。这相当于将自编码器的训练样本集看作监督学习的训练样本集 $\{(\boldsymbol{X}_1, \boldsymbol{X}_1), (\boldsymbol{X}_2, \boldsymbol{X}_2) \ldots, (\boldsymbol{X}_{\mathrm{n}}, \boldsymbol{X}_{\mathrm{n}})\}$，即根据自编码器的特点将无监督学习方式转化为监督学习方式。

可得自编码器模型优化目标函数：

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{k=1}^{n} L\left(\boldsymbol{X}_k, \boldsymbol{X}_k'\right) \tag{10.5.6}$$

其中 W 为自编码器的参数向量。

损失函数 L 通常采用 $\boldsymbol{X}_k$,$\boldsymbol{X}_k^{'}$ 之间欧式距离的平方，即 $L\left(\boldsymbol{X}_k, \boldsymbol{X}_k'\right) = \|\boldsymbol{X}_k - \boldsymbol{X}_k'\|_2^2$。由此可得目标函数的具体形式为：

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{k=1}^{n} \|\boldsymbol{X}_k - \boldsymbol{X}_k'\|_2^2 \tag{10.5.7}$$

确定了目标函数之后，可采用适当模型优化算法并结合反向传播算法对模型参数进行优化，得到所求的自编码器。具体过程与反向传播神经网络优化过程类似。

如上所述，若自编码器隐含层神经元数目 s 大于输入数据维度 m，则可实现对输入数据的升维，此时只需确保编码后数据向量中取值为 0 的分量较多即可实现对原始数据的稀疏编码。

由于自编码器隐含层输出向量 $\mathrm{y} = (\mathrm{f_1}(\boldsymbol{X}), \mathrm{f_2}(\boldsymbol{X}), \ldots, \mathrm{f_s}(\boldsymbol{X}))^{\mathrm{T}}$ 即为编码数据，故只需限制 y 中取值为 0 的分量较多便可实现对原始数据的稀疏编码。令自编码器隐含层神经元采用 Sigmoid 激活函数，则对于训练样本集 $D = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_n, \}$ 自编码器隐含层第 j 个神经元的平均激活程度为：

$$\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X}) = \frac{1}{\mathrm{n}}\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{f_j}\left(\boldsymbol{X}_{\mathrm{i}}\right) \tag{10.5.8}$$

为将隐含层第 j 个神经元的输出值限制为 0，可令 $\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X}) = \varepsilon$。这里 ε 为某个接近于 0 的正数。由于 $f_j(\boldsymbol{X}_i) > 0$ 且 $f_j(\boldsymbol{X}_i)$ 在数据集 D 上期望接近于 0，故 $f_j(\boldsymbol{X}_i)$ 的取值也接近于 0。若对隐含层中大部分神经元输出均施加此约束条件，即 $\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X}) = \varepsilon, j = 1, 2, \cdots, s$，则可保证 y 中大部分元素取值均接近于 0，实现对原始数据的稀疏编码。为将该约束条件纳入训练过程，需调整模型优化的目标函数，即在原始目标函数 $J(W)$ 基础上添加约束 $\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X}) = \varepsilon, j = 1, 2, \cdots, s$ 的惩罚项 $\lambda\left(\bar{f}_j(\boldsymbol{X})\right)$，将目标函数转化为如下形式：

$$\begin{aligned} J(\boldsymbol{W}) &= J'(W) = J(W) + \lambda\left(\bar{f}_j(\boldsymbol{X})\right) \\ &= \frac{1}{n}\sum_{k=1}^{n} L\left(\boldsymbol{X}_k, \boldsymbol{X}'_k\right) + \alpha\lambda\left(\bar{f}_j(\boldsymbol{X})\right) \end{aligned} \tag{10.5.9}$$

其中，α 为惩罚项的权重。当 α 取值较大时，所求自编码器的编码数据具有较好的稀疏性，但会舍弃较多的原始数据信息；当 α 取值较小时，所求自编码器的编码数据会保留较多的原始数据信息，但稀疏性较差。

一般，惩罚项 $\lambda\left(\bar{f}_j(\boldsymbol{X})\right)$ 具有如下形式：

$$\lambda\left(\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})\right) = \frac{1}{\mathrm{s}}\sum_{\mathrm{j}=1}^{\mathrm{s}}\left(\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X}) - \varepsilon\right)^2 \tag{10.5.10}$$

除此之外，还可使用 $\left(\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})\right)$ 与 ϵ 之间的 K-L 散度作为单个隐含层结点的惩罚项，由此得到如下惩罚项：

$$\lambda\left(\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})\right) = \frac{1}{\mathrm{s}}\sum_{\mathrm{j}=1}^{\mathrm{s}}\mathrm{KL}\left(\varepsilon, \bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})\right) = \frac{1}{\mathrm{s}}\sum_{\mathrm{j}=1}^{\mathrm{s}}\left[\varepsilon\log\frac{\varepsilon}{\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})} + (1-\varepsilon)\log\frac{1-\varepsilon}{1-\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})}\right] \tag{10.5.11}$$

其中 $\mathrm{KL}\left(\varepsilon, \bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})\right)$ 为 $\bar{\mathrm{f}}_{\mathrm{j}}(\boldsymbol{X})$ 与 ϵ 之间的 K-L 散度。

上述自编码器模型均要求模型的输入与输出尽可能相一致。这种自编码器得到的编码数据有时未必是对原始数据的最优表示。

如果某种自编码器能对被破坏的数据 $\boldsymbol{X}^b$ 进行编码并将其解码为真实原始数据 $\boldsymbol{X}$，则该自编码器的编码方式显然更为有效。通常称这种通过引入噪声来增加编码鲁棒性的自编码器为降噪自编码器，如图10.26所示。

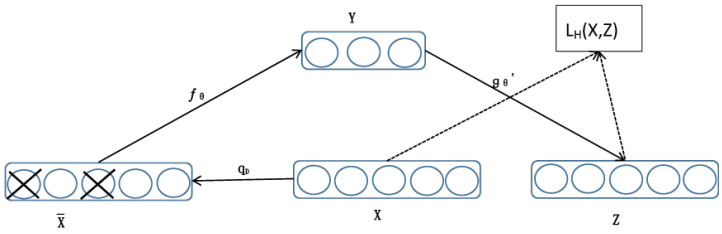


图 10.26 降噪自编码器模型

对于输入数据 $\boldsymbol{X}$，按照 q_D 分布进行加噪 “损坏”，由图10.26可以看到，加噪过程是按照一定的概率将输入层的某些节点清零，然后将其作为自编码器的输入进行训练，除了输入层数据的处理不同，其余部分都和自编码器相同。

10.6 玻尔兹曼机

玻尔兹曼机（BM，Boltzmann Machines）是一种反馈随机神经网络，其具有两种输出状态，即 0 或 1。输出状态的取值根据概率统计方法取得的结果而决定，这种概率统计方法类似于一个 Boltzmann 分布。在神经元状态变化中引入了统计概率，网络的平衡状态服从 Boltzmann 分布，其运行机制基于模拟退火算法，可以理解为：离散 Hopfield 神经网络 + 模拟退火 + 隐单元 =BM。

10.6.1 随机神经网络

如果将 BP 算法中的误差函数看作一种能量函数，则 BP 算法通过不断调整网络参数使其能量函数按梯度单调下降，而反馈网络（Hopfield 神经网络）通过动态演变过程使网络的能量函数沿着梯度单调下降，在这一点上两类网络的指导思想是一致的。正因如此，常常导致网络落入局部极小点而达不到全局最小点，对于 BP 网，局部极小点意味着训练可能不收敛；对于 Hopfield 网络，则得不到期望的最优解。导致这两类网络陷入局部极小点的原因是：网络的误差函数或能量函数是具有多个极小点的非线性空间，而所用的算法却一味追求网络误差或能量函数的单调下降。也就是说，算法赋予网络的是只会“下山”而不具备“爬山”的能力。如果为具有多个局部极小点的系统打一个形象的比喻，设想托盘上有一个凸凹不平的多维能量曲面，若在该曲面上放置一个小球，它在重力作用下．将滚入最邻近的一个低谷（局部最小点）而无法跳出。但该低谷不一定就是曲面上最低的那个低谷（全局最小点）。因此，局部极小问题只能通过改进算法来解决。随机神经网络可赋予网络既能“下坡”也能“爬山”的本领，因而能有效地克服上述缺陷。随机网络与其他神经网络相比有两个主要区别：

（1）在学习阶段，随机网络不像其他网络那样基于某种确定性算法调整权值，而是按某种概率分布进行修改。

（2）在运行阶段，随机网络不是按某种确定性的网络方程进行状态演变，而是按某种概率分布决定其状态的转移。神经元的净输入不能决定其状态取 1 还是取 0，但能决定其状态取 1 还是取 0 的概率。这就是随机神经网络算法的基本概念，图10.27为随机网络算法与梯度下降算法区别的示意图。

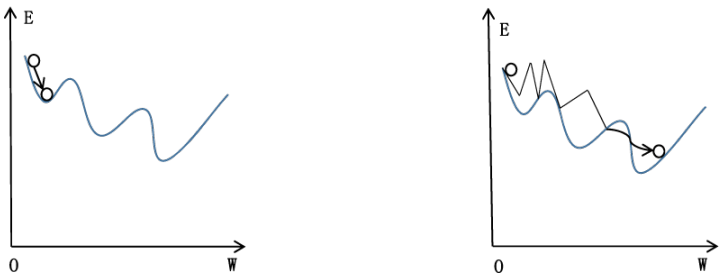


图 10.27 梯度下降算法与随机网络算法区别的示意图

10.6.2 模拟退火算法

模拟退火算法是随机网络中解决能量局部极小问题的一个有效方法，其基本思想是模拟金属退火过程，金属退火过程大致是，先将物体加热至高温，使其原子处于高速运动状态，此时物体具有较高的内能；然后，缓慢降温，随着温度的下降，原子运动速度减慢，内能下降；最后，整个物体达到内能最低的状态。模拟退火过程相当于沿水平方向晃动托盘，温度高则意味着晃动的幅度大，小球肯定会从任何低谷中跳出，而落入另一个低谷。这个低谷的高度（网络能量）可能比小球原来所在低谷的高度低（网络能量下降），但也可能反而比原来高（能量上升）。后一种情况的出现. 从局部和当前来看，这个运动方向似乎是错误的；但从全局和发展的角度看，正是由于给小球赋予了“爬山”的本事，才使它有可能跳出局部低谷而最终落入全局低谷。当然，晃动托盘的力度要合适，并且还要由强至弱（温度逐渐下降），小球才不致因为有了“爬山”的本领而越爬越高。

在随机网络学习过程中，先令网络权值作随机变化，然后计算变化后的网络能量函数。网络权值的修改应遵循以下准则：若权值变化后能量变小，则接受这种变化；否则也不应完全拒绝这种变化，而是按预先选定的概率分布接受权值的这种变化。其目的在于赋予网络一定的“爬山”能力。实现这一思想的一个有效方法就是 Metropol 等 [124] 提出的模拟退火算法。

10.6.3 BM

神经网络模型的训练构造通常使用目标函数最小化的优化方式实现。前述各类前馈神经网络模型的训练构造均从误差最小化的角度设计目标函数，对此类目标函数进行优化后可保证在训练集上的整体预测误差达到最小且具备一定的泛化能力。事实上，还可从系统稳定性角度出发设计目标函数。由于系统越稳定则其能量越低，故为得到一个稳定的模型输出，可设计与网络模型相关的能量函数作为网络模型优

化的目标函数，由此实现对神经网络模型的优化求解。BM 便是此类神经网络的代表模型之一。BM 包含可视层与隐含层（如图10.28）所示，通过可视层神经元完成与外部的信息交互且可视层与隐含层的所有神经元均参与信息处理过程。该模型的理想效果是获得训练集 $D=\{\boldsymbol{X}_1,\boldsymbol{X}_2,...,\boldsymbol{X}_n\}$ 中样本在模型稳定状态下的输出值。

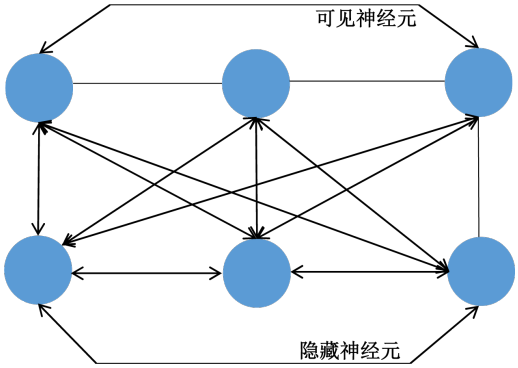


图 10.28 BM 网络运行图

BM 中所有神经元两两之间均存在信息传递且任意两个神经元之间的连接权重均相等，若使用 w_{ij} 表示 BM 中的 i 个神经元到第 j 个神经元的连接权重，则有 $w_{ij}=w_{ji}$，当 i=j 时有 $w_{ij}=w_{ji}=0$。BM 中每个神经元的输出信号均限制为 0 或 1，并且每个神经元的状态取值具有一定的随机性，即以一定概率输出 0 或 1，这个概率与该神经元的输入相关。

具体地说，对于包含 k 个神经元的 BM，由于其第 j 个神经元的输入数据为其他所有神经元的输出信号，故该结点的总输入为：

$$I_j=\sum_{i=1}^{k}w_{ij}O_i+\theta_j \tag{10.6.1}$$

其中 O_i 为第 i 个神经元的输出信号，θ_j 为第 j 个神经元所对应的偏置项。在网络中添加一个取值恒为 1 的第 0 个神经元作为偏置项，则可将偏置项 θ_j 表示为连接权重 w_{0j}，则有：

$$I_j=\sum_{i=0}^{k}w_{ij}O_i \tag{10.6.2}$$

此时可将第 j 个神经元的输出信号 O_j 取值为 1 的概率定义为

$$P\left(O_j=1\right)=\frac{1}{1+\mathrm{e}^{-\frac{I_j}{T}}} \tag{10.6.3}$$

相应地，该神经元输出为 0 的概率为：

$$P\left(O_j=0\right)=\frac{\mathrm{e}^{-\frac{I_j}{T}}}{1+\mathrm{e}^{-\frac{I_j}{T}}} \tag{10.6.4}$$

通常称参数 T 称为温度。

BM 所采用能量函数具体形式如下：

$$J\left(w_{ij}, O_i, O_j\right) = -\frac{1}{2}\sum_{i=0}^{k}\sum_{j=0}^{k} w_{ij} O_i O_j \tag{10.6.5}$$

对于第 j 个结点的输出 O_j：

若 $O_j = 0$，则对于 $i \leq j$，有:$J\left(w_{ij}, O_i, 0\right) = 0$；

若 $O_j = 1$，则对于 $i \leq j$，有：$J\left(w_{ij}, O_i, 1\right) = -\frac{1}{2}\sum_{i=0}^{k} w_{ij} O_i = -\frac{1}{2} I_j$。

若满足 $J\left(w_{ij}, O_i, 1\right) > J\left(w_{ij}, O_i, 0\right) = 0$，则说明 O_j 取 1 时的能量高于 O_j 取 0 时能量且 $I_j < 0$，可知 $P\left(O_j = 1\right) < 0.5 < P\left(O_j = 0\right)$，此时第 j 个神经元以较大概率输出使得网络能量降低的取值，但仍有可能选择使得网络能量升高的取值。

若满足 $J\left(w_{ij}, O_i, 1\right) < J\left(w_{ij}, O_i\, 0\right) = 0$，则说明 O_j 取 1 时的能量低于 O_j 取 0 时能量且 $I_j > 0$，故有 $P\left(O_j = 1\right) > 0.5 > P\left(O_j = 0\right)$，此时第 j 个神经元倾向于选择使得网络能量更低的取值作为输出。

由以上分析可知，BM 的各神经元均倾向于选择使得网络能量降低的输出值，故该网络模型的能量函数取值呈现总体下降趋势，但亦存在能量函数取值上升的可能性。这样可有效避免网络模型的优化计算陷入局部最优。

用 BM 网络进行联想时，可通过学习用网络稳定状态的概率来模拟训练集样本的出现概率。根据学习类型，BM 网络可分为自联想和异联想两种情况（如图10.29所示）。自联想 BM 网络中的可见节点既是输入节点又是输出节点，隐节点的数目由学习的需要而定，最少可以为 0。异联想 BM 网络中的可见节点需按功能分为输入节点组和输出节点组。

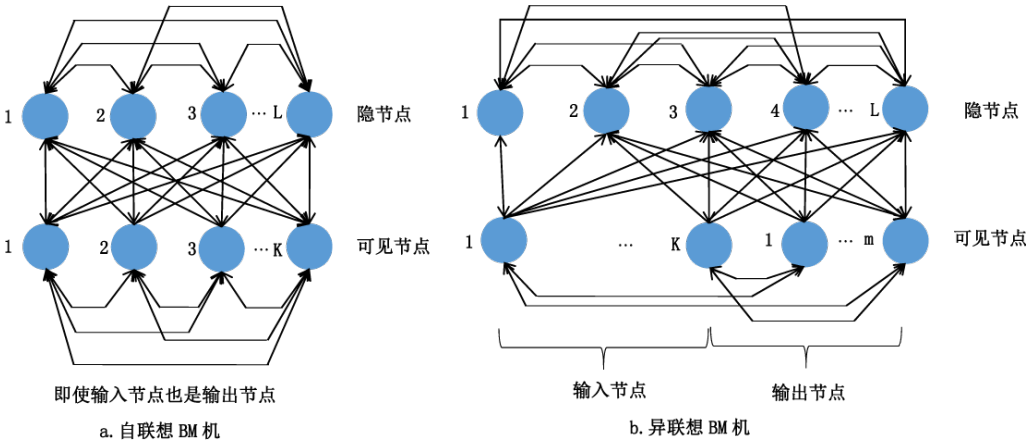


图 10.29 BM 网络拓扑结构

10.7 深度学习实践

10.7.1 R 语言实践

在本小节中，我们将利用 R 中的 keras 包，以 IMDB 数据集为例展示深度学习中的循环神经网络。IMDB 数据集包含了 50000 条偏向明显的评论，其中 25000 条作为训练集，25000 作为测试集。label 为 pos（positive）和 neg（negative），属于分类问题。

详细构骤实现代码如下：

```
library(keras)
install_keras(tensorflow = 'nightly')#导入kears包与tensorflow后端
max_unique_word <- 2500#设定取频率靠前的2500单词
max_review_len <- 100#设定影评最大长度
my_imdb <- dataset_imdb(num_words = max_unique_word)
str(my_imdb)#检查加载对象的结构
x_train <- my_imdb$train$x#数据集加载
y_train <- my_imdb$train$y
x_test  <- my_imdb$test$x
y_test  <- my_imdb$test$y
x_train <- pad_sequences(x_train, maxlen = max_review_len)
x_test <- pad_sequences(x_test, maxlen = max_review_len)#执行填充和截断过程
#使用简单循环神经网络模型
rnn_model <- keras_model_sequential()
rnn_model %>%
layer_embedding(input_dim = max_unique_word, output_dim = 128) %>%
layer_simple_rnn(units = 64, dropout = 0.2, recurrent_dropout = 0.2) %>%
layer_dense(units = 1, activation = 'sigmoid')
rnn_model %>% compile(loss = 'binary_crossentropy',optimizer = 'adam',
metrics = c('accuracy'))#编译 RNN 模型
batch_size = 128
epochs = 5
validation_split = 0.2
rnn_history <- rnn_model %>% fit(x_train, y_train,batch_size = batch_size,
epochs = epochs,validation_split = validation_split)
plot(rnn_history)
rnn_model %>% evaluate(x_test, y_test)

#部分输出结果截取如下:
24512/25000 [============================>.] - ETA: 0s - loss: 0.6390 - acc: 0.6187
24704/25000 [============================>.] - ETA: 0s - loss: 0.6387 - acc: 0.6189
25000/25000 [==============================] - 7s 292us/sample - loss: 0.6385 - acc:
                                        0.6188
$loss
[1] 0.6385054
$acc
[1] 0.61884
```

简单的 RNN 循环性能可能不够优秀，因此我们尝试使用 LSTM 模型进行优化。

```
lstm_model <- keras_model_sequential()
lstm_model %>%
layer_embedding(input_dim = max_unique_word, output_dim = 128) %>%
layer_lstm(units = 64, dropout = 0.2, recurrent_dropout = 0.2) %>%
layer_dense(units = 1, activation = 'sigmoid')
lstm_model %>% compile(loss = 'binary_crossentropy',
optimizer = 'adam',metrics = c('accuracy'))
batch_size = 128
epochs = 5
validation_split = 0.2
lstm_history <- lstm_model %>% fit(x_train, y_train,batch_size = batch_size,
epochs = epochs,validation_split = validation_split)
plot(lstm_history)
lstm_model %>%
evaluate(x_test, y_test)

#部分输出结果截取如下:
24640/25000 [============================>.] - ETA: 0s - loss: 0.3774 - acc: 0.8390
24768/25000 [============================>.] - ETA: 0s - loss: 0.3772 - acc: 0.8391
24992/25000 [============================>.] - ETA: 0s - loss: 0.3766 - acc: 0.8393
25000/25000 [==============================] - 19s 774us/sample - loss: 0.3767 - acc:
                                          0.8392
$loss
[1] 0.3766563
$acc
[1] 0.83924
```

从这个简单的例子中，我们可以看到 LSTM 模型的性能比简单的 RNN 模型有了显著的提高，但与此同时，LSTM 模型的计算时间大约翻了一倍。

10.7.2 Python 语言实践

基于 PyTorch 的深度神经网络的构建

本节使用的数据集为 CIFAR-10 数据集，该数据集共有 60000 张彩色图像，这些图像是 32×32，分为 10 个类，每类 6000 张图。这里面有 50000 张用于训练，构成了 5 个训练批，每一批 10000 张图；另外 10000 用于测试，单独构成一批。

实现代码如下：首先使用 PyTorch 构建 CNN

```
import torch.nn
from torch import nn
from torch.nn import Conv2d,Sequential,MaxPool2d,Flatten,Linear
#构造网络结构
class NetModel(nn.Module):
    def __init__(self):
        super(Module, self).__init__()
        self.model=Sequential(Conv2d(3,32,5,padding=2),
```

```
        #输入通道数为3，卷积核数为32，方形卷积核长度为5，两边填充2个维度的0
            MaxPool2d(2),#池化视野大小为2
            Conv2d(32,32,5,padding=2), MaxPool2d(2),
            Conv2d(32,64,5,padding=2), MaxPool2d(2),
            Flatten(),#在0维展开
            Linear(1024,64),#定义两个全连接层
            Linear(64,10))
    def forward(self,x):
        x=self.model(x)
        return x #使用默认激活函数
```

接着读取数据，进行数据划分

```
#读取数据
from torch.utils.data import DataLoader
import  torchvision
import  torch.nn
train_data = torchvision.datasets.CIFAR10#读入训练集
(root = './cifar10data', train=True, transform=torchvision.transforms.ToTensor())
test_data=torchvision.datasets.CIFAR10#读入测试集(
root='./cifar10data', train=False, transform=torchvision.transforms.ToTensor())
train_len = len(train_data)
test_len = len(test_data)
print("训练集的长度为：{}".format(train_len))
print("测试集的长度为：{}".format(test_len))
```

再设置相应参数，代入数据进行模型训练

```
train_loader = DataLoader(train_data,batch_size=64)#加载数据模型
test_loader = DataLoader(test_data,batch_size=64)
net = NetModel()#加载网络模型
loss_fn = nn.CrossEntropyLoss()#定义损失函数
learing_rate = 1e-2#定义优化器
optimzer = torch.optim.SGD(net.parameters(),lr=learing_rate)
total_train_step=0#记录训练的次数
total_test_step=0#记录测试的次数
epoch=10#训练的轮数
net.train()#训练模型
for i in range(epoch):
    print("------第{}轮数训练开始------".format(i+1))
    for data in train_loader:
        img,target = data
        output = net(img)
        loss=loss_fn(output,target)
        optimzer.zero_grad()#对模型进行优化
        loss.backward()
        optimzer.step()
        total_train_step=total_train_step+1
        if total_train_step %  100 ==0:
            print("训练次数{},Loss:{}".format(total_train_step,loss))
    net.eval()#测试步骤开始
    total_test_loss=0
    total_test_accuracy=0
```

```
    with torch.no_grad():
        for data in test_loader:
            img,target = data
            output =net(img)
            loss = loss_fn(output,target)
            total_test_loss+=loss
            accuracy = (output.argmax(1)==target).sum()  #1代表横向
            total_test_accuracy+=accuracy.item()
        print("在整个测试集上的损失率为:{}".format(total_test_loss))
        print("在整个测试集上的正确率为:{}".format(total_test_accuracy/test_len))
```

在输出结果中可以看到，经过 10 轮训练，正确率有了大幅度的提升。

基于 keras 的深度神经网络的构建

下面将以 MNIST 手写数字识别数据集为例展示深度学习神经网络的构建方法。

LeNet-5 是由有着卷积神经网络之父美誉的 Yann LeCun（中文译为杨立昆）于 1998 年提出的一种经典的卷积网络结构 [117]。它是第一个成功应用于数字识别问题的卷积神经网络。在 MNIST 数据集上，LeNet-5 模型可以达到大约 99.2% 的正确率。作为早期的一种卷积神经网络结构，LeNet-5 的提出极大的推动了后续卷积神经网络的发展，它通常被认为是 CNN 的开山之作。其构建过程代码如下：首先展示 MNIST 数据集，并完成数据准备

```
from keras.datasets import mnist
from keras.utils import np_utils
from matplotlib import pyplot as plt
(X0,Y0),(X1,Y1) = mnist.load_data()#读取数据
print(X0.shape)
plt.figure()
fig,ax = plt.subplots(2,5)
ax = ax.flatten()
for i in range(10):
  Im = X0[Y0==i][0]
  ax[i].imshow(Im)
plt.show();
N0 = X0.shape[0];N1 = X1.shape[0]
print([N0,N1])
X0 = X0.reshape(N0,28,28,1) / 255
X1 = X1.reshape(N1,28,28,1) / 255
YY0 = np_utils.to_categorical(Y0)
YY1 = np_utils.to_categorical(Y1)
print(YY1)
```

再基于 keras 进行模型搭建

```
from keras.layers import Conv2D,Dense,Flatten,Input,MaxPooling2D
from keras import Model
input_layer = Input([28,28,1])#构建模型
x = input_layer
x = Conv2D(6,[5,5],padding = "same", activation = 'relu')(x)
```

```
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)
x = Conv2D(16,[5,5],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)
x = Flatten()(x)
x = Dense(120,activation = 'relu')(x)
x = Dense(84,activation = 'relu')(x)
x = Dense(10,activation = 'softmax')(x)
output_layer = x
model = Model(input_layer, output_layer)
model.summary()
```

模型的编译通过 model.compile 实现。它的损失函数为 categorical_crossentory，这等价于要优化的是一个对数似然函数。选择 Adam 优化算法，需要监控预测精度，因此定义 metrics 为 accuracy。详细代码如下：

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X0, YY0, epochs=10, batch_size=200, validation_data=[X1, YY1])
#模型的输出结果为：
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 5s 82us/step - loss: 0.3830 - accuracy
: 0.8897 - val_loss: 0.0941 - val_accuracy: 0.9709
...
Epoch 10/10
60000/60000 [==============================] - 2s 33us/step - loss: 0.0199 - accuracy
: 0.9935 - val_loss: 0.0331 - val_accuracy: 0.9896
<keras.callbacks.callbacks.History at 0x22dc0ce7c88>
```

在输出结果中可以看到，LeNet-5 在这个手写数字识别数据集上已经达到了一个很高的精度，其预测结果的正确率已经达到了 98.96% 。

总结

本章节针对图像数据，讨论了卷积神经网络、循环神经网络、长短时记忆网络、自编码器等网络的结构和计算过程。但限于篇幅，并未讨论视频、文本、自然语言等数据类型中的问题，也未讨论生成对抗网络、迁移学习、强化学习等比较新的深度学习模型，请参考邱锡鹏所著《神经网络与深度学习》。

10.8 习题

1. 总结 CNN 算法的计算过程，并指出其存在的问题。
2. 当图像尺寸变为 2 倍，CNN 的参数数量变为几倍？为什么？
3. 解释 Batch Normalization 的意义。

4. 模型的超参数是什么？它和参数有什么不同？针对上一节中的程序，请进行超参数优化。

5. 对于课本中第一个例子，进行网络结构优化，以期望达到最佳分类正确率。

6. 根据课本中描述的常见卷积神经网络，请用 PyTorch 或 Kearas 构建网络模型并编程实现。

第五部分

集成学习

第十一章　随机森林

集成学习（Ensemble Learning）的主核心思想是通过结合多个学习器的预测结果，以获得比任何单一学习器更好的性能。集成学习的目标是通过利用多样性和集体智慧来提高模型的泛化能力和鲁棒性，适用于分类、回归和聚类等各种机器学习任务。集成学习方组合的多个学习器可以是同质的也可以是异质的，对于数据中的噪声和异常值有更强的鲁棒性，但更难解释整体模型的决策过程。集成学习常常被用作提高模型性能的有效手段，通过与其他机器学习方法结合适应，可以达到更好的效果。当单个模型具有不同的优点和缺点，或者当数据集很大且多样化以及在面对不确定性和复杂任务时，集成学习通常表现得更为出色。

本章及接下来两章将介绍三种主要的集成学习方法：随机森林（Random Forest）、Boosting 方法（Boosting Method）和模型平均（Model Average）。

11.1　简介

回归树和分类树的优点是解释性强且易于计算，然而就像一个硬币具有两面，基于单棵树的回归或者分类模型也具有明显的缺陷：

1、由于模型的简单性及仅用单一的常数作为最终区域的预测值，从而使得单棵树的回归或分类模型很难具有最优的预测能力 [71]；

2、单棵树的预测是不稳定的 [125] [70]，数据有较小的变动就可能会导致完全不一样的分裂变量和分裂点；

3、单棵树具有容易遭受选择偏差影响的问题，即取值多的分类自变量比取值少的分类自变量更容易被选择为分裂变量 [126] [127] [128]。

为了克服单一统计模型（单棵树）稳定性差（方差大）的缺陷，集成学习方法得到了广泛的研究和发展。所谓集成学习，就是指分类（回归）器的集成。集成学习通过构建并结合多个弱学习器来完成学习任务，一般的方法是先产生一组个体学习器，再用某种策略将它们结合起来，常见的结合策略有：平均法、投票法和学习法等。例如Bagging方法利用 Bootstrap 方法（有放回抽样）对训练集进行抽样，得到一系列新的训练集，对每个训练集都构建一棵树，最后通过平均法、投票法组合所有预测器得到最终的预测模型。

后来，为了克服单棵树的缺陷及降低每次 Bootstrap 抽样之间的相关性，Breiman [129] 综合以往集成学习的优缺点提出了一种新的集成学习方法——随机森林（Ran-

dom Forests)。接下来我们将详细介绍随机森林的基本思想、算法步骤及变量重要性评价等内容。

11.2 随机森林基本概况

在引入随机森林这一算法之前，我们先考虑与之相关的概念——Bagging。Bagging 是 Bootstrap Aggregating 的缩写。

特别的，我们考虑每个预测器都是决策树模型的 Bagging 算法。在 Bagging 算法中，我们注意到在对训练集 $N_0 = \{(\boldsymbol{X}_1, Y_1), \cdots\cdots, (\boldsymbol{X}_n, Y_n)\}$ 进行 Bootstrap 抽样（样本量为 n ）以获得 M 个新的训练集，记为 $\{N_m, m = 1, 2, \cdots, M\}$，鉴于 Bootstrap 抽样的性质，可以证明新训练集 $N_b, b \in \{1, 2, \cdots, M\}$ 大约只包含原训练集 N_0 的三分之二（因为一个样本，经 n 次有放回抽样，仍未被抽中的概率是 $\lim\limits_{n\to\infty}\left(1-\dfrac{1}{n}\right)^n = \mathrm{e}^{-1} \approx \dfrac{1}{3}$)。这些未被使用的观测值称为此树的袋外观测值（Out-Of-Bag, OOB)。袋外观测值构成的集合称为袋外示例。

可以将袋外示例作为对应训练集生成的树的测试集来评估训练的结果，即可以用所有将第 i 个观测值作为 OOB 的树来预测第 i 个观测值的响应值。我们可以对这些预测响应值求平均（回归情况下）或执行多数投票（分类情况下)，已得到第 i 个观测值的一个 OOB 预测。用这种方法可以求出每个观测值的 OOB 预测，根据这些就可以计算总体的 OOB 均方误差（对回归问题）或分类误差（对分类问题)。由此得到的 OOB 误差是对 Bagging 模型测试误差的有效估计。我们将应用 OOB 误差来评估随机森林变量的重要性。

随机森林的基本思想是：为了降低单棵树的缺陷及各次抽样间的相关性，随机森林采用有放回抽样，每次抽取 n 个样本，再无放回随机抽取 p 个属性 (自变量) 中的 k（一般取 k 大约为 $\sqrt{p}$）个属性，把这 k 个属性当成新的特征，并结合因变量和抽取的 n 个样本，生成一棵回归树或者分类树，重复这一过程 M 次得到 M 棵回归树或分类树，随机森林是通过集成上述 M 棵回归树或分类树而成。通过集成 M 棵树可以有效避免单棵树的不稳定性，而每一棵树只用 k 个属性代替 p 个属性来建模，不仅可以有效降低树之间的相关性，还能提高计算速度和节省计算机内存。

随机森林算法的步骤如下：

(1) 从数据集 $(\boldsymbol{X}_1, Y_1), \cdots, (\boldsymbol{X}_n, Y_n)$ 中进行 Bootstrap 抽样（有放回抽样)，抽取 n 个样本，得到样本集 N_m；

(2) 利用 N_m 建立一棵决策树，对于树上的每个节点，重复以下步骤，直到节点的样本数达到指定的最小限定值 $n_{\min}$ ：

a）从全部 p 个随机变量中随机取 $k(k < p)$ 个；

b）从这 k 个变量中选取最优分裂变量，将此节点分裂成两个子节点。

注：对于分类问题，构造每棵树时默认使用 $k=\sqrt{p}$ 个随机变量，节点最小样本数为 1；对于回归问题，构造每棵树时默认使用 $k=\frac{p}{3}$ 个随机变量，节点最小样本数为 5 。

（3）重复以上过程 M 次，得到 M 棵树构成一个随机森林。

（4）当对新样本进行预测时，由每个决策树得到一个预测结果，再进行平均或“投票”得出最后的结果。a）对于回归问题，最后的预测结果为所有决策树预测值的平均数；b）对于分类问题，最终的预测结果为所有决策树预测结果中最多的那类，即采用“投票”得出最后的分类结果。

从随机森林的基本思想和算法可以看出，随机森林具有以下的特点：

（1）与其他的集成学习如 Bagging 相比，由于每次只选取 k（$k<p$）个预测，能够有效的降低树间的相关性，从而能最大程度的减少预测方差，提高预测的精度；

（2）由于每次只用到 k 个自变量，因此能有效节省计算时间和计算机内存；

(3) 与 Bagging 相比，随机森林最大的不同就在于自变量子集的规模 k 。若取 $k=p$ 建立随机森林，则等同于建立 Bagging 树。因此，Bagging 是随机森林的特例。

变量重要性评估：与所有集成学习方法一样，随机森林很难得到自变量（特征）与因变量间的一个直接的显式表达关系。因而，很难评估自变量的重要性。考虑到随机森林只用到了部分自变量，Breiman [129] 建议通过如下方式来度量某个特征 X_j 的重要性：

（1）根据未被抽取样本 OOB 计算随机森林中第 i 棵回归树的袋外误差 e_i；

（2）随机打乱训练集在变量 X_j 所在列的取值顺序，并计算新的袋外误差 e_i^j；

（3）重复步骤直至计算出所有决策树的误差变化，最后变量 X_j 预测误差的平均变化，即重要性指标：$\mathrm{V}(X_j)=\sum_{i=1}^{M}\left(e_i^j-e_i\right)^2/M$。

这里袋外误差是指我们使用针对某一棵树的袋外数据得到的预测误差的均值。因为共有 M 棵树，故而有 M 个袋外示例。由上述袋外示例会生成 M 个袋外误差 $e_i\quad i=1,2,\cdots,M$。由此可知，若特征变量 X_j 的变化引起重要性指标增加越大，精度减少得越多，则说明该变量越重要。

11.3 随机森林基本理论

在上一节我们已经详细介绍了随机森林算法的基本思想、算法及与其他集成学习相比的优缺点等，这一节我们将详细介绍随机森林相关的基本理论。类似与 6.1.1 和 6.1.2 节，我们将分别介绍回归树和分类树的基本理论。

11.3.1 回归树基本理论

假设随机森林是通过树的预测 $h(\boldsymbol{X},\Theta)$ 生成的，其中，Θ 是 q 维随机向量，$h(\boldsymbol{X},\Theta)$ 是 $R^{p+q}:\to R$ 上的实值函数。不妨假设 $(\boldsymbol{X}_1,Y_1),\cdots,(\boldsymbol{X}_n,Y_n)$ 独立同分布地来自 $(\boldsymbol{X},Y)$，并定义均方误差为：$\mathrm{E}_{\boldsymbol{X},Y}(Y-h(\boldsymbol{X},\Theta))^2$。

通过对 M 棵单一回归树 $h(\boldsymbol{X},\Theta_i)$ 取平均来生成随机森林的预测，当 $M\to\infty$，我们可以得到定理11.1的结论，即随机森林预测是均方收敛的。

定理 11.1 随着随机森林中树的数目趋向无穷，则如下结论几乎处处成立：

$$\lim_{M\to\infty} E_{\boldsymbol{X},Y}\left(Y-\frac{1}{M}\sum_{i=1}^{M}h\left(\boldsymbol{X},\Theta_i\right)\right)^2\to \mathrm{E}_{\boldsymbol{X},Y}\left(Y-\mathrm{E}_{\Theta}h(\boldsymbol{X},\Theta)\right)^2$$

由定理11.1可知当树的数目 $M\to\infty$ 时，随机森林的预测误差趋向于总体预测误差。接下来我们将推导出随机森林预测误差的上界，其结果将在如下的定理11.2中给出。

定理 11.2 假设对所有 Θ，$\mathrm{E}Y=\mathrm{E}_{\boldsymbol{X}}\left(\mathrm{h}(\boldsymbol{X},\Theta)\right)$，则如下结论成立：

$$\mathrm{E}_{\boldsymbol{X},Y}[\mathrm{E}_{\Theta}\left(Y-h(\boldsymbol{X},\Theta)\right)]^2\leq\rho\mathrm{E}_{\Theta}\left[\mathrm{E}_{\boldsymbol{X},Y}(Y-h(\boldsymbol{X},\Theta))^2\right]$$

其中，ρ 是 $Y-h(\boldsymbol{X},\Theta)$ 和 $Y-h(\boldsymbol{X},\Theta')$ 间的加权相关系数，Θ 和 Θ' 相互独立。

证明

$$\mathrm{E}_{\boldsymbol{X},Y}[\mathrm{E}_{\Theta}\left(Y-h(\boldsymbol{X},\Theta)\right)]^2=\mathrm{E}_{\Theta}\mathrm{E}_{\Theta'}\mathrm{E}_{\boldsymbol{X},Y}(Y-h(\boldsymbol{X},\Theta))(Y-h(\boldsymbol{X},\Theta')) \tag{11.3.1}$$

式 (11.3.1) 的右边是一个协方差并且可以写成:

$$\mathrm{E}_{\Theta}\mathrm{E}_{\Theta'}(\rho(\Theta,\Theta')\mathrm{sd}(\Theta)\mathrm{sd}(\Theta')), \tag{11.3.2}$$

其中, $\mathrm{sd}(\Theta)=\sqrt{\mathrm{E}_{\boldsymbol{X},Y}(Y-h(\boldsymbol{X},\Theta))^2}$. 加权相关系数的定义为:

$$\rho=\mathrm{E}_{\Theta}\mathrm{E}_{\Theta'}(\rho(\Theta,\Theta')\mathrm{sd}(\Theta)\mathrm{sd}(\Theta'))/(\mathrm{E}_{\Theta}\mathrm{sd}(\Theta))^2. \tag{11.3.3}$$

因此,

$$\begin{aligned}\mathrm{E}_{\boldsymbol{X},Y}[\mathrm{E}_{\Theta}\left(Y-h(\boldsymbol{X},\Theta)\right)]^2&=\rho(\mathrm{E}_{\Theta}\mathrm{sd}(\Theta))^2\\&\leqslant\rho\mathrm{E}_{\Theta}[\mathrm{E}_{\boldsymbol{X},Y}(Y-h(\boldsymbol{X},\Theta))^2].\end{aligned} \tag{11.3.4}$$

证毕。 □

由定理11.2可以看出随机森林预测的准确性取决于单棵树的预测能力及树之间相关性的强弱。

11.3.2 分类树基本理论

假设随机森林是树型分类器 $\{h(\boldsymbol{X},\Theta_i),i=1,\cdots,M\}$ 的集合，其中，$\boldsymbol{X}$ 是预测向量；Θ_i 是独立同分布的随机向量，决定了单棵分类树的生成过程；元分类器 $h\left(\boldsymbol{X},\Theta_i\right)$ 是用 CART [67] 算法构建的无剪枝的分类决策树。则当 M 趋向无穷时，对分类树也是收敛的，即有如下结论：

定理 11.3 随着随机森林中树的数目趋向无穷，则如下结论成立：

$$\lim_{M\to\infty}\Pr_{\boldsymbol{X},Y}\left(\left[\frac{1}{M}\sum_{i=1}^{M}I\left(h\left(\boldsymbol{X},\Theta_i\right)=Y\right)-\max_{j\neq Y}\frac{1}{M}\sum_{i=1}^{M}I\left(h\left(\boldsymbol{X},\Theta_i\right)=j\right)\right]<0\right)$$

$$\to\Pr_{\boldsymbol{X},Y}\left(\left[\Pr_{\Theta}(h(\boldsymbol{X},\Theta)=Y)-\max_{j\neq Y}\Pr_{\Theta}(h(\boldsymbol{X},\Theta)=j)\right]<0\right),$$

其中，$I(\cdot)$ 是示性函数。

证明 容易证明，参数空间 $\Theta_1,\Theta_2,\ldots,\Theta_M$ 上存在一个零概率集合 C，在 C 之外，对于所有的 $\boldsymbol{X}$，有下式成立

$$\frac{1}{M}\sum_{i=1}^{M}I\left(h\left(\boldsymbol{X},\Theta_i\right)=g\right)\to\Pr_{\Theta}(h(\boldsymbol{X},\Theta)=g)$$

在一个固定的训练集和参数空间 Θ 上，所有满足 $h(\Theta,\boldsymbol{X})=g$ 的 $\boldsymbol{X}$ 构成的集合是一个超矩形单元。对于所有 $h(\Theta,\boldsymbol{X})$ 只有有限的 K 个这种超矩阵单元，记作 $S_1,\ldots,S_K$。若 $\{\boldsymbol{X}:h(\Theta,\boldsymbol{X})=g\}=S_g$，此时则定义 $\phi(\Theta)=g$，并令 N_g 为前 N 次试验中 $\phi(\Theta_n)=g$ 的次数。那么有

$$\frac{1}{M}\sum_{i=1}^{M}I\left(h\left(\boldsymbol{X},\Theta_i\right)=g\right)=\frac{1}{M}\sum_{g=1}^{K}N_gI(\boldsymbol{X}\in S_g)$$

再由大数定理可得

$$N_g=\frac{1}{M}\sum_{i=1}^{M}I(\phi(\Theta_i)=g)$$

会收敛到 $\Pr_{\Theta}(\phi(\Theta)=g)$。对于 g 的某个值，所有集合的并集都不会发生收敛，得到一个概率为零的集合 C，因此在 C 之外有

$$\frac{1}{M}\sum_{i=1}^{M}I\left(h\left(\boldsymbol{X},\Theta_i\right)=g\right)\to\sum_{g=1}^{K}\Pr_{\Theta}(\phi(\Theta)=g)I(\boldsymbol{X}\in S_g).$$

上式右边即是 $\Pr_{\Theta}(h(\boldsymbol{X},\Theta)=g)$。这样我们就证明了定理。 □

由定理11.3可知，随着随机森林中树的数量的增加，模型的分类误差上限趋于一个固定值。即随机森林不会随着分类树数目的增加而产生过度拟合的问题，将对未知实例预测提供较好的参考思路和应用性。类似定理11.2，我们将给出随机森林分类误差的一个上界，为叙述方便，我们先给出如下定义：

给定一组分类器 $h(\boldsymbol{X},\Theta_1),h(\boldsymbol{X},\Theta_2),\ldots,h(\boldsymbol{X},\Theta_M)$，并使用从随机向量 $(\boldsymbol{X},Y)$ 的分布中随机抽取的训练集，将边际函数定义为

$$\mathrm{mg}(\boldsymbol{X},Y)=\frac{1}{M}\sum_{i=1}^{M}I((h\left(\boldsymbol{X},\Theta_i\right))=Y)-\max_{j\neq Y}\frac{1}{M}\sum_{i=1}^{M}I((h\left(\boldsymbol{X},\Theta_i\right)=j)),$$

其中，$I(\cdot)$ 是指示函数。

边际函数衡量的是正确分类在 $(\boldsymbol{X},Y)$ 的平均投票数超过任何其他分类的平均投票数的程度。差距越大，对分类的信心就越大。一般泛化误差由下式给出：

$$\mathrm{PE}^{*}=\mathrm{Pr}_{\boldsymbol{X},Y}(\mathrm{mg}(\boldsymbol{X},Y)<0)$$

随机森林的边际函数为

$$\mathrm{mg}(\boldsymbol{X},Y)=\mathrm{Pr}_{\Theta}(h(\boldsymbol{X},\Theta)=Y)-\max_{j\neq Y}\mathrm{Pr}_{\Theta}(h(\boldsymbol{X},\Theta)=j)$$

分类器集 $h(\boldsymbol{X},\Theta)$ 的强度定义为：

$$\mathrm{s}=\mathrm{E}_{\boldsymbol{X},Y}\mathrm{mg}(\boldsymbol{X},Y).$$

不妨设 $\mathrm{s}\geq 0$，由切比雪夫不等式可得下式成立：

$$\mathrm{PE}^{*}\leq \mathrm{var}(\mathrm{mg}(\boldsymbol{X},Y))/\mathrm{s}^{2} \tag{11.3.5}$$

$\mathrm{mg}(\boldsymbol{X},Y)$ 的方差的一个显式的表达为：

$$\widehat{\jmath}(\boldsymbol{X},Y)=\arg\max_{j\neq Y}\mathrm{Pr}_{\Theta}(h(\boldsymbol{X},\Theta)=j)$$

故

$$\begin{aligned}\mathrm{mg}(\boldsymbol{X},Y)&=\mathrm{Pr}_{\Theta}(h(\boldsymbol{X},\Theta)=Y)-\mathrm{Pr}_{\Theta}(h(\boldsymbol{X},\Theta)=\widehat{\jmath}(\boldsymbol{X},Y))\\&=E_{\Theta}[I(h(\boldsymbol{X},\Theta)=Y)-I(h(\boldsymbol{X},\Theta)=\widehat{\jmath}(\boldsymbol{X},Y))]\end{aligned}$$

原始边际函数定义为：

$$\mathrm{rmg}(\Theta,\boldsymbol{X},Y)=I(h(\boldsymbol{X},\Theta)=Y)-I(h(\boldsymbol{X},\Theta)=\widehat{\jmath}(\boldsymbol{X},Y))$$

因此，$\mathrm{mg}(\boldsymbol{X},Y)$ 是 $\mathrm{rmg}(\Theta,\boldsymbol{X},Y)$ 关于 Θ 的期望。对于任意函数 f，当 Θ，Θ' 独立同分布时，有下式成立：

$$[E_{\Theta}f(\Theta)]^{2}=E_{\Theta,\Theta'}f(\Theta)f(\Theta')$$

即有

$$\mathrm{mg}(\boldsymbol{X},Y)^2 = E_{\Theta,\Theta'}\mathrm{rmg}(\Theta,\boldsymbol{X},Y)\mathrm{rmg}(\Theta',\boldsymbol{X},Y) \tag{11.3.6}$$

再由式 (11.3.2) 可得

$$\begin{aligned}\mathrm{var}(\mathrm{mg}(\boldsymbol{X},Y)) &= \mathrm{E}_{\Theta,\Theta'}(\mathrm{cov}_{\boldsymbol{X},Y}(\mathrm{rmg}(\Theta,\boldsymbol{X},Y)\mathrm{rmg}(\Theta',\boldsymbol{X},Y)))\\ &= \mathrm{E}_{\Theta,\Theta'}(\rho(\Theta,\Theta')\mathrm{sd}(\Theta)\mathrm{sd}(\Theta'))\end{aligned}$$

其中，$\rho(\Theta,\Theta')$ 是 $\mathrm{rmg}(\Theta,\boldsymbol{X},Y)$ 和 $\mathrm{rmg}(\Theta',\boldsymbol{X},Y)$ 间的相关系数，$\mathrm{sd}(\Theta)$ 是 $\mathrm{rmg}(\Theta,\boldsymbol{X},Y)$ 的标准差。那么，

$$\begin{aligned}\mathrm{var}(\mathrm{mg}(\boldsymbol{X},Y)) &= \rho(\mathrm{E}_{\Theta}\mathrm{sd}(\Theta))^2\\ &\leq \rho\mathrm{E}_{\Theta}\mathrm{var}(\Theta)\end{aligned} \tag{11.3.7}$$

其中，ρ 是相关系数的平均值，即

$$\rho = \mathrm{E}_{\Theta,\Theta'}(\rho(\Theta,\Theta')\mathrm{sd}(\Theta)\mathrm{sd}(\Theta'))/\mathrm{E}_{\Theta,\Theta'}(\mathrm{sd}(\Theta)\mathrm{sd}(\Theta'))$$

$$\begin{aligned}\mathrm{E}_{\Theta}\mathrm{var}(\Theta) &\leq \mathrm{E}_{\Theta}(\mathrm{E}_{\boldsymbol{X},Y}\mathrm{rmg}(\Theta,\boldsymbol{X},Y))^2 - \mathrm{s}^2\\ &\leq 1-\mathrm{s}^2\end{aligned} \tag{11.3.8}$$

由式 (11.3.7) 及 (11.3.8) 可得到随机森林分类误差的上界。

定理 11.4 随机森林泛化误差的上界由下式给出

$$\mathrm{PE}^* \leq \rho(1-s^2)/s^2$$

由定理11.4 可知，分类错误的上界受随机森林中单个分类器强弱及分类间的相关性影响。

11.4 随机森林实践

11.4.1 R 语言实践

本文将应用随机森林来预测网络文章信息传播热度和文章热度的分类问题。R 语言中常用的随机森林工具包是 randomForest 包，这个包在 R 语言的核心安装包中自带，所以我们不需要手动安装它。以下是使用 iris 数据集进行随机森林分类的 R 语言代码：

```
# 加载iris数据集
data(iris)
# 将字符型变量转换为因子型变量
iris$Species <- factor(iris$Species)
# 拆分数据集为训练集和测试集
set.seed(123)
train_index <- sample(nrow(iris), nrow(iris) * 0.7)
train_data <- iris[train_index, ]
test_data <- iris[-train_index, ]
# 使用随机森林拟合训练集数据
library(randomForest)
rf_model <- randomForest(Species ~ ., data = train_data, ntree = 500, mtry = 2,
                                                    importance = TRUE)
# 使用测试集数据评估模型的性能
rf_pred <- predict(rf_model, test_data[, -5])
rf_accuracy <- mean(rf_pred == test_data$Species)
print(paste0("Random Forest Accuracy: ", rf_accuracy))
# 特征重要性评估
varImpPlot(rf_model)
```

11.4.2 Python 语言实践

在 Python 的编辑器中，我们可以通过导入 sklearn 模块中的 RandomForestClassifier 和 RandomForestRegressor 来创建随机森林实例，其中，RandomForestClassifier 用于分类任务，RandomForestRegressor 用于回归任务。

回归树实现

```
# 导入需要的库和数据集
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# 加载数据集
boston = load_boston()
X = boston.data
y = boston.target
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
                                                    =42)
# 创建随机森林回归模型
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
# 训练模型
rf_model.fit(X_train, y_train)
# 预测测试集结果
y_pred = rf_model.predict(X_test)
```

```
# 评估模型性能
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

在这个例子中，我们使用随机森林算法对波士顿房价数据集进行回归分析，均方误差为 9.35，意味着随机森林算法在测试集上的平均预测误差的平方为 9.35，即每个房价预测结果的平均偏差为约 3.05 美元的平方。总的来说，均方误差越小，代表模型的预测精度越高。但需要注意的是，MSE 的大小也受到数据本身的影响，因此 MSE 的大小并不能完全反映模型的好坏。

特征重要性

```
import matplotlib.pyplot as plt
# 计算特征重要性
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
# 输出特征重要性
print("Feature ranking:")
for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
# 绘制柱状图
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices], color="r", align="center")
plt.xticks(range(X_train.shape[1]), boston.feature_names[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
plt.show()
```

图 6.1 展示了每个特征对预测目标的贡献度，可以帮助我们了解哪些特征是最重要的。在本例中，我们可以看到，房间数 RM 和低收入人群比例 LSTAT 这两个特征的重要性最高，说明它们对于房价的预测起到了最大的作用。

分类树实现

以下是使用 Scikit-learn 自带的红酒质量数据集进行随机森林分类的 Python 代码：

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# 加载红酒质量数据集
wine = load_wine()
# 将数据集转换为Pandas DataFrame
data = pd.DataFrame(wine.data, columns=wine.feature_names)
data['target'] = wine.target
```

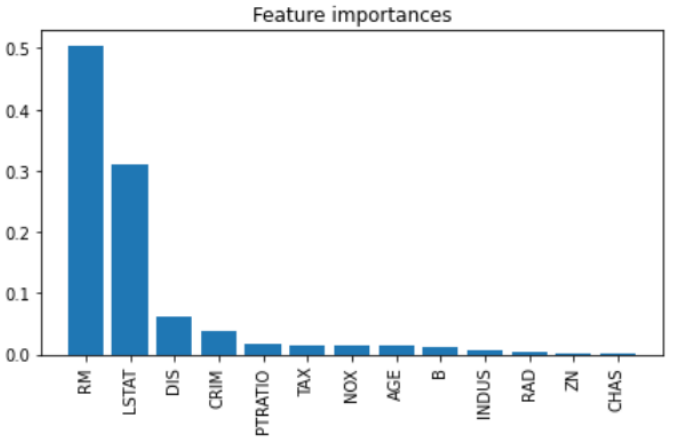


图 11.1 特征重要性柱状图

```
# 将数据集拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.iloc[:, -
                                                1], test_size=0.3, random_state=42)
# 使用随机森林分类器拟合训练集数据
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# 在测试集上进行预测
y_pred = rf.predict(X_test)
# 使用测试集数据评估模型的性能
accuracy = rf_model.score(X_test, y_test)
print(f"Accuracy: {accuracy}")
# 特征重要性评估
feature_importances = pd.DataFrame(rf_model.feature_importances_, index=X_train.
                                                columns, columns=['importance']).
                                                sort_values('importance', ascending=False)
# 柱状图可视化特征重要性
plt.figure(figsize=(8, 6))
plt.title("Feature Importance")
plt.bar(feature_importances.index, feature_importances['importance'])
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

总结

这里介绍的随机森林是专门基于树的结构提出的集成学习方法，类似于集成方法 Bagging（Bootsrap AGGregatING）[134] 和 Boosting，随机森林的思想也可以推广到其他的统计模型或弱分类器中。Bagging 是利用 bootstrap 抽样来建立单棵树，但是每次都是利用所有 p 个预测来建立单棵树。因此，与 Bagging 相比，随机森林具有两个明显的优势，第一，随机森林只用 k 个特征，可以节省计算量和计算机内存；第二，当预测中存在高度相关的自变量时，在减少树之间的相关性方面，随机森

林具有明显的优势。关于集成学习 Bagging 和 Boosting 更多的介绍，有兴趣的读者可以参考 [71] 和 [70] 两本书。

11.5 习题

1、证明定理11.1。

2、集成学习是否可以避免过拟合?

3、随机森林的不同基决策树模型差异体现在那些地方?

4、随机森林和 Bangging 类方法有什么异同?

5、随机森林的投票方式是怎么样的?

6、编写程序实现表**??**的内容，并画出各方法的 ROC 曲线。

7、用 Logistic 回归代替随机森林中树的分类方法，并与本章所介绍方法进行比较，并讨论各自方法的优缺点。

8、模拟或者找一个实际数据 Y 是连续随机变量，$\boldsymbol{X}$ 是随机向量，分别用 R 软件包 rpart，party 和 RWeka 等运行这一数据并讨论各算法的优缺点。

9、模拟或者找一个实际数据 Y 是连续随机变量，$\boldsymbol{X}$ 是随机向量，对这一数据比较 Bagging、Boosting 和随机森林等集成方法的表现，并总结各自方法的优缺点。

第十二章　Boosting 方法

Boosting 是另一种集成学习方法，它通过迭代训练一系列弱学习器，每个学习器都试图纠正前一个学习器的错误。在每一轮中，对之前错误分类的样本增加权重，使其在下一轮中更受关注。与随机森林选择随机特征和直接投票不同，Boosting 算法使用所有特征进行训练，对预测结果进行加权投票。

12.1　简介

如何根据历史数据，建立精确的预测模型，是机器学习方法研究的主要目的之一。例如，我们拟构建一个可以区分垃圾邮件和正常邮件的电子邮件过滤器。通常机器学习方法解决此问题的思路如下：首先收集尽可能多的垃圾电子邮件和非垃圾电子邮件的案例。然后，应用已有的机器学习算法对收集到的邮件和其对应的标签进行训练，建立一个分类准则，基于已知数据建立的准则可以对新的、未加标签的电子邮件进行自动的分类。一个自然的期望是建立的分类（预测）准则能够对新的电子邮件做出精确的预测。

通常，建立一个高度准确的预测或分类准则往往是困难的。但是，建立系列相对准确、比随机猜测稍好的经验法则就容易实现的多。对上例的电子邮件问题，可以建立这样一个简单规则：如果电子邮件中出现“立即购买”的短语，那么就预测它是垃圾邮件。一方面，此规则并不能涵盖所有垃圾邮件；另一方面，这一规则显然要好于随机猜测。对于任何一个遭受过垃圾邮件骚扰的人来说，一些识别垃圾邮件的规则会迅速在我们脑海中闪现。例如，如果邮件里有“购买”、“Viagra”这种词，那么该邮件很可能就是垃圾邮件。但是，根据个人经验总结的规则对于垃圾邮件的正确区分是远远不够的。一方面，例如把出现“购买”的邮件都划分为垃圾邮件，其他邮件都视作正常邮件，那么也会出错。但是另一方面，这些规则也并非毫无用处，至少给我们提供了有价值的信息。虽然其准确率较低，但至少比随机猜测的效果要好。另外，发现这些“弱”的准则也相对容易。

Boosting 方法正是基于这样一种考虑：找到许多粗糙的、比随机猜测稍好的经验准则要比找到单一的、高度精确的预测规则容易得多。这些粗糙的、比随机猜测稍好的经验准则或算法，通常被称为弱学习算法或者基学习算法。

Boosting 算法反复调用这些弱学习算法，每次让其学习训练样本的不同子集。更准确地说，是训练数据的不同分布或权重。每次调用时，基学习算法都会生成一个

新的弱学习算法（学习器）。重复建立多个弱学习算法后，Boosting 算法最终将这些弱学习算法（学习器）有效组合成最后的强学习算法（强学习器），强学习器有望比任何一个弱学习器都准确得多。

因此，Boosting 的核心思想就是：对于一个复杂的任务而言，有效的综合多个专家的预测进而所得出的新的预测，要优于其中任何一个专家的单独预测。即通常所说的“**三个臭皮匠顶个诸葛亮**”的道理。

Boosting 集成学习方法基本流程如图 12.1 所示。

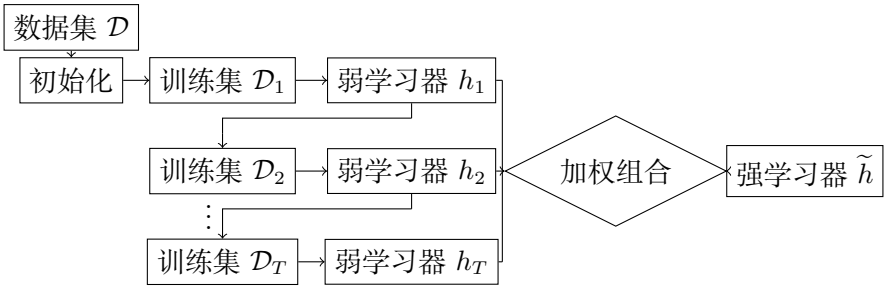


图 12.1 Boosting 集成学习方法基本流程

当然，要使上述 Boosting 方法切实可行，还有以下两个基本问题需要解决：

1、在每一轮训练中，如何改变训练数据的权值或概率分布；

2、最终，如何将所得到的弱分类器组合成为一个强学习器。

关于第 1 个问题，Boosting 算法家族中的代表性算法 AdaBoost 的处理方式是：提高那些被前一轮弱学习器错误分类样本的权值，而降低那些被正确分类样本的权值。这样一来，那些没有得到正确分类的数据，由于其权值的加大而受到后一轮弱学习器的更大关注。于是得到的系列弱学习器之间具有“互补”的特点。至于第 2 个问题，AdaBoost 算法通常采取加权投票的方法。具体地，增加分类误差率小的弱学习器的权值，使其在表决中起较大作用；减少分类误差率大的弱学习器的权值，使其在表决中起较小的作用。

12.1.1 Boosting 方法起源

Boosting 方法起源于一个纯理论性的公开问题，即所谓的 **Boosting 公开问题**。

1984 年，Valiant [135] 在研究 PAC （probably approximately correct）学习框架时，给出了强可学习（或称可学习）和弱可学习的概念。1988 年，Kearns 和 Valiant [136] 在研究 PAC 学习模型时，针对以上弱可学习与强可学习提出如下公开问题：是否可以将性能仅略好于随机猜测的弱学习算法“Boosting”提升为为具有任意精确度的强学习算法？即 Boosting 公开问题。此问题非常重要，因为获得一个弱学习器要比获得一个强学习器要容易的多。如果该问题的答案是肯定的，那么任何

弱学习器都有可能被提升为强学习器。这一思想对今后机器学习算法尤其是集成学习的发展产生了深远的影响。

1989 年，Schapire 给出了肯定的答复，在 PAC 框架下，强可学习与弱可学习是等价的，即: 一个概念是强可学习的充要条件是这个概念是弱可学习的。同时，他在文章中给出的构造性证明也成为最早的 Boosting 算法 [137]。随后，在 1990 年，Freund 开发了一个效率更高的且具有某种最优性的 Boosting 算法 [138] 。Drucker 等人利用这些早期的 Boosting 方法在 OCR （Optical Character Recognition）任务上进行了首次实验验证 [139]。然而，由于上述算法需要提前知晓弱学习器的错误率上界，这通常在实际应用中是未知的。因此上述 Boosting 算法并不具备实际应用性。

Freund 与 Schapire 在随后的研究中发现，“Online” 学习与 Boosting 问题之间存在着极大的共性。他们将其与加权投票的相关研究成果进行融合，并在 Boosting 问题中进行相应推广，得到了著名的 AdaBoost 算法。特别地，此算法不需要提前预知弱学习器的分类精度等相关的任何先验知识，在实践中获得了极大成功。凭借此工作，Freund 和 Schapire 获得了 2003 年度理论计算机的最高奖——哥德尔奖（Godel Prize）。AdaBoost 一举成为最具影响力的集成算法之一，被评为数据挖掘十大算法之一 [140]。

12.1.2 AdaBoost 算法

AdaBoost 通用算法

AdaBoost 算法可以做出非常精准的预测，其过程却非常简单。现举例如下：

以二元问题为例，设 $\mathcal{X}$ 为自变量组成的样本空间，其中的样本都是从分布 $\mathcal{D}$ 中随机抽取，且满足独立同分布性；其学习目标函数记为 Y。设 $\mathcal{X}$ 由 $\mathcal{X}_1$、$\mathcal{X}_2$ 和 $\mathcal{X}_3$ 三部分组成，每个部分占 1/3，通过随机猜测工作的学习器在这个问题上有 0.5 的分类误差。我们想在这个问题上得到一个精确的（例如零误差）学习器。可借助的只有一个弱学习器 h_1，它在样本空间 $\mathcal{X}_1$ 和 $\mathcal{X}_2$ 中有正确的分类，在 $\mathcal{X}_3$ 有错误的分类。如何将此弱学习器 h_1 “Boosting” 为强学习器呢?

一个自然的想法就是纠正 h_1 所犯的错误。首先，通过 $\mathcal{D}$ 派生出新的分布 $\mathcal{D}_1$。例如通过提高那些被 h_1 错误学习样本的权值，降低那些被 h_1 正确预测的样本的权值诱导出新分布 $\mathcal{D}_1$。显然在 $\mathcal{D}_1$ 上，h_1 的错误被彰显。然后用 $\mathcal{D}_1$ 训练得到学习器 h_2。此时得到的学习器 h_2 极有可能也是一个弱学习器。假设它在 $\mathcal{X}_1$ 和 $\mathcal{X}_3$ 中有正确的预测，在 $\mathcal{X}_2$ 有错误的预测。通过以某种适当的方式组合 h_1 和 h_2，组合的学习器将在 $\mathcal{X}_1$ 中具有正确的预测，并且可能在 $\mathcal{X}_2$ 和 $\mathcal{X}_3$ 中仍有错误。类似的，为了使组合学习器的错误彰显，我们再次派生出一个新的分布 $\mathcal{D}_2$，并从 $\mathcal{D}_2$ 训练出新的学习器 h_3，使 h_3 在 $\mathcal{X}_2$ 和 $\mathcal{X}_3$ 有正确的预测。最后，通过组合 h_1、h_2 和 h_3，就得到一个强学习器，因为在 $\mathcal{X}_1$、$\mathcal{X}_2$ 和 $\mathcal{X}_3$ 的每个空间中，至少有两个学习器是正确的。

简而言之，Boosting 方法就是顺序训练一族弱学习器，并将它们组合以形成强学习器来进行预测。训练过程中，让后建立的学习器更多地关注前序学习器的错误预测样本。通用 Boosting 算法如下：

通用 Boosting 算法

输入：样本权值分布 $\mathcal{D}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程：

1. $\mathcal{D}_1 = \mathcal{D}$; % 初始化分布
2. **for** $t = 1, ..., T$:
3. h_t=$\mathcal{L}(\mathcal{D}_t)$; % 依分布训练弱学习器
4. $\mathrm{err}_{\mathcal{D}_t} = \Pr_{\mathcal{D}_t}(h_t(\boldsymbol{X}) \neq Y)$; % 度量误差
5. $\mathcal{D}_{t+1}$= Adjust_Distribution$(\mathcal{D}_t, \mathrm{err}_{\mathcal{D}_t})$; % 更新分布
6. **end**

输出：$\mathcal{H}(\boldsymbol{X}) = \text{Combine\_Outputs}\left(\{h_1(\boldsymbol{X}), \cdots, h_T(\boldsymbol{X})\}\right)$

AdaBoost 算法

输入：训练数据集 $\{(\boldsymbol{X}_1, Y_1), (\boldsymbol{X}_2, Y_2), \cdots, (\boldsymbol{X}_n, Y_n)\}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程：

1. $\mathcal{D}_1(\boldsymbol{X}_i) = 1/n$; % 初始化权重分布
2. **for** $t = 1, ..., T$:
3. $h_t(\boldsymbol{X}) = \mathcal{L}(\mathcal{D}, \mathcal{D}_t)$, 且 $h_t(\boldsymbol{X}) : \mathcal{X} \to \{-1, 1\}$; % 依分布训练输出弱学习器
4. $\mathrm{err}_{\mathcal{D}_t} = \sum_{i=1}^{n} \Pr_{\boldsymbol{X}_i \sim \mathcal{D}_t}(h_t(\boldsymbol{X}_i) \neq Y_i)$; % 度量误差
5. **if** $\mathrm{err}_{\mathcal{D}_t} \geq 0.5$ **then break**
6. $\alpha_t = \dfrac{1}{2} \ln \left(\dfrac{1 - \mathrm{err}_{\mathcal{D}_t}}{\mathrm{err}_{\mathcal{D}_t}} \right)$; % 计算弱学习器 h_t 的权重
7. $\mathcal{D}_{t+1}(\boldsymbol{X}_i) = \dfrac{\mathcal{D}_t(\boldsymbol{X}_i)}{Z_t} \exp(-\alpha_t Y_i h_t(\boldsymbol{X}_i))$, 其中
$Z_t = \sum_{i=1}^{n} \mathcal{D}_t(\boldsymbol{X}_i) \exp(-\alpha_t Y_i h_t(\boldsymbol{X}_i))$; % 更新分布, Z_t 为规范化因子
8. **end**

输出：$\mathcal{H}(\boldsymbol{X}) = \mathrm{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X}))$

AdaBoost 具体算法

对上述 Boosting 通用算法，将 Adjust_Distribution$(\cdot,\cdot)$ 和 Combine_Outputs $(,\cdots,)$ 取成不同的形式，则会衍生出各种类型的 Boosting 算法，如 AdaBoost.M$_1$、AdaBoost.MR、FilterBoost、GentleBoost、GradientBoost、LogitBoost 等，进而形成庞大的 Boosting 算法族。我们下面重点介绍 AdaBoost 算法。

假设 $\mathcal{X}\subset\mathbf{R}^{n\times p}$ 是自变量生成的样本空间，$\mathcal{Y}$ 是标签集，通常假设 $\mathcal{Y}=\{-1,1\}$。给定一个二元训练数据集 $\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$ 每个训练数据由自变量样本与标签组成，其中 $\boldsymbol{X}_i\in\mathcal{X}$ 为自变量样本空间，$Y_i\in\mathcal{Y}$ 为标签，$i=1,2,\cdots,n$。AdaBoost 算法如上。

注 12.1 上述算法通常被称为 “离散型 AdaBoost”，因为弱学习器 $h_t(\boldsymbol{X})$ 返回离散的标签。如果弱学习器返回的预测值为连续实数，则可以对 AdaBoost 做恰当的修改，参见文献 [141]。此外，上述算法中的关键量 h_t 和 α_t 的选取原理将在下一章给出具体的解释。

注 12.2 对上述 AdaBoost 算法做一些简单说明：

A. 为保证第 1 步能够输出基本学习器 $h_1(\boldsymbol{X})$，我们需要假设原始训练数据具有已知的权值分布。为简单起见，通常假定训练数据具有均匀分布。

B. 根据已有的符号表示，可以将训练数据权值的分布写成如下形式：

$$\mathcal{D}_t=(w_1^{(t)},w_2^{(t)},\cdots,w_i^{(t)},\cdots,w_n^{(t)}),\quad t=1,2,\cdots,T, \tag{12.1.1}$$

其中，$w_i^{(1)}=\dfrac{1}{n}$, $i=1,2,\cdots,n$.

C. 计算基本学习器 $h_t(\boldsymbol{X})$ 在分布为 $\mathcal{D}_t$ 的训练数据集上的误差率：

$$\begin{aligned}\mathrm{err}_{\mathcal{D}_t}&=\sum_{i=1}^{n}\Pr_{\boldsymbol{X}_i\sim\mathcal{D}_t}(h_t(\boldsymbol{X}_i)\neq Y_i)\\&=\sum_{h_t(\boldsymbol{X}_i)\neq Y_i}w_i^{(t)}\end{aligned} \tag{12.1.2}$$

此式表明弱学习器 $h_t(\boldsymbol{X})$ 在加权训练数据集上的分类误差等于被 $h_t(\boldsymbol{X})$ 误分的样本权值之和。

D. 上述算法给出弱学习器的系数表达式为

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1-\mathrm{err}_{\mathcal{D}_t}}{\mathrm{err}_{\mathcal{D}_t}}\right) \tag{12.1.3}$$

显然当 $\mathrm{err}_{\mathcal{D}_t}\leq 0.5$ 时，其系数 $\alpha_t\geq 0$，且 α_t 为 $\mathrm{err}_{\mathcal{D}_t}$ 的减函数。所以误差率 $\mathrm{err}_{\mathcal{D}_t}$ 越小的学习器其权重 α_t 就越大。

E. 根据算法中的第 7 个式子以及式 (12.1.1) 可知，更新权值分布可以写成：

$$\begin{aligned} w_i^{(t+1)} &= \frac{w_i^{(t)}}{Z_t}\exp(-\alpha_t Y_i h_t(\boldsymbol{X}_i)) \\ &= \begin{cases} \dfrac{w_i^{(t)}}{Z_t}e^{-\alpha_t}, & h_t(\boldsymbol{X}_i) = Y_i \\ \dfrac{w_i^{(t)}}{Z_t}e^{\alpha_t}, & h_t(\boldsymbol{X}_i) \neq Y_i \end{cases} \end{aligned} \tag{12.1.4}$$

其中

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} \mathcal{D}_t(\boldsymbol{X}_i)\exp(-\alpha_t Y_i h_t(\boldsymbol{X}_i)) \\ &= \sum_{i=1}^{n} w_i^{(t)}\exp(-\alpha_t Y_i h_t(\boldsymbol{X}_i)) \\ &= \sum_{Y_i = h_t(\boldsymbol{X}_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{Y_i \neq h_t(\boldsymbol{X}_i)} w_i^{(t)} e^{\alpha_t} \end{aligned} \tag{12.1.5}$$

式 (12.1.4) 说明，如果样本 $\boldsymbol{X}_i$ 被正确学习，即 $h_t(\boldsymbol{X}_i) = Y_i$，则其权重会减小；如果 $\boldsymbol{X}_i$ 被错误学习，即 $h_t(\boldsymbol{X}_i) \neq Y_i$，则其权重会增大。这样，误分学习样本在下一轮学习中就会受到基学习器的更大关注而起到更大作用。

F. 记

$$\widetilde{h}_T(\boldsymbol{X}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X}) \tag{12.1.6}$$

则 AdaBoost 的最后输出结果为

$$\mathcal{H}(\boldsymbol{X}) = \operatorname{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X})) \tag{12.1.7}$$

此式表明对于任何数据的分类采用加权投票表决的方法，其中 $\mathcal{H}(\boldsymbol{X})$ 的符号决定数据 $\boldsymbol{X}$ 的类。需要注意的是，基本学习器 $h_t(\boldsymbol{X})$ 的系数 α_t 未必具有归一性。

G. AdBoost 具备以下两个特点：

(a) AdBoost 不更新训练数据，只是不断更新训练数据的权值分布，使得训练数据在基本分类器的学习中起不同的作用。

(b) AdBoost 最终分类器 $\widetilde{h}_T(\boldsymbol{X})$ 的构成是利用基本分类器 h_t，$t = 1, 2, \cdots, T$ 的线性组合构建，因而可视其为加法模型。

12.1.3 AdaBoost 实例

例 12.1 给定如表 12.1 所示训练数据。假设弱学习器(或称阈值函数)由 $X < v$ 或 $X > v$ 产生,其阈值 v 使该学习器在训练数据集上分类误差率最低。试用 AdBoost 算法学习一个强学习器。

表 12.1 训练数据表

序号	1	2	3	4	5	6	7	8	9	10
X	1	2	3	4	5	6	7	8	9	10
Y	1	1	1	−1	1	−1	−1	1	−1	−1

解: 初始化数据权值分布

$$\mathcal{D}_1 = (w_1^{(1)}\ w_2^{(1)}, \cdots, w_n^{(1)})$$

$$w_i^{(1)} = 0.1, \quad i = 1, 2, \cdots, 10$$

1、对 $t = 1$:

(1) 在权值分布为 $\mathcal{D}_1$ 的训练数据上,阈值 v 取 3.5 时分类误差率最低,故基本分类器为

$$h_1(X) = \begin{cases} 1, & X < 3.5 \\ -1, & X \geq 3.5 \end{cases}$$

(2) $h_1(X)$ 在训练数据集上的误差率: $\mathrm{err}_{\mathcal{D}_1} = \sum_{i=1}^{n} \mathrm{Pr}_{X_i \sim \mathcal{D}_t}(h_1(X_i) \neq Y_i) = 0.2$

(3) 计算 $h_1(X)$ 的系数: $\alpha_1 = \frac{1}{2} \ln \frac{1 - \mathrm{err}_{\mathcal{D}_1}}{\mathrm{err}_{\mathcal{D}_1}} = 0.6931$

(4) 更新训练数据的权值分布:

$$\mathcal{D}_2 = (w_1^{(2)}, w_2^{(2)}, \cdots, w_n^{(2)})$$

$$w_i^{(2)} = \frac{w_i^{(1)}}{Z_1} \exp(-\alpha_1 Y_i h_1(X_i)), i = 1, 2, \cdots, 10$$

$$\mathcal{D}_2 = (0.0625, 0.0625, 0.0625, 0.0625, 0.2500, 0.0625,$$
$$0.0625, 0.2500, 0.0625, 0.0625)$$

$$\widetilde{h}_1(X) = 0.6931 h_1(X)$$

分类器 $\mathrm{sign}(\widetilde{h}_1(X))$ 在训练数据集上有 2 个误分类点。

2、对 $t=2$:

(1) 在权值分布为 $\mathcal{D}_2$ 的训练数据上，阈值 v 是 8.5 时分类误差率最低，基本分类器为

$$h_2(X)=\begin{cases}1, & X<8.5\\ -1, & X\geq 8.5\end{cases}$$

(2) $h_2(X)$ 在训练数据集上的误差率 $\mathrm{err}_{\mathcal{D}_2}=0.1875$

(3) 计算 $\alpha_2=0.7332$

(4) 更新训练数据权值分布:

$$\mathcal{D}_3=(0.0385,0.0385,0.0385,0.1667,0.1539,0.1667,$$
$$0.1667,0.1539,0.0385,0.0385)$$
$$\widetilde{h}_2(X)=0.6931h_1(X)+0.7332h_2(X)$$

分类器 $\mathrm{sign}(\widetilde{h}_2(X))$ 在训练数据集上有 3 个误分类点。

3、对 $t=3$:

(1) 在权值分布为 $\mathcal{D}_3$ 的训练数据上，阈值 v 是 5.5 时分类误差率最低，基本分类器为

$$h_3(X)=\begin{cases}1, & X<5.5\\ -1, & X\geq 5.5\end{cases}$$

(2) $h_3(X)$ 在训练数据集上的误差率 $\mathrm{err}_{\mathcal{D}_3}=0.3206$

(3) 计算 $\alpha_3=0.3755$

(4) 更新训练数据权值分布:

$$\mathcal{D}_4=(0.028,0.028,0.028,0.260,0.113,0.123,$$
$$0.123,0.240,0.028,0.028)$$
$$\widetilde{h}_3(X)=0.4236h_1(X)+0.6496h_2(X)+0.3755h_3(X)$$

分类器 $\mathrm{sign}(\widetilde{h}_3(X))$ 在训练数据集上有 0 个误分类点。

于是最终分类器为:

$$\mathcal{H}(X)=\mathrm{sign}(\widetilde{h}_3(X))=\mathrm{sign}\{0.4236h_1(X)+0.6496h_2(X)+0.3755h_3(X)\}$$

12.2 AdaBoost 算法的误差分析

12.2.1 AdaBoost 算法的训练误差

AdaBoost 最基本的理论性质涉及其减少训练误差的能力，即训练集上的误差率。Freund 和 Schapire 给出了误差界 [142]：

定理 12.1 若 AdaBoost 在每一轮迭代中生成弱分类器 h_t 的误差率是 $\mathrm{err}_{\mathcal{D}_t}$，$t=1,2,\cdots,T$. 则最终分类器 $\mathcal{H}(\boldsymbol{X})$ 的训练误差率 $\mathrm{err}_{\mathcal{D}}=\sum\limits_{i=1}^{n}\mathrm{Pr}_{\boldsymbol{X}_i\sim\mathcal{D}}\left(\mathcal{H}(\boldsymbol{X}_i)\neq Y_i\right)$ 满足

$$\mathrm{err}_{\mathcal{D}}\leq 2^T\prod_{i=1}^{T}\sqrt{\mathrm{err}_{\mathcal{D}_t}(1-\mathrm{err}_{\mathcal{D}_t})}$$

此定理的证明具有一定的技巧性与特殊性，因而在后续研究中极少被采用。随后 Schapire 和 Singer 进行了更深入研究 [143]，得到如下结论：

定理 12.2 AdaBoost 算法最终分类器的训练误差满足

$$\frac{1}{n}\sum_{i=1}^{n}I(\mathcal{H}(\boldsymbol{X}_i)\neq Y_i)\leq\frac{1}{n}\sum_{i=1}^{n}\exp(-Y_i\widetilde{h}_T(\boldsymbol{X}_i))=\prod_{t=1}^{T}Z_t \tag{12.2.1}$$

其中 $I()$ 是示性函数，Z_t，$\widetilde{h}_T(\boldsymbol{X})$ 和 $\mathcal{H}(\boldsymbol{X})$ 分别见式 (12.1.5)，(12.1.6) 和 (12.1.7)。

证明 当 $\widetilde{h}_T(\boldsymbol{X}_i)\neq Y_i$ 时，$Y_i\widetilde{h}_T(\boldsymbol{X}_i)<0$，因而 $\exp(-Y_i\widetilde{h}_T(\boldsymbol{X}_i))\geq 1$ 。由此可知 (12.2.1) 的不等式成立。

下面证明 (12.2.1) 中等式是成立的。首先，由式 (12.1.4) 可知：

$$w_i^{(t)}\exp(-\alpha_tY_ih_t(\boldsymbol{X}_i))=Z_tw_i^{(t+1)}\quad i=1,2,\cdots,n.$$

根据上式得到：

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\exp(-Y_i\widetilde{h}_T(\boldsymbol{X}_i))&=\frac{1}{n}\sum_{i=1}^{n}\exp\left(-\sum_{t=1}^{T}\alpha_tY_ih_t(\boldsymbol{X}_i)\right)\\&=\sum_{i=1}^{n}w_i^{(1)}\prod_{t=1}^{T}\exp\left(-\alpha_tY_ih_t(\boldsymbol{X}_i)\right)\\&=Z_1\sum_{i=1}^{n}w_i^{(2)}\prod_{t=2}^{T}\exp\left(-\alpha_tY_ih_t(\boldsymbol{X}_i)\right)\\&=Z_1Z_2\sum_{i=1}^{n}w_i^{(3)}\prod_{t=3}^{T}\exp\left(-\alpha_tY_ih_t(\boldsymbol{X}_i)\right)\\&=\cdots\\&=Z_1Z_2\cdots Z_{T-1}\sum_{i=1}^{n}w_i^{(T)}\exp\left(-\alpha_TY_ih_T(\boldsymbol{X}_i)\right)\end{aligned}\tag{12.2.2}$$

再由式 (12.1.5) 中 Z_t 的定义可知:

$$Z_T = \sum_{i=1}^{n} w_i^{(T)} \exp\left(-\alpha_T Y_i h_T(\boldsymbol{X}_i)\right) \tag{12.2.3}$$

证毕。 □

注 12.3 1、定理12.2的左边是训练误差,右边是所有归一化因子 Z_t $(t = 1, 2, \cdots, T)$ 的乘积。通过上式可以知道，要想得到对训练样本拟合精度高的强学习器，需要选择弱学习器 $h_t(\boldsymbol{X})$ 以及其权值 α_t，使 $\prod_{t=1}^{T} Z_t$ 最小化。虽然上式给出的是相对松弛的训练误差界，但因其更好的解释性和可操作性，可得到更广泛的应用。

2、要实现 $\prod_{t=1}^{T} Z_t$ 的最小化，每加入一个新的弱学习器，都可能要修改已有弱学习器的集成方式，其复杂度太高。AdaBoost 的思想是不改变已有预测准则的形式，以线性加和的方式加入新的弱学习器，只最小化当前迭代的归一化因子 Z_t。

定理 12.3

$$\prod_{t=1}^{T} Z_t = 2^T \prod_{t=1}^{T} \sqrt{\mathrm{err}_{\mathcal{D}_t}(1 - \mathrm{err}_{\mathcal{D}_t})} = \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp(-2\sum_{t=1}^{T} \gamma_t^2) \tag{12.2.4}$$

其中 $\gamma_t = \dfrac{1}{2} - \mathrm{err}_{\mathcal{D}_t}$，称为 h_t 的 边界 (edge)。

本定理证明过程略，感兴趣的读者可参考 [144]。

推论 12.1 如果存在 $\gamma > 0$，对所有 t 有 $\gamma_t \geq \gamma$，则

$$\frac{1}{n} \sum_{i=1}^{n} I(\mathcal{H}(\boldsymbol{X}_i) \neq Y_i) \leq \exp(-2T\gamma^2) \tag{12.2.5}$$

因为 $\mathrm{err}_{\mathcal{D}_t} = \dfrac{1}{2} - \gamma_t$，说明边界 γ_t 度量第 t 个弱学习器 h_t 的误差率比随机猜测的错误率好。

此推论告诉我们这样一个事实：只要弱学习器的学习能力比随机猜测稍好 ($\gamma_t = \dfrac{1}{2} - \mathrm{err}_{\mathcal{D}_t} \geq \gamma > 0$)，那么通过 AdaBoost 输出的强学习器 $\mathcal{H}$ 的训练误差就是随着轮数 T 呈指数下降的。例如令 $\gamma = 0.1$，即所有弱学习器的误差率不超过 0.4，则上式表明强预测准则 $\mathcal{H}$ 的训练误差率至多是

$$\left(\sqrt{1 - 4 \times 0.1^2}\right)^T \approx (0.9798)^T$$

但其实 AdaBoost 算法及其分析并不需要知道这个下界 γ，而且能适应弱学习器各自的训练误差率，由此获得了自适应 Boosting 的名称，即 AdaBoost （Ada 是 Adaptive 的简写)。如果某些 γ_t 很大，那么训练误差界的减少将会更大。

此外，通过此推论可知：若想使训练误差率 $\mathrm{err}_{\mathcal{D}} \leq \epsilon$，则可让训练轮数 T 为

$$\left(\frac{1}{2\gamma^2}\ln\frac{1}{\epsilon}\right)$$

上述式子说明可以通过有限个学习器就可以使得训练误差足够的小。

12.2.2 AdaBoost 算法的泛化误差

定理 12.1，12.2 和 12.3 讨论了 AdaBoost 算法训练误差的下界。然而，在机器学习中我们真正关心的是它在测试数据上的泛化能力。事实上，任一种能够降低训练误差的算法不一定有资格作为 Boosting 算法。Boosting 算法是一种可以使泛化误差（generalization error）任意接近零的算法。直观来说，它是一种对测试数据具有接近完美的预测能力的学习算法。当然，以上结论的成立需要对算法提供合理规模的训练样本，所使用的弱学习算法也一直有比随机猜测好的弱学习器。

许多实验都表明 AdaBoost 算法在迭代次数很高时似乎并不容易出现过拟合。特别地，在训练误差率降到 0 以后，继续增大学习轮数 T，AdaBoost 的测试误差率在某种程度上仍在降低。例如，Schapire 绘制了效果曲线，如图 12.2 (a) 所示。对于 AdaBoost 算法，在最初几轮训练误差率已降至 0 后的很长一段时间内，测试误差率仍持续降低。从表面上看，随着更多子分类器的加入，集成分类器形式趋于复杂，分类精度却仍在提高，似乎违背了科学研究中的“奥卡姆剃刀”原理。因此，如何解释 AdaBoost 为什么不容易过拟合成为了 AdaBoost 算法中最迷人的理论问题，吸引了大量关注。

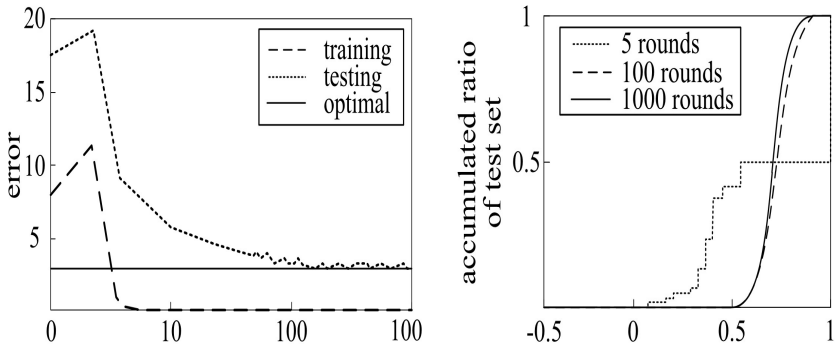
为了解释这一现象，分析 AdaBoost 算法的强泛化能力，Schapire 将统计学习中分类间隔（例如支持向量机中定义的间隔）的相关分析理论引入 AdaBoost 算法 [145]。由此，Schapire et al. 指出：AdaBoost 之所以没有过拟合，是因为在训练误差达到 0 后，强分类器随着学习轮数增加，其间隔还在增大，因而泛化误差仍在减小。这也成为了目前最流行的分析方法。

根据泛化误差依赖于最小间隔 θ 的值。我们可以最大化最小间隔来得到更紧的泛化误差。这也正是 Breiman 设计 Arc-gv 算法的主要思想。在每轮迭代中，Arc-gv 根据上述思想更新 α_t 形式如下：

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1+\gamma_t}{1-\gamma_t}\right) - \frac{1}{2}\ln\left(\frac{1+\theta_t}{1-\theta_t}\right)$$

其中 $\gamma_t = \dfrac{1}{2} - \mathrm{err}_{\mathcal{D}_t}$，$\theta_t$ 是截止当前学习轮次的组合分类器的最小间隔。

与 AdaBoost 相比，Arc-gv 能够得到更大的最小分类间隔以及更好的分类间隔分布。而且，基于间隔理论，Breiman 给出了更紧的泛化误差界 [146]。



(a) 训练误差与测试误差 (b) 间隔

图 12.2 AdaBoost 的间隔解释

按照理论分析，Arc-gv 具有更强的泛化能力，Arc-gv 应当比 AdaBoost 性能要好。然而 Breiman 在实验中发现：尽管 Arc-gv 能够得到比 AdaBoost 更大的最小间隔，但 Arc-gv 的测试误差率却总是高于 AdaBoost。因此，Breiman 对 Schapire 的间隔分析理论提出了质疑，使得 AdaBoost 的间隔分析方法受到了极大的挑战。

随后的几年，虽然有诸多学者给出了 AdaBoost 更紧致的泛化误差界，并指出其实是分类间隔分布影响着 AdaBoost 的泛化误差界。但是关于 Breiman 的质疑，都没有给出正面回答。七年后，Reyzin 和 Schapire 发现了一个有趣的现象。考虑到泛化误差界与间隔 θ、学习轮数 T 以及子学习器的复杂度相关，因此为了研究间隔的影响，需要固定另外的两个因素。Breiman 在比较 Arc-gv 和 AdaBoost 时，在实验中以 CART 为弱分类器，通过固定叶节点数目来控制弱分类器复杂度。然而，Reyzin 和 Schapire 却进一步发现，当叶节点数目相同时，Arc-gv 与 AdaBoost 生成的决策树有着很大的不同 [147]。Arc-gv 生成的树通常拥有较大的深度，而 AdaBoost 生成的树则宽度较大。一般情况下，更深的决策树由于进行了更多分裂，可能具有更大的模型复杂度。在这种情况下比较 Arc-gv 与 AdaBoost 有失公平。Reyzin 和 Schapire 重新进行了 Breiman 的实验，但使用复杂度相同的决策树作为基分类器。此时，AdaBoost 比 Arc-gv 的间隔分布要优。虽然 Reyzin 和 Schapire 指出分类间隔分布是影响算法泛化能力的关键，但并没有给出比 Breiman 更紧致地泛化误差界。

为彻底解决此类问题，需要通过分类间隔分布给出 AdaBoost 算法更紧致的泛化误差界。而且，要想正面回答 Breiman 的问题，这个泛化误差界应该比 Breiman 给出的基于最小分类间隔的泛化误差界更紧致。也就是说，是分类间隔分布，而不是最小分类间隔决定了算法的泛化能力。2008 年，王立威等构造了一种与分类间隔分布有关，与最小分类间隔几乎无关的平衡分类间隔 (Equilibrium margin，Emargin)，给出了基于平衡分类间隔更紧致的 Emargin Bounds [148]。通过实验证明：与 Arc-gv 相比，AdaBoost 有着更大的平衡分类间隔以及更低的测试误差率，实现了实验测试

与理论分析结果的一致。但是证明过程中考虑了比 Breiman 和 Schapire 模型更多的信息，因而无法直接说明平衡分类间隔比最小间隔更本质。

为更进一步解决这个问题，2013 年，周志华进行了更深入的研究 [149]。首先，他们引入了第 k 间隔界（the kth margin bound），并且研究其与最小间隔界与 Emargin 界的关系。然后，通过改进 Bernstein 界最终给出了更好的泛化误差界。总之，周志华他们的结果不仅正面回答了 Breiman 的疑问，合理解释了 AdaBoost 不会过拟合的原因，而且进一步巩固了间隔分析理论知识结构。

12.3 AdaBoost 算法原理探析

AdaBoost 算法最初的方法融合了“Online”学习与加权投票的思想。但随着研究的深入，不同学者从不同的视域提出了各种新的理论模型，越来越多的将 Boosting 算法设计与优化理论联系起来，从不同的视角解释 AdaBoost 算法的原理和有效性。这些不同的视域分析也为新算法的设计提供了更广阔的思路。本章将分别从几种主流的视域对 AdaBoost 算法进行分析，并对其算法的原理进行讨论。

12.3.1 损失函数最小化视域

多数统计与机器学习问题都可视为对目标函数或者损失函数的优化。例如，最简单的一元线性回归问题。给定样本 $\{(\boldsymbol{X}_1, Y_1), \cdots, (\boldsymbol{X}_n, Y_n)\}$。最小二乘估计的目标是找到参数 $\boldsymbol{\beta} = (\beta_1, \boldsymbol{\beta}_2)^{\mathrm{T}}$，使其最小化残差平方和

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(Y_i - \beta_1 - \boldsymbol{X}_i^{\mathrm{T}} \beta_2\right)^2.$$

其中，$\mathrm{RSS}(\boldsymbol{\beta})$ 就是损失函数。许多其他方法，如神经网络、最大似然估计、支持向量机、Logistic 回归等，都可视为某种程度下的目标函数优化问题。

现在的问题是：AdaBoost 是否也可视为是最优化目标函数的过程？后面的研究表明，AdaBoost 确实可视为优化某种损失函数的一种算法。

指数损失函数

那么 AdaBoost 关联的损失函数如何定义呢？我们前面得到分类器的误差率为

$$\frac{1}{n} \sum_{i=1}^{n} I\{\mathcal{H}(\boldsymbol{X}_i) \neq Y_i\} \tag{12.3.1}$$

那么 AdaBoost 算法就是最小化 (12.3.1) 所示的目标函数吗？其实并不是，而是最小化式 (12.3.1) 的一个上界。因为 $\mathcal{H}(X)=\text{sign}(\widetilde{h}_T(\boldsymbol{X}))$，则

$$\frac{1}{n}\sum_{i=1}^{n} I\{\mathcal{H}(\boldsymbol{X}_i)\neq Y_i\}=\frac{1}{n}\sum_{i=1}^{n} I\{\text{sign}(\widetilde{h}_T(\boldsymbol{X}_i))\neq Y_i\}=\frac{1}{n}\sum_{i=1}^{n} I\{\widetilde{h}_T(\boldsymbol{X}_i)Y_i\leq 0\}. \tag{12.3.2}$$

利用 $I\{x\leq 0\}\leq e^{-x}$，则知误差率上界可取为

$$\mathscr{L}_{\exp}(\widetilde{h}|\mathcal{D})=\text{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_T(\boldsymbol{X})}] \tag{12.3.3}$$

其中，$Y\widetilde{h}_T(\boldsymbol{X})$ 称为假定问题的分类间隔。

要使上述损失函数达到最小，关键要解决以下两个问题：

1、如何确定一系列的弱学习器 h_t？

2、如何确定合适的权重 α_t？

利用式 (12.1.6) 可知，只需求

$$\mathscr{L}_{\exp}(\alpha_t h_t|\mathcal{D}_t)=\text{E}_{\boldsymbol{X}\sim\mathcal{D}_t}[e^{-Y\alpha_t h_t}]. \tag{12.3.4}$$

在分布 $\mathcal{D}_t$ 下关于权重 α_t 和学习器 h_t 的最小值。

$$\begin{aligned}
&\text{E}_{\boldsymbol{X}\sim\mathcal{D}_t}[e^{-Y\alpha_t h_t}]\\
=&E_{\boldsymbol{X}\sim\mathcal{D}_t}[e^{-\alpha_t}I(Y=h_t(\boldsymbol{X}))+e^{\alpha_t}I(Y\neq h_t(\boldsymbol{X}))]\\
=&e^{-\alpha_t}\text{Pr}_{\boldsymbol{X}\sim\mathcal{D}_t}(Y=h_t(\boldsymbol{X}))+e^{\alpha_t}\text{Pr}_{\boldsymbol{X}\sim\mathcal{D}_t}(Y\neq h_t(\boldsymbol{X}))\\
=&e^{-\alpha_t}(1-\text{err}_{\mathcal{D}_t})+e^{\alpha_t}\text{err}_{\mathcal{D}_t}
\end{aligned} \tag{12.3.5}$$

其中 $\text{err}_{\mathcal{D}_t}=\text{Pr}_{\boldsymbol{X}\sim\mathcal{D}_t}(Y\neq h_t(\boldsymbol{X}))$。为得到最优 α_t，将指数损失函数求导令其为零，则

$$\frac{\partial\mathscr{L}_{\exp}(\alpha_t h_t|\mathcal{D}_t)}{\partial\alpha_t}=-e^{-\alpha_t}(1-\text{err}_{\mathcal{D}_t})+e^{\alpha_t}\text{err}_{\mathcal{D}_t}=0 \tag{12.3.6}$$

解得

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1-\text{err}_{\mathcal{D}_t}}{\text{err}_{\mathcal{D}_t}}\right) \tag{12.3.7}$$

这正是 AdaBoost 算法中计算 α_t 的方法。下面讨论 h_t 的选取。

假设已获得了一系列弱学习器并知其权重，则组合形成 $\widetilde{h}_{t-1}$。关于 h_t 的理想选择就是纠正 $\widetilde{h}_{t-1}$ 的错误，最小化指数损失函数

$$\begin{aligned}
&\mathscr{L}_{\exp}(\widetilde{h}_{t-1}+h_t|\mathcal{D})\\
=&\text{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y(\widetilde{h}_{t-1}(\boldsymbol{X})+h_t(\boldsymbol{X}))}]
\end{aligned}$$

$$=\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\cdot e^{-Yh_t(\boldsymbol{X})}] \tag{12.3.8}$$

对 $e^{-Yh_t(\boldsymbol{X})}$ 利用 Taylor 展式，则

$$\begin{aligned}&\mathscr{L}_{\exp}(\widetilde{h}_{t-1}+h_t|\mathcal{D})\\ \approx&\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\left(1-Yh_t(\boldsymbol{X})+\frac{Y^2h_t(\boldsymbol{X})^2}{2}\right)\right]\\ =&\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\left(1-Yh_t(\boldsymbol{X})+\frac{1}{2}\right)\right]\end{aligned} \tag{12.3.9}$$

因此，h_t 的理想选择为

$$\begin{aligned}h_t(\boldsymbol{X})&=\mathrm{argmin}_h\mathscr{L}_{\exp}(\widetilde{h}_{t-1}+h_t|\mathcal{D})\\ &=\mathrm{argmin}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\left(1-Yh_t(\boldsymbol{X})+\frac{1}{2}\right)\right]\\ &=\mathrm{argmax}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}Yh_t(\boldsymbol{X})\right]\\ &=\mathrm{argmax}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[\frac{e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}}{\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}]}Yh_t(\boldsymbol{X})\right]\end{aligned} \tag{12.3.10}$$

记

$$\mathcal{D}_t(\boldsymbol{X})=\frac{\mathcal{D}(\boldsymbol{X})e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}}{E_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\right]} \tag{12.3.11}$$

则利用期望的定义，可得

$$\begin{aligned}h_t(\boldsymbol{X})&=\mathrm{argmax}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[\frac{e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}}{\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}]}Yh_t(\boldsymbol{X})\right]\\ &=\mathrm{argmax}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}_t}[Yh_t(\boldsymbol{X})]\end{aligned} \tag{12.3.12}$$

又由于 $Yh_t(\boldsymbol{X})=1-2I(Y\neq h_t(\boldsymbol{X}))$，因而 $h_t(\boldsymbol{X})$ 满足

$$h_t(\boldsymbol{X})=\mathrm{argmin}_h\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}_t}[I(Y\neq h_t(\boldsymbol{X}))] \tag{12.3.13}$$

通过式 (12.3.11)，可得

$$\begin{aligned}\mathcal{D}_{t+1}(\boldsymbol{X})&=\frac{\mathcal{D}(\boldsymbol{X})e^{-Y\widetilde{h}_t(\boldsymbol{X})}}{\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[e^{-Y\widetilde{h}_t(\boldsymbol{X})}\right]}\\ &=\frac{\mathcal{D}(\boldsymbol{X})e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}\cdot e^{-Y\alpha_th_t(\boldsymbol{X})}}{\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_t(\boldsymbol{X})}]}\\ &=\mathcal{D}_t(\boldsymbol{X})\cdot e^{-Y\alpha_th_t(\boldsymbol{X})}\frac{\mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_{t-1}(\boldsymbol{X})}]}{E_{\boldsymbol{X}\sim\mathcal{D}}[e^{-Y\widetilde{h}_t(\boldsymbol{X})}]}\end{aligned} \tag{12.3.14}$$

这恰好是 AdaBoost 算法更新分布的方法。

注 12.4 从优化损失函数的角度分析 AdaBoost 算法，有以下优点：

1、帮助我们明朗化 AdaBoost 算法的学习目标，有助于理解 AdaBoost 的理论原理及相关性质。

2、实现学习目标（损失函数最小化）与实现目标（数值方法）之间的“解耦”（decoupling）。一方面可以修改目标函数以适应新的学习模型，另一方面可以引入快速实用的数值方法。从而可以建立其他类型的 AdaBoost 算法 [150]。如优化任意可导函数的 AnyBoost；优化基于分类间隔损失函数的 MarginBoost 等 [151]。

Logistic 损失函数

对于 AdaBoost 算法，如注 12.4 所言，可以根据需要使用不同的损失函数。

对于指数损失函数 $e^{-Y\widetilde{h}_T(\boldsymbol{X})}$，其优点是性质好且容易处理。但如果预测错误，则 $Y\widetilde{h}_T(\boldsymbol{X})$ 为负，从而 $e^{-Y\widetilde{h}_T(\boldsymbol{X})}$ 为指数增长。这意味着此时对扰动很敏感，即稳健性较差。

为此，在当代 Boosting 方法中，常用 Logistic 损失函数：

$$\mathscr{L}_{\log}(\widetilde{h}|\mathcal{D}) = \mathrm{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[\ln\left(1+e^{-Y\widetilde{h}_T(\boldsymbol{X})}\right)\right]. \tag{12.3.15}$$

显然根据指数函数与对数函数的性质知式 (12.3.3) 与式 (12.3.15) 会被相同的函数最小化。事实上，可以通过极大似然的思想来解释 $\ln\left(1+e^{-Y\widetilde{h}_T(\boldsymbol{X})}\right)$ 的来源。已知 $Y_i\in\{-1,1\}$，且服从 Logit 模型，则有：

$$\Pr(Y_i|\boldsymbol{X}_i) = \begin{cases} \dfrac{1}{1+e^{-\widetilde{h}_T(\boldsymbol{X}_i)}}, & \text{if } Y_i = 1 \\ \dfrac{1}{1+e^{\widetilde{h}_T(\boldsymbol{X}_i)}}, & \text{if } Y_i = -1 \end{cases}$$

考虑到 Y_i 的取值为 -1 或者 1，将上式合并成

$$\Pr(Y_i|\boldsymbol{X}_i) = \begin{cases} \dfrac{1}{1+e^{-Y_i\widetilde{h}_T(\boldsymbol{X}_i)}}, & \text{if } Y_i = 1 \\ \dfrac{1}{1+e^{-Y_i\widetilde{h}_T(\boldsymbol{X}_i)}}, & \text{if } Y_i = -1 \end{cases}$$

即条件概率的表达式统一为

$$\Pr(Y_i|\boldsymbol{X}_i) = \frac{1}{1+e^{-Y_i\widetilde{h}_T(\boldsymbol{X}_i)}}$$

从而极大似然函数为

$$\prod_{i=1}^{n}\Pr(Y_i|\boldsymbol{X}_i) = \prod_{i=1}^{n}\frac{1}{1+e^{-Y_i\widetilde{h}_T(\boldsymbol{X}_i)}} = \prod_{i=1}^{n}\left(1+e^{-Y_i\widetilde{h}_T(\boldsymbol{X}_i)}\right)^{-1} \tag{12.3.16}$$

上式 (12.3.16) 的最大值等价成取负对数的最小值，即式 (12.3.15).

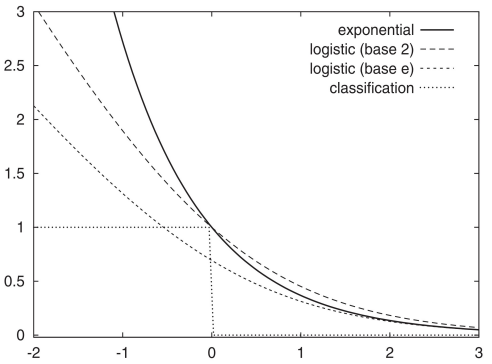


图 12.3 指数损失、Logistic 损失、分类损失

注 12.5 指数损失函数与 Logistic 损失函数都可视作对 0-1 损失函数的光滑近似。相比较而言，Logistic 的近似程度要更好。见图 12.3。

Friedman et al. 认为，与其用 AdaBoost 中的拟牛顿更新策略，不如通过梯度下降的方式优化 Logistic 损失函数，并据此给出了 LogitBoost 算法 [141]。

LogitBoost 算法

输入：训练数据集 $\{(\boldsymbol{X}_1, Y_1), (\boldsymbol{X}_2, Y_2), \cdots, (\boldsymbol{X}_n, Y_n)\}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程：

1. $Y_0(\boldsymbol{X}_i) = Y_i$; % 初始化目标
2. $\widetilde{h}_0(\boldsymbol{X}) = 0$; % 初始化函数
3. **for** $t = 1, ..., T$:
4. $p_t(\boldsymbol{X}) = \dfrac{1}{1 + e^{-2\widetilde{h}_{t-1}(\boldsymbol{X})}}$; % 计算概率
5. $Y_t(\boldsymbol{X}) = \dfrac{Y_{t-1}(\boldsymbol{X}) - p_t(\boldsymbol{X})}{p_t(\boldsymbol{X})(1 - p_t(\boldsymbol{X}))}$; % 更新目标
6. $\mathcal{D}_t(\boldsymbol{X}) = p_t(\boldsymbol{X})(1 - p_t(\boldsymbol{X}))$; % 更新权重
7. $h_t = \mathcal{L}(\mathcal{D}, Y_t, \mathcal{D}_t)$; % 训练分类器 h_t 在分布 $\mathcal{D}_t$ 下拟合样本集中的 $Y_t()$
8. $\widetilde{h}_t = \widetilde{h}_{t-1} + \dfrac{1}{2} h_t$; % 更新组合分类器
9. **end**

输出：$\mathcal{H}(\boldsymbol{X}) = \text{sign}(\sum_{t=1}^{T} h_t(\boldsymbol{X}))$

12.3.2 向前逐段可加视域

向前逐段算法

对于二元分类问题的 AdaBoost 算法，其最终表达式为

$$\mathcal{H}(\boldsymbol{X}) = \text{sign}(\widetilde{h}_T(\boldsymbol{X})) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X}))$$

可将 $h_t(\boldsymbol{X}) \in \{-1, 1\}$ 视为基函数，类似于泰勒展式的函数项。不失一般性，将希望学到的函数 $f(x)$ 做基函数展开，可以得到加法模型：

$$f(x) = \sum_{t=1}^{T} \alpha_t h(x; \gamma_t) \tag{12.3.17}$$

其中，α_t 为展开系数，$h(x; \gamma_t)$ 为基函数，γ_t 为参数。

为估计展开式系数 α_t 与参数 γ_t，可以最小化以下目标函数：

$$\min_{\alpha_t's, \gamma_t's} \sum_{i=1}^{n} L\left(Y_i, \sum_{t=1}^{T} \alpha_t h(\boldsymbol{X}_i; \gamma_t)\right) \tag{12.3.18}$$

其中，$L(Y_i, f(\boldsymbol{X}_i))$ 为损失函数。例如 $L(Y_i, f(\boldsymbol{X}_i)) = (Y_i - f(\boldsymbol{X}_i))^2$，$L(Y_i, f(\boldsymbol{X}_i)) = |Y_i - f(\boldsymbol{X}_i)|$ 或者 0-1 损失函数 $I(Y_i \neq f(\boldsymbol{X}_i))$。

显然，这是一个复杂的优化问题。向前分段算法（forward stagewise algorithm）求解这一优化问题的想法是：利用加性模型的性质，考虑从向前的分步计算，每一步只学习一个基函数及其系数，逐步逼近优化目标函数式 (12.3.18)。具体地，每步只需优化如下损失函数：

$$\min_{\alpha, \gamma} \sum_{i=1}^{n} L\left(Y_i, \alpha h(\boldsymbol{X}_i; \gamma)\right) \tag{12.3.19}$$

向前分段算法与 AdaBoost 算法

由向前分段算法可以推导出 AdaBoost，用如下定理叙述这一关系。

定理 12.4 二元分类问题的 AdaBoost 算法是向前分段加法算法的特例。其中模型是由基本分类器组成的加法模型，损失函数为 $L(Y, f(\boldsymbol{X})) = e^{-Yf(\boldsymbol{X})}$。

该定理的证明关键是找出 $h_t(\boldsymbol{X})$ 和 α_t，其证明过程类似于前面的推导过程，可参见 [19]。

12.4 Boosting 算法的演化

基于 AdaBoost 的理论剖析，人们对 AdaBoost 算法进行了各种改进与推广，并进行了广泛应用。

12.4.1 回归问题的 Boosting 算法

最初的 AdaBoost 算法是针对于分类问题而设计的，但其算法思想极其具有一般性，因而可将其应用于其他问题，下面重点介绍回归问题的 LSBoost 方法。

对于回归问题，一般采用均方误差为损失函数：

$$\mathscr{L}_{\text{mse}} = \left(Y - \widetilde{h}_T(\boldsymbol{X})\right)^2 \tag{12.4.1}$$

将 12.3.2 节中的向前逐段加法模型代入该损失函数如下，并且最小化该损失函数可得 α_t,γ_t：

$$\begin{aligned}
&\min_{\alpha,\gamma} \sum_{i=1}^{n} \mathscr{L}_{\text{mse}} \left(Y_i, \widetilde{h}_{t-1}(\boldsymbol{X}_i) + \alpha \cdot h(\boldsymbol{X}_i;\gamma)\right) \\
=&\min_{\alpha,\gamma} \sum_{i=1}^{n} \left(Y_i - \widetilde{h}_{t-1}(\boldsymbol{X}_i) - \alpha \cdot h(\boldsymbol{X}_i;\gamma)\right)^2 \\
=&\min_{\alpha,\gamma} \sum_{i=1}^{n} \left(r_{ti} - \alpha \cdot h(\boldsymbol{X}_i;\gamma)\right)^2 .
\end{aligned} \tag{12.4.2}$$

其中，r_{ti} 为当前阶段模型的残差。因而算法的更新思想是以当前残差 r_{ti} 为因变量，对自变量 $h(\boldsymbol{X}_i;\gamma)$ 进行回归即可。进行回归时，可以采用普通的线性回归方法，也可以使用如下所述的回归树。

树模型

分类与回归树模型（classification and regression tree，CART）主要由 Quinlan、Breiman 等人创立 [67] [131]，是一种基本的分类与回归方法。用于分类问题时，称为分类树（classification tree）；用于回归问题，称为回归树（regression tree）。这些是前面介绍过的内容。

对于分类问题，我们回忆例 12.1 的做法。基本分类器可以看作是由一个根节点连接两个叶节点的简单决策树，常称为决策树桩（decision stump）。最终的强分类器形式为 $\widetilde{h}_T(\boldsymbol{X}) = \sum_{t=1}^{T} h_t(\boldsymbol{X})$。抽象成一般形式应用于回归问题，则回归 Boosting 树模型可以表示以决策树为基函数的加法模型：

$$\widetilde{h}_T(\boldsymbol{X}) = \sum_{t=1}^{T} G(\boldsymbol{X};\Theta_t) \tag{12.4.3}$$

其中，$G(\boldsymbol{X};\Theta_t)$ 表示决策树，Θ_t 为决策树的参数，T 为树的数量。

回归问题的 Boosting 树算法

假设 $\mathcal{X}$ 为输入空间，$\mathcal{Y}$ 为输出空间，$\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$ 为训练数据集。

一棵回归树由输入空间的一个划分以及在划分单元上的取值两部分决定。假设已将输入空间 $\mathcal{X}$ 划分为 K 个单元：$R_1,\cdots,R_K$，并且在每个单元 R_k 上的取值为 c_k，于是回归树具有如下形式：

$$G(\boldsymbol{X};\Theta)=\sum_{k=1}^{K}c_kI(\boldsymbol{X}\in R_k) \tag{12.4.4}$$

其中，参数 $\Theta=\{(R_1,c_1),(R_2,c_2),\cdots,(R_K,c_K)\}$ 表示树的区域划分和各区域上的常数。K 是回归树的复杂度，即叶节点个数。

回归问题的 Boosting 树算法采用向前分段算法。首先确定初始 Boosting 树 $\widetilde{h}_0(\boldsymbol{X})=0$，设 $\widetilde{h}_{t-1}(\boldsymbol{X})$ 为当前模型，则第 t 步的模型是

$$\widetilde{h}_t(\boldsymbol{X})=\widetilde{h}_{t-1}(\boldsymbol{X})+G(\boldsymbol{X};\Theta_t)$$

其中 Θ_t 的选取遵循损失最小化原则，即

$$\Theta_t=\mathrm{argmin}_{\Theta}\sum_{i=1}^{n}\mathscr{L}(Y_i,\widetilde{h}_{t-1}(\boldsymbol{X}_i)+G(\boldsymbol{X}_i;\Theta))$$

由于树的线性组合可以很好的拟合训练数据，所以回归问题的 Boosting 树是一个很高效的学习方法。此算法代码如下。

回归问题的 Boosting 树算法

输入：训练数据集 $\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$；
损失函数 $\mathscr{L}_{\text{mse}}(Y,\widetilde{h}_T(\boldsymbol{X}))$；基函数集 $\{G(\boldsymbol{X};\Theta_t)\}$；
学习轮数 T。

过程：

```
For t = 1, ..., T:
    Θ_t = argmin_Θ Σ_{i=1}^{n} L_mse(Y_i, h̃_{t-1}(X_i) + G(X_i; Θ));   % 拟合残差得到回归树
    h̃_t(X) = h̃_{t-1}(X) + G(X; Θ_t);                                  % 更新加法模型
end
```

输出：$\widetilde{h}_T(\boldsymbol{X})=\sum_{t=1}^{T}G(\boldsymbol{X};\Theta_t)$。

回归问题其他损失函数

对于回归问题的损失函数，可以根据需要取成不同的形式。

1、Laplace 损失函数：

$$\mathscr{L}_{ll}\left(Y,\widetilde{h}_T(\boldsymbol{X})\right)=\left|Y-\widetilde{h}_T(\boldsymbol{X})\right|$$

相比较于平方损失函数 $\mathscr{L}_{\text{mse}}$，Laplace 损失函数不易受到异常值的影响，因而更稳健。

2、Huber 损失函数：

$$\mathscr{L}_{hl}\left(Y,\widetilde{h}_T(\boldsymbol{X})\right)=\begin{cases}\dfrac{1}{2}\left[Y-\widetilde{h}_T(\boldsymbol{X})\right]^2, & \text{if } |Y-\widetilde{h}_T(\boldsymbol{X})|\leq\delta\\ \delta\left[|Y-\widetilde{h}_T(\boldsymbol{X})|-\dfrac{\delta}{2}\right], & \text{if } |Y-\widetilde{h}_T(\boldsymbol{X})|>\delta\end{cases}$$

其中，δ 为松弛参数。通过其表达式可以看出，如果误差绝对值较小时，使用平方损失函数；如果误差绝对值较大，则使用绝对值损失函数。因此，Huber 损失函数结合了平方损失函数与绝对值损失函数的优点。

12.4.2 梯度 Boosting 方法

梯度 Boosting 算法

根据前面的理论剖析，AdaBoost 可视为加性模型中利用坐标下降法优化指数型损失函数的过程。在此基础上，Friedman 提出了梯度 Boosting 算法（gradient boosting），简称为 GBM （gradient boosting machine）算法 [152]。GBM 思想：以非参数方法估计基函数，并在“函数空间”使用梯度下降法求解。

假设训练数据 $\mathcal{D}=\{(\boldsymbol{X}_1,Y_1),\cdots,(\boldsymbol{X}_n,Y_n)\}$，若以函数 $\widetilde{h}_T(\boldsymbol{X})$ 来预测 Y，则在总体中的平均损失（expected loss function）为

$$\mathrm{E}_{Y,\boldsymbol{X}}\mathscr{L}\left[Y,\widetilde{h}_T(\boldsymbol{X})\right]$$

其中 $\mathscr{L}[\cdot,\cdot]$ 为损失函数。目标变成找出能够最小化损失的函数 $\widetilde{h}^*(\boldsymbol{X})$，即

$$\widetilde{h}^*(\boldsymbol{X})=\mathrm{argmin}_{\widetilde{h}}\mathrm{E}_{Y,\boldsymbol{X}}\mathscr{L}\left[Y,\widetilde{h}_T(\boldsymbol{X})\right]$$

利用期望的性质将上式写成

$$\begin{aligned}\widetilde{h}^*(\boldsymbol{X})&=\mathrm{argmin}_{\widetilde{h}}\mathrm{E}_{Y,\boldsymbol{X}}\mathscr{L}\left(Y,\widetilde{h}_T(\boldsymbol{X})\right)\\&=\mathrm{argmin}_{\widetilde{h}}\mathrm{E}_{\boldsymbol{X}}\left(\mathrm{E}_Y\left(\mathscr{L}\left(Y,\widetilde{h}_T(\boldsymbol{X})\right)|\boldsymbol{X}\right)\right)\end{aligned}$$

因而最小化问题等价为

$$\min_{\widetilde{h}}\mathrm{E}_Y\left[\mathscr{L}(Y,\widetilde{h}_T(\boldsymbol{X}))|\boldsymbol{X}\right]\tag{12.4.5}$$

使用非参数思想，将 $\widetilde{h}_T(\boldsymbol{X})$ 在每个 $\boldsymbol{X}$ 的取值均看作参数。由此，$\widetilde{h}_T(\boldsymbol{X})$ 可视为无穷维向量，故有无穷多参数。在函数空间上，进行“泛函梯度下降”方法找最优解。

假设模型为加法模型：

$$\widetilde{h}^*(\boldsymbol{X})=\sum_{t=1}^{T}f_t(\boldsymbol{X})$$

利用梯度下降算法，则

$$f_t(\boldsymbol{X}) = -\rho_t g_t(\boldsymbol{X}) \tag{12.4.6}$$

其中 ρ_t 为步长，也称为学习率（learning rate），且

$$g_t(\boldsymbol{X}) = \left[\frac{\partial \mathrm{E}_Y\left(\mathscr{L}\left(Y, \widetilde{h}_T(\boldsymbol{X})\right)|\boldsymbol{X}\right)}{\partial \widetilde{h}_T(\boldsymbol{X})}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)}, \quad \widetilde{h}_{t-1}(\boldsymbol{X}) = \sum_{j=1}^{t-1} f_j(\boldsymbol{X}) \tag{12.4.7}$$

假设泛函正则性足够好，能够交换微积分次序，则

$$g_t(\boldsymbol{X}) = \mathrm{E}_Y\left[\frac{\partial \mathscr{L}(Y, \widetilde{h}_T(\boldsymbol{X}))}{\partial \widetilde{h}_T(\boldsymbol{X})}|\boldsymbol{X}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)}. \tag{12.4.8}$$

ρ_t 通过下式给出

$$\rho_t = \mathrm{argmin}_\rho \mathrm{E}_{Y,\boldsymbol{X}} \mathscr{L}(Y, \widetilde{h}_{t-1}(\boldsymbol{X}) - \rho g_t(\boldsymbol{X})) \tag{12.4.9}$$

因为训练数据的有限性，这种非参数统计的方法在实际应用中是行不通的。为解决此问题，首先假设

$$\widetilde{h}_T(\boldsymbol{X}; \boldsymbol{\alpha}, \boldsymbol{\gamma}) = \sum_{t=1}^{T} \alpha_t h(\boldsymbol{X}; \gamma_t) \tag{12.4.10}$$

其中 $h(\boldsymbol{X}; \gamma_t)$ 由基学习器决定。根据训练数据，则其梯度方向为

$$g_t(\boldsymbol{X}_i) = \left[\frac{\partial \mathscr{L}(Y_i, \widetilde{h}_T(\boldsymbol{X}_i))}{\partial \widetilde{h}_T(\boldsymbol{X}_i)}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)}, \quad i = 1, 2, \cdots, n \tag{12.4.11}$$

因而，选择与负梯度方向最为接近的弱学习器，即

$$\gamma_t = \mathrm{argmin}_\gamma \sum_{i=1}^{n} \left[-g_t(\boldsymbol{X}_i) - h(\boldsymbol{X}_i; \gamma)\right]^2. \tag{12.4.12}$$

求得最优 $h(\boldsymbol{X}_i; \gamma_t)$ 之后，则函数更新为

$$\widetilde{h}_t = \widetilde{h}_{t-1} + \alpha_t h(\boldsymbol{X}; \gamma_t) \tag{12.4.13}$$

其中，步长 ρ_t 利用直线搜索（line search）取为

$$\alpha_t = \mathrm{argmin}_\alpha \sum_{i=1}^{n} \mathscr{L}(Y_i, \widetilde{h}_{t-1}(\boldsymbol{X}_i) + \alpha \cdot h(\boldsymbol{X}_i; \gamma_t)) \tag{12.4.14}$$

GBDT 方法

以 GBM 算法为基础，结合树的思想给出 梯度提升决策树（gradient boosting decision tree，GBDT）方法。GBDT 以决策树为弱学习器的学习模型，利用梯度 Boosting 的方法，完成机器学习任务。直接给出算法如下

GBDT 算法

输入：训练数据集 $D=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$；
损失函数 $\mathscr{L}(\cdot,\cdot)$；
学习轮数 T.

过程：

1. $\widetilde{h}_0(\boldsymbol{X})=\text{argmin}_c\sum_{i=1}^{n}\mathscr{L}(Y_i,c)$； % 构造初始学习器
2. **for** $t=1,...,T$:
3. $r_{ti}=-\left[\dfrac{\partial\mathscr{L}(Y_i,\widetilde{h}_T(\boldsymbol{X}_i))}{\partial\widetilde{h}_T(\boldsymbol{X}_i)}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)}$； % 计算样本负梯度
4. 利用 $D_t=\{(\boldsymbol{X}_1,r_{t1}),\cdots,(\boldsymbol{X}_n,r_{tn})\}$ 学习得到新的叶节点 R_{tj}； %
5. $c_{tj}=\text{argmin}_c\sum_{i\in R_{tj}}\mathscr{L}(Y_i,\widetilde{h}_{t-1}(\boldsymbol{X}_i)+c)$,
   R_{tj} 表示第 t 棵树的第 j 个叶节点区域, $j=1,2,\cdots,J$. % 生成决策树
6. $\widetilde{h}_t(\boldsymbol{X})=\widetilde{h}_{t-1}(\boldsymbol{X})+\sum_{j=1}^{J}c_{tj}I(\boldsymbol{X}\in R_{tj})$； % 更新决策树
7. **end**

输出：$\widetilde{h}_T(\boldsymbol{X})=\sum_{t=1}^{T}\sum_{j=1}^{J}c_{tj}I(\boldsymbol{X}\in R_{tj})$.

注 12.6 对上述 GBDT 算法做一些说明：

(1) 算法第 1 步构造初始学习器，即只有一个根节点的初始决策树。

(2) 对于决策树模型，损失函数的常用方式有均方误差损失函数、Laplace 损失函数、Huber 损失函数等。

(3) 算法第 3 步计算出 r_{ti}，将其作为残差估计，称其为拟残差（pseudo residuals）。

(4) 算法第 4 步建立新的训练样本集 D_t：

$$D_t=\{(\boldsymbol{X}_1,r_{t1}),\cdots,(\boldsymbol{X}_n,r_{tn})\} \tag{12.4.15}$$

并用 D_t 作为训练样本集构造一棵决策树，取此决策树为第 $t+1$ 个基学习器。

(5) 算法第 5 步表示基学习器 $\widetilde{h}_T(\boldsymbol{X})$ 中每个叶节点的输出均使得上一轮迭代所取得模型的预测误差达到最小。

注 12.7 将 GBDT 应用于回归问题。假设损失函数取为均方损失函数，则

$$\mathscr{L}(Y,\widetilde{h}_T(\boldsymbol{X}))=\frac{1}{2}[Y-\widetilde{h}_T(\boldsymbol{X})]^2$$

则此时步骤 3 中拟残差的表达式变为

$$\begin{aligned}
r_{ti} &= -\left[\frac{\partial \mathscr{L}[Y_i,\widetilde{h}_T(\boldsymbol{X}_i)]}{\partial \widetilde{h}_T(\boldsymbol{X}_i)}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)} \\
&= -\left[\frac{\partial \frac{1}{2}[Y_i-\widetilde{h}_T(\boldsymbol{X}_i)]^2}{\partial \widetilde{h}_T(\boldsymbol{X}_i)}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)} \\
&= [Y_i-\widetilde{h}_T(\boldsymbol{X}_i)]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)} \\
&= Y_i-\widetilde{h}_{t-1}(\boldsymbol{X}_i)
\end{aligned} \tag{12.4.16}$$

此时对于回归问题，拟残差 r_{ti} 就是真正的残差 $Y_i-\widetilde{h}_{t-1}(\boldsymbol{X}_i)$。进一步计算最优步长

$$\begin{aligned}
\alpha_t &= \operatorname{argmin}_\alpha \sum_{i=1}^{n}(Y_i-\widetilde{h}_{t-1}(\boldsymbol{X}_i)-\alpha\cdot h(\boldsymbol{X}_i;\gamma_t))^2 \\
&= \operatorname{argmin}_\alpha \sum_{i=1}^{n}(r_{ti}-\alpha\cdot h(\boldsymbol{X}_i;\gamma_t))^2
\end{aligned} \tag{12.4.17}$$

因此知当基学习器为决策树时，GBDT 恰好是前面的回归 Boosting 方法。

12.5 AdaBoost 算法实践

AdaBoost 算法作为集成算法的杰出代表，不仅能够显著改善子分类器预测精度，而且本身有深厚的理论支撑，是理论与应用的完美结合。因而一经提出就受到不同领域研究人员的关注，在解决各行业应用问题中都获得了极大的成功。其应用领域举例如下：

- 手写字体识别
- 人脸检测
- 文本分类、文本过滤与信息检索

除此之外，AdaBoost 及其变种算法在机器视觉领域中被应用于目标检测、车辆识别、视频文字定位等；在计算机安全领域用于垃圾邮件分类、钓鱼网站检测等；在计算生物信息学中用于 DNA 序列分析；在高炉炼铁中应用于温度控制。以下两应用举例均来自参考文献《机器学习及 R 应用》，稍有改动 [153]。

12.5.1 R 语言实践

利用 AdaBoost 算法研究波士顿房价问题，其 R 语言代码如下：

```
install.packages("gbm")
library(gbm)
library(MASS)
set.seed(1)
train_index<-sample(506,354)
train<-Boston[train_index,]#设定训练集
test<-Boston[-train_index,]#设定测试集
set.seed(123)
#调用梯度提升算法：假设输出变量服从高斯分布；迭代次数为5000；基学习器深度为4
fit<-gbm(medv~.,data=train,distribution="gaussian",n.trees=5000,cv.fold=5,
interaction.depth=4,shrinkage=0.01,bag.fraction=0.5,n.minobsinnode=5)
summary(fit)
#分别画出偏依赖图
plot(fit,i.var="rm",main="Partial Importance Plot",ylab="medv")
plot(fit,i.var="lstat",main="Partial Importance Plot",ylab="medv")
#使用 predict() 函数，在测试集中进行预测，并计算均方误差
pred<-predict(fit,newdata=test,n.tees=5000)
y.test<-test[,"medv"]
mean((pred-y.test)^2)
```

从图 12.4 中看出，影响最大的变量依次是 rm 与 lstat。下面使用 gbm.perf() 考察决策树数目 M 对于训练误差与交叉验证误差的影响，结果如图 12.7。黑实线表示训练误差，绿色实线表示交叉验证误差，而蓝色虚线表示使交叉验证最小的决策树数目。上图可知，超出最优决策树数目 4960 后，随着迭代次数都加，训练误差趋于 0，而交叉验证误差则变大。

```
gbm.perf(fit,method="cv")
abline(h=0,lty=2)
legend("top",legend=c("Training Error","CV Error"),lty=1, col=c("black","green"))
#训练误差与交叉验证误差
min(fit$train.error)
which.min(fit$train.error)
min(fit$cv.error)
which.min(fit$cv.error)
```

考察决策树数目 $[4500, 5000]$ 区间的交叉验证误差，见图 12.8。

```
plot(4500:5000,fit$cv.error[4500:5000],type="l",xlab="Trees",
```

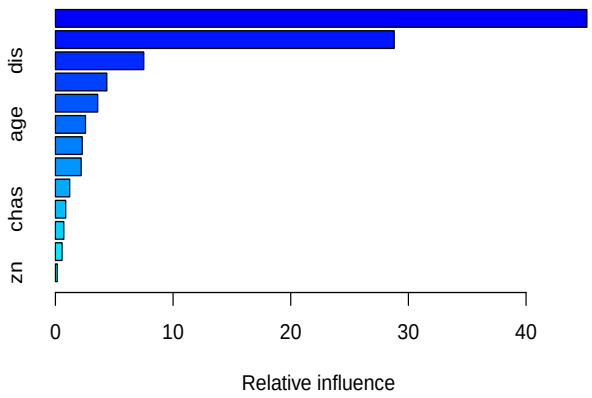


图 12.4 变量重要性

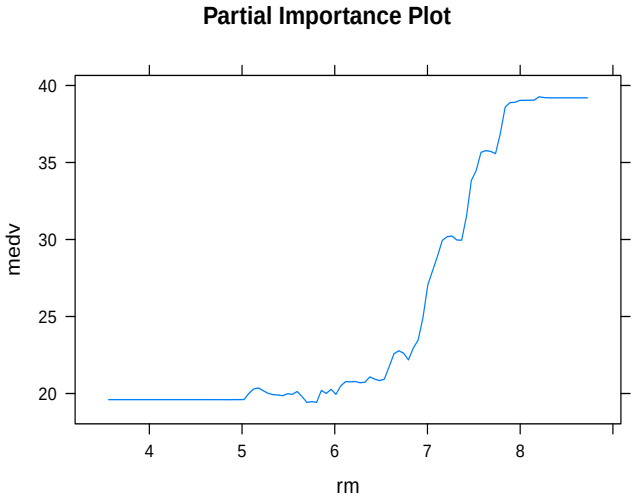


图 12.5 变量 rm 偏依赖图

```
ylab="Squared error loss",main="CV Error")
abline(v=which.min(fit$cv.error),lty=2)
#选择参数的最优值，使用下面的 expand.grid() 函数
grid <- expand.grid(shrinkage = c(.01,.1),interaction.depth = c(2, 4),
n.minobsinnode = c(5, 10),bag.fraction = c(.5, 1),optimal_trees = 0,min_MSE = 0)
grid
#进行网格搜索，找出最优值。将结果的前 6 行进行显示如下
for(i in 1:nrow(grid))
    {
    set.seed(123)
    gbm.tune <- gbm(medv~.,data=train,distribution = "gaussian",
```

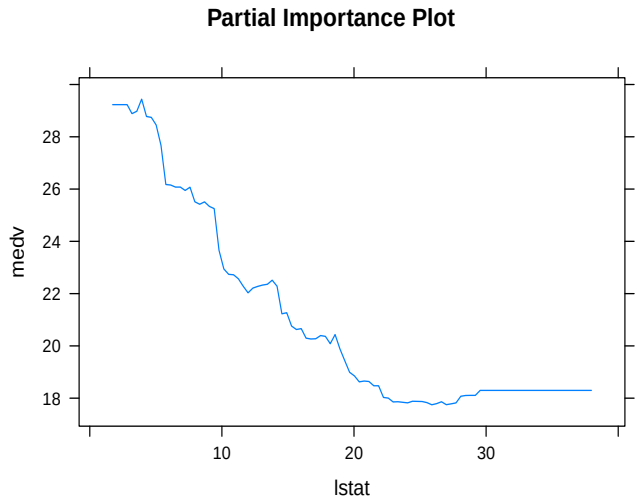


图 12.6 变量 lstat 偏依赖图

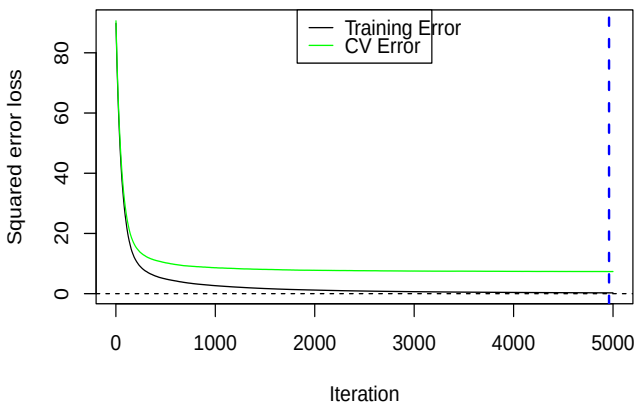


图 12.7 决策树数目对于训练误差与交叉验证误差的影响

```
    n.trees = 5000,interaction.depth = grid$interaction.depth[i],
    shrinkage = grid$shrinkage[i],n.minobsinnode = grid$n.minobsinnode[i],
    bag.fraction = grid$bag.fraction[i],cv.folds = 5)
    grid$optimal_trees[i] <- which.min(gbm.tune$cv.error)
    grid$min_MSE[i] <- min(gbm.tune$cv.error)
    }
head(grid[order(grid$min_MSE),])
```

根据网格搜索找出的最优值，最优参数分别为：收缩值 $\eta = 0.01$，交互深度 $d = 4$，终节点最小规模为 5，子抽样比例为 0.5。

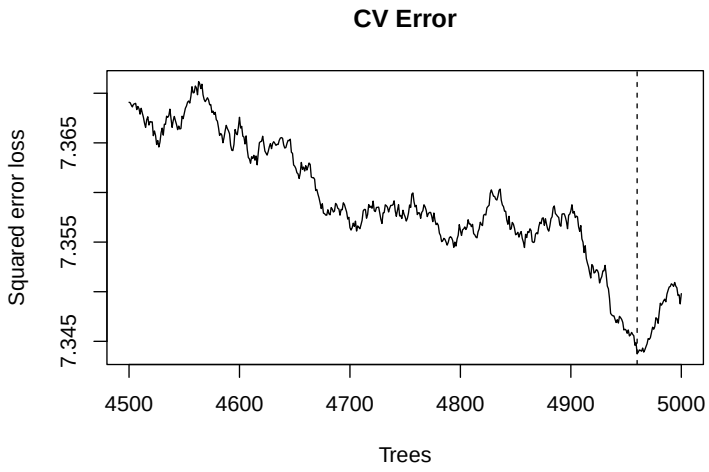


图 12.8 决策树数目对于交叉验证误差的影响

12.5.2 Python 语言实践

本小节将基于 Python 语言 sklearn.ensemble 函数库中的 AdaBoostClassifier 函数实现 AdaBoost 算法。具体实现过程如下：

1、导入需要的函数库，并生成数据集。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_gaussian_quantiles
X1, y1 = make_gaussian_quantiles(cov=2.0,n_samples=500, n_features=2,n_classes=2,
random_state=1)
X2, y2 = make_gaussian_quantiles(mean=(3, 3), cov=1.5,n_samples=400, n_features=2,
n_classes=2, random_state=1)
#生成两组二维正态分布数据
X = np.concatenate((X1, X2))#合成一组数据
y = np.concatenate((y1, - y2 + 1))
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y)#数据集可视化
```

我们通过可视化观察我们的分类数据。如图12.9所示，它有两个特征，两个输出类别，用颜色区别，可以看到数据有些混杂。

2、我们现在用基于决策树的 AdaBoost 来做分类拟合，这里我们选择了 SAMME 算法，最多 200 个弱分类器，步长 0.8，在实际运用中可能需要通过交叉验证调参而选择最好的参数。拟合完成后，绘制拟合区域图像并查看拟合分数。具体实现代码如下：

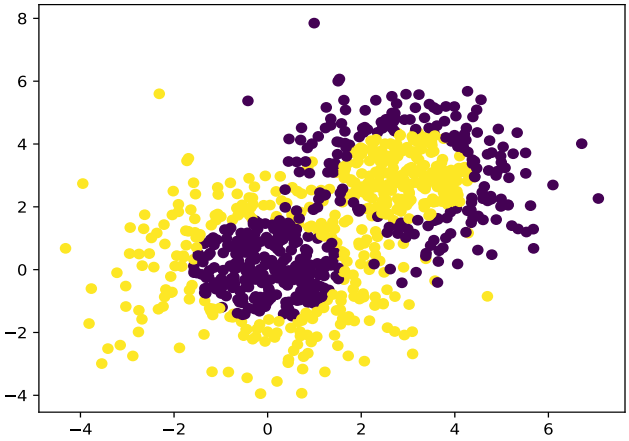


图 12.9 可视化数据集

```
bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=20,
min_samples_leaf=5), algorithm="SAMME", n_estimators=200, learning_rate=0.8)
bdt.fit(X, y)#进行分类拟合
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
Z = bdt.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y)
plt.show()#查看拟合区域
print("Score:", bdt.score(X,y))#输出拟合分数
```

拟合的区域如图12.10所示。AdaBoost 的拟合效果较为不错。拟合分数为 0.91 也同样不错，但需要注意的是拟合分数不一定越高越好，因为可能出现过拟合的情况。

3、现在我们将最大弱分离器个数从 200 增加到 300。再来看看拟合分数，具体实现代码如下：

```
bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=20,
min_samples_leaf=5), algorithm="SAMME", n_estimators=300, learning_rate=0.8)
bdt.fit(X, y)
print("Score:", bdt.score(X,y))
#输出结果：
Score: 0.962222222222
```

这说明弱分离器个数越多，则拟合程度越好，当然也越容易过拟合。

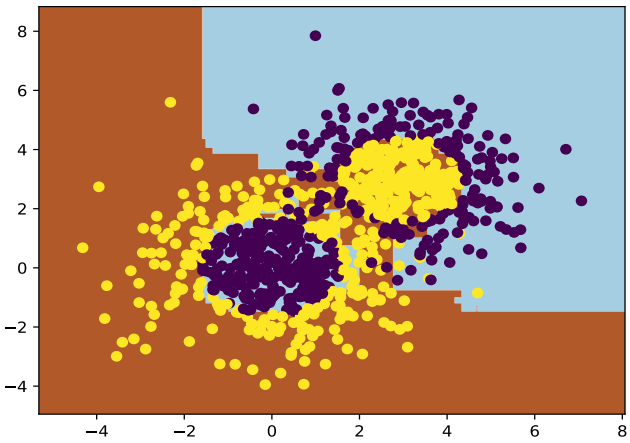


图 12.10 拟合区域图

总结

本章首先介绍 Boosting 方法及其代表性算法 AdaBoost；然后进行 AdaBoost 算法的误差分析，进一步解释 AdaBoost 算法提高学习精度的原因并叙述 Boosting 方法更具体的实例——Boosting tree （提升树）；最后，讨论 AdaBoost 及其变种算法和它们在实际问题中的应用。

接着讲述 Python 程序实现相关问题: Python AdaBoost 包在 Python 的编辑器中，可以通过导入 sklearn.ensemble 模块中的 AdaBoostClassifier 和 AdaBoostRegressor 来创建 AdaBoost 实例。其中，AdaBoostClassifier 用于分类任务，AdaBoostRegressor 用于分类任务。导入代码如图所示。

下来说明上述 AdaBoostClassifier、AdaBoostRegressor 实例的参数及其可选范围。

1、AdaBoostClassifier：

（1）$base_estimator$：object 型，默认值为 None 基分类器，默认是决策树，理论上可以是任意一个分类器，但是如果是其他分类器时需要指明样本权重。

（2）$n_estimator$：int 型，默认值为 50 集成的基分类器数量，取值必须大于等 1。

（3）$Learning_rate$：float 型，默认值为 1.0 每个基分类器的集成权重。

algorithm 默认值为 “SAMME.R”，基分类器权重的度量。支持 “SAMME” 和 “SAMME.R” 两种方式，前者是使用对样本集分类效果作为基分类器的权重，而后者使用对样本集的预测概率大小作为基分类器的权重，如果使用的是’ SAMME.R’，则基分类器参数 $base_estimator$ 必须限制使用支持概率预测的分类器。

$random_state$：int 型，默认值为 None，随机种子设置

3、AdaBoostRegressor:

(1) $base\_estimator$：同上

(2) $n_estimator$：同上

(3) $learning_rate$：同上

loss 默认值为 “linear” 作为损失函数的形式。支持的形式有 “linear” 线性损失、“square” 平方损失、“exponential” 指数损失.

12.6 习题

1、给定如表 12.2 所示训练数据。假设弱分类器为阈值函数，选择弱分类器的标准是其阈值 s 使该分类器在训练数据集上分类误差率最低。试用 AdBoost 算法学习一个强分类器，并尝试给出算法代码。

表 12.2 训练数据表

序号	1	2	3	4	5	6	7	8	9	10
X_i	0	1	2	3	4	5	6	7	8	9
Y_i	−1	−1	−1	1	1	1	−1	−1	−1	1

2、某公司招聘职员考察身体、业务能力、发展潜力这 3 项。身体分为合格 1、不合格 0 两级；业务能力和发展潜力分为上 1、中 2、下 3 三级；分类标签分为合格 1、不合格 −1 两类。已知 10 人数据，如表 12.3 所示。假设弱分类器为决策树桩，试用 AdaBoost 算法学习一个强分类器，并尝试给出算法代码。

表 12.3 应聘人员情况数据表

序号	1	2	3	4	5	6	7	8	9	10
身体	0	0	1	1	1	0	1	1	1	0
业务能力	1	3	2	1	2	1	1	1	3	2
发展潜力	3	1	2	3	3	2	2	1	1	1
分类	−1	−1	−1	−1	−1	−1	1	1	−1	−1

3、已知如表 12.4 所示的训练数据，X_i 的取值范围为区间 $(0.5, 10.5)$，Y_i 的取值范围为区间 $(5.0, 10.0)$，请使用回归问题的 Boosting 树算法进行建模分析，考虑只用树桩作为基函数。

4、现有某工厂四位工人的考评信息及月薪数据，如表 12.5 所示。试根据该数据集和 GBDT 学习算法构造包含两个个体学习器的集成模型，并使用该集成模型预测工龄为 25 年，绩效得分为 65 分的工人的月薪。

表 12.4 训练数据表

X_i	1	2	3	4	5	6	7	8	9	10
Y_i	5.56	5.70	5.91	6.40	6.80	7.05	8.90	8.70	9.00	9.05

表 12.5 员工信息表

编号	工龄 (年)	绩效得分	月薪 (万元)
1	5	20	1.1
2	7	30	1.3
3	21	70	1.7
4	30	60	1.8

5、第 4 题的损失函数为均方损失函数，尝试分别利用 Laplace 损失函数、Huber 损失函数，仍然根据表 12.5，利用 GBDT 算法构造包含 3 个个体的学习器。

第十三章　模型平均

模型平均是另一种集成学习技术，通过独立训练多个相同类型的学习器，然后对它们的预测结果进行平均或投票，可以降低过拟合风险，提高模型的鲁棒性。相较于随机森林，模型平均是一种更通用的技术，可以应用于不同类型的模型。

13.1　简介

模型平均（Model Averaging）通过整合多个独立模型的预测结果，旨在提高整体估计的准确性和鲁棒性。从统计学角度看，模型平均可视为对概率分布的加权融合，以降低估计的方差。通过简单平均或加权平均多个模型的输出，可以有效减小由单一模型引入的估计误差。关键在于确保模型之间的适度独立性，以最大程度地利用它们的差异性。模型平均在处理有限数据、降低过拟合风险和提高整体模型鲁棒性方面具有广泛应用。

13.1.1　模型不确定性

在使用统计方法进行数据分析时，研究者通常根据分析的问题假定统计模型，并假定其为真实模型，即生成实际数据的模型。这种所谓的真实模型一般是根据研究者的经验或研究者使用数据里面的一些先验信息来建立的。但这种假设往往是不正确的，因为这个事先给定的模型只是真实模型的一个近似，并且有可能是错误的，并且在很多实际问题的数据分析研究中真实模型都是未知的，这就是模型不确定性带来的不良影响。从建模角度看，模型不确定性一方面体现在采用的模型形式不确定，例如函数形式、分布假定、模型结构或使用不同的预测变量等，另一方面是由于模型选择导致的不确定性，例如不同的模型选择方法对应不同的模型选择结果或同一模型选择方法的选择结果会不稳定。

13.1.2　模型选择与模型平均

考虑到模型不确定性时，传统的一种做法是利用模型选择方法从众多的候选模型中选出最优模型，并将其看作真实模型进行后续的统计推断。前面章节已经详细

介绍了模型选择，解决了过度复杂或简单的模型均可能使估计或者预测的方差偏大的问题。研究者们常采用的模型选择方法和准则包括逐步回归、AIC、BIC、C_p、交叉验证、Lasso 回归、SCAD 或 MCP 方法等，这些方法通过选择最优模型的过程看似在一定程度上解决了模型不确定性问题。但是模型选择方法导致了人们忽视模型选择过程所带来的不确定性，即模型选择本身就是不确定性的。

模型平均方法 (Model Averaging) 起源于 20 世纪 60 年代，早期最有影响的是 Bates 教授和诺贝尔经济学奖获得者 Granger 所做的工作，他们通过组合两个无偏的预测来说明组合预测的优越性 [155]。他们把来自不同模型的估计或者预测通过一定的权重平均起来，有时也称为模型组合，一般包括组合估计和组合预测。模型平均方法没有追求最优模型，而以特定的权重对所有候补模型的统计结果进行平均，在避免了遗失有用信息的同时也充分考虑了模型不确定性，并且使得估计更加稳健。

因此，模型平均解决了模型选择带来的一系列问题：

（1）. 模型平均使用连续的权重去组合来自不同模型的估计，在表示形式上，模型选择可以看作模型平均的特例，但它的权重只取 0 或者 1，因而模型平均估计一般更加稳健。

（2）. 模型选择通过一系列的模型选择方法选出一个最优模型，这样就可能导致遗失其他模型的信息或者其他变量所特有的影响因素信息。但是，模型平均方法不会把某个选定的模型当作真实模型，因为模型平均不轻易地排除任何模型，给每一个模型赋予一个权重去充分利用每一个模型的信息，因而减少信息的遗失。

（3）. 模型选择也会导致模型不确定性，因为可能选到一个与真实模型相差甚远的模型，统计推断就可能存在很高的风险。模型平均提供了一种保障机制，规避了这种风险，也可以说是避免了把鸡蛋放在同一个篮子里。

总的来说，模型选择是统计推断的基础，模型选择旨在选定一个最优模型，基于该模型进行统计推断。但是，模型选择的不确定性会影响到统计推断，从而使得分析结果会出现问题。模型平均方法从估计和预测角度来看是模型选择的推广，相比传统的模型选择方法来确定唯一的最优模型，模型平均方法通过组合不同的模型进行估计和预测，解决模型选择带来的模型不确定性问题，常常能够减小估计风险，得到更加有效的结论。模型平均本质上是一类集成学习方法。所谓集成学习方法，就是使用多种学习器进行学习，并使用某种规则把各个学习结果进行集成汇总，从而获得比单个学习器更好的学习效果的一种机器学习方法，它可以用于分类问题集成、回归问题集成等。

在模型平均的理论研究过程中，如何确定组合权重是最重要的问题。下面通过组合预测的角度，介绍模型平均的两类方法：贝叶斯模型平均（BMA：Bayesian Model Averaging）和频率模型平均（FMA：Frequentist Model Averaging）。

13.2 贝叶斯模型平均

贝叶斯模型平均是一种基于贝叶斯理论并且将模型本身的不确定性考虑在内的方法。基本步骤是: 设定待组合模型的先验概率和各个模型中参数的先验分布，然后用经典的贝叶斯方法进行统计分析。

设 Y 为组合预测随机变量，D 为已获得的数据。因为并不知道哪一个模型是最优模型，即模型本身存在着不确定性，我们设定 $\mathcal{M}=\{M_1,M_2,...,M_K\}$ 代表所有可能模型组成的模型空间，

根据贝叶斯模型平均方法，组合预测随机变量 Y 的后验分布为:

$$\begin{aligned}\Pr(y|D)=&\sum_{k=1}^{K}\Pr(y,M_k|D)\quad(\text{全概率公式})\\=&\sum_{k=1}^{K}\Pr(M_k|D)\Pr(y|M_k,D)\ (\text{条件概率公式})\end{aligned}\tag{13.2.1}$$

其中 $\Pr(M_k|D)$ 为给定数据 D 的条件下模型 M_k 的后验分布, 反映了研究人员对于真实模型的不确定性。根据贝叶斯公式，其形式为

$$\begin{aligned}\Pr(M_k|D)=&\frac{\Pr(D|M_k)\Pr(M_k)}{\Pr(D)}\qquad(\text{贝叶斯公式})\\=&\frac{\Pr(D|M_k)\Pr(M_k)}{\sum_{k=1}^{K}\Pr(D|M_k)\Pr(M_k)}\ (\text{全概率公式})\end{aligned}\tag{13.2.2}$$

其中 $\Pr(M_k)$ 为候补模型 M_k 的先验概率，在没有特别先验信息的条件下可取均匀分布，即 $\Pr(M_k)=1/K$；$\Pr(D|M_k)$ 为模型 M_k 的似然函数。在参数模型假设下，假定 $\boldsymbol{\theta}_k$ 为模型 M_k 的参数向量，例如线性回归模型 $\boldsymbol{\theta}_k$ 包含了回归系数和误差的方差，$\Pr(\boldsymbol{\theta}_k|M_k)$ 是给定模型 M_k 下 $\boldsymbol{\theta}_k$ 的先验概率函数，$\Pr(D|\boldsymbol{\theta}_k,M_k)$ 是给定模型 M_k 下参数化的似然函数。可以推出

$$\Pr(D|M_k)=\int\Pr(D|\boldsymbol{\theta}_k,M_k)\Pr(\boldsymbol{\theta}_k|M_k)d\boldsymbol{\theta}_k\tag{13.2.3}$$

从组合预测随机变量的后验分布可以发现，贝叶斯模型平均方法实际上是以模型的后验分布为权重，对所有模型的预测后验分布进行加权。使用贝叶斯模型平均的关键在于确定组合的模型以及各单项模型的后验概率，即权重。另外也可以看出，贝叶斯模型平均有两个困难：第一是 $\Pr(D|M_k)$ 计算中涉及到积分运算，如果模型复杂，积分也会变得困难；第二是候补模型的个数也需要科学方法去确定。

记 μ_k 和 σ_k^2 为给定数据 D 和候补模型 M_k, 则组合预测随机变量 Y 的条件期望和条件方差表示为 $\mathrm{E}(Y\mid D,M_k)=\mu_k$ 和 $\mathrm{var}(Y\mid D,M_k)=\sigma_k^2$。令候补模型 M_k

的后验概率为 $\Pr(M_k \mid D) = w_k$，则组合预测随机变量 Y 条件期望分解形式:

$$\begin{aligned}\mu(\boldsymbol{w}) = \mathrm{E}(Y \mid D) =& \mathrm{E}_{\mathcal{M}}[\mathrm{E}(Y \mid D, \mathcal{M})] \\ =& \sum_{k=1}^{K} \Pr(M_k \mid D)\, \mathrm{E}(Y \mid D, M_k) \\ =& \sum_{k=1}^{K} w_k \mu_k\end{aligned}$$

$\boldsymbol{w} = (w_1, ..., w_K)^{\mathrm{T}}$。在参数模型假设下，我们可以使用贝叶斯估计框架，使用方程（13.2.3）得到似然函数 $\Pr(D|M_k)$ 的估计。接下来使用公式（13.2.2）和给定的候补模型 M_k 的先验概率 $\Pr(M_k)$，估计出所有候补模型的权重 $\widehat{w}_k$。另外，我们根据基于模型 M_k 的后验概率分布 $\Pr(y|M_k, D)$ 估计出 $\widehat{\mu}_k$，则贝叶斯模型平均估计的预测值是

$$\widehat{\mu}(\widehat{\boldsymbol{w}}) = \sum_{k=1}^{K} \widehat{w}_k \widehat{\mu}_k$$

另外，条件方差 $\mathrm{var}(Y \mid D)$ 可以分解为:

$$\begin{aligned}\mathrm{var}(Y \mid D) =& \mathrm{E}_{\mathcal{M}}[\mathrm{var}(Y \mid D, \mathcal{M})] + \mathrm{var}_{\mathcal{M}}[\mathrm{E}(Y \mid D, \mathcal{M})] \\ =& \sum_{k=1}^{K} w_k \sigma_k^2 + \sum_{k=1}^{K} w_k \left(\mu_k - \mu\right)^2\end{aligned}$$

研究者认为上述条件方差被作为衡量模型不确定性的基础。$\mathrm{E}_{\mathcal{M}}[\mathrm{var}(Y \mid D, \mathcal{M})]$ 被认为是模型结构内方差, $\mathrm{var}_{\mathcal{M}}[\mathrm{E}(Y \mid D, \mathcal{M})]$ 则是结构间方差, 是由模型结构的不确定性引起的, 原因是当能够确定真实模型时, $\mathrm{var}_{\mathcal{M}}[\mathrm{E}(Y \mid D, \mathcal{M})]$ 为 0 。

13.3 频率模型平均

与贝叶斯模型平均相比较，频率模型平均方法不需要考虑如何设置候补模型（Candidate Model）的先验概率，模型估计和权重估计完全由数据确定。频率模型平均过程为：假设有 K 个模型（或者 K 种估计方法），每个模型（或方法）估计出一个预测值，我们不妨将其写为

$$\widehat{u}_k, \quad k = 1, ..., K,$$

那么频率模型平均方法得到的最终估计的预测值为:

$$\widehat{u} = w_1 \widehat{u}_1 + w_2 \widehat{u}_2 + \cdots + w_K \widehat{u}_K$$

其中 $\boldsymbol{w}=(w_1,\ldots,w_K)$ 为权重向量，通常满足 $0\leq w_k\leq 1$，$\sum_{k=1}^{K} w_k=1$。

若我们定义权重系数为示性函数，即令

$$w_k = I\{\text{第}k\text{个模型被选到}\}$$

则模型平均变为模型选择，从这里可以看出模型选择是模型平均的一个特例，模型平均是模型选择的推广。和贝叶斯模型平均方法一样，频率模型平均法也需要确定权重值，那么怎么求取候补模型对应的权重呢？下面介绍几种权重选择方法。

13.4 权重选择方法

13.4.1 基于信息准则

所谓信息准则（IC）的权重选择方法即对于每一个候补模型给出一个基于信息准则的得分，最后依据得分多少来描述候补模型的重要性。上节在介绍模型选择时已给出了 AIC 和 BIC 的公式，我们可以先计算每一个候补模型的 AIC 和 BIC 值，然后通过如下公式计算各个组合权重：

$$w_k=\frac{\exp(-\frac{\mathrm{IC}_k}{2})}{\sum_{k=1}^{K}\exp(-\frac{\mathrm{IC}_k}{2})} \tag{13.4.1}$$

其中 K 表示候补模型集合中模型的个数，w_k 表示第 k 个候补模型的权重；IC_k 为第 k 个候补模型的 AIC 或 BIC 值。由上式可看出基于信息准则的组合权重计算方法比较简单，只需计算每个候补模型下的 AIC 或 BIC 值即可，对应的模型平均方法称为AIC 模型平均和BIC 模型平均。由于计算方便，基于信息准则的思路是比较常用的权重选择方法。

13.4.2 基于 Mallows 准则

在模型选择准则的介绍中介绍了 C_p 准则，C_p 准则的全称为 Mallows's C_p 准则。我们接下来要介绍的 Mallows 准则是对 Mallows's C_p 准则的推广，基于该准则的权重选择方法是最小化 Mallows 准则来得到组合预测的权重。下面以线性模型为例介绍具体的过程。

记响应变量的观测样本 $\mathbf{Y}=(Y_1,...,Y_n)^{\mathrm{T}}$，估计的预测值记为 $\widehat{\boldsymbol{\mu}}=(\widehat{\mu}_1,\ldots,\widehat{\mu}_n)$，协变量样本矩阵记为 $\mathbf{X}=(\boldsymbol{X}_1,\ldots,\boldsymbol{X}_n)^{\mathrm{T}}$，其中 $\boldsymbol{X}_i=(1,X_{i1},\ldots,X_{ip})^{\mathrm{T}}$，$p$ 是变量

的个数，误差项 $\boldsymbol{e}=(e_1,\ldots,e_n)^{\mathrm{T}}$。协变量与响应变量之间的线性关系表示为

$$\mathbf{Y}=\mathbf{X}\boldsymbol{\beta}+\boldsymbol{e}$$

其中 $\boldsymbol{\beta}$ 为协变量对应的系数向量，形式为 $\boldsymbol{\beta}=(\beta_0,\beta_1,\ldots,\beta_p)^{\mathrm{T}}$。

假设研究者采用 K 个候补模型去获得 $\widehat{\boldsymbol{\mu}}$，其中第 k 个采用的候补模型为

$$M_k:\quad \mathbf{Y}=\mathbf{X}_k\boldsymbol{\beta}_k+\boldsymbol{e}$$

其中 $\mathbf{X}_k=(\boldsymbol{X}_{k1},...,\boldsymbol{X}_{kn})^{\mathrm{T}}$ 是 $n\times p_k$ 的第一列为 1 的矩阵，即它是由 $\mathbf{X}$ 除去第一列外的任意 p_k-1 列组成的，$\boldsymbol{\beta}_k$ 是其相应的 p_k 维的系数向量。一般情况下 $\mathbf{X}_k^{\mathrm{T}}\mathbf{X}_k$ 是可逆的，故 $\boldsymbol{\beta}_k$ 基于第 k 个候补模型的最小二乘估计为 $\widehat{\boldsymbol{\beta}}_k=(\mathbf{X}_k^{\mathrm{T}}\mathbf{X}_k)^{-1}\mathbf{X}_k^{\mathrm{T}}\mathbf{Y}$。相应地，$\boldsymbol{\mu}_k$ 的估计为 $\widehat{\boldsymbol{\mu}}_k$=$\mathbf{X}_k(\mathbf{X}_k^{\mathrm{T}}\mathbf{X}_k)^{-1}\mathbf{X}_k^{\mathrm{T}}\mathbf{Y}=H_k\mathbf{Y}$。其中 $H_k=\mathbf{X}_k(\mathbf{X}_k^{\mathrm{T}}\mathbf{X}_k)^{-1}\mathbf{X}_k^{\mathrm{T}}$ 是帽子矩阵。记权重向量 $\boldsymbol{w}=(w_1,...,w_K)^{\mathrm{T}}$，且满足：

$$\mathrm{H}_n=\{\boldsymbol{w}\in[0,1]^K:\sum_{k=1}^{K}w_k=1\}$$

那么 $\boldsymbol{\mu}$ 的模型平均估计为

$$\widehat{\boldsymbol{\mu}}(\boldsymbol{w})=\sum_{k=1}^{K}w_k\widehat{\boldsymbol{\mu}}_k \tag{13.4.2}$$

接下来，采用以下 Mallows 准则估计权重：

$$\mathrm{C}_n(\boldsymbol{w})=\boldsymbol{w}^{\mathrm{T}}\widehat{\mathbf{E}}^{\mathrm{T}}\widehat{\mathbf{E}}\boldsymbol{w}+2\widehat{\sigma}^2\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}$$

其中 $\widehat{\mathbf{E}}=(\widehat{\boldsymbol{e}}_1,...,\widehat{\boldsymbol{e}}_K)$，$\widehat{\boldsymbol{e}}_k=\mathbf{Y}-\widehat{\boldsymbol{\mu}}_k$ 为基于第 k 个候补模型估计的残差向量，$\boldsymbol{\phi}=(p_1,...,p_K)^{\mathrm{T}}$，$\widehat{\sigma}^2=\dfrac{\widehat{\boldsymbol{e}}_{\widetilde{k}}^{\mathrm{T}}\widehat{\boldsymbol{e}}_{\widetilde{k}}}{n-\boldsymbol{\phi}_{\widetilde{k}}}$，$\widetilde{k}$ 满足 $\boldsymbol{\phi}_{\widetilde{k}}=\max\{p_1,...,p_K\}$。通过极小化 Mallows 准则所得到的权重为：

$$\widehat{\boldsymbol{w}}_{\mathrm{M}}=\mathrm{argmin}_{\boldsymbol{w}\in\mathrm{H}_n}\mathrm{C}_n(\boldsymbol{w}) \tag{13.4.3}$$

该权重对应的模型平均估计成为Mallows 模型平均 (MMA：Mallows Model Average) 估计，将其带入式 (13.4.2) 可得到 MMA 估计的观测值 $\widehat{\boldsymbol{\mu}}(\widehat{\boldsymbol{w}})$。

13.4.3 基于 Jackknife 准则

基于 Jackknife 准则的模型平均方法是通过最小化 Jackknife 准则得到组合预测的权重。该方法适用于随机误差项异方差的情形，即当 $\mathrm{cov}(\boldsymbol{e}|\mathbf{X})=\mathrm{diag}(\sigma_1^2,...,\sigma_n^2)$ 的时候可用基于 Jackknife 准则的模型平均方法去选择权重。

假设研究者依然以线性模型为例，并且采用 K 个候补模型，其中第 k 个采用的候补模型使用的协变量观测矩阵是 $\mathbf{X}_k=(\boldsymbol{X}_{k1},...,\boldsymbol{X}_{kn})^{\mathrm{T}}$。使用 Jackknife 估计方法对第 k 个候补模型估计预测值的具体流程是：对于 $i=1,...,n$,

（1）. 删除第 i 个样本，使用最小二乘得到回归参数的估计

$$\widehat{\boldsymbol{\beta}}_{(-i)}=(\mathbf{X}_{k(-i)}^{\mathrm{T}}\mathbf{X}_{k(-i)})^{-1}\mathbf{X}_{k(-i)}^{\mathrm{T}}\mathbf{Y}_{(-i)}$$

其中 $\mathbf{X}_{k(-i)}$ 和 $\mathbf{Y}_{(-i)}$ 分别是去掉第 i 个样本之后的 $(n-1)\times p_k$ 协变量观测矩阵和 $(n-1)$ 维的响应变量观测向量。

（2）. 估计出第 i 个样本的预测值为 $\widetilde{\mu}_{ki}=\boldsymbol{X}_{ki}^{\mathrm{T}}\widehat{\boldsymbol{\beta}}_{(-i)}$。

最后得到基于第 k 个候补模型的对应 n 个样本的预测向量 $\widetilde{\boldsymbol{\mu}}_k=(\widetilde{\mu}_{k1},...,\widetilde{\mu}_{kn})^{\mathrm{T}}$。注意到 $\mathbf{X}_{k(-i)}^{\mathrm{T}}\mathbf{X}_{k(-i)}=\mathbf{X}_k^{\mathrm{T}}\mathbf{X}_k-\boldsymbol{X}_{ki}\boldsymbol{X}_{ki}^{\mathrm{T}}$，根据 Sherman–Morrison 公式，我们可以推导出

$$\widetilde{\mu}_{ki}=\sum_{j\neq i}\frac{H_{k,ij}}{1-H_{k,ii}}Y_j$$

其中 $H_{k,ij}$ 代表第 k 个模型的帽子矩阵第 i 行第 j 列的元素。上式表明 $\widetilde{\mu}_{ki}$ 不依赖于 Y_i。可以用矩阵表示第 k 个候补模型的预测向量为

$$\widetilde{\boldsymbol{\mu}}_k=(D_k(H_k-I_n)+I_n)\mathbf{Y}$$

其中 D_k 是对角阵，它的第 i 个对角元素为 $(1-H_{k,ii})^{-1}$，并且 $D_{k,ii}(H_{k,ii}-1)=\dfrac{H_{k,ii}-1}{1-H_{k,ii}}=-1$，$I_n$ 是单位阵。

令 $\widetilde{\mathbf{E}}=(\widetilde{\boldsymbol{e}}_1,\widetilde{\boldsymbol{e}}_2,...,\widetilde{\boldsymbol{e}}_K)$，$\widetilde{\boldsymbol{e}}_k=\mathbf{Y}-\widetilde{\boldsymbol{\mu}}_k$ 为基于第 k 个模型估计的残差向量，而且 $\widetilde{\boldsymbol{e}}_k$ 是弃一的交叉验证的误差向量，在统计学中弃一的交叉验证又称为 Jackknife。Jackknife 准则定义为：

$$\mathrm{J}_n(\boldsymbol{w})=\boldsymbol{w}^{\mathrm{T}}\widetilde{\mathbf{E}}^{\mathrm{T}}\widetilde{\mathbf{E}}\boldsymbol{w} \tag{13.4.4}$$

并且通过极小化 Jackknife 准则所得到的权重为：

$$\widehat{\boldsymbol{w}}_{\mathrm{J}}=\arg\min_{\boldsymbol{w}\in\mathrm{H}_n}\mathrm{J}_n(\boldsymbol{w})$$

并且基于 Jackknife 准则得出的组合模型权重对应的模型平均方法为Jackknife 模型平均（JMA：Jackknife Model Average）。

13.5 模型平均实践

13.5.1 R 语言实践

下面介绍怎么用 R 语言实现上面提到的模型平均方法。

贝叶斯模型平均-线性模型

```
library(BMA)
library(MASS)
data(UScrime)
x.crime<- UScrime[,-16]
y.crime<- log(UScrime[,16])
x.crime[,-2]<- log(x.crime[,-2])
crime.bicreg <- bicreg(x.crime, y.crime)
summary (crime.bicreg, digits=2)
```

输出结果为:

```
Call:
bicreg(x = x.crime, y = y.crime)
115 models were selected
Best 5 models (cumulative posterior probability = 0.2 ):
           p!=0    EV      SD    model 1  model 2  model 3  model 4  model 5
Intercept 100.0  -23.453 5.589 -22.637  -24.384  -25.946  -22.806  -24.505
M          97.3   1.381   0.535  1.478    1.514    1.605    1.268    1.461
So         11.7   0.014   0.056    .        .        .        .        .
Ed        100.0   2.121   0.525  2.221    2.389    2.000    2.178    2.399
Po1        72.2   0.648   0.465  0.852    0.910    0.736    0.986      .
Po2        32.0   0.247   0.438    .        .        .        .      0.907
LF          6.0   0.018   0.162    .        .        .        .        .
M.F         7.0  -0.063   0.466    .        .        .        .        .
Pop        30.1  -0.019   0.036    .        .        .     -0.057      .
NW         88.0   0.089   0.051  0.109    0.085    0.112    0.097    0.085
U1         15.1  -0.033   0.146    .        .        .        .        .
U2         80.7   0.268   0.199  0.289    0.322    0.274    0.281    0.330
GDP        31.9   0.187   0.350    .        .      0.541      .        .
Ineq      100.0   1.382   0.335  1.238    1.231    1.419    1.322    1.294
Prob       99.2  -0.250   0.100 -0.310   -0.191   -0.300   -0.216   -0.206
Time       43.7  -0.125   0.176 -0.287      .     -0.297      .        .
nVar                              8        7        9        8        7
r2                              0.842    0.826    0.851    0.838    0.823
BIC                           -55.912  -55.365  -54.692  -54.604  -54.408
post prob                       0.062    0.047    0.034    0.032    0.029
```

输出结果中标题为 “p!=0” 的列显示了变量在模型中的后验概率。标题为 “EV” 的列表示 BMA 后验估计的平均值，标题为 “SD” 的列表示每个变量的 BMA 后验估计的标准差。后面五列显示了找到的五个最佳模型的参数估计，以及它们包含的变量数量。还有 R^2 值、BIC 值和后验模型概率。

贝叶斯模型平均-logistic 模型

```
library(MASS)
data(birthwt)
birthwt$race <- as.factor (birthwt$race)
birthwt$ptl <- as.factor (birthwt$ptl)
```

```
bwt.bic.glm <- bic.glm (low~age+lwt+race+smoke+ptl+ht+ui+ftv,
data=birthwt, glm.family="binomial")
summary (bwt.bic.glm,conditional=T,digits=2)
```

输出结果与线性模型输出结果一样，故在此省略。

基于 AIC 与 BIC 信息准则

```
maic<-function(y,x,subset){
y <- as.matrix(y)
x <- as.matrix(x)
n <- nrow(x)
p <- ncol(x)
aic<-c()
bic<-c()
for(k in 1:nrow(subset)){
lm1<-lm(y~x[,which(subset[k,]==1)])
aic[k]<-AIC(lm1)
bic[k]<-BIC(lm1)
}
weight_aic<-exp(-aic/2)/sum(exp(-aic/2))
weight_bic<-exp(-bic/2)/sum(exp(-bic/2))
list(WAIC=weight_aic,WBIC=weight_bic)
}
n<-50
p<-4
rho<-0.7
library(MASS)
M1 <- matrix(rep(1:p,p),ncol=p,byrow=F)
corM <- rho^(abs(M1-t(M1)))
X<- mvrnorm(n,rep(0,p),corM)
epsilon <- rnorm(n)
Y <-0.5*exp(X[,1])+2*tan(X[,2])+X[,3]^3 + epsilon
subset<- matrix(c(1,1,1,1,0,1,0,1,0,0,0,1),3,4)
res_weight<-maic(Y,X,subset)
res_weight$WAIC
res_weight$WBIC
```

基于 Mallows 与 Jackknife 准则

```
library(quadprog)
gmaN <- function(y,x,method,subset){
  y <- as.matrix(y)
  x <- as.matrix(x)
  s <- as.matrix(subset)
  n <- nrow(x)
  p <- ncol(x)
  if ((nrow(s)==1) && (ncol(s)==1)){
     if (subset == 1){
        s <- matrix(1,nrow=p,ncol=p)
```

```
        s[upper.tri(s)] <- 0
        zero <- matrix(0,nrow=1,ncol=p)
        s <- rbind(zero,s)
    }
    if (subset == 2){
        s <- matrix(0,nrow=2^p,ncol=p)
        s0 <- matrix(c(1,rep(0,p-1)),1,p)
        s1 <- matrix(c(rep(0,p)),1,p)
        for (i in 2:2^p){
            s1 <- s0 + s1
            for (j in 1:p){
                if (s1[1,j] == 2){
                    s1[1,j+1] <- s1[1,j+1]+1
                    s1[1,j] <- 0
                }
            }
            s[i,] <- s1
        }
    }
}
```

进一步地，对参数进行定义：

```
m <- nrow(s)
bbeta <- matrix(0,nrow=p,ncol=m)
if (method == 2) ee <- matrix(0,nrow=n,ncol=m)

for (j in 1:m){
    ss <- matrix(1,nrow=n,ncol=1) %*% s[j,]
    indx1 <- which(ss[,]==1)
    xs <- as.matrix(x[indx1])
    xs <- matrix(xs,nrow=n,ncol=nrow(xs)/n)
    if (sum(ss)==0){
        xs <- x
        betas <- matrix(0,nrow=p,ncol=1)
        indx2 <- matrix(c(1:p),nrow=p,ncol=1)
    }
    if (sum(ss)>0){
        betas <- solve(t(xs)%*%xs)%*%t(xs)%*%y
        indx2 <- as.matrix(which(s[j,]==1))
    }
    beta0 <- matrix(0,nrow=p,ncol=1)
    beta0[indx2] <- betas
    bbeta[,j] <- beta0
    if (method == 2){
        ei <- y - xs %*% betas
        hi <- diag(xs %*% solve(t(xs) %*% xs) %*% t(xs))
        ee[,j] <- ei*(1/(1-hi))
    }
}
```

接着上面的代码继续分析：

```
  if (method == 1){
     ee <- y %*% matrix(1,nrow=1,ncol=m) - x %*% bbeta
     ehat <- y - x %*% bbeta[,m]
     sighat <- (t(ehat) %*% ehat)/(n-p)
  }

  a1 <- t(ee) %*% ee
  if (qr(a1)$rank<ncol(ee)) a1 <- a1 + diag(m)*1e-10
  if (method == 1) a2 <- matrix(c(-c(sighat)*rowSums(s)),m,1)
  if (method == 2) a2 <- matrix(0,nrow=m,ncol=1)
  a3 <- t(rbind(matrix(1,nrow=1,ncol=m),diag(m),-diag(m)))
  a4 <- rbind(1,matrix(0,nrow=m,ncol=1),matrix(-1,nrow=m,ncol=1))

  w0 <- matrix(1,nrow=m,ncol=1)/m
  QP <- solve.QP(a1,a2,a3,a4,1)
  w <- QP$solution
  w <- as.matrix(w)
  w <- w*(w>0)
  w <- w/sum(w0)
  betahat <- bbeta %*% w
  ybar <- mean(y)
  yhat <- x %*% betahat
  ehat <- y-yhat
  r2 <- sum((yhat-ybar)^2)/sum((y-ybar)^2)
  if (method == 1) cn=(t(w) %*% a1 %*% w - 2*t(a2) %*% w)/n
  if (method == 2) cn=(t(w) %*% a1 %*% w)/n
  list(betahat=betahat,w=w,yhat=yhat,ehat=ehat,r2=r2,cn=cn)
}
```

下面是关于上述函数输入以及输出的解释。

```
Inputs:
y            nx1   dependent variable
x            nxp   regressor matrix
method       1x1   set to 1 for Mallows model average estimates
                   set to 2 for Jackknife model average estimates
subset       1x1   set to 1 for pure nested subsets
                   set to 2 for all combinations of subsets
             mxp   input the (mxp) selection matrix, where m is the number of models
                    Example:
                      Suppose there are 3 candidate models.
                       Model 1: y=beta1*x1+beta2*x2+e
                       Model 2: y=beta1*x1+beta3*x3+e
                       Model 3: y=beta1*x1+beta2*x2+beta4*x4+e
                       Then subset <- matrix(c(1,1,1,1,0,1,0,1,0,0,0,1),3,4)

Outputs:
betahat      px1   parameter estimate
w            mx1   weight vector
yhat         nx1   fitted values
ehat         nx1   fitted residuals
r2           1x1   R-squared
```

```
cn          1x1    Value of Mallows criterion or Cross-Validation criterion
```

13.5.2 Python 语言实践

下面介绍怎么用 Python 语言实现上面提到的模型平均方法。

贝叶斯模型平均-线性模型

```
#载入需要用到的包
import numpy as np
import pandas as pd
from statsmodels.regression.linear_model import OLS
from statsmodels.tools import add_constant
import statsmodels.api as sm
from itertools import combinations
#读取数据
df = pd.read_csv('D:/Guber1999data.csv') #此处为数据存放的路径
```

接下来建立贝叶斯模型平均线性回归模型。首先定义 BMA 类:

```
#贝叶斯模型平均
class BMA:
    def __init__(self, y, X, **kwargs): #设置基本的变量
        self.y = y;self.X = X
        self.names = list(X.columns)
        self.nRows, self.nCols = np.shape(X)
        self.likelihoods = np.zeros(self.nCols)
        self.coefficients = np.zeros(self.nCols)
        self.probabilities = np.zeros(self.nCols)
        self.names = list(X.columns)
        #检查模型的最大尺寸 (模型中使用的预测变量的最大数量)
        if 'MaxVars' in kwargs.keys():
            self.MaxVars = kwargs['MaxVars']
        else:
            self.MaxVars = self.nCols
        if 'Priors' in kwargs.keys():#如果提供了先验参数, 则准备好这些参数
            if np.size(kwargs['Priors']) == self.nCols:
                self.Priors = kwargs['Priors']
            else:
                print("WARNING: Provided priors error.  Using equal priors instead.")
                print("The priors should be a numpy array of length equal to the
                number of regressor variables.")
                self.Priors = np.ones(self.nCols)
        else:
            self.Priors = np.ones(self.nCols)
```

建立 BMA_continue 子类继承前面的 BMA 父类，继续完善贝叶斯模型平均代码:

```
class BMA_continute(BMA):
  def __init__(self,y, X, **kwargs):
    super(BMA_continute,self).__init__(y, X, **kwargs)
  def fit(self):
        likelighood_sum = 0 #将所有模型的可能性之和初始化为0
        #为了方便在所有可能的模型中进行迭代，我们从遍历模型中元素的数量开始
        for num_elements in range(1,self.MaxVars+1):
            #列出所有这个尺寸的模型的索引集
            model_index_sets = list(combinations(list(range(self.nCols)),
            num_elements))
            #遍历给定大小的所有可能的模型
            for model_index_set in model_index_sets:
                #计算线性回归模型
                model_X = self.X.iloc[:,list(model_index_set)]
                model_regr = OLS(self.y, model_X).fit()
                #计算模型的似然（乘以先验）
                model_likelihood = np.exp(-model_regr.bic/2)*np.prod(
                self.Priors[list(model_index_set)])
                print("Model Variables:",model_index_set,
                "likelihood=", model_likelihood)
                likelighood_sum = likelighood_sum + model_likelihood
                #将此可能性(乘以先验)加到模型中每个变量的似然值
                for idx, i in zip(model_index_set, range(num_elements)):
                    self.likelihoods[idx] = self.likelihoods[idx] + model_likelihood
                    self.coefficients[idx] = self.coefficients[idx]
                    + model_regr.params[i]*model_likelihood
        #用贝叶斯定理除以分母来标准化概率
        self.probabilities = self.likelihoods / likelighood_sum
        self.coefficients = self.coefficients / likelighood_sum
        #返回新的BMA对象
        return self
  def summary(self):
    df = pd.DataFrame([self.names, list(self.probabilities),
    list(self.coefficients)], ["Variable Name", "Probability", "Avg.Coefficient"]).T
    return df
```

将数据代入，运用定义的 BMA_continute 类进行分析

```
X = df[["Spend", "StuTeaRat", "Salary", "PrcntTake"]]
y = df["SATT"]
result = BMA_continute(y, add_constant(X)).fit()
result.summary()
```

查看输出结果：

```
Model Variables: (0,) likelihood= 7.193780355030914e-126
Model Variables: (1,) likelihood= 1.8226074496883656e-152
...
Model Variables: (1, 2, 3, 4) likelihood= 3.4556604391428603e-134
Model Variables: (0, 1, 2, 3, 4) likelihood= 2.26958690901297e-110

  Variable Name Probability Avg. Coefficient
```

```
0          const          1.0      1015.358718
1          Spend     0.701118         8.388865
2     StuTeaRat     0.296262        -1.032458
3        Salary     0.300254         0.513476
4     PrcntTake          1.0        -2.822242
```

首先，我们观察到 PrcntTake 变量的概率是 1。这意味着不包括这个变量的模型的可能性很小。第二，关于支出和 SAT 成绩之间的关系的问题，我们看到，在其他条件相同的情况下，支出的增加对应于 SAT 成绩的增加。

贝叶斯模型平均 -logistic 模型

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.tools import add_constant
from itertools import combinations
from sklearn.model_selection import train_test_split#加载需要用到的包
df = pd.read_csv('D:/CHDdata.csv')#读取数据
df["famhist"] = (df["famhist"] == "Present")*1#将famhit转换为0(没有hist)和1(有hist)
```

接下来先建立 logistic 回归模型，并拟合数据。

```
# 建立logisitc回归并拟合数据
X = df.drop(["chd"], axis=1)
y = df["chd"]
log_reg = sm.Logit(y, add_constant(X)).fit()
log_reg.summary()
```

利用定义好的类，代入数据进行模型拟合，注意此处回归类型为‘Logit’:

```
result = BMA_continute(y,add_constant(X), RegType = 'Logit', Verbose=True).fit()
result.summary() #查看结果的概要
```

基于 AIC 与 BIC 信息准则

我们基于 AIC 和 BIC 信息准则对“Credit”数据集进行建模。

```
#载入需要用到的包
import itertools
import time
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from statsmodels.regression.linear_model import OLS
from statsmodels.tools import add_constant
from itertools import combinations
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
from sklearn.metrics import mean_squared_error
credit = pd.read_csv('D:/Credit.csv', usecols=list(range(1,12)))#读取数据
```

接下来，对将定性数据进行编码：

```
credit = pd.get_dummies(credit, columns=['Gender', 'Student','Married','Ethnicity'],
drop_first = True)
credit.head(3)
```

我们对定性数据进行编码，将女性编码为 1，男性编码为 0,；将学生编码为 1，非学生编码为 0；将已婚编码为 1，未婚编码为 0；将亚洲人编码为 1，非亚洲人编码为 0。查看“Credit”数据前 3 行的输出结果：

```
    Income  Limit  Rating  ...  Married_Yes  Ethnicity_Asian  Ethnicity_Caucasian
0   14.891   3606     283  ...            1                0                    1
1  106.025   6645     483  ...            1                1                    0
2  104.593   7075     514  ...            0                1                    0
[3 rows x 12 columns]
```

建立模型，并进行接下来的分析。

```
#拟合线性回归，返回RSS和R方
def fit_linear_reg(X,Y):
    model_k = linear_model.LinearRegression(fit_intercept = True)
    model_k.fit(X,Y)
    RSS = mean_squared_error(Y,model_k.predict(X)) * len(Y)
    R_squared = model_k.score(X,Y)
    return RSS, R_squared
```

接下来，选择子集 subset：

```
from tqdm import tnrange, tqdm_notebook
#变量初始化
Y = credit.Balance
X = credit.drop(columns = 'Balance', axis = 1);k = 11
RSS_list, R_squared_list, feature_list = [],[], []
numb_features = []
#在X中循环k = 1到k = 11个特征
for k in tnrange(1,len(X.columns) + 1, desc = 'Loop...'):
    ##循环所有可能的组合:从11个中选择k个
    for combo in itertools.combinations(X.columns,k):
        tmp_result = fit_linear_reg(X[list(combo)], Y)
        RSS_list.append(tmp_result[0])
        R_squared_list.append(tmp_result[1])
        feature_list.append(combo)
        numb_features.append(len(combo))
#存储在数据框中
df = pd.DataFrame({'numb_features': numb_features,'RSS': RSS_list, 'R_squared':
R_squared_list,'features':feature_list})
```

接下来为不同数量的特征找到最佳子集。

```
#为不同数量的特征找到最佳子集
df_min = df[df.groupby('numb_features')['RSS'].transform(min) == df['RSS']]
df_max = df[df.groupby('numb_features')
['R_squared'].transform(max) == df['R_squared']]
display(df_min.head(3))
display(df_max.head(3))
```

进一步，对上面输出的数据框的格式进行调整。

```
#调整上面输出的数据框
df['min_RSS'] = df.groupby('numb_features')['RSS'].transform(min)
df['max_R_squared'] = df.groupby('numb_features')['R_squared'].transform(max)
df.head()
```

再分别计算各模型的 RSS：

```
#初始化变量
Y = credit.Balance
X = credit.drop(columns = 'Balance', axis = 1)
k = 11
remaining_features = list(X.columns.values)
features = []
RSS_list, R_squared_list = [np.inf], [np.inf]
features_list = dict()
for i in range(1,k+1):
    best_RSS = np.inf
    for combo in itertools.combinations(remaining_features,1):
            #存储临时结果
            RSS = fit_linear_reg(X[list(combo) + features],Y)
            if RSS[0] < best_RSS:
                best_RSS = RSS[0]
                best_R_squared = RSS[1]
                best_feature = combo[0]
    #为下个循环更新变量
    features.append(best_feature)
    remaining_features.remove(best_feature)
    #将值储存起来
    RSS_list.append(best_RSS)
    R_squared_list.append(best_R_squared)
    features_list[i] = features.copy()
```

接下来为最优模型的 AIC、BIC 以及调整后 R^2 计算过程。

```
#定义用到的新数据框
df1 = pd.concat([pd.DataFrame({'features':features_list}),pd.DataFrame(
{'RSS':RSS_list, 'R_squared': R_squared_list})], axis=1, join='inner')
df1['numb_features'] = df1.index
#初始化用到的变量
m = len(Y)
p = 11
hat_sigma_squared = (1 / (m - p -1)) * min(df1['RSS'])
#计算AIC和BIC
```

```
df1['AIC'] = (1/(m*hat_sigma_squared)) * (df1['RSS'] + 2 * df1['numb_features'] *
hat_sigma_squared )
df1['BIC'] = (1/(m*hat_sigma_squared)) * (df1['RSS'] +  np.log(m) *
df1['numb_features'] * hat_sigma_squared )
df1['R_squared_adj'] = 1 - ( (1 - df1['R_squared'])  *(m - 1) /
(m - df1['numb_features'] -1))
df1 #输出数据框df1
```

基于 Mallows 与 Jackknife 准则

基于上面的代码以及“Credit”数据，进行这一部分的建模分析。

```
#计算
df1['C_p'] = (1/m) * (df1['RSS'] + 2 * df1['numb_features'] * hat_sigma_squared)
df1
```

首先进行数据的预处理

```
#设置X,Y变量，添加常数项
Y = credit.Balance
X = credit[['Ethnicity_Asian','Ethnicity_Caucasian']]
X = sm.add_constant(X)
X.head()
```

再根据上面提到的准则建模，代码如下。

```
#拟合模型并输出结果
model_1 = sm.OLS(Y, X)
result_1 = model_1.fit()
result_1.summary()
```

13.6 习题

1. 假设观察到随机样本 $\mathbf{Y}=(Y_1,\cdots,Y_n)^{\mathrm{T}}$，$\mathbf{X}=(\mathbf{X}_1,\cdots,\mathbf{X}_n)^{\mathrm{T}}$ 是 $n\times p$ 的设计矩阵，且满足线性模型

$$\mathbf{Y}=\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\varepsilon},$$

其中 $\boldsymbol{\varepsilon}\sim N(\mathbf{0},\sigma^2 I_n)$。对于参数 $\boldsymbol{\beta}$，σ^2，我们考虑标准正态伽马共轭先验，即

$$\boldsymbol{\beta}\sim N\left(\boldsymbol{\mu},\sigma^2 V\right),$$
$$\frac{\nu\lambda}{\sigma^2}\sim\chi^2_\nu,$$

其中 ν，λ，$\boldsymbol{\mu}$，V 是超参数且已知。若采用的候补模型是全模型，试证明 $p(D|M)=f(\mathbf{Y}|\mathbf{X})$ 服从非中心的 t 分布，自由度是 ν，期望是 $\mathbf{X}\boldsymbol{\mu}$，方差是 $[\nu/(\nu-2)]\lambda\left(I_n+\mathbf{X}V\mathbf{X}^{\mathrm{T}}\right)$。

2. 请推导 Jackknife 准则中的结论：$\widetilde{\boldsymbol{\mu}}_k = (D_k(H_k - I_n) + I_n)\mathbf{Y}$，$D_k$ 是对角阵，它的第 i 个对角元素为 $(1 - H_{k,ii})^{-1}$，而 $H_{k,ii}$ 是 H_k 的第 i 个对角元素。

（提示：使用 Sherman–Morrison 公式：$\left(A + \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}\right)^{-1} = A^{-1} - \dfrac{A^{-1}\boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}A^{-1}}{1 + \boldsymbol{V}^{\mathrm{T}}A^{-1}u}$，其中 A 是 $r \times r$ 可逆方阵，$\boldsymbol{U}, \boldsymbol{V}$ 分别是 $r \times 1$ 列向量，$1 + \boldsymbol{V}^{\mathrm{T}}A^{-1}\boldsymbol{U} \neq 0$ ）

3. 本数据来自 1991 年的收入和项目参与的调查（SIPP），由 9915 个观测样本和 44 个变量组成。我们对个人总财富（total wealth）变量感兴趣，因此选取为响应变量。基于 R 或 Python 语言，

（1）. 建立线性模型，并使用最小二乘方法估计参数，并计算 MSE: $\dfrac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)$。

（2）. 根据（1）中回归系数估计量的绝对值的大小对变量的重要性进行排序，并根据变量排序，构造 4 个候补模型，每个候补模型 11 个变量。分别使用基于 AIC、BIC、Mallows 准则和 Jackknife 准则的模型平均方法获得预测值 $\widehat{Y}_i$，并计算 MSE。

（3）. 对比（1）和（2）预测结果并解释。

第六部分

增量学习

第十四章　在线学习

增量学习（Incremental Learning）是一种持续学习的方法，模型可以在不断接收新数据的情况下进行更新和改进。与传统的离线学习不同，增量学习的目标是使模型能够适应新的数据，而不需要重新训练整个模型。与前面提到的机器学习方法相比，增量学习主要用于处理数据流或者动态环境下的学习，允许模型在接收到新数据时进行实时更新、在线学习。增量学习具有实时性、持续适应、适用于大规模数据和适应非平稳环境的优势，在在线广告、推荐系统、传感器网络、自适应控制等领域有广泛应用。

在某些情况下，可以将增量学习和前面的方法结合使用。例如，可以使用前面介绍的机器学习方法构建基础模型，然后通过增量学习来适应新的数据，以保持模型的实时性。

本章及下面两章将介绍三种主要的深度学习方法：在线学习（Online Learning）并行计算（Parallel Computing）和迁移学习（Transfer Learning）。

14.1　简介

随着科技的快速发展，我们迎来了大数据时代。通过对海量数据的处理与分析，可以发现巨大的社会价值，服务于生产生活。日益增长的海量数据往往呈现“流”特征，亦称“数据流”，此类大数据对存储承载力与计算机的计算性能等方面提出了更高的要求，传统的基于离线优化的机器学习算法面对极大的挑战。因此，在线学习的出现解决了这一问题。在线学习算法可被理解为：在重复决策的过程中，算法基于之前的经验以及当前的数据做出预测，以实现实时决策并在不断地对模型进行改进，来提高预测的精度。在线学习算法已经被广泛应用于数据流的分析中。

在线学习是基于机器学习的方法，对海量数据流进行训练，基于之前的经验（即保存下来的估计），来不断更新最佳的预测，而非传统地以批量处理的方式运行。在数据流框架下，传统的批量学习方法具有时间和空间成本高、效率低、扩展性差的特点，在线学习方法仅仅基于当前数据和历史估计结果进行更新，算法的效率和可扩展性大大提高。

接下来介绍一个在线学习的应用实例，比如在云计算中使用并行处理时，需要实现一种分配资源和调度任务执行顺序的机制。由于资源和任务在过程中不断更新，后台研究人员无法一次性得到所有的信息进行离线训练。因此可以利用在线学习算

法根据实际任务执行的更新信息动态调整资源分配，提高云的利用率。

在线学习适合大数据时代的分布式数据流的处理流程，目前，在线学习已经成为机器学习与人工智能领域的重要研究课题，其研究方向可能集中在对维度更敏感的高效在线图优化算法、理论方面上下限分析及最优解的寻找等方面。

下面以均值模型、线性模型、岭回归以及高维模型讲解在线学习算法，主要介绍 3 种方法：累积统计量，在线梯度下降以及正则化的在线梯度下降。

14.2 累积统计量在线学习

14.2.1 均值模型

平均值反映数据的位置，描述数据中心。对于一组 p 维的数据 $\boldsymbol{X}_1,\cdots,\boldsymbol{X}_n$，我们可构建均值模型为

$$\boldsymbol{X}_i=\boldsymbol{\mu}+\boldsymbol{\epsilon}_i,\ i=1,\cdots,n$$

模型通过误差 $\boldsymbol{\epsilon}_i$ 来描述以 $\boldsymbol{\mu}$ 为中心的样本值。对于参数 $\boldsymbol{\mu}$，用样本均值进行参数估计，即

$$\widehat{\boldsymbol{\mu}}=\bar{\boldsymbol{X}}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{X}_i$$

然而，对于实时获取的海量数据流来讲，我们会按照时间观测到数据 $\{\boldsymbol{X}_{ti}:i=1,\cdots,n_t,t=1,\cdots\}$, $\boldsymbol{X}_{ti}$ 代表第 t 批次数据流第 i 个样本，n_t 是第 t 批次数据的样本量。数据的更新意味着需要对模型进行不断地重复计算。受限于数据存储承载力与计算机的计算性能，传统的离线算法不适用于数据流的均值估计。

因此，我们可以构建在线学习均值模型，即在保存的前面算法结果的基础上仅对新数据进行处理，从而不断更新总体期望的估计结果。截止到第 t 批次，均值估计过程如下，

$$\widehat{\boldsymbol{\mu}}_t=\bar{\boldsymbol{X}}_t=\frac{1}{N_t}\sum_{j=1}^{t}\sum_{i=1}^{n_j}\boldsymbol{X}_{ji}=\frac{1}{N_t}\{\sum_{j=1}^{t-1}\sum_{i=1}^{n_j}\boldsymbol{X}_{ji}+\sum_{i=1}^{n_t}\boldsymbol{X}_{ti}\}$$

其中 $N_t=n_1+\cdots+n_t$。通过在线学习均值模型的构建，在已知 $\sum_{j=1}^{t-1}\sum_{i=1}^{n_j}\boldsymbol{X}_{ji}$（已保存，仅为 p 维向量）的基础上，只需计算 $\sum_{i=1}^{n_t}\boldsymbol{X}_{ti}$ 以及更新 N_t 的值，便可得到新数据下参数 $\boldsymbol{\mu}$ 的估计。这样一来，可从多输入源分布式地输入数据，算法的效率和可扩展性都大大提高。

在线更新样本均值算法流程可以表示如下：

表 14.1 均值模型在线学习算法

算法 1：均值模型累积统计量在线学习算法

输入： 初始化数据 $\boldsymbol{X}_{11},\cdots,\boldsymbol{X}_{1n_1}$

输出： 实时的样本均值 $\bar{\boldsymbol{X}}_t$

for $t=2$ to ∞ do

计算新的样本总数 $N_t=N_{t-1}+n_t$

计算并输出新的样本总和 $\sum\limits_{j=1}^{t}\sum\limits_{i=1}^{n_j}\boldsymbol{X}_{ji}=\sum\limits_{j=1}^{t-1}\sum\limits_{i=1}^{n_j}\boldsymbol{X}_{ji}+\sum\limits_{i=1}^{n_t}\boldsymbol{X}_{ti}$

计算新的样本均值 $\bar{\boldsymbol{X}}_t=\dfrac{1}{N_t}\sum\limits_{j=1}^{t}\sum\limits_{i=1}^{n_j}\boldsymbol{X}_{ji}$

当然，我们可以选择终止距离 ε，在某一时刻终止上述在线算法，得到收敛值，并作为最终的均值估计值。

14.2.2 线性模型

线性模型是一种描述因变量和自变量之间线性关系的模型，假设因变量 Y 和 p 个自变量 X_1,...,X_p 之间存在简单的线性关系，可构建如下形式：

$$Y=h_{\boldsymbol{\beta}}(\boldsymbol{X})=\beta_0+\beta_1X_1+\cdots+\beta_pX_p+\epsilon$$

对于实时获取的海量回归数据流来讲，我们假设观测到数据 $\{(\boldsymbol{X}_{ti}^{\mathrm{T}},Y_{ti})^{\mathrm{T}}:i=1,\cdots,n_t,t=1,\cdots\}$，令

$$\mathbf{Y}_t=\mathbf{X}_t\boldsymbol{\beta}+\boldsymbol{\epsilon}_t,$$

$$\mathbf{Y}_t=\begin{pmatrix}Y_{t1}\\Y_{t2}\\\vdots\\Y_{tn_t}\end{pmatrix}\quad\mathbf{X}_t=\begin{pmatrix}1&X_{t1,1}&\cdots&X_{t1,p}\\1&X_{t2,1}&\cdots&X_{t2,p}\\\vdots&\vdots&\ddots&\vdots\\1&X_{tn_t,1}&\cdots&X_{tn_t,p}\end{pmatrix}\quad\boldsymbol{\beta}=\begin{pmatrix}\beta_0\\\beta_1\\\beta_2\\\vdots\\\beta_p\end{pmatrix}\quad\boldsymbol{\epsilon}_t=\begin{pmatrix}\epsilon_{t1}\\\epsilon_{t2}\\\vdots\\\epsilon_{tn_t}\end{pmatrix}$$

截止到第 t 批次数据的到来，回归模型的参数估计过程如下，

$$\widehat{\boldsymbol{\beta}}_t=(\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j)^{-1}(\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j)=(\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j+\mathbf{X}_t^{\mathrm{T}}\mathbf{X}_t)^{-1}(\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j+\mathbf{X}_t^{\mathrm{T}}\mathbf{Y}_t)$$

通过在线学习线性模型的构建，在已知 $\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j$（已保存，仅为 $p+1\times p+1$ 维矩阵）和 $\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j$（已保存，仅为 $p+1$ 维向量）的基础上，只需计算 $\mathbf{X}_t^{\mathrm{T}}\mathbf{X}_t$ 以及 $\mathbf{X}_t^{\mathrm{T}}\mathbf{Y}_t$ 的值，便可得到新数据下参数 $\boldsymbol{\beta}$ 的估计。这样一来，可从多输入源分布式地输入数据，算法的效率和可扩展性都大大提高。

在线更新回归参数算法流程可以表示如下：

表 14.2 线性模型在线学习算法

算法 2：线性模型累积统计量在线学习算法

输入： 初始化数据 $\mathbf{X}_1$，$\mathbf{Y}_1$

输出： 实时的回归参数估计 $\widehat{\boldsymbol{\beta}}_t$

for $t=2$ to ∞ do

更新 $p\times p$ 矩阵 $\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j=\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j+\mathbf{X}_t^{\mathrm{T}}\mathbf{X}_t$

更新 p 维向量 $\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j=\sum_{j=1}^{t-1}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j+\mathbf{X}_t^{\mathrm{T}}\mathbf{Y}_t$

估计回归参数 $\widehat{\boldsymbol{\beta}}_t=(\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{X}_j)^{-1}(\sum_{j=1}^{t}\mathbf{X}_j^{\mathrm{T}}\mathbf{Y}_j)$

14.3 在线梯度下降

本章节介绍在线梯度下降算法（OGD；online gradient descent），并研究算法收敛性以及在线性模型和岭回归模型中的应用。

14.3.1 OGD 算法一般形式

假设第 t 次批量数据到来，在传统离线优化算法中，我们采用一般形式的损失函数去估计感兴趣参数 $\boldsymbol{\beta}$：

$$L_t(\boldsymbol{\beta})=\sum_{j=1}^{t}\ell_j(\boldsymbol{\beta})$$

其中 $\ell_j(\boldsymbol{\beta})=\ell(h_{\boldsymbol{\beta}}(\mathbf{X}_j),\mathbf{Y}_j)$ 代表第 j 批量数据下的损失函数，例如二次损失或负对数似然等，且满足 Lipschiz 连续性。然而，若采用上述形式损失函数更新参数，则需要使用数据流的所有样本进行估计，模型训练速度会随着时间的增加变得越来越慢。

接下来，我们采用在线梯度下降算法对其进行无约束优化，估计模型参数。梯度下降法是一种最优化算法，从几何意义上来讲，梯度是函数变化增加最快的方向，沿着梯度向量相反的方向，梯度减少更快，更加容易找到函数的最小值。根据损失函数的选取、更新参数时使用样本数量的不同，常见的梯度下降法分为批量梯度下降法 BGD、随机梯度下降法 SGD、小批量梯度下降法 MBGD 等。

对于日益增加的样本量，若使用传统离线的优化方法，虽然模型拟合精度得到提高，但是计算复杂度不断提高，训练速度会越来越来慢，并且历史数据也被重复使用。因此，Zinkevich [156] 提出了在线梯度下降（OGD；online gradient descent）的在线学习算法。在线梯度下降算法思想是在第 $t+1$ 阶段，仅使用第 t 阶段的数据对应的损失函数 $l_t(\boldsymbol{\beta})$ 进行梯度下降。计算出直接下降结果后，若可行域 $\mathcal{K}$ 受限，还需将梯度下降后的结果投影回可行空间。

在明确回归函数 $h_{\boldsymbol{\beta}}(\boldsymbol{X})$ 以及损失函数 $l_t(\boldsymbol{\beta})$ 的具体形式后，OGD 算法流程的一般框架如下所示：

表 14.3 OGD 算法

算法 3-1：OGD 算法

输入： 初始化参数 $\boldsymbol{\beta}_0$，终止距离 ε，学习速率 α, 可行域 $\mathcal{K}$

输出： 参数 $\boldsymbol{\beta}$ 的收敛值

for $t = 0$ to ∞ do

计算梯度下降点 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\beta}_t - \alpha l_t'(\boldsymbol{\beta}_t)$

计算投影点 $\boldsymbol{\beta}_{t+1} = \text{argmin}_{\boldsymbol{\beta}\in\mathcal{K}}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\beta}\|^2$

计算当前损失 $l_t(\boldsymbol{\beta}_{t+1})$ 及其梯度 $l_t'(\boldsymbol{\beta}_{t+1})$

如果前后步距离 $\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t\| < \varepsilon$

end for

14.3.2 OGD 算法收敛性分析

定义遗憾函数（Regret Function）：

$$R_T = \sum_{t=1}^{T} \left[\ell_t\left(\boldsymbol{\beta}_t\right) - \ell_t\left(\boldsymbol{\beta}^*\right)\right] \tag{14.3.1}$$

其中 $\boldsymbol{\beta}^* = \text{argmin}_{\boldsymbol{\beta}} \sum_{t=1}^{T} \ell_t\left(\boldsymbol{\beta}\right)$，$\boldsymbol{\beta}_t$ 表示第 t 轮迭代值。随着时间 T 的增加，遗憾函数的值接近于一个常量，则算法的收敛值 $\boldsymbol{\beta}_T$ 与最优的 $\boldsymbol{\beta}^*$ 是一致的，即可证明此在线学习方法是有效的。

接下来, 我们分析 OGD 算法的收敛性。根据投影点的对应关系, 我们可以得到

$$\|\boldsymbol{\beta}_{t+1}-\boldsymbol{\beta}^*\|^2 \leq \|\boldsymbol{\theta}_{t+1}-\boldsymbol{\beta}^*\|^2 = \|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\|^2 - 2\alpha\left\langle \ell_t'\left(\boldsymbol{\beta}_t\right),\left(\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right)\right\rangle + \alpha^2\left\|\ell_t'\left(\boldsymbol{\beta}_t\right)\right\|^2$$

由此式变形后, 可以得到

$$\left\langle \ell_t'\left(\boldsymbol{\beta}_t\right),\left(\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right)\right\rangle \leq \frac{1}{2\alpha}\left(\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\|^2 - \|\boldsymbol{\beta}_{t+1}-\boldsymbol{\beta}^*\|^2\right) + \frac{\alpha}{2}\left\|\ell_t'\left(\boldsymbol{\beta}_t\right)\right\|^2$$

另一方面, 根据损失函数的凸性可以得到

$$R_T = \sum_{t=1}^{T}\left[\ell_t\left(\boldsymbol{\beta}_t\right) - \ell_t\left(\boldsymbol{\beta}^*\right)\right] \leq \sum_{t=1}^{T}\left\langle \ell_t'\left(\boldsymbol{\beta}_t\right),\left(\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right)\right\rangle$$

合并上述两式可得

$$R_T \leq \frac{1}{2\alpha}\|\boldsymbol{\beta}_1-\boldsymbol{\beta}^*\|^2 + \frac{\alpha}{2}\sum_{t=1}^{T}\left\|\ell_t'\left(\boldsymbol{\beta}_t\right)\right\|^2$$

再由损失函数的 Lipschiz 连续性 (假设常数为 L) 可知, $\forall t, \left\|\ell_t'\left(\boldsymbol{\beta}_t\right)\right\| \leq L$, 因此得到 R_T 上界的最终形式如下

$$R_T \leq \frac{1}{2\alpha}\|\boldsymbol{\beta}_1-\boldsymbol{\beta}^*\|^2 + \frac{\alpha L^2 T}{2}$$

当且仅当 $\frac{1}{2\alpha}\|\boldsymbol{\beta}_1-\boldsymbol{\beta}^*\|^2 = \frac{\alpha L^2 T}{2}$ 时, 即 $\alpha = \frac{\|\boldsymbol{\beta}_1-\boldsymbol{\beta}^*\|}{L\sqrt{T}}$ 时, 式右边取最小值, 即 $R_T \leq \|\boldsymbol{\beta}_1-\boldsymbol{\beta}^*\| L\sqrt{T}$。因此, 我们推出 OGD 算法的性能最差情况是 $\mathcal{O}(\sqrt{T})$。

14.3.3 线性模型的 OGD 算法

注意到上述 OGD 算法是对于一般模型而言的，实现了在线梯度下降的算法功能，具有一定的包容性与可扩展性，相比于传统离线优化算法大大减少了计算的复杂度, 因为损失函数 $l_t(\boldsymbol{\beta})$ 仅使用第 t 阶段的数据。借助 OGD 的算法思想，假设第 t 阶段的二次损失函数为

$$l_t(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}\|_2^2$$

对应的梯度是

$$l_t'(\boldsymbol{\beta}) = -\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta})$$

在基于二次损失构造线性模型在线学习算法时，我们不妨在第 t 阶段，选取第 $t-1$ 阶段求得的 $\boldsymbol{\beta}_{t-1}$ 计算梯度值，这样不仅能通过新数据不断迭代更新参数估计，还减少了第 t 阶段模型拟合带来的计算复杂度。因此第 t 阶段的梯度的计算公式如下:

$$l_t'(\boldsymbol{\beta}_{t-1}) = -\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}_{t-1})$$

另外，考虑到线性模型的可行域为 $\mathcal{R}^{p+1}$，梯度下降后无需将中间迭代结果投影回可行域。因此基于二次损失的线性模型在线学习算法流程可以表示如下：

表 14.4 OGD 算法

算法 3-2：OGD 算法-基于二次损失

输入： 初始化参数 $\boldsymbol{\beta}_0$，终止距离 ε，学习速率参数 λ

输出： 参数 $\boldsymbol{\beta}$ 的收敛值

for $t = 1$ to ∞ do

计算第 t 阶段梯度 $l_t'(\boldsymbol{\beta}_{t-1}) = -\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}_{t-1})$

更新学习速率 $\alpha = \lambda t^{-\frac{1}{2}}$

更新参数 $\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \alpha l_t'(\boldsymbol{\beta}_{t-1})$

如果前后步距离 $\|\alpha l_t'(\boldsymbol{\beta}_{t-1})\| < \varepsilon$

end for

14.3.4 岭回归模型的 OGD 算法

岭回归方法是一种解决回归数据共线性问题的监督学习方法。其使用的目标函数是在二次损失的基础上，增加 L_2 正则项，详细介绍参考回归模型章节。下面介绍基于二次损失的岭回归在线学习算法。假设观测到第 t 次批量数据的到来，在传统岭回归模型离线优化算法中，我们采用如下二次损失函数去估计感兴趣参数 $\boldsymbol{\beta}$：

$$l_t(\boldsymbol{\beta}) = \frac{1}{2n_t}\|\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2}\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\beta}$$

对应的梯度是

$$l_t'(\boldsymbol{\beta}) = -\frac{1}{n_t}\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}$$

考虑到模型的训练速度以及计算的复杂度，不妨借助上一节线性模型在线学习的思想，第 t 阶段对应的梯度函数，选取第 $t-1$ 阶段求得的 $\boldsymbol{\beta}_{t-1}$ 进行下一步的迭代，这样不仅能通过新数据不断迭代更新参数估计，还不用基于当前数据拟合模型，即此时的梯度是

$$l_t'(\boldsymbol{\beta}_{t-1}) = -\frac{1}{n_t}\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t - \mathbf{X}_t\boldsymbol{\beta}_{t-1}) + \lambda\boldsymbol{\beta}_{t-1}$$

以上岭回归模型的在线学习算法流程可以表示如下：

表 14.5 岭回归在线学习算法

算法 4：岭回归在线学习算法-基于二次损失

输入： 初始化参数 $\boldsymbol{\beta}_0$，终止距离 ε，学习速率 α，调谐参数 λ

输出： 参数 $\boldsymbol{\beta}$ 的收敛值

for $t=1$ to ∞ do

计算第 t 阶段梯度 $l_t'(\boldsymbol{\beta}_{t-1})=-\frac{1}{n_t}\mathbf{X}_t^{\mathrm{T}}(\mathbf{Y}_t-\mathbf{X}_t\boldsymbol{\beta}_{t-1})+\lambda\boldsymbol{\beta}_{t-1}$

更新参数 $\boldsymbol{\beta}_t=\boldsymbol{\beta}_{t-1}-\alpha l_t'(\boldsymbol{\beta}_{t-1})$

如果前后步距离 $\|\alpha l_t'(\boldsymbol{\beta}_{t-1})\|<\varepsilon$

end for

14.4 基于正则化的在线梯度下降

14.4.1 FTL 算法

FTL（Follow The Leader）算法在损失函数强凸的情形下能有效解决在线优化问题，其算法的思想是通过最小化累计损失来更新参数，公式如下：

$$\boldsymbol{\beta}_{t+1}=\mathrm{argmin}_{\boldsymbol{\beta}}\sum_{j=1}^{t}l_j(\boldsymbol{\beta})$$

由归纳法得 FTL 算法的遗憾函数上限为：

$$R_T\leq\sum_{t=1}^{T}(l_t(\boldsymbol{\beta}_t)-l_t(\boldsymbol{\beta}_{t+1}))$$

可见，在损失函数强凸的情形下，$\boldsymbol{\beta}_t$ 会收敛到 $\boldsymbol{\beta}^*$。但是，在在线线性优化 (Online Linear Optimization) 问题中不一定成立。接下来，我们考虑一维的感兴趣参数 β，损失函数关于 β 是线性的，假设为 $l_t(\beta)=G_t\beta$，$\beta\in[-1,1]$。此时 G_t 是损失函数的梯度，其取值为：

$$G_t=\begin{cases}-0.5, & t{=}1\\ 1, & t\text{ 为偶数}\\ -1, & t{>}1\text{ 且 }t\text{ 为奇数}\end{cases}$$

由参数更新公式得：

$$\beta_{t+1}=\mathrm{argmin}_{\beta}\sum_{j=1}^{t}G_j\beta$$

使用 FTL 算法则会出现参数震荡现象：

t	G_t	$\beta_{t+1}=\text{argmin}_\beta \sum_{j=1}^{t} G_j\beta$
1	-0.5	$\text{argmin}_\beta\{-0.5\beta\}$=1
2	1	$\text{argmin}_\beta\{-0.5\beta+1\beta\}$=-1
3	-1	$\text{argmin}_\beta\{-0.5\beta+1\beta-1\beta\}$=1
4	1	$\text{argmin}_\beta\{-0.5\beta+1\beta-1\beta+1\beta\}$=-1
...	...	...

经过计算，β 最终的“收敛值”在 1 和-1 之间震荡，并不收敛，FTL 算法在这个的在线线性优化问题上失效。因此，FTL 算法在损失函数强凸的情形下虽然有效，但在一般凸函数情形下不满足遗憾函数次线性，导致参数不收敛，可见算法不稳定。

14.4.2 FTRL 算法

为了解决上述问题，FTRL（Follow the Regularized Leader）算法在 FTL 算法的基础上增加正则化项 $P_\alpha(\boldsymbol{\beta})$ 来使算法更稳定。FTRL 算法的参数更新公式如下：

$$\boldsymbol{\beta}_{t+1}=\text{argmin}_{\boldsymbol{\beta}}\{\sum_{j=1}^{t} l_j(\boldsymbol{\beta})+P_\alpha(\boldsymbol{\beta})\}$$

容易证明 FTRL 算法的遗憾函数上限为：

$$R_T\leq P_\alpha(\boldsymbol{\beta}^*)+\sum_{t=1}^{T}[l_t(\boldsymbol{\beta}_t)-l_t(\boldsymbol{\beta}_{t+1})]$$

针对上述的在线线性优化问题，不妨令 $P_\alpha(\boldsymbol{\beta})=\frac{1}{2\alpha}\|\boldsymbol{\beta}\|_2^2$，求导得出参数更新公式为：$\boldsymbol{\beta}_{t+1}=\boldsymbol{\beta}_t-\alpha\boldsymbol{G}_t$，注意此时我们考虑的是多维的参兴趣参数 $\boldsymbol{\beta}$，$\boldsymbol{G}_t$ 是梯度向量。

FTRL 算法的遗憾函数上限为：

$$R_T\leq\frac{1}{2\alpha}\|\boldsymbol{\beta}^*\|_2^2+\alpha\sum_{t=1}^{T}\|\boldsymbol{G}_t\|_2^2$$

若参数 $\boldsymbol{\beta}\in\{\boldsymbol{\beta}:\|\boldsymbol{\beta}\|_2\leq B\}$，损失函数 l 满足 L-Lipschitz 条件，即 $\frac{1}{T}\sum_{t=1}^{T}\|\boldsymbol{G}_t\|_2^2\leq L^2$，则

$$R_T\leq\frac{1}{2\alpha}B^2+\alpha TL^2$$

当且仅当 $\frac{1}{2\alpha}B^2=\alpha TL^2$，即 $\alpha=\frac{B}{L\sqrt{2T}}$ 时，上式右边取得最小值。因此带入 α，则遗憾函数满足

$$R_T\leq BL\sqrt{2T}$$

可以发现：

$$\lim_{T\to\infty}\frac{dR_T}{dT}=\lim_{T\to\infty}\frac{BL}{\sqrt{2T}}=0$$

因此，$\boldsymbol{\beta}_t$ 收敛，则 FTRL 为有效算法，可见其相比于 FTL 算法更加稳定。

14.4.3 FTRL-Proximal 算法

FTRL-Proximal 算法是一种用于点击率预估的在线机器学习系统的核心算法，其可以看做 RDA（Regularized Dual Averaging Algorithm）算法与 FOBOS（Forward Backward Splitting）算法的结合体，在 FTL 算法的基础上通过增加 L_1、L_2 正则项的方式来兼顾算法的稀疏性与精确度。

为了进一步提高算法的稀疏性，FTRL-Proximal 算法在 FTRL 算法的基础上添加 L_1 正则项。FTRL-Proximal 算法使用如下损失函数更新参数：

$$\boldsymbol{\beta}_{t+1}=\operatorname{argmin}\left(\sum_{j=1}^{t}\boldsymbol{G}_j^{\mathrm{T}}\boldsymbol{\beta}+\frac{1}{2}\sum_{j=1}^{t}\sigma_j\|\boldsymbol{\beta}-\boldsymbol{\beta}_j\|_2^2+\lambda\|\boldsymbol{\beta}\|_1\right)$$

其中 $\sigma_j=\frac{1}{\alpha_j}-\frac{1}{\alpha_{j-1}}$，$\alpha_t$ 是 t 时刻的学习率，λ 是 L_1 正则化调谐参数，$\boldsymbol{\beta}$ 为感兴趣参数。值得注意的是，$\sum_{j=1}^{t}\boldsymbol{G}_j^{\mathrm{T}}\boldsymbol{\beta}$ 保证向正确的方向更新，而使用历史累计梯度保证不会过早地将重要特征的参数约束为 0。$\frac{1}{2}\sum_{j=1}^{t}\sigma_j\|\boldsymbol{\beta}-\boldsymbol{\beta}_j\|_2^2$ 要求新产生的参数不要偏离历史参数太远，即参数更新不要太激进，$\lambda\|\boldsymbol{\beta}\|_1$ 保证解的稀疏性。

在参数更新过程中，我们将特征参数的各个维度拆解成独立的标量最小化问题，从而简化算法，具体计算过程如下：

令 $\boldsymbol{Z}_t=\sum_{j=1}^{t}\boldsymbol{G}_j-\sum_{j=1}^{t}\sigma_j\boldsymbol{\beta}_j$，$\sigma_t=\frac{1}{\alpha_t}-\frac{1}{\alpha_{t-1}}$，我们不妨构造函数 $F(\boldsymbol{\beta})$ 为：

$$F(\boldsymbol{\beta})=\boldsymbol{Z}_t^{\mathrm{T}}\boldsymbol{\beta}+\frac{1}{2\alpha_t}\|\boldsymbol{\beta}\|_2^2+\lambda\|\boldsymbol{\beta}\|_1+(\text{const})$$

即：$\boldsymbol{\beta}_{t+1}=\operatorname{argmin}_{\boldsymbol{\beta}}F(\boldsymbol{\beta})$。下面求解使得 $F(\boldsymbol{\beta})$ 最小的 $\boldsymbol{\beta}$。由于 $\|\boldsymbol{\beta}\|_1$ 在零点处不可导，我们考虑使用次梯度方法。

$$\partial_{\boldsymbol{\beta}}F(\boldsymbol{\beta})=\boldsymbol{Z}_t+\frac{\boldsymbol{\beta}}{\alpha_t}+\lambda\partial_{\boldsymbol{\beta}}\|\boldsymbol{\beta}\|_1$$

令其次梯度为 0，并根据条件限制得到各维度参数更新公式如下：

$$\beta_{t+1,k} = \begin{cases} (\lambda - Z_{t,k})\alpha_t, & Z_{t,k} > \lambda \\ 0, & |Z_{t,k}| \le \lambda \\ (-\lambda - Z_{t,k})\alpha_t, & Z_{t,k} < -\lambda \end{cases} \tag{14.4.1}$$

另外，FTRL-Proximal 算法考虑了数据在不同维度上的特征分布的不均匀性，建议每个维度采用的学习率 α_t 是不一样的并改为如下形式的 $\alpha_{t,k}$：

$$\alpha_{t,k} = \frac{\omega}{\gamma + \sqrt{\sum_{j=1}^{t} G_{j,k}^2}}$$

其中，$\alpha_{t,k}$ 为 t 时刻 k 维度的学习率，ω，γ 为超参数，$G_{t,k}$ 为 t 时刻 k 维度的损失函数的梯度。

FTRL-Proximal 算法流程如下：

表 14.6 FTRL-Proximal 算法

算法 5：FTRL-Proximal 算法

输入： 超参数 ω，γ，正则化参数 λ, 总时间 T，以及初始化参数 $\boldsymbol{Z}_0$，$\boldsymbol{\alpha}_0$，$\boldsymbol{\beta}_1$

输出： $\boldsymbol{\beta}_T$

for $t = 1$ to T do

计算损失梯度向量 $\boldsymbol{G}_t = \boldsymbol{G}_t(\boldsymbol{\beta}_t)$

for $k = 1$ to p do

计算每一维度的学习速率 $\alpha_{t,k} = \dfrac{\omega}{\gamma + \sqrt{\sum_{j=1}^{t} G_{j,k}^2}}$

end for

计算对角阵 $\boldsymbol{\sigma}_t = \mathrm{diag}\{\dfrac{1}{\boldsymbol{\alpha}_t} - \dfrac{1}{\boldsymbol{\alpha}_{t-1}}\}$

计算向量 $\boldsymbol{Z}_t = \boldsymbol{Z}_{t-1} + \boldsymbol{G}_t - \sigma_t \boldsymbol{\beta}_t$

使用 (14.4.1) 更新参数下一步迭代值 $\boldsymbol{\beta}_{t+1}$

end for

14.5 在线学习实践

14.5.1 R 语言实践

在线均值模型

下面介绍如何使用 R 语言实现均值模型的在线学习算法。首先我们要随机生成数组 X，并计算其初始样本均值。

```
set.seed(12345)#设定随机数种子
N <- 100
X <- rnorm(N)#随机生成生态分布后抽取N个值
X.mean <- mean(X)#计算初始样本均值
```

如果查看初始的样本均值，可输入如下命令:

```
X.mean
[1]0.2451972
```

其次，在循环下，我们可以构造得到新的数据，随之更新样本数量和样本总和，从而计算新的样本均值。

```
b <- 100
X.sum <- N*X.mean#计算初始样本总和
for(j in 2:b){
  x <- rnorm(j*b)#随机产生j×b个新样本
  N <- N + length(x)#更新样本数量
  X.sum <- X.sum + sum(x)#更新样本总和
  X.mean <- X.sum/N#更新样本均值
}
```

如果查看最终的样本均值，可输入如下命令:

```
X.mean
[1]-6.21109e-05
```

可以从 $\bar{X}$ 的结果对比中看出：经过数据的更新，样本均值从 0.2451972 大幅减少为-6.21109e-05。

在线线性模型

下面介绍如何使用 R 语言实现线性模型在线学习的算法功能。首先我们要随机生成设计矩阵 X 和观测向量 Y。

```
N <- 2000
p <- 1
X <- matrix(rnorm(N*p), ncol=p)#生成设计矩阵X
beta <- c(-1,1)
eps <- rnorm(N)#随机生成扰动
y <- cbind(1, X) %*% beta + eps#生成观测向量Y
```

其次，初始化参数 $\boldsymbol{\beta}$ 和学习速率 α，$\boldsymbol{\beta}_{\text{final}}$ 可在循环中与 $\boldsymbol{\beta}_{\text{initial}}$ 交换赋值实现迭代。

```
beta.initial <- rep(0, p+1)#初始化t阶段参数beta
alpha <- 0.5#初始化学习速率
beta.final <- matrix(0, nrow=N, ncol=p+1)#初始化储存参数更新值矩阵
```

接着，利用循环不断根据新数据计算梯度、更新参数 $\boldsymbol{\beta}$ 的估计。另外，在优化的过程中还需对学习速率 α 不断进行更新。

```
for(j in 1:N){
  grad <- -c(1, X[j,])*(y[j] - as.numeric(c(1, X[j,]) %*% beta.initial))#计算梯度
  alpha <- 0.5*j^(-1/2)#更新学习速率
  beta.final[j,] <- beta.initial - alpha * grad#更新参数
  beta.initial <- beta.final[j,]#交换参数
}
```

最终，通过可视化方法观察参数迭代收敛过程。

```
#可视化参数迭代过程
png("R-linear model.png",units="in", width=8, height=6,res=2000)
x <- c(1:N)
plot(x, beta.final[,1], type='p', col='blue',cex=0.1,pch=20, ylim=c(-1.5, 1.5),
xlab ='epoch', ylab='parameters')
abline(h=-1,lty=2, col='blue')
par(new = TRUE)
plot(x, beta.final[,2], type='p', col='red',cex=0.1,pch=20,axes = FALSE,
xlab='',ylab='')
abline(h=1,lty=2, col='red')
dev.off()
```

图14.1表示迭代过程中参数值的变化。我们可以从图像中观察到参数逐渐收敛。

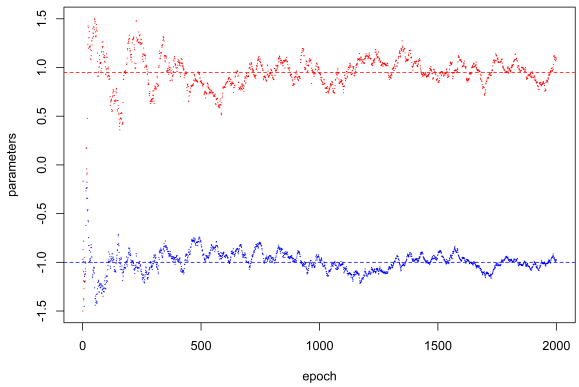


图 14.1 在线线性模型参数迭代值的变化

在线岭回归

下面介绍如何使用 R 语言实现岭回归模型的在线学习算法。

首先，我们定义一个读取数据的方程，需要分别提取文件中的自变量数据和因变量数据。另外，需要注意的是，我们在函数中将数据进行标准化处理，旨在提高运行速度。

```
## 用于读取数据的函数
read_data <- function(fname, sc){
  data <- read.csv(file = fname, head = TRUE, sep = ",")
  nr = dim(data)[1]#提取数据行数
  nc = dim(data)[2]#提取数据列数
  x = data[1:nr, 1:(nc-1)]#提取自变量数据
  y = data[1:nr, nc]#提取因变量数据
  if (isTRUE(sc)){
    x = scale(x)#将数据进行标准化，提高运算速度
    y = scale(y)
  }
  return (list("x" = x, "y" = y))
}
```

其次，我们定义一个用于计算损失函数值与参数迭代矩阵的函数。

```
## 用于计算损失函数值与参数迭代矩阵的函数
sgd_train <- function(train_x, train_y, lambda, alpha, epsilon, max_epoch){
  Phi <- as.matrix(cbind('X0'= 1, train.data))#构造设计矩阵
  ## 计算SGD的最大迭代次数
  train_len = dim(train_x)[1]
  tau_max = max_epoch * train_len
  ## 初始化参数矩阵
  beta <- matrix(, nrow=tau_max, ncol=ncol(Phi))#初始化参数迭代矩阵
  obj_func_val <- matrix(, nrow=tau_max, ncol=1)#初始化损失函数值的矩阵
  ## 第一次迭代
  tau = 1#tau为迭代次数
  set.seed(12345)#设定种子
  ## 生成第一次迭代参数
  beta[1,] <- runif(ncol(Phi))
  ## 计算第一次迭代损失函数值
  obj_func_val[tau,1] = .5 * (train_y[1] - t(beta[tau,]) %*% Phi[1,])^2
  + .5 * lambda * t(beta[tau,]) %*% beta[tau,]
  ## 循环迭代开始
  while (tau <= tau_max){
    if (obj_func_val[tau,1] <= epsilon) {break}#检查终止条件
    ## sample函数用于重排数据
    train_index <- sample(1:train_len, train_len, replace = FALSE)
    ## 循环遍历每个数据点
    for (i in train_index) {
      tau <- tau + 1
      ##检查终止条件
      if (tau > tau_max) {break}
      y_pred <- t(Phi[i,]) %*% beta[tau-1,]#计算预测值y_pred
      grad <- -(train_y[i] - y_pred) * Phi[i,]#计算梯度
      beta[tau,] <- beta[tau-1,] * (1-alpha * lambda) - alpha * grad#更新参数
      obj_func_val[tau,1] = .5 * (train_y[i] - t(beta[tau,]) %*% Phi[i,])^2
      + .5 * lambda * t(beta[tau,]) %*% beta[tau,]#计算该阶段的损失函数值
```

```
    }
  }
  return(list('vals' = obj_func_val, 'beta' = beta))
}
```

接下来，我们将开始进行试验。首先我们需要加载 R 语言中相关库。

```
## 加载库
library(ggplot2)
library(reshape2)
library(mvtnorm)
```

其次，我们要添加上述定义函数的路径并用 source 函数将上述两个函数加载到主函数中。

```
## 添加路径
path="C:/Users/Administrator/Desktop"
setwd(path)
## source函数执行R脚本
source('read_data.R')
source('sgd_train.R')
```

接着，我们应运用 read_data 函数加载训练集数据及训练标签。training datasets 数据集由 $n \times p$ 维自变量 x 与 $n \times 1$ 维因变量 y 组成。

```
## 加载训练集数据和训练标签
train.data <- read_data('training_datasets.csv',TRUE)$x
train.label <- read_data('training_datasets.csv',TRUE)$y
```

此外，我们还需对最大循环次数、终止阈值、学习效率、正则化参数进行设定。

```
## 设置最大的循环次数
max_epoch = 15
## 设定相关参数
epsilon = .001#终止阈值
alpha = .04#学习效率
lambda= 0.3#正则化参数
```

接着，我们可以用 sgd_train 函数计算迭代过程中损失函数值以及参数 $\boldsymbol{\beta}$ 的迭代矩阵。

```
## 函数返回损失函数值以及参数迭代矩阵
train_res = sgd_train(train.data, train.label, lambda, alpha, epsilon, max_epoch)
```

最后，我们画出迭代过程中参数迭代值的变化来评估模型的效果。

```
## 可视化迭代过程中参数迭代值的变化
plot(train_res$beta[,1], type='p', col='blue',cex=0.1,pch=20, ylim=c(-2,2),
xlab ='epoch', ylab='parameters')
abline(h=1.5,lty=2, col='red')
par(new = TRUE)
plot(train_res$beta[,2], type='p', col='red',cex=0.1,pch=20,ylim=c(-2,2),
axes = FALSE,xlab='',ylab='')
```

```
abline(h=-1.5,lty=2, col='blue')
```

图14.2表示迭代过程中参数迭代值的变化。我们可以从图像中观察到参数收敛速度很快。

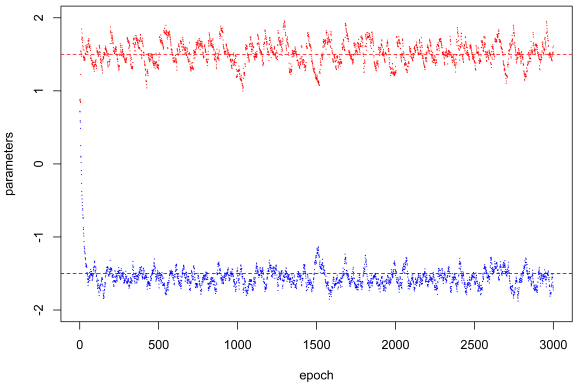


图 14.2 在线岭回归模型参数迭代值的变化

在线高维模型

下面介绍如何使用 R 语言实现高维模型的在线学习算法。首先，我们应生成符合逻辑回归的数据集，并进行数据读取。

```
## 生成逻辑回归数据集
set.seed(12345)#设定随机数种子
N <- 8000
p <- 1
beta <- c(-1,1)
eps <- rnorm(N)
X <- as.matrix(cbind('X0'= 1, matrix(rnorm(N*p), ncol=p)))#构造设计矩阵
y <- X %*% beta + eps#构造观测向量
logistic_y <- 1/(1+exp(-y))#逻辑回归
l <- length(logistic_y)
for(i in 1:l){
  if(logistic_y[i]<=0.5)#按0.5划分二分类
    logistic_y[i]=0
  else
    logistic_y[i]=1
}
## 读取数据
train.data <- X
train.label <- logistic_y
T <- length(train.label)#计算样本大小
N <- dim(train.data)[2]#计算参数向量的维度
```

其次，初始化超参数 α、$\boldsymbol{\beta}$，正则化参数 λ，并初始化所需的各参数矩阵，用于存储迭代过程中生成的值。

```
## 初始化参数
alpha=0.1; beta=1#超参数
lambda=0.01#正则化参数
W <- matrix(0, nrow=T, ncol=N)#初始化参数迭代矩阵
Z <- matrix(0, nrow=T, ncol=N)#初始化中间矩阵
eta <- matrix(0, nrow=T, ncol=N)#初始化学习率矩阵
g <- matrix(0, nrow=T, ncol=N)#初始化损失梯度矩阵
sigma <- matrix(0, nrow=T, ncol=N)#初始化学习速率矩阵
```

最后，通过 for 循环对每个阶段的参数进行更新迭代。

```
for(t in 1:T){
  #利用for循环对每个维度更新参数W
  if(t != 1){
    for(i in 1:N){
      if(Z[t-1,i] > lambda){
        W[t,i] <- (lambda-Z[t-1,i])*eta[t-1,i]
      }else if(Z[t-1,i] < -lambda){
        W[t,i] <- (-lambda-Z[t-1,i])*eta[t-1,i]
      }else if(abs(Z[t-1,i]) <= lambda){
        W[t,i] = 0
      }
    }
  }
  #计算预测值pt
  tmp <- train.data[t,]%*%W[t,]
  pt <- 1/(1+exp(-tmp))#运用sigmoid函数计算
  #利用for循环对每个维度进行更新各参数值
  for(r in 1:N){
    g[t,r] <- (pt-train.label[t])*train.data[t,r]#计算损失梯度
    #计算g_{1:t}
    sum=0
    for(s in 1:t)
    {
      sum=sum+g[s,r]*g[s,r]
    }
    eta[t,r]=alpha/(beta+sqrt(sum))#计算学习率矩阵
    if(t != 1){
      sigma[t,r] <- 1/eta[t,r] - 1/eta[t-1,r]#计算学习速率
      Z[t,r]<- Z[t-1,r]+g[t,r]-sigma[t,r]%*%W[t,r]
    }
  }
}
```

最终，我们通过可视化方法观察参数迭代收敛过程。

```
png("FTRL.png",units="in", width=8, height=6,res=2000)
x <- c(1:T)
plot(x, W[,1], type='p', col='blue',cex=0.1,pch=20, ylim=c(-2,2),
xlab ='epoch', ylab='parameters')
```

```
abline(h=1.8,lty=2, col='red')
par(new = TRUE)
plot(x, W[,2], type='p', col='red',cex=0.1,pch=20,ylim=c(-2,2),
axes = FALSE,xlab='',ylab='')
abline(h=-1.8,lty=2, col='blue')
dev.off()
```

图14.3表示迭代过程中参数迭代值的变化。我们可以从图像中观察到，参数逐渐收敛。

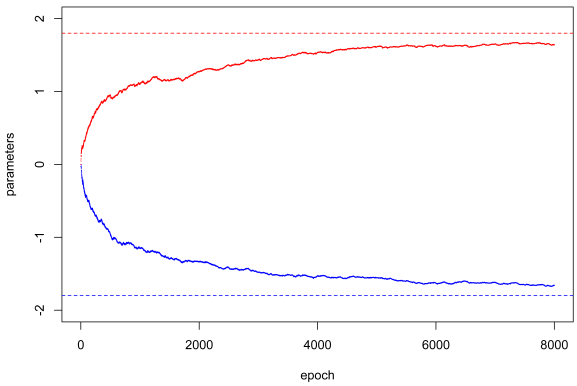


图 14.3 在线高维模型参数迭代值的变化

14.5.2 Python 语言实践

在线均值模型

下面介绍如何使用 Python 语言实现均值模型的在线学习算法。首先导入实验过程中需要的库并设定随机数种子。

```
#加载库
import numpy as np
import random
#设定随机数种子
random.seed(12345)
np.random.seed(12345)
```

其次，构造初始数据集并计算初始样本均值。

```
#构造初始数据集
X = np.random.randn(100)  #生成多个服从标准正态分布的随机数组
X_length = len(X)         #计算初始样本量
X_mean = sum(X) / len(X)  #计算初始样本均值
```

```
X_sum = sum(X)              #计算初始样本总和
b = 100    #设置迭代次数
print("初始样本均值为: %10.10f" %X_mean)
```

其次，在循环下，我们可以构造得到新的数据，随之更新样本数量和样本总和，从而计算新的样本均值。

```
#利用for循环更新样本均值
for i in range(2,100):
    x = np.random.randn(i * b)      #随机生成新样本
    X_length = X_length + len(x)    #更新样本容量
    X_sum = X_sum + sum(x)          #更新样本总和
    X_mean = X_sum / X_length       #更新样本均值
print("更新后样本均值为: %10.10f" %X_mean)
```

可以从 $\bar{X}$ 的结果对比中看出，经过数据的更新，样本均值从 0.0336143882 大幅减少为 0.0006315946。

在线线性模型

下面介绍如何使用 Python 语言实现线性模型的在线学习算法。首先导入实验过程中需要的库并设定随机数种子。

```
#加载库
import numpy as np
import random
import matplotlib.pyplot as plt
#设定随机数种子
random.seed(12345)
np.random.seed(12345)
```

其次，构造数据集，随机生成设计矩阵 X 和观测向量 Y。

```
#构造数据集
N = 2000    #设置样本容量
p = 2       #设置参数个数
X1 = np.random.randn(N * (p-1), p-1)
X = np.hstack((np.ones((N, 1)), X1))      #生成设计矩阵
beta = np.array([1, -1])     #设置参数
eps = np.random.randn(N)     #随机生成扰动
y = np.dot(X, beta) + eps    #生成观测向量
```

接着，初始化参数 $\boldsymbol{\beta}$ 和学习速率 α，$\boldsymbol{\beta}_{\text{final}}$ 可在循环中与 $\boldsymbol{\beta}_{\text{initial}}$ 交换赋值实现迭代。

```
#初始化参数
beta_initial = np.zeros((p, 1))      #初始化参数beta
alpha = 0.5                          #初始化学习速率
beta_final = np.zeros((N-1, p))      #初始化储存参数更新矩阵
```

接着，利用循环不断根据新数据计算梯度、更新参数 $\boldsymbol{\beta}$ 的估计。另外，在优化的过程中还需对学习速率 α 不断进行更新。

```
#利用for循环更新模型参数
for i in range(0,N-1):
    grad = -1 * X[i, :] * (y[i] - np.dot(X[i, :], beta_initial))  #计算梯度
    alpha = 0.5 * pow((i+1), (-1/2))                    #更新学习速率
    beta_final[i, :] = beta_initial.T - alpha * grad    #更新参数
    beta_initial = beta_final[i, :]                     #交换参数
```

最终，通过可视化方法观察参数迭代收敛过程。

```
#可视化参数迭代过程
fig = plt.figure()
plt.scatter(range(1, N), beta_final[:,0], color='r', s=0.7, marker='.')
plt.scatter(range(1, N), beta_final[:,1], color='b', s=0.7, marker='.')
plt.axhline(y=1, color='r', linestyle='--')
plt.axhline(y=-1, color='b', linestyle='--')
plt.xlabel('epoch')
plt.ylabel('parameters')
plt.show()
```

图14.4表示迭代过程中参数迭代值的变化。我们可以从图像中观察到，参数逐渐收敛。

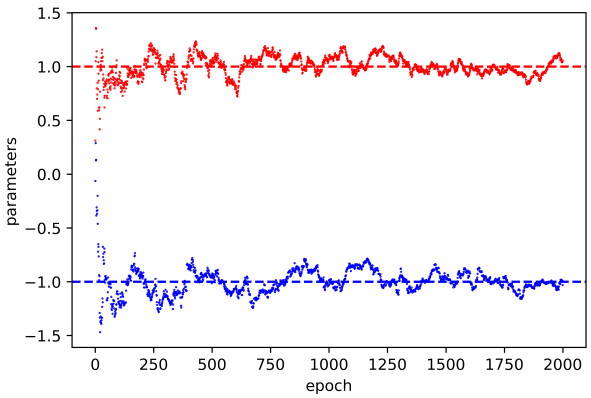


图 14.4 在线线性模型参数迭代值的变化

在线岭回归

下面介绍如何使用 Python 语言实现岭回归模型的在线学习算法。首先导入实验过程中需要的库并设定随机数种子。

```
#加载库
import numpy as np
import random
import matplotlib.pyplot as plt
```

```
from random import sample
#设定随机数种子
random.seed(12345)
np.random.seed(12345)
```

其次，我们定义一个用于计算损失函数值与参数迭代矩阵的函数。

```
# 岭回归标准方程法求解回归参数
def sgd_train(train_data, train_label, lambda1, eta, epsilon, max_epoch):
    #计算最大迭代次数
    train_len = train_data.shape[0]
    tau_max = max_epoch * train_len
    #初始化参数迭代矩阵
    beta = np.zeros((tau_max, train_data.shape[1]))
    value = np.zeros((tau_max, 1))
    #第一次迭代
    #迭代参数&损失函数值
    beta[0,:] = np.zeros((train_data.shape[1]))
    value[0] = 0.5*pow((train_label[0] - np.dot(beta[0,:].T, train_data[0,:])),2) +
    0.5*lambda1*np.dot(beta[0,:].T, beta[0,:])
    tau = 1
    while tau <= tau_max :
        if(value[tau-1] <= epsilon):#检查终止条件
            break
        else:
            train_index = sample(range(0, train_len), train_len)
            #循环遍历每个数据点
            for i in train_index:
                tau = tau + 1
                if(tau > tau_max):#检查终止条件
                    break
                else:
                    t = tau - 1#矩阵从0开始计数
                    y_pred = np.dot(train_data[i, :].T , beta[t-1, :])#计算预测值
                    #计算梯度
                    grad = (-1) * np.dot((train_label[i] - y_pred), train_data[i, :])
                    beta[t,:] = beta[t-1,:] * (1-eta * lambda1) - eta * grad#更新参数
                    #计算损失函数值
                    value[t] = 0.5 * pow((train_label[i] - np.dot(beta[t,:].T,
                    train_data[i,:])), 2) + 0.5 * lambda1 * np.dot(beta[t,:].T, beta[
                                                                    t,:])
    return value, beta
```

接着，我们构造初始数据集并对实验进行参数设置。

```
#构造数据集
N = 500#设置样本容量
p = 2#设置参数个数
X1 = np.random.randn(N * (p-1), p-1)
X = np.hstack((np.ones((N, 1)), X1))#生成设计矩阵
beta = np.array([1, -1])#设置参数
eps = np.random.randn(N)#随机生成扰动
y = np.dot(X, beta) + eps#生成观测向量
```

```
#参数设置
max_epoch = 30#最大循环次数
epsilon = 0.001#设置终止阈值
eta = 0.0005#学习效率
lambda1 = 0.05#正则化参数
```

最终，我们设置训练集数据及训练标签进行实验并对参数迭代矩阵进行可视化。

```
#设置训练集数据和训练标签
train_data = X
train_label = y
#计算参数更新过程中损失函数值以及参数迭代值
value, beta = sgd_train(train_data, train_label, lambda1, eta, epsilon, max_epoch)
#可视化迭代过程中损失函数值的变化与迭代矩阵的值
fig = plt.figure()
plt.scatter(range(1,len(beta)+1), beta[:, 0], color='r', s=0.3, marker='.')
plt.scatter(range(1,len(beta)+1), beta[:, 1], color='b', s=0.3, marker='.')
plt.axhline(y=1, color='r', linestyle='--')
plt.axhline(y=-1, color='b', linestyle='--')
plt.xlabel('epoch')
plt.ylabel('parameters')
plt.show()
```

图14.5表示迭代过程中参数迭代值的变化。我们可以从图像中观察到，参数逐渐收敛。

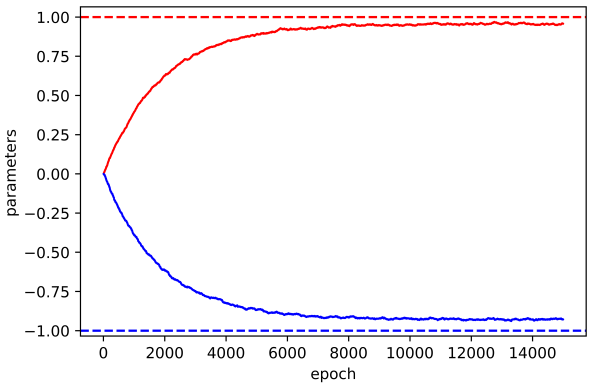


图 14.5 在线岭回归模型参数迭代值的变化

在线高维模型

下面介绍如何使用 Python 语言实现高维模型的在线学习算法。首先导入实验过程中需要的库并设定随机数种子

```
#加载库
import numpy as np
import random
import matplotlib.pyplot as plt
```

```
import math
#设定随机数种子
random.seed(12345)
np.random.seed(12345)
```

其次，我们生成符合逻辑回归的数据集，并进行数据读取。

```
#构造数据集
N = 10000#设置样本容量
p = 2#设置参数个数
X1 = np.random.randn(N * (p-1), p-1)
X = np.hstack((np.ones((N,1)), X1))#生成设计矩阵
beta = np.array([1, -1])#设置参数
eps = np.random.randn(N)#随机生成扰动
y = np.dot(X, beta) + eps#生成观测向量
#逻辑回归
logistic_y = 1/(1 + np.exp(-y))
l = len(logistic_y)
for i in range(0, l):
    if(logistic_y[i] <= 0.5):#以0.5进行二分类
        logistic_y[i] = 0
    else:
        logistic_y[i] = 1
#设置训练集数据和训练标签
train_data = X
train_label = logistic_y
```

接着，我们进行参数的设置。

```
#设置参数
omega = 0.1#超参数
gamma = 1#超参数
lambda1 = 0.01#正则化参数
T = len(train_label)#计算样本大小
N = train_data.shape[1]#计算参数向量的维度
beta_final = np.zeros((T, N))#初始化参数迭代矩阵
alpha = np.zeros((T, N))#初始化学习率矩阵
g = np.zeros((T, N))#初始化损失梯度矩阵
sigma = np.zeros((T, N))#初始化中间矩阵
Z = np.zeros((T, N))#初始化中间矩阵
```

再利用循环不断更新各维度参数值。

```
for t in range(0, T):
    #利用for循环对每个维度更新参数W
    if(t != 0):
        for i in range(0, N):
            if(Z[t-1,i] > lambda1):
                beta_final[t, i] = (lambda1 - Z[t-1, i]) * alpha[t-1, i]
            elif(Z[t-1,i] < -lambda1):
                beta_final[t, i] = (-lambda1 - Z[t-1, i]) * alpha[t-1, i]
            else:
                beta_final[t, i] = 0
```

```
    #计算预测值
    tmp = np.dot(train_data[t, :], beta_final[t, :].reshape(-1, 1))
    pt = 1 / (1 + np.exp(-tmp))
    #利用for循环对每个维度更新各参数的值
    for r in range(0, N):
        g[t, r] = (pt - train_label[t]) * train_data[t, r]
        sum = 0
        for s in range(0, t):
            sum = sum + g[s,r] * g[s,r]
        alpha[t, r] = omega / (gamma + math.sqrt(sum))#更新学习率
        if(t != 0):
            sigma[t, r] = 1/alpha[t, r] - 1/alpha[t-1, r]
            Z[t, r] = Z[t-1, r] + g[t, r] - sigma[t, r] * beta_final[t, r]
```

最终，我们借助可视化的方法观察参数迭代过程。

```
#可视化参数迭代矩阵
fig = plt.figure()
plt.scatter(range(1, l+1), beta_final[:, 0], color='r', s=0.7, marker='.')
plt.scatter(range(1,l+1), beta_final[:,1], color='b', s=0.7, marker='.')
plt.axhline(y=1.8, color='r', linestyle='--')
plt.axhline(y=-1.8, color='b', linestyle='--')
plt.xlabel('epoch')
plt.ylabel('parameters')
plt.show()
```

图14.6表示迭代过程中参数迭代值的变化。我们可以从图像中观察到，参数逐渐收敛。

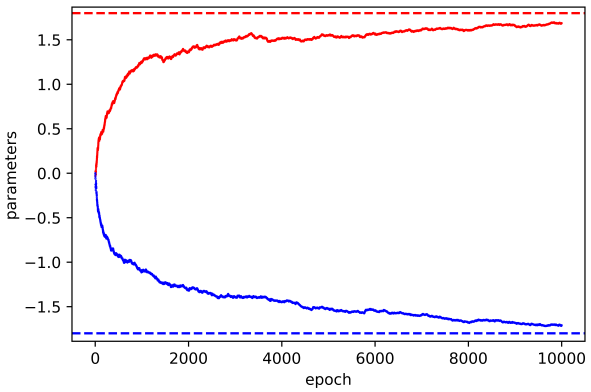


图 14.6 在线高维模型参数迭代值的变化

14.6 习题

1. 假设观测到一维数据 $\{X_{ti} : i = 1, \cdots, n_t, t = 1, \cdots\}$, X_{ti} 代表第 t 批次数据流第 i 个样本，n_t 是第 t 批次的样本量，并且 $\mathrm{var}(X_{ti}) = \sigma^2$。根据在线学习样本均值估计量的形式，请写出在线方差估计量 $\widehat{\sigma}^2$ 形式以及具体更新算法。

2. 假设观测到回归数据 $\{(X_{ti}^{\mathrm{T}}, Y_{ti}) : i = 1, \cdots, n_t, t = 1, \cdots\}$，满足模型

$$Y_{ti} = X_{ti}^{\mathrm{T}}\boldsymbol{\beta} + \epsilon_{ti},$$

根据在线学习样本均值估计量的形式（不用 SGD），请写出在线学习最小二乘估计量 $\widehat{\boldsymbol{\beta}}$ 形式以及具体更新算法。

3. 根据 FTRL-Proximal，写出基于 Lasso 惩罚函数的在线最小二乘估计过程以及程序实现。

4. 本练习中的数据是来自于摩洛哥北部泰图安市的 3 个区域的用电量以及影响到用电量的 6 个特征：时间，温度，湿度，风速，一般漫反射流，漫反射流。数据取自于监控和数据采集系统（SCADA）的 2017 年的全年数据，数据每 10 分钟采集一次。为公用事业公司提供智能策略并帮助其改进重要任务，采用统计模型对用电量进行分析。

（1）. 基于均值模型以及在线学习更新算法，分别估计 3 个区域的用电量。

（2）. 建立线性模型分析区域 1 的用电量，分别使用基于岭回归模型的在线学习算法和离线学习算法估计参数，并比较估计结果。

（3）. 建立线性模型分析区域 1 的用电量，分别使用基于 11.2.4 节高维在线学习算法和离线学习算法估计参数，并比较估计结果。

第十五章　并行计算

并行计算是一种通过同时处理多个计算任务来提高计算效率的方式，通过同时处理多个任务来提高整体计算速度，减少计算时间，同时处理的多个计算任务，可以在不同处理单元上并行执行。并行计算广泛应用于大规模数据处理、复杂模型训练、科学计算等领域。在线学习注重模型的持续适应新数据的能力，而并行计算注重通过同时处理多个任务来提高计算效率。

15.1　简介

随着数据的增加，机器学习中的计算瓶颈越发凸显，并行计算就成为解决这个瓶颈的关键技术。根据计算设备的不同，一般分为基于 CPU 的并行计算和基于 GPU 的并行计算，本章基于 Python 和 R 编程环境，给出了基于 CPU 和基于 GPU 的并行计算的相关概念、原理和计算流程。

15.2　并行计算相关概念

15.2.1　进程

进程的概念是 60 年代初首先由麻省理工学院的 MULTICS 系统和 IBM 公司的 CTSS/360 系统引入的。进程（Process）是具有一定独立功能的程序在某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位。程序只是一组指令的有序集合，它本身没有任何运行的含义，只是一个静态实体。而进程则不同，它是程序在某个数据集上的执行，是一个动态实体。它因创建而产生，因调度而运行，因等待资源或事件而被处于等待状态，因完成任务而被撤消，反映了一个程序在一定的数据集上运行的全部动态过程。图15.1为 Windows 任务管理器中进程一览表，每个进程表现为一个独立的执行程序。

1、进程的特征

根据进程的特点，它具有以下特征：

（1）动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的。

（2）并发性：任何进程都可以同其他进程一起并发执行。

（3）独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位。

（4）异步性：由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进。

（5）结构特征：进程由程序、数据和进程控制块三部分组成。

多个不同的进程可以包含相同的程序：一个程序在不同的数据集里就构成不同的进程，能得到不同的结果；但是执行过程中，程序不能发生改变。

2、进程的状态

进程执行时的间断性，决定了进程可能具有多种状态。事实上，运行中的进程可能具有如图15.2所示的三种基本状态：运行态、就绪态和阻塞态。

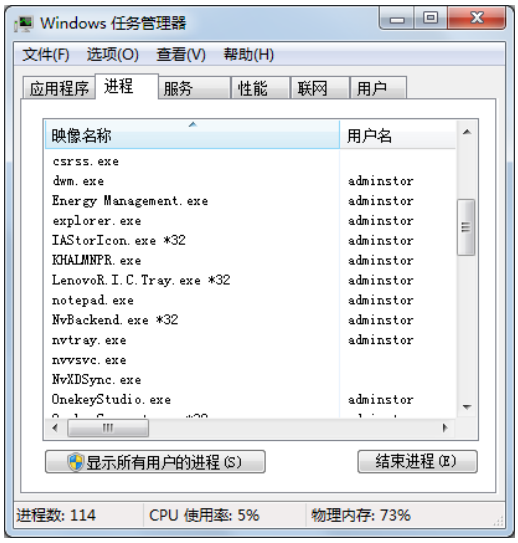


图 15.1 Windows 任务管理器之进程显示

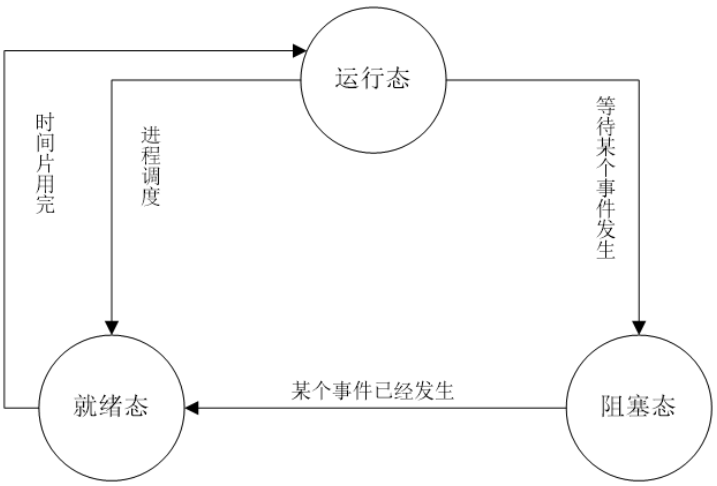


图 15.2 进程状态图

（1）就绪态（Ready）

进程已获得除处理器外的所需资源，等待分配处理器资源；只要分配了处理器进程就可执行。就绪进程可以按多个优先级来划分队列。例如，当一个进程由于时间片用完而进入就绪状态时，排入低优先级队列；当进程由 I/O 操作完成而进入就绪状态时，排入高优先级队列。

（2）运行态（Running）

进程占用处理器资源，处于此状态的进程的数目小于等于处理器的数目。在没有其他进程可以执行时（如所有进程都在阻塞状态），通常会自动执行系统的空闲进程。

（3）阻塞态（Blocked）

由于进程等待某种条件（如 I/O 操作或进程同步），在条件满足之前无法继续执行。该事件发生前即使把处理机分配给该进程，也无法运行。

15.2.2 线程

线程（Thread）是进程的一个实体，是 CPU 调度和分派的基本单位。线程不能够独立执行，必须依存在进程中，由进程提供多个线程执行控制。从内核角度讲线程是活动体对象，而进程只是一组静态的对象集，进程必须至少拥有一个活动线程才能维持运转。图15.3表明一个进程最多可以调用 8 个线程。

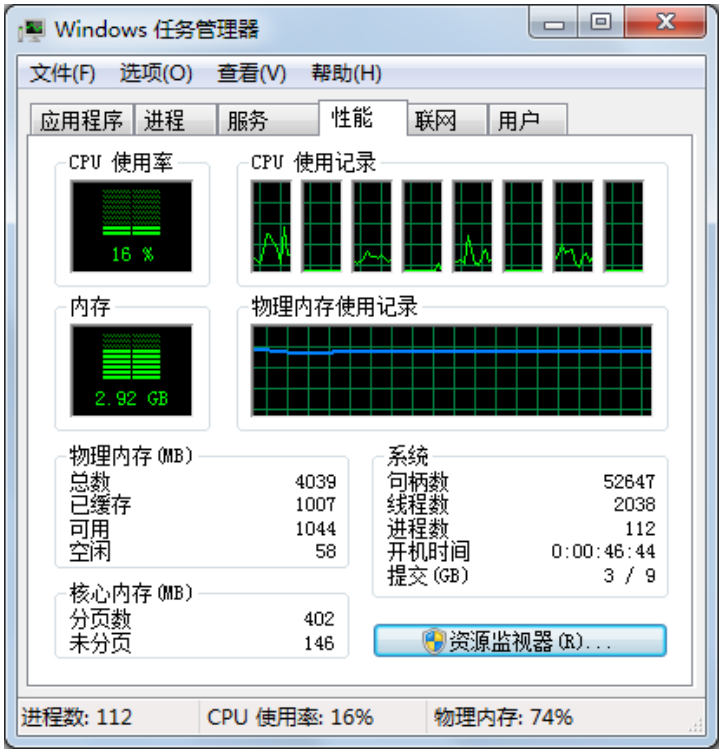


图 15.3 进程状态图

线程和进程的关系是：线程是属于进程的，线程运行在进程空间内，同一进程所产生的线程共享同一内存空间，当进程退出时该进程所产生的线程都会被强制退出并清除。线程可与属于同一进程的其它线程共享进程所拥有的全部资源，但是其本身基本上不拥有系统资源，只拥有一点在运行中必不可少的信息（如程序计数器、一组寄存器和栈）。

在操作系统中引入线程带来的主要好处是：

（1）在进程内创建、终止线程比创建、终止进程要快。

（2）同一进程内的线程间切换比进程间的切换要快，尤其是用户级线程间的切换。

另外，线程的出现还因为以下几个原因：

（1）并发程序的并发执行，在多处理环境下更为有效。一个并发程序可以建立一个进程，而这个并发程序中的若干并发程序段就可以分别建立若干线程，使这些线程在不同的处理机上执行。

（2）每个进程具有独立的地址空间，而该进程内的所有线程共享该地址空间。这样可以解决父子进程模型中，子进程必须复制父进程地址空间的问题。

（3）线程对解决客户/服务器模型非常有效。

不同的平台对线程的状态定义不同，大致可以定义为运行、挂起、睡眠、阻塞、就绪、终止这六种，如图15.4所示。

运行：就是线程获得了 CPU 的控制权，正在执行计算。

挂起：一般是指被挂起，因为同一时刻，需要“同步”运行的线程不止他一个，所以基于时间片轮转的原则，它在独占了一段时间的 CPU 后，被挂起，线程环境被压栈。

睡眠：一般是指主动挂起，这种情况在 WINDOWS 平台不存在。

阻塞：与挂起和睡眠类似，都是失去 CPU 的控制权。与挂起更相像，也是被挂起的。不同之处在于，被挂起的线程没有额外的表示，而被阻塞的线程会被记录下来，当等待的因素就绪后，线程会转为就绪状态。例如你在线程中调用一些系统服务函数，会引起线程控制权的一次裁决，从而挂起本线程，造成本线程的阻塞。挂起、睡眠、阻塞看起来差不多，但其实本质上还是有以上所述区别的。

就绪：顾名思义，就是指它准备好了，一旦轮到它，它就可以转为运行状态。

终止：线程结束。

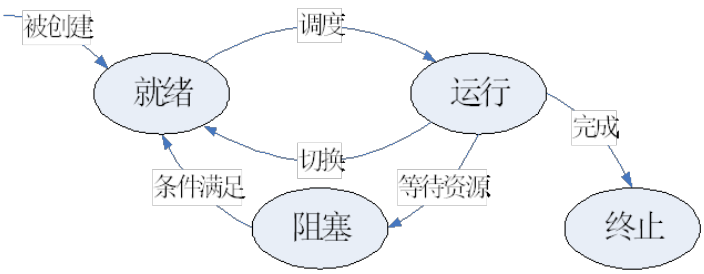


图 15.4 线程状态图

15.2.3 并行计算与分布式计算

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程，是提高计算机系统计算速度和处理能力的一种有效手段。它的基本思想是用多个处理器（多个核）来协同求解同一问题，即将被求解的问题分解成若干个部分，各部分均由一个独立的处理机来并行计算。并行计算系统既可以是专门设计的、含有多个处理器的超级计算机，也可以是以某种方式互连的若干台的独立计算机构成的集群。通过并行计算集群完成数据的处理，再将处理的结果返回给用户。

并行计算可分为时间上的并行和空间上的并行。时间上的并行是指流水线技术。而空间上的并行是指多个处理机并发的执行计算，即通过网络将两个以上的处理机连接起来，达到同时计算同一个任务的不同部分，或者单个处理机无法解决的大型问题。但这个空间一般是指集中放在一起的集群计算机。并行计算中主要研究的是空间上的并行问题。从程序和算法设计人员的角度来看，并行计算又可分为数据并行和任务并行。空间上的并行导致了两类并行机的产生，按照 Flynn（弗林）分类法

为：单指令流多数据流（SIMD）和多指令流多数据流（MIMD）。我们常用的串行机也叫做单指令流单数据流（SISD）。

而分布式计算（Distributed computing）也是一种并行计算，主要是指通过网络相互连接的两个以上的处理机相互协调，各自执行相互依赖的不同应用，从而达到协调资源访问，提高资源使用效率的目的。但是，它无法达到并行计算所倡导的提高求解同一个应用的速度，或者提高求解同一个应用的问题规模的目的。对于一些复杂应用系统，分布式计算和并行计算通常相互配合，既要通过分布式计算协调不同应用之间的关系，又要通过并行计算提高求解单个应用的能力。

因此，并行计算一般在企业内部进行，而分布式计算可能会跨越局域网，或者直接部署在互联网上，节点之间几乎不互相通信。很多公益性的项目，就是使用分布式计算的方式在互联网上实现，比如以寻找外星人为目的的 SETI 项目。

15.2.4 同步与异步

同步（Synchronization）：进程之间的关系不是相互排斥临界资源的关系，而是相互依赖的关系。进一步的说明：就是前一个进程的输出作为后一个进程的输入，当第一个进程没有输出时第二个进程必须等待。具有同步关系的一组并发进程相互发送的信息称为消息或事件。

异步（Asynchronization）：异步和同步是相对的一个概念，同步就是顺序执行，执行完一个再执行下一个，需要等待、协调运行。异步就是彼此独立，在等待某事件的过程中继续做自己的事，不需要等待这一事件完成后再工作。线程就是实现异步的一个方式。异步让调用方法的主线程不需要同步等待另一线程的完成，从而可以让主线程干其它的事情。

15.2.5 通信

在并行计算中，通信是指进程或线程之间的数据传输或内存访问。一般分为两种：共享内存和消息传递。

如图15.5，共享内存这种方式在多核并行计算中比较常见，通常会设置一个共享变量，然后多个线程去操作同一个共享变量，从而达到线程通讯的目的。这种通讯模式中，不同的线程之间是没有直接联系的。都是通过共享变量这个“中间人”来进行交互。而这个“中间人”必要情况下还需被保护在临界区内（加锁或同步）。由此可见，一旦共享变量变得多起来，并且涉及到多种不同线程对象的交互，这种管理会变得非常复杂，极容易出现数据竞争、死锁等问题。

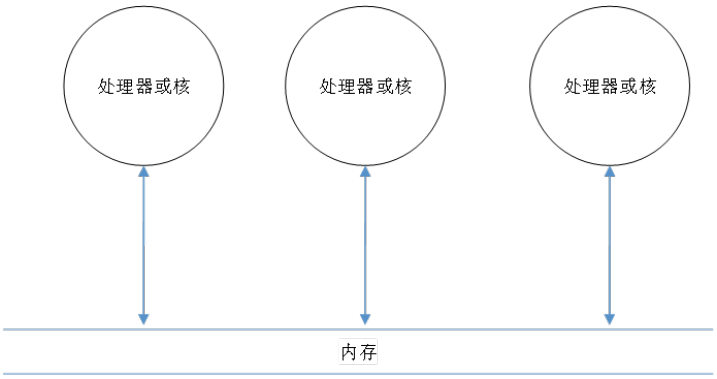


图 15.5 共享内存

而消息传递方式主要针对多处理器或多节点并行计算，采取的是进程或线程之间的直接通信，不同的进程或线程之间通过显式的发送消息来达到交互目的。如图15.6，消息传递模型有以下特征：

（1）计算时任务集可以用它们自己的内存。多任务可以在相同的物理处理器上，同时可以访问任意数量的处理器。

（2）任务之间通过接收和发送消息来进行数据通信。

（3）数据传输通常需要每个处理器协调操作来完成。例如，发送操作有一个接受操作来配合。

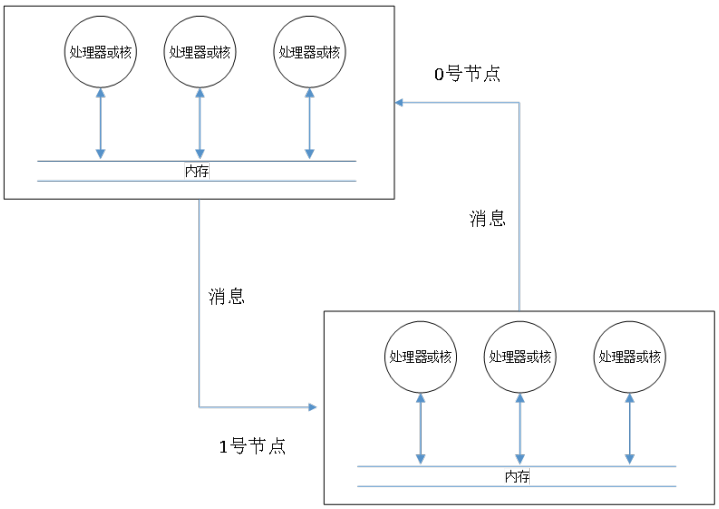


图 15.6 消息传递

表15.1总结了这两种通讯模式的异同情况。

并发模型	通信机制	同步机制
共享内存	线程之间共享程序的公共状态，线程之间通过写-读内存中的公共状态来隐式进行通信。	同步是显式进行的。程序员必须显式指定某个方法或某段代码需要在线程之间互斥执行。
消息传递	线程之间没有公共状态，线程之间必须通过明确的发送消息来显式进行通信。	由于消息的发送必须在消息的接收之前，因此同步是隐式进行的。

表 15.1 共享内存与消息传递的异同

15.2.6 加速比

为了更方便地描述并行计算的性能，一般采用加速比指标来进行度量。加速比定义如下：

$$S = \frac{T_s}{T_p} \tag{15.2.1}$$

其中，T_s 表示单处理器上最优串行化算法计算的时间，T_p 表示使用 p 个 CPU 处理器并行计算的时间。

当加速比 $S<1$ 意味着并行计算的时间比串行计算的时间还长，并行计算效率反而降低；$S<p$ 时，表示次线性加速；$S \approx p$ 时，表示线性加速；$S>p$ 时，表示超线性加速。一般来说，加速比通常都小于 CPU 核数，只有极少数并行算法可以获得超线性加速比，例如并行搜索工作量少于串行搜索工作量等算法，另外，也有可能是由于高速缓存产生的额外加速效果所导致。因此，在并行算法设计中，其加速比一般要求向 CPU 核数靠近，加速比越接近线性加速，那么程序性能就越好。

但是，对于某些串行程序，并不是所有部分都可以用并行程序替代，有一部分必须要串行执行。令 W_s 为程序中的串行部分，W_p 为程序中的并行部分，则 $W = W_s + W_p$. 根据 Amdahl 定律，在计算规模一定的情况下，加速比定义为:

$$S = \frac{W}{W_s + W_p/p} \tag{15.2.2}$$

串行部分比例为 $f = \dfrac{W_s}{W}$，则式 (15.2.2) 为

$$S = \frac{1}{f + (1-f)/p} \tag{15.2.3}$$

随着 CPU 核数 p 的增加，这是一个递增的函数，但这个函数有上限，当 $p \to \infty$ 时，有

$$S = \frac{1}{f}$$

图15.7为串行程序比例分别是 0.0，0.1，…，0.2 时的加速比随 CPU 核数 p 变化（这里核数取 2、4、8、16、24、48）的情形。从图中可以看出，当一定的情况下，加速比随核数的增加而增加，但是在 16 核之前增速较快，之后增速较为缓慢；当 CPU 核数一定时，加速比随着串行程序占比的减少而增加。因此，要想提高并行计算效率，需要从 CPU 核数和串行程序占比两个因素综合考虑。

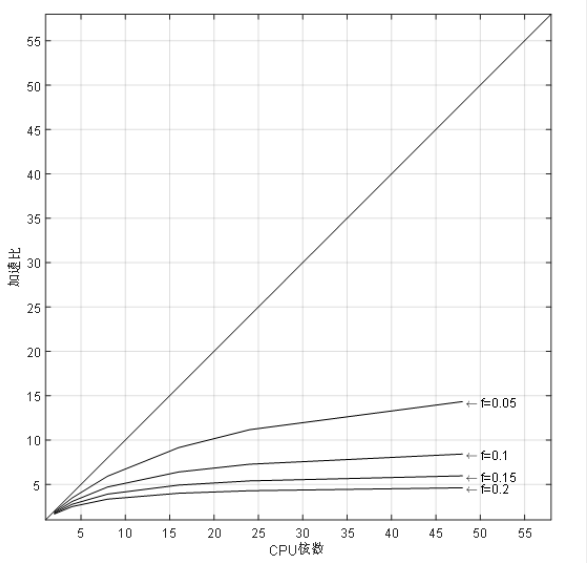


图 15.7 加速比随 CPU 核数 p 变化的情形

如果考虑并行计算时的通信、同步和归约等操作所花费的额外开销 W_0，则

$$S = \frac{W}{W_s + W_p/p + W_0}, \tag{15.2.4}$$

$$S = \frac{1}{f + (1-f)/p + W_0/W}. \tag{15.2.5}$$

当并行计算时的通信、同步和归约等操作所花费的额外开销比重较大时，并行效率降低，严重时加速比小于 1。在后面例子将遇到这种情况。对于很多大型计算，精度要求高，而计算时间要求不能增加。在这种计算规模增加的情况下，要想保持原有计算时间，必须增加处理器才能完成计算任务。Gustafson [157] 提出了变问题规模的加速比模型如下：

$$S = \frac{f + (1-f)p}{1 + W_0/W}. \tag{15.2.6}$$

从式 (15.2.6) 可以看出，当处理器个数增加时，必须控制额外开销的增加，才能达到线性加速。因此，在并行计算中，如何优化程序，是一个值得思考的问题。考虑到加速比计算的简便性，在本书中采用加速比公式 (15.2.1)。

15.3 基于 CPU 线程的并行计算

在 Python 中，提供了基于线程的并行模块 threading，threading 模块除了 Thread 类之外，还包括其它很多的同步机制，这些同步机制可以避免数据竞争问题的发生。

15.3.1 创建线程

使用 Thread 类主要有两种方法创建线程：

- 创建 Thread 类的实例，传递一个函数；
- 派生 Thread 类的子类，并创建子类的实例。

本教材主要采用第一种方式创建线程。对于 Thread 类，其主要属性和方法见表15.2。

属性/方法	描述
Thread 类属性	
name	线程名
ident	线程的标识符
daemon	布尔值，表示这个线程是否是守护线程
Thread 类方法	
init(group,...)	实例化一个线程对象，需要一个可调用的 target 对象，以及参数 args 或者 kwargs。还可以传递 name 和 group 参数。daemon 的值将会设定 thread.daemon 的属性。
start()	线程启动函数。开始执行该线程
run()	定义线程的方法。(通常在子类中重写)
join(timeout=None)	等待函数。直至启动的线程终止之前一直挂起，除非给出了 timeout(单位秒)，否则一直被阻塞。

表 15.2 Thread 类主要属性和方法

注意：守护线程一般是一个等待客户端请求的服务器。如果没有客户端请求，守护线程就是空闲的。如果把一个线程设置为守护线程，就表示这个线程是不重要的，进程退出时不需要等待这个线程执行完成。使用下面的语句：thread.daemon=True 可以将一个线程设置为守护线程，同样的也可以通过这个值来查看线程的守护状态。对于主线程，将在所有的非守护线程退出之后才退出。

如图15.8，当线程对象被创建，其活动可通过调用线程的 start() 方法开始。一旦线程活动开始，该线程会被启动。主线程可以调用一个线程的 join() 方法，这会阻塞调用该方法的线程，直到被调用 join() 方法的线程终结。

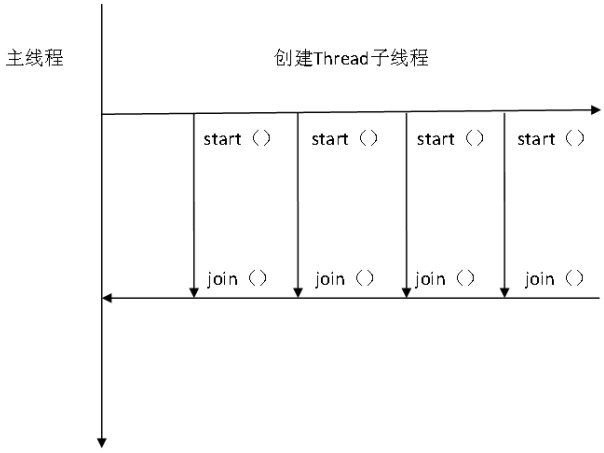


图 15.8 线程创建过程

下面给出具体的 Thread 类构造函数：class threading.Thread(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None)

调用这个构造函数时，必需带有关键字参数。参数如下：

group 应该为 None，为了日后扩展 ThreadGroup 类实现而保留。

target 是用于 run() 方法调用的可调用对象，通常是启动一个线程活动时要执行的目标函数。默认是 None，表示不需要调用任何方法。

name 是线程名称。默认情况下，由 “Thread-N” 格式构成一个唯一的名称，其中 N 是小的十进制数。

args 是用于调用目标函数的参数元组。默认是 ()。

kwargs 是用于调用目标函数的关键字参数字典，默认是 {}。

daemon 如果不是 None，将显式地设置该线程是否为守护模式。如果是 None（默认值），线程将继承当前线程的守护模式属性。

例 15.1 定义线程函数 HelloWord，其参数为自定义的线程号（从 0 开始），其内容为延迟 1 秒后，输出形如 Hello world! thread id:0 这样的字符串。使用 Thread 创建当前计算机最大 CPU 或核数的线程数，并用 start 函数启动该线程，同时使用计时函数来了解使用 join 函数的作用。

```
import threading
from time import sleep
from time import perf_counter
import multiprocessing
N_core = multiprocessing.cpu_count()#获取当前计算机最大CPU或核数
```

```
def HelloWord(thread_id):#定义线程函数
    sleep(1)#延迟1秒
    print('Hello world! thread id:{} \n'.format(thread_id))
    return
if __name__ == "__main__":
    start = perf_counter()#计时开始
    for i in range(N_core):
        t = threading.Thread(target=HelloWord,args=(i,))#创建子线程
        t.start()#启动线程
        t.join()#阻塞
    end = perf_counter()#计时结束
    print('运行时间为: %s Seconds'%(end-start))
```

输出结果为:

```
  Hello world! thread id:2
  Hello world! thread id:1
  Hello world! thread id:0
  Hello world! thread id:5
  Hello world! thread id:6
  Hello world! thread id:4
  Hello world! thread id:3
  Hello world! thread id:7
运行时间为: 0.002644299998792121 Seconds
```

在主线程中创建了 8 个子线程，每个子线程都调用了线程函数 HelloWord，计时函数对主线程创建和启动子线程进行计时，由于不需要等待子线程结束，所以先输出了主线程运行时间，然后乱序输出各子线程的输出内容。如果在程序中加入 t.join() 等待函数，输出结果如下:

```
  Hello world! thread id:0
  Hello world! thread id:1
  Hello world! thread id:2
  Hello world! thread id:3
  Hello world! thread id:4
  Hello world! thread id:5
  Hello world! thread id:6
  Hello world! thread id:7
  运行时间为: 8.120153000000073 Seconds
```

很明显，主线程等子线程一个个执行完，再输出运行时间，该时间包含了等待时间。从时间上看与串行计算无异，也就是说并没有真正按照图15.8运行。因此，对于 join 函数，多数情况下根本不需要调用它。一旦线程启动，就会一直执行，直到给定的函数完成后退出。如果主线程还有其它事情要做（并不需要等待这些线程完成），可以不调用 join 函数。join 函数只有在你需要等待线程完成时候才是有用的。

15.3.2 同步

在多线程程序中，一般有一些特定的函数或代码块不希望被多个线程同时执行，这就需要使用同步了。threading 模块的同步机制，如表15.3所示，主要有锁机制、事件、信号量、栅栏。

对象	描述
Lock	锁对象
RLock	递归锁，是一个线程可以再次拥有已持有的锁对象
Condition	条件变量对象，使一个线程等待另一个线程满足特定的条件触发
Event	事件对象，普通版的 Condition
Semaphore	信号量，为线程间共享的资源提供一个“计数器”，计数开始值为设置的值，默认为 1
BoundedSemaphore	与 Semaphore 相同，有边界，不能超过设置的值
Timer	定时运行的线程对象，定时器
Barrier	栅栏，当达到某一栅栏后才可以继续执行

表 15.3 同步机制

1、锁（Lock）

锁仅有锁定和非锁定两种状态。它被创建时为非锁定状态。它有两个基本方法：acquire 和 release。当状态为非锁定时，acquire 将状态改为锁定并立即返回。当状态是锁定时，acquire 将阻塞至其它线程，调用 release 将其改为非锁定状态，然后 acquire 调用重置其为锁定状态并返回。release 只在锁定状态下调用，它将状态改为非锁定并立即返回。如果尝试释放一个非锁定的锁，则会引发 RuntimeError 异常。

当多个线程在 acquire 等待状态转变为未锁定被阻塞，然后 release 重置状态为未锁定时，只有一个线程能继续执行；至于哪个等待线程继续执行没有定义，并且会根据实现而不同。

具体方法如下：

acquire(blocking=True, timeout=-1)

可以阻塞或非阻塞地获得锁。当调用时参数 blocking 设置为 True（缺省值），阻塞直到锁被释放，然后将锁锁定并返回 True 。在参数 blocking 被设置为 False 的情况下调用，将不会发生阻塞。如果调用时 blocking 设为 True 会阻塞，并立即返回 False，否则，将锁锁定并返回 True。

release()

释放一个锁。这个方法可以在任何线程中调用，不单指获得锁的线程。当锁被锁定，将它重置为未锁定，并返回。如果其他线程正在等待这个锁解锁而被阻塞，只允许其中一个允许。在未锁定的锁调用时，会引发 RuntimeError 异常。没有返回值。

locked()

如果获得了锁则返回真值。

例 15.2 定义计数函数 CountNum(thread_id)，使用两个线程调用该函数，其中第一个线程延迟 1 秒，而第二个线程延迟 2 秒，在主线程中输出总计数。定义全局变量 counts，程序设计如下：

```
import threading
from time import sleep
counts=0
N_threads=2
def CountNum(thread_id):#定义线程函数
    global counts
    sleep(thread_id)
    for i in range(1,101):
        counts=counts+1
    print('thread id:{},its counts is {}\n'.format(thread_id,counts))
    return
if __name__ == "__main__":
    for i in range(N_threads):
        t = threading.Thread(target=CountNum,args=(i+1,))#创建子线程
        t.start()#启动线程
    print('counts=%d Seconds'%counts)
```

运行结果如下：

```
  counts=0 Seconds
  thread id:1,its counts is 100
  thread id:2,its counts is 200
```

很明显，主线程并没有得到子线程的计数结果。如果采用锁，当子线程各自计数时，主线程等待其完成后再进行输出，其结果就是总计数了。程序修改为：

```
import threading
from time import sleep
counts=0
N_threads=2
lock = threading.Lock()#创建锁
def CountNum(thread_id):#定义线程函数
    lock.acquire()#请求锁
    global counts
    sleep(thread_id)
    for i in range(1,101):
        counts=counts+1
    print('thread id:{},its counts is {}\n'.format(thread_id,counts))
    lock.release()#释放锁
    return
if __name__ == "__main__":
    for i in range(N_threads):
        t = threading.Thread(target=CountNum,args=(i+1,))#创建子线程
        t.start()#启动线程
```

```
    lock.acquire()#请求锁
    print('counts=%d Seconds'%counts)
    lock.release()#释放锁
```

运行结果为：

```
thread id:1,its counts is 100
thread id:2,its counts is 200
counts=200 Seconds
```

2. 事件（Event）

事件用于不同进程间的相互通信。事件对象管理一个内部标识，调用 set 方法可将其设置为 true。调用 clear 方法可将其设置为 false。调用 wait 方法将进入阻塞直到标识为 true，这个标识初始时为 false。

具体方法如下：

is_set()

当且仅当内部标识为 true 时返回 True。

set()

将内部标识设置为 true，所有正在等待这个事件的线程将被唤醒。当标识为 true 时，调用 wait() 方法的线程不会被被阻塞。

clear()

将内部标识设置为 false，之后调用 wait() 方法的线程将会被阻塞，直到调用 set() 方法将内部标识再次设置为 true。

wait(timeout=None)

阻塞线程直到内部变量为 true。如果调用时内部标识为 true，将立即返回。否则将阻塞线程，直到调用 set() 方法将标识设置为 true 或者发生可选的超时。当提供了 timeout 参数且不是 None 时，它应该是一个浮点数，代表操作的超时时间，以秒为单位（可以为小数）。

例 15.3 针对例15.2，使用事件输出线程函数各自的计数。

```
import threading
from time import sleep
counts = 0
N_threads = 2
evGetData = threading.Event()#创建事件对象，内部标志默认为False
evOutput = threading.Event()
def CountNum(thread_id):#定义线程函数
    evGetData.wait()#等待主线程激活
    global counts
    counts = 0
    sleep(thread_id)
    for i in range(1, 100*thread_id+1):
        counts = counts + 1
    print('thread id:{}, its counts is {}'.format(thread_id, counts))
    evOutput.set()#激活主线程
```

```
    return
if __name__ == "__main__":
    for i in range(N_threads):
        t = threading.Thread(target=CountNum, args=(i+1,))#创建子线程
        t.start()#启动线程
        evGetData.set()#激活从线程
        evOutput.wait()#等待从线程激活
        print('counts=%d Seconds'%counts)
        evOutput.clear()
```

输出结果为:

```
thread id:1,its counts is 100
counts=100 Seconds
thread id:2,its counts is 200
counts=200 Seconds
```

15.4 基于 CPU 进程的并行计算

在 Python 程序中，代码执行由 Python 虚拟机（解释器主循环）来控制。对 Python 虚拟机的访问由 GIL（global interpreter lock，全局解释器锁）控制，GIL 保证同一时刻只有一个线程在执行。由于 GIL 的限制，python 多线程实际只能运行在单核 CPU，所以15.3节中的 threading 模块并不能真正实现多核 CPU 并行计算。如要实现多核 CPU 并行，只能通过多进程的方式实现。而 multiprocessing 模块是最常用的多进程模块，它同时提供了本地和远程并发操作，通过使用子进程而非线程有效地绕过了 GIL。因此，multiprocessing 模块允许程序员充分利用给定机器上的多个处理器或核进行并行计算。

15.4.1 创建进程

使用 multiprocessing 模块创建进程主要有两种方式：一是使用 Process 对象，二是采用 Pool 进程池。

1. **Process 对象**

在 multiprocessing 中，通过创建一个 Process 对象然后调用它的 start 方法来生成进程，创建过程同线程创建过程。

Process 对象表示在单独进程中运行的活动，具体的 Process 构造函数形式为:

class multiprocessing.Process(group=None, target=None, name=None, args=(), kwargs=, *, daemon=None)

其中，group 应该是 None，它仅用于兼容 threading.Thread。

target 是由 run 方法调用的可调用对象，它默认为 None，意味着什么都没有被调用。

name 是进程名称。

args 是目标调用的参数元组。

kwargs 是目标调用的关键字参数字典。

daemon 可以设置为 True 或 False。如果是 None （默认值），则该标志将从创建的进程继承。

Process 对象拥有和 threading.Thread 等价的大部分方法。具体说明如下：

run()表示进程活动的方法，可以在子类中重载此方法。标准 run() 方法调用传递给对象构造函数的可调用对象作为目标参数（如果有），分别从 args 和 kwargs 参数中获取顺序和关键字参数。

start()为启动进程活动，这个方法每个进程对象最多只能调用一次，它会将对象的 run() 方法安排在一个单独的进程中调用。

join([timeout])表示如果可选参数 timeout 是 None（默认值），则该方法将阻塞，直到调用 join() 方法的进程终止。如果 timeout 是一个正数，它最多会阻塞 timeout 秒。请注意，如果进程终止或方法超时，则该方法返回 None 。检查进程的 exitcode 以确定它是否终止。一个进程可以被 join 多次。进程无法 join 自身，因为这会导致死锁。尝试在启动进程之前 join 进程是错误的。

name为进程的名称。该名称是一个字符串，仅用于识别目的，它没有语义，可以为多个进程指定相同的名称。初始名称由构造器设定，如果没有为构造器提供显式名称，则会构造一个形式为 “Process-N1:N2:...:Nk” 的名称。

daemon为进程的守护标志，一个布尔值。这必须在 start() 被调用之前设置。注意：在 Windows 上要想使用进程模块，就必须把有关进程的代码写在当前.py 文件的 if _name_ == ‘_main_’ ：语句的下面，才能正常使用 Windows 下的进程模块。Unix/Linux 下则不需要。

例 15.4 定义函数 HelloWord，其内容为延迟 1 秒后，输出形如 Hello world! current_process name=Process-1 这样的字符串。在主进程中创建 4 个进程，同时在主进程中输出形如 Hello world! current_process name=MainProcess 这样的字符串。

在 py 文件中输入程序：

```
from time import sleep
import multiprocessing as mp
def HelloWord():
    name=mp.current_process().name
    sleep(1)#延迟1秒
    print('Hello world! current_process name=%s \n'%name)
    return
if __name__ == "__main__":
    name=mp.current_process().name
```

```
    print('Hello world! current_process name=%s \n'%name)
    for i in range(4):
        p = mp.Process(target=HelloWord)
        p.start()
        p.join()
        name=mp.current_process().name
    print('Hello world! current_process name=%s \n'%name)
打开Anaconda Prompt，键入以下命令（文件位置要换成自己的位置）：
python E:\MyPython\并行计算\eg4_1.py
```

输出结果如下：

```
  Hello world! current_process name=MainProcess
  Hello world! current_process name=Process-1
  Hello world! current_process name=Process-2
  Hello world! current_process name=Process-3
  Hello world! current_process name=Process-4
  Hello world! current_process name=MainProcess
```

一般来讲，在主程序中创建并启动子进程，然后主进程有处理数据、与子进程通信等工作，并等待子进程完成任务，基于这种框架，上述程序修改为：

```
from time import sleep
import multiprocessing as mp
N_core=4#进程数
def HelloWord(ID):
    name=mp.current_process().name
    sleep(1)#延迟1秒
    print('Hello world! current_process name=%s,ID=%d \n'%(name,ID))
    return
if __name__ == "__main__":
    name=mp.current_process().name#输出主进程名
    print('Hello world! current_process name=%s \n'%name)
    processes = []#进程列表
    for id in range(N_core):#创建子进程
        p = mp.Process(target=HelloWord,args=(id+1, ))
        p.start()
        processes.append(p)
    name=mp.current_process().name#输出主进程名
    print('Hello world! current_process name=%s \n'%name)
    #主进程处理
    #......
    #主进程处理结束
    #等待各子进程处理结果
    for p in processes:
        p.join()
```

2. Pool 进程池

Pool 进程池可以提供指定数量的进程供用户调用，当有新的请求提交到 Pool 中时，如果池还没有满，就会创建一个新的进程来执行请求。如果池满，请求就会告知先等待，直到池中有进程结束，才会创建新的进程来执行这些请求。

Pool 进程池常见的方法有：

apply 函数，原型：apply(func[, args=()[, kwds=]])

该函数用于传递不定参数，使用阻塞方式调用 func，执行完一个进程后再去执行其它进程。

apply_async 函数，原型：apply_async(func[, args=()[, kwds=[, callback=None]]])

该函数与 apply 用法一致，但它是非阻塞的且支持结果返回后进行回调。能够多个线程同时异步执行。

map() 函数，原型：map(func, iterable[, chunksize=None])

该函数与内置的 map 函数用法行为基本一致，它会使进程阻塞直到结果返回。注意：虽然第二个参数是一个迭代器，但在实际使用中，必须在整个队列都就绪后，程序才会运行子进程。

map_async() 函数，原型：map_async(func, iterable[, chunksize[, callback]])

该函数与 map 用法一致，但是它是非阻塞的。其有关事项见 apply_async。

close 函数：关闭进程池，使其不再接受新的任务。

terminal：结束工作进程，不再处理未处理的任务。

join 函数：表示主进程阻塞等待子进程的退出，join 要在 close 或 terminate 之后使用。

例 15.5 用 Pool 进程池完成例15.4。

```
from time import sleep
import multiprocessing as mp
N_core=4#进程数
def HelloWord(ID):
    name = mp.current_process().name
    sleep(1)#延迟1秒
    print('Hello world! current_process name=%s,ID=%d \n'%(name,ID))
    return
if __name__ == "__main__":
    name = mp.current_process().name#输出主进程名
    print('Hello world! current_process name=%s \n'%name)
    pools = []#进程池列表
    pool = mp.Pool(processes=N_core)#创建进程池
    for id in range(N_core):
        pools.append(pool.apply_async(HelloWord, args = (id+1,)))
        pool.close()
    name=mp.current_process().name#输出主进程名
    print('Hello world! current_process name=%s \n'%name)
    #主进程处理
    #......
    #主进程处理结束
    #等待各子进程处理结果
    pool.join()
打开Anaconda Prompt，键入以下命令（文件位置要换成自己的位置）：
python E:\MyPython\并行计算\eg5.py
```

输出结果如下：

```
Hello world! current_process name=MainProcess
Hello world! current_process name=MainProcess
Hello world! current_process name=SpawnPoolWorker-1,ID=1
Hello world! current_process name=SpawnPoolWorker-2,ID=2
Hello world! current_process name=SpawnPoolWorker-3,ID=3
Hello world! current_process name=SpawnPoolWorker-4,ID=4
```

15.4.2 进程间通信

在 multiprocessing 中，进程间通信主要有数据共享和数据传递，本书只介绍数据传递，而数据传递有队列（Queue）和管道（Pipe）两种，主要区别为：

（1）Queue 使用 put 和 get 维护队列，Pipe 使用 send 和 recv 维护队列。

（2）Pipe 只提供两个端点，而 Queue 没有限制。

（3）Queue 的封装比较好，Queue 只提供一个结果，可以被多个进程同时调用；而 Pipe 返回两个结果，分别由两个进程调用。

（4）Queue 的实现基于 Pipe，所以 Pipe 的运行速度比 Queue 快很多。

（5）当只需要两个进程时，Pipe 更快，当需要多个进程同时操作队列时，使用 Queue。

1. **队列（Queue）**

队列是一个先进先出（FIFO）的数据结构，很多场景需要按先来后到的顺序进行处理。其形式为：

class multiprocessing.Queue([maxsize])

Queue 返回一个使用一个管道和少量锁和信号量实现的共享队列实例。当一个进程将一个对象放进队列中时，一个写入线程会启动并将对象从缓冲区写入管道中。

除了 task_done() 和 join() 之外，Queue 实现了标准库类 queue.Queue 中所有的方法。常见方法有：

put(obj[, block[, timeout]])

将 obj 放入队列。如果可选参数 block 是 True （默认值），而且 timeout 是 None （默认值), 将会阻塞当前进程，直到有空的缓冲槽。如果 timeout 是正数，将会在阻塞了最多 timeout 秒之后还是没有可用的缓冲槽时抛出 queue.Full 异常。反之（block 是 False 时），仅当有可用缓冲槽时才放入对象，否则抛出 queue.Full 异常（在这种情形下 timeout 参数会被忽略）。

get([block[, timeout]])

从队列中取出并返回对象。如果可选参数 block 是 True （默认值）而且 timeout 是 None （默认值），将会阻塞当前进程，直到队列中出现可用的对象。如果 timeout 是正数，将会在阻塞了最多 timeout 秒之后还是没有可用的对象时抛出

queue.Empty 异常。反之（block 是 False 时），仅当有可用对象能够取出时返回，否则抛出 queue.Empty 异常（在这种情形下 timeout 参数会被忽略）。

close()

指示当前进程将不会再往队列中放入对象。一旦所有缓冲区中的数据被写入管道之后，后台的线程会退出。

join_thread()

等待后台线程。这个方法仅在调用了 close() 方法之后可用。这会阻塞当前进程，直到后台线程退出，确保所有缓冲区中的数据都被写入管道中。默认情况下，如果一个不是队列创建者的进程试图退出，它会尝试等待这个队列的后台线程。这个进程可以使用 cancel_join_thread() 让 join_thread() 方法什么都不做直接跳过。

cancel_join_thread()

防止 join_thread() 方法阻塞当前进程。具体而言，这防止进程退出时自动等待后台线程退出。

例 15.6 定义函数 HelloWord，形参为队列和 ID 号，获得当前进程的 name，把形如 ['Hello world!', ID,name] 这样的列表加入队列中。在主进程中创建 4 个子进程，同时在主进程中得到并输出子进程的列表。

```
import multiprocessing as mp
N_core=4#进程数
def HelloWord(q,ID):
    name=mp.current_process().name
    q.put([ 'Hello world!', ID,name])
    return
if __name__ == '__main__':
    q = mp.Queue()
    processes = []#进程列表
    for id in range(N_core):
        p = mp.Process(target=HelloWord,args=(q,id+1))
        p.start()
        processes.append(p)
    print(q.get())
    for p in processes:#等待各子进程处理结果
        p.join()
打开Anaconda Prompt，键入以下命令（文件位置要换成自己的位置）：
python E:\MyPython\并行计算\eg6.py
```

输出结果如下：

```
['Hello world!', 1, 'Process-1']
['Hello world!', 2, 'Process-2']
['Hello world!', 3, 'Process-3']
['Hello world!', 4, 'Process-4']
```

2. 管道（Pipe）

管道返回一个由管道连接的连接对象，默认情况下是双向。每个连接对象都有 send() 和 recv() 方法（相互之间的）。请注意，如果两个进程（或线程）同时尝试读取或写入管道的同一端，则管道中的数据可能会损坏。当然，在不同进程中同时使用管道的不同端的情况下不存在损坏的风险。其形式为：

multiprocessing.Pipe([duplex])

返回一对 Connection 对象 (conn1, conn2) ，分别表示管道的两端。如果 duplex 被置为 True （默认值），那么该管道是双向的。如果 duplex 被置为 False ，那么该管道是单向的，即 conn1 只能用于接收消息，而 conn2 仅能用于发送消息。

例 15.7 定义函数 HelloWord，形参为队列和 ID 号，获得当前进程的 name，把形如 ['Hello world!', ID,name] 这样的列表发送到主进程中。在主进程中创建 4 个子进程，同时在主进程中接收子进程传输的数据。

```
import  multiprocessing as mp
N_core=4#进程数
def HelloWord(conn,ID):
    name=mp.current_process().name
    conn.send(['Hello world!', ID,name])
    conn.close()
    return
if __name__ == '__main__':
    parent_conn, child_conn = mp.Pipe()
    processes = []#进程列表
    for id in range(N_core):
        p = mp.Process(target=HelloWord,args=(child_conn,id+1))
        p.start()
        processes.append(p)
    print(parent_conn.recv())
    for p in processes:#等待各子进程处理结果
        p.join()
```

运行情况同例15.6 。

15.4.3 同步

在 threading 模块中的同步机制，在多进程 multiprocessing 模块中也有，比如锁机制、事件、信号量、栅栏。

15.5 基于 GPU 线程的并行计算

前面两节是基于 CPU 的并行计算，实际上，在机器学习和深度学习中，使用更多的是基于 GPU 线程的并行计算，其优势在于核数众多，计算速度更快，特别对于矩阵乘法和卷积具有极大的计算优势。另外，GPU 还拥有大量且快速的寄存器以及 L1 缓存的易于编程性，使得 GPU 非常适合用于深度学习。目前，常用的 GPU 计算范式是来自英伟达的 CUDA 并行计算框架，这种架构可以通过 GPU 加速使得机器学习并行化。

15.5.1 CUDA 基本概念

CUDA并不是一个独立运行的计算平台，而需要与 CPU 协同工作，可以看成是 CPU 的协处理器，因此当我们在说 CUDA 并行计算时，其实是指的基于 CPU+GPU 的异构计算架构。在异构计算架构中，GPU 与 CPU 通过 PCIe 总线连接在一起来协同工作，如图15.9所示。

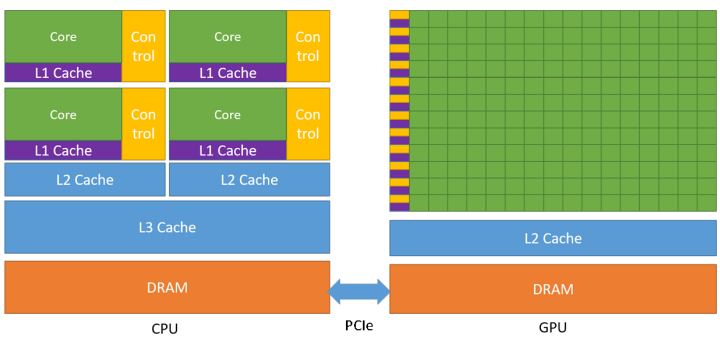


图 15.9 基于 CPU+GPU 的异构计算

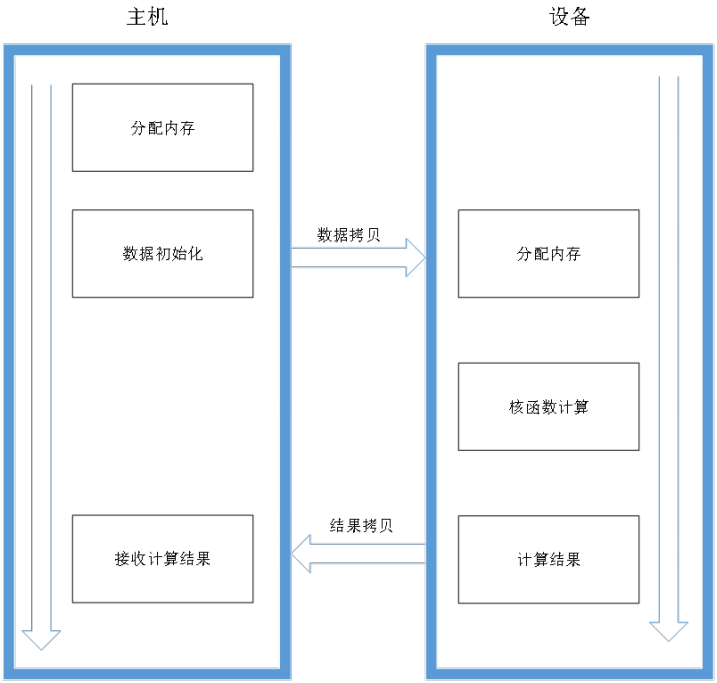


图 15.10 CUDA 执行流程

其中，CPU 所在位置称为为主机（Host），又称为主处理器，在并行计算中，主要涉及 CPU 及其内存，而 GPU 所在位置称为设备（Device），又称为协处理器，在并行计算中，主要涉及 GPU 及其内存。在 CUDA 程序中既包含主机程序，又包含设备程序，它们分别在 CPU 和 GPU 上运行。同时，主机与设备之间可以进行通信，即可以相互进行数据拷贝。在设备上的计算通过核函数（Kernel）形式执行，该函数以 _global_ 为前缀作为标记。如图15.10所示，典型的 CUDA 程序的执行流程如下：

（1）分配主机内存，并进行数据初始化；

（2）配设备内存，并从主机将数据拷贝到设备上；

（3）调用 CUDA 的核函数在设备上完成指定的运算；

（4）将设备上的运算结果拷贝到主机上；

（5）释放设备和主机上分配的内存。

15.5.2 CUDA 线程组织

如图15.11所示，CUDA 二维线程组织从逻辑层面来讲，主要分两层，外层称为网格（Grid），同一个网格上的线程共享相同的全局内存空间，里层称为线程块（Block）。对于网格，是由若干线程块组成，而每个线程块里面包含很多线程。线程坐标的原点在左上角，向右为 x 方向，向左为 y 方向。

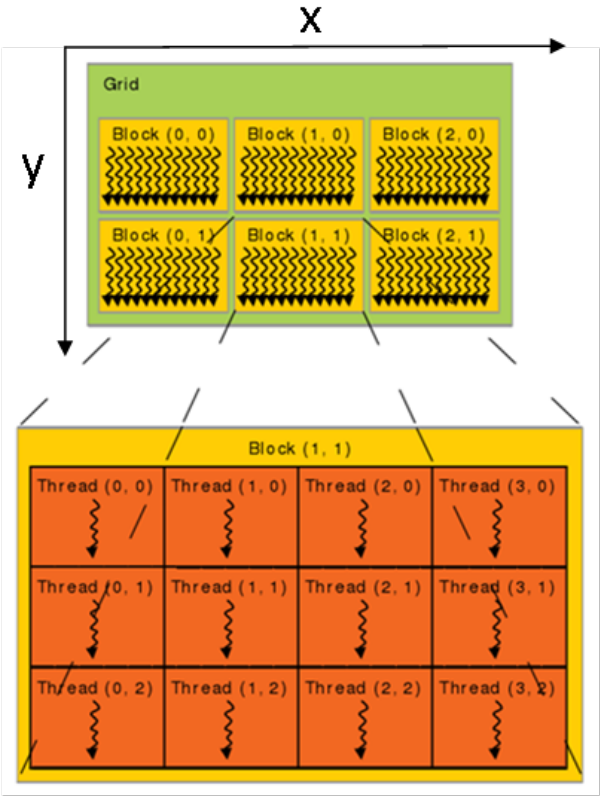


图 15.11 CUDA 二维线程组织

网格和线程块都是定义为 dim3 类型的变量，dim3 可以看成是包含三个无符号整数 (x, y, z) 成员的结构体变量，在定义时，缺省值初始化为 1。因此，网格和线程块可以灵活地定义为 1-dim，2-dim 以及 3-dim 结构。核函数在调用时也必须通过执行配置 <<<grid, block>>> 来指定核函数所使用的线程数及结构。图15.11中的网格和线程块可以这样定义：

```
dim3 grid(3, 2,1);
dim3 block(4, 3,1);
kernel_fun<<< grid, block >>>(prams...);
```

所以，一个线程需要两个内置的坐标变量 (blockIdx,threadIdx) 来唯一标识，它们都是 dim3 类型变量，其中 blockIdx 指明线程所在 grid 中的位置，而 threaIdx 指明线程所在 block 中的位置，如图中的 Thread(1,1) 满足：threadIdx.x = 1, threadIdx.y = 1, blockIdx.x = 1, blockIdx.y = 1。

对于每个线程，以一维为例，在实际计算中，坐标 x 的计算方式如图15.12，箭头所在位置的线程的坐标 $x = 5$.

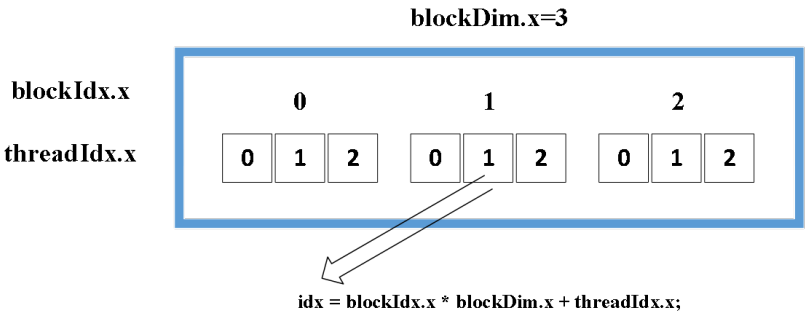


图 15.12 线程坐标 x 的计算方式

同理，可以得到二维的索引坐标：idx = blockIdx.x * blockDim.x + threadIdx.x; idy = blockIdx.y * blockDim.y + threadIdx.y。

对于核函数来讲，一般情况下，一个核函数运行在一个网格上，如图15.13所示。

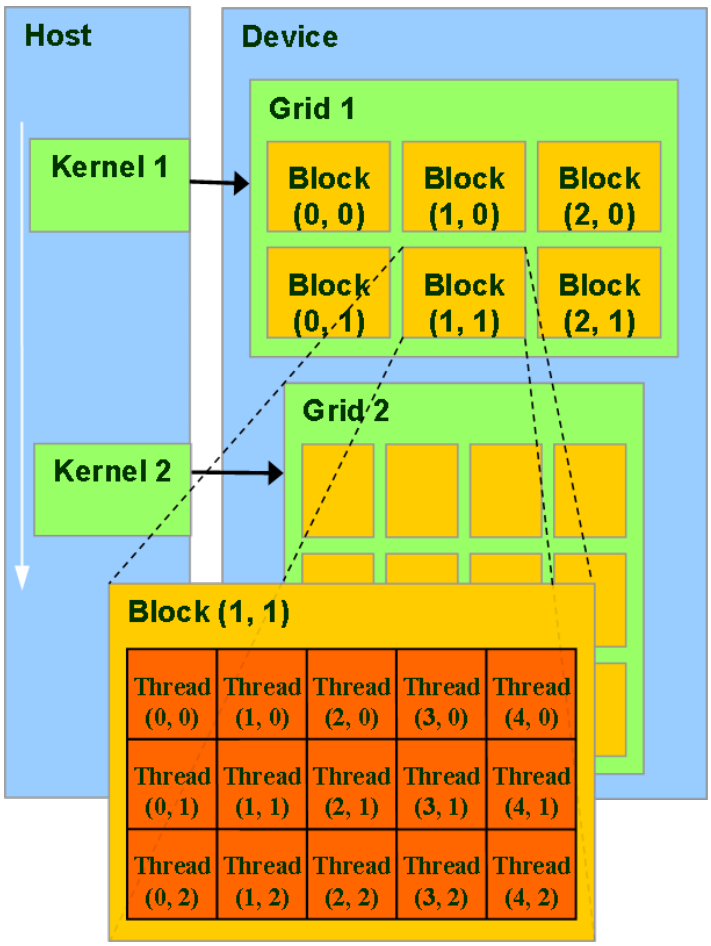


图 15.13 核函数与网格

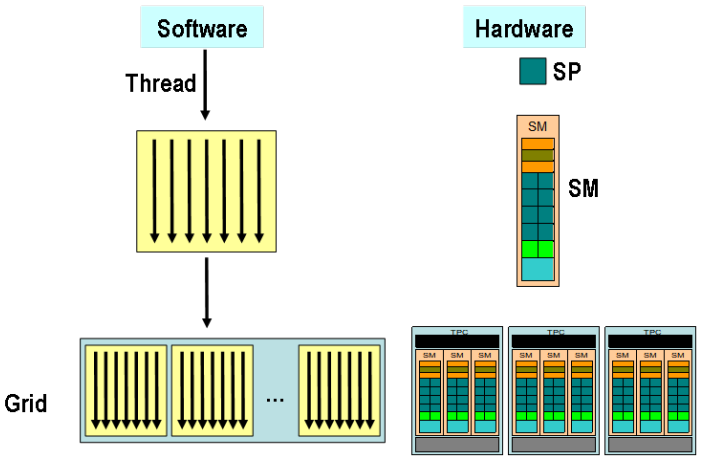


图 15.14 CUDA 编程的逻辑层和物理层

上面讲的是从软件逻辑层方面的线程组织方式。如图15.14所示，从显卡的物理层面来看，核心组件是流式多处理器（Streaming Multiprocessor），简称 SM，包括 CUDA 核心（也称为 Streaming Processor（SP））、共享内存、寄存器等，SM 可以并发地执行数百个线程，并发能力就取决于 SM 所拥有的资源数。当一个核函数被执行时，它的网格中的线程块被分配到 SM 上，一个线程块只能在一个 SM 上被调度，而 SM 一般可以调度多个线程块，基本的执行单元是线程束（Warps），线程束包含 32 个线程，这些线程同时执行相同的指令，但是每个线程都包含自己的指令地址计数器和寄存器状态，也有自己独立的执行路径。

15.5.3 CUDA 内存组织

CUDA 内存组织如图15.15所示。可以看到，每个线程有自己的私有本地内存（Local Memory），而每个线程块有包含共享内存（Shared Memory），可以被线程块中所有线程共享，其生命周期与线程块一致。此外，所有的线程都可以访问全局内存（Global Memory）。还可以访问一些只读内存块：常量内存（Constant Memory）和纹理内存（Texture Memory）。

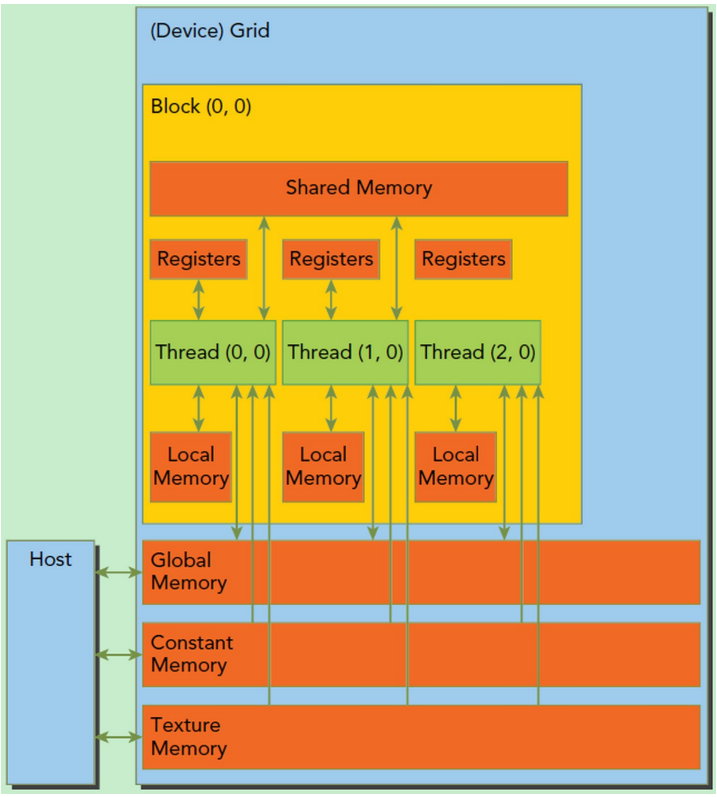


图 15.15 CUDA 内存组织

15.5.4 PyCUDA

PyCUDA 作为 Python 的第三方库，可以访问 NVDIA 的 CUDA 并行计算 API。它的异构编程模式和前面 CUDA 一致，它自己具有的特点是：

（1）对象的自动清理。这它使编写正确、无泄漏和无崩溃的代码变得更加容易。PyCUDA 在其分配的所有内存被释放之前，不会与上下文分离。

（2）编程方便。提供了 pycuda.compiler.SourceModule 和 pycuda.gpuarray.GPUArray 等接口，让 CUDA 编程甚至比 NVDIA 的基于 C 语言的运行时更方便。

（3）完整支持 CUDA。PyCUDA 能够完全支持 CUDA 的 API。并能够自动进行错误检查，把所有 CUDA 错误自动转换为 Python 异常。

（4）加速效果好。PyCUDA 的基础层是用 C++ 编写的，因此计算速度比较快。

1. **pycuda 安装**

到官网下载 CUDA 包并安装，到微软官网下载 Visual Studio 2015 以上的 C++ 开发工具并安装。到 Python 环境使用 pip install pycuda 安装 pycuda。

2. **实现过程**

下面以随机生成的两个向量的点积计算为例，讲解实现过程。其中，driver 是导入 CUDA 的 API，autoinit 是用来启动和自动初始化 GPU 系统，SourceModule 是 NVDIA 编译器（nvcc）的指令，指示要编译并上传到设备的对象。很明显，以下核函数是 C 语言形式。

```
#导入相关库
import pycuda.driver as cuda
import pycuda.autoinit
import numpy
from pycuda.compiler import SourceModule
#在CPU生成随机向量
a = numpy.random.randn(5).astype(numpy.float32)
b = numpy.random.randn(5).astype(numpy.float32)
#定义核函数
mod = SourceModule("""
__global__ void vector_dot(float *dest, float *a, float *b)
{const int i = threadIdx.x;dest[i] = a[i] * b[i];}
""")
func = mod.get_function("vector_dot")
#线程设置和执行核函数
dest = numpy.zeros_like(a)
func(drv.Out(dest), drv.In(a), drv.In(b), block=(400,1,1), grid=(1,1))
print(dest)#输出结果
```

15.5.5 TensorFlow

TensorFlow 是一个用于设计和部署数值计算的软件库，主要专注于机器学习中的应用程序。借助这个库，可以将算法描述为相关运算的图形，这些运算可以在各种支持 GPU 的平台上执行，包括便携式设备、台式机和高端服务器。

安装好 GPU 版的 TensorFlow 后，使用下面程序来测试 Tensorflow 是否支持 GPU：

```
import tensorflow as tf
tf.config.list_physical_devices()
```

如下输出结果表示支持 GPU：

```
 [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

可以用以下程序测试 CPU 和 GPU 下运行时间：

```
import tensorflow as tf
import timeit
def cpu_run():
    with tf.device('/cpu:0'):
    c = tf.matmul(cpu_a, cpu_b)
```

```
    return c
def gpu_run():
    with tf.device('/gpu:0'):
    c = tf.matmul(gpu_a, gpu_b)
    return c
cpu_time = timeit.timeit(cpu_run, number=10)
gpu_time = timeit.timeit(gpu_run, number=10)
print('run time:', cpu_time, gpu_time)
```

15.6 并行计算实践

以上内容介绍了并行计算的相关概念以及例题的讲解也是在 Python 框架讲解的。在本节我们除了继续介绍 Python 语言实践之外，也会补充 R 语言实践。

15.6.1 R 语言实践

在 R 环境下的并行计算主要通过它内置的任务并行包来实现，包括 parallel 包和 foreach 包，其中，parallel 包是针对任务的粗粒度并行计算，而 foreach 包则是针对 for 循环的细粒度并行计算。接下来分别介绍这两个包。

parallel 包

parallel 包是建立在 multicore 和 snow 包的基础上，涵盖了这两个包的大多数函数，并且加入了随机数的生成。由于是粗粒度并行计算，parallel 包处理比较大的计算块，一个典型的例子是在许多不同的数据集上计算相同的 R 函数。关键的一点是，这些计算块是不相关的，不需要以任何方式进行通信。通常情况下，这些块的时间长度大致相同，也就是负载基本均衡。其计算模型如下：

（1）启动 M 个 worker 进程（计算进程），并在 worker 进程上做任何需要的初始化。

（2）将每项任务所需的任何数据发送给 worker 进程。

（3）将任务分成大小大致相等的 M 个块，并将这些块（包括所需的 R 代码）发送给 worker 进程。

（4）等待所有的 worker 进程完成它们的任务，并获得它们的结果。

（5）对其它任务重复步骤（2）- （4）。

（6）关闭 worker 进程。

在并行计算前，先要获得该计算设备的核数，可以使用函数 detectCores 获取，一般用 detectCores(logical=F) 获取实际物理核心数。然后初始化并行计算环境的工

作是由 cl=makeCluster(cores) 完成，它可以申请 cores 个核参与计算。对于并行任务分配主要有两种方法：

一是针对表达式进行并行处理，由 clusterEvalQ(cl,expr) 函数利用创建的 cl 执行 expr。这里利用刚才创建的 cl 核并行计算 expr。expr 是执行命令的语句，不过如果命令太长的话，一般写到文件里比较好。比如把想执行的命令放在 Rcode.r 里：

```
clusterEvalQ(cl,source(file="Rcode.r"))
```

二是采用 par 开头的 apply 函数族进行任务并行计算。这族函数和 apply 的用法基本一样，不过要多加一个参数 cl。一般如果 cl 创建如上面 cl <- makeCluster(cl.cores) 的话，这个参数可以直接用作 parApply(cl=cl,...)。当然 Apply 也可以是 Sapply,Lapply 等等。注意 par 后面的第一个字母是要大写，而一般的 apply 函数族第一个字母不大写。

另外，parallel 包可以通过 library(help = “parallel”) 来查找相关信息。

例 15.8 定义 HelloWorld 函数，使其输出 Hello World! 和所在核的 ID 号。利用 parLapply 函数完成并行计算。

```
library(parallel)
HelloWorld <-function(i){
  sprintf('Hello World!The number:%d core ID is %d',i,Sys.getpid( ))
}
#cores <- detectCores()#获取计算机线程数
cores <- detectCores(logical=F)#获取实际物理核心数
cl <- makeCluster(cores)#初始化并行计算环境，申请cores个核参与计算
parLapply(cl,1:cores,HelloWorld)
stopCluster(cl)#终止并行计算
#输出结果如下:
[[1]]
[1] "Hello World!The number:1 core ID is 15472"
[[2]]
[1] "Hello World!The number:2 core ID is 18184"
[[3]]
[1] "Hello World!The number:3 core ID is 9168"
[[4]]
[1] "Hello World!The number:4 core ID is 3332"
```

注意每次计算的 ID 地址不一定一样。

例 15.9 利用 runif 函数生成 [0,1] 之间的随机向量，并进行向量加法运算。分别采用串行和并行计算，给出计算时间。

```
a <- runif(1000000, 0, 1)#生成[0,1]的随机向量
b <- runif(1000000, 0, 1)#生成[0,1]的随机向量
#串行计算
serial_logical_compute <-function(a,b){
  if(length(a) != length(b)){
```

```
    print("向量长度不一致")
    return(0)
  }
  c<-integer(length(a))#生成初始化为0的向量
  for(i in 1:length(a)){
    c[i] <- a[i] * b[i]
  }
  sum_c <- sum(c)
  print(sum_c)
}
start_time <- Sys.time()#起始时间
serial_logical_compute(a,b)
end_time <- Sys.time()#结束时间
s_time <- end_time-start_time
print(s_time)
#并行计算
library(parallel)
parallel_logical_compute <-function(id)
{
  if(length(a) != length(b)){
    print("向量长度不一致")
    return(0)
  }
  c<-integer(length(a))#生成初始化为0的向量
  for(i in seq(id,length(a),cores)){
    c[i] <- a[i] * b[i]
  }
  sum_c <- sum(c)
  print(sum_c)
}
start_time <- Sys.time()#起始时间
cores <- detectCores(logical=F)#获取实际物理核心数
cl <- makeCluster(cores,type="FORK")#初始化并行计算环境，申请cores个核参与计算
clusterExport(cl,c("a","b","cores"))#传入外部变量
result <- parLapply(cl,1:cores,parallel_logical_compute)
sum_c <- do.call(sum,result)
print(sum_c)
stopCluster(cl)#终止并行计算
end_time <- Sys.time()#结束时间
p_time <- end_time-start_time
print(p_time)
#输出结果如下:
[1] 250111
Time difference of 0.120677 secs
[1] 250111
Time difference of 1.94301 secs
```

从计算时间上看，并行计算的时间反而更长。当计算量少的时候，使用 parLapply 执行任务并行时，创建并行计算环境的时间远大于计算本身的时间，计算效率并不高。

例 15.10 利用 rnorm 生成 100 行 500000 列的随机矩阵，对矩阵的每列进行从小到大排序，并获取前 5 行数据。分别用串行计算和并行计算实现，计算加速比。

```
M <- matrix (rnorm (50000000) , 100 , 500000)#随机初始化矩阵
mysort <- function (x){
  return (sort (x) [1:5])#排序并返回前5数据
}
do_apply <- function (M) {
  return (apply (M , 2 , mysort))#对M矩阵每列从小到大数据排序
}
do_parallel <- function (M , ncl){#使用ncl个核并行处理M矩阵每列从小到大数据排序
  cl <- makeCluster (getOption ("cl.cores" , ncl))
  ans <- parApply (cl , M , 2 , mysort)
  stopCluster (cl)
  return (ans)
}
#串行计算
start_time <- Sys.time()#起始时间
ans1 <- do_apply ( M )
end_time <- Sys.time()#结束时间
s_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Serial computing time:%f secs',s_time)
#并行计算
library(parallel)
start_time <- Sys.time()#起始时间
cores <- detectCores(logical=F)#获取实际物理核心数
cl <- makeCluster(cores)#初始化并行计算环境，申请cores个核参与计算
ans2 <- do_parallel ( M , cores )
end_time <- Sys.time()#结束时间
p_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Parallel computing time:%f secs',p_time)
sprintf('Acceleration ratio:%f',s_time/p_time)
#输出结果如下:
[1] "Serial computing time:27.547455 secs"
[1] "Parallel computing time:17.326461 secs"
[1] "Acceleration ratio:1.589907"
```

从结果可以看出，数据计算之间无依赖性，且计算量非常大的时候，并行计算可以提高计算效率。

foreach 包

foreach 包是一个支持在 R 语言中调用多进程功能的第三方包，但必须与 doParallel 包一同使用，doParallel 包使 foreach 并行计算成为可能。foreach 的功能类似于 for 或者 lapply 函数，其最大的好处在于代码简单而且容易采用并行方式进行计算。foreach 函数形式为：

```
foreach(..., .combine, .init, .final=NULL, .inorder=TRUE,
        .multicombine=FALSE,
        .maxcombine=if (.multicombine) 100 else 2,
```

```
        .errorhandling=c('stop', 'remove', 'pass'),
        .packages=NULL, .export=NULL, .noexport=NULL,
        .verbose=FALSE)

when(cond)
e1 %:% e2
obj %do% ex
obj %dopar% ex
times(n)
```

各参数说明如下：

（1）第 1 个参数是必备参数，即必须有输入参数，结果默认返回 list；

（2）.combine：运算之后结果的显示方式，default 是 list，“c” 返回 vector，cbind 和 rbind 返回矩阵，”+” 和”*” 可以返回 rbind 之后的 “+” 或者 “*”，相当于归约操作；

（3）.init：.combine 函数的第一个变量；

（4）.final：返回最后结果；

（5）.inorder：TRUE 则返回和原始输入相同顺序的结果（对结果的顺序要求严格的时候），FALSE 返回没有顺序的结果（可以提高运算效率）。这个参数适合于设定对结果顺序没有需求的情况；

（6）.muticombine：设定.combine 函数的传递参数，default 是 FALSE 表示其参数是 2，TRUE 可以设定多个参数；

（7）.maxcombine：设定.combine 的最大参数；

（8）.errorhandling：如果循环中出现错误，对错误的处理方法；

（9）.packages：指定在 %dopar% 运算过程中依赖的 package(%do% 会忽略这个选项)，用于并行一些机器学习算法；

（10）.export: 在编译函数的时候需要预先加载一些内容进去，类似 parallel 的 clusterExport；

（11）e1 %:% e2，对象合并；

（12）%do% 严格按照顺序执行任务 (串行计算)，%dopar% 并行执行任务，%do% 时候就像 sapply 或 lapply，%dopar% 就是并行启动器；

（13）times(n)，返回计算时间，不过几乎不用。

例 15.11 对于级数 $\sum_{i=1}^{\infty}\frac{1}{i^2}=\frac{\pi^2}{6}$，令 $s(n)=\sum_{i=1}^{n}\frac{1}{i^2}$，计算当 n=10000 时的值。

```
library("foreach")
library("doParallel")
N <- 10000
pi <-3.1415926535
#串行计算
start_time <- Sys.time()#起始时间
sum <- foreach(i=1:N,.combine="+") %do% {1.0/i^2}
```

```
end_time <- Sys.time()#结束时间
s_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Serial computing time:%f secs',s_time)
sprintf('Serial computing result:%f,the error:%f',sum,abs(sum-pi^2/6))
#并行计算
start_time <- Sys.time()#起始时间
cores <- detectCores(logical=F)#获取实际物理核心数
cl <- makeCluster(cores)#初始化并行计算环境，申请cores个核参与计算
registerDoParallel(cl)#进行进程注册
sum <- foreach(i=1:N,.combine="+") %dopar% {1.0/i^2}
stopCluster(cl)
end_time <- Sys.time()#结束时间
p_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Parallel computing time:%f secs',p_time)
sprintf('Parallel computing result:%f,the error:$\%$f',sum,abs(sum-pi^2/6))
sprintf('Acceleration ratio:%f ',s_time/p_time)
#输出结果如下:
[1] "Serial computing time:4.662530 secs"
[1] "Serial computing result:1.644834,the error:0.000100"
[1] "Parallel computing time:6.940762 secs"
[1] "Parallel computing result:1.644834,the error:0.000100"
[1] "Acceleration ratio:0.671761"
```

从结果上看，计算值是一样的，但并行计算没有发挥出作用来，原因和前面例题中的情况一样，也是因为创建并行计算的环境的时间比计算本身还耗时，实际计算时应避免这种情况。

蒙特卡罗算法计算 π 值

例 15.12 单位圆的 1/4 面积是一个扇形，它是边长为 1 单位正方形的一部分，如图15.16所示。只要能求出扇形面积 S_1 在正方形面积 S 中占的比例 $K = S_1/S$ 就立即能得到 S_1，从而得到 π 的值。怎样求出扇形面积在正方形面积中占的比例 K 呢？一个办法是在正方形中随机投入很多点，使所投的点落在正方形中每一个位置的机会相等看其中有多少个点落在扇形内。将落在扇形内的点数 m 与所投点的总数 n 的比 m/n 作为 K 的近似值。点落在扇形内的充要条件是 $x^2 + y^2 \leq 1$。则

$$\frac{m}{n} = \frac{S_1}{S} = \frac{4}{\pi}, \pi = \frac{4m}{n}$$

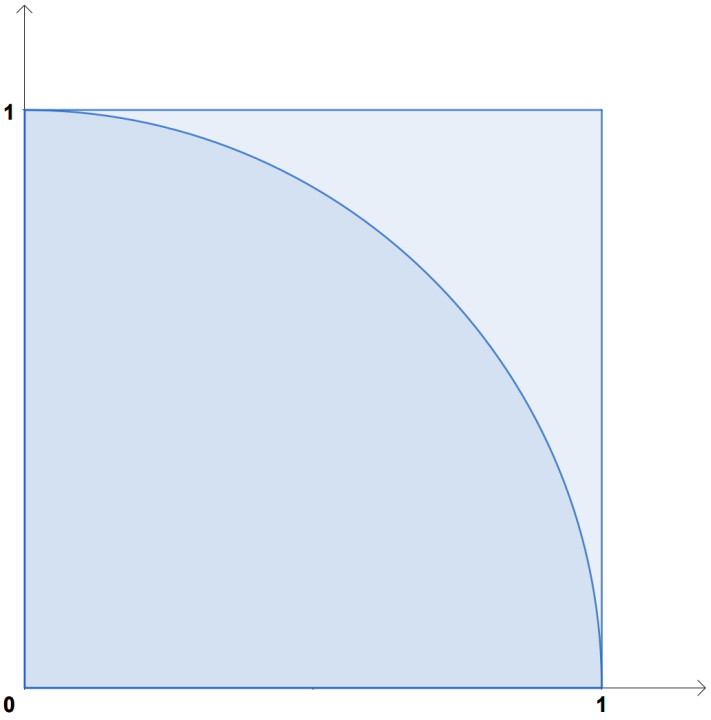


图 15.16 四分之一圆域

基于 R 语言的计算方法如下：

```
#串行计算
Serial_pi <- function(){
  N <- 1000000
  cnt <- 0
  for (i in 1:N){
    x <- runif(1, 0, 1)
    y <- runif(1, 0, 1)
    if ((x * x + y * y) < 1)
      cnt = cnt + 1
  }
return (4 * cnt / N)
}
start_time <- Sys.time()#起始时间
result <- Serial_pi()
end_time <- Sys.time()#结束时间
s_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Serial computing time:%f secs',s_time)
sprintf('Serial computing result:%f',result)
#并行计算
library(parallel)
parallel_pi <- function(id){
  N <- 1000000
  cnt <- 0
```

```
  for (i in seq(id,N,cores)){
    x <- runif(1, 0, 1)
    y <- runif(1, 0, 1)
    if ((x * x + y * y) < 1)
      cnt = cnt + 1
  }
return (4 * cnt / N)
}
start_time <- Sys.time()#起始时间
cores <- detectCores(logical=F)#获取实际物理核心数
cl <- makeCluster(cores)#初始化并行计算环境，申请cores个核参与计算
clusterExport(cl,c("cores"))#传入外部变量
result <- parLapply(cl,1:cores,parallel_pi)
result <- do.call(sum,result)
end_time <- Sys.time()#结束时间
p_time <- as.double(difftime(end_time,start_time),units="secs")#按秒计算时间差
sprintf('Parallel computing time:%f secs',p_time)
sprintf('Parallel computing result:%f',result)
sprintf('Acceleration ratio:%f',s_time/p_time)
#输出结果如下:
[1] "Serial computing time:11.167637 secs"
[1] "Serial computing result:3.141252"
[1] "Parallel computing time:2.148666 secs"
[1] "Parallel computing result:3.143928"
[1] "Acceleration ratio:5.197474"
```

从结果看，并行计算效果非常好，已经是超线性加速了。

15.6.2 Python 语言实践

蒙特卡罗算法计算 π 值

在上一小节中用 R 语言给出了具体计算过程，问题的详细说明也见上一小节，这里再用 Python 语言给出计算方法。容易写出其串行程序，具体程序如下:

```
from random import random
from math import sqrt
from time import perf_counter
times=10000000
counts=0
start = perf_counter()#计时开始
for i in range(1,times):
    x,y=random(),random()
    dist=sqrt(x**2+y**2)
    if dist <= 1.0:
        counts=counts+1
pi = 4*(counts/times)
end = perf_counter()#计时结束
print('pi=%f'%pi)
```

```
print('运行时间为: %s Seconds'%(end-start))
```

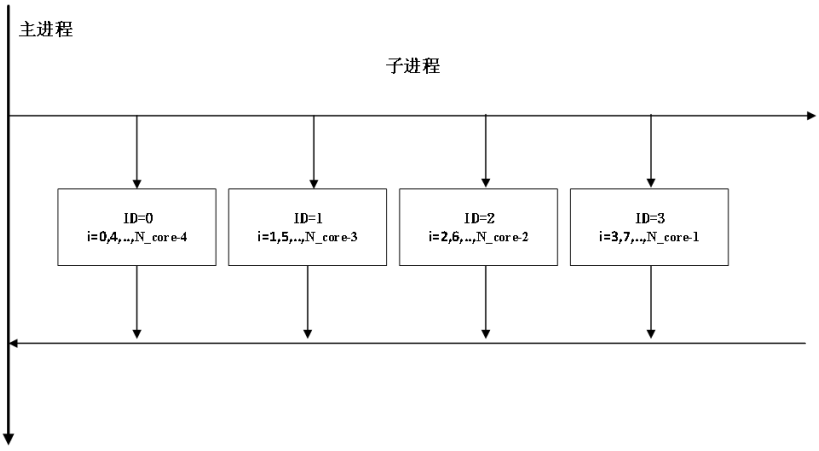


图 15.17 用 ID 号划分计算任务

从上面的程序中可以看出，计算是独立的，也就不存在数据竞争，完全可以进行并行计算。如果利用 multiprocessing 进行多核并行计算，只需要合理设计进程函数，每个进程通过调用该进程函数完成相应的计算，这就需要在进程函数内部合理分配计算任务，最好的方式就是均分，从而达到任务均衡。自定义进程 ID，若总进程为 N_core，则标记为 0，1，$\cdots$，N_core-1。如图15.17所示，根据线程 ID 号来分配不同的任务，把循环语句改造为：foriinrange(ID, times, N_core) 这样线程负载均衡，可用 multiprocessing 中 map 函数完成。并行程序实现如下：

```
##蒙特卡洛算法并行计算Pi值
from random import random
from math import sqrt
from time import perf_counter
import  multiprocessing as mp
times=10000000
N_core=4#进程数
def com_pi(ID):
    counts=0
    for i in range(ID,times,N_core):
        x,y=random(),random()
        dist=sqrt(x**2+y**2)
        if dist <= 1.0:
            counts=counts+1
    name=mp.current_process().name
    print('Current process name=%s,ID=%d \n'%(name,ID))
    return  counts
if __name__ == "__main__":
    ids = list(range(0,N_core))#id列表
    start = perf_counter()#计时开始
    pool = mp.Pool(processes=N_core) #创建进程池
```

```
    pool_result=pool.map(com_pi,ids)
    pool.close()
    pool.join()
    counts=sum(pool_result)#把各子进程计算结果汇总
    pi = 4*(counts/times)
    end = perf_counter()#计时结束
    name=mp.current_process().name#主进程输出结果
    print('Hello world! current_process name=%s \n'%name)
    print('pi值=%f,并行运行时间为: %s Seconds' %(pi,(end-start)))
```

打开 Anaconda Prompt，键入以下命令（文件位置要换成自己的位置），即可完成计算：python E:\MyPython\ 并行计算 \eg8.py，

为有利于计算加速比，把串行和并行程序放在一个文件 eg9.py 中，完整程序如下：

```
from random import random
from math import sqrt
from time import perf_counter
import  multiprocessing as mp
times=10000000
N_core=4#进程数
def serial_com_pi():
    counts=0
    for i in range(1,times):
    x,y=random(),random()
    dist=sqrt(x**2+y**2)
    if dist <= 1.0:
        counts=counts+1
    return  counts
def parallel_com_pi(ID):
    counts=0
    for i in range(ID,times,N_core):
        x,y=random(),random()
        dist=sqrt(x**2+y**2)
        if dist <= 1.0:
            counts=counts+1
    name=mp.current_process().name
    print('Current process name=%s,ID=%d \n'%(name,ID))
    return  counts
if __name__ == "__main__":
    #串行计算
    start = perf_counter()#计时开始
    counts=serial_com_pi()
    pi = 4*(counts/times)
    end = perf_counter()#计时结束
    serial_time=end-start
    print('pi值=%f,串行运行时间为: %s Seconds' %(pi,serial_time))
    #并行计算
    ids = list(range(0,N_core))#id列表
    start = perf_counter()#计时开始
    pool = mp.Pool(processes=N_core)#创建进程池
```

```
pool_result=pool.map(parallel_com_pi,ids)
pool.close()
pool.join()
counts=sum(pool_result)#把各子进程计算结果汇总
pi = 4*(counts/times)
end = perf_counter()# 计时结束
parallel_time = end - start
name = mp.current_process().name#主进程输出结果
print('Hello world! current_process name=%s \n'%name)
print('pi值=%f,并行运行时间为: %s Seconds' %(pi,parallel_time))
speedup = serial_time / parallel_time
print('加速比=%f' %speedup)
```

打开 Anaconda Prompt，键入以下命令（文件位置要换成自己的位置），即可完成计算：python E:\MyPython\ 并行计算 \eg9.py，

总结

本章节通过给出进程、线程等并行计算概念，讨论了在 Python 和 R 语言环境下基于 CPU 的多核并行计算方式，并讨论了基于 CUDA 的 GPU 并行计算原理，但并未给出分布式并行计算的机器学习方法，请参考刘铁岩所著的《分布式机器学习：算法、理论与实践》。

15.7 习题

使用各种并行计算方法计算下列各题，并进行性能比较。

1. 使用级数或蒙特卡洛方法计算底数 e。
2. 使用级数或蒙特卡洛方法计算 π。
3. 计算级数 $\sum_{i=0}^{\infty}\frac{(-2)^i}{(2i+1)!}$。
4. 使用蒙特卡洛方法计算三重积分 $\int_1^2\int_1^2\int_1^2\left(x^2+y^2+z^2\right)dxdydz$，算法描述如下：

取 0 到 1 之间均匀分布的随机数点列

$$\left(t_0,t_1,\cdots;t_{n-1}\right),i=0,1\cdots,m-1$$

令

$$x_j^j=a+\left(b_j-a\right)_j j^j j=0,1\cdots,n-1$$

取 m 足够大，则有

$$s=\frac{1}{m}\sum_{i=0}^{n+1}f\left(x_0^i,x_1^i,\cdot,x_n^i\right)\prod_{j=0}^{n-1}\left(b_j-a_j\right)$$

5. 使用本文中并行计算方法对前面各章涉及的机器学习算法（可选择一两个）进行并行化，比较其计算效率，并给出实验报告。

6. 对于例15.2和15.3，请用 multiprocessing 模块中同名的锁和事件重新完成。

第十六章　迁移学习

迁移学习是一种通过在一个任务上学到的知识来改善在另一个相关任务上的性能的方法，模型在一个源领域上训练，然后通过迁移学习的方法适应到一个不同但相关的目标领域上，常见于数据稀缺或者在新领域中数据收集成本较高的场景。相较于在线学习和并行计算处理相同的任务，迁移学习侧重学习知识在不同任务间的应用。

16.1　迁移学习的概述

迁移学习指的是通过从现有领域中迁移知识，从而有效且高效地实现目标的过程。迁移学习的概念最初诞生于心理学和教育学，心理学家所说的“学习迁移”，指的是一个学习过程对另一个学习过程的影响。它也常常出现在我们的日常生活中，例如，如果我们会打羽毛球，那么我们就可以学习打网球，因为打羽毛球和网球有一些相似的策略和技巧；如果我们会下中国象棋，那么我们就可以借鉴它与国际象棋之间的相似规则来学习下国际象棋等。

我们首先引入域这一重要概念，域是执行迁移学习的主体，它由数据和生成数据的分布两部分组成。我们经常用 $\mathcal{D}$ 来表示一个域，域的样本可以表示为输入 $\boldsymbol{X}$ 和输出 $\boldsymbol{Y}$，其概率分布表示为 $\Pr(\boldsymbol{X},\boldsymbol{Y})$，表明来自 $\mathcal{D}$ 的数据服从分布：$(\boldsymbol{X},\boldsymbol{Y}) \sim P(\boldsymbol{X},\boldsymbol{Y})$。使用 $\mathcal{X}$ 和 $\mathcal{Y}$ 分别表示特征空间和标签空间，对于任意样本 $(\boldsymbol{X}_i,\boldsymbol{Y}_i)$，有 $\boldsymbol{X}_i \in \mathcal{X}$，$\boldsymbol{Y}_i \in \mathcal{Y}$，一个域就可以定义为 $\mathcal{D} = \{\mathcal{X},\mathcal{Y},\Pr(\boldsymbol{X},\boldsymbol{Y})\}$。

在迁移学习中，至少有两个域：一个是拥有丰富知识的域，也就是我们迁移的域，另一个是我们想要学习的域。前者称为源域，后者称为目标域，我们经常用下标 s 和 t 来表示它们，$\mathcal{D}_s$ 是源域，$\mathcal{D}_t$ 是目标域。当 $\mathcal{D}_s \neq \mathcal{D}_t$ 时，$\mathcal{X}_s \neq \mathcal{X}_t$，$\mathcal{Y}_s \neq \mathcal{Y}_t$ 或 $\Pr_s(\boldsymbol{X},\boldsymbol{Y}) \neq \Pr_t(\boldsymbol{X},\boldsymbol{Y})$。

定义 16.1（迁移学习）给定一个源域 $\mathcal{D}_s = \{\boldsymbol{X}_i,\boldsymbol{Y}_i\}_{i=1}^{n_s}$ 和一个目标域 $\mathcal{D}_t = \{\boldsymbol{X}_j,\boldsymbol{Y}_j\}_{j=1}^{n_t}$，目标域用 ℓ 评估，其中 $\boldsymbol{X}_i \in \mathcal{X}$，$\boldsymbol{Y}_i \in \mathcal{Y}$。迁移学习的目标是利用源域数据学习一个预测函数 $f : \boldsymbol{X}_t \mapsto \boldsymbol{Y}_t$ 使 f 在目标域上达到最小预测风险：

$$f^* = \underset{f}{\operatorname{argmin}} \operatorname{E}_{(\boldsymbol{X},\boldsymbol{Y})\in\mathcal{D}_t} \ell\left(f\left(\boldsymbol{X}\right),\boldsymbol{Y}\right) \tag{16.1.1}$$

当以下三个条件之一成立时:

1. 不同的特征空间，即 $\mathcal{X}_s \neq \mathcal{X}_t$；
2. 不同的标签空间，即 $\mathcal{Y}_s \neq \mathcal{Y}_t$；
3. 具有相同特征和标签空间的不同概率分布，即 $\Pr_s(\boldsymbol{X}, \boldsymbol{Y}) \neq \Pr_t(\boldsymbol{X}, \boldsymbol{Y})$。

16.1.1 分布散度的度量

迁移学习的目标是开发算法来更好地利用现有知识并促进目标领域的学习，为了利用现有的知识，关键是找到两个领域之间的相似性，而相似性往往通过测量两个域之间的分布散度来计算。

根据概率论的基本知识，联合分布、边际分布和条件分布之间存在关系:

$$\Pr(\boldsymbol{X}, \boldsymbol{Y}) = \Pr(\boldsymbol{X}) \Pr(\boldsymbol{Y} \mid \boldsymbol{X}) = \Pr(\boldsymbol{Y}) \Pr(\boldsymbol{X} \mid \boldsymbol{Y}) \tag{16.1.2}$$

根据该公式，我们可以通过概率分布匹配的分类法将现有的迁移学习算法分为以下几类:

1. 边际分布适应 (MDA)
2. 条件分布适应 (CDA)
3. 联合分销适应 (JDA)
4. 动态分布适应 (DDA)

我们在图16.1中进一步说明了什么是边际分布、条件分布和联合分布，以便读者更直观地理解。显然，当目标域为图16.1中的情形 1 时，由于边缘分布与全局不同，我们应该更多地考虑匹配边缘分布；当目标域是情形 2 时，我们应该更多地考虑条件分布，因为它们从整体上看是相同的，只是与每个类的分布不同。

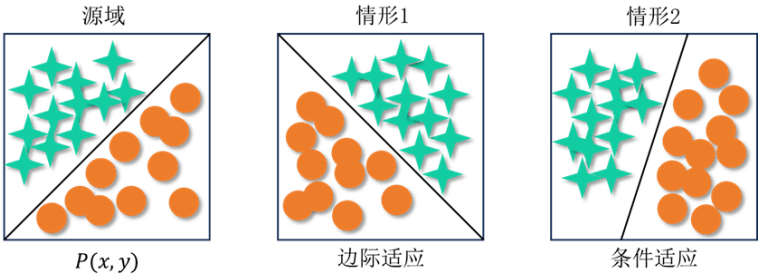


图 16.1 源域和目标域的不同分布情况

16.1.2 分布散度的统一表示

表16.1中给出了各种分布匹配方法的假设和问题，其中函数 $D(\cdot,\cdot)$ 表示分布匹配度量函数，我们将其作为预定义函数。从该表中可以清楚地看到，随着假设的变化，问题的定义也不同。动态分布适应（DDA）是最一般的情况，通过改变平衡因子μ 的值，可以很容易地将其表述为其他情况:

1. 当 $\mu=0$ 时，为边际分布适应；
2. 当 $\mu=1$ 时，为条件分布适应；
3. 当 $\mu=0.5$ 时，为联合分布自适应。

表 16.1 迁移学习中的分布度量

方法	假设	问题
边际分布适应	$\Pr_s(\boldsymbol{Y}\mid\boldsymbol{X})=\Pr_t(\boldsymbol{Y}\mid\boldsymbol{X})$	$\min\ D(\Pr_s(\boldsymbol{X}),\Pr_t(\boldsymbol{X}))$
条件分布适应	$\Pr_s(\boldsymbol{X})=\Pr_t(\boldsymbol{X})$	$\min\ D(\Pr_s(\boldsymbol{Y}\mid\boldsymbol{X}),\Pr_t(\boldsymbol{Y}\mid\boldsymbol{X}))$
联合分布适应	$\Pr_s(\boldsymbol{X},\boldsymbol{Y})\neq\Pr_t(\boldsymbol{X},\boldsymbol{Y})$	$\min\ D(\Pr_s(\boldsymbol{X}),\Pr_t(\boldsymbol{X}))+D(\Pr_s(\boldsymbol{Y}\mid\boldsymbol{X}),\Pr_t(\boldsymbol{Y}\mid\boldsymbol{X}))$
动态分布适应	$\Pr_s(\boldsymbol{X},\boldsymbol{Y})\neq\Pr_t(\boldsymbol{X},\boldsymbol{Y})$	$\min(1-\mu)D(\Pr_s(\boldsymbol{X}),\Pr_t(\boldsymbol{X}))$ $+\mu D(\Pr_s(\boldsymbol{Y}\mid\boldsymbol{X}),\Pr_t(\boldsymbol{Y}\mid\boldsymbol{X}))$

平衡因子 μ 的估计通常有两种方法: 随机猜测法（Random Guess）和最小最大值平均法（Min-Max Averaging），但这两种方法需要繁琐的计算，也不能从理论上很好地解释。Wang et al.（2018b）和 Wang et al.（2020）提出了迁移学习的动态分布适应方法，并首次提出了 μ 的精确估计。他们利用分布的全局和局部性质来计算 μ，Ben-David et al.（2007）采用$\mathcal{A}$-距离（$\mathcal{A}$-distance）作为基本的分布度量。$\mathcal{A}$-距离可以看作是利用二分类器的误差对两个域进行分类。

形式上，我们将 $\epsilon(h)$ 表示为分类器 h 对两个域 $\mathcal{D}_s$ 和 $\mathcal{D}_s$ 进行分类的误差, 则可以将 $\mathcal{A}$-距离定义为

$$d_{\mathcal{A}}(\mathcal{D}_s,\mathcal{D}_t)=2(1-2\epsilon(h)) \tag{16.1.3}$$

我们可以直接用方程来计算两个域之间边际分布的 $\mathcal{A}$-距离 d_M，而对于条件分布的 $\mathcal{A}$-距离，用 $d_c=d_{\mathcal{A}}\left(\mathcal{D}_S^{(c)},\mathcal{D}_t^{(c)}\right)$ 表示 c 类上的 $\mathcal{A}$-距离，其中 $\mathcal{D}_S^{(c)}$ 和 $\mathcal{D}_T^{(c)}$ 是 c 类的样本。因此，可以计算 μ 的值为

$$\widehat{\mu}=1-\frac{d_M}{d_M+\sum_{c=1}^{n_c}d_c}. \tag{16.1.4}$$

16.1.3 迁移学习的统一框架

基于机器学习中结构风险最小化的原理，我们建立了统一的迁移学习框架:

定义 16.2（迁移学习的统一框架）给定一个有标记的源域 $\mathcal{D}_s = \{\boldsymbol{X}_i, \boldsymbol{Y}_i\}_{i=1}^{n_s}$ 和一个未标记的目标域 $\mathcal{D}_t = \{\boldsymbol{X}_j\}_{j=1}^{n_t}$，它们的联合分布不同，即 $\Pr_s(\boldsymbol{X}, \boldsymbol{Y}) \neq \Pr_t(\boldsymbol{X}, \boldsymbol{Y})$，用 $f \in \mathcal{H}$ 表示目标函数，$\mathcal{H}$ 是它的假设空间。迁移学习的统一框架为

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(f\left(v_i \boldsymbol{X}_i\right), \boldsymbol{Y}_i\right) + \lambda R\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right), \tag{16.1.5}$$

- $\boldsymbol{v} = (v_1, \cdots, v_s) \in \mathcal{R}^{n_s}$ 表示源样本的权重，$v_i \in [0, 1]$，n_s 是源域样本数；
- $T(\cdot)$ 是两个域上的特征变换函数；
- 当引入权重 v_i 时，可以使用加权平均来代替平均值的计算。

我们使用 $R\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$ 代替结构风险最小化的正则化 $R(f)$，以更好地拟合迁移学习问题。在这个统一的框架下，迁移学习问题可以概括为寻找最优正则化泛函，这也意味着与传统的机器学习相比，迁移学习更关注源域和目标域之间的关系。

这个统一的框架总结了大多数迁移学习算法，具体来说，我们可以在式16.1.5中赋予 v_i 和 T 不同的值或函数改变它的形式，可以变换得到三种基本的迁移学习算法:

1. 实例加权方法（Instance Weighting Methods）：该方法的关键在于学习样本权重 v_i；
2. 特征变换方法（Feature Transformation Methods）：这种方法适合实际情况 $v_i = 1$，$\forall i$，其目标是学习一个特征变换函数 T 以减小正则化损失 $R(\cdot, \cdot)$；
3. 模型预训练方法（Model Pre-training Methods）：这种方法对应于 $v_i = 1$，$\forall i$，$R\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right) = R(\mathcal{D}_t; f_s)$，其目标是设计策略来正则化源函数 f_s 并对目标域进行微调。

这三种算法可以总结现有文献中大多数流行的迁移学习方法，我们将在后面的章节中详细介绍它们，在这里只对它们做一个简短的介绍。

16.2 实例加权方法

实例加权方法是迁移学习中最有效的方法之一，从技术上讲，任何加权方法都可用来评估每个实例的重要性。本节中，我们主要关注两种基本方法：实例选择方法和权重自适应方法。这两种方法在现有的迁移学习研究中得到了广泛的应用，也成为了更复杂系统的基本模块。

16.2.1 问题定义

迁移学习的核心思想是减少源域和目标域之间的分布差异，那么，实例加权方法能为实现这个目标做些什么呢？由于迁移学习中样本的维数和数量通常非常大，因此不可能直接估计 $\Pr_s(\boldsymbol{X})$ 和 $\Pr_t(\boldsymbol{X})$，相反，我们可以从标记的源域中选择一些样本，构成一个子集，使该子集的分布与目标域相似，然后，我们可以使用传统的机器学习方法建立模型，这种方法的关键是如何设计选择标准。另一方面，数据选择可以与如何设计样本加权规则完全相同。（数据选择可以看作是加权的特殊情况。例如，我们可以很容易地使用权重 1 和 0 来指示我们是否选择了一个样本。）

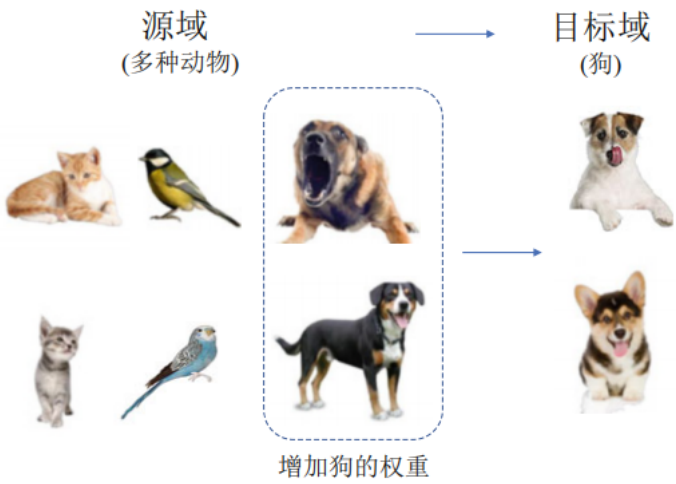


图 16.2 实例加权方法的介绍

图 16.2 显示了实例加权方法的基本思想：有一些动物属于不同的类别，如狗、猫和鸟类，但在目标域中只有一个主要的类别（狗）。在迁移学习中，为了使源域和目标域更相似，我们可以设计加权策略来增加狗类的权重，即在训练中赋予它更大的权重。

定义 16.3（迁移学习的实例加权）给定一个有标签的源域 $\mathcal{D}_s = \{(\boldsymbol{X}_i, \boldsymbol{Y}_i)\}_{i=1}^{n_s}$ 和一个无标签的目标域 $\mathcal{D}_t = \{(\boldsymbol{X}_j)\}_{j=1}^{n_t}$，且它们的联合分布不同，即 $\Pr_s(\boldsymbol{X}, \boldsymbol{Y}) \neq \Pr_t(\boldsymbol{X}, \boldsymbol{Y})$，设向量 $\boldsymbol{v} \in \mathcal{R}^{n_s}$ 表示源域中每个样本的权重，那么，实例加权方法的目标是学习一个最优的加权向量 $\boldsymbol{v}^*$，这样分布差异就可以最小化：

$$D\left(\Pr_s(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{v}), \Pr_t(\boldsymbol{X}, \boldsymbol{Y})\right) < D\left(\Pr_s(\boldsymbol{X}, \boldsymbol{Y}), \Pr_t(\boldsymbol{X}, \boldsymbol{Y})\right)$$

然后，目标的风险可以最小化：

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(v_i f(\boldsymbol{X}_i), \boldsymbol{Y}_i\right) + \lambda R(\mathcal{D}_s, \mathcal{D}_t)$$

其中向量 $\boldsymbol{v}$ 是我们的学习目标。

接下来我们将介绍实例选择方法和权重自适应方法，$v_i \in [0,1]$。

16.2.2 实例选择方法

实例选择方法通常假设源域和目标域之间的边际分布是相同的，即 $\Pr_s(\boldsymbol{x}) \approx \Pr_t(\boldsymbol{x})$，当它们的条件分布不同时，我们应该设计一些选择策略来选择合适的样本。事实上，如果我们把整个选择过程视为一个决策过程，那么它可以如图 16.3 所示：

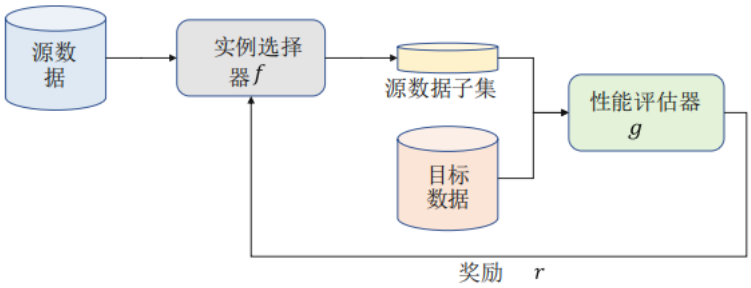


图 16.3 迁移学习的实例选择方法

本过程主要由以下各个模块组成：

- 实例选择器 f，它是从源域中选择与目标域分布相似的样本子集。
- 性能评估器 g，即评估所选子集与目标域之间的分布差异。
- 奖励 r，即为被选择的子集提供奖励，然后可以调整其选择过程。

上述过程可以被看作是强化学习中的一个马尔可夫决策过程（MDP）（Sutton 和 Barto，2018），因此，我们有了一个很自然的想法：我们可以通过设计适当的选择器、奖励和性能评估器，设计强化学习算法来完成这个任务。下面，我们根据实例选择方法是否采用强化学习算法，将其分为两类：基于非强化学习的方法和基于强化学习的方法。

首先介绍第一类基于非强化学习的方法。在深度学习之前，研究人员经常利用传统的学习方法来进行实例选择。现有的研究工作可以进一步分为三个类：基于距离测量的方法、基于元学习的方法和其他方法。

第二类是基于强化学习的方法（Feng 等人，2018）。该方法将源域划分为几个批次，然后学习每个批次中每个样本的权重。值得注意的是，为了度量域之间的分布差异，需要首先选择一些有标记的样本作为指导集，然后，在批级训练中指导集可以指导权重学习和特征学习过程。在应用强化学习方法时，定义状态、行动和奖励这三个关键概念是很重要的，例如，Liu 等人（2019）将这三个概念定义为：

- 状态由当前批的权向量和特征提取器的参数构成。
- 行动作为选择过程实现，因此它是一个二进制向量：0 表示没有选择这个样本，1

表示选择这个样本。

- 奖励是实现在源域和目标域之间的分布差异。

  奖励功能是基于强化学习的方法的关键，它被实现为

$$r\left(s,a,s'\right)=d\left(\Phi_{B_{j-1}}^{s},\Phi_{t}^{s}\right)-\gamma d\left(\Phi_{B_{j}}^{s'},\Phi_{t}^{s'}\right)$$

其中，下标 t 表示目标域，$d\left(\cdot,\cdot\right)$ 是一个分布测量函数，在实验中可以是 MMD 和 Reny 距离，(s,a,s') 表示从状态 s 采取动作 a 成为状态 s'，B_{j-1} 和 B_j 分别表示第 $j-1$ 次和第 j 次迭代时的批次，Φ 是一个特性。

16.2.3 权重自适应方法

与实例选择方法不同，权重自适应方法假设两个域之间的条件分布相同，而它们的边际分布不同，即 $\Pr_s\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)\approx\Pr_t\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)$ 但 $\Pr_s\left(\boldsymbol{X}\right)\neq\Pr_t\left(\boldsymbol{X}\right)$。受 Jiang 和 Zhai（2007）经典著作的启发，我们利用极大似然估计方法解决权重自适应问题。设 θ 表示模型的可学习参数，则目标域上的最优超参数可以表示为：

$$\theta_t^*=\arg\max_{\theta}\int_{\boldsymbol{X}}\sum_{\boldsymbol{Y}\in\mathcal{Y}}\Pr_t\left(\boldsymbol{X},\boldsymbol{Y}\right)\log\Pr\left(\boldsymbol{Y}\mid\boldsymbol{X};\theta\right)d\boldsymbol{X}$$

利用贝叶斯定理，上面的方程可以计算为：

$$\theta_t^*=\arg\max_{\theta}\int_{\boldsymbol{X}}\Pr_t\left(\boldsymbol{X}\right)\sum_{\boldsymbol{Y}\in\mathcal{Y}}\left[\Pr_t\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)\right]\log\Pr\left(\boldsymbol{Y}\mid\boldsymbol{X};\theta\right)d\boldsymbol{X}$$

其中只有一个未知量 $\Pr_t\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)$，这正是我们的学习目标。但是，我们只能利用 $\Pr_s\left(\boldsymbol{X},\boldsymbol{Y}\right)$，我们可以进行一些转换，规避条件分布 $\Pr_t\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)$ 的计算来学习 θ_t^*。构造两个概率之间的关系，然后利用它们的条件分布几乎相同 $\Pr_s\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)\approx\Pr_t\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)$ 这一假设来进行以下转换：

$$\begin{aligned}\theta_t^*&\approx\arg\max_{\theta}\int_{\boldsymbol{X}}\frac{\Pr_t\left(\boldsymbol{X}\right)}{\Pr_s\left(\boldsymbol{X}\right)}\Pr_s\left(\boldsymbol{X}\right)\sum_{\boldsymbol{Y}\in\mathcal{Y}}\Pr_s\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)\log\Pr\left(\boldsymbol{Y}\mid\boldsymbol{X};\theta\right)d\boldsymbol{X}\\&\approx\arg\max_{\theta}\int_{\boldsymbol{X}}\frac{\Pr_t\left(\boldsymbol{X}\right)}{\Pr_s\left(\boldsymbol{X}\right)}\widetilde{P}_s\left(\boldsymbol{X}\right)\sum_{\boldsymbol{Y}\in\boldsymbol{Y}}\widetilde{P}_s\left(\boldsymbol{Y}\mid\boldsymbol{X}\right)\log\Pr\left(\boldsymbol{Y}\mid\boldsymbol{X};\theta\right)d\boldsymbol{X}\\&\approx\arg\max_{\theta}\frac{1}{n_s}\sum_{i=1}^{n_s}\frac{\Pr_t\left(\boldsymbol{X}_i^s\right)}{\Pr_s\left(\boldsymbol{X}_i^s\right)}\log\Pr\left(\boldsymbol{Y}_i^s\mid\boldsymbol{X}_i^s;\theta\right)\end{aligned}$$

其中 $\dfrac{\Pr_t\left(\boldsymbol{X}_i^s\right)}{\Pr_s\left(\boldsymbol{X}_i^s\right)}$ 被称为密度比，它可以指导实例加权过程。我们可以利用密度比来建立源域和目标域之间的关系。这样，目标域上的参数就可以表示为：

$$\theta_t^*\approx\arg\max_{\theta}\frac{1}{n_s}\sum_{i=1}^{n_s}\frac{\Pr_t\left(\boldsymbol{X}_i^s\right)}{\Pr_s\left(\boldsymbol{X}_i^s\right)}\log\Pr\left(\boldsymbol{Y}_i^s\mid\boldsymbol{X}_i^s;\theta\right)$$

其中每一部分都可以学习，此时，这个问题可以得到解决。

从上述分析中我们知道，概率密度比可以帮助建立源分布和目标分布之间的关系。为简单起见，我们表示密度比为：

$$\beta_i = \frac{\Pr_t\left(\boldsymbol{X}_i^s\right)}{\Pr_s\left(\boldsymbol{X}_i^s\right)}$$

因此，向量 $\boldsymbol{\beta}$ 可以用来表示概率密度比。我们可以将目标域内的预测函数重新表述为：

$$f^* = \arg\min_{f\in\mathcal{H}} \sum_i^{n_s} \beta_i \ell\left(f\left(\boldsymbol{X}_i\right), \boldsymbol{Y}_i\right) + \lambda R\left(\mathcal{D}_s, \mathcal{D}_t\right)$$

上述公式是一种通用的表示算法，应用于逻辑回归、SVM、与特征转换方法（这将在下一节中介绍）集成等某些特定算法中时可以重新表述为特定公式。比如，权重自适应方法在与特征转换方法集成时，如果我们结合密度比和 MMD 的学习，那么它可以表示为：

$$\begin{aligned} MMD\left(\mathcal{D}_s, \mathcal{D}_t\right) &= \sup_f \mathrm{E}_{\mathrm{p}}[\frac{1}{n_s}\sum_{i=1}^{n_s}\beta_i f\left(\boldsymbol{X}_i\right) - \frac{1}{n_t}\sum_{j=1}^{n_t} f\left(\boldsymbol{X}_j\right)] \\ &= \frac{1}{n_s^2}\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta} - \frac{2}{n_t^2}\boldsymbol{\kappa}^{\mathrm{T}}\boldsymbol{\beta} + const \end{aligned}$$

通过采用核技巧，上述问题可以表述为：

$$\begin{aligned} &\min_{\boldsymbol{\beta}} \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta} - \boldsymbol{\kappa}^{\mathrm{T}}\boldsymbol{\beta} \\ &\text{s.t.}\boldsymbol{\beta} \in [0, B], |\sum_{i=1}^{n_s}\beta_i - n_s| \leq n_s\epsilon \end{aligned}$$

这被称为核均值匹配算法（KMM）（Huang 等人，2007），其中 ϵ 和 B 是预定义的阈值。

有很多的实例权重自适应方法，值得注意的是，这类方法可以直接集成到深度学习中，以学习样本权重。

16.3 统计特征变换方法

在本节中，我们介绍迁移学习中的统计特征变换方法，这种方法在近期的深度神经网络研究中非常流行且能够得到很好的结果。本节的叙述安排如下，首先给出特征变换方法的定义，详细介绍基于最大均值差异 MMD 的特征变换方法和基于度量学习的特征变换方法。

16.3.1 特征变换方法及问题定义

我们之前已经接触过许多统计特征，如均值、方差、假设检验等，我们着重关注在迁移学习领域广泛应用的几个统计特征。

定义 16.4 给定一个有标签的源域 $\mathcal{D}_s = \{(\boldsymbol{X}_i, Y_i)\}_{i=1}^{n_s}$ 和一个无标签的目标域 $\mathcal{D}_t = \{(\boldsymbol{X}_j)\}_{j=1}^{n_t}$，且它们的联合分布不同，即 $P_s(\boldsymbol{X}, Y) \neq P_t(\boldsymbol{X}, Y)$，特征变换的核心是学习特征变换函数 T 从而得到最优预测函数 f：

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(f\left(\boldsymbol{X}_i\right), Y_i\right) + \lambda R\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$$

基于分布散度度量函数的性质，可以给出如下两类特征变换函数的定义：

定义 16.5 若采用一些预定义的或已有的分布散度度量函数来度量两个分布之间的散度，并进行特征变换，称这里的特征变换是显式的特征变换(Explicit Feature Transformation)，它具有预定义的或已有的分布散度度量 $D(\cdot, \cdot)$：

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(f\left(\boldsymbol{X}_i\right), Y_i\right) + \lambda D\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$$

常见的预定义的度量函数有欧几里得距离、余弦相似度、KL 散度和最大均值差异 MMD (Maximum Mean Discrepancy)。

定义 16.6 若分布散度度量函数是由模型经过学习得到的而非预先定义的，称该特征变换为隐式特征变换(Implicit Feature Transformation)，它具有可学习的度量函数 Metric $(\cdot, \cdot)$

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(f\left(\boldsymbol{X}_i\right), Y_i\right) + \lambda \operatorname{Metric}\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$$

上述的隐式特征变换包括度量学习、几何特征对齐和对抗学习。

16.3.2 基于最大均值差异 MMD 的方法

最大均值差异(Maximum Mean Discrepancy) 是使用最多的统计距离度量函数之一，它最初是一种应用于假设检验中有效的双样本检测手段。本小节中我们主要介绍 MMD 的理论细节并叙述其在迁移学习中的使用。

我们用 $\mathcal{H}_k$ 表示一个以 k 为核函数的再生核希尔伯特空间 (RKHS)，一个概率分布 p 的核均值嵌入记为 $\mu_k(p)$ ，即 $\mu_k(p)$ 为空间 $\mathcal{H}_k$ 的唯一元素，也就是说，对任意 $\mathcal{H}_k$ 中的函数 f, 都有

$$E_{\boldsymbol{X}\sim p} f(\boldsymbol{X}) = \langle f(\boldsymbol{X}), \mu_k(p) \rangle_{\mathcal{H}_k}$$

用 $d_k(p,q)$ 表示两概率分布 p 和 q 的最大均值差异，那么其平方即为再生核希尔伯特空间中的两均值嵌入的距离:

$$d_k^2(p,q) \triangleq \| E_{\boldsymbol{X}\sim p}[\phi(\boldsymbol{X})] - E_{\boldsymbol{X}\sim q}[\phi(\boldsymbol{X})] \|_{\mathcal{H}_k}^2$$

其中 $\phi(\cdot)$ 定义了一个由原空间映射到再生核希尔伯特空间的映射函数，其核函数定义为映射函数的内积:

$$k\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right) = \left\langle \phi\left(\boldsymbol{X}_i\right), \phi\left(\boldsymbol{X}_j\right) \right\rangle$$

其中 $\langle \cdot, \cdot \rangle$ 表示内积运算符。MMD 的作用就是将两个分布映射到另一个空间，然后计算它们的均值差异。因此我们可以利用上述式子去计算两个分布的散度。

下面我们介绍基于 MMD 的迁移学习。首先回忆前一小节中的式:

$$f^* = \arg\min_{f\in\mathcal{H}} \frac{1}{n_s} \sum_{i=1}^{n_s} \ell\left(f\left(\boldsymbol{X}_i\right), Y_i\right) + \lambda R\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$$

建立 MMD 与特征变换函数 T 之间的关系，分布散度的一般形式可表示为

$$D\left(\mathcal{D}_s, \mathcal{D}_t\right) \approx (1-\mu) D\left(P_s(\boldsymbol{X}), P_t(\boldsymbol{X})\right) + \mu D\left(P_s(Y \mid \boldsymbol{X}), P_t(Y \mid \boldsymbol{X})\right)$$

可以看出，MMD 可以直接用于计算边际分布散度 $D\left(P_s(\boldsymbol{X}), P_t(\boldsymbol{X})\right)$ ，这与经典的迁移学习方法迁移成分分析 (TCA) 相对应。

我们使用半定矩阵 $\mathbf{A}$ 来表示使用 MMD 计算的特征变换矩阵，那么矩阵 $\mathbf{A}$ 即为我们在基于 MMD 的迁移学习方法中的学习目标, 两个边际分布之间的 MMD 可以表示为

$$\operatorname{MMD}\left(P_s(\boldsymbol{X}), P_t(\boldsymbol{X})\right) = \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \mathbf{A}^{\mathrm{T}} \boldsymbol{X}_i - \frac{1}{n_t} \sum_{j=1}^{n_t} \mathbf{A}^{\mathrm{T}} \boldsymbol{X}_j \right\|_{\mathcal{H}}^2 \tag{16.3.1}$$

下面首先介绍一个概念称为充分统计(Sufficient Statistics)：在样本量充分大时，若存在许多未知的统计量, 我们可以选择一些已知的统计量来近似我们的目标量。我们利用这个概念近似计算目标域上的分布 $P_t(Y \mid \boldsymbol{X})$。根据贝叶斯公式 $P_t(Y \mid \boldsymbol{X}) = P_t(Y) P_t(\boldsymbol{X} \mid Y)$, 忽略 $P_t(Y)$ 项，就可以用类条件概率 $P_t(\boldsymbol{X} \mid Y)$ 来近似 $P_t(Y \mid \boldsymbol{X})$ ，而目标域是无标签的，通常采用迭代式的训练策略：首先，使用 $(\boldsymbol{X}_s, Y_s)$ 来训练一个分类器 (例如 KNN 和逻辑回归等)，从而得到未标记目标域的伪标签 $\widehat{Y}_t$, 这个伪

标签可以用于迁移学习，经过特征变换后，伪标签就能够在以后的迭代中进行更新。使用这种方法我们可以计算得到 $P_t(\boldsymbol{X} \mid Y)$ 从而得到两条件分布间的 MMD 如下：

$$\operatorname{MMD}\left(P_s(Y \mid \boldsymbol{X}), P_t(Y \mid \boldsymbol{X})\right) = \sum_{c=1}^{C} \left\| \frac{1}{N_s^{(c)}} \sum_{\boldsymbol{X}_i \in \mathcal{D}_s^{(c)}} \mathbf{A}^{\mathrm{T}} \boldsymbol{X}_i - \frac{1}{N_t^{(c)}} \sum_{\boldsymbol{X}_j \in \mathcal{D}_t^{(c)}} \mathbf{A}^{\mathrm{T}} \boldsymbol{X}_j \right\|_{\mathcal{H}}^2$$

其中 C 表示所有分类的总数，$N_s^{(c)}$ 和 $N_t^{(c)}$ 分别表示源域和目标域中第 c 类样本集 $\mathcal{D}_s^{(c)}$ 和 $\mathcal{D}_t^{(c)}$ 中的样本个数。经求解得到基于 MMD 的迁移学习方法形式为

$$\min \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}} \mathbf{X} \mathbf{M} \mathbf{X}^{\mathrm{T}} \mathbf{A}\right) \tag{16.3.2}$$

其中 $\operatorname{tr}(\cdot)$ 表示矩阵的迹，$\mathbf{X}$ 是由源域和目标域中的特征所构成的矩阵，$\mathbf{M}$ 是 MMD 矩阵，计算公式为:

$$\mathbf{M} = (1-\mu)\mathbf{M}_0 + \mu \sum_{c=1}^{C} \mathbf{M}_c$$

其中边际 MMD 矩阵与条件 MMD 矩阵计算公式分别为

$$(\mathbf{M}_0)_{ij} = \begin{cases} \dfrac{1}{n_s^2}, & \boldsymbol{X}_i, \boldsymbol{X}_j \in \mathcal{D}_s \\ \dfrac{1}{n_t^2}, & \boldsymbol{X}_i, \boldsymbol{X}_j \in \mathcal{D}_t \\ -\dfrac{1}{n_s n_t}, & \text{otherwise} \end{cases}$$

$$(\mathbf{M}_c)_{ij} = \begin{cases} \dfrac{1}{\left(n_s^{(c)}\right)^2}, & \boldsymbol{X}_i, \boldsymbol{X}_j \in \mathcal{D}_s^{(c)} \\ \dfrac{1}{\left(n_t^{(c)}\right)^2}, & \boldsymbol{X}_i, \boldsymbol{X}_j \in \mathcal{D}_t^{(c)} \\ -\dfrac{1}{n_s^{(c)} n_t^{(c)}}, & \begin{cases} \boldsymbol{X}_i \in \mathcal{D}_s^{(c)}, \boldsymbol{X}_j \in \mathcal{D}_t^{(c)} \\ \boldsymbol{X}_i \in \mathcal{D}_t^{(c)}, \boldsymbol{X}_j \in \mathcal{D}_s^{(c)} \end{cases} \\ 0, & \text{otherwise} \end{cases}$$

详细的求解步骤篇幅较长此处从略。

最小化式 (16.3.2) 的同时需要考虑到其约束条件，在这里我们考虑特征变换前后的协方差矩阵。给定样本 $\boldsymbol{X}$ ，它的协方差矩阵 $\mathbf{S}$ 可表示为

$$\begin{aligned} \mathbf{S} &= \sum_{j=1}^{n} \left(\boldsymbol{X}_j - \bar{\boldsymbol{X}}\right)\left(\boldsymbol{X}_j - \bar{\boldsymbol{X}}\right)^{\mathrm{T}} \\ &= \sum_{j=1}^{n} \left(\boldsymbol{X}_j - \bar{\boldsymbol{X}}\right) \otimes \left(\boldsymbol{X}_j - \bar{\boldsymbol{X}}\right) \\ &= \left(\sum_{j=1}^{n} \boldsymbol{X}_j \boldsymbol{X}_j^{\mathrm{T}}\right) - n \overline{\boldsymbol{X}} \overline{\boldsymbol{X}}^{\mathrm{T}} \end{aligned}$$

其中 $\bar{\boldsymbol{X}} = \dfrac{1}{n}\sum_{j=1}^{n} \boldsymbol{X}_j$ 表示样本均值，$\otimes$ 表示外积。用 $\mathbf{H} = \mathbf{I} - (1/n)\mathbf{1}$ 表示中心矩阵，$\mathbf{I} \in \mathcal{R}^{(n+m)\times(n+m)}$ 表示单位矩阵，则协方差矩阵可用矩阵表示为

$$\mathbf{S} = \mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}$$

将 $\mathbf{A}$ 代入上式得到方差的最大化表示：

$$\max \left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\right) \mathbf{H} \left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\right)^{\mathrm{T}}$$

结合 (16.3.2) 式，可以得到基于 MMD 的迁移学习方法的最终形式为

$$\begin{aligned} &\min \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right) + \lambda\|\mathbf{A}\|_F^2, \\ &\text{s.t.} \quad \mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A} = \mathbf{I} \end{aligned} \tag{16.3.3}$$

其中正则项 $\lambda\|\mathbf{A}\|_F^2$ 的引入可以保证问题的良定性，$\lambda > 0$ 是一个超参数。

通常使用 Lagrange 方法求解该最优化问题，首先构造 Lagrange 函数：

$$L = \operatorname{tr}\left(\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}\right)\mathbf{A}\right) + \operatorname{tr}\left(\left(\mathbf{I} - \mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right)\Phi\right)$$

令 $\partial L/\partial \mathbf{A} = 0$，得到

$$\left(\mathbf{X}\mathbf{M}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}\right)\mathbf{A} = \mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\Phi$$

其中 Φ 为 Lagrange 乘子。利用 Python 或者 MATLAB 容易求解上式得到变换矩阵 $\mathbf{A}$ 。

我们可以使用前文叙述的多次迭代的方法使目标域的伪标签更准确，从而改进我们的最终结果。同时，如果将 μ 取为不同的值，我们就可以得到相应的边际、条件和联合分布方法的最终形式，这一部分留给读者完成。

基于 MMD 的迁移学习的完整步骤总结如下：

(1) 输入两个特征矩阵，使用一个简单的分类器（如 KNN）来计算目标域的伪标签；

(2) 计算矩阵 $\mathbf{M}$ 和 $\mathbf{H}$；

(3) 选择一些常见的核函数 (如线性核、多项核、高斯核等) 来计算核；

(4) 求解等式 (16.3.3) 得到源域和目标域的变换特征。取它的前 m 个特征，即为矩阵 $\mathbf{A}$；

(5) 可以进一步使用多次迭代的方法使伪标签更加准确。

16.3.3 基于度量学习的方法

在本节中, 我们将介绍如何使用度量学习得到特征变换 T, 也就是

$$f^* = \arg\min_{f\in\mathcal{H}} \frac{1}{n_s}\sum_{i=1}^{n_s} \ell\left(f\left(\boldsymbol{X}_i\right), Y_i\right) + \lambda \operatorname{Metric}\left(T\left(\mathcal{D}_s\right), T\left(\mathcal{D}_t\right)\right)$$

中的 Metric($\cdot$, $\cdot$)。

首先给出度量学习的基本介绍。度量学习(Metric Learning) 是机器学习中的一个十分重要的研究方向。实际上, 度量两个样本之间的距离涉及到分类、回归和聚类等重要问题, 选择一个好的度量可以用来构造好的特征表示从而建立一个更好的模型。度量学习在计算机视觉、文本挖掘和生物信息学等领域有广泛的应用。可以说, 如果没有适当的度量, 在机器学习中就没有好的模型。欧几里得距离、马氏距离、余弦相似度和 MMD 都是度量的例子, 它们都是预定义的距离。而在某些特定的应用场景中, 这些度量并不能保证我们的模型总是获得最好的表现, 因此我们转而求助于度量学习。

度量学习的基本过程是为给定的样本集计算一个更好的距离度量, 使该度量能够反映数据集的重要性质。这些样品通常包含一些先验知识, 度量学习算法可以基于这些先验知识建立目标函数从而得出一个针对这些样本的更好的度量。从这个角度来看, 度量学习可以看作是在一定条件下的一个最优化问题。

度量学习的核心是聚类假设: 属于同一聚类的数据很有可能属于同一个类。因此, 度量学习重点关注成对的距离, 同时考虑类内距离和类间距离的作用。为了评估样本之间的相似性, 度量学习采用线性判别分析 (LDA) 中的方法来计算类内和类间的距离, 目标是使得类间距离较大, 类内距离较小, 分别用 $S_c^{(M)}$ 和 $S_b^{(M)}$ 分别表示类内和类间的距离, 计算方法如下:

$$S_c^{(M)} = \frac{1}{nk_1}\sum_{i=1}^{n}\sum_{j=1}^{n} P_{ij} d^2\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$$

$$S_b^{(M)} = \frac{1}{nk_2}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij} d^2\left(\boldsymbol{X}_i, \boldsymbol{X}_j\right)$$

其中 P_{ij} 和 Q_{ij} 分别表示类内和类间距离, 若 $\boldsymbol{X}_i$ 为 $\boldsymbol{X}_j$ 的 k_1 个邻居之一, 则 $P_{ij}=1$, 否则为 0 ; 类似地, 若 $\boldsymbol{X}_i$ 为 $\boldsymbol{X}_j$ 的 k_2 个邻居之一, 则 $Q_{ij}=1$, 否则为 0 。$d(\cdot,\cdot)$ 的定义将在下文中介绍。

现有的迁移学习方法大多基于一个预定义的距离函数, 比如上一小节介绍的 MMD, 而在一般场合下, 这种度量并不能很好的推广。通过在迁移学习中使用度量学习的方法, 我们能够得到更好的距离度量函数。我们将 MMD 集成到度量学习中, 从而得到以下优化目标函数:

$$J = S_c^{(M)} - \alpha S_b^{(M)} + \beta D_{\mathrm{MMD}}\left(\boldsymbol{X}_s, \boldsymbol{X}_t\right)$$

其中 β 为表示权重的超参数。令 $\mathbf{M} \in \mathcal{R}^{d\times d}$ 为一个半定矩阵，则样本 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_j$ 之间的马氏距离可表示为

$$d_{ij} = \sqrt{(\boldsymbol{X}_i - \boldsymbol{X}_j)^{\mathrm{T}} \mathbf{M} (\boldsymbol{X}_i - \boldsymbol{X}_j)}$$

由于 $\mathbf{M}$ 为半定的，由半定矩阵的性质可分解为 $\mathbf{M} = \mathbf{A}^T\mathbf{A}$，其中 $\mathbf{A} \in \mathcal{R}^{d\times d}$，则上式可化为

$$d_{ij} = \sqrt{(\boldsymbol{X}_i - \boldsymbol{X}_j)^{\mathrm{T}} \mathbf{M} (\boldsymbol{X}_i - \boldsymbol{X}_j)} = \sqrt{(\mathbf{A}\boldsymbol{X}_i - \mathbf{A}\boldsymbol{X}_j)^{\mathrm{T}} (\mathbf{A}\boldsymbol{X}_i - \mathbf{A}\boldsymbol{X}_j)}$$

以上结果表明，要得到马氏距离矩阵 $\mathbf{M}$ 等同于找到源域和目标域之间的线性特征变换 $\mathbf{A}$ 。因此，我们不需要直接计算 $\mathbf{M}$，而是通过上一小节的方法计算线性变换矩阵 $\mathbf{A}$（或使用核方法进行非线性变换）来计算 d_{ij} ，故最优化过程也可仿照上一小节进行。

16.4 几何特征变换方法

在这一小节，我们介绍用于迁移学习的几何特征变换方法，这与上一小节中的统计特征变换有所不同。几何特征变换可以利用潜在的几何特征来获得简洁有效的表示，并具有显著的性能。与统计特征相似，也有许多几何特征。我们主要介绍三种类型的几何特征变换方法：子空间学习、流形学习和最优传输方法。这些方法在方法论上有所不同，但在迁移学习中都很重要。

16.4.1 子空间学习方法

几何特征变换方法属于隐含的特征变换方法。因此，尽管我们不能直接测量分布散度，但可以通过应用几何特征变换来降低。

子空间学习经常假设经过特征变换后源数据和目标数据在子空间中有相似的分布。在子空间中，我们可以实行分布对齐和使用传统机器学习方法来建立模型。对齐的概念有着直观的几何信息：如果来自两个领域的数据是对齐的，我们认为它们之间的分布差距是最小的。因此，子空间学习可用于分布对齐。子空间对齐是一种经典的子空间学习方法。SA 的目标是找到一种可以进行域对齐的线性特征变换 $\mathbf{M}$ 。让 $\mathbf{X}_s$ 和 $\mathbf{X}_t$ 分别表示对源特征 $\boldsymbol{S}$ 和目标特征 $\boldsymbol{T}$ 进行 PCA 变换的 d 维特征矩阵，称为子空间。然后，SA 的目标可以表述为

$$F(\mathbf{M}) = ||\mathbf{X}_s\mathbf{M} - \mathbf{X}_t||_F^2$$

特征变换矩阵 $\mathbf{M}$ 的最优解 $\mathbf{M}^*$ 可以表示为

$$\mathbf{M}^* = \arg\min_{\mathbf{M}} F(\mathbf{M})$$

因此，由于子空间学习的正交性，即 $\mathbf{X}_s^{\mathrm{T}}\mathbf{X}_s = \mathbf{I}$，我们可以直接得到上述问题的封闭解。

$$F(\mathbf{M}) = ||\mathbf{X}_s^{\mathrm{T}}\mathbf{X}_s\mathbf{M} - \mathbf{X}_s^{\mathrm{T}}\mathbf{X}_t||_F^2 = ||\mathbf{M} - \mathbf{X}_s^{\mathrm{T}}\mathbf{X}_t||_F^2$$

因此，特征变换矩阵 $\mathbf{M}$ 的最优解计算为 $\mathbf{M}^* = \mathbf{X}_s^{\mathrm{T}}\mathbf{X}_t$。这意味着源域和目标域相同时（即 $\mathbf{X}_s = \mathbf{X}_t$），那么 $\mathbf{M}^*$ 应为单位阵。我们称 $\mathbf{X}_a = \mathbf{X}_s\mathbf{X}_s^{\mathrm{T}}\mathbf{X}_t$ 为目标对齐源坐标系，它通过下式将源域转换为一个新的子空间。

$$\boldsymbol{S}_a = \boldsymbol{S}\mathbf{X}_a$$

相似地，目标域可以转换为 $\boldsymbol{T}_t = \boldsymbol{T}\mathbf{X}_t$ 。最终我们可以使用 $\boldsymbol{S}_a$ 和 $\boldsymbol{T}_t$ 建立机器学习模型，而不是原始的特征 $\boldsymbol{S}$ 和 $\boldsymbol{T}$。这使得 SA 在实践中实现起来非常简单。基于 SA, Sun 和 Saenko(2015) 提出了子空间分布对齐 (SDA)，将概率分布适应加入到子空间学习中。具体来说，SDA 认为除了子空间学习矩阵 $\mathbf{G}$，我们也应该加入一个分布变换矩阵 $\mathbf{A}$。SDA 的优化目标被表述为

$$\mathbf{M} = \mathbf{X}_s\mathbf{G}\mathbf{A}\mathbf{X}_t^G$$

然后，我们可以得到两个域的变换特征，然后按照 SA 中类似的步骤建立模型。

不同于 SA 和 SDA 只进行一阶对齐，Sun 等人提出了相关性对齐（CORAL）来进行二阶对齐。假设 $\mathbf{C}_s$ 和 $\mathbf{C}_t$ 分别是源域和目标域的协方差矩阵，那么 CORAL 学习了一个二阶特征变换矩阵 $\mathbf{A}$ 来降低它们的分布差异：

$$\min_{\mathbf{A}} ||\mathbf{A}^{\mathrm{T}}\mathbf{C}_s\mathbf{A} - \mathbf{C}_t||_F^2$$

这样，源特征和目标特征可以通过下式进行变换

$$z^r = \begin{cases} \mathbf{X}^r \cdot (\mathbf{C}_s + \mathbf{E}_s)^{-\frac{1}{2}} \cdot (\mathbf{C}_t + \mathbf{E}_t)^{\frac{1}{2}}, & r = s \\ \mathbf{X}^r, & r = t \end{cases} \tag{16.4.1}$$

其中 $\mathbf{E}_s$ 和 $\mathbf{E}_t$ 分别是与源域和目标域相同大小的单位矩阵。我们可以把这一步骤看做每个子空间一个重新着色的过程，其中方程（16.4.1）通过重新着色去噪后的源特征，使其与目标分布的协方差对齐。

CORAL 随后被扩展到深度神经网络中，被称为 **DCORAL(深度 CORAL)**(Sun 和 Saenko, 2016)。在 DCORAL 中，CORAL 被用于构建网络中的一个适应性损失，可以替换现有的 MMD 损失。深度学习中的 CORAL 损失定义为

$$\ell_{\mathrm{CORAL}} = \frac{1}{4d^2}||\mathbf{C}_s - \mathbf{C}_t||_F^2$$

其中 d 是特征维度的数量。CORAL 的计算也非常简单，不需要调整任何超参数。此外，CORAL 在领域自适应和领域泛化方面也取得了优异的性能。在本章的实践部分，我们将展示 CORAL 在比其他方法更简便的情况下，也能取得优异的性能。

16.4.2 流形学习方法

自 2000 年在《科学》杂志首次提出以来（Seung 和 Lee, 2000），流形学习已成为机器学习和数据挖掘领域的热门研究话题。流形学习通常假设当前数据是从高维空间中采样的，并且具有低维流形结构。流形是一类几何对象。一般来说，我们无法直接从原始数据中观察到隐藏结构，但我们可以想象数据位于高维空间中，其中具有某种可观察的形状。一个很好的例子是天空中星座的形状。为了描述所有星座，我们可以想象它们在天空中具有某种形状，从而产生了许多受欢迎的星座，如天琴座和猎户座。经典流形学习方法包括等距映射、局部线性嵌入和拉普拉斯特征映射等（Zhou，2016；Bishop，2006）。流形学习的核心是利用几何结构简化问题。距离测量在流形学习中也很重要，因为我们可以利用几何结构获得更好的距离。那么，在流形学习中，两点之间的最短路径是什么？在二维空间中，两点之间的最短距离是线段。但在三维、四维或 n 维空间（$n > 4$）中呢？实际上，当我们展开地球空间时，地球上两点之间的最短路径是直线。这条直线实际上是一条曲线，被称为测地线。一般来说，测地线距离是任何空间中任意两点之间的最短路径。例如，图16.4显示了在三维空间中，球体上两点 A 和 B 之间的最短路径是曲线。

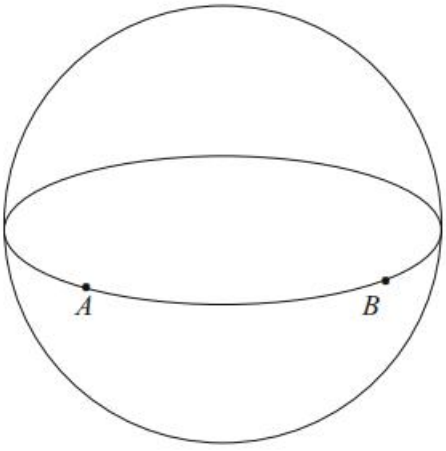


图 16.4 两点间的测地距离

我们熟悉的欧几里得空间也是一种流形结构。事实上，惠特尼嵌入定理（Greene 和 Jacobowitz，1971）表明，任何流形都可以嵌入到高维欧几里得空间中，这使得

通过流形进行计算成为可能。流形学习方法通常采用流形假设（Belkin 等人，2006），即数据在其流形嵌入空间中通常具有与邻居相似的几何性质。

由于流形空间中的数据通常具有良好的几何性质，可以克服特征失真，因此可以采用流形学习进行迁移学习。在许多现有的流形中，Grassmann 流形 $\mathbb{G}(d)$ 将原始 d 维空间视为其元素，从而便于分类器学习。在 Grassmann 流形中，特征变换和分布适应通常具有有效的数值形式，可以容易地解决（Hamm 和 Lee，2008）。因此，我们可以使用 Grassmann 流形进行迁移学习（Gopalan 等人，2011；Baktashmotlagh 等人，2014）。

基于流形学习的迁移学习受到增量学习的启发：如果一个人想要从当前点移动到另一个点，他需要一步一步地走。然后，如果我们将源域和目标域视为高维空间中的两个点，那么我们可以模拟人类的行走过程，逐步进行特征变换，直到到达目标域。图16.5 简要展示了这一过程。在该图中，源域通过特征变换函数 $\Phi(\cdot)$ 从起始点移动到终点，以完成流形学习。

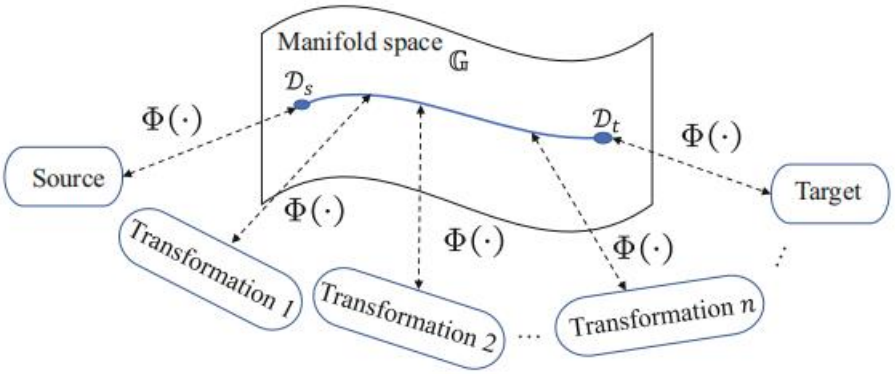


图 16.5 流形迁移学习的说明

早期的方法将这个问题优化为在流形空间中采样 d 个点，然后构建一个测地线流。我们只需要找到每个点的变换。这种类型的方法被称为采样测地线流（SGF）（Gopalan 等人，2011），这是第一个提出这种想法的工作。然而，SGF 有几个局限性：我们应该找到多少个中间点，以及如何做高效计算？之后，Gong 等人提出了测地流核（GFK）来解决这个问题。简单来说，为确定中间点，GFK 采用一种核学习方法来利用测地线上无限个点。

我们用 $\mathcal{S}_s$ 和 $\mathcal{S}_t$ 表示 PCA 变换后的子空间。然后，$\mathbf{G}$ 可以被看作是所有 d 维空间的集合，并且每个 d 维子空间可以被看作是 $\mathbf{G}$ 上的一个点。因此，两点之间的测地线$\boldsymbol{\Phi}(t) : 0 \leq t \leq 1$ 是这两点之间的最短路径。

若我们令 $\mathcal{S}_s = \boldsymbol{\Phi}(0)$，$\mathcal{S}_t = \boldsymbol{\Phi}(1)$ 那么寻找从 $\boldsymbol{\Phi}(0)$ 到 $\boldsymbol{\Phi}(1)$ 的测地路径等同于将原始特征转换为无限维空间，并且最终减少域偏移问题。特别地，流形空间中的特征可以表示为 $\boldsymbol{Z} = \boldsymbol{\Phi}(t)^{\mathrm{T}}\mathbf{X}$，经过变换的特征 $\boldsymbol{Z}_i$ 和 $\boldsymbol{Z}_j$ 定义了一个正半定测地线

流核：

$$\langle \boldsymbol{Z}_i, \boldsymbol{Z}_j \rangle = \int_0^1 (\boldsymbol{\Phi}(t)^{\mathrm{T}} \boldsymbol{X}_i)^{\mathrm{T}} (\boldsymbol{\Phi}(t)^{\mathrm{T}} \boldsymbol{X}_j) \, \mathrm{d}t = \boldsymbol{X}_i^{\mathrm{T}} \mathbf{G} \boldsymbol{X}_j$$

测地线流核计算为

$$\boldsymbol{\Phi}(t) = \mathbf{P}_s \mathbf{U}_1 \boldsymbol{\Gamma}(t) - \mathbf{R}_s \mathbf{U}_2 \boldsymbol{\Sigma}(t) = \begin{bmatrix} \mathbf{P}_s & \mathbf{R}_s \end{bmatrix} \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Gamma}(t) \\ \boldsymbol{\Sigma}(t) \end{bmatrix}$$

其中 $\mathbf{R}_s \in \mathcal{R}^{D \times d}$ 是 $\mathbf{P}_s$ 的补元素。$\mathbf{U}_1 \in \mathcal{R}^{D \times d}$ 和 $\mathbf{U}_2 \in \mathcal{R}^{D \times d}$ 是两个正交矩阵：

$$\mathbf{P}_s^{\mathrm{T}} \mathbf{P}_t = \mathbf{U}_1 \, \mathbf{V}^{\mathrm{T}}, \boldsymbol{R}_s^{\mathrm{T}} \mathbf{P}_t = -\mathbf{U}_2 \, \mathbf{V}^{\mathrm{T}}$$

核 $\mathbf{G}$ 可以计算为

$$\mathbf{G} = \begin{bmatrix} \mathbf{P}_s \mathbf{U}_1 & \mathbf{R}_s \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Lambda}_1 & \boldsymbol{\Lambda}_2 \\ \boldsymbol{\Lambda}_3 & \boldsymbol{\Lambda}_4 \end{bmatrix} \begin{bmatrix} \mathbf{U}_1^{\mathrm{T}} \mathbf{P}_s^{\mathrm{T}} \\ \mathbf{U}_2^{\mathrm{T}} \mathbf{R}_s^{\mathrm{T}} \end{bmatrix}$$

其中，$\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2, \boldsymbol{\Lambda}_3$ 是三个对角矩阵，它们的角度 θ_i 由奇异值分解 (SVD) 计算得出：

$$\begin{aligned} \lambda_{1i} &= \int_0^1 \cos^2(t\theta_i) \, \mathrm{d}t = 1 + \frac{\sin(2\theta_i)}{2\theta_i}, \\ \lambda_{2i} &= -\int_0^1 \cos(t\theta_i) \sin(t\theta_i) \, \mathrm{d}t = \frac{\cos(2\theta_i) - 1}{2\theta_i}, \\ \lambda_{3i} &= \int_0^1 \sin^2(t\theta_i) \, \mathrm{d}t = 1 - \frac{\sin(2\theta_i)}{2\theta_i} \end{aligned} \tag{16.4.2}$$

然后，原始空间中的特征可以通过 $\boldsymbol{Z} = \sqrt{\mathbf{G}} \boldsymbol{X}$ 映射到 Grassmann 流形。核 $\mathbf{G}$ 可以使用 SVD 轻松计算。GFK 的实现相当简单，并且可以作为许多方法的特征处理步骤。例如，在流形嵌入分布对齐（MEDA）（Wang 等人，2018）中，作者建议在应用分布对齐之前使用 GFK 进行特征提取。如图16.6所示，GFK 的集成提高了算法的性能和稳健性。

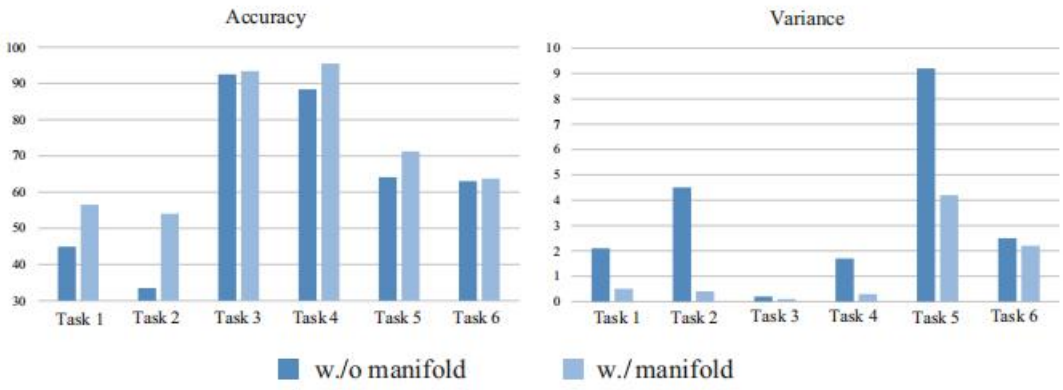


图 16.6 添加流形学习方法的 MEDA 的准确性和方差

在后续研究中，秦等人（2019）提出了时变 GFK，通过添加时间变量来扩展原始 GFK，以解决跨领域活动识别问题。他们声称 GFK 的“行走”过程是一个马尔可夫过程：当前变换仅依赖于前一个时间步。当 GFK 逐渐将源域转换为目标域的子空间时，后者的变换具有更好的影响。对于 $t_1 \leq t_2$，t_2 时刻的变换对最终结果的影响应该比 t_1 时刻更大。时变广义卡尔曼滤波器（temporally adaptive GFK）将时间变量添加到方程（16.4.2）中：

$$\begin{aligned}
\lambda_{1i} &= \int_0^1 t\cos^2(t\theta_i)\,\mathrm{d}t = \frac{1}{4} - \frac{1}{4\theta_i^2}\sin^2\theta_i + \frac{1}{4\theta_i}\sin 2\theta_i, \\
\lambda_{2i} &= -\int_0^1 t\cos(t\theta_i)\sin(t\theta_i)\,\mathrm{d}t = \frac{\cos(2\theta_i)}{4\theta_i} - \frac{\sin 2\theta_i}{8\theta_i^2}, \\
\lambda_{3i} &= \int_0^1 t\sin^2(t\theta_i)\,\mathrm{d}t = \frac{1}{4} + \frac{1}{4\theta_i^2}\sin^2\theta_i - \frac{1}{4\theta_i}\sin 2\theta_i
\end{aligned}$$

时间自适应 GFK 的性能甚至优于原始 GFK。此外，还有许多其他流形迁移学习方法。例如，Baktashmotlagh 等人（2014）提出了使用 Riemann 流形中的 Hellinger 距离来计算从源域到目标域的变换。Guerrero 等人（2014）还提出了一种联合流形适应方法。

16.4.3 最优传输方法

在这一部分，我们介绍了基于最优传输的迁移学习方法，它提供了几何特征变换的另一个视角。最优传输是一个经典的研究领域，已经研究了很长时间。最优传输具有优美的理论基础，因此为数学、计算机科学和经济学中许多应用的独特研究提供了独特的研究意义。最优传输（OT）最初由法国数学家 Gaspard Monge 于 18 世纪提出。第二次世界大战期间，俄罗斯数学家和经济学家 Kantorovich 对其产生了极大关注，为线性规划奠定了基础。1975 年，Kantorovich 因其在最优资源分配方面的贡献获得了诺贝尔经济学奖。我们通常将经典的 OT 问题称为 Monge 问题。

最优传输具有可靠的实践环境。我们将展示一个例子来说明这一点。杰克和罗斯一起成长。他们的家人靠经营仓库为生。有一天，罗斯的房子着火了，她急需一些应急用品。现在，杰克必须挺身而出帮助她！我们假设杰克有 n 个不同的仓库。每个仓库都有一定数量的套件，我们表示为 $\{G_i\}_{i=1}^n$，其中 G_i 是第 i 个仓库的套件数量。这些 n 个仓库的位置表示为 $\{\boldsymbol{X}_i\}_{i=1}^n$。同样，罗斯有 m 个不同的仓库，其位置表示为 $\{\boldsymbol{Y}_i\}_{i=1}^m$。每个仓库需要 $\{H_i\}_{i=1}^m$ 个套件。我们用 $\{c(\boldsymbol{X}_i, \boldsymbol{X}_j)\}_{i,j=1}^{m,n}$ 表示 Jack 的仓库 i 到 Rose 的仓库 j 之间的距离。已知运输成本会随着距离的增加而增加。那么，我们的问题是：杰克如何才能以最低成本帮助罗斯？我们使用一个矩阵 $\mathbf{T} \in \mathcal{R}^{n\times m}$ 来表示运输关系，其中每个元素 T_{ij} 表示从杰克的仓库 i 到罗斯的仓库 j 的套件数

量。然后，这个问题可以表示为

$$\min \sum_{i,j=1}^{n,m} T_{ij} c(\boldsymbol{X}_i, \boldsymbol{Y}_j)$$
$$\text{s.t.} \quad \sum_j T_{ij} = G_i, \ \sum_i T_{ij} = H_j$$

这是一个最优传输的应用。我们可以将仓库和套件分别看做概率分布和随机变量。然后，最优运输的形成被定义为确定将分布 $\Pr(\boldsymbol{X})$ 转换为 $Q(\boldsymbol{Y})$ 的最小成本，其公式为

$$L = \arg\min_{\pi} \iint_{\boldsymbol{X},\boldsymbol{Y}} \pi(\boldsymbol{X}, \boldsymbol{Y}) c(\boldsymbol{X}, \boldsymbol{Y}) d\boldsymbol{X} d\boldsymbol{Y} \tag{16.4.3}$$

给定以下约束条件（$\pi(\boldsymbol{X}, \boldsymbol{Y})$ 是它们的联合分布）：

$$\int_{\boldsymbol{Y}} \pi(\boldsymbol{X}, \boldsymbol{Y}) d\boldsymbol{Y} = \Pr(\boldsymbol{X})$$
$$\int_{\boldsymbol{X}} \pi(\boldsymbol{X}, \boldsymbol{Y}) d\boldsymbol{X} = Q(\boldsymbol{Y})$$

上述方程表明，最优传输是关于分布的连接，这显然与迁移学习有关。

为了在迁移学习中使用最优传输，我们修改方程（16.4.3）以得到由最优传输定义的分布散度：

$$D(P, Q) = \inf_{\pi} \iint_{\boldsymbol{X} \times \boldsymbol{Y}} \pi(\boldsymbol{X}, \boldsymbol{Y}) c(\boldsymbol{X}, \boldsymbol{Y}) d\boldsymbol{X} d\boldsymbol{Y} \tag{16.4.4}$$

我们常用L2 距离来计算，即

$$c(\boldsymbol{X}, \boldsymbol{Y}) = ||\boldsymbol{X} - \boldsymbol{Y}||_2^2$$

通过采用 L2 距离，方程（16.4.4）变成二阶 Wasserstein 距离

$$W_2^2(P, Q) = \inf_{\pi} \int_{\boldsymbol{X} \times \boldsymbol{Y}} \pi(\boldsymbol{X}, \boldsymbol{Y}) ||\boldsymbol{X} - \boldsymbol{Y}||_2^2 d\boldsymbol{X} d\boldsymbol{Y}$$

不同于传统的特征变换方法，最优传输研究点之间的耦合矩阵 $\mathbf{T}$。然后，在 $\mathbf{T}$ 的映射之后，源分布可以以最小成本映射到目标域。对于一个数据分布 μ，经过重力映射和耦合矩阵 $\mathbf{T}$ 之后，我们可以得到分布 μ。它的新特征向量是

$$\widehat{\boldsymbol{X}}_i = \arg\min_{\boldsymbol{X} \in \mathcal{R}^d} \sum_j \mathbf{T}(i, j) c(\boldsymbol{X}, \boldsymbol{X}_j)$$

那么，如何确定这个耦合矩阵 $\mathbf{T}$ 呢？这通常与成本有关。在最优传输中，我们通常使用变换成本来评估成本，用 $C(\mathbf{T})$ 表示。带有概率度量 μ 的 $\mathbf{T}$ 的成本定义为

$$C(\mathbf{T}) = \int_{\Omega_s} c(\boldsymbol{X}, \mathbf{T}(\boldsymbol{X})) d\mu(\boldsymbol{X})$$

其中 $c(\boldsymbol{X}, \mathbf{T}(\boldsymbol{X}))$ 是成本函数，也可以理解为距离函数。我们可以使用以下变换将源分布转换为目标分布：

$$\gamma_0 = \arg\min_{\gamma \in \Pi} \int_{\Omega_s \times \Omega_t} c(\boldsymbol{X}^s, \boldsymbol{X}^t) d\gamma(\boldsymbol{X}^s, \boldsymbol{X}^t)$$

对于分布适应，我们需要进行边缘适应、条件适应和动态适应。Courty 等人（2016）、Courty 等人（2014）提出了使用最优传输来学习特征变换 T，以减少边缘分布距离。然后，作者提出了联合分布最优传输（JDOT）（Courty 等人，2017）以加入条件分布适应。JDOT 的核心公式化为

$$\gamma_0 = \underset{\gamma \in \Pi(\psi_s, \psi_t)}{\arg\min} \int_{\Omega \times C}^{2} \mathcal{D}(\boldsymbol{X}_1, \boldsymbol{Y}_1; \boldsymbol{X}_2, \boldsymbol{Y}_2) d\gamma(\boldsymbol{X}_1, \boldsymbol{Y}_1; \boldsymbol{X}_2, \boldsymbol{Y}_2)$$

其成本函数被表示为边际分布差异和条件分布差异的加权和：

$$\mathcal{D} = \alpha d\left(\boldsymbol{X}_i^s, \boldsymbol{X}_j^t\right) + \mathcal{L}\left(\boldsymbol{Y}_i^s, f\left(\boldsymbol{X}_j^t\right)\right)$$

最优运输问题可以使用一些流行的工具来解决，例如 PythonOT。最优运输也可以应用于深度学习，例如 Xu 等人（2020b），Xu 等人（2020a），Bhushan Damodaran 等人（2018）和李等人（2019）。最近，Lu 等人（2021）提出了将基于最优传输的域自适应应用于跨域人类活动识别，并取得了很好的性能。他们的方法被称为子结构最优传输（SOT）。他们认为，域级和类级的最优传输过于粗略，可能导致欠适应，而样本级的匹配可能受到噪声的严重影响，最终导致过适应。SOT 通过用聚类方法获取的活动的子结构来利用活动的局部信息，并寻求不同领域之间加权子结构的耦合。因此，它可以被视为更细粒度的最优传输，并取得了比传统域级最优传输更好的效果。

16.5 迁移学习实践

16.5.1 Python 语言实践

迁移学习框架

为了进一步说明迁移学习算法的必要性，我们采用 KNN 分类器作为传统的迁移学习方法，构建 KNN 分类器对数据进行分类。代码实现如下：

```
# 导入相应模块
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np
```

首先，定义函数从 CSV 文件中加载数据。

```
def load_csv(folder, src_domain, tar_domain):
    # 加载源域数据
    data_s = np.loadtxt(f'{folder}/amazon_{src_domain}.csv', delimiter=',')
    # 加载目标域数据
    data_t = np.loadtxt(f'{folder}/amazon_{tar_domain}.csv', delimiter=',')
    Xs, Ys = data_s[:, :-1], data_s[:, -1] # 源域特征和标签
    Xt, Yt = data_t[:, :-1], data_t[:, -1] # 目标域特征和标签
    return Xs, Ys, Xt, Yt # 返回加载的数据
```

然后，构建 KNN 分类器，该分类器从源域和目标域获取特征值和标签对样本进行分类。

```
def knn_classify(Xs, Ys, Xt, Yt, k=1, norm=False):
    model = KNeighborsClassifier(n_neighbors=k) # 创建KNN分类器模型
    Ys = Ys.ravel() # 将源域标签展平
    Yt = Yt.ravel() # 将目标域标签展平
    if norm:
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        Xs = scaler.fit_transform(Xs) # 对源域特征进行标准化
        Xt = scaler.fit_transform(Xt) # 对目标域特征进行标准化
    model.fit(Xs, Ys) # 在源域上训练模型
    Yt_pred = model.predict(Xt) # 在目标域上进行预测
    acc = accuracy_score(Yt, Yt_pred) # 计算准确率
    print(f'Accuracy: {acc * 100:.2f}%')
```

最后，通过在主函数中调用函数来完成迁移学习的分类任务。

```
if __name__ == "__main__":
    folder = '../../office31_resnet50'
    src_domain = 'amazon'
    tar_domain = 'webcam'
    Xs, Ys, Xt, Yt = load_csv(folder, src_domain, tar_domain)
    print('Source:', src_domain, Xs.shape, Ys.shape)
    print('Target:', tar_domain, Xt.shape, Yt.shape)

    norm = True
    knn_classify(Xs, Ys, Xt, Yt, k=1, norm=norm)
```

上述代码演示了如何使用 Python 中的 Scikit-learn 库来实现基于 KNN 算法的域自适应分类器。通过加载源域和目标域的数据，然后利用 KNN 算法进行跨域分类，并计算分类准确率。这个示例向学习者展示了如何在实际问题中应用迁移学习算法，以及如何处理不同域之间的数据差异，从而实现跨域分类的技术。

实例加权方法

在本小节中，我们将实现迁移学习中实例加权方法的核均值匹配（KMM）算法。KMM 的核心是通过建立一个二次规划方程来学习源域和目标域的权重比，我们可

以使用 cvxopt 包来进行二次规划。然后，我们可以使用 KNN 分类器对目标域进行分类。KMM 算法的代码如下所示：

```
pip install cvxopt # 安装cvxopt包
# 导入相应模块
import numpy as np
import sklearn.metrics
import os
import argparse
from cvxopt import matrix, solvers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

定义并解析命令行参数。

```
parser = argparse.ArgumentParser() # 创建解析器对象
parser.add_argument('--norm', action='store_true') # 添加一个名为 --norm 的可选参数
args = parser.parse_args() # 解析命令行参数并将其存储在 args 中
```

定义核函数，类型有 *'linear'* 和 *'rbf'*。

```
def kernel(ker, X1, X2, gamma):
    K = None
    if ker == 'linear': # 使用线性核函数
        if X2 is not None:
            K = sklearn.metrics.pairwise.linear_kernel(
            np.asarray(X1), np.asarray(X2))
        else:
            K = sklearn.metrics.pairwise.linear_kernel(np.asarray(X1))
    elif ker == 'rbf': # 使用径向基函数
        if X2 is not None:
            K = sklearn.metrics.pairwise.rbf_kernel(
            np.asarray(X1), np.asarray(X2), gamma)
        else:
            K = sklearn.metrics.pairwise.rbf_kernel(np.asarray(X1), None, gamma)
    return K
```

KMM 算法如下：

```
class KMM:
    def __init__(self, kernel_type='linear', gamma=1.0, B=1.0, eps=None):
        self.kernel_type = kernel_type # 核函数的类型，有 'linear' | 'rbf'
        self.gamma = gamma # rbf核函数的核带宽
        self.B = B # beta的上界
        self.eps = eps # epsilon的上界
    def fit(self, Xs, Xt): # 用KMM来拟合源域和目标域（计算系数）
        ns = Xs.shape[0]
        nt = Xt.shape[0]
        if self.eps == None:
           self.eps = self.B / np.sqrt(ns)
        K = kernel(self.kernel_type, Xs, None, self.gamma)
        kappa = np.sum(kernel(self.kernel_type, Xs, Xt, self.gamma) *
                       float(ns) / float(nt), axis=1)
```

```
        # 使用cvxopt库求解最优化的beta
        K = matrix(K.astype(np.double))
        kappa = matrix(kappa.astype(np.double))
        G = matrix(np.r_[np.ones((1, ns)), -np.ones((1, ns)),
                   np.eye(ns), -np.eye(ns)])
        h = matrix(np.r_[ns * (1 + self.eps), ns * (self.eps - 1),
                         self.B * np.ones((ns,)), np.zeros((ns,))])
        sol = solvers.qp(K, -kappa, G, h)
        beta = np.array(sol['x'])
        return beta
```

使用 KNN 进行分类。

```
def knn_classify(Xs, Ys, Xt, Yt, k=1, norm=False):
    model = KNeighborsClassifier(n_neighbors=k)
    Ys = Ys.ravel()
    Yt = Yt.ravel()
    if norm:
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        Xs = scaler.fit_transform(Xs)
        Xt = scaler.fit_transform(Xt)
    model.fit(Xs, Ys)
    Yt_pred = model.predict(Xt)
    acc = accuracy_score(Yt, Yt_pred)
    print(f'Accuracy using kNN: {acc * 100:.2f}%')
```

然后，我们可以使用 KNN 分类器来获得目标域上的分类结果。注意，我们需要使用 KMM 权值来转换源特性，函数如下所示：

```
def load_data(folder, domain): # 定义load_data函数，从指定文件夹中加载数据
    from scipy import io
    data = io.loadmat(os.path.join(folder, domain + '_fc6.mat'))
    return data['fts'], data['labels']
if __name__ == "__main__":
    # 数据集： https://www.jianguoyun.com/p/DcNAUgQQmN7PCBiF9asD (密码：qqLA7D)
    folder = 'E:\下载\office31' # 数据集的存储位置
    src_domain = 'amazon'
    tar_domain = 'webcam'
    Xs, Ys = load_data(folder, src_domain)
    Xt, Yt = load_data(folder, tar_domain)
    print('Source:', src_domain, Xs.shape, Ys.shape)
    print('Target:', tar_domain, Xt.shape, Yt.shape)

    kmm = KMM(kernel_type='rbf', B=10)
    beta = kmm.fit(Xs, Xt)
    print(beta)
    print(beta.shape)
    Xs_new = beta * Xs
    knn_classify(Xs_new, Ys, Xt, Yt, k=1, norm=args.norm)
```

统计特征变换方法

在本小节中,我们实现经典的迁移学习方法:迁移成分分析(Transfer Component Analysis),这里实现的核心是使用 Python 包进行 TCA 的特征分解。

```
# 导入相应模块
import numpy as np
import scipy.io
import scipy.linalg
import sklearn.metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

首先定义核函数,类型有 *'primal'*, *'linear'* 和 *'rbf'*。

```
def kernel(ker, X1, X2, gamma):
    K = None
    if not ker or ker == 'primal':
        K = X1
    elif ker == 'linear':
        if X2 is not None:
            K = sklearn.metrics.pairwise.linear_kernel(np.asarray(X1).T, np.asarray(
                                                    X2).T)
        else:
            K = sklearn.metrics.pairwise.linear_kernel(np.asarray(X1).T)
    elif ker == 'rbf':
        if X2 is not None:
            K = sklearn.metrics.pairwise.rbf_kernel(
                np.asarray(X1).T, np.asarray(X2).T, gamma)
        else:
            K = sklearn.metrics.pairwise.rbf_kernel(
                np.asarray(X1).T, None, gamma)
    return K
```

使用 scipy.linag.eig 函数实现 TCA:

```
class TCA:
    def __init__(self, kernel_type='primal', dim=30, lamb=1, gamma=1):
        self.kernel_type = kernel_type
        self.dim = dim # 迁移后的维数
        self.lamb = lamb # 公式中的lambda值
        self.gamma = gamma # rbf核函数的核带宽
```

对源特征 Xs 和目标特征 Xt 进行变换,返回经过 TCA 方法后的 Xs_new 和 Xt_new。

```
    def fit(self, Xs, Xt):
        X = np.hstack((Xs.T, Xt.T))
        X /= np.linalg.norm(X, axis=0)
        m, n = X.shape
        ns, nt = len(Xs), len(Xt)
        e = np.vstack((1 / ns * np.ones((ns, 1)), -1 / nt * np.ones((nt, 1))))
        M = e * e.T
```

```
        M = M / np.linalg.norm(M, 'fro')
        H = np.eye(n) - 1 / n * np.ones((n, n))
        K = kernel(self.kernel_type, X, None, gamma=self.gamma)
        n_eye = m if self.kernel_type == 'primal' else n
        a, b = K @ M @ K.T + self.lamb * np.eye(n_eye), K @ H @ K.T
        w, V = scipy.linalg.eig(a, b)
        ind = np.argsort(w)
        A = V[:, ind[:self.dim]]
        Z = A.T @ K
        Z /= np.linalg.norm(Z, axis=0)
        Xs_new, Xt_new = Z[:, :ns].T, Z[:, ns:].T
        return Xs_new, Xt_new
```

对 Xs 和 Xt 变换, 然后使用 1NN 对目标进行预测。

```
    # 传入参数Xs表示源特征, Ys表示源标签; Xt表示目标特征, Yt表示目标标签
    def fit_predict(self, Xs, Ys, Xt, Yt):
        Xs_new, Xt_new = self.fit(Xs, Xt)
        clf = KNeighborsClassifier(n_neighbors=1)
        clf.fit(Xs_new, Ys.ravel())
        y_pred = clf.predict(Xt_new)
        acc = sklearn.metrics.accuracy_score(Yt, y_pred)
        return acc, y_pred # 返回预测准确度和目标域上的预测标签
```

在主函数中载入数据并使用 TCA 的方法:

```
if __name__ == '__main__':
    domains = ['caltech_decaf.mat', 'amazon_decaf.mat',
               'webcam_decaf.mat', 'dslr_decaf.mat']
    for i in [0]:
        for j in [1]:
            if i != j:
                src, tar = domains[i], domains[j]
                src_domain, tar_domain = scipy.io.loadmat(src),scipy.io.loadmat(tar)
                Xs, Ys, Xt, Yt = src_domain['feas'],src_domain['labels'],
                                 tar_domain['feas'],tar_domain['labels']
                tca = TCA(kernel_type='linear', dim=30, lamb=1, gamma=1)
                acc, ypred = tca.fit_predict(Xs, Ys, Xt, Yt)
                print(f'Accuracy: {acc:.3f}') # 0.897
```

几何特征变换方法

在本小节中, 我们实现了 CORrelation ALignment (CORAL) (Sun 等, 2016) 来执行几何特征变换。CORAL 的实现非常简单, 因为我们只需要求解协方差矩阵即可。我们编写一个名为 fit 的函数, 它接受源和目标特征 $\mathbf{X}_s$ 和 $\mathbf{X}_t$, 然后返回经过 CORAL 变换后的源域。代码如下:

```
#导入相应模块
import numpy as np
import scipy.io
import scipy.linalg
```

```
import sklearn.metrics
import sklearn.neighbors
```

```
#定义函数从CSV文件中加载数据
def load_csv(folder, src_domain, tar_domain):
    data_s = np.loadtxt('F:/pythonStu/数据/amazon_amazon.csv', delimiter=',')
    data_t = np.loadtxt('F:/pythonStu/数据/amazon_webcam.csv', delimiter=',')
    Xs, Ys = data_s[:, :-1], data_s[:, -1]
    Xt, Yt = data_t[:, :-1], data_t[:, -1]
    return Xs, Ys, Xt, Yt
#数据来源：https://www.jianguoyun.com/p/DSO1BC0QmN7PCBicraIE（密码：eS5fMT）
class CORAL:
    def __init__(self):
        super(CORAL, self).__init__()
    #对源域特征实行CORAL，返回新的源域特征
    def fit(self, Xs, Xt):
        cov_src = np.cov(Xs.T) + np.eye(Xs.shape[1])
        cov_tar = np.cov(Xt.T) + np.eye(Xt.shape[1])
        A_coral = np.dot(
            scipy.linalg.fractional_matrix_power(cov_src, -0.5),
            scipy.linalg.fractional_matrix_power(cov_tar, 0.5))
        Xs_new = np.real(np.dot(Xs, A_coral))
        return Xs_new
```

在 CORAL 之后，我们使用 scikit-learn 构建一个 KNN 分类器来计算目标域的准确性。这可以通过以下函数实现：

```
#实行CORAL，使用KNN分类器进行预测，并返回准确率
def fit_predict(self, Xs, Ys, Xt, Yt):
    Xs_new = self.fit(Xs, Xt)
    clf = sklearn.neighbors.KNeighborsClassifier(n_neighbors=1)
    clf.fit(Xs_new, Ys.ravel())
    y_pred = clf.predict(Xt)
    acc = sklearn.metrics.accuracy_score(Yt, y_pred)
    return acc, y_pred
if __name__ == "__main__":
    folder = '../../office31_resnet50'
    src_domain = 'amazon'
    tar_domain = 'webcam'
    Xs, Ys, Xt, Yt = load_csv(folder, src_domain, tar_domain)
    print('Source:', src_domain, Xs.shape, Ys.shape)
    print('Target:', tar_domain, Xt.shape, Yt.shape)
    coral = CORAL()
    acc, ypre = coral.fit_predict(Xs, Ys, Xt, Yt)
    print(f'Accuracy: {acc:.2f}')
```

16.5.2 R 语言实践

迁移学习框架

代码演示了如何使用 R 语言中的 class 包，采用 KNN 分类器作为传统的迁移学习方法，构建 KNN 分类器对数据进行分类。代码实现如下：

```
library(class) # 导入class包

# 定义函数load_csv，用于加载CSV数据
load_csv <- function(folder, src_domain, tar_domain) {
  # 从CSV文件中读取源域数据
  data_s <- read.csv(file.path(folder, paste0("amazon_", src_domain, ".csv")), header
                                              =TRUE, sep=",")
  # 从CSV文件中读取目标域数据
  data_t <- read.csv(file.path(folder, paste0("amazon_", tar_domain, ".csv")), header
                                              =TRUE, sep=",")
  Xs <- data_s[, -ncol(data_s)] # 提取源域特征
  Ys <- data_s[, ncol(data_s)] # 提取源域标签
  Xt <- data_t[, -ncol(data_t)] # 提取目标域特征
  Yt <- data_t[, ncol(data_t)]  # 提取目标域标签
  return(list(Xs = Xs, Ys = Ys, Xt = Xt, Yt = Yt)) # 返回一个提取的数据的列表
}
```

然后，使用 KNN 算法从源域和目标域获取特征值和标签对样本进行分类。

```
knn_classify <- function(Xs, Ys, Xt, Yt, k=1, norm=FALSE) {
  if (norm) {
    Xs <- as.data.frame(scale(Xs)) # 对源域特征进行标准化处理
    Xt <- as.data.frame(scale(Xt)) # 对目标域特征进行标准化处理
  }
  model <- knn(train = Xs, test = Xt, cl = Ys, k = k) # 使用KNN算法进行分类
  Yt_pred <- model # 获取目标域的预测结果
  acc <- sum(Yt_pred == Yt) / length(Yt) # 计算分类准确率
  cat("Accuracy:", acc * 100, "%\n") # 输出分类准确率
}
```

最后，调用函数来完成迁移学习的分类任务。

```
folder <- "../../office31_resnet50"
src_domain <- "amazon"
tar_domain <- "webcam"
# 加载数据并输出信息
data <- load_csv(folder, src_domain, tar_domain)
cat("Source:", src_domain, dim(data$Xs), length(data$Ys), "\n")
cat("Target:", tar_domain, dim(data$Xt), length(data$Yt), "\n")

# 调用knn_classify函数进行分类，并指定k=1，norm=TRUE
knn_classify(data$Xs, data$Ys, data$Xt, data$Yt, k=1, norm=TRUE)
```

实例加权方法

本小节中，我们使用 R 语言实现核均值匹配算法，具体代码如下：

```
# 导入相应的库
library(class)
library(e1071)
library(Matrix)
library(ROCR)
library(proxy)
library(quadprog)
```

定义核函数。

```
kernel <- function(ker, X1, X2, gamma) {
  if (ker == 'linear') {
    if (!is.null(X2)) {
      return(t(X1) %*% X2)
    } else {
      return(t(X1) %*% X1)
    }
  } else if (ker == 'rbf') {
    if (!is.null(X2)) {
      return(exp(-gamma * as.matrix(dist(X1, X2))^2))
    } else {
      return(exp(-gamma * as.matrix(dist(X1))^2))
    }
  }
}
```

KMM 算法如下：

```
KMM <- function(kernel_type = 'linear', gamma = 1.0, B = 1.0, eps = NULL) {
  list(
       kernel_type = kernel_type,
       gamma = gamma,
       B = B,
       eps = eps
  )
}
fit <- function(self, Xs, Xt) {
   ns <- nrow(Xs)
   nt <- nrow(Xt)
   if (is.null(self$eps)) {
       self$eps <- self$B / sqrt(ns)
   }
   K <- kernel(self$kernel_type, Xs, NULL, self$gamma)
   kappa <- colSums(kernel(self$kernel_type, Xs, Xt, self$gamma) * (ns / nt))
   Dmat <- as.matrix(K)
   dvec <- -kappa
   Amat <- rbind(1, -1, diag(ns), -diag(ns))
   bvec <- c(ns * (1 + self$eps), ns * (self$eps - 1), self$B * rep(1, ns), rep(0, ns
                                                                                 ))
```

```
    sol <- solve.QP(Dmat, dvec, Amat, bvec, meq = 1)
    beta <- sol$solution
return(beta)
}
```

使用 KNN 进行分类。

```
knn_classify <- function(Xs, Ys, Xt, Yt, k = 1, norm = FALSE) {
  if (norm) {
    Xs <- scale(Xs)
    Xt <- scale(Xt)
  }
  model <- ksvm(Xs, as.factor(Ys), type = "C-svc",
                kernel = "vanilladot", C = 10)
  Yt_pred <- predict(model, newdata = Xt)
  acc <- sum(Yt_pred == Yt) / length(Yt)
  cat(paste("Accuracy using kNN:", acc * 100, "%\n"))
}
```

定义用于加载 mat 文件的函数。

```
load_data <- function(folder, domain) {
  data <- readRDS(file.path(folder, paste0(domain, '_fc6.rds')))
  return(list(fts = data$fts, labels = data$labels))
}
```

加载数据，然后，我们可以使用 KNN 分类器来获得目标域上的分类结果。

```
folder <- 'E:/下载/office31' # 数据集的存储位置
src_domain <- 'amazon'
tar_domain <- 'webcam'

data_src <- load_data(folder, src_domain)
data_tar <- load_data(folder, tar_domain)

Xs <- data_src$fts
Ys <- data_src$labels
Xt <- data_tar$fts
Yt <- data_tar$labels

cat('Source:', src_domain, dim(Xs), dim(Ys), '\n')
cat('Target:', tar_domain, dim(Xt), dim(Yt), '\n')

kmm <- KMM(kernel_type = 'rbf', B = 10)
beta <- fit(kmm, Xs, Xt)
cat(beta, '\n')

Xs_new <- beta %*% t(Xs)
knn_classify(Xs_new, Ys, Xt, Yt, k = 1, norm = FALSE)
```

统计特征变换方法

本小节中，我们使用 R 语言实现迁移成分分析方法，首先导入必要的库和数据：

```
library(readr)
library(proxy)
library(caret)
library(class)
library(e1071)
# 定义load_csv函数
load_csv <- function(folder, src_domain, tar_domain) {
  data_s <- read.csv(paste0(folder, "/amazon_", src_domain, ".csv"), header = TRUE)
  data_t <- read.csv(paste0(folder, "/amazon_", tar_domain, ".csv"), header = TRUE)

  # 将数据划分为特征和标签
  Xs <- data_s[, -1] # 源特征
  Ys <- data_s[, 1]   # 源标签

  Xt <- data_t[, -1] # 目标特征
  Yt <- data_t[, 1]   # 目标标签

  return(list(Xs = Xs, Ys = Ys, Xt = Xt, Yt = Yt))
}
```

然后定义核函数，类型有 *'primal'*，*'linear'* 和 *'rbf'*。

```
kernel <- function(ker, X1, X2, gamma) {
  K <- NULL
  if (missing(ker) || ker == "primal") {
    K <- as.matrix(X1)
  } else if (ker == "linear") {
    if (!missing(X2)) {
      K <- apply(as.matrix(X1), 2, function(x) as.vector(dist(x, as.matrix(X2))))
    } else {
      K <- as.matrix(X1) # 对于线性核，只需返回X1作为矩阵
    }
  } else if (ker == "rbf") {
    if (!missing(X2)) {
      K <- apply(as.matrix(X1), 2, function(x) as.vector(dist2(x, as.matrix(X2),
                                                method = "euclidean"))) * gamma
    } else {
      K <- apply(as.matrix(X1), 2, function(x) as.vector(dist2(x, NULL, method = "
                                                euclidean"))) * gamma # 对于RBF核，
                                                只需返回X1作为矩阵并乘以gamma
    }
  }
  return(K)
}
```

实现 TCA 类并初始化：

```
TCA <- function(kernel_type = "primal", dim = 30, lamb = 1, gamma = 1) {
  list(
```

```
    kernel_type = kernel_type,
    dim = dim,
    lamb = lamb,
    gamma = gamma
  )
}
```

定义 fit 函数：

```
fit <- function(Xs, Xt) {
  X <- rbind(t(Xs), t(Xt))  # 将Xs和Xt进行堆叠
  X <- apply(X, 2, function(x) x / sum(abs(x)))  # 对X进行归一化处理
  m <- nrow(X)
  n <- ncol(X)
  e <- cbind(1 / m * rep(1, m), -1 / n * rep(1, n))
  M <- e %*% e.t()
  M <- M / norm(M, "fro")  # 对M矩阵进行归一化处理
  H <- diag(n) - 1 / n * rep(1, n)
  K <- kernel(this$kernel_type, X, NULL, gamma=this$gamma)  # 计算核矩阵K
  n_eye <- if (kernel_type == "primal") m else n
  a <- K %*% M %*% K.t() + this$lamb * diag(n_eye)
  b <- K %*% H %*% K.t()
  w <- eig(a, b)$values[1:this$dim] # 取前dim个特征值
  V <- eig(a, b)$vectors[, 1:this$dim] # 取前dim个特征向量
  ind <- order(w)  # 对特征值进行排序，并获取排序后的索引
  A <- V[, ind[1:dim]] # 取排序后的特征向量
  Z <- A.t() %*% K # 对原始数据进行转换
  Z <- Z / sum(abs(Z), axis=2) * sqrt(2) # 对每一列进行归一化处理，使每一列的长度为根
                                          号2乘以相应维度的最大值
  return(list(Xs_new = Z[, 1:m], Xt_new = Z[, m+1:n]))
}
```

定义预测方法如下：

```
fit_predict <- function(object, Xs, Ys, Xt, Yt) {
  Xs_new <- object$fit(Xs, Xt)$Xs_new  # 变换 Xs 和 Xt
  Xt_new <- object$fit(Xs, Xt)$Xt_new
  clf <- knn(train = Xs_new, test = Ys, k = 1)  # 在源域上训练1NN分类器
  y_pred <- clf$class$predict(Xt_new)  # 在目标域上做预测
  acc <- accuracy(Yt, y_pred)  # 计算准确度
  return(list(acc = acc, y_pred))
}
```

下载好数据集并载入数据：

```
folder <- '/../../resnet50_feature'
src_domain <- 'amazon'
tar_domain <- 'webcam'
data <- load_csv(folder, src_domain, tar_domain)
Xs <- data$Xs
Ys <- data$Ys
Xt <- data$Xt
Yt <- data$Yt
```

创建 TCA 实例并输出准确度：

```
tca <- TCA(kernel_type = "primal", dim = 40, lamb = 0.1, gamma = 1)
result <- tca\$fit_predict(Xs, Ys, Xt, Yt)
acc <- result$acc
y_pred <- result$y_pred
print(paste("Accuracy:", acc))
```

16.5.3 几何特征变换方法

在本小节中，我们使用 R 语言实现迁移学习的 CORAL 方法。代码如下：

```
library(readr)
library(proxy)
library(caret)
library(class)
library(e1071)
#定义函数从CSV文件中加载数据
load_csv <- function(folder, src_domain, tar_domain) {
  file_path_s <- paste0('F:/pythonStu/数据/', src_domain, '_', src_domain, '.csv')
  file_path_t <- paste0('F:/pythonStu/数据/', src_domain, '_', tar_domain, '.csv')
  data_s <- read.csv(file_path_s, header = FALSE, sep = ',')
  data_t <- read.csv(file_path_t, header = FALSE, sep = ',')
  Xs <- data_s[, -ncol(data_s)]
  Ys <- data_s[, ncol(data_s)]
  Xt <- data_t[, -ncol(data_t)]
  Yt <- data_t[, ncol(data_t)]
  return(list(Xs = Xs, Ys = Ys, Xt = Xt, Yt = Yt))
}
#数据来源：https://www.jianguoyun.com/p/DSO1BC0QmN7PCBicraIE（密码：eS5fMT）
```

```
#对源域特征实行CORAL，返回新的源域特征
fit <- function(Xs, Xt) {
  cov_src <- cov(Xs) + diag(rep(1, ncol(Xs)))
  cov_tar <- cov(Xt) + diag(rep(1, ncol(Xt)))
  A_coral <- solve(sqrtm(cov_src)) %*% sqrtm(cov_tar)
  Xs_new <- Re(Xs %*% A_coral)
  return(Xs_new)
}
```

在 CORAL 之后，我们使用 scikit-learn 构建一个 KNN 分类器来计算目标域的准确性。这可以通过以下函数实现：

```
#实行CORAL，使用KNN分类器进行预测，并返回准确率
fit_predict <- function(Xs, Ys, Xt, Yt) {
  Xs_new <- fit(Xs, Xt)
  clf <- knn(train = Xs_new, test = Xt, cl = Ys, k = 1)
  acc <- sum(clf == Yt) / length(Yt)
  return(list(accuracy = acc, y_pred = clf))
```

```
}
folder <- '../../office31_resnet50'
src_domain <- 'amazon'
tar_domain <- 'webcam'
data <- load_csv(folder, src_domain, tar_domain)
Xs <- data$Xs
Ys <- data$Ys
Xt <- data$Xt
Yt <- data$Yt

result <- fit_predict(Xs, Ys, Xt, Yt)
print(paste('Source:', src_domain, dim(Xs), dim(Ys)))
print(paste('Target:', tar_domain, dim(Xt), dim(Yt)))
print(paste('Accuracy:', result$accuracy))
```

16.6 习题

1、思考给定一个目标领域，如何找到相对应的源领域，然后进行迁移？什么时候不可以进行迁移？

2、试给出由 (16.3.1) 式到 (16.3.2) 式的推导过程。

3、写出当 μ 取为不同的值时，相应的基于 MMD 的边际、条件和联合分布的迁移学习方法的最终形式。

4、文中提到的三种几何特征变换方法分别有什么联系与区别，是否还有其他类型的几何特征变换方法？

5、试给出目标域内的预测函数应用于逻辑回归和 SVM 时的特定表述公式。

第七部分

生成学习

第十七章　生成式机器学习

生成学习（Generative Learning）是一类专注于生成新样本或模拟数据分布的方法，主要任务是学习数据的分布，然后利用这个模型生成与原始数据相似的新样本。与前面提到的机器学习算法相比，其他方法更注重对数据的分类、回归、决策等任务；生成学习通常使用生成模型，而其他方法则使用判别模型。生成学习具有数据增强、样本生成、概率建模图像合成和风格迁移以及对抗训练等优势，在图像生成、自然语言处理、医学图像处理和风格迁移等领域具有广泛应用。

生成学习可以与其他机器学习方法相结合，采用深度学习中的深度神经网络技术，生成的数据可以应用到集成学习中提高集成模型的多样性，应用到增量学习中提高模型的泛化性能。

在本章中，我们将深入探讨生成式机器学习的核心概念和方法，以及它在机器学习和数据科学中的广泛应用。通过本章的学习，您将掌握生成式学习的基本原理，了解如何构建生成模型，以及如何应用这些模型来解决实际问题。

17.1　简介

生成式机器学习（Generative Machine Learning）是机器学习领域中一颗璀璨的明星，其历史可追溯到 20 世纪中叶。其核心理念是通过建立概率模型来描述数据的生成过程，并通过从这个模型中采样，创造全新的数据。想象一下，我们拥有一个包含马的图像数据集或者是一个各项药物的化学成分表，现在我们的目标是创建一个能够生成一张全新的、以前从未存在过的马的图像以及一种还未被研发出来的药物的各种成分，但看起来仍然非常真实，因为模型已经学会了样本的一般特征。这正是生成式学习所能解决的问题之一。

在本章中，我们将深入探讨生成式机器学习的核心概念和方法，以及它在机器学习和数据科学中的广泛应用。通过本章的学习，您将掌握生成式学习的基本原理，了解如何构建生成模型，以及如何应用这些模型来解决实际问题。

本章内容将包括多个主题，包括变分自动编码器（Variational Auto Encoder）、生成对抗网络（Generative Neural Network）等常用生成模型，以及如何防止模型过拟合和优化生成性能。我们还将提供实际的示例代码，帮助您在 Python 中实际应用生成式学习方法。

生成式机器学习是一个充满活力和创新的领域，不断推动着数据科学和机器学

习的发展。让我们一起踏上这段令人激动的学习之旅，探索生成式学习的无限可能性！

17.2 生成机器学习基础

17.2.1 生成模型与判别模型

为了更加深入理解生成模型的重要性，我们来试着比较一下生成模型和判别模型（discriminative model）。前面我们所接触的决策树、支持向量机等模型都属于判别模型，为了更直观地理解他们之间的差异，让我们看一个例子。

假设我们有一组文本数据集，其中一些是摘自作者 A 的作品，而另一些是由其他作者完成。通过足够的数据，我们可以训练一个判别模型来预测一段文字是否为作者 A 所写。而生成模型将学会，某些文字、语句和段落更有可能表明一段文字是由作者 A 完成的，此时该模型对就会相应地提高具有这一系列特征的预测值。

生成模型总是在试图学习数据样本的分布，并基于学习到的分布信息生成新的数据点，使得这些数据点在统计上与原始数据样本相似。换句话说，生成模型的关键问题是“给定一个模型，如何生成与训练数据类似的新数据?”，而判别模型所关心的问题通常是“给定一些数据，如何将其分为不同的类别或进行预测?”

因此在训练判别模型时，训练数据中的每个观测样本都有一个对应的标签，也就是说，建立判别模型属于监督学习。在上述判别文字作者的二分类问题中，作者 A 所写文本被标记为 1，其他则为 0。该判别模型训练完成后，就能够输出新的观测样本被标记为 1 的概率。

而建立生成模型则通常基于没有标签的数据集，当然也可以基于有标签的数据集，那么模型就能够学习到如何从相应的类中生成观测值。

换句话说，判别模型旨在准确估计观测值 $\boldsymbol{X}$ 属于类别 $\boldsymbol{Y}$ 的概率 $P(\boldsymbol{Y}|\boldsymbol{X})$。而生成建模则并不特别需要观测值的标签，相反，它试图估计观测值 $\boldsymbol{X}$ 本身出现的概率 $P(\boldsymbol{X})$，当然若数据集带有标签，则可以估计已知标签下的条件概率 $P(\boldsymbol{X}|\boldsymbol{Y})$。

正因如此，即使我们能够训练出一个完美的判别模型来准确识别一段文字是否出自作者 A 之手，但该模型仍不知道究竟如何写出这样一段作者 A 风格的文字，它只能输出相对于已有文字的概率，因为这是它被训练来做的事情。相反，如果我们训练出一个成功的生成模型，它就能够输出一段文字，这些文字极有可能属于原始的文本数据集。

17.2.2 概率分布的基本概念

也就是说，生成模型的核心思想是对数据的概率分布进行建模，那么让我们首先回顾一下概率分布中的一些基本概念。

样本空间是指数据的所有可能取值的集合。例如，如果我们在图像生成任务中使用生成模型，样本空间包括了所有可能的图像，无论是真实的还是虚构的。

概率分布是一个数学模型，用来描述随机变量的可能取值及其对应的概率。它告诉我们不同取值的相对可能性，从而帮助我们理解和模拟随机现象。

概率密度函数（PDF）$f(x)$ 是一个函数，用来描述样本空间连续时的概率分布，它能够将样本空间中的样本点 x 映射到区间 $[0,1]$ 中的一个数字。

概率质量函数（PMF）则用于描述样本空间离散时的概率分布，数学上表示为 $P(X=x)$，其中 $P(X=x)$ 表示随机变量 X 取值为 x 的概率。

需要注意的是，样本空间中所有样本点的概率质量函数之和为 1（对于离散型分布）或概率密度函数在整个样本空间中的积分等于 1（对于连续型分布）。

数字特征描述了随机变量的中心位置和离散度等其他信息，作为对概率密度函数和概率质量函数的补充。常用的数字特征主要有期望$\mathrm{E}(X)$ 和方差$\mathrm{Var}(X)$。

对于连续性随机变量，数学表达为：

$$\begin{aligned}\mathrm{E}(X) &= \int x \cdot f(x)dx \\ \mathrm{Var}(X) &= \int (x-\mathrm{E}(X))^2 \cdot f(x)dx\end{aligned} \tag{17.2.1}$$

对于离散型随机变量，则相应表示为：

$$\begin{aligned}\mathrm{E}(X) &= \sum x \cdot P(X=x) \\ \mathrm{Var}(X) &= \sum (x-\mu)^2 \cdot P(X=x)\end{aligned} \tag{17.2.2}$$

17.2.3 生成模型的基本思路

掌握了概率分布中的这些基本概念后，让我们再从宏观上了解一下构建一个生成模型的基本思路：

（1）假设我们有一个观测数据集 $\mathbf{X}$；

（2）假设这些观测观测样本来自某个未知概率分布 P_{data}；

（3）生成模型试图通过 $\mathbf{X}$ 学习该未知分布 P_{data}，得到一个 P_{model}，若成功实现了这一目标，那么即可直接从 P_{model} 中采样，生成新的观测数据集 $\mathbf{X}_{\mathbf{new}}$，并且这些生成的观测样本看起来就来自于 P_{data}；

（4）通常对生成模型的要求主要有两点：首先，该模型生成的观测数据集 $\mathbf{X}_{\mathbf{new}}$ 必须看起来就来自于 P_{data}，这是定义使然。其次，$\mathbf{X}_{\mathbf{new}}$ 要与原始观测数据集 $\mathbf{X}$ 存在明显不同，也就是说，生成模型不可以只是简单复制原始数据集中已有的样本点。

需要注意的是，真正生成观测数据集 $\mathbf{X}$ 的密度函数 P_{data} 只有一个，但用来估计 P_{data} 的密度函数 P_{model} 却可以有无数个。生成模型通常是参数化的，这意味着模型包含一组参数 θ，用来调整模型的行为以最好地拟合数据，可以表示为 $p_{\text{model}}(\boldsymbol{X};\theta)$。因此直观地来说，建立这样的参数化模型，我们的重点就在于使用一种优化方法，让我们从这无数个各种形式的 $p_{\text{model}}(\boldsymbol{X};\theta)$ 中找到最适合数据的参数组合，使生成模型的概率分布最好地拟合数据。这种方法被称为极大似然估计（Maximum Likelihood Estimation，MLE）。

下面我们再引出似然函数的定义。似然函数是一种关于统计模型中参数的函数，表示在给定模型参数 θ 的情况下，观察到一组数据集（或样本）$\mathcal{D}$ 的概率。它描述了参数 θ 如何解释或生成观察到的数据，其定义可以表示为：

$$L(\theta|\mathcal{D}) = P(\mathcal{D}|\theta) \tag{17.2.3}$$

其中，$P(\mathcal{D}|\theta)$ 表示在参数 θ 下观察到数据 $\mathcal{D}$ 的条件概率。

通常将似然函数取为样本的联合概率函数，即假设总体的概率函数为 $p\left(\boldsymbol{X};\theta\right)$ 或 $f\left(\boldsymbol{X};\theta\right)$，似然函数 $L(\theta)$ 表示为：

$$\begin{aligned} L\left(\theta\right) &= L\left(\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n; \theta\right) = \prod_{i=1}^{n} p\left(\boldsymbol{X}_i; \theta\right) \\ L\left(\theta\right) &= L\left(\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n; \theta\right) = \prod_{i=1}^{n} f\left(\boldsymbol{X}_i; \theta\right) \end{aligned} \tag{17.2.4}$$

而极大似然估计的目标便是是找到使似然函数 $L(\theta|\mathcal{D})$ 最大化的参数值 $\hat{\theta}$，即：

$$\hat{\theta} = \arg\max_{\theta} L(\theta|\mathcal{D})$$

换句话说，极大似然估计就是利用已知的样本结果信息，反推出最具有可能导致这些观测样本结果出现的一组模型参数值 $\hat{\theta}$。由于似然函数 $L(\theta|\mathcal{D})$ 常表示为样本密度函数的连乘形式，给求解带来了麻烦；并且 $\ln \boldsymbol{X}$ 是 $\boldsymbol{X}$ 的单调增函数，因此使得对数似然函数 $lnL(\theta)$ 达到最大与使 $L(\theta)$ 达到最大是等价的。所以通常通过从对数似然函数 $\ln L(\theta)$ 出发，通过求导或其他数值方法，找出 θ 的极大似然估计 $\hat{\theta}$。

当我们通过极大似然估计获得了最优参数值 $\hat{\theta}$，也就是得到了训练好的生成模型，我们就可以使用它来从学习好的概率分布中采样新的数据点。采样过程通常涉及到从概率分布中生成数据的方法。对于连续数据，这可以通过概率密度函数的反函数 $f^{-1}(\cdot)$ 来实现；对于离散数据，则通过根据概率质量函数的值来进行抽样。

17.3 浅层生成模型

浅层生成模型和深层生成模型是两种不同类型的生成模型，它们的主要区别在模型的层次结构和复杂度上。相较于深层生成模型，浅层生成模型通常由一个或几

个单层模型组成，直接从输入数据生成输出数据，没有中间的隐含层或抽象表示；其具有相对有限的参数数量，对数据的理解和学习主要依赖于特征工程，因此对于复杂的数据分布建模能力有限，故它们一般适用于一些相对简单的数据分布；对于模型参数的估计，浅层生成模型通常使用传统的最大似然估计或期望最大化算法，且训练相对简单，需要的数据量较少。

典型的浅层生成模型包括朴素贝叶斯模型（Naive Bayesian Model）、高斯混合模型（Gaussian Mixture Model，GMM）、隐马尔可夫模型（Hidden Markov Model，HMM）、潜在狄利克雷分布（Latent Dirichlet Allocation）、因子模型（Factorial Model）等。

17.3.1 朴素贝叶斯模型

朴素贝叶斯模型（Naive Bayesian Model）是一种基于贝叶斯定理与特征条件独立性假设构建的分类模型，其简单高效的特点使其在自然语言处理、垃圾邮件过滤、情感分析等领域得到了广泛的应用。

在理解朴素贝叶斯模型之前，我们首先需要了解贝叶斯定理。贝叶斯定理是一种用于在给定经验信息的情况下，计算后验概率的方法，其数学表达式如下：

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{17.3.1}$$

其中，$P(A|B)$ 是在给定条件 B 下的 A 的概率，称为 A 的“后验概率”；$P(B|A)$ 是在给定条件 A 下的 B 的概率，称为“似然度”；$P(A)$ 是没有任何条件下的 A 的概率，称为 A 的“先验概率”；$P(B)$ 是 B 的先验概率或边缘概率，称为“标准化常量”。

朴素贝叶斯模型的核心思想是通过已知的数据来估计不同类别的条件概率，然后使用贝叶斯定理来进行分类。在文本分类问题中，朴素贝叶斯模型假设各特征词之间相互独立，这就是“朴素”（Naive）的来源；同时，模型还假设已知各类别发生的无条件概率。其中，独立性假设使得模型计算变得简单，但同时也可能会导致一些信息损失，因为在实际情况下，文本中的词汇通常不是独立的。

具体来说，朴素贝叶斯模型的分类原理如下：

（1）根据训练数据集计算每个类别 C_i 的先验概率 $P(C_i)$；

（2）对于给定的特征 $\boldsymbol{X}_j$，计算每个特征在类别下的条件概率 $P(\boldsymbol{X}_j|C_i)$；

（3）根据贝叶斯定理计算每个类别下的后验概率 $P(C_i|\boldsymbol{X}_j)$；

（4）选择上述后验概率最大的类别，作为特征分类结果。

基于上述原理，朴素贝叶斯模型实现通常分为以下几步：首先，收集和整理用于训练和测试的数据集，包括对文本数据进行分词、消除失效词以及构建词袋模型或词向量等操作；其次，使用训练数据集训练朴素贝叶斯模型，计算类别的先验概

率和特征的条件概率；然后使用数据集评估模型的性能，通常使用准确率、精确测试度、响应率、$F1$ 分数等指标进行评估；并且还可以根据性能评估结果来调整模型的超参数，如平滑参数（smoothing parameter）等，以优化模型性能；最后，在模型训练和评估完成后，可以将模型部署到实际应用中，用于实时分类或预测任务。

可以看出，朴素贝叶斯模型具有以下一些优点：

（1）简单高效：该模型的计算和预测过程非常简单，适合处理大规模数据集。这使其成为许多实际问题的快速解决方案；（2）高度可扩展：朴素贝叶斯模型可以很容易地划分多类别分类问题，并且可以与其他机器学习方法结合使用，提高分类性能；（3）处理高维数据表现良好：因为它假设特征之间是独立的，可以减少灾难问题的影响；（4）可解释性强：朴素贝叶斯模型的分类过程非常透明，易于解释和理解，这在法律和医学等领域具有重要意义。

但同时，尽管朴素贝叶斯模型在许多应用中表现出色，它也有一些明显的缺点：

（1）过于朴素的特征独立性假设：朴素贝叶斯模型假设特征之间是相互独立的，这在实际数据中通常不成立，所以这个假设可能导致模型的性能受到影响；（2）处理连续特征困难：对于连续特征，朴素贝叶斯模型通常需要进行特征工程，将其离散化，以适应模型的离散概率分布；（3）需要大量数据：朴素贝叶斯模型对数据量的需求极大，如果数据量不足，模型可能会出现过拟合问题；（4）对噪声敏感：朴素贝叶斯模型对输入数据中的噪声和错误信息敏感，这可能会影响分类性能。

17.3.2 混合模型

混合模型（Mixture Model）是一类统计模型。通过结合多个概率分布函数，该模型能更为灵活地描述数据的分布情况。在混合模型中，我们假设数据是从多个不同的概率分布中抽取得到的，每个概率分配对应一个“成分”或“组件”，其中，“成分”或“组件”可以是一个简单的概率分布，如高斯分布、泊松分布、指数分布等，也可以是更复杂的分布。而混合模型的核心任务就是通过学习每个组件的参数以及组件各自的权重来对数据进行建模，这通常会使用各种统计方法，如：极大似然估计、期望最大化算法（EM）等。

混合模型的数学表达式通常如下：

$$P(\boldsymbol{X}) = \sum_{i=1}^{k} \pi_i \cdot P(\boldsymbol{X}|C_i) \tag{17.3.2}$$

其中，$P(\boldsymbol{X})$ 是初始数据的概率分布，k 是组件的数量，π_i 是第 i 个组件的权重，其满足 $\sum_{i=1}^{k} \pi_i = 1$，$P(\boldsymbol{X}|C_i)$ 是第 i 个组件下初始数据的条件概率分布。

混合模型在许多领域中都有广泛的应用，包括模式识别、精密估计、异常检测等。它可以处理多种类型的数据，包括连续型混合型、离散型和混合型数据。此外，其也被广泛用于机器学习中的无监督学习问题。

在本节，我们仅讨论其中最具有代表性的高斯混合模型(Gaussian Mixture Model，GMM)。高斯混合模型是单一高斯概率密度函数的延伸，GMM 能够平滑地近似任意形状的密度分布，并且后续在学习生成模型 VAE 的过程中的很多前置知识都是来源于 GMM 和 EM 算法，因此需要对其做一些了解。

高斯混合模型将数据建模为多个高斯分布的组合，其数学表示如下：

$$P(\boldsymbol{X}) = \sum_{i=1}^{k} \pi_i \cdot N(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}) \tag{17.3.3}$$

其中，k 是高斯分布组件的数量，π_i 是第 i 个高斯分布组件的权重，其满足 $\sum_{i=1}^{k} \pi_i = 1$，$N(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$ 是第 i 个高斯分布，其中 $\boldsymbol{\mu_i}$ 是均值向量，$\boldsymbol{\Sigma_i}$ 是协方差矩阵。

EM 算法是一种迭代优化算法，高斯混合模型的参数估计通常就借助该算法来完成，其步骤如下：

(1) 初始化：随机初始化 GMM 的参数，包括各组件的权重 π_i、均值向量 $\boldsymbol{\mu_i}$ 和协方差矩阵 $\boldsymbol{\Sigma_i}$；

(2) 期望步骤：计算每个数据点所属各个组件的后验概率，这可以使用贝叶斯公式来完成。对于 GMM 来说，即为计算每个数据点对应各个高斯分布组件的概率密度；

(3) 最大化步骤：利用期望步骤中计算的后验概率来更新模型的参数。具体来说，通过似然函数最大化，来更新模型的权重 π_i、均值向量 $\boldsymbol{\mu_i}$ 和协方差矩阵 $\boldsymbol{\Sigma_i}$；

(4) 迭代：重复执行上述期望步骤与最大化步骤，直到模型的参数收敛或达到指定的迭代次数，详细的参数迭代更新策略可见本书 1.5 节-高斯混合聚类。

EM 算法通常能够在 GMM 中找到局部最优解，但不能保证找到全局最优解。因此，初始参数的选择可能会影响算法的结果。

由其模型构建原理可知，GMM 具有多方面用途：(1) GMM 可以用于聚类分析，其中每个组别代表一个聚类簇，通过将数据点分配到最有可能的组别，实现数据聚类；(2) GMM 可以用于对数据分布的密度估计，在异常检测和模式识别领域中具有重要应用；(3) GMM 可以用作生成模型，用于生成与训练数据类似的新数据点，这在图像生成、文本生成等领域得到广泛使用；(4) GMM 可以用于降维，将高维数据映射到低维空间，实现降维和数据可视化。

17.3.3 时间序列模型中的浅层生成模型

时间序列模型（Time Series Model）并不是严格意义上的生成模型，而是一种用于描述和预测时间序列数据的模型。在时间序列模型中，较为典型的浅层生成模型包括隐马尔可夫模型（Hidden Markov Model，HMM）、卡尔曼滤波（Kalman Filter）、粒子滤波（Particle Filter）等，本节主要介绍隐马尔可夫模型（HMM）。

隐马尔科夫模型（Hidden Markov Model，HMM）是一种用于建模时间序列数据的统计模型，特别适用于具有潜在隐含结构的数据，其核心思想是将时间序列分解为一个不可见的马尔科夫链（即隐含状态序列）和一个可见的随机过程（即观测数据序列）。

HMM 的数学模型主要包括以下内容：

（1）状态集合（State Space）：HMM 包含一组隐含状态，通常用 $\boldsymbol{Q}$ 表示，$\boldsymbol{Q}=\{q_1,q_2,\cdots,q_N\}$，其中 N 是可能的状态数；

（2）观测集合（Observation Space）：HMM 包含一组观测状态，通常用 $\boldsymbol{V}$ 表示，$\boldsymbol{V}=\{v_1,v_2,\cdots,v_M\}$，其中 M 是可能的观测数，观测状态是可见的，通常是实际测量到的数据或观测值；

（3）隐含状态序列（Hidden State Sequence）：代表在时间序列中的不可见状态，表示系统在每个时间步骤下的状态。每个隐含状态对应一个观测状态，表示了在某个时间点上的系统状态，通常用 $\boldsymbol{I}=\{i_1,i_2,\cdots,i_T\}$ 表示，其中 T 表示时间步骤的数量；

（4）观测状态序列（Observation State Sequence）：代表在时间序列中的可见状态，表示在每个时间步骤下观测到的观测值。观测状态受到与它相关的隐含状态的影响。通常用 $\boldsymbol{O}=\{o_1,o_2,\cdots,o_T\}$ 表示，其中 T 表示时间步骤的数量；

（5）状态转移概率矩阵（State Transition Probability Matrix）：描述一个时间步骤到另一个时间步骤时，隐含状态如何从一个状态转移到另一个状态的概率分布矩阵，通常用 $\mathbf{A}=[a_{ij}]_{N\times N}$ 表示，其中 $a_{ij}=P(i_{t+1}=q_j|i_t=q_i)$ $(i,j=1,2,\cdots,N)$ 表示系统在时刻 t 从状态 q_i 转移到在时刻 $t+1$ 时状态 q_j 的概率，该矩阵的每一行之和应该等于 1，以确保状态转移概率的归一化；

（6）初始状态概率分布（Initial State Probability Distribution）：描述在时间序列开始时，系统处于每个隐含状态的概率分布，通常用 $\pi=(\pi_i)$ 表示，其中 $\pi_i=P(i_1=q_i)(i=1,2,\cdots,N)$ 表示系统在时刻 $t=1$ 时处于状态 q_i 的概率；

（7）观测概率分布（Observation Probability Distribution）：描述给定隐含状态时，观测状态的概率分布，通常用 $\mathbf{B}=[b_j(k)]_{N\times M}$ 表示，其中 $b_j(k)=P(o_t=v_k|i_t=q_j)$ $(k=1,2,\cdots,M;j=1,2,\cdots,N)$ 表示系统在时刻 t 处于状态 q_j 的条件下观测为 v_k 的概率。

隐马尔科夫模型（HMM）由初始状态概率向量 π、状态转移矩阵 $\mathbf{A}$ 和观测概率矩阵 $\mathbf{B}$ 所决定。π 和 $\mathbf{A}$ 确定隐藏的马尔科夫链，生成不可观测的隐含状态序列；$\mathbf{B}$ 则确定产生可见的观测状态序列，因此，隐马尔科夫模型 λ 可以用三元符号表示，

即 $\lambda=(\pi,\mathbf{A},\mathbf{B})$，其基本假设如下：

（1）离散时间步骤假设：HMM 假设时间被离散化为一系列时间步骤，通常表示为 $\{1,2,3,\cdots,T\}$，其中 T 是序列的长度，每个时间步骤对应一个状态和一个观测值，模型在每个时间步骤上执行状态转移和生成观测的操作；

（2）马尔科夫性假设：即系统的未来状态 i_{t+1} 只依赖于当前状态 i_t，而不依赖于过去的状态及观测序列。该假设通常表示为：

$$P(i_{t+1}|i_1,i_2,\cdots,i_t\ o_1,o_2,\cdots,o_t)=P(i_{t+1}|i_t),t=1,2,\cdots,T \tag{17.3.4}$$

其中，i_t 表示时刻 t 的隐含状态,o_t 表示时刻 t 的观测状态；

（3）观测独立性假设：在给定当前时刻隐含状态的情况下，观测状态之间是相互独立的。换句话说，每个观测状态只依赖于当前时间步骤的隐含状态，与其他观测状态无关。这个假设通常表示为：

$$P(o_t|i_1,i_2,\cdots,i_t;o_1,o_2,\cdots,o_{t-1})=P(o_t|i_t) \tag{17.3.5}$$

其中，i_t 表示时刻 t 的隐含状态，o_t 表示时刻 t 的观测状态。

基于上述假设，可写出隐含状态序列与观测状态序列的联合概率函数：

$$\begin{aligned}&P(i_1,i_2,\cdots,i_T;o_1,o_2,\cdots,o_T) \qquad (17.3.6)\\ &=P(i_1)P(o_1|i_1)P(i_2|i_1)P(o_2|i_2)P(i_3|i_2)P(o_3|i_3)\cdots\cdots P(i_T|i_{T-1})P(o_T|i_T)\\ &=P(i_1)P(o_1|i_1)\prod_{t=2}^{T}(P(i_t|i_{t-1})P(o_t|i_t))\end{aligned}$$

由上述定义与分布结构可以看到，该联合概率函数只依赖于初始状态概率分布 π、状态转移概率矩阵 $\mathbf{A}$、观测概率分布 $\mathbf{B}$。

HMM 模型相关的应用问题一般可以归结为以下这三个基本问题中的一个：

（1）评估问题（Evaluation）：给定模型 $\lambda=(\pi,\mathbf{A},\mathbf{B})$ 与观测序列 $\boldsymbol{O}$，计算在模型 λ 下该观测序列 $\boldsymbol{O}$ 出现的概率 $P(\boldsymbol{O}|\lambda)$，即“似然度”，通常使用前向算法来解决；

（2）解码问题（Decoding）：在已知模型 λ 和观测序列 $\boldsymbol{O}$ 的条件下，找到给定观测序列条件下最有可能的隐含状态序列 $\boldsymbol{I}$，即找到使得 $P(\boldsymbol{I}|\boldsymbol{O})$ 概率最大的 $\boldsymbol{I}$，通常使用 Viterbi 算法来解决；

（3）学习问题（Learning）：已知观测序列 $\boldsymbol{O}$，去估计模型 λ 的参数，包括状态转移概率矩阵 $\mathbf{A}$、初始状态概率分布 π 和观测概率分布 $\mathbf{B}$，使得在该模型下观测序列概率 $P(O|\lambda)$ 最大，通常使用 EM 算法或 Baum-Welch 算法来解决。

而 HMM 所对应的生成问题，正是上述三个问题中的评估问题，具体来说，该问题可以描述为：给定 HMM 模型的参数（初始状态分布 π、状态转移概率矩阵 $\mathbf{A}$、观测概率矩阵 $\mathbf{B}$）和一个观测序列 $\boldsymbol{O}=\{o_1,o_2,\cdots,o_T\}$，计算该观测序列的概率 $P(\boldsymbol{O}|\lambda)$。

前向算法（Forward Algorithm）是解决这类问题的一种常用方法。它可以高效地计算出给定模型参数下观测序列的概率，其基本思想是利用动态规划来计算在每个时刻 t 下的前向概率（forward probability）$\alpha_t(i)$，其中 $\alpha_t(i) = P(o_1, o_2, \cdots, o_t; i_t = q_i|\lambda)$，表示在给定模型 λ 时刻 t 下，观测序列为 $\{o_1, o_2, \cdots, o_t\}$ 且处于状态 q_i 的概率，其可以通过以下递推公式计算：

（1）$\alpha_1(i) = P(o_1, i_1 = q_i) = P(i_1 = q_i)P(o_1|i_1 = q_i) = \pi_i b_i(o_1), i = 1, 2, \cdots, N$

（2）对 $t = 1, 2, \cdots, T-1, \alpha_{t+1}(i) = \big[\sum_{j=1}^{N} \alpha_t(j)a_{ji}\big] b_i(o_{t+1}), i = 1, 2, \cdots, N$

（3）$P(O|\lambda) = P(o_1, o_2, \cdots, o_T|\lambda) = \sum_{i=1}^{N} P(o_1, o_2, \cdots, o_T; i_T = q_i|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

其中，$\alpha_{t+1}(i)$ 的递推过程如下：

$$
\begin{aligned}
&= P(o_1, o_2, \cdots, o_t, o_{t+1}; i_{t+1} = q_i|\lambda) \\
&= \sum_{j=1}^{N} P(o_1, o_2, \cdots, o_t, o_{t+1}; i_{t+1} = q_i, i_t = q_j|\lambda) \\
&= \sum_{j=1}^{N} P(o_{t+1}|o_1, \cdots, o_t; i_{t+1} = q_i, i_t = q_j; \lambda) P(o_1, \cdots, o_t; i_{t+1} = q_i, i_t = q_j|\lambda) \\
&= \sum_{j=1}^{N} P(o_{t+1}|i_{t+1} = q_i; \lambda) P(i_{t+1} = q_i|i_t = q_j; o_1, \cdots, o_t; \lambda) P(i_t = q_j; o_1, \cdots, o_t|\lambda) \\
&= \sum_{j=1}^{N} P(o_{t+1}|i_{t+1} = q_i; \lambda) P(i_{t+1} = q_i|i_t = q_j; \lambda) P(i_t = q_j; o_1, \cdots, o_t|\lambda) \\
&= \sum_{j=1}^{N} b_i(o_{t+1}) a_{ji} \alpha_t(j) \\
&= \big[\sum_{j=1}^{N} \alpha_t(j)a_{ji}\big] b_i(o_{t+1})
\end{aligned}
\tag{17.3.7}
$$

可以看到，通过前向算法，我们利用状态转移概率矩阵 $\mathbf{A}$、观测概率矩阵 $\mathbf{B}$）对时刻 t 到时刻 $t+1$ 的状态和观测值进行递推，从而局部计算出前向概率，然后再递推到全局，最终得到 $P(\boldsymbol{O}|\lambda)$，有效地估计了观测序列的生成概率，使得 HMM 在实际应用中得以真正地应用于识别、分类、预测等任务。

HMM 是一个强大的时间序列建模工具，但它也有一些限制，例如对隐含状态数量的选择通常需要领域知识或其他启发式方法，以及对数据分布的特定假设。此外，HMM 常常用于建模序列数据中的独立性假设，因此可能无法很好地处理某些类型的数据。在这些情况下，更复杂的模型，如隐马尔科夫网络（Hidden Markov Networks，HMNs）或条件随机场（Conditional Random Fields，CRFs）可能更合

适。

17.3.4 混合成员模型

混合成员模型（Mixed Membership Model）是浅层生成模型中的一种重要范例，通常用于建模多元数据和多标签数据，以捕捉数据中的潜在结构和多样性。它是一种概率图模型，假设每个数据点都可以同时属于多个成员资格或主题，而不是被划分为单一的类别或主题。这种模型广泛应用于多元数据和多标签数据的建模，具有很高的灵活性，能够更准确地捕捉数据的内在结构。

混合成员模型的基本要素包括以下内容：

（1）成员资格（Membership）：每个数据点都与多个成员资格相关联，表示数据点在不同主题或类别上的权重或概率。这些成员资格通常以概率分布的形式表示，用于描述每个数据点属于不同主题或类别的可能性；

（2）主题或类别：混合成员模型假设存在多个主题或类别，每个主题都对应于一组特征或属性。数据点的成员资格决定了它与不同主题的关联程度，因此一个数据点可以同时属于多个主题；

（3）生成过程：模型定义了数据点的生成过程，通常采用概率图模型，如潜在狄利克雷分布（Latent Dirichlet Allocation）等。生成过程描述了如何从主题中抽样生成数据点的成员资格以及如何根据成员资格生成观测数据；

（4）推断：在实际应用中，需要进行推断来估计模型参数和每个数据点的成员资格。常用的推断算法包括吉布斯抽样、变分推断等。

混合成员模型在各种领域得到了广泛应用，例如在文本分析中，LDA 主题模型能够发现文档集合中的主题，每个文档可以属于多个主题，更好地反映文档的多样性。在社交网络分析中，块主题模型能够识别用户的多重兴趣和社交圈子，允许用户同时属于多个社交群体。在推荐系统领域，混合成员模型也可以用于建模用户对多个兴趣领域的兴趣，以改进个性化推荐。在生物信息学相关领域，混合成员模型则被用于分析基因表达数据，以发现基因的多重功能。

潜在狄利克雷分布（LDA）是一种生成式概率模型，用于解释文档集合中的主题结构。它基于“每个文档可以被看作是多个主题的混合物”和“每个主题可以被看作是多个词汇的混合物”的核心基本假设，旨在通过观察文档中的词汇出现情况，推断出文档中的主题分布以及主题中的词汇分布。

在 LDA 模型中，离不开文档、主题和词汇这三个基本概念。文档集合中的每篇文档都是由一组词汇组成的。同时 LDA 假设文档集合中包含了多个主题。主题是词汇的分布，表示了每个主题中哪些词汇更有可能出现，每个主题都是一个多项分布。并且文档中的每个词汇都可以被分配给一个主题，文档的主题分布表示了文档中每个词汇所属的主题。

于是 LDA 模型有以下模型参数：

n_K：主题的数量。n_V：词汇表的大小，即不同的词汇数。n_M：文档的数量。

还含有 α 和 β 两个超参数，α 为文档-主题分布的超参数，用于控制每个文档中主题的多样性。β 为主题-词汇分布的超参数，用于控制每个主题中词汇的多样性。

LDA 基于如下这两个概率分布来生成文档：

（1）文档-主题分布：每个文档都有一个文档-主题分布 θ_d，其中 d 表示文档的索引。θ_d 是一个长度为 n_K 的向量，表示文档中各主题的分布。这个文档-主题分布 θ_d 服从狄利克雷分布，参数为 α；

（2）主题-词汇分布：每个主题都有一个主题-词汇分布 ϕ_k，其中 k 表示主题的索引。ϕ_k 是一个长度为 n_V 的向量，表示主题中各词汇的分布。每个 ϕ_k 也服从狄利克雷分布，参数为 β。

在 LDA 模型的文档生成过程中，首先为每个主题 k 从狄利克雷分布 Dirichlet(β) 中抽样主题-词汇分布 ϕ_k。再为每个文档 d 从狄利克雷分布 Dirichlet(α) 中抽样文档-主题分布 θ_d。而对于每个文档 d 中的每个词汇位置 n，先从文档-主题分布 θ_d 中抽样得到主题 z_{dn}，再从主题-词汇分布 $\phi_{z_{dn}}$ 中抽样得到词汇 w_{dn}。

通过以上生成过程，可以生成一组文档，每个文档都是由多个主题的混合构成，而每个主题又是由多个词汇的混合构成。这使得 LDA 模型能够捕捉文档集合中的主题结构，为每个词汇分配一个概率分布。

在实际应用中，通常使用推断方法来估计模型参数拟合模型，特别是文档-主题分布 θ 和主题-词汇分布 ϕ，常用的推断方法包括变分推断和吉布斯抽样。

变分推断（Variational Inference）是一种常用于 LDA 的推断方法，其目标是找到近似后验分布 $p(\theta, \phi, z|w, \alpha, \beta)$ 的解析近似。这里的参数意义与前文保持一致，w 是观察到的文档数据，α 和 β 是模型的超参数，θ 是文档-主题分布，ϕ 是主题-词汇分布，z 是主题分配。

它的基本思想是引入一个变分分布 $q(\theta, \phi, z|w)$ 来近似后验分布 $p(\theta, \phi, z|w, \alpha, \beta)$。然后，通过优化变分分布 $q(\theta, \phi, z|w)$，使其与真实后验分布尽可能接近。

变分推断的过程包含三个环节：

（1）初始化变分参数：随机初始化变分分布 $q(\theta, \phi, z|w)$ 的参数；

（2）迭代优化：通过迭代优化来更新变分参数，以使变分分布逐渐逼近真实后验分布。这通常涉及到最大化一个被称为变分下界（Variational Lower Bound）的目标函数；

（3）最终收敛：当变分下界收敛时，停止迭代。此时，变分分布 $q(\theta, \phi, z|w)$ 就可以用于估计文档-主题分布 θ 和主题-词汇分布 ϕ。

变分推断的优点包括速度较快和可扩展性强，但它是一种近似方法，可能无法完美地捕捉真实后验分布。

吉布斯抽样（Gibbs Sampling）是一种马尔科夫链蒙特卡罗（MCMC）方法，用于从后验分布 $p(\theta, \phi, z|w, \alpha, \beta)$ 中采样。与变分推断不同，吉布斯抽样直接对模型的

隐含变量进行抽样，而不需要近似。

吉布斯抽样的过程则包括以下步骤：

1、初始化：随机初始化文档-主题分布 θ、主题-词汇分布 ϕ 和主题分配 z；

2、迭代抽样：在每个迭代步骤中，依次对每个文档的每个词汇进行抽样：

（1）固定其他变量，抽样文档-主题分布 θ；

（2）固定其他变量，抽样主题-词汇分布 ϕ；

（3）固定其他变量，抽样主题分配 z；

3、重复迭代：重复上述迭代过程，直到样本收敛到真实后验分布。

吉布斯抽样的优点是它能够精确地从后验分布中采样，但缺点是它通常需要更多的计算时间，特别是对于大规模数据集。

总之，潜在狄利克雷分布（LDA）是一种用于文本分析和主题建模的强大工具，通过对文档生成过程的建模，可以推断文档中的主题结构，帮助理解文本数据的内在语义。

块主题模型（Block Topic Model），同样是一种用于处理文本数据的概率图模型。块模型扩展了标准的潜在狄利克雷分布（LDA）模型，以考虑文档之间的组块结构。

块模型的基本思想是将文档划分成不同的组块（blocks），每个组块包含一组相关的文档。这些组块可以代表不同的主题、领域或者其他文档集合的子集。块模型的目标是同时建模文档内部的主题结构和文档之间的组块结构。

块模型的数学原理与标准的 LDA 模型类似，但它引入了额外的层次结构来处理组块。以下是块模型的关键组成部分：

（1）文档-组块分布（Document-Block Distribution）：块模型引入了文档-组块分布 γ，它表示每个文档属于哪个组块。γ 是一个参数化的分布，通常服从多项分布；

（2）组块-主题分布（Block-Topic Distribution）：类似于 LDA 中的主题-词汇分布，块模型引入了组块-主题分布 ϕ，它表示每个组块包含哪些主题。ϕ 也是一个参数化的分布；

（3）文档-主题分布（Document-Topic Distribution）：与 LDA 相同，块模型中的每个文档都有一个文档-主题分布 θ，表示文档中的主题分布；

（4）主题-词汇分布（Topic-Word Distribution）：同样，块模型中的主题也有一个主题-词汇分布 β，表示主题中的词汇分布。

块模型的生成过程与 LDA 类似，但考虑了文档的组块结构。以下是块模型的生成过程：

1、为每个组块 b 从组块-主题分布 ϕ 中抽样组块-主题分布参数；

2、为每个文档-组块分布 γ 中的文档 d 抽样文档-组块分布参数；

3、对于每个文档 d 中的每个词汇位置 n：

（1）从文档-主题分布 θ 中抽样得到文档的主题 z_{dn}；

（2）从主题-词汇分布 β 中抽样得到词汇 w_{dn}。

块模型的关键特点是它能够捕捉文档的组块结构，这有助于更好地理解文档集合中的主题和领域分布。

块模型广泛应用于文本分析、主题建模和信息检索等领域。它可以用于发现文档集合中的主题分布，同时考虑文档之间的组块关系，例如新闻文章中的不同板块、学术论文中的不同领域、社交媒体中的不同话题等。

总之，块模型是一种强大的概率图模型，用于处理具有组块结构的文本数据，帮助研究者更好地理解文档集合中的主题和结构。

17.3.5 因子模型

因子模型(Factorial model) 是生成式学习中用于建模和生成多维数据的概率模型。这些模型通常用于描述观测数据的联合概率分布，并试图捕捉不同变量之间的相互关系，以便生成新的数据样本或执行其他任务，如降维、特征提取等。

Factorial 模型的核心思想是将多维数据建模为多个随机变量的乘积或组合，其中每个随机变量表示数据的一个维度或特征。这些模型通常包括以下关键概念：

（1）联合概率分布：Factorial 模型希望建立一个联合概率分布，这个分布能够描述多维数据中各个变量之间的统计依赖关系。这个联合分布可以用于生成新的数据样本，执行概率推断等任务。

（2）潜在变量：为了更好地捕捉数据中的结构，Factorial 模型引入潜在变量（latent variables），潜在变量是未观测到的变量，通常用来表示数据中的一些潜在特征或因素。Factorial 模型将多维数据的生成过程建模为潜在变量与观测变量之间的关系。这个生成过程可以表示为一个概率分布，其中潜在变量的分布和潜在变量如何影响观测变量的分布都是模型的组成部分。潜在变量可以帮助捕捉数据中的隐藏模式和变异性。

（3）参数估计：为了构建 Factorial 模型，需要估计模型的参数，包括概率分布的参数和可能的潜在变量。这个过程通常采用最大似然估计、变分推断或贝叶斯方法等统计技术。

Factorial 模型在生成式学习中有广泛的应用，例如独立成分分析（Independent Component Analysis，ICA）是一种用于盲源分离（Blind Source Separation，BSS）的统计方法，经过变换可以从混合信号中分离出原始信号或成分。概率主成分分析（Probabilistic PCA，PPCA）概率主成分分析（Probabilistic PCA，PPCA）是主成分分析（PCA）的概率化版本，它引入了概率模型来描述数据生成过程。概率因子分析（Probabilistic Factor Analysis，PFA）是一种用于建模和分析多维数据的概率模型。它类似于传统的因子分析（Factor Analysis，FA），但引入了概率模型来描述数据生成过程等。这些模型可以用于生成文本、图像、音频等多种类型的数据，也可以用于降维、特征提取、异常检测等任务。

独立成分分析（Independent Component Analysis，ICA）是一种利用统计原理进行计算的方法，其本质是一个线性变换，该变换把数据或信号分离成统计独立的非高斯的信号源的线性组合。

独立成分分析的经典问题是“鸡尾酒会问题”。在一个酒会上，n 个人站在不同的位置同时说话，另外有 n 个麦克风放在房间不同的位置录音，由于每个麦克风、人的位置均有不同，所以这 n 个麦克风录下的声音是有差别的，现在的问题即：要用这 n 个麦克风的录音，还原这 n 个人的说话声音。

为了简化问题，我们把某时刻下的某个声音看作一个实数，n 维列向量 $\boldsymbol{S^{(i)}}$ 代表时刻 (i) 原始 n 个人说话的声音，实数 $s_j^{(i)}$ 表示时刻 (i) 原始 n 个人中的第 j 个人说话的声音；n 维列向量 $\boldsymbol{X^{(i)}}$ 代表时刻 (i) 下 n 个麦克风的录音，实数 $x_j^{(i)}$ 表示时刻 (i) 下第 j 个麦克风的录音。

现在，我们得到了若干个时刻的录音数据 $\boldsymbol{X} = \left\{\boldsymbol{X^{(1)}}, \cdots, \boldsymbol{X^{(m)}}\right\}$，目标是要还原每个时刻的原始说话声音 $\boldsymbol{S} = \left\{\boldsymbol{s^{(1)}}, \cdots, \boldsymbol{S^{(m)}}\right\}$。我们用 $\boldsymbol{X} = \mathbf{A}\boldsymbol{S}$ 表示 n 个人说话声音混在一起被麦克风录下的过程，其中，$n * n$ 维矩阵 $\mathbf{A}$ 被称为混合矩阵（mixing matrix）。

满足 ICA 模型的假设:

(1) 独立成分分析中最重要的假设就是信号源统计独立。这个假设在大多数盲信号分离的情况中符合实际情况。即使当该假设不满足时，仍然可以用独立成分分析来把观察信号统计独立化，从而进一步分析数据的特性。

(2) 独立成分 $\boldsymbol{S}^{(i)}$ 服从非高斯分布。

(3) $\mathbf{A}$ 矩阵是可逆的，$\mathbf{A}$ 的逆矩阵记为 $\mathbf{W}$，称为解混矩阵 (unmixing matrix)，只要能求出 $\mathbf{W}$，就能通过 $\boldsymbol{S}^{(i)} = \mathbf{W}\boldsymbol{X}^{(i)}$ 得到结果。令 $\boldsymbol{W}_i^T$ 为矩阵 $\mathbf{W}$ 的第 i 行 $(\boldsymbol{W}_i \in \mathcal{R}^n)$，则 $\mathbf{W}$ 可表示为 $\begin{pmatrix} \boldsymbol{W}_1^T \\ \vdots \\ \boldsymbol{W}_n^T \end{pmatrix}$，于是时刻 i 的第 j 个声音来源为 $\mathbf{s}_j^{(i)} = \boldsymbol{W}_j^T \boldsymbol{X}^{(i)}$。

在介绍独立成分分析前，首先了解一下线性变换对密度函数的影响。

$$F_{\mathbf{X}}(\boldsymbol{X}) = P(\mathbf{X} \leq \boldsymbol{X}) = P(\mathbf{A}\boldsymbol{S} \leq \boldsymbol{X}) = P(\boldsymbol{S} \leq \mathbf{W}\boldsymbol{X}) = F_{\boldsymbol{S}}(\mathbf{W}\boldsymbol{X})$$

则有

$$p_{\boldsymbol{X}}(\boldsymbol{S}) = \frac{d}{d\boldsymbol{X}} F_{\boldsymbol{X}}(\boldsymbol{X}) = \frac{d}{d\boldsymbol{X}} F_{\boldsymbol{S}}(\mathbf{W}\boldsymbol{X}) = p_{\boldsymbol{S}}(\mathbf{W}\boldsymbol{X})|\mathbf{W}|$$

下面从最大似然估计的角度来介绍 ICA 算法。

首先假设每个说话人的原始声音 $\boldsymbol{S}_i$ ($\boldsymbol{S}_i$ 为 $\boldsymbol{S}$ 的第 i 行) 的分布是由密度函数 $p_{\boldsymbol{S}}$ 决定的。那么 n 个人的原始声音 $\boldsymbol{S}$ 的联合密度为:

$$p(\boldsymbol{S}) = \prod_{i=1}^{n} p_{\boldsymbol{S}}\left(\boldsymbol{S}_i\right)$$

又 $\boldsymbol{S}=\mathbf{W}\boldsymbol{X}, \boldsymbol{S_i}=\boldsymbol{W_i}^T\boldsymbol{X}$, 则有

$$p(\boldsymbol{X})=p_{\boldsymbol{X}}(\mathbf{W}\boldsymbol{X})|\mathbf{W}|=\prod_{i=1}^{n}p_s\left(\boldsymbol{W}_i^T\boldsymbol{X}_i\right)|\mathbf{W}|$$

因此我们需要确定每个原始声音的分布密度函数 p_s 。对于实数值的随机变量 z 而言, 其累积分布函数是 $F(z_0)=P(z\leq z_0)=\int_{-\infty}^{z_0}p_z(z)dz$, 为了确定 $p_{\boldsymbol{s}}$, 我们首先确定它对应的累积分布函数, 这里我们选择 Sigmoid 函数作为累积分布函数, $g(\boldsymbol{s})=\dfrac{1}{1+e^{-\boldsymbol{s}}}$, 则 $p_{\boldsymbol{s}}(\boldsymbol{s})=g'(\boldsymbol{s})$ 对于大小为 m 的训练集 $\left\{\boldsymbol{X}^{(1)},\ldots,\boldsymbol{X}^{(m)}\right\}$, 我们可以写出关于 $\mathbf{W}$ 的似然函数:

$$\begin{aligned}L(\mathbf{W})&=\prod_{i=1}^{m}p\left(\boldsymbol{X}^{(i)}\right)=\prod_{i=1}^{m}p_{\boldsymbol{s}}\left(\mathbf{W}\boldsymbol{X}^{(i)}\right)|\mathbf{W}|=\prod_{i=1}^{m}\left(\prod_{j=1}^{n}p_{\boldsymbol{s}}\left(\boldsymbol{w}_j^T\boldsymbol{X}^{(i)}\right)\right)|\mathbf{W}|\\&=\prod_{i=1}^{m}\left(\prod_{j=1}^{n}g'\left(\boldsymbol{w}_j^T\boldsymbol{X}^{(i)}\right)\right)|\mathbf{W}|\end{aligned}$$

得到对数似然函数为:

$$\begin{aligned}l(\mathbf{W})&=\sum_{i=1}^{m}\log\left(\left[\prod_{j=1}^{n}g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right]|\mathbf{W}|\right)\\&=\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\log\left(g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)+\log|\mathbf{W}|\right)\end{aligned}$$

令 $\nabla_{\mathbf{W}}l=0$, 我们有

$$\frac{\partial(l(\mathbf{W}))}{\partial\mathbf{W}}=\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\left(\frac{\partial\log\left(g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial\mathbf{W}}\right)+\frac{\partial\log|\mathbf{W}|}{\partial\mathbf{W}}\right]$$

由 $g'(\boldsymbol{X})=g(\boldsymbol{X})(1-g(\boldsymbol{X}))$ 得

$$\frac{\partial\log\left(g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial\mathbf{W}}=\frac{\partial\log\left(g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial\mathbf{W}}+\frac{\partial\log\left(1-g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial\mathbf{W}}$$

上式中第一项为

$$\frac{\partial\log\left(g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial\mathbf{W}}=\frac{1}{g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)}g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\frac{\partial\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}}{\partial\mathbf{W}}$$

$$=\left(1-g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)\begin{pmatrix}0&\ldots&0\\\vdots&&\vdots\\x_1^{(i)}&\ldots&x_n^{(i)}\\\vdots&&\vdots\\0&\ldots&0\end{pmatrix}\text{第 j 行}$$

第二项为

$$\frac{\partial \log\left(1-g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)}{\partial \mathbf{W}}$$

$$=\frac{1}{1-g\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)}\left(-g'\left(\boldsymbol{W}_j^T\boldsymbol{X}^{(i)}\right)\right)\frac{\partial \boldsymbol{W}_j^T\boldsymbol{X}^{(i)}}{\partial \mathbf{W}}\begin{pmatrix}0 & \dots & 0\\ \vdots & & \vdots\\ x_1^{(i)} & \dots & x_n^{(i)}\\ \vdots & & \vdots\\ 0 & \dots & 0\end{pmatrix}\text{第 j 行}$$

利用公式 $\nabla_{\mathbf{W}}|\mathbf{W}|=|\mathbf{W}|\left(\mathbf{W}^{-1}\right)^T$，我们有

$$\frac{\partial(l(\mathbf{W}))}{\partial \mathbf{W}}=\sum_{i=1}^{m}\left[\sum_{j=1}^{n}\left(1-2g\left(\boldsymbol{w}_j^T\boldsymbol{x}^{(i)}\right)\right)\begin{pmatrix}0 & \dots & 0\\ \vdots & & \vdots\\ x_1^{(i)} & \dots & x_n^{(i)}\\ \vdots & & \vdots\\ 0 & \dots & 0\end{pmatrix}+\left(\mathbf{W}^T\right)^{-1}\right]$$

$$=\sum_{i=1}^{m}\left[\begin{pmatrix}1-2g\left(\mathbf{W}_1^T\mathbf{X}^{(i)}\right)\\ \vdots\\ 1-2g\left(\mathbf{W}_n^T\mathbf{X}^{(i)}\right)\end{pmatrix}\boldsymbol{X}^{(i)^T}+\left(\mathbf{W}^T\right)^{-1}\right]=0$$

得到梯度上升算法的 $\mathbf{W}$ 的更新公式为:

$$\mathbf{W}:=\mathbf{W}+\alpha\left(\begin{bmatrix}1-2g\left(\boldsymbol{w}_1^T\boldsymbol{X}^{(i)}\right)\\ \vdots\\ 1-2g\left(\boldsymbol{w}_n^T\boldsymbol{X}^{(i)}\right)\end{bmatrix}\boldsymbol{X}^{(i)^T}+\left(\mathbf{W}^T\right)^{-1}\right)$$

基于 $\mathbf{W}$，就可以通过 $\boldsymbol{S}=\mathbf{W}\boldsymbol{X}$ 还原出原信号 $\boldsymbol{S}$。

注:

(1) 在这里我们假定观察数据已被预处理过且有 0 均值, 因为对于 $\boldsymbol{S}$ 的分布函数为 Sigmoid 的假设, 自然有 $E(\boldsymbol{S})=0$, 这表明 $E(\boldsymbol{S})=E(\mathbf{A}\boldsymbol{S})=0$。对于一般的观测数据在使用 ICA 之前通常需要进行中心化和白化处理。

(2) 实际上，这里的假设 $\boldsymbol{S}^{(i)}$ 之间相互独立，往往是不成立的。因为一般 $\boldsymbol{X}^{(i)}$ 是连续一段时间上的数据，相邻时刻的数据关联性很强。但我们可以通过随机梯度下降算法, 对全体 $\boldsymbol{X}^{(i)}$ 随机重新排列 (random shuffle), 这样就可以近似地认为 $\boldsymbol{X}^{(i)}$ 之间相互独立。

概率主成分分析（Probabilistic PCA，PPCA）概率主成分分析是主成分分析（PCA）的一种概率性扩展，是一种对数据进行降维、特征提取以及噪声建模的分析方法。

概率 PCA 模型运用了这样一种观测现象：除了一些微小残余的重构误差，数据中的大多数变化可以由潜变量 Z 来描述。

首先 PCA 是一种常用的降维方法，它的基本思想是：通过线性变换将原始数据变换为一组各维度线性无关的表示。如果有一份样本为 n，维度为 d 的数据 $\mathbf{X} \in \mathcal{R}^{n\times d}$, 如果能找到一个矩阵 $\mathbf{Z} \in \mathcal{R}^{n\times k}$，$k << d$，使得 $\mathbf{X}$ 与 $\mathbf{ZW}^T$ 尽可能接近，其中 $\mathbf{W} \in \mathcal{R}^{d\times k}$，即

$$\mathbf{X} \approx \mathbf{ZW^T}$$

更具体的说，对于某个样本 $\boldsymbol{X_i} \in \mathcal{R}^{1\times d}$，其近似方式为

$$\boldsymbol{X_i} \approx \boldsymbol{Z_i}\mathbf{W}^T$$

一般来说，我们只需要估计 $\mathbf{W}$ 的值，因为如果 $\mathbf{W}$ 已知的话，$\boldsymbol{Z}_i$ 是可以直接用 $\boldsymbol{X}_i$ 求出来的：

$$\boldsymbol{Z_i} = \arg\min_t \parallel \boldsymbol{X}_i - \boldsymbol{Z}_i\mathbf{W}^T \parallel^2 = \boldsymbol{X}_i(\mathbf{WW}^T)^{-1}\mathbf{W}^T$$

所以问题转化为了求出 $\mathbf{W}$，可以依照以下步骤求解

（1）将原始数据按列组成 n 行 d 列矩阵 $\mathbf{X}$,

（2）将 X 的每一行 (代表一个属性字段) 进行零均值化，即减去这一行的均值,

（3）求出协方差矩阵 $\mathbf{C} = \frac{1}{d}\mathbf{XX}^T$,

（4）求出协方差矩阵的特征值及对应的特征向量,

（5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 $\mathbf{P}$,$\mathbf{P}=(\mathbf{W^{-1}})^T$,

（6）$\mathbf{Z} = \mathbf{PX}$,$\mathbf{Z}$ 即为降为 k 维后的数据。

以高斯分布举例，在 PCA 中，对于零均值化的数据 $\mathbf{X}$，我们的假设是

$$\mathbf{X} \approx \mathbf{ZW^T}$$

但是在 PPCA（Probabilistic PCA）中，假设是

$$\mathbf{X} \sim \mathcal{N}(\mathbf{ZW}^T, \sigma^2\mathbf{I})\ \mathbf{Z} \sim \mathcal{N}(0, \mathbf{I})$$

显然当 $\sigma \to 0$ 时，PPCA 与 PCA 是等价的。而实际情况中 PPCA 很灵活，可以是高斯分布，也可以是拉普拉斯分布等。

17.4 深层生成模型

深层生成模型（Deep Generative Models）是机器学习领域的一个重要分支，它涉及使用深度学习技术来建立能够生成复杂数据的模型。这些模型具有强大的生成

能力，能够生成高质量的图像、文本、音频等多种类型的数据。深层生成模型的核心思想是使用深度神经网络来建模数据的生成过程，通常包括生成网络（Generator Network）与判别网络（Discriminator Network）两个部分。生成网络是一个深度神经网络，它接受随机噪声或其他输入，并尝试生成与训练数据相似的新数据样本。生成网络通常包括多个层次，从低级特征到高级特征逐渐生成数据。判别网络也是一个深度神经网络，它的任务是将真实数据与生成数据区分开来，从而评估生成网络生成的数据样本的真实性，判别网络的训练能够使生成网络生成更逼真的数据。

常见的深层生成模型有变分自动编码器（Variational Auto Encoder）、生成对抗网络（Generative Adversarial Network）。VAE 是一种基于概率的生成模型。它包括一个编码器和一个解码器，用于将输入数据映射到潜在空间并从潜在空间生成数据。它使用变分推断来估计潜在变量的分布，使其能够生成具有一定多样性的数据。GAN 是最著名的深层生成模型之一，它包括一个生成器和一个判别器，通过博弈过程相互对抗。GAN 中的生成器试图生成逼真的数据以欺骗判别器，而判别器试图区分真实数据和生成数据。这个过程迭代进行，直到生成器生成高质量的数据。

17.4.1 自回归网络

自回归（Auto-regressive）是统计学中处理时间序列的方法，其用同一变量之前各个时刻的观测值来预测该变量当前时刻的观测值。自回归网络（Auto-regressive Network）则是用来形容用形如：$P(x_d|x_{d-1},\cdots,x_1)$ 这样的条件概率表示可见层数据（即观测变量）相邻元素的关系，以这些条件概率乘积表示观测变量的联合概率分布的模型。这样的模型也被称为完全可见的贝叶斯网络（fully-visible Bayes networks, FVBN），并成功地以许多形式使用。

自回归网络的基本形式通常包括线性自回归网络（Linear AutoRegressive Network）和神经自回归网络（Neural AutoRegressive Network）两种。其中，线性自回归网络是自回归网络中最简单的形式，其没有隐藏单元、参数和特征共享，每个 $P(x_d|x_{d-1},\cdots,x_1)$ 均被参数化为一个线性模型，见图17.1：目标任务是从前 $d-1$ 个变量预测第 d 个变量，其中每个预测由线性预测器做出。（对于实值数据的线性回归、对于二值数据的逻辑回归、对于离散数据的 $softmax$ 回归；如果变量是连续的，线性自回归网络只是表示多元高斯分布的另一种方式，只能用以捕获观测变量间线性的成对相互作用）。

本质上，线性自回归网络其实是线性分类方法在生成模型上的推广。像线性分类器一样，它们可以用凸损失函数训练，但模型本身不提供增加其容量的方法，因此必须使用其他技术（如输入的基扩展或核技巧）来提高容量。

接下来我们介绍一下神经自回归网络。神经自回归网络是具有与线性自回归相同结构（图17.1）的有向图模型，但该模型采用不同的条件分布参数，能够根据实际

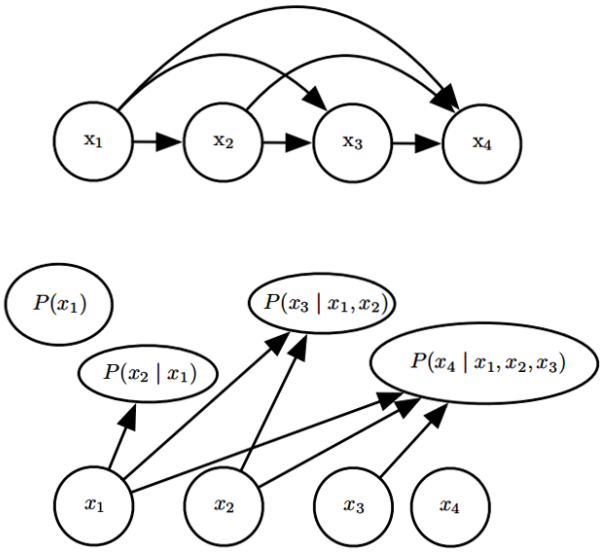


图 17.1 线性自回归模型

需求增加容量，允许近似任意联合分布，并且神经自回归网络还可以使用深度学习中常见的参数共享和特征共享等方法增强泛化能力，通过堆叠神经层来捕捉输入序列的自回归性质，使其能够处理各种时间序列建模任务，同时能避免如 DBN 等传统概率图模型中高维数据引发的维数灾难。

比如在表格离散概率模型中，每个条件分布由概率表表示，其中所涉及的变量的每个可能配置都具有一个条目和一个参数。通过使用神经网络，可以获得两个优点：

（1）通过具有 $(d-1) \times k$ 个输入和 k 个输出的神经网络（如果变量是离散的并有 k 个值，使用 $one-hot$ 编码），参数化每个 $P(x_d|x_{d-1},\cdots,x_1)$，让我们不需要指数量级参数（和样本）的情况下就能估计条件概率，并仍能够捕获随机变量之间的高阶依赖性。

（2）不需要对预测每个 x_d 使用不同的神经网络，如图17.2。即：预测 x_d 所计算的隐藏层特征（即表示为 h_i 的隐藏单元的组）可以重新用于预测 $x_{d+k}(k>0)$。

17.4.2 变分自动编码器

变分自动编码器（Variational Auto Encoder）是一种生成式深度学习模型，结合了自动编码器（Autoencoder）的编码和解码过程以及概率建模的思想，旨在学习数据的潜在分布，并能够生成具有多样性的新数据样本。

VAE 的核心思想在于将数据视为从潜在空间中采样的结果，其中每个数据点 $\boldsymbol{X}$ 都被认为是从潜在变量 $\boldsymbol{Z}$（也可称为潜在编码）采样而来。因此 VAE 的目标就是学

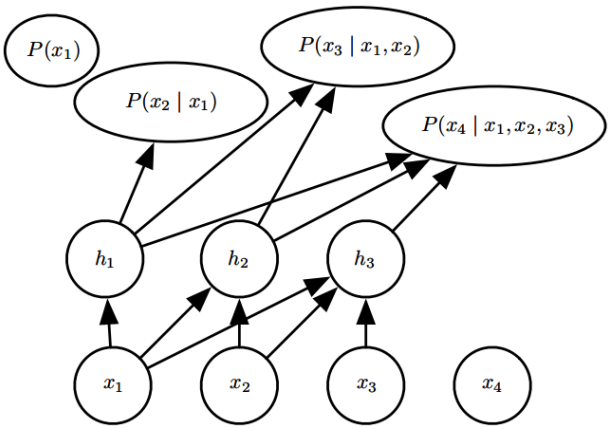


图 17.2 神经自回归模型

习如何将输入数据 $\boldsymbol{X}$ 映射到潜在空间中的概率分布 $p(\boldsymbol{Z}|\boldsymbol{X})$，以及如何从潜在空间中采样以生成新的数据样本。

VAE 的结构由编码器（Encoder）和解码器（Decoder）这两个主要组件组成。

编码器负责将输入数据 $\boldsymbol{X}$ 映射到潜在空间中的分布参数，通常使用均值向量 $\boldsymbol{\mu}$ 和对数方差向量 $\log(\boldsymbol{\sigma^2})$ 来表示潜在空间中的正态分布。这些参数是根据输入数据 $\boldsymbol{X}$ 计算的，具体如下：

$$\begin{aligned} \boldsymbol{\mu} &= f_{\text{enc}}(\boldsymbol{X}) \\ \log(\boldsymbol{\sigma^2}) &= f_{\text{enc}}(\boldsymbol{X}) \end{aligned} \tag{17.4.1}$$

其中，$f_{\text{enc}}(\boldsymbol{X})$ 是编码器网络的输出。

解码器负责将从潜在空间中采样的点 $\boldsymbol{Z}$ 映射回数据空间，以生成数据的重建 $\hat{\boldsymbol{X}}$。通常，解码器会使用一个神经网络来实现这个映射，具体如下：

$$\hat{\boldsymbol{X}} = f_{\text{dec}}(\boldsymbol{Z}) \tag{17.4.2}$$

其中，$f_{\text{dec}}(\boldsymbol{Z})$ 是解码器网络的输出。

既然 VAE 的结构如此设计，其目的是将输入数据映射到潜在空间并生成数据的重建，那么如何让模型学会有效地完成这一任务？这就需要引入它独特的损失函数来指导模型的训练。接下来，我们将深入探讨 VAE 的损失函数，以理解模型是如何在训练中优化自己的能力以生成高质量数据和有意义的潜在表示的。

VAE 的损失函数由两个部分组成：

（1）重建损失（Reconstruction Loss）：重建损失度量了解码器生成的数据重建与原始输入数据之间的差异，其主要作用在于促使 VAE 生成与输入数据高度相似的数据样本。通过最小化重建损失，VAE 模型学习将输入数据映射到潜在空间中的特定分布，从而使得 VAE 能够生成逼真的数据样本。在实践中，通常采用均方误差

（MSE）或交叉熵（CrossEntropy）作为重建损失的具体形式，数学表达如下：

$$\text{Reconstruction Loss} = -\mathrm{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X})}\left[\log p(\boldsymbol{X}|\boldsymbol{Z})\right] \tag{17.4.3}$$

其中，$q(\boldsymbol{Z}|\boldsymbol{X})$ 是编码器输出的潜在空间中的后验分布。

（2）KL 散度损失（KL Divergence）：设置 KL 散度损失的目的在于确保编码器生成的潜在表示分布逼近标准正态分布，这一过程有助于对模型进行正则化，以避免潜在表示过于离散或过于聚集在特定区域。KL 散度损失通过促进潜在表示的平滑性，推动模型在潜在空间中学习到富有信息的特征。具体而言，KL 散度损失用于衡量编码器输出的潜在分布与标准正态分布之间的差异，以鼓励学习一个接近正态分布的潜在表示。这种正则化有助于确保模型在潜在空间中能够产生连续且有意义的编码，从而提高了 VAE 的性能和泛化能力，数学表达如下：

$$\text{KL Divergence Loss} = -\frac{1}{2}\sum_{j=1}^{J}\left(1+\log(\sigma_j^2)-\mu_j^2-\sigma_j^2\right) \tag{17.4.4}$$

其中，μ_j 和 σ_j^2 是编码器输出的均值和方差的元素。

因此，VAE 的总损失函数即重建损失和 KL 散度损失的和：

$$\mathcal{L} = \text{Reconstruction Loss} + \text{KL Divergence Loss} \tag{17.4.5}$$

在 VAE 的训练过程中，最小化总损失函数 $\mathcal{L}$ 的同时，这两个损失项协同工作，使 VAE 能够学习有效的潜在表示并生成高质量的数据样本。VAE 的训练过程同样包括前向传播和反向传播过程，下面是 VAE 的训练过程的详细步骤：

（1）前向传播：给定输入数据 $\boldsymbol{X}$，编码器（Encoder）网络 $f_{\text{enc}}(\boldsymbol{X})$ 通过均值向量 $\boldsymbol{\mu}$ 和对数方差向量 $\log(\boldsymbol{\sigma^2})$ 表示潜在分布 $q(\boldsymbol{Z}|\boldsymbol{X})$。再从 $q(\boldsymbol{Z}|\boldsymbol{X})$ 中随机采样得到一个潜在变量 $\boldsymbol{Z}$；

（2）解码过程：解码器（Decoder）网络 $f_{\text{dec}}(\boldsymbol{Z})$ 将潜在变量 $\boldsymbol{Z}$ 映射回数据空间，生成数据重建 $\hat{\boldsymbol{X}}$；

（3）计算损失：分别计算前文提到的重建损失 ReconstructionLoss 和 KL 散度损失 KLDivergenceLoss，得到总损失 $\mathcal{L}$；

（4）反向传播和参数更新：使用反向传播算法计算损失函数 $\mathcal{L}$ 关于模型参数（编码器和解码器的权重和偏差）的梯度。使用梯度下降方法或其变种，如 Adam，来更新模型参数，以减小总损失 $\mathcal{L}$。

（5）迭代训练：重复上述步骤，通过多次迭代训练模型，使其逐渐学会生成高质量的数据重建和有意义的潜在表示。

VAE 训练完成后，便可以使用模型进行采样和生成新的数据样本：

（1）从潜在空间中采样：随机从标准正态分布中采样一个潜在变量 $\boldsymbol{Z}$；

（2）解码生成：将采样的潜在变量 $\boldsymbol{Z}$ 输入解码器网络 $f_{\text{dec}}(\boldsymbol{Z})$。解码器再将 $\boldsymbol{Z}$ 映射回数据空间，生成新的数据重建 $\hat{\boldsymbol{X}}$。

生成的 $\hat{\boldsymbol{X}}$ 即为模型生成的新数据样本。这些样本具有与训练数据相似的特征，但通常会具有一定的多样性，因为潜在变量 $\boldsymbol{Z}$ 是从标准正态分布中采样的。通过重复上述步骤，可以生成多个新的数据样本，从而实现了对数据分布的建模和生成新数据的能力。这使得 VAE 成为了强大的生成模型，适用于多种应用，如图像生成、数据降维、数据去噪等。

值得注意的是，VAE 通过学习潜在空间的结构，使得潜在空间中的不同区域对应于不同的数据特征。这意味着在潜在空间中，可以沿着不同方向进行插值和探索，以生成具有特定特征的新数据样本。例如，可以在潜在空间中找到一个点 $\boldsymbol{Z}_1$ 对应于一张人脸图片，另一个点 $\boldsymbol{z}\boldsymbol{Z}_2$ 对应于另一张人脸图片，然后在它们之间进行插值，生成介于两张图片之间的新的人脸图片。

这种采样和生成的能力使得 VAE 在生成式深度学习中具有广泛的应用前景。在实际应用中，可以使用训练好的 VAE 模型来生成符合特定需求的新数据，从而为各种任务提供了有力的工具。

17.4.3 生成对抗网络

生成对抗网络（Generative Adversarial Network，GAN）是当下机器学习领域最热门的研究方向之一，在图像生成领域占有绝对优势。GAN 本质上是将难以求解的似然函数转化成神经网络，让模型自己训练出合适的参数拟合似然函数，这个神经网络就是 GAN 中的判别器。

GAN 内部对抗的结构可以看成是一个训练框架，原理上可以训练任意的生成模型，通过两类模型之间的对抗行为来优化模型参数，巧妙地避开求解似然函数的过程。这个优势使 GAN 具有很强的适用性和可塑性，可以根据不同的需求改变生成器和和判别器，尽管模型本身具有很多训练上的难点，但随着各种解决方法的出现，人们逐渐解决了 GAN 模型的训练问题。

该模型原理简单且易理解，其生成的图片清晰度和分辨率超过其他生成模型，但缺点是训练不稳定，因此人们最关注的是模型的生成效果和训练稳定性两方面。本节首先介绍 GAN 的基本原理和模型结构，然后基于 WGAN 的稳定性研究和 GAN 模型框架的发展作进一步说明。

GAN 基本原理

GAN 中的博弈方是一个生成器和一个判别器，生成器的目标是生成逼真的伪样本让判别器无法判别出真伪，判别器的目标是正确区分数据是真实样本还是来自生成器的伪样本，在博弈的过程中，两个竞争者需要不断优化自身的生成能力和判别能力，而博弈的结果是找到两者之间的纳什均衡，当判别器的识别能力达到一定程度却无法正确判断数据来源时，就获得了一个学习到真实数据分布的生成器，GAN 的模型结构如图 17.3 所示。

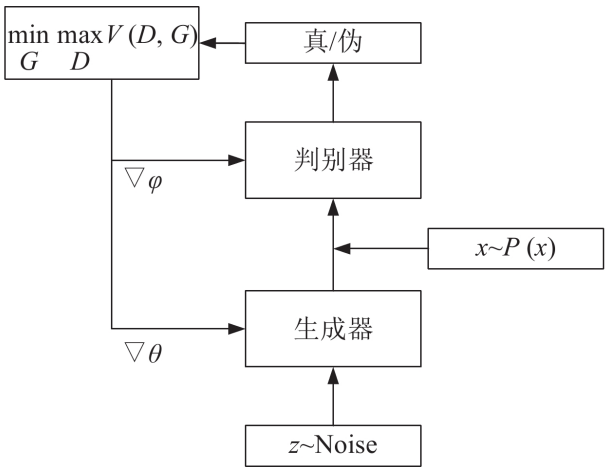


图 17.3 GAN 模型结构

GAN 中的生成器和判别器可以是任意可微函数，通常用多层的神经网络表示。生成器 $G(\boldsymbol{Z};\theta)$ 是输入为随机噪声、输出伪样本、参数为 θ 的网络，判别器 $D(\boldsymbol{X};\varphi)$ 是输入为真实样本和伪样本、输出为 0 或 1（分别对应伪样本和真实样本）、参数为 φ 的二分类网络。GAN 根据生成器和判别器不同的损失函数分别优化生成器和判别器的参数，避免了计算似然函数的过程。

GAN 训练方法

GAN 的训练机制由生成器优化和判别器优化两部分构成，下面分析两者的目标函数和优化过程：

1、优化判别器：固定生成器 $G(\boldsymbol{Z};\theta)$ 后优化判别器 $D(\boldsymbol{X};\varphi)$，由于判别器是二分类模型，目标函数选用交叉熵函数：

$$\max_{D} V(D) = \mathrm{E}_{\mathrm{X}\sim\mathrm{P_r}}[\log D(\boldsymbol{X})] + \mathrm{E}_{\boldsymbol{X}\sim P_g}[\log(1 - D(\boldsymbol{X}))] \tag{17.4.6}$$

其中，P_r 是真实样本分布，P_g 表示由生成器产生的样本分布。判别器的目标是正确分辨出所有样本的真伪，该目标函数由两部分组成：

（1）对于所有的真实样本，判别器应该将其判定为真样本使输出 $D(\boldsymbol{X})$ 趋近 1，即最大化 $\mathrm{E}_{\boldsymbol{X}\sim P_r}[\log D(\boldsymbol{X})]$；

（2）对于生成器伪造的所有假样本，判别器应该将其判定为假样本使输出尽量接近 0，即最大化 $\mathrm{E}_{\boldsymbol{X}\sim P_g}[\log(1 - D(\boldsymbol{X}))]$。

2、优化生成器：固定训练好的判别器参数，考虑优化生成器模型参数。生成器希望学习到真实样本分布，因此优化目的是生成的样本可以让判别器误判为 1，即最大化 $\mathrm{E}_{\boldsymbol{X}\sim P_g}[\log(D(\boldsymbol{X}))]$，所有生成器的目标函数为：$\min_{G} V(G)\mathrm{E}_{\boldsymbol{X}\sim P_g}[\log(1 - D(\boldsymbol{X}))]$；后来又提出了一个改进的函数为：$\min_{G} V(G)\mathrm{E}_{\boldsymbol{X}\sim P_g}[-\log D(\boldsymbol{X})]$。

从该目标函数可以看出，生成器的梯度更新信息来自判别器的结果而不是来自数据样本，相当于用神经网络拟合出数据分布和模型分布之间的距离，从根本上回

避了似然函数的难点，这一思想对深度学习领域产生了深远的影响，也是 GAN 模型的优势所在。

固定生成器参数，根据判别器目标函数可得到：$-P_r(\boldsymbol{X})\log D(\boldsymbol{X})-P_g(\boldsymbol{X})\log[1-D(\boldsymbol{X})]$，令上式对 $D(\boldsymbol{X})$ 的导数为 0，即可得到判别器最优解的表达式：

$$D^*(\boldsymbol{X})=\frac{P_r(\boldsymbol{X})}{P_r(\boldsymbol{X})+P_g(\boldsymbol{X})} \tag{17.4.7}$$

然后固定判别器最优解 D^* 的参数，基于此训练好的生成器就是最优生成器。此时 $P_r(\boldsymbol{X})=P_g(\boldsymbol{X})$，判别器认为该样本是真样本还是假样本的概率均为 0.5，说明此时的生成器已可以生成足够逼真的样本。

GAN 存在的问题

GAN 模型刚提出时存在很多严重缺陷，效果也不突出，从模型结构到稳定性、收敛性等都处于探索阶段，导致一段时间内没有展现出应有的能力。其不足之处可以总结为以下几点：

（1）模型难以训练，经常出现梯度消失导致模型无法继续训练；生成器形式过于自由，训练时梯度波动极大造成训练不稳定；需要小心地平衡生成器和判别器的训练程度，使用更新一次判别器后更新 k 次生成器的交替训练法并不能很好地缓解训练问题；

（2）出现模式崩溃（Model collapse），具体表现为生成样本单一，无法生成其他类别的样本；

（3）目标函数的形式导致模型在训练过程没有任何可以指示训练进度的指标。

Arjovsky、Huszar 等人分别对 GAN 模型存在的问题进行了分析，并作出了相应改进，但由于其改进方法均基于 KL 散度或 JS 散度，所以模型仍存在某些问题；而 WGAN（Wasserstein GAN）提出用 Wasserstein 距离替代 KL 散度和 JS 散度，改变了生成器和判别器的目标函数，并对判别器施加 Lipschitz 约束以限制判别器的梯度，只用几处微小的改动就解决了 GAN 不稳定的问题，基本消除了简单数据集上的模型崩溃问题。

当然，基于 GAN 的改进模型还有很多，如：InfoGAN 将生成器的输入噪声分解为一个噪声和表示真实数据分布的结构化语义特征，用于表示生成数据的不同特征维度，并使用基于互信息的正则化项用于约束生成器，使 GAN 能够学习解开纠缠的引表示；PGGAN 可以随着训练的进行逐渐增加层数，在训练好的浅层网络权重的基础上逐渐加深网络的深度，使模型能够学习更高分辨率的图像；自注意力生成对抗网络（Self-Attention generative adversarial，SAGAN）将注意力机制引入到 GAN 结构中，使模型能够更好的处理图像中大范围、多层次的依赖关系，并在生成器中应用谱归一化方法作为正则项约束生成器参数等，就不再一一详述。

在可预见的未来，GAN 模型必将是图像生成领域内最具代表性、最成功的深度生成模型。很多与 GAN 结合的生成式模型应用在自然语言处理中，例如 VAE 与 GAN 的结合、利用强化学习的 SeqGAN 等。这些模型的提出不仅扩展了 GAN 的

应用领域，而且提高了 GAN 理论上的完备程度。

17.4.4 流模型

流模型（Flow model）也称为 Normalizing Flow 模型，是一类用于建模和处理概率分布的生成模型。Flow 模型的主要目标是将一个简单的概率分布（通常是均匀分布或高斯分布）通过一系列可逆的变换映射成复杂的概率分布，从而能够生成新的样本或对数据进行概率密度估计。

Flow 模型基本原理

Flow 模型结构如图 17.4 所示，该模型的原理用一句话来说就是变量替换，通过对简单分布做可逆变换，从而变成一个复杂的分布，从而对数据进行估计。设 z_0 是一个分布简单的随机变量（比如高斯分布，均匀分布等），记密度函数为 $P_{Z_0}(z_0)$，将它进行一个变换 f_0，其中 f_0 使连续且可逆，记变换后的随机变量为 z_1，密度函数为 $P_{Z_1}(z_1)$，那么

$$\int_{Z_0} P_{Z_0}(z_0)dz_0 = \int_{Z_1} P_{Z_1}(z_1)dz_1$$

$$P_{Z_1}(z_1) = |\frac{dz_0}{dz_1}|P_{Z_0}(z_0)$$

由于 $z_1 = f_0(z_0)$,f_0 是可逆的，

$$z_0 = f_0^{-1}(z_1)$$

则，

$$P_{Z_1}(z_1) = |det(\frac{dz_0}{dz_1})|P_{Z_0}(z_0)$$

其中，$det(\frac{dz_0}{dz_1})$ 是若尔当矩阵的行列式。

经过推导我们发现，在 Flow 模型中，由于每一个 f_i 都是给定的，可以通过变量替换找到最终的复杂概率分布与最初简单分布的关系，这意味着可以轻松地在生成样本和估计密度之间切换。

Flow 模型在生成式建模、数据增强、密度估计等任务中具有广泛的应用。一些常见的 Flow 模型包括：

(1) RealNVP (Real-valued Non-Volume Preserving)：一种基于条件变换的 Flow 模型，常用于图像生成和密度估计。

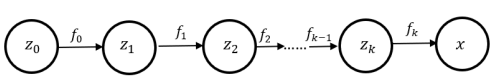


图 17.4 流模型

（2）Glow（Generative Flow with Invertible 1x1 Convolutions）：具有 1x1 可逆卷积的 Flow 模型，用于图像生成和图像超分辨率。

（3）MAF（Masked Autoregressive Flow）：基于自回归结构的 Flow 模型，常用于文本生成和密度估计。

（4）IAF（Inverted Autoregressive Flow）：MAF 的变种，能够更好地处理长依赖关系的数据。

Flow 模型的主要优势之一是其可逆性，使得在生成和推断之间无缝切换成为可能。然而，Flow 模型的训练和推断可能较为复杂，且需要适当的处理来确保生成的样本质量。它们已经在生成式深度学习中引起了广泛的关注，并取得了一些重要的研究进展。

17.4.5 概率扩散去噪生成模型

DDPM 是概率扩散去噪生成模型（Denoising Diffusion Probabilistic Models）的缩写。DDPM 在生成高质量图像和视频方面表现出色，特别是处理高分辨率、复杂纹理或结构的情况。之前主流的生成网络 GAN 存在一些缺点：其要训练两个网络，所以难度较大、容易不收敛、多样性较差，并且模型在训练过程中不稳定，只要骗过判别器即可。而概率扩散去噪生成模型这一模型使用了”扩散”的过程，用一种更简单的方法诠释了生成模型应该如何学习和生成，相比起来更加简便，之后扩散模型可能会替代 GAN 成为主流的生成模型。

DDPM 基本原理

概率扩散去噪生成模型这一模型的核心思想是通过”扩散”（扩散是指物质粒子从高浓度区域向低浓度区域移动的过程）的过程，逐渐将数据点从一个初始状态引入噪声，直到达到目标分布。

以 ai 绘图为例，先描述前向加噪的过程。对于一张彩色图片，常见的 RGB24 格式是指每个像素使用 R、G、B 三个通道表示，每个通道的取值范围在 $[0,255]$ 之间，分别代表红、绿、蓝三原色的比重，

1、将三个通道的数据通过归一化处理（以 a 为例）

$$b=\frac{a}{255}\times 2-1$$

映射到 [-1,1] 上，

2、通过随机采样一个同样大小的图片，所有像素通道数值服从标准正态分布，这个图片就是一个噪声

3、选取一定的参数将原图片与随机采样生成的图片进行混合，β 为参数，且 $\beta\in(0,1)$，

$$\boldsymbol{X}=\sqrt{\beta}\boldsymbol{\epsilon}+\sqrt{1-\beta}\boldsymbol{b}$$

这样就完成了一个加噪的过程，再使用此公式不断迭代，可以从原始图片 $\boldsymbol{X}_0$ 生成 $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots$，直到生成目标结果 $\boldsymbol{X}_T$，可以用如下公式来表示每一个 $\boldsymbol{X}_t$ 和前一个 $\boldsymbol{X}_{t-1}$ 的关系：

$$\boldsymbol{X}_t = \sqrt{\beta_t}\boldsymbol{\epsilon}_t + \sqrt{1-\beta_t}\boldsymbol{X}_{t-1}$$

在这个式子中要注意的是，迭代过程的 β 不是固定的，而是从一个接近 0 数逐渐增大到接近 1，即 $\beta_1 < \beta_2 < \beta_3 < \cdots < \beta_t < 1$, 原因是扩散速度是不断加快的。

那么能不能避免多次迭代，直接从 $\boldsymbol{X}_0$ 生成 $\boldsymbol{X}_t$ 呢？答案是肯定的。

为了简化推导，引入 $\alpha = 1 - \beta$

$$\boldsymbol{X}_t = \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t + \sqrt{\alpha_t}\boldsymbol{X}_{t-1}$$

首先思考 $\boldsymbol{X}_{t-2}$ 与 $\boldsymbol{X}_t$ 的关系，由于

$$\boldsymbol{X}_t = \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t + \sqrt{\alpha_t}\boldsymbol{X}_{t-1}$$

$$\boldsymbol{X}_{t-1} = \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-1} + \sqrt{\alpha_{t-1}}\boldsymbol{X}_{t-2}$$

则，

$$\boldsymbol{X}_t = \sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t + \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{X}_{t-2}$$

由于 $\boldsymbol{\epsilon}_{t-1}$ 与 $\boldsymbol{\epsilon}_t$ 相互独立，并且都是服从标准正态分布的随机变量，所以可以取一个新的服从标准正态分布的随机变量 $\boldsymbol{\epsilon}$ 来表示 $\boldsymbol{\epsilon}_{t-1}$ 与 $\boldsymbol{\epsilon}_t$ 的和，我们有：

$$\sqrt{1-\alpha_t\alpha_{t-1}}\boldsymbol{\epsilon}\sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t \sim N(0, (1-\alpha_t\alpha_{t-1})\boldsymbol{I})$$

那么

$$\boldsymbol{X}_t = \sqrt{1-\alpha_t\alpha_{t-1}}\boldsymbol{\epsilon} + \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{X}_{t-2}$$

同样的道理，对于 $\boldsymbol{X}_{t-3}$ 和 $\boldsymbol{X}_t$，有

$$\boldsymbol{X}_t = \sqrt{1-\alpha_t\alpha_{t-1}\alpha_{t-2}}\boldsymbol{\epsilon} + \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}}\boldsymbol{X}_{t-3}$$

通过数学归纳法将这个结果推广到 $\boldsymbol{X}_{t-k}$,

$$\boldsymbol{X}_t = \sqrt{1-\prod_{i=1}^{k}\alpha_{t-i-1}}\boldsymbol{\epsilon} + \sqrt{\prod_{i=1}^{k}\alpha_{t-i-1}}\boldsymbol{X}_{t-k}$$

不妨记 $\bar{\alpha}_k = \prod_{i=1}^{k}\alpha_{t-i-1}$

$$\boldsymbol{X}_t = \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon} + \sqrt{\bar{\alpha}_k}\boldsymbol{X}_{t-k}$$

公式就变得简单了。所以只要确定了每个参数 α_i 就得到了 $\boldsymbol{X}_0$ 和 $\boldsymbol{X}_t$ 的关系，所以可以将一个原本清晰的图画变成一个模糊不清，甚至类似于噪声的图画，从而前向加噪的过程就完成了。

反过来，我们同样可以将一个随机生成的噪声（通常是正态分布生成的）进行反向降噪的处理，并且给予根据相应的约束条件，就可以得到想要的图片，这就是一些图像生成和图像修复软件，医学图像处理技术的原理。

依照上述假设，我们现在有一个依照正态分布随机生成的图片 $\boldsymbol{X}_T$，需要把它按目标还原成图片 $\boldsymbol{X}_0$，也就是根据 $\boldsymbol{X}_T$ 的分布来确定 $\boldsymbol{X}_0$ 的分布。首先分析 $\boldsymbol{X}_t$ 和 $\boldsymbol{X}_{t-1}$ 的关系，根据贝叶斯公式，有：

$$p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t,\boldsymbol{X}_0)=\frac{p(\boldsymbol{X}_t|\boldsymbol{X}_{t-1},x_0)p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_0)}{p(\boldsymbol{X}_t|\boldsymbol{X}_0)}$$

由于扩散过程是马尔科夫过程，$p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t,\boldsymbol{X}_0)=p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t)$，$p(\boldsymbol{X}_t|\boldsymbol{X}_{t-1},\boldsymbol{X}_0)=p(\boldsymbol{X}_t|\boldsymbol{X}_{t-1})$ 根据公式 $\boldsymbol{X}_t=\sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t+\sqrt{\alpha_t}\boldsymbol{X}_{t-1}$ 和公式 $\boldsymbol{X}_t=\sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}+\sqrt{\bar{\alpha}_k}\boldsymbol{X}_{t-k}$ 可知，

$$p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t)=\frac{1}{\sqrt{2\pi}\sqrt{1-\alpha_t}}e^{-\frac{(\boldsymbol{X}_t-\sqrt{\alpha_t}\boldsymbol{X}_{t-1})^2}{2(1-\alpha_t)}}$$

$$p(\boldsymbol{X}_t|\boldsymbol{X}_0)=\frac{1}{\sqrt{2\pi}\sqrt{1-\bar{\alpha}_t}}e^{-\frac{(\boldsymbol{X}_t-\sqrt{\bar{\alpha}_t}\boldsymbol{X}_0)^2}{2(1-\bar{\alpha}_t)}}$$

所以，

$$\boldsymbol{X}_{t-1}|\boldsymbol{X}_t\sim N(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{X}_t+\frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}\boldsymbol{X}_0,(\frac{\sqrt{1-\alpha_t}\sqrt{1-\bar{\alpha}_{t-1}}}{\sqrt{1-\bar{\alpha}_t}})^2\boldsymbol{I})$$

$\boldsymbol{X}_0$ 是要求的结果，但是此时出现在了 $\boldsymbol{X}_{t-1}|\boldsymbol{X}_t$ 的密度函数中，我们可以根据公式 $\boldsymbol{X}_t=\sqrt{1-\bar{\alpha}_t}\epsilon+\sqrt{\bar{\alpha}_t}\boldsymbol{X}_0$，把 $\boldsymbol{X}_0$ 带入上式，

$$\boldsymbol{X}_{t-1}|\boldsymbol{X}_t\sim N(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{X}_t+\frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}\frac{\boldsymbol{X}_t}{\sqrt{\bar{\alpha}_t}}-\frac{\sqrt{1-\bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t}}\boldsymbol{\epsilon},(\frac{\sqrt{1-\alpha_t}\sqrt{1-\bar{\alpha}_{t-1}}}{\sqrt{1-\bar{\alpha}_t}})^2\boldsymbol{I})$$

现在我们只要知道了 $\boldsymbol{\epsilon}$ 就能将分布的期望 $\boldsymbol{\mu}$ 求出来，得到整个密度函数 $p(\boldsymbol{X}_{t-1}|\boldsymbol{X}_t)$，进而将 $\boldsymbol{X}_{t-1}$ 采样出来，完成一次去噪过程。那么 $\boldsymbol{\epsilon}$ 怎么求呢？

这里我们可以训练一个神经网络模型，输入 t 时刻的图像来预测子图像相对于 $\boldsymbol{X}_0$ 原图加入的噪声 $\boldsymbol{\epsilon}$，由此 $\boldsymbol{\epsilon}$ 得到前一时刻的概率分布，再有此概率分布进行随机采样，可以得到前一时刻的一个图像，然后将次图像再次输入神经网络预测 $\boldsymbol{\epsilon}$，不断重复此过程直到得到 $\boldsymbol{X}_0$。

另外，为什么可以从一个依照正态分布随机生成的图片 $\boldsymbol{X}_T$ 开始呢？因为对于 T 时刻，可以认为此时 $\bar{\alpha}$ 趋近于 0，那么可以认为 $\boldsymbol{X}_T\approx\boldsymbol{\epsilon}$ 也就是说 $\boldsymbol{X}_T$ 近似于标准正态分布，因此可以认为任何一张标准正态图片都是 $\boldsymbol{X}_0$ 加噪来的，反过来讲，自然也就可以由 $\boldsymbol{X}_T$ 降噪生成 $\boldsymbol{X}_0$。

DDPM 在生成高质量图像和视频方面表现出色，特别是处理高分辨率、复杂纹理或结构的情况。DDPM 已经在计算机视觉、图像生成和视频生成领域得到了广泛的应用和认可，常被用于超分辨率图像生成、视频帧预测、图像合成等任务。这个模型代表了生成模型领域中的一个重要研究方向，吸引了众多研究者的关注和进一步的改进。

17.5 Python 语言实践

17.5.1 浅层生成模型

朴素贝叶斯模型

朴素贝叶斯常用于文本分类和垃圾邮件检测等任务，首先让我们来看一下如何利用 Python 建议朴素贝叶斯模型进行文本分类。

```
import numpy as np # 导入相应模块
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

texts = ["这是一个正面的文本。",  # 准备示例文本数据
         "这是一个负面的文本。",
         "这是一个中性的文本。",
         "这个文本与主题无关。",
         "这个文本包含一些关键词。",
         "朴素贝叶斯是一种强大的分类算法。",
         "文本分类是机器学习的一个重要应用领域。"]

labels = ["positive", "negative", "neutral", # 给出示例文本数据的标签
          "neutral", "neutral", "positive", "positive"]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)          # 将文本数据转换成词袋表示
X_train, X_test, y_train, y_test =           # 划分训练集和测试集
train_test_split(X, labels, test_size=0.6, random_state=40)
nb_classifier = MultinomialNB()              # 初始化朴素贝叶斯分类器
nb_classifier.fit(X_train, y_train)          # 在训练集上训练分类器
y_pred = nb_classifier.predict(X_test)       # 在测试集上进行预测
accuracy = accuracy_score(y_test, y_pred)    # 计算分类准确度
print("分类准确度: ", accuracy)
```

首先准备示例的文本数据和对应的标签，然后使用 CountVectorizer 将文本数据转换成词袋（Bag of Words）表示，接着划分训练集和测试集。然后，我们初始化一个朴素贝叶斯分类器 MultinomialNB 并在训练集上进行训练，最后在测试集上进行预测并计算分类准确度。

请注意，这只是一个简单的示例，朴素贝叶斯模型在实际应用中可以用于更复杂的文本分类任务，还可以考虑使用 TF-IDF 来表示文本特征，也可以使用更大规模的文本数据集来提高模型性能。

高斯混合模型

当使用高斯混合模型作为生成模型时，我们可以使用已拟合的 GMM 模型来生成新的数据点，模拟与训练数据类似的数据分布。如下 Python 代码演示了如何使用 GMM 生成新的数据点：

首先导入相关模块，并生成一个示例二维数据集。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

np.random.seed(0) # 创建示例数据
data1 = np.random.randn(100, 2)
data2 = 2 * np.random.randn(100, 2) + np.array([5, 5])
data3 = 1.5 * np.random.randn(100, 2) + np.array([-5, 5])
data = np.vstack((data1, data2, data3))
```

再使用 GMM 模型对数据进行拟合。接着，使用 sample 方法从已拟合的 GMM 模型中生成新的数据点，数量为 100。

```
gmm = GaussianMixture(n_components=3) # 初始化 GMM 模型，假设有3个混合成分
gmm.fit(data) # 在数据上拟合 GMM 模型
new_data = gmm.sample(100) # 生成新的数据点
```

最后将生成的数据点与原始数据一起可视化显示。

```
plt.figure(figsize=(8, 6))
plt.scatter(new_data[0][:, 0], new_data[0][:, 1], marker='x',
            s=50, label='Generated Data', c='red')
plt.scatter(data[:, 0], data[:, 1], c='blue', s=30, label='Original Data')
plt.title('Generated Data using GMM')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show() # 可视化生成的数据点
```

这个示例演示了如何使用 GMM 作为生成模型，生成与训练数据类似的新数据点。在实际应用中，还可以根据任务需求调整模型的参数，生成更多的数据样本。

隐马尔可夫模型

隐马尔可夫模型通常用于序列数据建模，如自然语言处理、语音识别、生物信息学等领域。以下是一个简单的示例 Python 代码，演示如何使用 HMM 解决评估问题。

在这个示例中，我们将使用 hmmlearn 库，它提供了用于 HMM 的工具。确保你已经安装了 hmmlearn 库。你可以使用以下命令安装它：

```
pip install hmmlearn
```

首先创建一个示例的 HMM 模型，并定义模型的相关参数，包括初始状态概率、状态转移矩阵和观测概率矩阵。

```
import numpy as np
from hmmlearn import hmm
```

```
np.random.seed(42) # 生成示例观测数据
observed_data = np.random.randint(1, 4, 100)  # 这里可替换为你的实际观测数据

model = hmm.MultinomialHMM(n_components=3, n_iter=100) # 定义HMM模型

model.startprob_ = np.array([0.6, 0.3, 0.1])   # 设置初始状态概率分布
model.transmat_ = np.array([[0.7, 0.2, 0.1],
                            [0.3, 0.6, 0.1],
                            [0.1, 0.2, 0.7]])  # 设置状态转移概率矩阵
model.emissionprob_ = np.array([[0.2, 0.3, 0.5],
                                [0.6, 0.3, 0.1],
                                [0.4, 0.2, 0.4]])  # 设置观测概率分布
```

然后可以使用 random 方法生成新的观测序列（可以替换为实际用于测试的观测序列），计算这个新观测序列的对数似然度。对数似然度越高，表示模型越能够解释观测数据。

```
model.fit(observed_data.reshape(-1, 1)) # 使用观测数据进行训练
new_observed_data = np.random.randint(1, 4, 50)  # 这里可替换为你的实际测试数据
log_likelihood = model.score(new_observed_data.reshape(-1, 1)) # 使用HMM模型进行评估
print(f"Log Likelihood of the new sequence: {log_likelihood}")
```

HMM 还可以应用于更复杂的序列数据建模和生成任务，例如语音识别、手写字识别等。根据任务需求，你可以调整模型的参数和数据来更好地应用 HMM。

LDA 主题模型

LDA 主题模型是一种用于主题建模的生成模型。如下示例 Python 代码，演示了如何使用 Gensim 库来拟合 LDA 模型并进行文本主题建模。

首先，需要确保你已经安装了 gensim 库。你可以使用以下命令安装它：

```
pip install gensim
```

下面再导入相应模块:

```
import gensim
from gensim import corpora
from gensim.models import LdaModel
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
import string
```

接下来定义一个示例的文本数据集 documents，然后对文本数据进行了预处理，包括分词、去除标点符号、去除停用词和词形还原，并使用 Gensim 创建词袋模型：

```
nltk.download('stopwords')
nltk.download('wordnet') # 下载停用词和 WordNet 词形还原器

documents = [
    "Machine learning is the future of technology.",
    "Natural language processing is a subfield of AI.",
    "Deep learning models require large amounts of data.",
    "Topic modeling is useful for text analysis.",
    "LDA stands for Latent Dirichlet Allocation.",
    "Artificial intelligence is changing the world."] # 示例文本数据

def preprocess(text): # 数据预处理
    tokens = word_tokenize(text.lower()) # 分词 # 去除标点
    tokens = [token for token in tokens if token not in string.punctuation]
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words] # 去除停用词
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens] # 词形还原
    return tokens

tokenized_documents = [preprocess(doc) for doc in documents] # 预处理文本数据

dictionary = corpora.Dictionary(tokenized_documents)
corpus = [dictionary.doc2bow(doc) for doc in tokenized_documents] # 创建词袋模型
```

最后训练了 LDA 模型来提取文本数据的主题信息，输出每个主题的关键词和主题分布。

```
lda_model = LdaModel(corpus, num_topics=2, id2word=dictionary, passes=10) # 训练模型
for topic in lda_model.print_topics(): # 输出主题分布和关键词
    print(topic)
```

块主题模型

当使用块主题模型（BTM）时，通常需要将文本数据划分成块，并在块之间建模主题分布。下面是一段示例代码，演示如何使用 Gensim 库创建块主题模型，以及如何对文本块进行主题建模。

首先定义 create_btm_model 函数，用于创建块主题模型。

```
import gensim
from gensim.models.wrappers import BtmModel

def create_btm_model(blocks, num_topics): # 创建块主题模型
    btm_model = BtmModel(corpus=blocks, num_topics=num_topics,
    id2word=gensim.corpora.Dictionary(blocks))
    return btm_model
```

然后定义数据预处理函数 preprocess，并使用它对文本数据进行了预处理，最终将文本数据划分成块并存储在 blocks 中。

```
documents = ["Machine learning is the future of technology.",
    "Natural language processing is a subfield of AI.",
    "Deep learning models require large amounts of data.",
    "Topic modeling is useful for text analysis.",
    "LDA stands for Latent Dirichlet Allocation.",
    "Artificial intelligence is changing the world."] # 示例文本数据

def preprocess(text): # 数据预处理函数，视具体情况对于预处理步骤进行指定
    tokens = word_tokenize(text.lower())  # 分词并且转换为小写
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.isalpha() and word not in stop_words
    and word not in string.punctuation] # 移除停用词和标点符号
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens] # 词干提取
    return tokens

tokenized_documents = [preprocess(doc) for doc in documents] # 将数据划分成块
blocks = tokenized_documents
```

接下来创建块主题模型 btm_model 并使用 build_topic_models 函数对每个文本块进行主题建模。

```
num_topics = 2
btm_model = create_btm_model(blocks, num_topics) # 创建块主题模型

def build_topic_models(btm_model, blocks): # 进行主题建模
    topic_models = []
    for i in range(len(blocks)):
        topics = btm_model.get_document_topics(i)
        topic_models.append(topics)
    return topic_models

topic_models = build_topic_models(btm_model, blocks) # 获取块的主题分布
```

最后输出每个块的主题分布：

```
for i, topics in enumerate(topic_models):
    print(f"Block {i + 1} - Topics: {topics}")
```

这个示例代码提供了一个基本的框架，你可以根据自己的数据和需求来扩展和优化，以进行更复杂的块主题建模和分析。

独立成分分析模型

独立成分分析（ICA）是一种用于盲源分离和信号处理的生成模型。以下是一个使用 Python 和 sklearn 库的示例代码，演示如何使用 ICA 模型进行信号分离。

首先生成三个不同的信号 s1、s2 和 s3，然后通过线性组合这些信号来模拟混合信号 X。

```
import numpy as np
from sklearn.decomposition import FastICA
import matplotlib.pyplot as plt

np.random.seed(0) # 生成混合信号
n_samples = 2000
time = np.linspace(0, 8, n_samples)

s1 = np.sin(2 * time)  # 正弦信号
s2 = np.sign(np.sin(3 * time))  # 方波信号
s3 = np.random.randn(n_samples)  # 高斯噪声

S = np.c_[s1, s2, s3]

A = np.array([[1, 1, 1], # 混合信号
              [0.5, 2, 1.0],
              [1.5, 1.0, 2.0]])

X = np.dot(S, A.T)
```

接下来，我们使用 FastICA 模型从混合信号中分离出原始信号 S_，再使用 Matplotlib 库绘制了原始信号、混合信号和分离后的信号的图形，以可视化分离效果。

```
ica = FastICA(n_components=3) # 应用 ICA 模型进行信号分离
S_ = ica.fit_transform(X)

plt.figure(figsize=(10, 5)) # 绘制原始信号和分离后的信号

plt.subplot(3, 1, 1)
plt.title('Original Signals')
plt.plot(S)

plt.subplot(3, 1, 2)
plt.title('Mixed Signals')
plt.plot(X)

plt.subplot(3, 1, 3)
plt.title('ICA Recovered Signals')
plt.plot(S_)

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

这个示例演示了如何使用 ICA 模型进行盲源分离，分离混合信号中的独立成分。你也可以根据自己的数据和需求来修改代码以适应不同的应用场景。

概率主成分分析模型

概率主成分分析（PPCA）是一种概率生成模型，用于降维和数据重建。以下是一个使用 Python 和 scikit-learn 库的示例代码，演示如何使用 PPCA 模型进行数据降维和重建。

首先生成示例数据 $\boldsymbol{X}$，然后使用 PCA 进行数据降维，将数据降维到 $n_{components}$=1 维。

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import make_multilabel_classification
import matplotlib.pyplot as plt
n_samples = 100 # 生成示例数据
n_features = 2
X, _ = make_multilabel_classification(n_samples=n_samples, n_features=n_features,
                                      n_classes=1, n_labels=1, random_state=0)

n_components = 1 # 使用 PCA 进行数据降维
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X)
```

接着，我们使用 PPCA 模型进行数据重建，得到了 $\boldsymbol{X}_{reconstructed}$。最后，我们使用 Matplotlib 库绘制了原始数据和重建数据的散点图，以可视化数据降维和重建效果。

```
X_reconstructed = np.dot(X_pca, pca.components_) # 使用 PPCA 模型进行数据重建
plt.figure(figsize=(8, 4)) # 绘制原始数据和重建数据的散点图
plt.scatter(X[:, 0], X[:, 1], label='Original Data', alpha=0.6)
plt.scatter(X_reconstructed[:, 0], X_reconstructed[:, 1],
label='Reconstructed Data', alpha=0.6)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title('PCA and PPCA Reconstruction')
plt.show()
```

17.5.2 深层生成模型

自回归网络

我们将使用 Pytorch 实现一个自回归网络生成模型，首先需要确保你已经安装了 Pytorch 库，你可以使用以下命令安装它们：

```
pip install torch
```

首先导入 Pytorch 库的相应模块，并进行数据准备：

```
import torch
import torch.nn as nn
import torch.optim as optim

text_sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # 数据准备
vocab_size = len(set(text_sequence))
X = torch.LongTensor(text_sequence[:-1]).view(1, -1, 1)
y = torch.LongTensor(text_sequence[1:]).view(1, -1)
```

接着定义一个类 AutoregressiveModel：

```
class AutoregressiveModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(AutoregressiveModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm_cell = nn.LSTMCell(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x, h, c):
        x = self.embedding(x)
        h, c = self.lstm_cell(x.view(x.size(0), -1), (h, c))
        output = self.fc(h)
        return output, h.detach(), c.detach()
```

再设置 embedding 层和隐含层的节点数，以及学习率等参数，并将 AutoregressiveModel 类实例化：

```
embedding_dim = 64; hidden_dim = 128 # 超参数设置
model = AutoregressiveModel(vocab_size, embedding_dim, hidden_dim) # 模型初始化
criterion = nn.CrossEntropyLoss() # 损失函数和优化器
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

进行模型的训练，即模型的反向传播：

```
num_epochs = 100 # 训练模型
sequence_length = X.size(1)

for epoch in range(num_epochs):
    h = torch.zeros(X.size(0), hidden_dim)
    c = torch.zeros(X.size(0), hidden_dim)

    for i in range(sequence_length):
        inputs = X[:, i]
        targets = y[:, i]
        outputs, h, c = model(inputs, h, c)
        targets = torch.clamp(targets, 0, vocab_size - 1) # 限制目标值在合理范围内
        loss = criterion(outputs, targets)
        optimizer.zero_grad()
        loss.backward(retain_graph=True)
        optimizer.step()
```

```
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
```

最后生成新的文本序列：

```
generated_sequence = [ ]  # 生成新序列
h = torch.zeros(1, hidden_dim)
c = torch.zeros(1, hidden_dim)
current_input = X[:, 0]

for i in range(sequence_length):
    outputs, h, c = model(current_input, h, c)
    _, predicted = torch.max(outputs, 1)
    predicted = torch.clamp(predicted, 0, vocab_size - 1) # 将预测值限制在合理范围内
    generated_sequence.append(predicted.item())
    current_input = predicted.view(1, -1)

print("Generated Sequence:", generated_sequence)
```

变分自动编码器

下面我们将使用 PyTorch 和 torchvision 库，演示如何实现一个简单的 VAE 模型并在 MNIST 数据集上进行训练和生成数字图像。

首先，要确保你已经安装了 PyTorch 和 torchvision 库。你可以使用以下命令安装它们：

```
pip install torch torchvision
```

然后先导入 Pytorch 和 torchvision 的相应模块，并且为了能够复现结果设置随机种子。

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
torch.manual_seed(42)
np.random.seed(42) # 设置随机种子以复现结果
```

再利用 Pytorch 中的相应功能定义一个 VAE 类，其结构包含编码器 encoder 和解码器 decoder。编码器负责将输入数据映射到潜在空间中的分布参数，包括了均值（mean）和对数方差（log variance），此处编码器由一个包含多个线性层和激活函数的神经网络组成。它将输入数据（MNIST 图像）展平为一维向量，然后通过神经网络将其映射为包含均值和对数方差的向量。这些均值和对数方差用于后续的潜在变量重参数化步骤，以便从潜在分布中采样潜在变量。解码器负责将从潜在空间中采

样的潜在变量重建为原始输入数据，此处解码器也由一个包含多个线性层和激活函数的神经网络组成。它将潜在变量映射回数据空间，生成与输入数据相似的重建数据。此处解码器的输出使用 Sigmoid 激活函数，以确保生成的图像像素值在 [0, 1] 范围内。

其中的 reparameterize 方法接收 mu（均值）和 logvar（对数方差）作为输入，并返回潜在变量 z，其计算过程包括对标准正态分布进行采样，然后使用重参数化技巧计算实现潜在变量的重参数化。forward 则方法定义了 VAE 模型的整个前向传播过程，即从输入数据到输出数据的整个流程。在前向传播中，数据首先通过编码器部分映射到潜在空间，然后从潜在空间中采样得到潜在变量，最后通过解码器部分映射回输入空间以生成数据重建。

```
class VAE(nn.Module): # 定义变分自动编码器（VAE）模型
    def __init__(self, input_dim, latent_dim):
        super(VAE, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim * 2) # 输出均值和对数方差
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, input_dim),
            nn.Sigmoid() # 输出图像像素值在 [0, 1] 范围内
        )

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x):
        x = x.view(x.size(0), -1) # 将输入图像展平
        x = self.encoder(x) # 编码器
        mu, logvar = torch.chunk(x, 2, dim=1) # 分离均值和对数方差
        z = self.reparameterize(mu, logvar)
        x_recon = self.decoder(z) # 解码器
        return x_recon, mu, logvar
```

接下来定义 train_vae 函数，用于训练 VAE 模型。它的作用是在给定训练数据集上执行 VAE 模型的训练过程，并优化模型参数，使其能够生成逼真的数据样本。

```
def train_vae(model, train_loader, optimizer, epoch): # 定义训练函数
```

```
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)

        reconstruction_loss = F.binary_cross_entropy(recon_batch, data.view(-1, 784),
        reduction='sum')
        kl_divergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        loss = reconstruction_loss + kl_divergence # 计算重建损失和 KL 散度损失
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(epoch,
            batch_idx * len(data), len(train_loader.dataset), 100. * batch_idx /
            len(train_loader), loss.item() / len(data)))

    print(f'====> Epoch: {epoch} Average loss: {train_loss /
    len(train_loader.dataset):.4f}')
```

其内部步骤主要为：

1、将模型切换到训练模式，通过 model.train() 实现。

2、使用一个循环遍历训练数据加载器中的每个批次（batch）。

3、在每个批次中，首先将优化器的梯度信息清零，通过 optimizer.zero_grad() 实现。

4、通过前向传播计算模型的输出，包括重建数据 recon_batch、潜在变量均值 mu 和对数方差 logvar。

5、计算两个损失项：

（1）reconstruction_loss：重建损失，衡量生成的图像与原始输入图像之间的差异，通常使用二进制交叉熵作为损失函数。

（2）kl_divergence：KL 散度损失，用于确保潜在变量分布接近标准正态分布。计算总损失 loss，它是重建损失和 KL 散度损失的加权和。

6、向后传播梯度，通过 loss.backward() 实现。

7、更新模型参数，通过 optimizer.step() 实现。

8、打印每个批次的训练信息，包括当前轮次、批次进度和损失。

9、最后，计算并打印当前轮次的平均损失。

再定义模型拟合完成后进行数据生成的生成函数 generate_samples：

```
def generate_samples(model, num_samples): # 定义生成函数
    model.eval()
    with torch.no_grad():
        z = torch.randn(num_samples, latent_dim).to(device)
        samples = model.decoder(z)
    return samples.view(-1, 1, 28, 28)
```

VAE 模型的主体结构搭建完成，接下来将利用定义好的 VAE 类和相关训练函数、生成函数，进行模型的实例化以及数据拟合、新数据的生成。

首先设置相关超参数，配置 GPU，并完成数据的加载与预处理：

```
batch_size = 128
epochs = 20
latent_dim = 20
log_interval = 10 # 参数设置


device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # 使用 GPU
transform = transforms.Compose([transforms.ToTensor()])
train_loader = DataLoader(datasets.MNIST('./data', train=True, download=True,
transform=transform), batch_size=batch_size, shuffle=True) # 数据加载和预处理
```

接下来进行 VAE 模型的实例化和数据的拟合，最后尝试生成新样本，并将其可视化：

```
model = VAE(input_dim=784, latent_dim=latent_dim).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # 创建 VAE 模型和优化器

for epoch in range(1, epochs + 1): # 训练 VAE 模型
    train_vae(model, train_loader, optimizer, epoch)

num_samples = 16 # 生成一些样本
generated_samples = generate_samples(model, num_samples)

fig, axes = plt.subplots(4, 4, figsize=(6, 6)) # 可视化生成的样本
for i, ax in enumerate(axes.flat):
    ax.imshow(generated_samples[i].cpu().numpy().squeeze(), cmap='gray')
    ax.axis('off')

plt.show()
```

生成对抗网络

接下来再利用 Pytorch 构建一个生成对抗网络，通过学习 MINST 数据集，生成一些示例图片。

同样首先导入 Pytorch：

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
```

接下来定义一个生成器网络 Generator：

```
class Generator(nn.Module): # 定义生成器网络
```

```
    def __init__(self, latent_dim, image_dim):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 784),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)
```

再定义一个判别器网络 Discriminator：

```
class Discriminator(nn.Module): # 定义判别器网络
    def __init__(self, image_dim):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(image_dim, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.main(x)
```

接着定义模型超参数、损失函数和优化器，并将生成器和判别器实例化：

```
latent_dim = 100 # 定义超参数
image_dim = 784  # 28x28 MNIST 图像展平为 784 维

generator = Generator(latent_dim, image_dim) # 创建生成器和判别器
discriminator = Discriminator(image_dim)

criterion = nn.BCELoss() # 定义损失函数和优化器
optimizer_g = optim.Adam(generator.parameters(), lr=0.0002)
optimizer_d = optim.Adam(discriminator.parameters(), lr=0.0002)
```

下面加载出 MNIST 数据集，将其转化为 torch 的数据结构。

```
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
mnist_dataset = torchvision.datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
dataloader = DataLoader(mnist_dataset, batch_size=64, shuffle=True, num_workers=2)
```

现在开始 GAN 模型的训练：

```
num_epochs = 50 # 训练 GAN

for epoch in range(num_epochs):
    for i, data in enumerate(dataloader, 0):
```

```
        real_images, _ = data
        batch_size = real_images.size(0)

        discriminator.zero_grad() # 训练判别器
        real_images = real_images.view(-1, image_dim)
        real_labels = torch.ones(batch_size, 1)
        fake_labels = torch.zeros(batch_size, 1)

        outputs_real = discriminator(real_images) # 判别器对真实图像的损失
        real_loss = criterion(outputs_real, real_labels)
        real_loss.backward()

        z = torch.randn(batch_size, latent_dim) # 生成器生成假图像
        fake_images = generator(z)

        outputs_fake = discriminator(fake_images.detach()) # 判别器对生成图像的损失
        fake_loss = criterion(outputs_fake, fake_labels)
        fake_loss.backward()

        d_loss = real_loss + fake_loss
        optimizer_d.step()

        generator.zero_grad() # 训练生成器
        outputs = discriminator(fake_images)
        g_loss = criterion(outputs, real_labels)
        g_loss.backward()
        optimizer_g.step()

        if i % 100 == 0:
            print(f"Epoch [{epoch}/{num_epochs}] Batch {i}/{len(dataloader)} "
                  f"D Loss: {d_loss.item():.4f} G Loss: {g_loss.item():.4f}")

torch.save(generator.state_dict(), 'generator.pth') # 保存生成器的模型参数
```

模型训练完毕，尝试生成一些图片，将其可视化：

```
z = torch.randn(16, latent_dim) # 生成一些样本
generated_images = generator(z).detach().numpy()

import matplotlib.pyplot as plt # 可视化生成的样本
fig, axes = plt.subplots(4, 4, figsize=(4, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(generated_images[i].reshape(28, 28), cmap='gray')
    ax.axis('off')
plt.show()
```

流模型

实现一个完整的流模型需要大量的代码和深度学习框架支持，而且需要考虑具体的应用场景。下面是一个简化版本的 Python 代码示例，用来演示如何创建一个基

本的 Flow 模型，以进行数据变换。这个示例使用 PyTorch 和 Torch Distributions 库：

```
import torch
import torch.nn as nn
from torch.distributions import MultivariateNormal
```

首先定义一个单层的可逆变换

```
class FlowLayer(nn.Module): # 定义一个单层的可逆变换
    def __init__(self, in_features):
        super(FlowLayer, self).__init__()
        self.mu = nn.Linear(in_features, in_features)
        self.log_sigma = nn.Linear(in_features, in_features)

    def forward(self, x):
        mu = self.mu(x)
        log_sigma = self.log_sigma(x)
        sigma = torch.exp(log_sigma)
        z = mu + sigma * torch.randn_like(x)
        return z, mu, log_sigma
```

再定义一个多层的 Normalizing Flow 模型

```
class NormalizingFlow(nn.Module): # 定义一个多层的 Normalizing Flow 模型
    def __init__(self, num_layers, in_features):
        super(NormalizingFlow, self).__init__()
        self.layers = nn.ModuleList([FlowLayer(in_features) for _ in range(num_layers
                                                    )])

    def forward(self, x):
        log_det = 0
        for layer in self.layers:
            x, mu, log_sigma = layer(x)
            log_det += torch.sum(log_sigma)
        return x, log_det
```

再创建示例训练数据并进行模型的训练

```
batch_size = 32 # 示例数据
data_dim = 2
data = MultivariateNormal(torch.zeros(data_dim),
torch.eye(data_dim)).sample((batch_size,))

num_layers = 5
flow_model = NormalizingFlow(num_layers, data_dim) # 创建 Normalizing Flow 模型

x_transformed, log_det = flow_model(data) # 训练模型或进行数据变换
```

概率扩散去噪生成模型

概率扩散去噪生成模型（DDPM）在生成高质量图像和视频方面应用最为广泛，特别是处理高分辨率、复杂纹理或结构的任务时表现更为出色，常被用于超分辨率图像生成、视频帧预测、图像合成等任务。下面是一个简化的 Python 代码框架，用于演示 DDPM 模型的基本结构：

```
import torch
import torch.nn as nn
from torch.distributions import Normal
import numpy as np
```

首先定义 DDPM 模型：

```
class DDPM(nn.Module): # 定义 DDPM 模型
    def __init__(self, data_dim, num_steps, num_hidden):
        super(DDPM, self).__init__()
        self.data_dim = data_dim
        self.num_steps = num_steps
        self.num_hidden = num_hidden

        # 定义扩散步骤的参数
        self.diffusion_params = nn.Parameter(torch.zeros(1, num_steps, data_dim))
        self.generator = nn.ModuleList([nn.Sequential(
            nn.Linear(data_dim, num_hidden),
            nn.ReLU(),
            nn.Linear(num_hidden, data_dim)
        ) for _ in range(num_steps)])

    def forward(self, x, noise):
        for t in range(self.num_steps):
            scale = torch.exp(self.diffusion_params[:, t, :])
            x = x + scale * noise[:, t, :]
            x = x + self.generator[t](x)
        return x
```

再训练 DDPM 模型：

```
def train_ddpm(data, num_steps, num_hidden, num_epochs, batch_size): # 训练 DDPM 模型
    data_dim = data.shape[1]
    model = DDPM(data_dim, num_steps, num_hidden)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(num_epochs): # 计算损失函数（通常是负对数似然）
        np.random.shuffle(data)
        for i in range(0, len(data), batch_size):
            batch = data[i:i+batch_size]
            noise = torch.randn(batch_size, num_steps, data_dim)
            generated_data = model(batch, noise)

            loss = torch.mean((generated_data - batch) ** 2)
```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}')
```

以下是一些示例数据

```
data_dim = 2 # 示例数据
num_samples = 1000
data = np.random.randn(num_samples, data_dim)
num_steps = 10
num_hidden = 128
num_epochs = 50
batch_size = 32

train_ddpm(data, num_steps, num_hidden, num_epochs, batch_size) # 训练 DDPM 模型
```

17.6 习题

1. 什么是朴素贝叶斯模型？请简要解释其基本原理。
2. 利用 scikit-learn 等机器学习库，实现一个朴素贝叶斯分类器来解决一个二分类或多分类问题。
3. 将 GMM 和 K-means 聚类算法应用于同一数据集，并比较它们的性能和适应，解释它们在不同情境下的优劣势。
4. 生成一个包含多个高斯分布的人工数据集，然后使用你实现的 GMM 模型进行拟合，并可视化原始数据和拟合结果。
5. 编写一个函数，能够根据给定的 HMM 模型参数，随机生成一个状态序列和对应的观测序列。
6. 编写一个函数，使用前向算法计算给定观测序列的概率，验证你的实现是否正确。
7. LDA 模型中的主题分布和文档分布是如何生成的？请尝试描述在 Gibbs 采样的过程中，主题分布和文档分布的更新步骤，并解释这些更新步骤背后的数学原理。
8. 使用 gensim 库中的 LdaModel，选择一个包含文档集合的语料库，尝试训练一个 LDA 模型。再通过 LdaModel 对一个测试文档进行主题推断。请编写 Python 代码，展示如何使用 LdaModel 进行主题推断，并输出推断结果。

9. 使用 Python 和深度学习框架（例如 TensorFlow 或 PyTorch），编写一个简单的自回归神经网络模型（考虑使用循环神经网络（RNN）或长短时记忆网络（LSTM）作为基础）。

10. 调整你的自回归神经网络模型的超参数，如隐藏层大小、学习率等，观察模型性能的变化，并将你的自回归神经网络与其他时间序列模型（如 ARIMA、Prophet 等）进行比较，分析它们在不同数据集上的性能。

11. 在 VAE 模型中，尝试解释 Kullback-Leibler（KL）散度损失的作用，请详细说明 KL 散度损失是如何影响模型训练的。

12. 使用 PyTorch 实现一个简单的 VAE 模型，包括编码器和解码器。选择一个适当的数据集，如 MNIST，进行模型训练。在模型训练后，编写代码生成一些样本图像，并与训练数据进行比较。

13. 使用 Python 和深度学习框架（如 TensorFlow 或 PyTorch），编写一个简单的生成对抗网络模型，实现生成器和判别器网络，并编写训练循环。使用 GAN 生成器生成一组合成图像。可以选择一个公开的图像数据集，如 MNIST 或 CIFAR-10，来进行训练和生成。

14. 了解 GAN 训练中可能出现的问题，如模式崩溃或训练不稳定。尝试应用一些技巧，如批标准化、渐变惩罚等，来改进模型的稳定性。

15. DDPM 模型是如何使用扩散过程和去噪分数匹配来生成数据的？请结合损失函数和生成器网络的结构（比如神经网络）提供一个简要的概述，并且推导这些过程背后的数学原理。

16. 使用 Python 和 sklearn 库，模拟数据并通过线性变换把模拟数据或信号分离成统计独立的非高斯的信号源的线性组合，实现一个简单的独立成分分析（ICA）模型。并且根据结果使用实际数据集（例如音频、图像等），尝试应用 ICA 模型进行信号分离。

第八部分

强化学习

第十八章　强化学习概述

强化学习（Reinforcement Learning）是一种机器学习的重要方法，涉及智能体（执行强化学习任务的实体）在与环境的交互中通过试错学习最优行为策略。在强化学习中，智能体从环境中接收状态信息，根据当前状态采取动作，环境返回奖励信号，智能体根据奖励信号调整策略，不断优化其行为，以达到获得最大累积奖励的目标。强化学习具有处理复杂问题、适应不确定性、实时决策和自主学习等优势，在游戏、机器人控制、自动驾驶、语音合成和处理方面有广泛应用。

强化学习算法可以和其他机器学习方法进行结合来提高各种算法的优势，例如与深度学习结合使用，形成同时具有较强学习能力和决策优化能力深度强化学习算法。

18.1　介绍

强化学习的学习思路和人比较类似，是在实践中学习，比如学习走路，如果摔倒了，那么我们大脑后面会给一个负面的奖励值，说明走的姿势不好。然后我们从摔倒状态中爬起来，如果后面正常走了一步，那么大脑会给一个正面的奖励值，我们会知道这是一个好的走路姿势。(用一个简单的例子说明：想想训练一条小狗，你不会告诉他该做什么，因为它听不懂。他做对了你给他奖励，做错了给惩罚。慢慢它就知道你啥意思了。强化学习也是一样。而你把这个小狗看成一个机器人，用这个相同的思想来训练的话，那就是强化学习了。) 在给出强化学习的定义之前先看强化学习的一些要素定义。

1. **智能体**：自主采取行动以完成任务的强化学习系统；指强化学习需要优化的部分，我们能精确控制。

2. **环境**：智能体的交互对象，我们不能直接控制。比如我和小明下棋，我就是智能体，小明是智能体交互的对象是环境，我可以控制我的行为但我不能控制小明的行为。智能体做的事为根据当前的状态选择采取一个什么样的动作；环境做的事为根据当前的状态与行为给出反馈值，进一步影响智能体对其下一步的动作调整。

    - 注：注意区分智能体和环境与物理界限不同。如：扫地机器人电动机和机械结构，以及它的传感硬件是环境，基于强化学习的路径规划算法被认为

是智能体。

3. **状态**：记为 S。t 时刻环境的状态 S_t 是它的环境状态集中某一个状态。状态是历史的一个函数，历史是一个状态、动作和回报的序列。表示为 $H_t = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_t, a_t, r_t)$ 是指智能体 t 时刻以前与环境的所有交互变量 s_1 a_1 r_1 … 组成的序列。本质上，状态是历史的一个函数 $S_t = (f(H_t))$。如下棋时，棋盘现在的布局可以被认为是状态 S_t 。如在状态 s_1 时智能体采取动作 a_1，此时环境给出一个反馈值和下一步的状态 s_2。

4. **行为（动作）**：记为 A。t 时刻个体采取的动作 a_t 是它的动作集中某一个动作。动作是智能体主动和环境交互的媒介，动作必须对环境起到一定的控制作用（尤其是对回报），如：打砖块时上/下动作不能改变环境，所以此动作不在强化学习的考虑范围，打砖块游戏只有左右这两个动作。

5. **回报**：记为 R。t 时刻个体在状态 S_t 采取的动作 a_t 对应的奖励 r_{t+1} 会在 $t+1$ 时刻得到。回报衡量了智能体在时间 t 上做的有多好，智能体的目标就是最大化累计回报。回报包括立即回报和长期回报。

   - 立即回报：当智能体在时间 t 做出动作 a_t 时，收到回报 R_t.
   - 长期回报：智能体与环境不断交互，会收到回报序列 R_t，R_{t+1} ,R_{t+2}…。一种通用的累计回报的方式是将这些回报值进行加权求和:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

   其中 γ 是折扣因子/衰减因子，衡量了未来时刻的回报在当前时刻的价值。在 $[0, 1]$ 之间。如果为 0，则是贪婪法，即价值只由当前延时奖励决定，如果是 1，则所有的后续状态奖励和当前奖励一视同仁。大多数时候，我们会取一个 0 到 1 之间的数字，即当前延时奖励的权重比后续奖励的权重大。

6. **策略**：记为 π, 它代表个体采取动作的依据，即个体会依据策略 π 来选择动作。最常见的策略表达方式是一个条件概率分布 $\pi(a|s)$, 这个概率分布表示了选择每个动作的概率，也就是在状态 s 时采取动作 a 的概率。即 $\pi(a|s) = p(A_t = a|S_t = s)$。

7. **值函数**：包括状态值函数与动作值函数。其是用来衡量一个智能体在某个状态有多好或者在某个状态选择某个动作有多好。某个策略 π 下选择某个状态 S 的值函数一般用 $v_\pi(S)$ 表示，定义为从 S 开始根据策略 π 选择行为能都得到的期望回报；行为值函数一般用 $Q_\pi(S, a)$ 表示，定义为在状态 S 选择行为 a 后智能体再根据策略 π 选择行为能够得到的期望回报。值函数一般是一个期望函数。虽然当前动作会给一个延时奖励 R_{t+1}, 但是光看这个延时奖励是不行的，

因为当前的延时奖励高，不代表到了 $t+1$,$t+2$,... 时刻的后续奖励也高。比如下象棋，我们可以某个动作可以吃掉对方的车，这个延时奖励是很高，但是接着后面我们输棋了。此时吃车的动作奖励值高但是价值并不高。因此我们的价值要综合考虑当前的延时奖励和后续的延时奖励。价值函数 $v_\pi(s)$ 和 $Q_\pi(S,a)$ 一般可以表示为下式，不同的算法会有对应的一些价值函数变种，但思路相同:

$$v_\pi(s) = E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots | S_t = s)$$

$$Q_\pi(s) = E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \ldots | S_t = s, A_t = a)$$

8. **状态转化模型**：表示为 $P_s^a s^{'}$，可以理解为一个概率状态机，它可以表示为一个概率模型，即在状态 s 下采取动作 a, 转到下一个状态 $s^{'}$ 的概率。

9. **探索率**：记为 ϵ，这个比率主要用在强化学习训练迭代过程中，由于我们一般会选择使当前轮迭代价值最大的动作，但是这会导致一些较好的但我们没有执行过的动作被错过。因此我们在训练选择最优动作时，会有一定的概率不选择使当前轮迭代价值最大的动作，而选择其他的动作。

以上就是强化学习的基本要素了。当然，在不同的强化学习模型中，会考虑一些其他的要素，或者不考虑上述要素的某几个，但是以上是大多数强化学习模型的基本要素。

18.2 强化学习整体结构

下面给出强化学习的定义：

定义：强化学习是智能体与环境进行交互，通过环境给出的的回报来调整自身行为，目的为使得到的累积期望回报最大。是一种以环境反馈作为输入的机器学习方法。

比如下棋，每走一步棋我的选择是通过计划——预测可能的回报——以及对特定位置和动作的期望判断来作出的。可能我走一个马被对方吃掉了一个车，立即回报是负值，但是我因为这一个马最后吃掉了对方的帅那我这一步的期望回报是大的。强化学习不拘泥于当前的回报，目标是使得累计期望回报最大。

18.2.1 强化学习的基本思路

智能体采取一个动作环境去接受这个动作并给出一个反馈值和下一步的状态。上面的智能体代表我们的算法执行智能体，我们可以操作智能体来做决策，即选择

一个合适的动作 a_t。下面的环境是智能体交互的对象，成为环境，它有自己的状态模型，我们选择了动作 a_t 后，环境的状态会变成 s_{t+1}，同时环境给出了我们采取动作 a_t 的延时奖励 r_{t+1}。然后智能体可以继续选择下一个合适的动作 a_{t+1}，然后环境的状态又会变成 s_{t+2}，又有新的奖励值 r_{t+1}，…，这就是强化学习的思路。

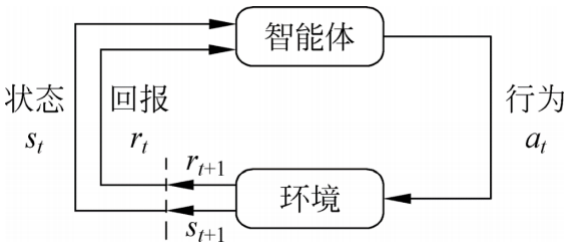


图 18.1 智能体与环境交互过程

18.2.2 强化学习和其他机器学习的关系

先来看机器学习的分类情况：

$$\text{机器学习}\begin{cases}\text{监督学习;}\\ \text{非监督学习;}\\ \text{强化学习.}\end{cases}$$

可知强化学习是和监督学习、非监督学习并列的机器学习方法。下面解释强化学习与其它两种机器学习的区别:

监督学习：带有标记数据，预测未知数据标记。是静态数据，数据之间相互独立。

非监督学习：无标记数据，挖掘数据潜在结构，如聚类。是静态数据，数据之间相互独立。

强化学习:

- 没有标记，但有延迟的回报；
- 序贯决策过程；
- 数据是智能体与环境不断交互产生，是动态数据；
- 数据之间高度相关。

总的来说强化学习和监督学习最大的区别是它是没有监督学习已经准备好的训练数据输出值的。强化学习只有奖励值，但是这个奖励值和监督学习的输出值不一样，它不是事先给出的，而是延后给出的，比如上面的例子里走路摔倒了才得到大

脑的奖励值。同时，强化学习的每一步与时间顺序前后关系紧密。而监督学习的训练数据之间一般都是独立的，没有这种前后的依赖关系。再来看看强化学习和非监督学习的区别。也还是在奖励值这个地方。非监督学习是没有输出值也没有奖励值的，它只有数据特征。同时和监督学习一样，数据之间也都是独立的，没有强化学习这样的前后依赖关系。

18.3 强化学习的环境

gym 包提供了强化学习中与 agent 交互的 environments，里面有很多的环境。gym 包安装问题说明：

要求 lstlisting 版本大于 3.5，使 pip 可以简单安装。安装的方式就是 pip install gym, 如果用 conda install gym 会有安装渠道的问题，根据提示进行安装。此时只是安装了 gym 中的 algorithms,toy text,classic control 这三种类型的环境，如果需要其它的环境可以单独安装，比如想使用 box2D 就使用命令 pip install box2D 进行安装该环境。

18.3.1 使用函数讲解

1. gym.make() 创建环境，可选择不同的环境.

2. reset() 为重新初始化函数。在强化学习算法中，智能体需要一次次地尝试，累积经验，然后从经验中学到好的动作。一次尝试我们称之为一条轨迹或一个 episode. 每次尝试都要到达终止状态. 每次尝试结束后，智能体需要从头开始，这就需要智能体具有重新初始化的功能，函数 reset() 就是这个作用，每次调用之后会有一个初始状态。

3. render() 该函数在这里扮演图像引擎的角色。一个仿真环境必不可少的两部分是物理引擎和图像引擎。物理引擎模拟环境中物体的运动规律；图像引擎来显示环境中的物体图像。其实，对于强化学习算法，该函数可以没有。但是，为了便于直观显示当前环境中物体的状态，图像引擎还是有必要的。另外，加入图像引擎可以方便我们调试代码

4. step() 该函数在仿真器中扮演物理引擎的角色。其输入是动作 a，输出是：下一步状态，立即回报，是否终止，调试项。该函数描述了智能体与环境交互的所有信息，是环境文件中最重要的函数。在该函数中，一般利用智能体的运动学模型和动力学模型计算下一步的状态和立即回报，并判断是否达到终止状态。

18.3.2 例子

官网例子

```
#加载gym包
import gym
#选择CartPole-v0的环境，也可以换成其他的环境 CartPole-v0、
    MountainCarContinuous-v0
env = gym.make('MountainCarContinuous-v0')
#进行智能体与环境的交互
env.reset()#环境初始化
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

例子中相当于做了 1000 次的尝试，每次尝试的步数不确定，根据是否结束即 done 是否为 true 来决定每条轨迹的步数。例如，在状态 s_1 下执行动作 a_1 done 的返回值为 true, 此时结束该次尝试，环境重启，进入下一次尝试，否则继续在状态 s_2 下执行动作 a_2.....。运行之后可以看到，当随机给动作时小车要么向前要么向后要么不动，如果开启上帝视角的话可以让小车向左滑动到达一个足够高的位置，然后释放使其到达终点，希望机器学到这个本事，就是目前要做的事。上述的例子其实是让大家理解环境，当前状态 s_1 有一个 a_1 环境会告诉我 s_2,r_1。这里都假定环境能给出 reward 进行后续的操作。

例子

```
import gym
#对环境进行输出
if __name__ == '__main__':
    env_names = list(gym.envs.registry.env_specs)
    print(env_names)
#选择一个环境，也可选择BipedalWalker-v3/CliffWalking-v0/CarRacing-v0
    env = gym.make('CarRacing-v0')
    print('env.action_space␣=␣', env.action_space)  #输出行为空间
    observation = env.reset()  #重置初始状态
    action_number = 0
    #智能体和环境进行交互
    for _ in range(100):
        env.render()
        action = env.action_space.sample()  # 随机选择一个动作
```

```
        next_observation, reward, done, info = env.step(action)
        action_number += 1
        print(action, next_observation, reward, done, action_number,
            info)
        #如果结束一次尝试执行下方命令进行下一次尝试
        if done:
            action_number = 0
            observation = env.reset()
    env.close()
```

上述例子所使用的 CarRacing-v0 环境中动作空间包含三个量，打方向盘、踩油门、刹车，三个量的取值范围分别为：打方向盘（-1，+1），踩油门（0，+1），刹车（0，+1）。状态空间是 96*96 的像素。使用该环境是赛车与跑道的交互，赛车随机选择一个动作如踩油门，做完该动作之后赛车到达下一个状态即下一个像素点，并得到一个奖励，环境给予智能体的奖励分为两个部分：一个是随着时间的流逝，会一直付出代价，-0.1/frame，如果按照 30frams/s 来算的话，就是-3/s；第二部分是经过赛道上铺的瓦片所获得的奖励，每经过一个瓦片都将给予智能体 1000/N 的奖励，N 为轨道上的总瓦片数，赛车碰到边界就结束。

18.4 解决强化学习问题

18.4.1 强化学习问题

强化学习的两个基本问题：

1. 预测问题。求解给定策略的状态价值函数的问题。这个问题的求解过程我们通常叫做策略评估。

2. 控制问题。求解最优的价值函数和策略。

下面介绍求解强化学习基本问题的方法，分为基于模型的方法和不基于模型的方法。其中基于模型的方法有动态规划，不基于模型的方法有蒙特卡罗法。

18.4.2 马尔可夫决策过程

首先看定义：

马尔可夫性： 某一状态蕴涵了所有相关的历史信息，当前状态已知时所有的历史信息就不再需要，即当前状态可以决策未来，则该状态具有马尔可夫性。如下棋时只要知道当前的棋盘状态就不需要知道之前的每一步是怎么下的。即

$$p(s_{t+1} \mid s_t) = p(s_{t+1} \mid s_t, s_{t-1}, ..., s_2, s_1) \tag{18.4.1}$$

马尔可夫过程： 具有马尔可夫性的随机过程。它是一个无记忆的随机过程，是一个二元组用 (S, P) 表示。

马尔可夫决策过程： 在强化学习中马尔可夫决策过程是一个五元组，用 $< S, A, P, R, \gamma >$ 表示。其中 S 表示环境的状态集合；A 表示智能体的动作集合；P 表示状态转移概率；R 表示回报函数；γ 表示折扣因子取值为 (0 1)。

引入马尔可夫决策过程有两个目的：

首先是对模型的简化。比如环境的状态转移概率模型，如果按照真实的环境转化过程看，由当前的状态 s 转化到下一个状态 $s^{'}$ 的概率既与上一个状态 s 有关，还与上上个状态，以及上上上个状态有关。这一会导致我们的环境转化模型非常复杂，复杂到难以建模。因此我们需要对强化学习的环境转化模型进行简化。简化的方法就是假设状态转化的马尔科夫性，也就是假设转化到下一个状态 $s^{'}$ 的概率仅与上一个状态 s 有关，与之前的状态无关；其次是马尔可夫决策过程是对强化学习问题的数学描述，几乎所有的强化学习问题都可以用马尔可夫决策过程描述。

18.4.3 贝尔曼方程

为求解值函数，引入贝尔曼方程。下面介绍表示值函数与后继值函数迭代关系的贝尔曼方程。在贝尔曼方程中值函数的表达式可以分解为立即回报和下一刻值函数的折扣期望。

对于状态值函数：

$$\begin{aligned} V_\pi(s) &= E(G_t|S_t = s) \\ &= E(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s) \\ &= E(R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots)|S_t = s) \\ &= E(R_{t+1} + \gamma G_{t+1}|S_t = s) \\ &= E(R_{t+1} + \gamma V_\pi(S+1)|S_t = s) \end{aligned} \tag{18.4.2}$$

同样对于动作值函数：

$$\begin{aligned} Q_\pi(s,a) &= E(G_t|S_t=s,A_t=a) \\ &= E(R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|S_t=s,A_t=a) \\ &= E(R_{t+1}+\gamma Q_\pi(S+1)|S_t=s,A_t=a) \end{aligned} \tag{18.4.3}$$

贝尔曼期望方程

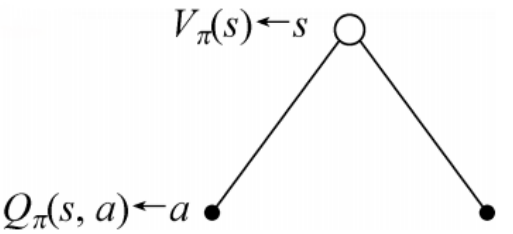


图 18.2

图18.2中空心圆圈代表状态，实心圆点代代表动作，状态指向动作表示在该状态下采取某种动作，动作指向状态表示在状态行为对下采取某动作发生了状态转变。其中在状态 s 时对应一个状态值函数，在该状态下采取动作 $a\in A$ 对应一个行为值函数，可以得到状态值函数与行为值函数的关系式：

$$V_\pi(s)=\Sigma\pi(a|s)Q_\pi(s,a) \tag{18.4.4}$$

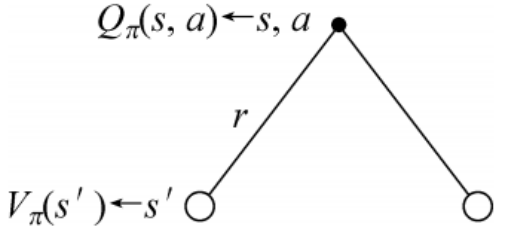


图 18.3

图18.3中状态行为对 (s,a) 对应一个行为值函数，在该状态 s 时采取动作 a，到达下一个状态 $s^{'}\in S$ 和一个立即回报 R_s^a。由此可得到行为值函数和后继状态值函

数的关系式：

$$Q_\pi(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} V_\pi(s^{'}) \tag{18.4.5}$$

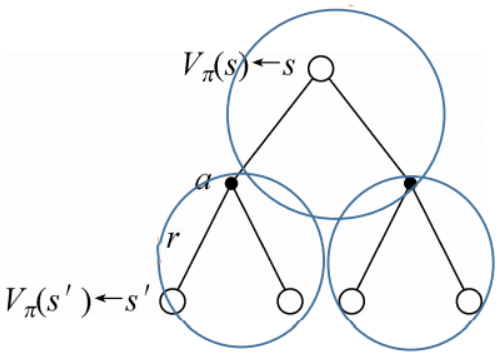


图 18.4

将两个图18.3置于图18.2之下可得到状态值函数与后继状态值函数的关系式：

$$V_\pi(s) = \Sigma \pi(a|s)(R_s^a + \gamma \Sigma P_s^a s^{'} V_\pi(s^{'})) \tag{18.4.6}$$

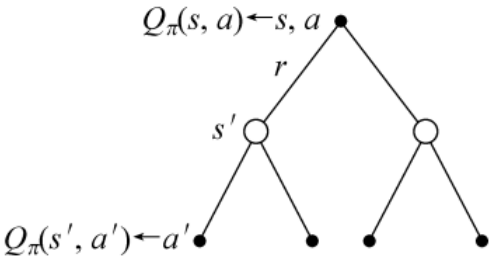


图 18.5

将两个图18.2置于图18.3之下可得到行为值函数与后继行为值函数的关系式：

$$Q_\pi(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} \Sigma \pi(a^{'}|s^{'}) Q_\pi(s^{'}, a^{'}) \tag{18.4.7}$$

下面看一个例子，来看看如何用贝尔曼期望方程来求解值函数。

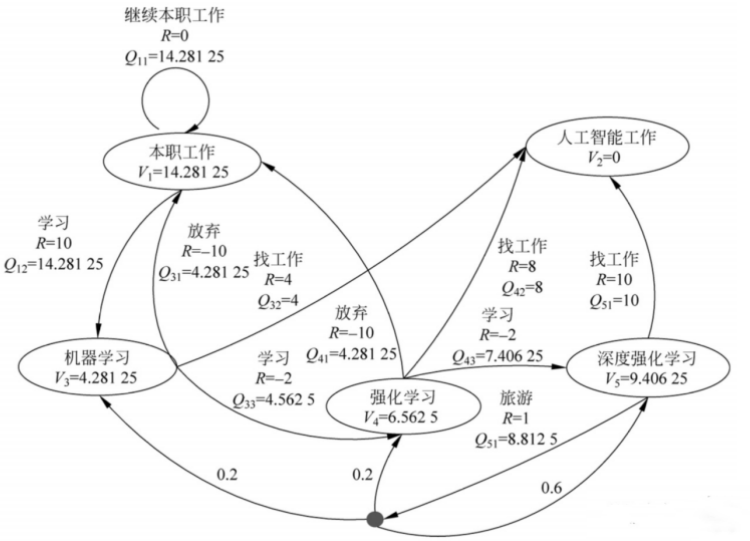


图 18.6 工作学习的 MDP

图18.6共有五个状态，分别为本职工作 s_1、人工智能工作 s_2、机器学习 s_3、强化学习 s_4 和深度强化学习 s_5；动作空间包括：继续本职工作 a_1、学习 a_2、放弃学习 a_3、找工作 a_4、旅游 a_5；其中小黑点处表示在深度强化学习状态时做了一个随机行为，有 0.2 的概率会到达机器学习这一状态即 $P_{s_5}^{a_5} s_3 = 0.2$、0.2 的概率会到达强化学习状态即 $P_{s_5}^{a_5} s_4 = 0.2$、0.6 的概率会继续留在深度强化学习这一状态，即 $P_{s_5}^{a_5} s_5 = 0.6$。除了小黑点处其余状态转移概率都是 1。假设折扣因子 $\gamma = 1$，策略为随机策略，则根据状态值函数与后继状态的值函数表达式可知：

$$
\begin{cases}
V_1 = \dfrac{1}{2}[(0+1*1*V_1)+(10+1*1*V_3)], & 172;\\
V_2 = 0, & 173;\\
V_3 = \dfrac{1}{3}[(4+1*1*V_2)+(-10+1*1*V_1)+(-2+1*1*V_4)], & 174;\\
V_4 = \dfrac{1}{3}[(-10+1*1*V_1)+(8+1*1*V_2)+(-2+1*1*V_5)], & 175;\\
V_5 = \dfrac{1}{2}[(10+1*1*V_2)+(1+(0.2*1*V_3)+(0.2*1*V_4)+(0.6*1*V_5))], & 176.
\end{cases}
$$

上面五个方程联立即可求出各个状态的值函数。

贝尔曼最优方程

解决强化学习问题意味着要寻找一个最优的策略，在该策略下让智能体在与环境交互过程中获得始终比其它策略都要多的收获，这个最优策略我们可以用 π^* 表示。一旦找到这个最优策略 π^*，那么我们就解决了这个强化学习问题。一般来说，比较难去找到一个最优策略，但是可以通过比较若干不同策略的优劣来确定一个较好的策略，也就是局部最优解。

如何比较策略的优劣呢？一般是通过对应的价值函数来比较的，也就是说，寻找较优策略可以通过寻找较优的值函数来完成。最优值函数定义为在所有策略中最大的值函数。
可以定义最优状态值函数是所有策略下产生的众多状态值函数中的最大者，即：

$$V^*(s) = \max V_\pi(s), s \in S$$

比如玩游戏时，当处在某个状态时，有两种策略可以选择，一种是激进策略 π_1（动作空间包括开火、快速前进等），一种是保守策略 π_2（动作空间包括躲闪、下蹲前进等），若在策略 π_1 下的状态值函数 $V_{\pi_1}(s)$ 大于在策略 π_2 下的状态值函数 $V_{\pi_2}(s)$，则认为 $V^*(s) = V_{\pi_1}(s)$。同理，最优的行为值函数是所有策略下产生的众多行为值函数中的最大者，即：

$$Q^*(s,a) = \max Q_\pi(s,a), s \in S$$

对比贝尔曼期望方程可以得到贝尔曼最优方程的四种形式:

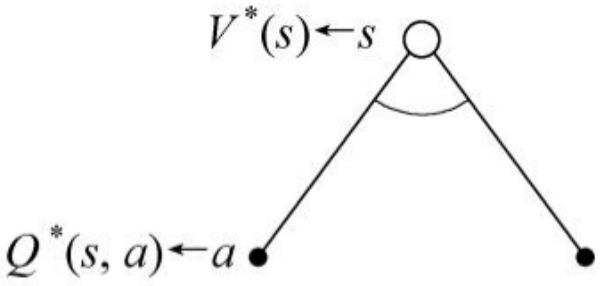


图 18.7

同样空心圆圈代表状态，实心圆点代表动作，状态指向动作表示在该状态下采取某种动作，动作指向状态表示在状态行为对下采取某动作发生了状态转变。图18.7表示在状态 s 时采取动作 $a \in A$，那么状态 s 对应的最优状态值函数与最优行为值函数的关系式为：

$$V^*(s) = \max Q^*(s,a) \tag{18.4.8}$$

这是因为智能体是可以选择动作的，在状态 s 时可以采取动作 a_1，也可以采取动作 a_2，.....，在每个动作下都有一个对应的最优行为值函数（比如采取动作 a_1，后继采取不同的策略得到不同的行为值函数，最大的行为值函数则是采取动作 a_1 最优的行为值函数），对比贝尔曼期望方程中的式18.4.1可知道上述等式成立。

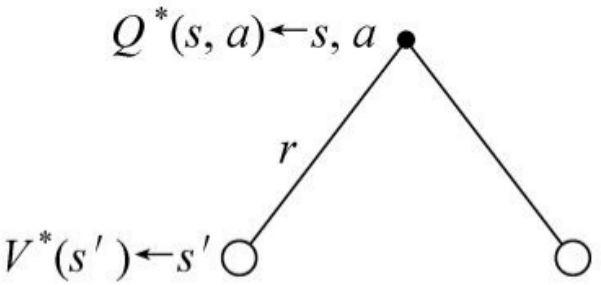


图 18.8

图18.8表示在该状态 s 时采取动作 a, 到达下一个状态 $s^{'} \in S$。对比贝尔曼期望方程式（2）可知在状态 s 时采取动作 a 的最优行为值函数与后继最优状态值函数的关系式为:

$$Q^*(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} V^*(s^{'}) \tag{18.4.9}$$

这是因为状态是环境给的，智能体不能自己选择状态，所以在贝尔曼期望方程的基础上取最优。

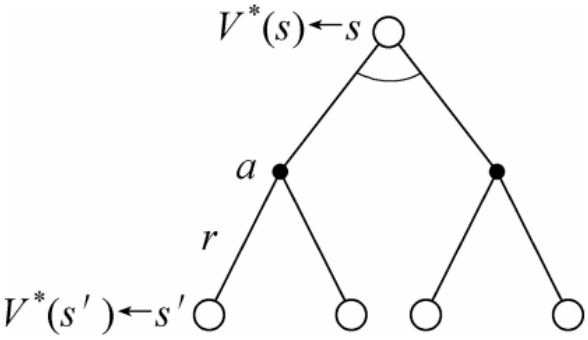


图 18.9

图18.9为两个图18.8加到图18.7下，得到最优值函数与后继最优值函数的关系式为:

$$V^*(s) = \max(R_s^a + \gamma \Sigma P_s^a s^{'} V^*(s^{'})) \tag{18.4.10}$$

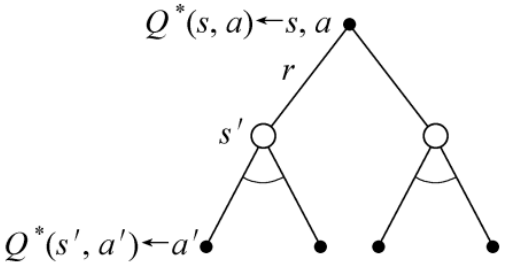


图 18.10

图18.10为两个图18.7加到图18.8下，得到最优行为值函数与后继最优行为值函数的关系式为：

$$Q^*(s,a) = R_s^a + \gamma \Sigma P_s^a s^{'} \max Q^*(s^{'}, a^{'}) \tag{18.4.11}$$

最优策略

定义：
对于任何状态 s，当且仅当遵循策略 π 的价值不小于遵循策略 $\pi^{'}$ 的价值时，则称策略 π 优于遵循策略 $\pi^{'}$，即：

$$\pi \geq \pi^{'} \; V_\pi(s) \geq V_{\pi'}(s) \; \forall s$$

对于任何 MDP，存在一个最优策略，即满足：$\pi^* \geq \pi \; \forall \pi$。每个策略对应一个值函数，最优策略对应最优值函数，找到最优值函数就找到了最优策略。
求解最优策略：

- 基于最优行为值函数 $\pi^*(a|s)=\begin{cases} 1, & a = \text{argmax}\mathrm{Q}^*(\mathrm{s},\mathrm{a}); \\ 0, & \text{其他}. \end{cases}$

- 基于最优状态值函数 $\pi^*(a|s) = \text{argmax}(\mathrm{R}_\mathrm{s}^\mathrm{a} + \gamma \, \mathrm{P}_\mathrm{s}^\mathrm{a}\mathrm{s}^{'}\mathrm{V}^*(\mathrm{s}^{'}))$

即在得到最优行为值函数时直接 argmax 求最优策略，在得到最优状态值函数时先转换为最优行为值函数再 argmax 求最优策略。下面给出基于最优行为值函数求最优策略的例子：

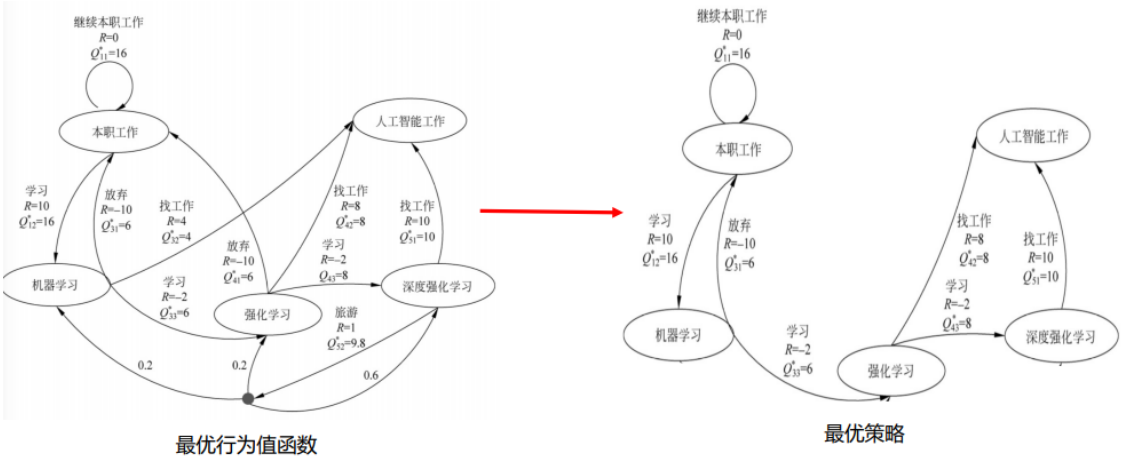


图 18.11 最优策略

若在当前状态 s 下有 m 个行为值函数相等且取值最大，则对应的行为概率均为 $1/m$。如在本职工作状态下可采取两个动作，采取继续本职工作这一动作的最优行为值函数 $Q_{11}^{*}=16$，采取学习这一动作的最优行为值函数 $Q_{12}^{*}=16$，则在本职工作状态下采取继续本职工作和学习的动作为 $1/2$，此时将这两个动作都保留。在深度强化学习状态下采取找工作这一动作的最优行为值函数 $Q_{51}^{*}=10$，采取旅游这一动作的最优行为值函数 $Q_{52}^{*}=9.8$，所以保留学习这一动作，其他状态类似，通过比较最优行为值函数留下一个最优值函数最大所对应的动作，从而得到最优策略。

18.5 动态规划

虽然 MDP 可以直接用方程组来直接求解简单的问题，但是更复杂的问题却没有办法求解，因此我们还需要寻找其他有效的求解强化学习的方法。从本节开始介绍强化学习的三个基本算法：动态规划、蒙特卡洛、时间差分。

18.5.1 动态规划简介

是运筹学分支，求解决策过程最优化的决策方法，由贝尔曼提出。基本思想：就是将待求解问题分解成若干子问题，通过解决子问题，将子问题的解进行累积，来解决原问题。

问题特征：

- **最优子结构**：复杂问题的最优解由数个子问题的最优解构成，通过寻找子问题的最优解得到复杂问题的最优解。(如最短路径：$A \to B \to C$，$A \to C$ 的最短路径由 $A \to B$ 的最短路径和 $B \to C$ 的最短路径结合得到)
- **子问题重叠**：子问题在复杂问题内重复出现，可将子问题的解存储起来，重复利用。

马尔可夫决策过程 (MDP) 具备上述两个特性：由贝尔曼方程可以看出

- 贝尔曼方程把问题递归为求解子问题
- 价值函数相当于存储了一些子问题的解

故马尔可夫决策过程可由动态规划求解。

18.5.2 动态规划解决强化学习问题

条件：需要知道 MDP 的所有元素，尤其是环境模型 (状态转移概率、回报)。动态规划是一个有模型的方法，需要知道 MDP 模型即五元组 $< S, A, P, R, \gamma >$。
步骤：使用规划方法进行策略评估和策略改进

- **策略评估 (预测)**：给定一个马尔可夫决策过程模型 MDP: $< S, A, P, R, \gamma >$ 和一个策略 π, 要求输出基于当前策略 π 的所有状态的值函数 V_π 。
- **策略改进 (控制)**：给定一个马尔可夫决策过程模型 MDP: $< S, A, P, R, \gamma >$ 和一个策略 π, 要求确定最优值函数 V^* 和最优策略 π^* 。

策略评估求解预测问题

策略评估的基本思路是从任意一个状态价值函数开始，依据给定的策略，结合贝尔曼期望方程、状态转移概率和奖励同步迭代更新状态价值函数，直至其收敛，得到该策略下最终的状态价值函数。
方法: 贝尔曼期望方程可以通过后继状态 $s^{'}$ 更新状态 s。

$$V_\pi(s) = \Sigma\pi(a|s)(R_s^a + \gamma\Sigma P_s^a s^{'} V_\pi(s^{'})) \tag{18.5.1}$$

由贝尔曼期望方程可知，初始时下一时刻的状态值函数并不知道，我们初始所有状态值函数全部为 0 。第 $k+1$ 次迭代求解时，使用第 k 次计算出来的值函数 $V_k(s^{'})$ 更新计算 $V_{k+1}(s)$，迭代公式为：

$$V_k(s) = \Sigma\pi(a|s)(R_s^a + \gamma\Sigma P_s^a s^{'} V_{k+1}(s^{'})) \tag{18.5.2}$$

该式和式18.5.1唯一的区别是由于我们的策略 π 已经给定，我们不再写出，对应加上了迭代轮数的下标。我们每一轮可以对计算得到的新的状态价值函数再次进行迭代，直至状态价值的值改变很小 (收敛)，那么我们就得出了预测问题的解，即给定策略的状态价值函数。

策略改进求解控制问题

目的：利用已经求得的值函数 V_π，找到最优策略 $\pi^{'}$。

方法：基于求得的 V_π ，针对每个状态采用贪心方法对策略进行改进。所谓贪心方法即利用贪心算法选取行为，直接将所选择的动作改变为当前最优的动作，也就是在当前状态下选取使得到达后继状态时值函数最大的动作。

$$\pi^{'} = \mathrm{greedyV}_\pi \text{ 等价于 } a = \mathrm{argmaxQ}_\pi(\mathrm{s},\mathrm{a})$$

策略评估与策略改进可以进行不同程度的组合（组合方式不同）让策略评估和策略改进交替运行，产生不同的算法：策略迭代和值迭代。

策略迭代

其中策略迭代分为两步：第一步是使用当前的策略，依据评估得到当前策略的最终价值函数，第二步是根据状态价值通过一定的方法（比如贪婪法）更新策略，接着回到第一步，一直迭代下去，最终得到收敛的策略和状态价值，具体的迭代过程如图18.12所示：

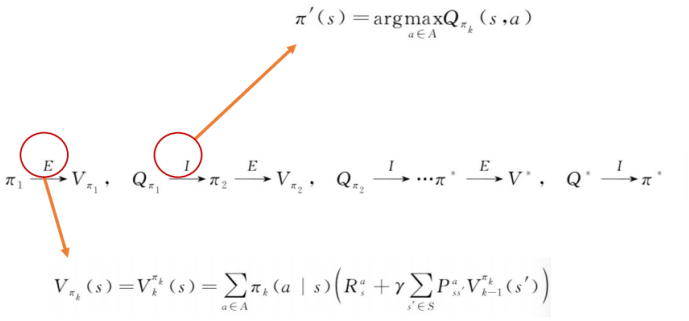


图 18.12 策略迭代过程

策略迭代的算法如表18.1所示：

输入：MDP 五元组 $M =< S, A, P, R, \gamma >$ 初始化值函数 $V(s) = 0$，初始化策略 π_1 为随机策略

For $k = 0, 1, 2, 3...do$
$\forall s \in S^{'}$: $V_{k+1}(s) = \Sigma_{a \in A}\pi(a\|s)(R_s^a + \gamma\Sigma_{s^{'} \in S}P_s^a s^{'} V_k(s^{'}))$
if $max(\|V_{k+1} - V_k\| < \theta)$
end if
end for
$\forall s \in S^{'}$:$\pi^{'}(s) = \mathrm{argmax_{a \in A}Q(s, a)}$
or $\pi^{'}(s) = \mathrm{argmax_{a \in A}(R_s^a + \gamma\ _{s \in S}P_s^a s^{'} V(s^{'}))}$
if $\forall s : \pi^{'}(s) = \pi(s)$ then
break
else
$\pi = \pi^{'}$
end if

输出: 最优策略 π

表 18.1 策略迭代的算法

一部分是策略评估，一部分是策略改进，策略评估到 V 收敛，策略改进用的是贪心算法，一次评估伴随着一次改进不断地循环至 π 收敛时（策略不发生变化）得到最优的策略 π。

例 1：网格世界寻宝

- 状态空间为 $S = 0, 1, ..., 24$，状态 8 为宝藏区，动作空间 $A =$ UP, RIGHT, DOWN,LEFT，宝藏区回报为 $r = 0$，其他位置回报为 $r = -1$ 。
- 状态转移概率 $P_a^s s^{'} = 1$，折扣因子 $\gamma = 1$。
- 初始策略为随机策略即选择上下左右这四个动作的概率均为 1/4。

目的：找到一个最优策略 (动作集合)，随便一个位置（状态）我能找到宝藏的最优路径是什么。

0	1	2	3	4
5	6	7	宝藏	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

利用贝尔曼期望方程中的式18.4.10可进行值函数的迭代求解，代码如下所示：

```
import numpy as np
from lib.envs.gridworld import GridworldEnv
import copy
import gym
#策略评估
def policy_eval(policy,env,discount_factor=1.0,threshold=0.00001):
    #初始化各状态的状态值函数
    V = np.zeros(env.nS)
    V_pre = np.zeros(env.nS)
    cishu = 0
    while True:
        value_delta = 0
        #遍历各状态
        for s in range(env.nS):
            v = 0
            #遍历各行为的概率(上,右,下,左)
            for a, action_prob in enumerate(policy[s]):
            #对于每个行为确认下个状态
#参数:prob:概率,next_state:下一个状态的索引,reward:回报,done:是否是终
    止状态
             for prob, next_state, reward, done in env.P[s][a]:
             #使用贝尔曼期望方程进行状态值函数的求解
             v+=action_prob*(reward+discount_factor*prob*V_pre[next_
                 state])
            #求出各状态和上一次求得状态的最大差值
            value_delta=max(value_delta,np.abs(v-V_pre[s]))
            V[s]=v
        V_pre=copy.deepcopy(V)
        cishu+=1
        print(V.reshape(5,5),cishu)
        #当前循环得出的各状态和上一次状态的最大差值小于阈值,则收敛停止
            运算
        if value_delta < threshold:
            break
    return np.array(V)
#策略改进
def policy_improvement(v, policy, discount_factor=1.0):
    def get_max_index(action_values):
        indexs = []
        policy_arr = np.zeros(len(action_values))
        action_max_value = np.max(action_values)
        for i in range(len(action_values)):
            action_value = action_values[i]
```

```
            if action_value == action_max_value:
                indexs.append(i)
                policy_arr[i] = 1
        return indexs, policy_arr
    #定义一个标识来证明本次是否有提升
    policy_stable = True
    #遍历各状态
    for s in range(env.nS):
        #取出当前状态下最优行为的索引值
        chosen_a = np.argmax(policy[s])
        #初始化行为数组[0,0,0,0]
        action_values = np.zeros(env.nA)
        for a in range(env.nA):
            #遍历各行为
            for prob, next_state, reward, done in env.P[s][a]:
            #根据各状态值函数求出行为值函数
            action_values[a]+=(reward+discount_factor*prob*v[next_
                state])
        best_a_arr, policy_arr = get_max_index(action_values)
       #如果求出的最大行为值函数的索引没有改变,则定义当前策略未改变,
          收敛输出
       #否则将当前的状态中将有最大行为值函数的方向置1,其余方向置0
        if chosen_a not in best_a_arr:
            policy_stable = False
        policy[s] = policy_arr
    return policy_stable, policy
#策略迭代
def policy_iteration(env):
    policy = np.ones([env.nS, env.nA])/env.nA #初始化一个随机策略
    #直至policy_stable=FALSE则结束迭代
    while True:
        #评估当前的策略,输出为各状态的当前的状态值函数
        v=policy_eval(policy, env)
        #进行策略改进
        policy_stable, policy = policy_improvement(v, policy)
        #如果当前策略没有发生改变,即已经到了最优策略,返回
        if policy_stable:
            return policy, v
if __name__ == "__main__":
    env = GridworldEnv() #对环境进行初始化
    v = policy_iteration(env)
    print("值函数:")
    print(v.reshape(5,5))
```

k 表示迭代次数，在 k 取不同值时对应的状态值函数如下所示：
$k=0$ 时随机策略下的值函数 $V_0^\pi(s)$ 为：

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

$k=1$ 时随机策略下的值函数 $V_1^\pi(s)$ 为：

-1	-1	-1	-0.75	-1
-1	-1	-0.75	0	-0.75
-1	-1	-1	-0.75	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

$k=2$ 时随机策略下的值函数 $V_2^\pi(s)$ 为：

-2	-2	-1.88	-1.44	-1.88
-2	-1.94	-1.5	0	-1.44
-2	-2	-1.88	-1.5	-1.88
-2	-2	-2	-1.94	-2
-2	-2	-2	-2	-2

$k=528$ 时值函数收敛 $V_{528}^\pi(s)$ 为：

-46.12	-40.73	-30.24	-17.62	-19.628
-47.55	-41.80	-28.38	0	-17.62
-50.70	-46.56	-38.47	-28.38	-30.24
-53.98	-51.27	-46.56	-41.80	-40.73
-55.98	-53.98	-50.70	-47.55	-46.14

针对初始策略进行策略改进，得到 π_1，对收敛的值函数 $V_{528}^\pi(s)$ 使用贪心算法进行策略改进，求取 $V_{528}^\pi(s)$ 对应的策略改进后的 π_1，则有 π_1：

→	→	→	↓	←↓
→	→	→	宝藏	←
→	→	↑→	↑	↑
↑	↑→	↑	↑	↑
↑→	→	↑	↑	↑

对 π_1 进行策略评估，经过 6 轮值函数收敛，得到 $V_6^{\pi_1}(s)$ 为：

-3	-2	-1	0	-1
-2	-1	0	0	0
-3	-2	-1	0	-1
-4	-3	-2	-1	-2
-10	-4	-3	-2	-3

对 $V_6^{\pi_1}(s)$ 使用贪心算法进行策略改进，求取 $V_6^{\pi_1}(s)$ 对应的策略改进后的 π_2，对 π_2 进行策略评估，对收敛值函数进行策略改进得到 π_3，.....，直至策略收敛，最优策略 π^* 为：

↓→	↓→	↓→	↓	←↓
→	→	→	宝藏	←
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑

值迭代

值迭代：借助贝尔曼最优方程，直接使用行为回报的最大值更新原来的值。使用贝尔曼最优方程进行更新：

$$V_{k+1}(s) = \max(\mathrm{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma\ \mathrm{P}_{\mathrm{s}}^{\mathrm{a}}\mathrm{s}^{'}\mathrm{V}_{\mathrm{k}}(\mathrm{s}^{'}))$$

迭代流程为：

$$V_1 \to V_1 \to V_2 \to ... \to V^*$$

注意：迭代收敛过程中，没有遵循任何策略。但是策略改进是隐含在值迭代过程中执行的。

输入：MDP 五元组 $M =< S, A, P, R, \gamma >$
初始化值函数 $V(s) = 0$，收敛阈值 θ
For $k = 0, 1, 2, 3...do$
$\forall s \in S^{'}$: $V^{'}(s) = \max(\mathrm{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma\, \mathrm{P}_{\mathrm{s}}^{\mathrm{a}}\mathrm{s}^{'}\mathrm{V}(\mathrm{s}^{'}))$
if $\max(\lvert\mathrm{V}^{'}(\mathrm{s}) - \mathrm{V}(\mathrm{s}^{'})\rvert < \theta)$ then
break
else $V = V^{'}$
end if
end for
$\forall s \in S^{'}$:$\pi^{'}(s) = \mathrm{argmax}_{\mathrm{a}\in\mathrm{A}}\mathrm{Q}(\mathrm{s}, \mathrm{a})$
or $\pi^{'}(s) = \mathrm{argmax}_{\mathrm{a}\in\mathrm{A}}(\mathrm{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma\, {}_{\mathrm{s}\in\mathrm{S}}\mathrm{P}_{\mathrm{s}}^{\mathrm{a}}\mathrm{s}^{'}\mathrm{V}(\mathrm{s}^{'}))$
if $\forall s : \pi^{'}(s) = \pi(s)$ then
break
else
$\pi = \pi^{'}$
end if
输出: 最优策略 π

表 18.2 值迭代的算法

例二：值迭代解决网格世界寻宝问题，仍是例一的环境，利用贝尔曼最优方程进行更新值函数，代码如下所示：

```
#值迭代
def value_iteration(env,discount_factor=1.0,threshold=0.00001):
    def get_max_index(action_values):
        indexs = []
        policy_arr = np.zeros(len(action_values))
        action_max_value = np.max(action_values)
        for i in range(len(action_values)):
            action_value = action_values[i]
            if action_value == action_max_value:
                indexs.append(i)
                policy_arr[i] = 1
        return indexs, policy_arr
    def one_step_lookahead(state, v):
        q = np.zeros(env.nA)
        for a in range(env.nA):
            for prob, next_state, reward, done in env.P[state][a]:
                q[a] += (reward + discount_factor * prob * v[next_
                    state])
```

```
        return q
    v = np.zeros(env.nS)
    while True:
        #停止条件
        delta = 0
        #遍历每个状态
        for s in range(env.nS):
            #计算当前状态的值函数
            q = one_step_lookahead(s, v)
            #找到最大值函数
            best_action_value = np.max(q)
            # 值函数更新前后求差
            delta = max(delta, np.abs(best_action_value - v[s]))
#更新当前状态的值函数，即：将最大的值函数赋值给值当前状态，用以更新当
    前状态的值函数
            v[s] = best_action_value
        #如果当前状态的值函数更新前后相差小于阈值,则说明已经收敛,结束
            循环
        if delta <= threshold:
            break
    #初始化策略
    policy = np.zeros([env.nS, env.nA])
    #遍历各状态
    for s in range(env.nS):
        #根据已经计算出的V,计算当前状态的各值函数
        q = one_step_lookahead(s,v)
        #求出当前最大值函数对应的动作索引
        #将初始策略中的对应的状态上将最大值函数方向置1,其余方向保持不
            变,仍为0
        best_a_arr, policy_arr = get_max_index(q)
        #将当前所有最优值赋值给当前状态
        policy[s] = policy_arr
    return policy,v
if __name__ == "__main__":
    env = GridworldEnv()
    v = value_iteration(env)
    print("值函数:")
    print(v)
```

k 表示迭代次数，当 k 取不同值时，基于贝尔曼最优方程得到得个状态下的值函数如下：

$k=0$ 时随机策略下的值函数 $V_0^{\pi}(s)$ 为：

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

$k=1$ 时得 $V_1^{\pi}(s)$ 为：

-1	-1	-1	-1	-1
-1	-1	-1	0	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

$k=2$ 时得 $V_2^{\pi}(s)$ 为：

-2	-2	-2	-1	-2
-2	-2	-1	0	-1
-2	-2	-2	-1	-2
-2	-2	-2	-2	-2
-2	-2	-2	-2	-2

$k=3$ 时得 $V_3^{\pi}(s)$ 为：

-3	-3	-2	-1	-2
-3	-2	-1	0	-1
-3	-3	-2	-1	-2
-3	-3	-3	-2	-3
-3	-3	-3	-3	-3

第七次之后各状态得最优行为值函数已经收敛，V_3^{π} 为：

-4	-3	-2	-1	-2
-3	-2	-1	0	-1
-4	-3	-2	-1	-2
-5	-4	-3	-2	-3
-6	-5	-4	-3	-4

对应得最优策略为：

↓→	↓→	↓→	↓	←↓
→	→	→	宝藏	←
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑
↑→	↑→	↑→	↑	←↑

18.6 蒙特卡罗

上面我们讨论了用动态规划来求解强化学习预测问题和控制问题的方法。但是由于动态规划法需要在每一次更新某一个状态的价值时，回溯到该状态的所有可能的后续状态，导致对于复杂问题计算量很大；同时很多时候，我们连环境的状态转化模型 P 都无法知道，这时动态规划法没有办法使用。这时候我们如何求解强化学习问题呢？本文要讨论的蒙特卡罗就是一种可行的方法。蒙特卡罗是在环境未知时，即智能体不知道环境的工作机制：不知道环境的状态转移概率、不知道环境奖励，智能体通过和环境交互获得的轨迹进行学习，与环境交互本质上相当于从概率分布 $P_a^s{}_{s'}$ 和 R_s^a 中进行采样，采样足够充分时，可以使用样本分布良好地刻画总体分布。

18.6.1 蒙特卡罗简介

蒙特·卡罗方法（Monte Carlo method），也称统计模拟方法，是二十世纪四十年代中期由于科学技术的发展和电子计算机的发明，而被提出的一种以概率统计理论为指导的一类非常重要的数值计算方法。是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。在强化学习问题中使用蒙特卡罗法的随机体现在采样上，也就是随机模拟模型的分布。蒙特卡罗法通过采样若干经历完整的状态序列 (*episode*) 来估计状态的真实价值。所谓的经历完整，就是这个序列必须是达到终点的。比如下棋问题分出输赢，驾车问题成功到达终点或者失败。有了很多组这样经历完整的状态序列，我们就可以来近似的估计状态价值，进而求解预测和控制问题了。

蒙特卡罗法和和动态规划比：一、它不需要依赖于模型状态转化概率; 二、完整的经历越多，学习效果越好。下面用一张图解释动态规划与蒙特卡罗的区别：

动态规划是基于贝尔曼方程进行计算当前状态的值函数，需要知道所有后继状态的值函数。

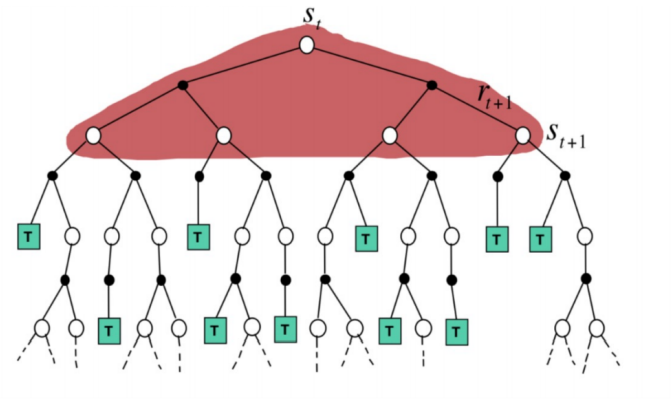


图 18.13 动态规划求值函数

当不知道所有后继状态的值函数时，蒙特卡罗最简单的思路是通过不断的采样，然后统计平均回报值来估计值函数。

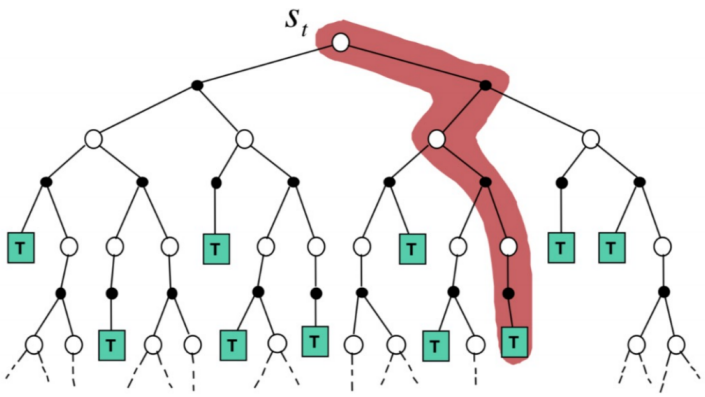


图 18.14 蒙特卡罗求值函数

18.6.2 蒙特卡罗评估

从值函数的定义来看:

$$V_\pi(s) = E(G_t|s_t = s) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{18.6.1}$$

每个状态的值函数等于所有该状态累积回报的期望。那么对于蒙特卡罗法来说，如果要求某一个状态的值函数，只需要求出所有的完整序列中该状态出现时候的收获累积回报再取平均值即可近似求解，也就是:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{18.6.2}$$

$$V_\pi(s) \approx \text{average}(\text{G}_\text{t}) \tag{18.6.3}$$

蒙特卡罗法的计算思路就是这样，但是有两个问题需要说明：

1. 同样一个状态可能在一个完整的状态序列中重复出现，那么该状态的回报该如何计算？有两种解决方法。第一种是仅把状态序列中第一次出现该状态时的收获值纳入到收获平均值的计算中；另一种是针对一个状态序列中每次出现的该状态，都计算对应的收获值并纳入到收获平均值的计算中。两种方法对应的蒙特卡罗法分别称为：首访法和每访法。第二种方法比第一种的计算量要大一些，但是在完整的经历样本序列少的场景下第一种方法更适用。

   首访法：每一条轨迹中，只记录状态 s 首次出现对应的回报.

   轨迹一：$< s_1, a_1, G_{11}, s_2, a_2, G_{12}, \ldots, s_k, a_k, G_{1k}, \ldots, s_t, a_t, G_{1t} >$

   轨迹二：$< s_3, a_1, G_{21}, s_1, a_2, G_{22}, \ldots, s_1, a_k, G_{2k}, \ldots, s_t, a_t, G_{2t} >$

   $$V(s_1) = (G_{11} + G_{22} + \ldots)/N(s_1) \tag{18.6.4}$$

   每访法：每一条轨迹，记录状态 s 出现每次出现对应的回报.

   仍以上面的两个轨迹为例：

   $$V(s_1) = (G_{11} + G_{22} + G_{2k} + \ldots)/N(s_1) \tag{18.6.5}$$

2. 在上面预测问题的求解公式里，我们有一个平均值的公式，意味着要保存所有该状态的收获值之和最后取平均。这样浪费了太多的存储空间。一个较好的方法是在迭代计算收获均值，即每次保存上一轮迭代得到的收获均值，当计算得到当前轮的收获时，即可计算当前轮收获均值。对于状态 s_t ，基于 k 次采样数据估计其值函数:

   $$\begin{aligned} V_k &= 1/k \sum_{j=1}^{k} G_j = 1/k(G_k + \sum_{j=1}^{k-1} G_j) \\ &= 1/k(G_k + (k-1)V_{k-1}) \\ &= V_{k-1} + 1/k(G_k - V_{k-1}) \end{aligned} \tag{18.6.6}$$

   k 次采样数据估计值 $=k-1$ 次采样估计值 $+$ 一个增量, 上一时刻的平均值和这一时刻的平均值建立联系。

蒙特卡罗方法值函数估计的更新公式：

$$V_{s_t} = V_{s_t} + \alpha(G_t - V_{s_t}) \tag{18.6.7}$$

在更新公式中用 $Q_\pi(s,a)$ 代替 V_s，原因是：通过 $Q_\pi(s,a)$ 求 π 更直接, 还有就是在模型未知的情况下无法通过 $V_\Pi(S)$ 求 π.

替换之后的 MC Q 值迭代公式为:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t)) \tag{18.6.8}$$

在状态 s_t 时采取行为 a_t 得到的目标值（累积回报）与当前值（上一次采样得到的行为值函数）的差值，给一个学习率去更新当前值。需注意这里的累积回报 G_t 当用其他值代替时即为 TD（sarsa Q-learning）后面会讲。

18.6.3 蒙特卡罗改进

蒙特卡罗法求解控制问题的思路和动态规划价值迭代的的思路类似。回忆下动态规划价值迭代的的思路，每轮迭代先做策略评估，计算出价值 $V_k(s)$，然后基于据一定的方法（比如贪婪法）更新当前策略 π。最后得到最优价值函数 V^* 和最优策略 π^*。

和动态规划比，蒙特卡罗法不同之处体现在三点：一是预测问题策略评估的方法不同，这个上面已经讲了。第二是蒙特卡罗法一般是优化最优动作价值函数 Q。三是动态规划一般基于贪婪法更新策略，而蒙特卡罗法一般采用 ϵ 贪婪法更新，ϵ 贪心法通过设置一个较小的 ϵ 值，使用 $1-\epsilon$ 的概率贪婪地选择目前认为是最大行为价值的行为，而用 ϵ 的概率随机的从所有 m 个可选行为中选择行为。这里相比于贪心算法增加了一个随机性，有可能那些没有使 Q 达到最大行为也会得到一个好的结果，ϵ 贪心法就是增加一个随机性有 ϵ 的概率随机选择. 所谓的 ϵ 贪心法是大概率选择使值函数最大的行为，小概率随机选择，结果证明该方法能改进任一策略，即使用该方法之后得到的策略对应的值函数比之前的值函数要大。

$$a=\begin{cases} \text{argmaxQ(s,a)}, & \text{1-}\epsilon; \\ random, & \epsilon. \end{cases} \quad \pi(a_t|s_t)=\begin{cases} \text{1-}\epsilon+\epsilon/\text{m}, & \text{a=}argmax_{a\in A}Q(s,a); \\ \epsilon/\text{m}, & a \neq argmax_{a\in A}Q(s,a). \end{cases}$$

第十九章　无模型强化学习

19.1　时序差分

上一章介绍的动态规划算法要求马尔可夫决策过程是已知的，即要求与智能体交互的环境是完全已知的。在此条件下，智能体其实并不需要和环境真正交互来采样数据，直接用动态规划算法就可以解出最优价值或策略。这就好比对于有监督学习任务，如果直接显式给出了数据的分布公式，那么也可以通过在期望层面上直接最小化模型的泛化误差来更新模型参数，并不需要采样任何数据点。但这在大部分场景下并不现实，机器学习的主要方法都是在数据分布未知的情况下针对具体的数据点来对模型做出更新的。对于大部分强化学习现实场景（例如电子游戏或者一些复杂物理环境），其马尔可夫决策过程的状态转移概率是无法写出来的，也就无法直接进行动态规划。在这种情况下，智能体只能和环境进行交互，通过采样到的数据来学习，这类学习方法统称为无模型的强化学习（model-free reinforcement learning）。

19.1.1　时序差分简介

不同于动态规划算法，无模型的强化学习算法不需要事先知道环境的奖励函数和状态转移函数，而是直接使用和环境交互的过程中采样到的数据来学习，这使得它可以被应用到一些简单的实际场景中。本章将要讲解无模型的强化学习中的两大经典算法：Sarsa 和 Q-learning，它们都是基于时序差分（temporal difference，TD）的强化学习算法。

时序差分是一种用来估计一个策略的价值函数的方法，它结合了蒙特卡洛和动态规划算法的思想。时序差分方法和蒙特卡洛的相似之处在于可以从样本数据中学习，不需要事先知道环境；和动态规划的相似之处在于根据贝尔曼方程的思想，利用后续状态的价值估计来更新当前状态的价值估计。回顾一下蒙特卡洛方法对价值函数的增量更新方式：

$$V\left(s_t\right) \leftarrow V\left(s_t\right)+\alpha\left[G_t-V\left(s_t\right)\right]$$

这里 α 表示对价值估计更新的步长。可以将 α 取为一个常数，此时更新方式不再像蒙特卡洛方法那样严格地取期望。蒙特卡洛方法必须要等整个序列结束之后才能计算得到这一次的回报 G_t ，而时序差分方法只需要当前步结束即可进行计算。具体来

说，时序差分算法用当前获得的奖励加上下一个状态的价值估计来作为在当前状态会获得的回报，即：

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_t + \gamma V(s_{t+1}) - V(s_t)\right]$$

其中 $R_t + \gamma V(s_{t+1}) - V(s_t)$ 通常被称为时序差分 (temporal difference，TD) 误差 (error)，时序差分算法将其与步长的乘积作为状态价值的更新量。可以用 $r_t + \gamma V(s_{t+1})$ 来代替 G_t 的原因是：

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_t + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_t + \gamma V_\pi(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

因此蒙特卡洛方法将上式第一行作为更新的目标，而时序差分算法将上式最后一行作为更新的目标。于是，在用策略和环境交互时，每采样一步，我们就可以用时序差分算法来更新状态价值估计。时序差分算法用到了 $V(s_{t+1})$ 的估计值，可以证明它最终收敛到策略 π 的价值函数，我们在此不对此进行展开说明。

蒙特卡罗强化学习：学习完整的采样轨迹，更新值函数和改进策略，学习效率很低。

动态规划强化学习：采用自举 (bootstapping) 的方法，用后继状态的值函数估计当前状态值函数，可以每一时间步更新一次，效率较高。

时间差分强化学习：充分结合动态规划的自举和蒙特卡罗的采样，通过学习后继状态的值函数来逼近当前状态值函数，实现对不完整轨迹的学习。要点：

- 时间差分和蒙特卡罗一样，它也需要采样，需要从轨迹中学习。
- 不需要了解模型本身，属于无模型方法。
- 可以学习不完整的轨迹。

19.1.2 三种方法的性质对比

接下来从值函数估计方式、偏差与方差、马尔可夫属性等方面对时序差分、动态规划和蒙特卡罗三种方法进行对比。

(1) 值函数估计

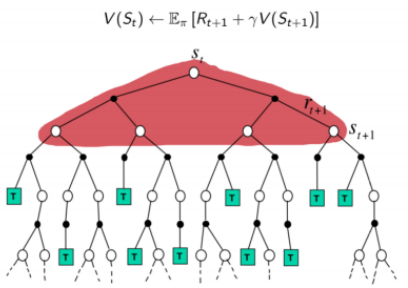


图 19.1 DP 方法

DP：没有采样。根据环境模型，获得状态 S 所有可能的转移状态 S'、转移概率、即时奖励来计算当前状态 S 的值函数。

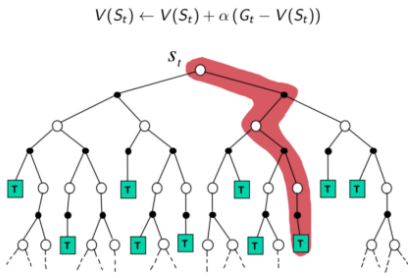


图 19.2 MC 方法

MC: 采样，获得完整轨迹。用实际回报更新当前状态值函数。

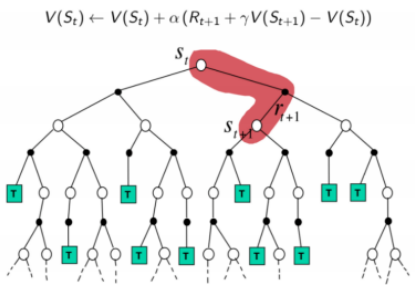


图 19.3 TD 方法

TD：采样，轨迹可不完整。用下一状态的值函数更新当前状态值函数。

(2) 偏差/方差

蒙特卡罗 (MC) 和时序差分 (TD) 均是利用样本去估计值函数，可以从统计学的角度来对比两种方法的期望和方差两个指标。

蒙特卡罗在估计值函数时，使用的是累积回报 G_t 的平均值。G_t 期望便是值函数的定义，因此蒙特卡罗方法是无偏估计。

表 19.1 时间差分、动态规划和蒙特卡洛三种方法

方法	值函数估计	是否自举	是否采样
DP	$V_\pi(S_t) = E_\pi[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$	自举	无须采样
MC	$V_\pi(S_t) \approx G_t \mid S_t = s$	不自举	采样, 完整轨迹
TD	$V_\pi(S_t) \approx R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s$	自举	采样, 不完整轨迹

蒙特卡罗在计算 G_t 值时，需要计算从当前状态到最终状态之间所有的回报，在这个过程中要经历很多随机的状态和动作，因此每次得到的随机性很大。所以尽管期望等于真值，但方差无穷大。

时序差分方法使用 $R_{t+1} + \gamma V(S_{t+1})$(也叫 TD 目标) 估计值函数，若 TD 目标采用真实值，是基于下一状态的实际价值对当前状态实际价值进行估计，则 TD 估计也是无偏估计。然而在实际中 TD 目标用的是估计值，即基于下一状态预估值函数计算当前预估值函数，因此时序差分估计属于有偏估计。跟蒙特卡罗相比，时序差分只用到了一步随机状态和动作，因此 TD 目标的随机性比蒙特卡罗方法中的 G_t 要小，其方差也比蒙特卡罗方法的方差小。

动态规划方法利用模型计算所有后继状态，借助贝尔曼方程，利用后继状态得到当前状态的真实值函数，不存在偏差和方差，三种方法的偏差和方差对比见下表。

表 19.2 三种方法的偏差和方差对比

方法	偏差	方差
DP	无偏差	无方差
MC	无偏差	高方差
TD	无偏 (真实 TD 目标) 有偏 (预估 TD 目标)	低方差

(3) 马尔可夫性

动态规划是基于模型的方法，基于现有的一个马尔可夫决策模型 MDP 的状态转移概率和回报，求解当前状态的值函数，因此该方法具有马尔可夫性。

蒙特卡罗和时序差分方法都是无模型方法，都需要通过学习采样轨迹估计当前状态值函数。所不同的是，应用时序差分 (TD) 算法时，时序差分算法试图利用现有的轨迹构建一个最大可能性的马尔可夫决策模型。即首先根据已有经验估计状态间的转移概率，同时估计一个状态的立即回报，最后计算该马尔可夫决策模型的状态值函数。蒙特卡罗算法并不试图构建马尔可夫决策模型，该算法试图最小化状态值函数与累计回报的均方误差。

通过比较可以看出，时序差分和动态规划均使用了马尔可夫决策模型问题的马尔可夫属性，在马尔可夫环境下更有效；但蒙特卡罗方法并不利用马尔可夫属性，通常在非马尔可夫环境下更有效。

表 19.3 三种方法马林科夫性对比

方法	是否使用马尔可夫属性
DP	是
MC	否
TD	是

19.1.3 Sarsa：在线策略 TD 算法

对于它的控制问题求解，和蒙特卡罗法类似，都是价值迭代，即通过价值函数的更新，来更新当前的策略，再通过新的策略，来产生新的状态和即时奖励，进而更新价值函数。一直进行下去，直到价值函数和策略都收敛。

时序差分法的控制问题，可以分为两类，一类是在线控制，即一直使用一个策略来更新价值函数和选择新的动作。而另一类是离线控制，会使用两个控制策略，一个策略用于选择新的动作，另一个策略用于更新价值函数。

我们的 Sarsa 算法，属于在线控制这一类，即一直使用一个策略来更新价值函数和选择新的动作，而这个策略是 ϵ 贪婪法，即通过设置一个较小的 ϵ 值，使用 ϵ 的概率贪婪地选择目前认为是最大行为价值的行为，而用 ϵ 的概率随机的从所有 m 个可选行为中选择行为。用公式可以表示为：

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if} \quad a^* = argmax_{a \in A} Q(s,a) \\ \epsilon/m & \text{else} \end{cases} \tag{19.1.1}$$

作为 Sarsa 算法的名字本身来说，它实际上是由 S,A,R,S,A 几个字母组成的。而 S,A,R 分别代表状态 (State)，动作 (Action)，奖励 (Reward)，这也是我们前面一直在使用的符号。这个流程体现在下图：

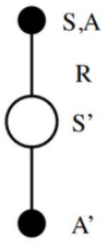


图 19.4 Sarsa 的名字来源及流程

在迭代的时候，我们首先基于 ϵ 贪婪法在当前状态 S 选择一个动作 A，这样系统会转到一个新的状态 S'，同时给我们一个即时奖励 R，在新的状态 S'，我们会基于

ϵ 贪妥法在状态 S' 选择一个动作 A', 但是注意这时候我们并不执行这个动作 A', 只是用来更新的我们的行为值函数, 行为值函数的更新公式如下:

$$Q(S,A) = Q(S,A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S,A)\right) \tag{19.1.2}$$

其中, γ 是衰减因子, α 是迭代步长。这里和蒙特卡罗法求解在线控制问题的迭代公式的区别主要是, 收获 G_t 的表达式不同, 对于时序差分, 收获 G_t 的表达式是 $R + \gamma Q\left(S', A'\right)$。

除了收获 G_t 的表达式不同, Sarsa 算法和蒙特卡罗在线控制算法基本类似。算法流程如下:

算法: Sarsa 算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a|s)=\frac{1}{|A|}$

For $k=0,1,\cdots,m$ do(针对每一条轨迹)
  初始化状态 s
  在 E 中通过 π 的 ε-贪心策略采取行为 a, 得到第一个状态行为对(s,a)
  For $t=0,1,2,3\cdots$ do(针对轨迹中的每一步)
    $r, s'=$在 E 中执行动作 a 产生的回报和转移的状态;
    基于 s', 通过 π 的 ε-贪心策略采取行为 a', 得到第二个状态行为对(s',a')
    更新(s,a)的 Q 值
    $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$;
    $s \leftarrow s', a \leftarrow a'$
  end for s 是一个终止状态
$\forall s_t \in S'$:

$$\pi(s_t) = \underset{a_t \in A}{\operatorname{argmax}} Q(s_t, a_t)$$

end for

输出: 最优策略 π

行为策略为 ε-贪心策略

目标策略均为ε-贪心策略

19.1.4 Q-learning：离线策略 TD 算法

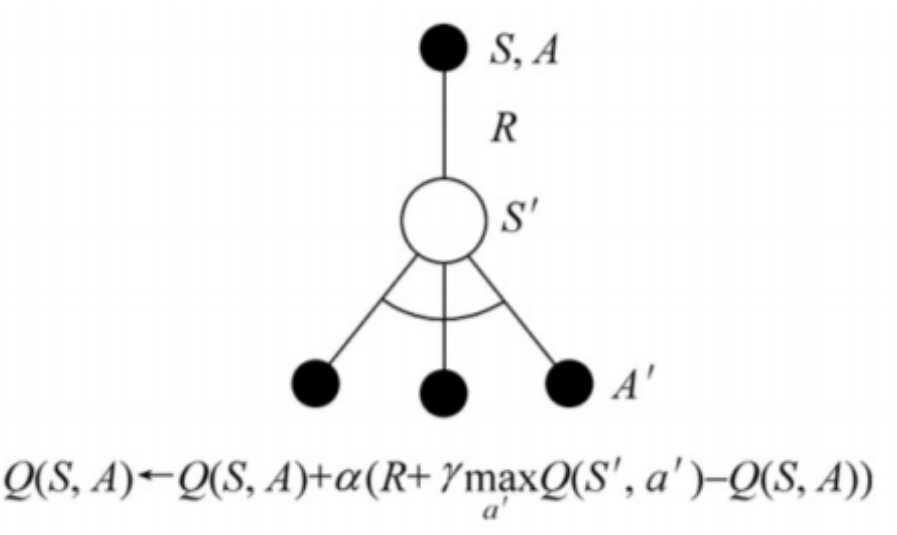


图 19.5 Q-learning 公式及图解

Q-learning 的问题求解不需要环境的状态转化模型，是不基于模型的强化学习问题求解方法。对于它的控制问题求解，和蒙特卡罗法类似，都是价值迭代，即通过价值函数的更新，来更新策略，通过策略来产生新的状态和即时奖励，进而更新价值函数。一直进行下去，直到价值函数和策略都收敛。

时序差分法的控制问题，可以分为两类，一类是在线控制，即一直使用一个策略来更新价值函数和选择新的动作，比如我们上面讲到的 Sarsa，而另一类是离线控制，会使用两个控制策略，一个策略用于选择新的动作，另一个策略用于更新价值函数。这一类的经典算法就是 Q-learning。

对于 Q-learning，我们会使用 ϵ 贪婪法来选择新的动作，这部分和 Sarsa 完全相同。但是对于价值函数的更新，Q-learning 使用的是贪婪法，而不是 Sarsa 的 ϵ 贪婪法。这一点就是 Sarsa 和 Q-learning 本质的区别。

证明: 使用贪心策略可以改进任意一个给定的策略

$\pi'(s) = \arg\max_a Q^\pi(s,a)$

$V^{\pi'}(s) \geq V^\pi(s)$, for all state s

$V^\pi(s) = Q^\pi(s,\pi(s)) \quad \leq \max_a Q^\pi(s,a) = Q^\pi(s,\pi'(s))$

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s,\pi'(s)) \\ &= E\left[r_{t+1} + V^\pi(s_{t+1}) \mid s_t = s, a_t = \pi'(s_t)\right] \\ &\leq E\left[r_{t+1} + Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s, a_t = \pi'(s_t)\right] \\ &= E\left[r_{t+1} + r_{t+2} + V^\pi(s_{t+2}) \mid \ldots\right] \end{aligned}$$

$$\leq E\left[r_{t+1}+r_{t+2}+Q^{\pi}\left(s_{t+2},\pi'\left(s_{t+2}\right)\right)\mid\ldots\right]\ldots\leq V^{\pi'}(s)$$

用待评估策略 (目标策略) 产生的下一个状态行为对的 Q 值, 来更新行为策略。值函数更新公式:

$$\pi\left(S_{t+1}\right)=\operatorname{argmax}Q\left(S_{t+1},a'\right) \tag{19.1.3}$$

有

$$R_{t+1}+\gamma Q\left(S_{t+1},A'\right)=R_{e+1}+\gamma Q\left(S_{Ht},\operatorname{argmax}Q\left(S_{H+1},a'\right)\right)=R_{t+1}+\max\gamma Q\left(S_{t+1},a'\right) \tag{19.1.4}$$

算法流程如下:

算法: Q-learning 算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a\mid s)=\frac{1}{|A|}$

For $k=0,1,\cdots,m$ do(针对每一条轨迹)
  初始化状态 s
  For $t=0,1,2,3\cdots$ do(针对轨迹中的每一步)
    在 E 中通过 π 的 ε-贪心策略采取行为 a
    $r,s'=$在 E 中执行动作 a 产生的回报和转移的状态;
    $Q(s,a)\leftarrow Q(s,a)+\alpha(r+\gamma\max_{a'}Q(s',a')-Q(s,a))$;
    $s\leftarrow s'$,
  end for s 为终止状态
end for

$$\pi^{*}(s)=\operatorname*{argmax}_{a\in A}Q(s,a)$$

输出: 最优策略 π^{*}

行为策略为 ε-贪心策略

目标策略均为贪心策略

19.2 资格迹

19.2.1 资格迹简介

蒙特卡罗方法和时序差分方法, 这两种算法之间存在一个关键的不同点: 更新当前状态的值函数时, 基于当前状态往未来看的距离不同。在蒙特卡罗算法中, 这个距离是整个轨迹的长度, 记为 N; 而在一步时序差分方法中, 这个距离是 1(单位是时间步)。那么在 1 ~ N 中, 就有很多可以选择的距离 d, 使得 1≤d≤N。通过利用这些不同距离, 构造出了新的算法类型——多步时序差分法 (也称资格迹法)。

关于多步时序差分法存在两种视角。

一种是前向视角, 向前看, 即由当前状态出发向还未访问的状态观察设计的一种算法。前向视角也叫理论视角, 它认为资格迹是连接时序差分方法和蒙特卡罗方

法的桥梁。当 n=1 步，资格迹法退化为一步时序差分法；当 n≥N 步，资格迹法发展为蒙特卡罗法；当 $1 < n < N$ 步，则产生了一系列介于时序差分和蒙特卡罗两者中间的多步时序差分方法。我们可以采用不同 n 值的线性组合来对参数进行更新，只要它们的权重值和为 1。

另一种是后向视角，向后看，即由当前状态向已经访问过的状态观察设计的一种算法。本章引入资格迹 (Eligibility Traces) 来对两种视角进行解释。资格迹是进行资格分配 (信用分配) 的方法，它是强化学习的一项基本机制。TD(A) 算法中的 x 就是对资格迹的运用。几乎所有的 TD 算法，包括 Q-learning 方法、Sarsa 方法，都可以结合资格迹来提升效率。后向视角也叫工程视角，它认为资格迹是事件发生的临时记录，如访问某个状态或采取某个行动。它为轨迹中每个状态 (或状态行为对) 附加一个属性，这个属性决定了该状态 (或状态行为对) 与当前正访问的状态 (或状态行为对) 的值函数更新量之间的关联程度，或者说影响程度。当目标误差 (TD 误差) 产生时，只有有资格的状态行为才能被分配回报或者惩罚。

虽然两种算法的表述不一样，但在本质上是统一的。前向视角告诉我们资格迹在理论层面是如何工作的；后向视角告诉我们资格迹在工程层面是如何实现的。实际中，因为前向算法计算量较大，一般都采用后向算法实现。

19.2.2 多步 TD 评估

使用蒙特卡罗算法来估计值函数:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

其中，更新目标 G_t 为累积回报:

$$G_t = r_{t+1} + \gamma_{r_{t+2}} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t+1} r_T$$

用一步 TD 估计值函数 $V(s_t)$ 时，更新目标为一步回报:

$$G_t^1 = r_{t+1} + \gamma V(s_{t+1})$$

两步 TD 的更新目标为两步回报:

$$G_t^2 = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$$

n 步 TD 的更新目标为 n 步回报

$$G_t^n = r_{t+1} + \gamma_{r_{t+2}} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$$

使用 n 步 TD 方法估计值函数时的更新公式为

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^n - V(s_t))$$

- $d=1$：一步时序差分方法，TD(1) 如 sarsa，Q-learning
- $d \geq N$（N 为轨迹长度）：蒙特卡罗算法
- $1<d<N$：多步时序差分法

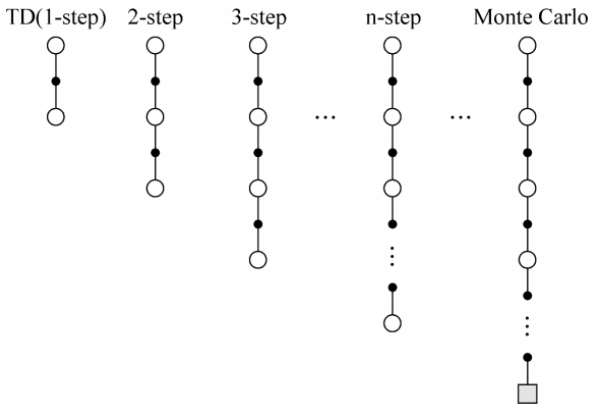


图 19.6 n 步预测估计值函数

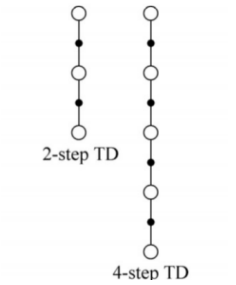
19.2.3 前向算法

问题: 既然存在 n-步时序差分方法，那么 $n=$? 时效果最好呢?

一种最简单的方法是通过平均多个不同的 n 步回报进行更新，即给每个回报赋予一定的权值，并确保这些权值的和为 1 。

比如选择 2 步 TD 和 4 步 TD 算法相结合。在每次状态更新时，2 步算法的回报占 1/2 权重，4 步算法的回报占 1/2 权重，然后求加权和。将 $\frac{1}{2}G_t^2+\frac{1}{2}G_4^t$ 作为最终回报.

实际操作: TD(λ) 直接平均了所有的 n 步回报，每个回报的权重是 $(1-\lambda)\lambda^{n-1}$ 。通过引入这个新的参数 λ, 综合考虑所有步数的回报，并且保证了最终所有权重之和为 1 。



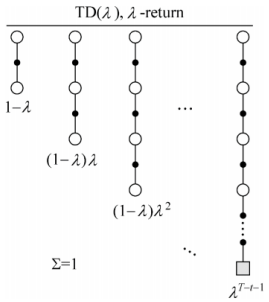


图 19.7 2-step TD 和 4-step TD 算法结合 TD(λ) 的向前视图

λ -回报:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^n$$

如果轨迹长度为 T:

$$G_i^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^n + \lambda^{T-t-1}G_t$$

值函数更新公式如下:

$$V(s_t) \leftarrow V(s_t) + \alpha\left(G_t^\lambda - V(s_t)\right)$$

如下前向视角示意图可知, 对于每个访问到的状态 s_t, 从它开始向前看所有的未来状态 s_{t+1}, s_{t+2}, ..., s_T, 并决定如何结合未来状态的回报来更新当前状态 s_t 的值函数 V (s_t) 。每更新完当前状态 s_t, 就转移至下一个状态 s_{t+1} ， 不再回头关心已更新的状态 st 前向观点通过观看未来状态的回报估计当前状态的值函数。缺点: 必须要走完整个轨迹，获得每一个状态的即时奖励，才去更新，更新较低效。

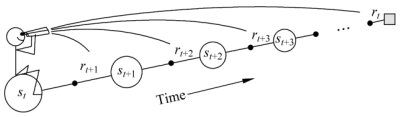


图 19.8 向前算法示意图

19.2.4 后向算法

利用 TD(λ) 的前向观点估计值函数时，每一个时间步都需要用到很多步之后的信息，这在工程上很不高效。可以说，前向视角只提供了一个非常好但却无法直接

实现的思路，这跟蒙特卡罗方法类似。实际中，我们需要一种无须等到实验结束就可以更新当前状态的值函数的更新方法。

这种增量式的更新方法需要利用多步时序差分的后向观点。它采用一种带有明确因果性的递增机制来实现值函数更新，恰恰解决了更新低效的问题。

后向视角在实现过程中，引入了一个和每个状态都相关的额外变量——资格迹。现在以一个“小狗死亡”的例子来解释资格迹的概念。如图下图所示，假设存在这样一个场景：一只小狗在连续接受了 3 次拳击和 1 次电击后死亡，那么在分析小狗的死亡原因时，到底是拳击的因素较重要还是电击的因素较重要呢？用资格迹表述就是哪个因素最有资格导致小狗死亡。

图 19.9 小狗死亡示例

实际中进行资格分配时，有两种方式：一种是频率启发式，将资格分配给最频繁的状态，如上述例子里的拳击。另一种是最近启发式：将资格分配给最近的状态，如电击。而本节所介介绍的资格迹同时结合了上述两种启发式。

在 t 时刻的状态 s 对应的资格迹, 标记为 $E_t(s)$:

$$
\begin{aligned}
&E_0(s) = 0 \\
&E_t(s) = \gamma\lambda E_{t-1}(s) + I\left(S_t = s\right)
\end{aligned}
$$

初始时刻, 每条轨迹中所有状态均有一个初始资格迹,$E_0(s) = 0$ 。下一时刻，被访问到的状态, 其资格迹为前一时刻该状态资格迹 $E_{t-1}(s)$ 乘以迹退化参数 λ 和衰减因子 γ, 然后加 1, 表示当前时刻该状态的资格迹变大。其他未被访问的状态, 其资格迹都只是在原有基础上乘以 λ 和 γ, 不用加 1, 表明它们的资格迹退化了。这种更新方式的资格迹为“累计型资格迹”。它在状态被访问的时候累计, 不被访问的时候退化, 如图所示。

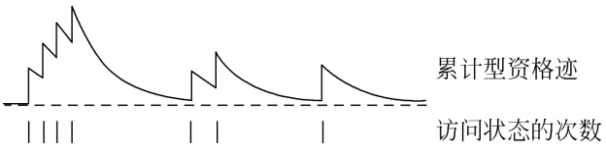


图 19.10 累计型资格迹

以“小狗死亡”的例子来对资格迹定义进行说明。假设此例子一共涉及两个状态：拳击和电击，分别用 s1、s2 表示。λ=0.9，γ=0.8。初始时，令所有状态的资格迹为 0：

$$E_0(s_1) = E_0(s_2) = 0$$

当 $t = 1$ 时, 有:

$$E_1(s_1) = \lambda\gamma E_0(s_1) + 1 = 1$$
$$E_1(s_2) = \lambda\gamma E_0(s_2) = 0$$

当 $t = 2$ 时, 有:

$$E_2(s_1) = \lambda\gamma E_1(s_1) + 1 = 1.72$$
$$E_2(s_2) = \lambda\gamma E_1(s_2) = 0$$

当 $t = 3$ 时, 有:

$$E_3(s_1) = \lambda\gamma E_2(s_1) + 1 = 2.24$$
$$E_3(s_2) = \lambda\gamma E_2(s_2) = 0$$

当 $t = 4$ 时, 有:

$$E_4(s_1) = \lambda\gamma E_3(s_1) = 1.61$$
$$E_4(s_2) = \lambda\gamma E_3(s_2) + 1 = 1$$

因此在推测小狗的致死原因时，拳击所占的比重更大一些。同时，从计算拳击和电击两个状态的资格迹的过程可见，资格迹定义同时结合了频率启发式和最近启发式。其中，$\lambda\gamma E_{t-1}(s)$ 代表频率启发式，指示函数 $(s_t = s)$ 代表最近启发式。

可用资格迹 $\mathrm{E_t(s)}$ 来分配各值函数更新的资格。可以使用资格迹来衡量当 TD 误差发生时，各状态的值函数更新会受到多大程度的影响。

TD 误差公式如下:

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$$

结合资格迹更新状态价值，则有:

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

后向算法示意图，每次当前状态获得一个误差量 t 时，这个误差量都会根据之前各状态的资格迹来分配误差，进行值函数更新。此时之前各状态值函数更新的大小与距离当前访问状态的时间步相关。假设当前状态为 s_t，TD 偏差为 δ_t，那么 s_{t-1} 处的值函数更新资格乘以 $\lambda\gamma$，状态 s_{t-2} 处的值函数更新乘以 $(\lambda\gamma)^2$，以此类推。

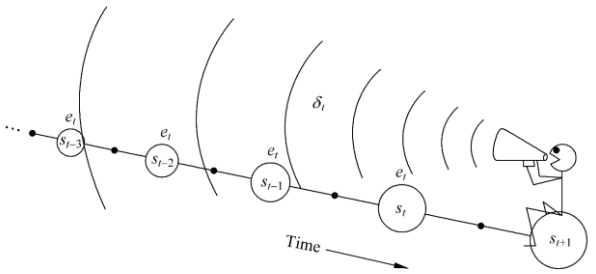


图 19.11 后向算法示意图

TD(λ) 后向算法流程如下:

算法:TD(λ) 后向算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化值函数 $V(s)=0$

For $k=0,1,\cdots,m$ do (针对每一条轨迹)

初始化资格迹 $Z(s)=0$, 对于所有的 $s\in S$

初始化状态 s

For $t=0,1,2,3\cdots$ do (针对轨迹中的每一步)

针对 s, 在 E 中通过 ϵ 贪心策略采取行为 a

$r,s'=$ 在 E 中执行动作 a 产生的回报和转移的状态;

$\delta\leftarrow R+\gamma V(s')-V(s)$

$Z(s)\leftarrow Z(s)+1$

对于所有的 $s\in S$

$V(s)\leftarrow V(s)\leftarrow\alpha\delta Z(s)$

$Z(s)\leftarrow\gamma\lambda Z(s)$

$s\leftarrow s'$

end for (s 为终止状态时, 轨迹结束)

end for

$\pi^*(s)=\underset{a\in A}{\operatorname{argmax}}R_s^a+\gamma\sum_{s'\in S}P_{ss'}^aV(s')$

19.2.5 Sarsa(λ) 方法

将多步时间差分方法和 Sarsa 算法结合, 得到一种新的方法—Sarsa(λ) 。p 区别: 不再去学习 $V_t(s)$, 而是去学习 $Q_t(s\ a)$。同样的, Sarsa(λ) 也分为前向 Sarsa(λ) 方法和后向 Sarsa(λ) 方法。

前向 Sarsa(λ) 方法

下图为 Sarsa(λ) 的前向视图，其主要思想同 TD(λ) 一样，是通过给每个回报哈子权值 $(1-\lambda)\lambda^{n-1}$ 来平均多个不同的 Q -回报进行更新。

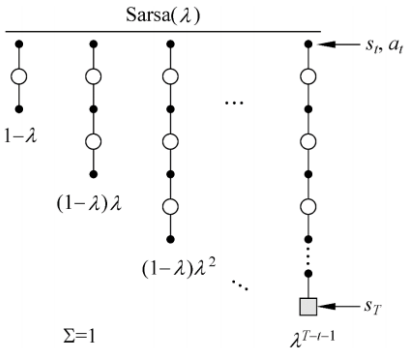


图 19.12 Sarsa(λ) 前向视图

这里的 Q -回报，相对于買积回报 G，指的是状态行为对 (s,a) 从 t 时刻开始往后所有的回报的有衰减的总和。

其中,n-step Sarsa 的 Q-回报为:

$$Q_t^{\pi} = r_{i+1} + \gamma r_{t+2} + \gamma^{n-1} r_{t+n} + \gamma'' Q\left(s_{t+\infty}, a_{t+n}\right)$$

对 Q-回报加权求和得到 Q 的 λ -回报 Q_i^{λ} :

$$Q_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} Q_i^{(n)}$$

结合 Sarsa 的更新公式，得到 Sarsa(λ) 的更新公式:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha(G_t^{\lambda} - Q(s_t, A_t))$$

同普通 TD(λ) 算法一样，前向 Sarsa(λ) 估计值函数时，需要用到很多步以后的 Q-回报，这在工程应用中很不高效，因此实际中用得比较多的还是增量更新的后向算法。

后向 Sarsa(λ) 方法

后向 Sarsa(λ) 算法通过引人资格迹，将当前行为值函数误差按比例抛给其他状态行为值函数，作为其更新的依据。不同的是，资格迹不再是 $E_1(s)$，而是 $E_e(s,a)$，即针对每一个状态行为对都有一个资格迹，公式如下:

$$\begin{aligned} E_0(s,a) &= 0 \\ E_r(s,a) &= \gamma\lambda E_{t-1}(s,a) + I\left(S_2 = s, A_i = a\right) \end{aligned}$$

其他的部分和后向 TD(λ) 一模一样。$Q_t(s,a)$ 的更新公式如下:

$$Q_{t+1}(s,a) < Q_t(s,a) + \alpha\delta_t E_t(s,a)$$

其中, $\delta_t = R_{t+1} + \gamma Q\left(s_{t+1}, a_{r+1}\right) - Q\left(s_t, a_t\right)$

$Sarsa(\lambda)$ 是在线策略算法, 也就是采样的策略和评估改进的策略是同一个策路。对于在线策略算法, 策略的更新方式有很多, 最简单的就是依据当前的行为值函数估计值采用 ϵ? 贪心算法进行更新。

$Sarsa(\lambda)$ 后向算法为单个轨迹内, 每进行一个时间步, 都会基于这个时间步的数据对行为值函数进行更新, 产生采样的策略和评估改进的策略都是 ϵ-贪心策略。

算法流程如下。

算法: Sarsa(λ) 后向算法
输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a \mid s) = \dfrac{1}{\lvert A\rvert}$
For $k = 0, 1, \cdots, m$ do (针对每一条轨迹) 初始化所有状态行为对的资格迹: $E(s,a) = 0$ 初始化状态 s 和行为 a, 得到第一个状态行为对 (s,a) For $t = 0, 1, 2, 3 \cdots$ do(针对轨迹中的每一步) $R, s' =$ 在 E 中执行动作 a 产生的回报和转移的状态; 基于 s', 通过 π 的 ε -贪心策略采取行为 a', 得到第二个状态行为对 (s', a') 求解 TD 误差 : $\delta \leftarrow R + \gamma Q\left(s', a'\right) - Q(s,a)$ 更新当前访问的状态行为对 (s,a) 的资格迹: $E(s,a) \leftarrow E(s,a) + 1$ 对于所有的状态行为对 (s,a) $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$ $E(s,a) \leftarrow \gamma\lambda E(s,a)$ $s \leftarrow s', a \leftarrow a'$ end for s 是一个终止状态 $\forall s_t \in S'$: $\pi(s) == \operatorname{argmax}_{a \in A} Q(s,a)$ end for
输出: 最优策略 π

19.2.6 Q(λ) 方法

当我们把资格迹和 Sarsa 方法结合后, 自然会想到是否资格迹也能和 Q-learning 方法结合。答案是可以, 得到的方法就是 Q(λ) 方法。

前向 Watkins's Q(λ) 方法

常规的 Q-learning 方法属于离线策略算法，即产生采样的策略 (行为策略) 和评估改进的策略 (目标策略) 不是同一个策略。也就是说：Q-learning 方法需要依据带有 ϵ-贪心行为的轨迹来学习一个贪心策略。

在进行行为值函数估计的时候，Q-learning 采取的更新公式如下：

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max Q\left(s', a'\right) - Q(s,a)\right)$$

对应的更新目标为：

$$Q_t = R + \gamma \max_{a'} Q\left(s', a'\right)$$

假设我们正在求解贪心策略在状态行为对 (s_t, a_t) 的行为值函数，前两个时间步选择的行为是贪婪行为，但是第三个时间步选择的行为是探索行为，那么 Watkins's Q(λ) 使用的有效轨迹长度，最长就到第二个时间步，从第三个时间步往后的序列都不再理会。也就是说，Watkins's Q(λ) 使用的有效轨迹长度最远到达第一个探索行为对应的时间步长。因为 Q(λ) 要学习的是贪心策略，而第三步采用的是探索行为，因此当 n 3 的 n 步回报已经和贪心策略没有联系了。

如何确定当前时间步选择的行为是不是贪婪行为呢？很简单，把当前选择的行为和当前的 $argmax_{a'}Q(s'\ a')$ 对比，如果一致就是贪婪行为，不一致就是探索行为。

因此，不同于 TD(A) 或者 Sarsa(λ)，Watkins's Q(λ) 所使用的有效轨迹长度不是整个轨迹从开始到结束，它只考虑最近的探索行为，一旦探索行为发生，则轨迹结束。

Q 若 a_{t+n} 是第一个探索行为，则轨迹以 s_{t+n} 为最后一个状态，则最长的 n 步 Q-回报为:

$$Q_t^n = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max Q\left(s_{t+n}, a\right)$$

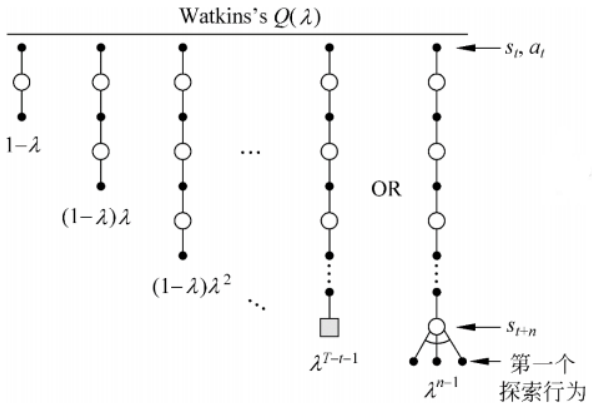


图 19.13 Watkins's Q(λ) 前向视图

可见，Watkins's Q(λ) 方法所使用的有效轨迹长度取决于第一个探索行为。如果第一个探索行为在轨迹结束前出现，有效轨迹长度最远到达此探索行为对应的时间步，否则等于整个轨迹长度。

对 Q-回报加权求和得到 Q 的 λ -回报 Q_t:

$$Q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}Q_t^{(n)}$$

Watkins's Q(λ) 更新公式为:

$$Q\left(s_t,a_t\right) \leftarrow Q\left(s_t,a_t\right) + \alpha\left(Q_t^\lambda - Q\left(s_t,a_t\right)\right)$$

后向 Watkins's Q(λ) 方法

后向 Watkins's Q(λ) 算法的算法流程如下:

后向 Watkins's $Q(\lambda)$ 算法

输入: 环境 E, 状态空间 S, 动作空间 A, 折扣回报 γ, 初始化行为值函数 $Q(s,a)=0$, $\pi(a \mid s)=\dfrac{1}{|A|}$

For $k=0,1,\cdots,m$ do (针对每一条轨迹)
- 初始化所有状态行为对的资格迹: $E(s,a)=0$
- 初始化状态 s 和行为 a, 得到第一个状态行为对 (s,a)
- For $t=0,1,2,3\cdots$ do(针对轨迹中的每一步)
  - $R,s'=$ 在 E 中执行动作 a 产生的回报和转移的状态;
  - 基于 s', 通过 π 的 ε -贪心策略采取行为 a', 得到第二个状态行为对 (s',a')
  - $a^* \leftarrow \arg\max_{b\in A} Q(s,b)$
  - $\delta^\star - R+\gamma Q\left(s',a^*\right)-Q(s,a)$
  - $E(s,a) \leftarrow E(s,a)+1$
  - 对于所有的状态行为对 (s,a)
    - $Q(s,a) \leftarrow Q(s,a)+\alpha\delta E(s,a)$
    - 如果 $a'=a^*$, 则有 $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    - 否则 $E(s,a) \leftarrow 0$
  - $s^\star s', a \leftarrow a'$
- end for s 是一个终止状态
- end for

$\forall s_t \in S' : \pi^*(s) = \arg\max_{a\in A} Q(s,a)$

输出: 最优策略 $\pi*$

后向 Watkins's Q(λ) 的资格迹如何更新呢?

对于所有状态行为对 $(s_t,\ a_t)$ 的资格迹，更新分两部分：首先，如果当前选择行为 a_t 是贪妥行为，资格迹乘以系数 $\gamma\lambda$, 否则资格迹变成 0；其次，对于当前正在访问的 $(s_t,\ a_t)$, 其资格迹单独加 1。

资格迹的更新公式如下：

$$E_t(s,a)=I_{ss_t}\cdot I_{aa_t}+\begin{cases}\gamma\lambda E_{t-1}(s,a) & \text{if } Q_{t-1}\left(s_t,a_t\right)=\max\limits_a Q_{t-1}\left(s_t,a\right)\\ 0 & \text{else}\end{cases}\tag{19.2.1}$$

其中，I_{ss_t} 和 I_{aa_t} 均为指示函数。I_{ss_t} 表示当 $s=s_t$ 时，其值为 1。

$Q(s,a)$ 更新公式如下:

$$Q_{t+1}(s,a)=Q_t(s,a)+\alpha\delta_t E_t(s,a)$$

δt 计算公式如下：

$$\delta_t=r_{t+1}+\gamma\max Q_t\left(s_{t+1},a'\right)-Q_t\left(s_t,a_t\right)$$

19.3 值函数逼近

前面已经介绍了强化学习的基本方法：基于动态规划的方法, 基于蒙特卡罗的方法和基于时序差分的方法。对于模型已知的系统，可以利用动态规划的方法得到值函数；对于模型未知的系统，可以利用蒙特卡罗的方法或时序差分的方法得到值函数。应注意到使用这些方法的一个基本前提条件：状态空间和动作空间是离散的, 而且状态空间和动作空间不能太大。具体来说，这里的值函数其实是一个表格。值函数的迭代更新实际上就是这张表的迭代更新。因此, 之前所给的强化学习算法又称为表格型强化学习。对于状态值函数，其表格的维数为状态的个数 $|\mathcal{S}|$ ，其中 $\mathcal{S}$ 为状态空间。若状态空间的维数很大，或者状态空间为连续空间，此时值函数无法用一张表格来表示。这时, 我们需要利用函数逼近的方法表示值函数。我们首先介绍了值函数逼近的思想。

在引入近似值函数后, 强化学习中不管是预测问题还是控制问题, 就转变成近似函数的设计以及求解近似函数这两个问题了。函数逼近方法可以分为参数逼近和非参数逼近。参数近似方法又可分为基于参数的线性函数近似和非线性近似两类，此时值函数可以由一组参数来近似表示，则值函数的更新也就等价于参数的更新。本节主要介绍参数近似方法，其中，通过建立目标函数，参数可使用梯度下降的办法进行训练求解。本节也就参数更新方法及常用的参数近似方法详细展开。

19.3.1 值函数逼近的思想

前面提到的基本方法中价值函数的更新是基于表格的迭代更新 (tabular solution), 这意味着每一个状态对应一个 $V(s)$ 或者每一个状态-行为对对应一个 $Q(s,a)$ 。而在现实中，状态空间一般都较大，就会产生 “维数灾难” 的问题。尤其是当状态空间是连续的时候，会有无穷的状态空间。比如，西洋双陆棋（Backgammon）所需要的状态空间是 10^{20}, 围棋 AlphaGo 有 10^{170} 个状态空间, 机器人控制以及无人机控制需要的是一个连续状态空间。对于类似的大规模问题, 按查表的方式更新值函数需要太多的内存来存储, 并且对于连续的状态空间无法用查表的方式更新值函数，而且有时候针对每一个状态学习得到值函数也是一个很慢的过程。此时，我们无法用之前基于表格的方法为每个状态或者状态动作对都求得一个值函数的特定值，因此需要有一种方法能够适应无限的状态集。值函数逼近可以将强化学习应用到这类大规模的问题中, 进而进行预测和控制。

值函数逼近即建立一个函数 $\hat{V}$, 这个函数由参数 w 描述, 它直接接受表示状态特征的连续变量 s 作为输入, 通过计算得到一个状态值函数, 通过调整参数 w 的取值, 使其符合基于某一策略 π 的最终状态值, 那么这个函数就是状态值函数 $V^{\pi}(s)$ 的近似表示，

$$\hat{V}(s;w) \approx V^{\pi}(s).$$

类似的，如果由参数 w 构成的函数 $\hat{Q}$ 同时接受状态变量 s 和行为变量 a，然后输出一个行为值函数，通过调整参数 w 的取值, 使其符合基于某一策略 π 的最终行为值函数, 那么这个函数就是行为值函数 $Q^{\pi}(s,a)$ 的近似表示，

$$\hat{Q}(s,a;w) \approx Q^{\pi}(s,a).$$

此外，如果由参数构成的函数仅接受状态变量 s 作为输入，输出针对行为空间中的每一个离散行为值 $Q(s,a_j;w), j=1,\cdots,|\mathcal{A}|$, 这给出另一种行为值函数的近似表示。在上面公式中, 描述状态的 s 不再是一个字符串或者一个索引, 而是由一系列的数据组成的向量, 构成向量的每一项称为状态的一个特征, 该项的数据值称为特征值; 参数 w 需要通过求解确定，w 通常是一个向量或矩阵等。构建了值函数的逼近形式后，强化学习中的预测和控制问题就转变为求解近似值中的参数 w 了。w 可以通过建立目标函数，然后使用梯度下降的方式进行求解。

注意到值函数逼近方法除节省存储空间之外，还能把从已知状态学到的函数通用化推广至那些未碰到的状态中 — 泛化性，即，假设离某个状态非常接近的另一状态，比如两状态之间的物理位置只相隔一毫米，则不必要单独存储其值函数。

概括来说，表格型强化学习和函数逼近方法的强化学习，值函数更新时的异同点可归结如下。(1) 表格型强化学习在更新值函数时, 只有当前状态 s_t 处的值函数改变, 其他地方的值函数不改变。(2) 值函数逼近方法更新值函数时, 更新的是参数 w, 而估计的值函数为 $\hat{V}(s;w)$, 所以当参数 w 发生改变, 任意状态处的值函数都会发生

改变。

19.3.2 目标函数及梯度下降

我们现在对值函数逼近思想已有了解, 即把用表格显式精确存储的值函数用近似函数来近似表示。也就是把原来的问题转移化为怎么使得近似变得更加准确的问题。接下来跟普遍机器学习的路数一样, 我们要定义一个目标函数, 并对这个目标函数进行优化迭代学习直到收敛, 求得最优解。但需要注意强化学习里只有即时奖励, 没有监督数据。所以我们要找到能替代 $V^{\pi}(s)$ 的目标值, 以便使用监督学习的算法学习到近似函数的参数。

首先回顾一下表格型强化学习值函数更新的公式，以更好理解值函数逼近下的更新公式。

蒙特卡罗方法, 值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(G_t - Q(s,a)\right).$$

TD(0) 方法值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma Q\left(s',a'\right) - Q(s,a)\right].$$

TD(λ) 方法值函数更新公式为

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[G_t^{\lambda} - Q(s,a)\right].$$

从上式值函数的更新过程可以看出，无论是蒙特卡罗方法还是时序差分方法, 值函数更新过程都是向着目标值函数靠近，这个目标值在蒙特卡罗方法中是 G_t，在时序差分方法中是 $r + \gamma Q\left(s',a'\right)$, 在 TD($\lambda$) 中是 G_t^{λ}。

现把上式中的值函数换成近似函数，则相应的更新公式就变成了基于近似值函数的值更新方法。比如，TD(0) 方法值函数逼近下的更新公式为

$$\hat{Q}(s,a;w) \leftarrow \hat{Q}(s,a;w) + \alpha \left[r + \gamma \hat{Q}\left(s',a';w\right) - \hat{Q}(s,a;w)\right]$$

假设我们找到了参数 w 使得逼近值函数最终收敛不再更新，则意味着对任何状态或状态行为对，有形式:

$$\hat{Q}(s,a;w) = r + \gamma \hat{Q}\left(s',a';w\right).$$

事实上, 很难找到参数 w 使得上式完全成立。同时由于算法是基于采样数据的, 即使上式对于采样得到的状态成立, 也很难对所有可能的状态成立。为衡量在采样产生的 T 个状态转换上近似值函数的收敛情况, 定义目标函数 J_w 为:

$$J(w) = \frac{1}{2T}\sum_{t=1}^{T}\left[\left(R_t + \gamma \hat{Q}\left(s_t',a_t';w\right)\right) - \hat{Q}\left(s_t,a_t;w\right)\right]^2 \tag{8.6}$$

公式 (8.6) 中 T 为采样得到的状态转换的总数。若近似值函数 $\hat{Q}(s,a;w)$ 收敛，则意味着 $J(w)$ 逐渐减小。如果 $T=1$, 则通常称为损失 (loss)。定义损失 $\text{loss}(w)$ 为:

$$\text{loss}(w)=\frac{1}{2}\left[\left(R+\gamma\hat{Q}\left(s',a';w\right)\right)-\hat{Q}\left(s,a;w\right)\right]^2. \tag{8.7}$$

对于蒙特卡罗方法, 使用 G_t 代替目标值，目标函数为:

$$J(w)=\frac{1}{2T}\sum_{t=1}^{T}\left[G_t-\hat{V}\left(s_t;w\right)\right]^2,$$

$$J(w)=\frac{1}{2T}\sum_{t=1}^{T}\left[G_t-\hat{Q}\left(s_t,a_t;w\right)\right]^2.$$

如果事先存在对于预测问题最终基于某一策略最终价值函数 $V^{\pi}(s)$ 或 $Q^{\pi}(s,a)$, 或者存在对于控制问题的最优价值函数 $V^{*}(s)$ 或 $Q^{*}(s,a)$, 那么可以使用这些价值来代替上式中的目标值, 这里集中使用 V_{target} 或 Q_{target} 来代表目标值, 使用期望代替平均值的方式, 那么目标函数的表达公式为:

$$J(w)=\frac{1}{2}\mathbb{E}_{\pi}\left[V_{\text{target}}(s)-\hat{V}(s;w)\right]^2, \tag{8.8}$$

$$J(w)=\frac{1}{2}\mathbb{E}_{\pi}\left[Q_{\text{target}}(s,a)-\hat{Q}(s,a;w)\right]^2. \tag{8.9}$$

对于大规模强化学习问题来说，并不能保证对所有的 $J(w)$ 直接对其求梯度，并基于所求梯度等于 0 来得到最优参数。在这种情况下可以通过迭代、使用梯度下降法来求解。具体过程如下:

1. 设置初始参数值 $w=(w_1,w_2,\ldots,w_d)$;

2. 获取一个状态转换, 代入目标函数 $J(w)$, 并计算 $J(w)$ 对各参数 w 的梯度:

$$\nabla_w J(w)=\begin{pmatrix}\dfrac{\partial J(w)}{\partial w_1}\\ \vdots\\ \dfrac{\partial J(w)}{\partial w_d}\end{pmatrix}$$

3. 设置一个正的较小的步长 α, 将原参数 w 朝着梯度的反方向更新:

$$\Delta w=-\alpha\nabla_w J(w)$$
$$w\leftarrow w+\Delta w$$

对于式 (8.8), w 更新的具体表达式推导如下:

$$\begin{aligned} w_{t+1} &= w_t - \alpha_t \nabla J(w_t) \quad \text{(梯度下降准则)} \\ &= w_t + \alpha_t \mathbb{E}_\pi \left[V_{\text{target}}(s_{t+1}) - \hat{V}(s_t; w_t) \right] \nabla_w \hat{V}(s_t; w)|_{w=w_t} \\ &\approx w_t + \alpha_t \left[V_{\text{target}}(s_{t+1}) - \hat{V}(s_t; w_t) \right] \nabla_w \hat{V}(s_t; w)|_{w=w_t} \quad \text{(抽样)} \end{aligned} \tag{8.10}$$

使用不同的学习方法时，V_{target} 由不同的目标值代替，比如，对于蒙特卡罗方法, 使用 G_t 代替 V_{target}，如表 8.1 所示。在时序差分方法中，用 $R_{t+1} + \hat{V}(s_{t+1}; w)$ 代替 V_{target}。对于式 (8.9)，参数 w 的更新表达式类似，同样用不同的目标值代替 Q_{target}。

8.1 基于梯度的蒙特卡罗值函数评估算法

输入: 要评估的策略 π，一个可微的逼近函数 $\hat{V} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
　　恰当地初始化值函数的权重 w （比如，$w = 0$）
Repeat:
　　利用策略 π 产生一幕数据
　　For $t = 0, 1, \cdots, T-1$
　　$w \leftarrow w + \alpha_t [G_t - \hat{V}(s_t; w)] \nabla_w \hat{V}(s_t; w)$

概括来说，式 (8.8), (8.9) 是我们要求解的。在实际求解近似值函数中的参数 w 时, 式中的期望 $\mathbb{E}_\pi$ 并不好处理，为了去掉期望 $\mathbb{E}_\pi$, 最好的方法就是将 π 的概率分布求出来, 然后再加权求和。这样操作过于复杂, 实际操作的时候会采用蒙特卡罗的形式, 只用一个样本或者一批样本进行更新。即，我们基于近似值函数的目标值来代替 V_{target} 或 Q_{target}，考虑抽样后的参数更新表达式来得到极小值 w。

注意到在时序差分方法中,要更新的参数 w 不仅出现在要估计的值函数 $\hat{V}(s_t; w)$ 中，还出现在目标值 V_{target} 中。这里只考虑参数 w 对估计值函数 $\hat{V}(s_t; w)$ 的影响而忽略对目标值函数 V_{target} 的影响, 这种方法称为基于半梯度的 TD(0) 值函数评估算法，如表 8.2 所示。表 8.3 给出了基于半梯度的 Sarsa 算法。

8.2 基于半梯度的 TD(0) 值函数评估算法

输入: 要评估的策略 π，一个可微的逼近函数 $\hat{V} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
　　恰当地初始化值函数的权重 w （比如，$w = 0$）
Repeat:
　　初始化状态 s
　　Repeat （对一幕中的每一步）
　　选择动作 $a \sim \pi(\cdot|s)$
　　采用动作 a 观测到回报 R, s'
　　$w \leftarrow w + \alpha_t [R + \gamma \hat{V}(s'; w) - \hat{V}(s_t; w)] \nabla_w \hat{V}(s_t; w)$
　　$s \leftarrow s'$

直到 s' 是终止状态

8.3 基于半梯度的 Sarsa 算法

输入: 一个可微的逼近函数 $\hat{Q}: \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

任意初始化值函数的权重 w （比如，$w = 0$）

Repeat (for each episode):

初始化状态行为对 s, a

Repeat （对一幕数据中的每一步）

采用动作 a 观测到回报 R, s'

如果 s' 是终止状态: $w \leftarrow w + \alpha_t[R - \hat{Q}(s, a; w)]\nabla_w \hat{Q}(s, a; w)$

进入下一幕

利用软策略选择一个动作 a'，以便估计动作值函数 $\hat{Q}(s', a'; w)$

$w \leftarrow w + \alpha_t[R + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)]\nabla_w \hat{Q}(s, a; w)$

$s \leftarrow s'$

$a \leftarrow a'$

19.3.3 线性逼近

到目前为止，我们已经知道了值函数逼近下的优化目标函数，优化策略一般用梯度下降来优化，接下来我们讨论要逼近的值函数的形式。理论上任何函数都可以被用作近似值函数，实际选择何种近似函数需根据问题的特点。比较常用的近似函数有线性模型、神经网络、决策树、最近邻法、傅里叶变换、小波变换等。总的来说就是基于一映射 $\phi: \mathcal{S} \to \mathbb{R}^d$ 构造特征集（基函数），希望每一个状态都能被恰当的表示。本小节主要介绍参数化的线性逼近和基于神经网络的非线性逼近。首先我们引入线性逼近。

相比于非线性逼近, 线性逼近的好处是只有一个最优值, 因此可以收敛到全局最优。线性逼近的表达形式为:

$$\hat{V}(s; w) = w^T \phi(s) = \sum_{j=1}^{d} w_j \phi_j(s)$$

其中 $\phi(s)$ 为状态 s 处的特征函数，或者称为基函数，w 为要求解的参数。常用的基函数的类型如下。

- 多项式基函数，如 $\left(1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, \cdots\right)$
- 傅里叶基函数: $\phi_j(s) = \cos(j\pi s), s \in [0, 1]$

- 径向基函数: $\phi_j(s) = \exp\left(-\frac{\|s - c_j\|^2}{2\sigma_j^2}\right)$

对于由线性函数 $\hat{V}(s;w)$ 近似的值函数, 其相应的目标函数 $J(w)$ 为:

$$J(w) = \frac{1}{2}\mathbb{E}_\pi\left[V_{\text{target}}(s) - w^T\phi(s)\right]^2$$

相应的梯度 $\nabla_w J(w)$ 为:

$$\nabla_w J(w) = -\mathbb{E}_\pi\left(V_{\text{target}}(s) - w^T\phi(s)\right)\phi(s)$$

基于样本，参数的更新表达式为:

$$w_{t+1} \leftarrow w_t + \alpha_t\left(V_{\text{target}}(s_{t+1}) - w_t^T\phi(s_t)\right)\phi(s_t)$$

正如前文提到的，使用不同的学习方法时, $V_{\text{target}}(s_{t+1})$ 由不同的目标值代替。

- 蒙特卡罗方法值函数线性逼近参数更新公式:

$$w_{t+1} \leftarrow w_t + \alpha_t\left[G_t - w^T\phi(s_t)\right]\phi(s_t)$$

- TD(0) 线性逼近参数更新公式:

$$w_{t+1} \leftarrow w_t + \alpha_t\left[R_{t+1} + \gamma w^T\phi(s_{t+1}) - w^T\phi(s_t)\right]\phi(s_t)$$

- 前向视角的 TD(λ) 参数更新公式:

$$w_{t+1} \leftarrow w_t + \alpha_t\left(G_t^\lambda - w^T\phi(s_t)\right)\phi(s_t)$$

- 后向视角的 TD(λ) 参数更新公式:

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma w^T\phi(s_{t+1}) - w^T\phi(s_t)\\ E_t &= \gamma\lambda E_{t-1} + \phi(s_t)\\ w_{t+1} &\leftarrow w_t + \alpha_t\delta_t E_t\end{aligned}$$

如式 (8.6) 所示，为衡量在采样产生的 T 个状态转换上近似值函数的收敛情况，希望充分利用样本数据 $D = \{\langle s_1, V_1^\pi\rangle, \langle s_2, V_2^\pi\rangle, \cdots, \langle s_T, V_T^\pi\rangle\}$, 找到最好的拟合函数 $\hat{V}(s,w)$, 使得目标函数，记为 $LS(w)$，最小，

$$LS(w) = \frac{1}{2T}\sum_{t=1}^{T}\left(V_t^\pi - w^T\phi(s_t)\right)^2.$$

基于 $LS(w)$ 可以直接找到参数使得 $LS(w)$ 为最小值，即线性最小二乘逼近，并且满足经验集中的各个状态，

$$\sum_{t=1}^{T}\left[V_t^\pi - w^T\phi\left(s_t\right)\right]\phi\left(s_t\right) = 0.$$

- 最小二乘蒙特卡罗方法参数为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \phi\left(s_t\right)^T \right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) G_t$$

- 最小二乘 TD(0) 方法为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \left(\phi\left(s_t\right) - \gamma \phi\left(\mathrm{s}_{t+1}\right)\right)^T \right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) R_{t+1}$$

- 最小二乘 TD(λ) 前向方法为

$$w = \left(\sum_{t=1}^{T} \phi\left(s_t\right) \phi\left(s_t\right)^T \right)^{-1} \sum_{t=1}^{T} \phi\left(s_t\right) G_t^{\lambda}$$

- 最小二乘 TD(λ) 后向方法为

$$w = \left(\sum_{t=1}^{T} E_t \left(\phi\left(s_t\right) - \gamma \phi\left(\mathrm{s}_{t+1}\right)\right)^T \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

19.3.4 非线性逼近

线性逼近时，首先要选基函数，再根据目标函数训练得到基函数所对应的参数。这种线性逼近的方法表示能力非常有限，因为数量太少的基函数无法很好的逼近复杂的函数，而且基函数的形式是事先选定的，这也限制了函数的逼近能力。这里我们介绍一些常用的基于卷积神经网络（深度学习）的 Q-learning 算法及其变体：Deep Q-learning （DQN），Double DQN 和 Dueling DQN。由于卷积神经网络可以看成是基函数参数化的一种方法，因此本小节给出的逼近方法也称为基于参数的非线形逼近。关于卷积神经网络的具体内容详见第三章。

DQN 算法

在之前的章节中我们学过，Q-learning 是一种离线学习法，它能学习当前经历着的，也能学习过去经历过的，甚至是学习别人的经历。而 DQN，就是在传统 Q-learning 的基础上做了三个改进，使得其效果显著提升。

DQN 对 Q-learning 的修改主要体现在以下三个方面：

1. DQN 利用深度卷积神经网络逼近值函数；

2. DQN 利用了经验回放训练强化学习的学习过程；

3. DQN 独立设置了目标网络来单独处理时序差分算法中的 TD 偏差.

下面就三个方面进行具体介绍。

（1）DQN 利用深度卷积神经网络逼近值函数

我们已经知道线性逼近值函数的方法，即值函数由一组基函数和一组与之对应的参数相乘得到，值函数是参数的线性函数。而 DQN 的行为值函数利用神经网络逼近，而且用的是强大的深度卷积神经网络，也就是 CNN，属于非线性逼近。虽然逼近方法不同，但都属于参数逼近。此处的值函数对应着一组参数，在神经网络里参数是每层网络的权重。此时值函数的更新对应参数的更新。

（2）DQN 利用了经验回放训练强化学习的学习过程

当时学者们发现利用神经网络，尤其是深度神经网络逼近值函数不太靠谱，因为常常出现不稳定不收敛的情况。主要原因是，训练神经网络时，存在的假设是训练数据是独立同分布的，但是通过强化学习采集的数据之间存在着关联性，利用这些数据进行顺序训练，神经网络自然不稳定。DeepMind 团队的研究人员构造了一种神经网络的训练方法：经验回放。具体来说，DQN 有一个记忆库用于学习之前的经历，每次 DQN 更新的时候，都可以随机抽取一些之前的经历进行学习，这种随机抽取做法打乱了经历之间的相关性，使得神经网络更新更有效率。

（3）DQN 独立设置了目标网络来单独处理时序差分算法中的 TD 偏差

与前面提到的表格型 Q-learning 算法不同的是，利用神经网络对值函数进行逼近时，值函数的更新步更新的是参数，其更新方法是梯度下降法。也就是说 Q-learning 值函数更新实际上变成了监督学习的一次更新过程：

$$w_{t+1} \leftarrow w_t + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; w_t\right) - Q(s_t, a_t; w_t) \right] \nabla_w Q(s_t, a_t; w)|_{w=w_t}$$

其中，$R_{t+1} + \gamma \max\limits_{a'} Q\left(s_{t+1}, a_{t+1}; w_t\right)$ 为 TD 目标值，在计算 $\max\limits_{a'} Q\left(s_{t+1}, a_{t+1}; w_t\right)$ 值时，用到的网络参数为 w_t。我们称计算 TD 目标时所用的网络为 TD 网络。在 DQN 算法出现之前，利用神经网络逼近值函数时，计算 TD 目标值时所用的网络参数，与梯度计算中要逼近的值函数 $Q(s_t, a_t; w_t)$ 所用的网络参数相同，这样就容易导致数据间存在关联性，从而使训练不稳定。为了解决此问题，DeepMind 团队提出使用两个结构相同但参数不同的神经网络，其中动作值函数逼近的网络具备最新的参数，而用于计算 TD 目标值的网络则是每隔固定的步数更新一次，这样的方式能让训练变得更加有效，更具有鲁棒性。值函数的更新变为，

$$w_{t+1} \leftarrow w_t + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'; w^-\right) - Q(s_t, a_t; w_t) \right] \nabla_w Q(s_t, a_t; w)|_{w=w_t}.$$

最后我们给出 DQN 的伪代码，如下所示。

DQN 算法
初始化回放记忆 $\mathcal{D}$，可容纳的数据条数为 N

利用随机权值 w 初始化动作-行为值函数 Q
令 $w^- = w$ 初始化，计算 TD 目标值 Q

1. For episode= $1, \cdots, M$ do
2. 初始化事件的第一个状态 $s_1 = \{x_1\}$，通过预处理得到状态对应的特征输入 $\phi_1 = \phi(s_1)$
3. For $t = 1, \cdots, T$ do
4. 利用概率 ε 选一个随机动作 a_t
5. 若小概率事件没发生，则用贪婪策略选择当前值函数最大的那个动作 $a_t = \arg\max Q\left(\phi\left(s_t\right), a; w\right)$
6. 在仿真器中执行动作 a_t，观测回报 R_t 以及图像 x_{t+1}
7. 设置 $s_{t+1} = s_t, a_t, x_{t+1}$，预处理 $\phi_{t+1} = \phi\left(s_{t+1}\right)$
8. 将转换 $(\phi_t, a_t, R_t, \phi_{t+1})$ 储存在回放记忆 $\mathcal{D}$ 中
9. 从回放记忆 $\mathcal{D}$ 中均匀随机采样一个转换样本数据，用 $(\phi_j, a_j, R_j, \phi_{j+1})$ 表示
10. 令 $y_j = \begin{cases} R_j, & \text{事件为终止状态;} \\ R_j + \gamma \max\limits_{a'} Q\left(\phi_{j+1}, a'; w^-\right), & \text{其它} \end{cases}$
11. 关于参数 w 执行一次梯度下降，目标函数为 $\left(y_j - Q(\phi_j, a_j; w)\right)^2$
12. 每隔 C 步更新一次 TD 目标网络权值，即令 $w^- = w$;
13. end for
14. end for

DQN 算法实践

这里我们首先引入常用的库

```
import random
import gym
import numpy as np
import collections
from tqdm import tqdm
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
```

首先定义经验回放池的类，用于储存数据和随机抽取一些之前的经历进。

```
class ReplayBuffer:      #定义一个类
    def __init__(self, capacity):
        #初始化这个类的属性，self.buffer是这个类的属性，赋予他一个默认
            值，之后会用这些属性。
        self.buffer = collections.deque(maxlen=capacity)
        # 相当于 list ，但这里运算更快。数据是队列先进先出，maxlen是指
            队列最大长。
```

```
    def add(self, state, action, reward, next_state, done):
        # 定义类 ReplayBuffer的名为add的方法，这一方法是对这个类的属性
            进行操作。(ReplayBuffer.add(参数)可以调用这一方法)
        self.buffer.append((state, action, reward, next_state, done))
        #这里state, action,reward,next_state,done等参数在调用的时候是要
            我们自己输入。 append方法的目的是将数据加入buffer这一属性

    def sample(self, batch_size):
        #从buffer中采样数据,数量为 batch_size。
        transitions = random.sample(self.buffer, batch_size)
        #random.sample()随机从self.buffer里面抽取数量为batch_size的数据
            储存在  transitions里。
        state, action, reward, next_state, done = zip(*transitions)
        return np.array(state), action, reward, np.array(next_state),
            done

    def size(self):
        #目前buffer中数据的数量
        return len(self.buffer)
```

接下来定义一个只有一层隐藏层的 Q 网络。之后我们会用到这样一个网络。

```
class Qnet(torch.nn.Module):
    #torch.nn.Module是一个父类，torch.nn.Module中的属性还有方法都传递
        给子类 Qnet.
    # 首先，导入torch.nn模块。实际上，“nn”是neural #networks（神经网
        络）的缩写。顾名思义，该模块定义了大量神经网络的层。
    def __init__(self, state_dim, hidden_dim, action_dim):
        super(Qnet, self).__init__()
        #子类继承了父类的所有属性和方法，父类属性自然会用父类方法来进行
            初始化
        self.fc1 = torch.nn.Linear(state_dim, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, action_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))  # 隐藏层使用ReLU激活函数
        return self.fc2(x)
```

有了这些基本组件之后，接来下开始实现 DQN 算法。

```
class DQN:
    '''␣DQN算法␣'''
    def __init__(self, state_dim, hidden_dim, action_dim, learning_
```

```
        rate, gamma,
                  epsilon, target_update, device):
        self.action_dim = action_dim
        self.q_net = Qnet(state_dim, hidden_dim,self.action_dim).to(
            device)
        # Q网络
        # 目标网络
        self.target_q_net = Qnet(state_dim,hidden_dim,self.action_dim).
            to(device)
        # 使用Adam优化器
        self.optimizer = torch.optim.Adam(self.q_net.parameters(),
                                          lr=learning_rate)
        self.gamma = gamma  # 折扣因子
        self.epsilon = epsilon  # epsilon-贪婪策略
        self.target_update = target_update  # 目标网络更新频率
        self.count = 0  # 计数器,记录更新次数
        self.device = device

    def take_action(self, state):  # epsilon-贪婪策略采取动作
        if np.random.random() < self.epsilon:
            action = np.random.randint(self.action_dim)
        else:
            state = torch.tensor([state], dtype=torch.float).to(self.
                device)
            action = self.q_net(state).argmax().item()      #.argmax()返
                回最大值的索引值,.item()将张量转化为标量。
        return action

    def update(self, transition_dict):
        #transition_dict是一个字典, 例如含有键对  'actions':3
        states = torch.tensor(transition_dict['states'], dtype=torch.
            float).to(self.device)
        #返回states对应数值。
        actions = torch.tensor(transition_dict['actions']).view(-1, 1).
            to(self.device)
        #.view(-1, 1)按照列展平数组, -1表示展平, 1表示按照列。例: view
            (3, 4)重塑形状为三行四列的张量。
        rewards = torch.tensor(transition_dict['rewards'],dtype=torch.
            float).view(-1, 1).to(self.device)
        next_states = torch.tensor(transition_dict['next_states'],
            dtype=torch.float).to(self.device)
        dones = torch.tensor(transition_dict['dones'],
        dtype=torch.float).view(-1, 1).to(self.device)
```

```
        q_values = self.q_net(states).gather(1, actions)
        # 返回Q值 .gather表示聚合，1表示按照列聚合，actions是每一列要聚
            会元素的索引
         # 下个状态的最大 Q值
        max_next_q_values = self.target_q_net(next_states).max(1)[0].
            view(
              -1, 1)
        q_targets = rewards + self.gamma * max_next_q_values * (1 -
            dones)
        # TD误差目标
        dqn_loss = torch.mean(F.mse_loss(q_values, q_targets))
        # 均方误差损失函数
        self.optimizer.zero_grad()
        # PyTorch中默认梯度会累积,这里需要显式将梯度置为0
        dqn_loss.backward()  # 反向传播更新参数
        self.optimizer.step()

        if self.count % self.target_update == 0:
            self.target_q_net.load_state_dict(
            self.q_net.state_dict())  # 更新目标网络
            self.count += 1
```

有了这些基本组件之后，接来下开始实现 DQN 算法。

```
lr = 2e-3
num_episodes = 500
hidden_dim = 128
gamma = 0.98
epsilon = 0.01
target_update = 10
buffer_size = 10000
minimal_size = 500
batch_size = 64
device = torch.device("cuda") if torch.cuda.is_available() else torch.
    device(
    "cpu")

env_name = 'CartPole-v0'
env = gym.make(env_name)
random.seed(0)
np.random.seed(0)
env.seed(0)
torch.manual_seed(0)
replay_buffer = ReplayBuffer(buffer_size)
```

```
state_dim = env.observation_space.shape[0]
action_dim = env.action_space.n
agent = DQN(state_dim, hidden_dim, action_dim, lr, gamma, epsilon,
            target_update, device)

return_list = []
for i in range(10):
    with tqdm(total=int(num_episodes / 10), desc='Iteration␣%d' % i)
        as pbar:
        for i_episode in range(int(num_episodes / 10)):
            env.render()
            episode_return = 0
            state = env.reset()
            done = False
            while not done:
                action = agent.take_action(state)
                next_state, reward, done, _ = env.step(action)
                replay_buffer.add(state, action, reward, next_state,
                    done)
                state = next_state
                episode_return += reward
                # 当buffer数据的数量超过一定值后,才进行Q网络训练
                if replay_buffer.size() > minimal_size:
                    b_s, b_a, b_r, b_ns, b_d = replay_buffer.sample(
                        batch_size) # 调用上文中的类方法 replay_buffer.
                        sample
                    transition_dict = {
                        'states': b_s,
                        'actions': b_a,
                        'next_states': b_ns,
                        'rewards': b_r,
                        'dones': b_d
                    }
                    agent.update(transition_dict)
            return_list.append(episode_return)
            if (i_episode + 1) % 10 == 0:
                pbar.set_postfix({
                    'episode':
                    '%d' % (num_episodes / 10 * i + i_episode + 1),
                    'return':
                    '%.3f' % np.mean(return_list[-10:])
                })
            pbar.update(1)
```

Double DQN

现在我们已了解到 DQN 利用了卷积神经网络表示行为值函数，并利用了经验回放和单独设立目标网络这两个技巧。但是 DQN 无法克服 Q-learning 本身所固有的缺点：过估计，即，估计的值函数比真实值函数要大。Q-learning 之所以存在过估计的问题，主要源于 Q-learning 中的最大化操作。具体来说，不论是表格型行为值函数的更新公式：

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right)+\alpha\left[R_t+\gamma \max_{a'} Q\left(s_{t+1}, a'\right)-Q\left(s_t, a_t\right)\right],$$

还是行为值函数逼近下的更新公式：

$$w_{t+1} \leftarrow w_t+\alpha_t\left(R_{t+1}+\gamma \max_{a'} Q\left(s_{t+1}, a' ; w_t\right)-Q\left(s_t, a_t ; w_t\right)\right) \nabla_w Q\left(s_t, a_t ; w\right) | w=w_t,$$

其中都有 max 操作。max 操作使得估计的值函数比真实值函数大。如果值函数中每一点的值都被过估计了相同的幅度，即过估计量是均匀的，那么基于最大的值函数找到的最优策略保持不变。也就是说，在这种情况下，即使值函数被过估计了，也不影响最优的策略。然而，在实际情况中，过估计量并非是均匀的，因此值函数的过估计会影响最终的策略决定，从而导致最终的策略并非最优。为了解决值函数过估计的问题，Hasselt 提出了 Double Q-learning 的方法：将动作的选择和动作的评估分别用不同的值函数来实现。我们先介绍一般 Q-learning 的动作选择和评估动作。

- 动作选择：在 Q-learning 的值函数更新中，TD 目标为

  $$Y_t^Q=R_{t+1}+\gamma \max_{a'} Q\left(s_{t+1}, a' ; w_t\right)$$

  在求 Y_t^Q 时，我们首先需要选择一个动作 a^*，该动作 a^* 使得在状态 s_{t+1} 处 $Q\left(s_{t+1}, a\right)$ 最大。

- 动作评估：选出 a^* 后，利用 a^* 处的行为值函数构造 TD 目标。

可以看出一般的 Q-learning，其动作选择和评估动作在同一个参数 w_t 下进行。Double Q-learning 考虑用不同的行为值函数选择和评估动作，其参数并不一样。动作选择所用的动作值函数为

$$\arg \max_{a'} Q\left(s_{t+1}, a' ; w_t\right),$$

此时动作值函数网络的参数为 w_t。当选出最大的动作 a^* 后，动作评估公式为

$$Y_t^{\mathrm{DoubleQ}}=R_{t+1}+\gamma Q\left(s_{t+1}, a^* ; w_t'\right)$$

此时动作评估所用的动作值函数网络参数为 w_t'。对应的 Double Q-learning 的 TD 目标为

$$Y_t^{\mathrm{DoubleQ}}=R_{t+1}+\gamma Q\left(s_{t+1}, \operatorname{argmax}_{a'} Q\left(s_{t+1}, a' ; w_t\right) ; w_t'\right).$$

将 Double Q-learning 的思想应用到 DQN 中, 则得到 Double DQN 算法, 其伪代码与 DQN 一致, 仅将 DQN 算法中的 TD 目标修改为

$$Y_t^{\text{DoubleQ}} = R_{t+1} + \gamma Q\left(s_{t+1}, \underset{a'}{\text{argmax}} Q\left(s_{t+1}, a'; w_t\right); w_t^-\right).$$

优先回放 (Prioritized Replay)

DQN 的成功归结于经验回放和独立的目标网络。Double DQN 改进了 Q-learning 中的 max 操作, 但经验回放仍然采用均匀分布。考虑到智能体的经验即经历过的数据, 对于智能体的学习并非具有同等重要的意义, 比如, 智能体在某些状态的学习效率比其他状态的学习效率高。所以经验回放利用均匀分布采样并没有高效的利用数据。优先回放的基本思想就是打破均匀采样, 赋予学习效率高的状态以更大的采样权重。

如何选择权重呢? 值函数更新中 TD 误差有着重要作用, 我们的目标就是让 TD 误差尽可能小, 如果 TD 误差比较大, 意味着我们当前的行为函数离目标函数差距还很大, 应多进行更新, 因此基于 TD 误差来设置权重。我们设样本 i 处的 TD 误差为 δ_i, 令样本处的采样概率为

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

其中 p_i 是样本 i 的优先级, 由 TD 误差 δ_i 决定, 指数 $\alpha \in [0,1]$ 是决定优先级化的程度, 当 $\alpha = 0$ 时, 退化成均匀采样。p_i 的设置一般有两种方法, 第一种方法是 $p_i = |\delta_i| + \epsilon$, ϵ 是一个很小的正常数为了使 TD 误差为 0 的特殊边缘样本也能够被抽取; 第二种方法是 $p_i = \dfrac{1}{\text{rank}(i)}$, 其中 $\text{rank}(i)$ 根据 $|\delta_i|$ 的排序得到。

当我们采用优先回放的概率分布采样时, 动作值函数的估计值是一个有偏估计。因为采样分布与动作值函数的分布是两个完全不同的分布, 为了矫正这个偏差, 引入重要性采样系数 $\theta_i = \left(\dfrac{1}{N} \cdot \dfrac{1}{P(i)}\right)^\beta$, 其中 N 是经验池容量。如果 $\beta = 1$, 则可以完全补偿优先采样带来的偏差。在实际操作中, 其实重要的是最终收敛的值函数无偏, 但在训练过程中本来就是不稳定的, 所以有点偏差可以容忍, 所以一种温和的校正偏差方式是, β 从一个 $\beta_0 < 1$ 的超参逐渐增大, 到训练结束时 $\beta = 1$。带有优先回放的 Double DQN 算法伪代码如下。

带有优先回放的 Double DQN 算法
1. 输入: 确定 minibatch 的大小 k, 步长 η, 回放周期 K, 存储数据的总大小 N, 常数 α, β, 总时间 T;
2. 初始化回放记忆库 $\mathcal{D} = \varnothing, \triangle = 0, p_1 = 0$;
3. 观测初试状态, 选择动作 $a_0 \sim \pi_w(s_0)$;
4. For $t = 1, \cdots, T$ do;

5. 利用动作 A 作用于环境，环境返回观测 s_t, R_t, γ_t;
6. 将数据 $(s_{t-1}, a_{t-1}, R_t, \gamma_t, s_t)$ 存储到记忆库 $\mathcal{D}$ 中，且令其优先级为 $p_t = \max_{i<t} p_i$，采用该优先级初始化的目的是保证每个样本至少被利用一次;
7. If $t = 0 \bmod K$ then （每隔 K 步回放一次）
8. for $j = 1, \cdots, k$ do （依次采集 k 个样本）;
9. 根据概率分布 $j \sim P(j) = p_j^\alpha / \Sigma_i p_i^\alpha$ 采样一个样本点;
10. 计算样本点的重要性权重 $\theta_j = (N \cdot P(j))^{-\beta} / \max_i \theta_i$;
11. 计算该样本点处的 TD 偏差，
$$\delta_j = R_j + \gamma_j Q_{w_{\text{target}}}(s_j, \arg\max_a Q(s_j, a)) - Q(s_{j-1}, a_{j-1});$$
12. 更新该样本的优先级 $p_j \leftarrow |\delta_j|$;
13. 累计权重的改变量 $\Delta \leftarrow \Delta + \theta_j \cdot \delta_j \cdot \nabla_w Q(s_{j-1}, a_{j-1})$;
14. end for
15. 处理完 k 个样本后更新参数值 $w \leftarrow w + \eta \cdot \Delta$，重新设置 $\Delta = 0$;
16. 偶尔地复制新权重到目标网络中，即 $w_{\text{target}} \leftarrow w$;
17. end if
18. 根据新的策略选择下一个动作，$a_t \sim \pi_w(s_t)$;
19. end for

Dueling DQN

不管是最初的 DQN，还是 DQN 的变体 Double DQN，在值函数逼近时所用的神经网络都是卷积神经网络（图 19.14）。Dueling DQN 则从网络结构上改进了 DQN（图 19.15）。

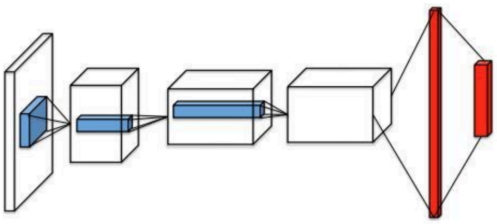


图 19.14 DQN 网络结构

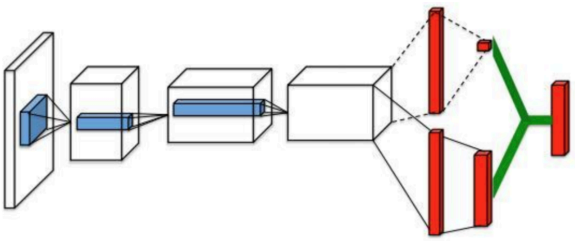


图 19.15 Dueling DQN 网络结构

Dueling DQN 考虑改进网络结构的原因在于：已知 $Q(s,a)$ 表示状态 s 下动作 a 的价值。因为有状态这个条件，$Q(s,a)$ 并不能完全代表状态 a 的价值，因为有时在某种状态下，无论做什么动作，对下一个状态都没有很大的影响：在一个好的状态下，无论做什么动作，都能得到很高价值；在一个很差的状态下，采取任何动作，都得到一个较低的价值。因此提出了 Dueling DQN 方法，希望衡量状态 s 的价值 $V(s)$ 和动作 a 的价值 $A(s,a)$。将状态 s 的价值 $V(s)$ 和动作的价值 $A(s,a)$ 相加得到状态 s 下动作 a 的价值 $Q(s,a)$，即

$$Q(s,a;\theta,\alpha,\beta)=V(s;\theta,\alpha)+A(s,a;\theta,\beta),$$

其中，$V(s;\theta,\alpha)$ 称为状态值函数，$A(s,a;\theta,\beta)$ 称为优势函数，θ 是公共部分的网络参数，α 是价值函数独有的部分网络参数，β 是优势函数独有的部分网络参数。但是上述公式并不能辨识最终输出中 $V(s;\theta,\alpha)$ 和 $A(s,a;\theta,\beta)$ 各自的作用，为了可以体现这种可辨识性 (identifiability)，实际使用的组合公式如下，

$$Q(s,a;\theta,\alpha,\beta)=V(s;\theta,\alpha)+\left(A(s,a;\theta,\beta)-\frac{1}{|\mathcal{A}|}\sum_{a'}A\left(s,a';\theta,\beta\right)\right).$$

19.4 策略梯度方法

之前介绍的强化学习方法都是基于值函数的，通过近似值函数从而间接得到我们最终需要的策略。然而在强化学习中我们最终的目标是得到一个最优的策略，所以一个自然的问题是有没有相应的算法可以直接近似策略函数，通过学习参数化的策略从而直接达到我们的目标。本章我们介绍这类重要的方法，我们叫做策略梯度方法。

策略梯度的基本思想是直接参数化策略函数，然后基于一些标量化指标 $J(\theta)$，通过极大化目标函数从而优化策略函数的参数。这些方法试图使性能最大化，因此它们的更新是对目标行数 J 执行梯度上升：

$$\theta_{t+1}=\theta_t+\alpha\widehat{\nabla J\left(\theta_t\right)}$$

其中 $\widehat{\nabla J(\theta_t)}$ 是目标函数梯度的一个随机估计，我们把遵循此类模式的方法都称为*策略梯度方法*，无论方法本身是否同时也近似学习了值函数。基于策略梯度的基本思想，我们推到得到常用的演员-评论家算法，其中“演员”所指学习策略，“评论家”所指学习值函数，进而我们也给出了一些主流策略梯度方法的简单介绍。

19.4.1 策略近似及其优势

在策略梯度方法中，我们只需要直接参数化策略函数，即 $\pi(a \mid s, \boldsymbol{\theta})$，我们重点关注的是离散动作空间的参数化方法，对于连续动作空间可以通过参数化正态分布输出连续动作空间来实现，感兴趣的作者可以查阅更多学习资料。

如果动作空间离散并且空间不是很大，一种常见的做法是对每个状态输出对应动作的概率，一种常见的参数化方法是基于 soft-max 方法：

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

其中输出函数 $h(s, a, \boldsymbol{\theta})$ 可以任意参数化，例如可以通过神经网络，或者通过对特征 $\mathbf{x}(s, a)$ 的线性化参数方法：

$$h(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s, a)$$

相较于基于值函数的方法，策略梯度方法具有如下的几个优点：

1. 近似策略可以接近确定性策略，而基于动作选择的 ϵ 贪婪总是有 ϵ 可能性选择随机动作，而最优策略通常是确定性的，所有从逼近的有效性角度来看，策略梯度方法更容易得到一个最优的策略。

2. 很多强化学习的问题用策略梯度方法更容易优化，从而更容易收敛到较优的策略。另外在具有显著函数逼近的问题中，最佳近似策略可能是随机的，而策略梯度方法能够选择具有任意概率的动作。

3. 策略梯度方法能够处理连续的动作空间，基于值函数的方法不能处理。

19.4.2 策略梯度定理

强化学习的目标是为智能体找到一个最优的行为策略从而获取最大的回报，策略梯度方法主要特点在于直接对策略进行建模并且通过优化参数来实现最大化回报的目标，具体来说，回报函数的值收到该策略的直接影响，因此可以利用很多算法来对参数进行优化进而最大化目标函数。

目标函数的定义如下：

$$J(\theta)=\sum_{s\in\mathcal{S}}d^{\pi}(s)V^{\pi}(s)=\sum_{s\in\mathcal{S}}d^{\pi}(s)\sum_{a\in\mathcal{A}}\pi_{\theta}(a\mid s)Q^{\pi}(s,a)$$

其中 $d^{\pi}(s)$ 代表由 π_{θ} 引出的马尔科夫链的平稳分布。如果我们能够直接计算得到目标函数的梯度，就可以利用梯度上升来更新参数了。但是我们发现计算梯度 $\nabla_{\theta}J(\theta)$ 是一件很困难的事情，因为梯度值不仅依赖于动作的选择（由 π_{θ} 直接决定），还依赖于由选择的动作而产生的状态的平稳分布（由 π_{θ} 间接决定）。因为我们发现环境通常是未知的，很难去估计策略的更新对于状态分布造成的影响。

这个是时候我们需要借助策略梯度定理来方便梯度的计算，定理的基本原理是对梯度的形式进行变形使其不依赖于状态分布的导数，从而在很大的程度上简化了梯度 $\nabla_{\theta}J(\theta)$ 的计算，具体的证明可以参考 sutton 的书，最终得到的梯度计算公式如下：

首先要注意的是，状态值函数的梯度可以写成动作值函数

$$\begin{aligned}
\nabla v_{\pi}(s)&=\nabla\left[\sum_{\bar{a}}\pi(a\mid s)q_{\pi}(s,a)\right],\quad \text{for all } s\in\mathcal{S}\\
&=\sum_{\bar{a}}\left[\nabla\pi(a\mid s)q_{\pi}(s,a)+\pi(a\mid s)\nabla q_{\pi}(s,a)\right]\\
&=\sum_{\bar{a}}\left[\nabla\pi(a\mid s)q_{\pi}(s,a)+\pi(a\mid s)\nabla\sum_{s',r}p\left(s',r\mid s,a\right)\left(r+v_{\pi}\left(s'\right)\right)\right]\\
&=\sum_{\bar{a}}\left[\nabla\pi(a\mid s)q_{\pi}(s,a)+\pi(a\mid s)\sum_{s'}p\left(s'\mid s,a\right)\nabla v_{\pi}\left(s'\right)\right]\nabla v_{\pi}\left(s'\right)\Big]\\
&=\sum_{\bar{a}}\Bigg[\nabla\pi(a\mid s)q_{\pi}(s,a)+\pi(a\mid s)\sum_{s'}p\left(s'\mid s,a\right)\\
&\qquad\sum_{a'}\left[\nabla\pi\left(a'\mid s'\right)q_{\pi}\left(s',a'\right)+\pi\left(a'\mid s'\right)\sum_{s''}p\left(s''\mid s',a'\right)\nabla v_{\pi}\left(s''\right)\right]\Bigg]\\
&=\sum_{x\in S}\sum_{k=0}^{\infty}\Pr(s\to x,k,\pi)\sum_{\bar{a}}\nabla\pi(a\mid x)q_{\pi}(x,a),
\end{aligned}$$

重复展开后

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi\left(s_0\right) \\
&= \sum_s\left(\sum_{k=0}^{\infty} \Pr\left(s_0 \rightarrow s, k, \pi\right)\right) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_s \eta(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_{s'} \eta\left(s'\right) \sum_s \frac{\eta(s)}{\sum_{s'} \eta\left(s'\right)} \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&= \sum_{s'} \eta\left(s'\right) \sum_s \mu(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a) \\
&\propto \sum_s \mu(s) \sum_{\bar{a}} \nabla \pi(a \mid s) q_\pi(s, a)
\end{aligned}
$$

最终得到的梯度计算公式为

$$
\begin{aligned}
\nabla_o J(\theta) &= \nabla_o \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi_o(a \mid s) \\
&\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \nabla_o \pi_o(a \mid s)
\end{aligned}
\tag{19.4.1}
$$

19.4.3 REINFORCE: 蒙特卡洛策略梯度算法

REINFORCE 算法依靠蒙特卡洛方法采样出的样本轨迹从而估计回报，进而通过梯度上升来更新策略的参数，具体的蒙特卡洛策略梯度公式推导如下：

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi\left[\sum_a \pi\left(a \mid S_t, \boldsymbol{\theta}\right) q_\pi\left(S_t, a\right) \frac{\nabla \pi\left(a \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(a \mid S_t, \boldsymbol{\theta}\right)}\right] \\
&= \mathbb{E}_\pi\left[q_\pi\left(S_t, A_t\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}\right] \\
&= \mathbb{E}_\pi\left[G_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}\right)}\right]
\end{aligned}
\tag{19.4.2}
$$

其中 G_t 是通过采样轨迹的累计奖励得到的对回报的估计，通过计算回报以后就可以直接更新策略的参数, 参数更新规则如下：

$$
\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}
$$

请注意，REINFORCE 算法使用的是时间 t 的累积回报，包括直到轨迹结束的所有未来奖励。从这个意义上说，REINFORCE 算法就是蒙特卡洛算法，并且仅在情节的情况下定义良好，所有更新都在轨迹结束后进行回顾。整个的算法流程也十分简洁：

REINFORCE: 蒙特卡洛策略梯度算法

1. 随机初始化策略网络的参数 θ
2. 通过蒙特卡洛方法采样生成当前策略 π_θ 的一条完整的轨迹：$S_1, A_1, R_2, S_2, A_2, ..., S_T$
3. 对于每个时间步 $t = 1, \cdots, T$ ；
   4. 计算累积回报 $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
   5. 通过梯度上升更新参数：$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi_\theta \left(A_t \mid S_t\right)$

19.4.4 带基线的 REINFORCE 算法

对于上述的 REINFORCE 算法一个直接的改进是从 G_t 中减去一个基准值 b_t 用来在保证偏差不变的情况下减少估计梯度产生的方差，一个重要的例子是我们用动作状态值函数减去状态值函数，这样实际中我们使用优势值：$A(s,a) = Q(s,a) - V(s)$ 进行梯度上升来更新参数。

我们引入一个基准函数到目标函数中，此时梯度可以表示为

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s,a) - b(s)\right) \nabla \pi(a \mid s, \boldsymbol{\theta})$$

基准函数可以是任何函数，甚至是一个随机变量，只要它不随动作 a 变化，这个方程仍然有效，因为减去的量是零：

$$\sum_{\bar{a}} b(s) \nabla \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla \sum_{\bar{a}} \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0$$

此时，新的参数更新规则如下：

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \left(G_t - b\left(S_t\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}$$

我们最终得到的更新规则是一个包含一般基线的新版本的 REINFORCE。因为基线可以一致为零，所以这个更新是一个严格的一般化 REINFORCE。

实际中，基线的一个自然选择是对状态值的估计 $\hat{v}\left(S_t\right)$ $\mathrm{w} \in \mathbb{R}^m$ 是权值向量。因为 REINFORCE 是一种学习策略参数的蒙特卡罗方法，θ，很自然地使用蒙特卡罗方法学习状态值权重，w。

我们通常采用 $\hat{v}\left(S_t, w\right)$ 来作为 $b\left(S_t\right)$, 最后对应的算法如下：

带基线的 REINFORCE 算法

1. 随机初始化策略网络的参数 θ 和值函数网络参数 w
2. 通过蒙特卡洛方法采样生成当前策略 π_θ 的一条完整的轨迹：$S_1, A_1, R_2, S_2, A_2, ..., S_T$
3. 对于每个时间步 $t = 1, \cdots, T$ ；
    4. 计算累积回报 $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    5. 计算带基线后的差值：$\delta \leftarrow G - \hat{v}\left(S_t, w\right)$
    6. 更新值函数网络的参数：$\mathrm{w} \leftarrow w + \alpha^w \delta \nabla \hat{v}\left(S_t, w\right)$
    7. 通过梯度上升更新策略参数：$\theta \leftarrow \theta + \alpha \gamma^t \delta \nabla_\theta \ln \pi_\theta\left(A_t \mid S_t\right)$

19.4.5 演员-评论家 Actor-Critic 及其改进算法

原始的带基线的 REINFORCE 算法需要借助蒙特卡洛方法估计回报 G_t，这对于具有在线增量学习的强化学习任务来说未必是适用的，很自然的改进方式是用时序差分方法代替蒙特卡洛方法进行在线学习，这诱导出了演员-评论家原始算法，本节给出了该重要框架的介绍，以及目前主流的基于演员-评论家的衍生算法的简单描述。

演员-评论家 Actor-Critic 算法

演员-评论家 Actor-Critic 算法可以看成带基线的 REINFORCE 算法的时序差分的版本。具体来说，我们这里用一步回报时序差分代替基于蒙特卡洛方法对整个轨迹的估计：

$$
\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\left(G_{t:t+1} - \hat{v}\left(S_t, \mathbf{w}\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \\
&= \boldsymbol{\theta}_t + \alpha\left(R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right) - \hat{v}\left(S_t, \mathbf{w}\right)\right) \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)} \\
&= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}{\pi\left(A_t \mid S_t, \boldsymbol{\theta}_t\right)}
\end{aligned}
\tag{19.4.3}
$$

一步时序差分方法的主要吸引力在于，它们完全在线且递增，同时避免了有效跟踪的复杂性。它们是有效跟踪方法的特殊情况，不像一般情况那样，但更容易理解。最终我们的演员-评论家 Actor-Critic 算法描述如下：

演员-评论家 Actor-Critic 算法

1. 随机初始化策略网络的参数 θ 和值函数网络参数 w
2. 初始状态 s 以及从初始策略中采样动作 $a \sim \pi_\theta(a|s)$

3. 对于每个时间步 $t = 1, \cdots, T$ ；
    4. 采样回报 $r_t \leftarrow R(s, a)$ 和下一步状态 $s' \leftarrow P(s'|s, a)$
    5. 采样下一个动作 $a' \sim \pi_\theta(a'|s')$
    6. 更新策略参数：$\theta \leftarrow \theta + \alpha_\theta Q_w(s, a)\nabla_\theta \ln \pi_\theta(a \mid s)$
    7. 对于当前步计算 TD 误差：$\delta_t = r_t + \gamma Q_w\left(s', a'\right) - Q_w(s, a)$
    6. 更新值函数网络的参数：$w \leftarrow w + \alpha_w \delta_t \nabla \hat{Q}_w\left(s, a\right)$
    7. 更新当前动作 $a \leftarrow a'$ 和状态 $s \leftarrow s'$

其中是 α_θ 和 α_w 两个预先定义的分别用来更新策略以及值函数的更新步长。

DDPG 算法

DDPG（Lillicrap et al. 2015）是深度确定性策略梯度（Deep Deterministic Policy Gradient）的缩写，是一个结合了 DPG 以及 DQN 的无模型离线演员-评论家算法。从 DQN 这边，DDPG 借入了经验回放机制以及目标网络两个核心思想，不同之处是 DDPG 中演员-评论家的目标网络的参数实行的是软更新（“保守策略迭代”）：

$$\theta' = \tau\theta + (1 - \tau)\theta'$$

其中 $\tau << 1$。：他们修改了 DQN 中最初使用的目标网络的更新频率。在 DQN 中，目标网络每隔几千步就会更新一次训练网络的参数。Lillicrap et al. 发现实际上更好的方法是让目标网络缓慢地跟踪训练过的网络，每次更新训练过的网络后，用行动者和批评家的滑动平均值更新目标网络的参数。使用该更新规则，目标网络总是比训练过的网络 “晚”，为 q 值的学习提供了更大的稳定性，从 DPG 借鉴的关键思想是行为人（Actor）的策略梯度，并通过常规的 Q-learning 和目标网络来学习：

$$J(\varphi) = \mathbb{E}_{s\sim\rho_\mu}\left[\left(r\left(s, a, s'\right) + \gamma Q_{\varphi'}\left(s'\mu_{\theta'}\left(s'\right)\right) - Q_\varphi(s, a)\right)^2\right]$$

另外 DDPG 对于确定的的策略网络的输出加入额外的噪音 ξ 来鼓励对环境的探索。

$$a_t = \mu_\theta\left(s_t\right) + \xi$$

PPO 算法

之前主流的 TRPO 算法的构造相对复杂，我们想要去实现一个类似的约束并且取得更好或者接近的结果，因此，近端策略优化（proximal policy optimization ，PPO）算法就被提出了，该算法在实现相似性能的同时通过使用一个截断的替代目标函数来简化 TRPO。

PPO 算法的设计首先是通过重新整理 TRPO 的替代损失函数为如下形式：

$$L^{\mathrm{CPI}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta\left(s_t, a_t\right)}{\pi_{\theta_{\mathrm{old}}}\left(s_t, a_t\right)} A_{\pi_{\theta_{\mathrm{old}}}}\left(s_t, a_t\right)\right] = \mathbb{E}_t\left[\rho_t(\theta) A_{\pi_{\theta_{\mathrm{old}}}}\left(s_t, a_t\right)\right]$$

其中 A 为优势函数。如果不添加任何约束求解上述优化问题可能会导致训练过程中的不稳定，因此为了增加训练稳定性，我们需要对策略比值的波动进行乘法，使其在一定区间范围内波动，对此我们通过剪切函数将其约束在 $1-\epsilon$ 和 $1+\epsilon$ 之间：

$$L^{\mathrm{CLIP}}(\theta)=\mathbb{E}_t\left[\min\left(\rho_t(\theta)A_{\pi_{\theta_{\mathrm{old}}}}\left(s_t,a_t\right),\mathrm{clip}\left(\rho_t(\theta),1-\epsilon,1+\epsilon\right)A_{\pi_{\theta_{\mathrm{old}}}}\left(s_t,a_t\right)\right)\right]$$

与 TRPO 相比，PPO 的主要优势在于它的简单性: 通过使用随机梯度下降或 Adam 等一阶方法，可以直接使被裁剪目标最大化。PPO 算法在各类游戏中取得了非常优秀的结果，已经成为了优先考虑的强化学习算法之一。

SAC 算法

软演员-评论家算法（Soft Actor-Critic，SAC，Haarnoja et al. 2018）将策略的熵度量纳入回报函数中用以鼓励探索：我们希望学习到一种尽可能随机行动的策略，同时仍然能够在任务中完成目标。它是一个遵循最大熵强化学习框架的离线演员-评论家模型。

基于最大熵 RL 框架，Haarnoja 等通过扩展目标的定义，提出了软 Q-learning：

$$J(\theta)=\sum_{t=1}^{T}\mathbb{E}_{\left(s_t,a_t\right)\sim\rho_{\pi_\theta}}\left[r\left(s_t,a_t\right)+\alpha\mathcal{H}\left(\pi_\theta\left(.\mid s_t\right)\right)\right]$$

策略的训练目标是同时最大化期望累积回报以及策略的熵度量。

在这个基于轨迹的公式中，智能体寻求的策略是最大化完整轨迹的熵，而不是每个状态下策略的熵。这是一个非常重要的区别: 智能体不仅搜索一个具有高熵的策略，而且搜索一个具有高熵的状态的策略，即智能体最不确定的状态。这就允许了非常有效的探索策略，智能体将尝试着减少其对环境的不确定性，并收集比简单地寻找一个好的政策时更多的信息。

Haarnoja et al 提出了软演员-评论家算法 (SAC)，允许有随机行为者 (与 DDPG 相反)，同时比政策上的方法 (如 PPO) 更优，样本效率更高。

SAC 基于软 Q-learning 来实现这些改进。准确来说，SAC 旨在学习三个函数：

1. 由 θ 参数化的策略函数 π_θ。
2. 由 w 参数化的软 Q 值函数 Q_w。
3. 由 ψ 参数化的软 Q 值函数 V_ψ。

通过基于最大熵策略的原则下对三个网络进行交互训练，SAC 算法也取得了非常令人瞩目的效果。具体的训练过程可以参考相应的论文。

参考文献

[1] Huang Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. Data mining and knowledge discovery, 2(3), 283-304.

[2] Pelleg D, & Moore A. (2000). X-means: extending K-means with efficient estimation of the number of the clusters[C].Proceedings of the 17th International Conference on Machine Learning. NewYork: ACM, 111-117.

[3] Madan S, & Dana KJ. (2016). Modified balanced iterative reducing and clustering using hierarchies (m-BIRCH) for visual clustering. Pattern Analysis and Applications, 19(4), 1023-1040.

[4] Trisminingsih R, & Shaztika SS. (2016). ST-DBSCAN clustering module in SpagoBI for hotspots distribution in Indonesia[C]//Proceedings of the 3rd International Conference on Information Technology. New York: ACM, 60-67.

[5] Ankerst M, Breunig M, Kriegel HP, & Sander J. (1999). OPTICS: Ordering points to identify the clustering structure. ACM Sigmod record, 28(2), 49-60.

[6] McParland D, & Gormley IC. (2016). Model based clustering for mixed data: clustMD. Advances in Data Analysis and Classification, 10(2), 155-169.

[7] Foss A, Markatou M, Ray B, & Heching A. (2016). A semiparametric method for clustering mixed data. Machine Learning, 105(3), 419-458.

[8] Dhillon IS. (2001). Co-clustering documents and words using bipartite spectral graph partitioning [A]. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining [C] ACM, 269-274.

[9] Dhillon I, Mallela S, & Modha D. (2003). Information-theoretic co-clustering [A]. In: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining [C]. ACM, 89-98.

[10] Chi EC, Allen GI, & Baraniuk RG. (2016). Convex biclustering . Biometrics, 73(1): 10-19.

[11] Pearson K. (1901). Principal components analysis. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 6(2), 559.

[12] Hotelling H. (1933). Analysis of a complex of statistical variables into principal components. Journal of educational psychology, 24(6), 417.

[13] Van der Maaten L, & Hinton G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).

[14] Hinton GE, & Roweis ST. (2003). Advances in Neural Information Processing Systems, Vol. 15.

[15] Zou H, Hastie T, & Tibshirani R. (2006). Sparse principal component analysis. Journal of computational and graphical statistics, 15(2), 265-286.

[16] Tucker LR. (1963). Implications of factor analysis of three way matrices for measurement of change. Problems in measuring change. Wisconsin:University of Wisconsin Press.

[17] Levin J. (1963). Three-mode factor analysis[D]. Urbana: University of Illinois.

[18] Wright J, Peng YG, Ma Y, et al. (2009). Robust principal component analysis: exact recovery of corrupted low-rank matrices by convex optimization[C]//Proceedings of Neural Information Processing Systems. Whistler: MIT Press.

[19] 李航. (2019). 统计学习方法 (第 2 版). 北京: 清华大学出版社.

[20] 范金城，梅长林. (2002). 数据分析. 北京: 科学出版社.

[21] 高惠璇. (2005). 应用多元统计分析. 北京：北京大学出版社.

[22] 何晓群. (2015). 多元统计分析. 北京：中国人民大学出版社.

[23] 任雪松，于秀林. (2011). 多元统计分析. 北京: 中国统计出版社.

[24] 王斌会. (2014). 多元统计分析及 R 语言建模. 广州: 暨南大学出版社.

[25] 王学民. (2017). 应用多元统计分析. 上海: 上海财经大学出版社.

[26] 袁志发，周静芋. (2002). 多元统计分析. 北京：科学出版社.

[27] 张润楚. (2006). 多元统计分析. 北京：科学出版社.

[28] 何晓群. (2017). 应用回归分析 (第 5 版). 北京: 电子工业出版社.

[29] 王黎明. (2019). 回归分析-概念、方法和应用. 上海: 上海财经大学出版社.

[30] 刘超. (2019). 回归分析-方法、数据与 R 的应用. 北京: 高等教育出版社.

[31] 唐年胜, 李会琼. (2014). 应用回归分析. 北京: 科学出版社.

[32] 王衡军. (2020). 机器学习. 北京: 清华大学出版社.

[33] Horel AE. (1962). Applications of ridge analysis toregression problems. Chem. Eng. Progress., 58, 54-59.

[34] Golub GH, Heath M, & Wahba G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. Technometrics, 21(2), 215-223.

[35] Bonat WH. (2018). Multiple response variables regression models in R: The mcglm package. Journal of Statistical Software, 84, 1-30.

[36] Bonat WH, Walmes Marques Zeviani, & Fernando de Pol Mayer. (2018). Package 'mcglm'. https://github.com/wbonat/mcglm.

[37] Bonat WH. (2018). mcglm: Multivariate Covariance Generalized Linear Models. https://CRAN.R-project.org/package=mcglm.

[38] Anderson TW. (1973). Asymptotically Efficient Estimation of Covariance Matrices with Linear Structure. The Annals of Statistics, 1(1), 135–141.

[39] Pourahmadi M. (2000). Maximum Likelihood Estimation of Generalised Linear Models for Multivariate Normal Covariance Matrix. Biometrika, 87(2), 425–435.

[40] Bonat WH, & Jørgensen B. (2016). Multivariate Covariance Generalized Linear Models. Journal of the Royal Statistical Society C, 65(5), 649–675.

[41] Martinez-Beneito MA. (2013). A General Modelling Framework for Multivariate Disease Mapping. Biometrika, 100(3), 539–553.

[42] Liang KY, & Zeger SL. (1986). Longitudinal Data Analysis Using Generalized Linear Models. Biometrika, 73(1), 13–22.

[43] Jørgensen B, & Knudsen SJ. (2004). Parameter Orthogonality and Bias Adjustment for Estimating Functions. Scandinavian Journal of Statistics, 31(1), 93–114.

[44] Bates D, & Maechler M. (2017). Matrix: Sparse and Dense Matrix Classes and Methods. R package version 1.2-12. https://CRAN.R-project.org/package=Matrix.

[45] Bivand R. (2017). spdep: Spatial Dependence: Weighting Schemes, Statistics and Models. R package version 0.7-4. https://CRAN.R-project.org/package=spdep.

[46] Bonat WH. (2018). mcglm: Multivariate Covariance Generalized Linear Models. R package version 0.4.0. https://CRAN.R-project.org/package=mcglm.

[47] Carey VJ. (2015). gee: Generalized Estimation Equation Solver. R package version 4.13-19. https://CRAN.R-project.org/package=gee.

[48] Hadfield JD. (2010). MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package. Journal of Statistical Software, 33(2), 1–22.

[49] Højsgaard S, Halekoh U, & Yan J. (2006). The R Package geepack for Generalized Estimating Equations. Journal of Statistical Software, 15(2), 1–11.

[50] Martin AD, Quinn KM, & Park JH. (2011). "MCMCpack: Markov Chain Monte Carlo in R." Journal of Statistical Software, 42(9), 1–22.

[51] Masarotto G, & Varin C. (2017). Gaussian Copula Regression in R. Journal of Statistical Software, 77(8), 1–26.

[52] McDaniel LS, Henderson NC, & Rathouz PJ. (2013). Fast Pure R Implementation of GEE: Application of the Matrix Package. The R Journal, 5(1), 181–187.

[53] Pinheiro J, Bates D, DebRoy S, Sarkar D, & R Core Team. (2017). nlme: Linear and Nonlinear Mixed Effects Models. R package version 3.1-131. https://CRAN.R-project.org/ package=nlme.

[54] R Core Team. (2017). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

[55] Ribeiro Jr PJ, & Diggle PJ. (2016). geoR: Analysis of Geostatistical Data. R package version 1.7-5.2. https://CRAN.R-project.org/package=geoR.

[56] Rönnegård L, Shen X, & Alam M. (2010). Fitting Hierarchical Generalized Linear Models. The R Journal, 2(2), 20–28. https://journal.R-project.org/archive/2010/RJ-2010-009.

[57] Wolak ME. (2012). nadiv: An R Package to Create Relatedness Matrices for Estimating Non-Additive Genetic Variances in Animal Models. Methods in Ecology and Evolution, 3(5), 792–796.

[58] Vapnik V. (1963). Pattern recognition using generalized portrait method. Automation and remote control, 24, 774-780.

[59] Tang Y, Jin B, Zhang Y, et al. (2004). Granular support vector machines for medical binary classification problems[C]//Proceedings of the IEEE CIBIB. Piscataway, HJ: IEEE Computationl Intelligence Society, 73-78.

[60] Tang Y, Jin B, & Zhang Y. (2005). Granular support vector machines with association rules mining for protein homology prediction[J]. Artificial Intelligence in Medicine, 35: 121-134.

[61] Lin C, & Wang S. (2002). Fuzzy support vector machines[J]. IEEE Transactions on Neural Networks, 3(2): 464-471.

[62] Herbrich R, Graepel T, & Obermayer K. (2000). Large margin rank boundaries for ordinal regression[C]//Advances in Large Margin Classifiers. Cambridge, MA: MIT Press, 7:115-132.

[63] Jayadcva R, & Khemchandani SC. (2007). Twin support vector machines for pattern classification[J]. IEEE Trans On Pattern Analysis and Machine Intelligence,29(5): 905-910.

[64] Kumar MA, & Gopal M. (2009). Least squares twin support vector machines for pattern classification[J]. Expert Systems with Applications, 36(4):7535-7543.

[65] Ye Q, Zhao C, Gao S, et al. (2012). Weighted twin support vector machines with local information and its application[J]. Neural Networks the Official Journal of the International Neural Net work Society, 35(11):31-39.

[66] Vapnik V N. (2000). 统计学习理论的本质 [M]. 张学工, 译. 北京: 清华大学出版社.

[67] Breiman L, Friedman J, Olshen R, & Stone C. (1984). Classification and Regression Trees. Chapman and Hall, New York.

[68] Shannon CE. (1948). A mathematical theory of communication. The Bell system technical journal, 27(3), 379-423.

[69] Quinlan JR. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

[70] Hastie T, Tibshirani R, & Friedman J. (2008). The Elements of Statistical Learning: Data Mining, Inferene and Prediction. Springer, New York.

[71] Kuhn M, & Johnson K. (2013). Applied Predictive Modeling. Springer, New York.

[72] Friedman JH. (1991). Multivariate adaptive regression splines. The annals of statistics, 19(1), 1-67.

[73] Vovk V, Gammerman A, & Shafer G. (2005). Conformal prediction. Algorithmic learning in a random world, 17-51.

[74] Vovk V, Nouretdinov I, & Gammerman A. (2009). On-line predictive linear regression. The Annals of Statistics, 1566-1590.

[75] Lei J, Robins J, & Wasserman L. (2013). Distribution-free prediction sets. Journal of the American Statistical Association, 108(501), 278-287.

[76] Lei J, & Wasserman L. (2014). Distribution-free prediction bands for non-parametric regression. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 76(1), 71-96.

[77] Lei J, G' Sell M, Rinaldo A, Tibshirani RJ, & Wasserman L. (2018). Distribution-free predictive inference for regression. Journal of the American Statistical Association, 113(523), 1094-1111.

[78] Lei J, Rinaldo A, & Wasserman L. (2015). A conformal prediction approach to explore functional data. Annals of Mathematics and Artificial Intelligence, 74(1), 29-43.

[79] Butler R, & Rothman ED. (1980). Predictive intervals based on reuse of the sample. Journal of the American Statistical Association, 75(372), 881-889.

[80] Steinberger L, & Leeb H. (2016). Leave-one-out prediction intervals in linear regression models with many variables. arXiv preprint arXiv:1602.05801.

[81] Akaike H. (1974). A new look at the statistical model identification. IEEE transactions on automatic control, 19(6), 716-723.

[82] Schwarz G. (1978). Estimating the dimension of a model. The annals of statistics, 461-464.

[83] Hoerl AE, & Kennard RW. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.

[84] Fan J, & Li R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96(456), 1348-1360.

[85] Zhang CH. (2010). Nearly unbiased variable selection under minimax concave penalty. The Annals of statistics, 38(2), 894-942.

[86] Zou H, & Hastie T. (2005). Regularization and variable selection via the elastic net. Journal of the royal statistical society: series B (statistical methodology), 67(2), 301-320.

[87] Yuan M, & Lin Y. (2006). Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 68(1), 49-67.

[88] Huang J, Ma S, Xie H, & Zhang CH. (2009). A group bridge approach for variable selection. Biometrika, 96(2), 339-355.

[89] Breheny P, & Huang J. (2009). Penalized methods for bi-level variable selection. Statistics and its interface, 2(3), 369.

[90] Simon N, Friedman J, Hastie T, & Tibshirani R. (2013). A sparse-group lasso. Journal of computational and graphical statistics, 22(2), 231-245.

[91] Fang K, Wang X, Zhang S, Zhu J, & Ma S. (2015). Bi-level variable selection via adaptive sparse group Lasso. Journal of Statistical Computation and Simulation, 85(13), 2750-2760.

[92] Fan J, & Lv J. (2008). Sure independence screening for ultrahigh dimensional feature space. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 70(5), 849-911.

[93] Hall P, & Miller H. (2009). Using generalized correlation to effect variable selection in very high dimensional problems. Journal of Computational and Graphical Statistics, 18(3), 533-550.

[94] Li G, Peng H, Zhang J, & Zhu L. (2012). Robust rank correlation based screening. The Annals of Statistics, 40(3), 1846-1877.

[95] Zhu LP, Li L, Li R, & Zhu LX. (2011). Model-free feature screening for ultrahigh-dimensional data. Journal of the American Statistical Association, 106(496), 1464-1475.

[96] Li R, Zhong W, & Zhu L. (2012). Feature screening via distance correlation learning. Journal of the American Statistical Association, 107(499), 1129-1139.

[97] Székely GJ, Rizzo ML, & Bakirov NK. (2007). Measuring and testing dependence by correlation of distances. The annals of statistics, 35(6), 2769-2794.

[98] Fan J, & Fan Y. (2008). High dimensional classification using features annealed independence rules. Annals of statistics, 36(6), 2605.

[99] Mai Q, & Zou H. (2013). The Kolmogorov filter for variable screening in high-dimensional binary classification. Biometrika, 100(1), 229-234.

[100] Cui H, Li R, & Zhong W. (2015). Model-free feature screening for ultrahigh dimensional discriminant analysis. Journal of the American Statistical Association, 110(510), 630-641.

[101] Xie J, Lin Y, Yan X, & Tang N. (2020). Category-adaptive variable screening for ultra-high dimensional heterogeneous categorical data. Journal of the American Statistical Association, 115(530), 747-760.

[102] Fan J, & Song R. (2010). Sure independence screening in generalized linear models with NP-dimensionality. The Annals of Statistics, 38(6), 3567-3604.

[103] Fan J, Feng Y, & Song R. (2011). Nonparametric independence screening in sparse ultra-high-dimensional additive models. Journal of the American Statistical Association, 106(494), 544-557.

[104] McCulloch WS, & Pitts W. (1943). Bulletin of Mathematical Biophysics.

[105] Kohonen T. (1988). An introduction to neural computing.Neural Networks.

[106] 周志华.(2016). 机器学习. 北京：清华大学出版社.

[107] Rosenblatt F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[108] Werbos P. (1974). Beyond regression:" new tools for prediction and analysis in the behavioral sciences. Ph. D. dissertation, Harvard University.

[109] Pineda FJ. (1987). Generalization of Back-Propagation to recurrent neural-networks.Physical Review Letters.

[110] Aarts E, & Korst J. (1989). Simulated Annealing and Boltzmann Machines:A Stochastic Approach to Combinatorial Optimization and Neural Computing. New York: John Wiley & Sons.

[111] Goldberg DE. (1989). Genetic Algorithms in Search, Optimizaiton and Machine Learning. Boston:Addison-Wesley.

[112] Carpenter GA, & Grossberg S. (1987). A massively parallel architecturefor a self-organizing neural pattern recognition machine. Computer Vision,Graphics,and Image Processing.

[113] Kohonen T. (1982). Self-organized formation of topologically correct featuremaps. Biological Cybernetics.

[114] Fahlman SE, & Lebiere C. (1990). The cascade-correlation learning architecture. Pittsburgh:chool of Computer Sciences,Carnergie Mellon University.

[115] Elman JL. (1990). Finding structure in time. Cognitive Science.

[116] LeCun Y, & Bengio Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10).

[117] LeCun Y, Bottou L, Bengio Y, & Haffner P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[118] Rumelhart DE, Hinton GE, & Williams RJ. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[119] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, & Jackel LD. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

[120] Hinton GE, & Salakhutdinov RR. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786), 504-507.

[121] Zeiler MD, & Fergus R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

[122] Simonyan K, & Zisserman A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[123] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, ... & Rabinovich A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[124] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, & Teller E. (1953). Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6), 1087-1092.

[125] Breiman L. (1996). Heuristics of Instability and Stabilization in Model Selection. The Annals of Statistics,26,2350-2383.

[126] Loh WY, & Shih YS. (1997). Split Selection Methods for Classification Trees. Statistica Sinica,7,815-840.

[127] Carolin C, Boulesteix A, & Augustin T. (2007). Unbiased Split Selection for Classification Trees Based on the Gini Index. Computational Statistics & Data Analysis,52,483-501.

[128] Loh WY. (2010). Tree-Structured Classifiers. Computational Statistics,2,364-369.

[129] Beriman L. (2001). Random Forests. Machine Learning,45,5-32.

[130] Tibshirani R. (1996). Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society Series B,58,267-288.

[131] Quinlan R. (1992). Learning with Continuous Classes. Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, 343-348.

[132] Zeileis A, Hothorn T, & Hornik K. (2008). Model-Based Recursive Partitioning. Journal of Computational and Graphical Statistics, 14,185-205.

[133] Quinlan JR. (2014). C4. 5: programs for machine learning[M]. Elsevier.

[134] Breiman L. (1996). Bagging Predictors. Machine Learning,24,123-140.

[135] Valiant LG. (1984). A theory of the learnable. Communications of the ACM, 27(11), 1134-1142.

[136] Kearns M, & Valiant LG. (1988). Learning Boolean formulae or finite automata is as hard as factoring. Technical Report TR-14-88, Harvard University Aiken Computation Laboratory.

[137] Schapire RE. (1990). The strength of weak learnability. Machine learning, 5(2), 197-227.

[138] Freund Y. (1995). Boosting a weak learning algorithm by majority. Information and computation, 121(2), 256-285.

[139] Drucker H, Schapire R, & Simard P. (1993). Boosting performance in neural networks. International Journal of Pattern Recognition and Artificial Intelligence, 7(4),705-719.

[140] Zhou ZH, Yang Y, Wu XD, & Kumar V. (2009). The Top Ten Algorithms in Data Mining. New York, USA: CRC Press, 127-149.

[141] Friedman J, Hastie T, & Tibshirani R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). The annals of statistics, 28(2), 337-407.

[142] Freund Y, & Schapire RE. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), 119-139.

[143] Schapire RE, & Singer Y. (1999). Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297-336.

[144] Schapire RE, & Freund Y. (2012). Boosting: Foundations and Algorithms. MIT Press, Cambridge, MA.

[145] Schapire RE, Freund Y, Bartlett P, & Lee WS. (1998). Boosting the margin: A new explanation for the effectiveness of votingmethods. Annals of Statistics, 26(5):1651-1686.

[146] Breiman L. (1999). Prediction games and arcing algorithms. Neural Computation, 11(7): 1493-1517.

[147] Reyzin L, & Schapire RE. (2006). How Boosting the margin can also boost classifier complexity. In: Proceedings of the 23rd International Conference on Machine Learning. New York,USA: ACM, 753-760.

[148] Wang LW, Sugiyama M, Yang C, Zhou ZH, & Feng JF. (2008). On the margin explanation of Boosting algorithms. In: Proceedings of the 21st Annual Conference on Learning Theory. Madison, USA: Omni Press, 479-490.

[149] Gao W, & Zhou ZH. (2013). On the doubt about margin explanation of boosting. Artificial Intelligence, 203:1-18.

[150] Mason L, Baxter J, Bartlett P, & Frean M. (2000). Functional gradient techniques for combining hypotheses. Advances in Large Margin Classifiers. Cambridge, MA: MIT Press, 221-247.

[151] Mason L, Baxter J, Bartlett P, & Frean M. (1999). Boosting algorithms as gradient descent. Advances in Neural Information Processing Systems, 12: 512-518.

[152] Friedman J. (2001). Greedy function approximation: A gradient boosting machine, Annals of Statistics, 29(5): 1189-1232.

[153] Q. Chen (陈强). (2020). 机器学习及 R 应用. 高等教育出版社.

[154] R.G. Wang (汪荣贵). (2020). 机器学习简明教程. 机械工业出版社.

[155] Bates JM, & Granger CW. (1969). The combination of forecasts. Journal of the operational research society, 20(4), 451-468.

[156] Zinkevich M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In Proceedings of the 20th international conference on machine learning (icml-03) (pp. 928-936).

[157] Gustafson JL. (1988). Reevaluating Amdahl's law. Communications of the ACM, 31(5), 532-533.

[158] threading-基于线程的并行.https://docs.python.org/zh-cn/3/library/threading.html.

[159] multiprocessing-基于进程的并行.https://docs.python.org/zh-cn/3/library/multiprocessing.html.

[160] CUDA C++ Programming Guide.https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[161] 樊哲勇. (2020). CUDA 编程：基础与实践 [M] . 北京：清华大学出版社.

[162] 詹卡洛 · 扎克内. (2021). Python 并行编程实战（第二版）[M] . 北京：中国电力出版社.

[163] 陈华. (2018). 多核并行计算 [M] . 青岛：中国石油大学出版社.

[164] Bray, C. W. (1928). Transfer of learning. Journal of Experimental Psychology, 11(6):443.

[165] Wang, J., Feng, W., Chen, Y., Yu, H., Huang, M., and Yu, P. S. (2018b). Visual domain adaptation with manifold embedded distribution alignment. In ACMMM, 402–410.

[166] Wang, J., Chen, Y., Feng, W., Yu, H., Huang, M., and Yang, Q. (2020). Transfer learning with dynamic distribution adaptation. ACM TIST, 11(1):1–25.

[167] Ben-David, S., Blitzer, J., Crammer, K., Pereira, F., et al. (2007). Analysis of representations for domain adaptation. In NIPS, volume 19.

[168] Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT Press.

[169] Feng, J., Huang, M., Zhao, L., Yang, Y., and Zhu, X. (2018). Reinforcement learning for relation classification from noisy data. In Thirty-Second AAAI Conference on Artificial Intelligence.

[170] Liu, M., Song, Y., Zou, H., and Zhang, T. (2019). Reinforced training data selection for domain adaptation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 1957–1968.

[171] Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in NLP. In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, 264–271.

[172] Huang, J., Smola, A. J., Gretton, A., Borgwardt, K. M., Schölkopf, B., et al. (2007). Correcting sample selection bias by unlabeled data. Advances in Neural Information Processing Systems, 19:601.

[173] Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schölkopf, B., and Smola, A. J. (2006).Integrating structured biological data by kernel maximum mean discrepancy.Bioinformatics,22(14):e49–e57.

[174] Dorri, F. and Ghodsi, A. (2012). Adapting component analysis. In Data Mining (ICDM), 2012 IEEE 12th International Conference on, 846–851.

[175] Duan, L., Tsang, I. W., and Xu, D. (2012). Domain transfer multiple kernel learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(3):465–479.

[176] Gretton, A., Sejdinovic, D., Strathmann, H., Balakrishnan, S., Pontil, M., Fukumizu, K., and Sriperumbudur, B. K. (2012). Optimal kernel choice for large-scale two-sample tests. In Advances in neural information processing systems, 1205–1213.

[177] Sun, B. and Saenko, K. (2015). Subspace distribution alignment for unsupervised domain adaptation. In BMVC, 24–1.

[178] Sun, B. and Saenko, K. (2016). Deep CORAL: Correlation alignment for deep domain adaptation. In ECCV, 443–450.

[179] Seung, H. S. and Lee, D. D. (2000). The manifold ways of perception. Science, 290(5500):2268–2269.

[180] Zhou, Z.-h. (2016). Machine learning. Tsinghua University Press.

[181] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

[182] Greene, R. E. and Jacobowitz, H. (1971). Analytic isometric embeddings. Annals of Mathematics, 189–204.

[183] Belkin, M., Niyogi, P., and Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research, 7(Nov):2399–2434.

[184] Hamm, J. and Lee, D. D. (2008). Grassmann discriminant analysis: a unifying view on subspace-based learning. In ICML, 376–383. ACM.

[185] Gopalan, R., Li, R., and Chellappa, R. (2011). Domain adaptation for object recognition: An unsupervised approach. In ICCV, 999–1006. IEEE.

[186] Baktashmotlagh, M., Harandi, M. T., Lovell, B. C., and Salzmann, M. (2014). Domain adaptation on the statistical manifold. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2481–2488.

[187] Wang, J., Feng, W., Chen, Y., Yu, H., Huang, M., and Yu, P. S. (2018). Visual domain adaptation with manifold embedded distribution alignment. In ACMMM, 402–410.

[188] Qin, X., Chen, Y., Wang, J., and Yu, C. (2019). Cross-dataset activity recognition via adaptive spatial-temporal transfer learning. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 3(4):1–25.

[189] Baktashmotlagh, M., Harandi, M. T., Lovell, B. C., and Salzmann, M. (2014). Domain adaptation on the statistical manifold. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2481–2488.

[190] Guerrero, R., Ledig, C., and Rueckert, D. (2014). Manifold alignment and transfer learning for classification of Alzheimer's disease. In International Workshop on Machine Learning in Medical Imaging, 77–84. Springer.

[191] Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2016). Optimal transport for domain adaptation. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[192] Courty, N., Flamary, R., and Tuia, D. (2014). Domain adaptation with regularized optimal transport. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 274–289. Springer.

[193] Courty, N., Flamary, R., Habrard, A., and Rakotomamonjy, A. (2017). Joint distribution optimal transportation for domain adaptation. In Advances in Neural Information Processing Systems, 3730–3739.

[194] Xu, R., Liu, P., Zhang, Y., Cai, F., Wang, J., Liang, S., Ying, H., and Yin, J. (2020b). Joint partial optimal transport for open set domain adaptation. In International Joint Conference on Artificial Intelligence, 2540–2546.

[195] Xu, R., Liu, P., Wang, L., Chen, C., and Wang, J. (2020a). Reliable weighted optimal transport for unsupervised domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4394–4403.

[196] Bhushan Damodaran, B., Kellenberger, B., Flamary, R., Tuia, D., and Courty, N. (2018). DeepJDOT: Deep joint distribution optimal transport for unsupervised domain adaptation. In Proceedings of the European Conference on Computer Vision (ECCV), 447–463.

[197] Lee, C.-Y., Batra, T., Baig, M. H., and Ulbricht, D. (2019). Sliced Wasserstein discrepancy for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 10285–10295.

[198] Lu, W., Chen, Y., Wang, J., and Qin, X. (2021). Cross-domain activity recognition via substructural optimal transport. Neurocomputing, 454:65–75.