# 第3章　机器学习基础

机器学习作为实现人工智能的一种手段，近年来日益流行。而本书的重点——深度学习，也正是实现机器学习的一种重要技术。因此，了解机器学习中的一些概念和算法对于理解深度学习算法有很大的帮助。在这一章，我们将介绍这些机器学习中的重要概念、数据处理方法、评价指标等，人工智能、机器学习和深度学习之间的关系，如图 3.1 所示。

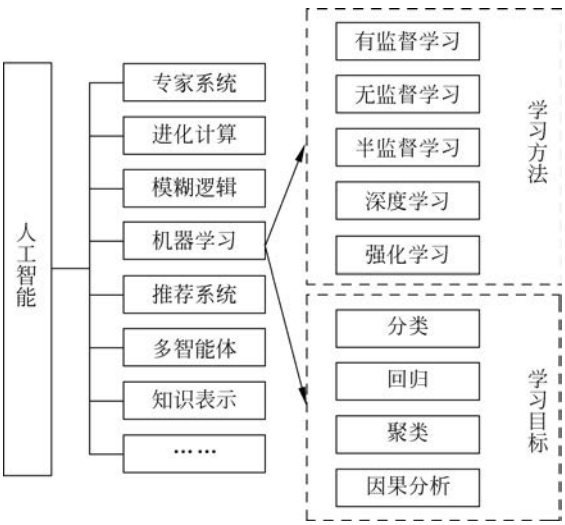


图 3.1　人工智能、机器学习和深度学习之间的关系

## 3.1　机器学习概述

### 3.1.1　机器学习定义与基本术语

首先，还是从人工智能出发来介绍机器学习。人工智能是一门研究用于模拟、延伸和拓展人的智能的理论和方法的学科。根据实现效果，可以将人工智能分为强人工智能和弱人

工智能。强人工智能是指机器能够实现推理、独立思考、解决未知问题并且拥有自我意识和价值观；弱人工智能是指机器不能真正实现自我思考、推理和解决问题，它们只是看起来像拥有了智能。虽然科幻电影中大多描绘的是强人工智能，但是目前人们做出的努力只是集中在弱人工智能部分，只能赋予机器感知环境的能力。而这部分的成功主要归功于一种实现人工智能的方法——机器学习。

**机器学习**(**Machine Learning**，**ML**)就是让机器通过学习数据来获得某种知识，从而获得解决问题的能力。从学科的角度出发，机器学习往往指一类通过学习数据来完成任务的算法。其实，这种通过学习数据来解决问题的思路还是源于人思考的方式。我们经常会听到很多的俗语，例如“朝霞不出门，晚霞行千里”“瑞雪兆丰年”“干冬湿年”等，这些都体现了从古至今人类的智慧。那么为什么朝霞出现就会下雨，晚霞出现天气就会晴朗呢？原因就在于人具有很强大的归纳能力，根据每天的观察和总结，慢慢“训练”出了这样一种分辨是否下雨的“分类器”。

针对机器学习的定义，Mitchell 给出了一个更形式化的说明：对于一个**任务**(**Task**) T 和**性能指标**(**Performance Metric**) P，如果程序通过**经验**(**Experience**) E 在任务 T 上的指标 P 获得了提升，那么我们就说针对 T 和 P，程序对 E 进行了学习。这个定义可能比较拗口，表 3.1 列举了几个例子来帮助理解。

表 3.1 机器学习中的任务、性能指标和经验

| | | |
|---|---|---|
| 实例 1 | T | 下象棋 |
| | P | 对弈任意对手的胜率 |
| | E | 与自己不断对战 |
| 实例 2 | T | 识别人脸 |
| | P | 识别结果的正确率、误检率和漏检率 |
| | E | 人工标定的图片数据集 |
| 实例 3 | T | 自动驾驶 |
| | P | 从出发点到目的地的碰撞次数、行驶时间、耗油量等 |
| | E | 有驾驶规则的行驶环境数据集 |
| 实例 4 | T | 通过面部观察判断罪犯 |
| | P | 识别结果的正确率、误检率和漏检率 |
| | E | 包含罪犯与非罪犯面部照片的数据集 |

了解了机器学习的定义之后，再来关注所有机器学习算法都会涉及的一些概念。以“预测下雨”为例，在预测之前，我们肯定需要获取一些**特征**(**Feature**)或**属性**(**Attribute**)，比如是否出现了朝霞、是否出现了晚霞、温度、空气湿度、云量，甚至是卫星云图，等等。通常，为了能够进行数学计算，我们需要将这些特征表示为一个 $d$ 维的**特征向量**(**Feature Vector**)，记作 $\boldsymbol{x}=[x_1,x_2,\cdots,x_d]^{\mathrm{T}}$，向量的每一个维度代表一个特征，总共选取了 $d$ 个特征。

这样的特征有无穷多种，但是并不是每一种都对最终的判断有帮助。所以，为了通过学习来了解哪些特征是有帮助的，以及这些特征取哪些值时会下雨，我们还要获得它们对应的**标签**(**Label**)。标签可以是连续值，比如下雨量、下雨持续时间等；标签也可以是离散的，比如是否会下雨。标签的选取通常与需要完成的任务有关。当标签是连续值时，这样的机器学习任务称为**回归**(**Regression**)问题；当标签是有限数量的离散值时，这样的机器学习任务

称为**分类**(**Classification**)问题；当标签是标记序列时，这样的机器学习任务称为**标注**(**Tagging**)问题。标注问题可以看成是分类问题的一种。

一组记录好的特征值以及它的标签称为一个**样本**(**Sample**)或**实例**(**Instance**)，例如(特征：(出现朝霞、没有出现晚霞、空气湿度为50%)，标签：(下雨))。一组样本构成的集合称为**数据集**(**Dataset**)。

现在再回顾机器学习的定义，为了能够在任务T上提高性能P，需要学习某种经验E。这里，需要学习的就是数据集，而为了确定性能P是否能够提高，还需要一个不同的数据集来测量性能P。因此，数据集需要分为两部分，用于学习的数据集称为**训练集**(**Training Set**)，用于测试最终性能P的数据集称为**测试集**(**Test Set**)。为了保证学习的有效性，我们需要保证这两个集合不相交。

数据集中的样本还需要保证一个基本的特性——**独立同分布**(**Identically and Independently Distributed，IID**)假设，即每一个样本都需要独立地从相同的数据分布中提取。“独立”保证了任意两个样本之间不存在依赖关系；“同分布”保证了数据分布的统一，从而在训练集上的训练结果对于测试集也是适用的。例如，当训练集的数据都是“地球的天气”，而测试集中都是“火星的天气”，这很显然是不合理的。

机器学习的重点是如何更好地利用这些数据。给定训练集，我们希望算法能够拟合一个函数$f(\boldsymbol{x},\boldsymbol{\theta})$来完成从输入特征向量到标签的映射。对于连续的标签或者非概率模型，我们通常会直接拟合标签的值：

$$\hat{y}=f(\boldsymbol{x},\boldsymbol{\theta})$$

其中，$\boldsymbol{\theta}$为算法模型可学习的参数。对于离散的标签或者概率模型，通常会拟合一个条件概率分布函数：

$$p(\hat{y}\mid\boldsymbol{x})=f(\boldsymbol{x},\boldsymbol{\theta})$$

用于预测每一类的概率值。

为了获得这样的一组模型参数$\boldsymbol{\theta}$，我们需要有一套**学习算法**(**Learning Algorithm**)来优化这个函数映射，这个优化的过程就称为学习(**Learning**)或者**训练**(**Training**)，这个需要拟合的函数就称为**模型**(**Model**)。学习的目的就在于找到一个最好的模型，而这样一个模型应当是输入空间至输出空间映射集合中的一个映射，这个映射集合称为**假设空间**(**Hypothesis Space**)。换句话说，学习的目的就在于从这个假设空间中选择出一个最好的元素。

### 3.1.2 机器学习的三要素

在了解了机器学习的基本概念之后，继续讨论机器学习算法的三个基本要素：模型、学习准则(策略)和优化算法。

(1) 模型。

机器学习的第一要素就是模型，而学习的目的就是在模型的假设空间中选择一个最佳的模型，即最接近真实映射的映射函数或条件概率分布，然后再利用该模型去完成相应的任务。

形式化的表述为，如果用$\boldsymbol{F}$表示该假设空间，则它可以定义为决策函数的集合：

$$\boldsymbol{F}=\{f \mid \boldsymbol{Y}=f_\theta(\boldsymbol{X}),\theta \in \mathbf{R}^m\}$$

其中，该函数族是由参数 $\theta$ 决定的，该参数 $\theta$ 所在的空间为 $m$ 维欧式空间，称为**参数空间**(**Parameter Space**)。学习的目的就变为在该参数空间中选择最优的参数。

另一方面，对于概率模型，假设空间可以构造为条件概率分布的集合：

$$\boldsymbol{F}=\{P \mid P_\theta(\boldsymbol{Y} \mid \boldsymbol{X}),\theta \in \mathbf{R}^m\}$$

其中，该条件概率分布族也是由参数 $\theta$ 决定，该参数 $\theta$ 所在的空间为 $m$ 维欧式空间，称为参数空间。学习的目的就变为在该参数空间中选择出最优的参数。

假设空间的分类方法有很多，上述表示就将其分为概率模型和非概率模型。另一种常见的分类方式是将假设空间分为线性和非线性两种，对应的模型就称为线性模型和非线性模型。

对于线性模型，它的假设空间是一个包含可学习参数的线性函数族：

$$f(\boldsymbol{x},\theta)=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$$

其中，参数向量 $\theta$ 由权重向量 $\boldsymbol{w}$ 和偏置 $b$ 组成。

对于非线性模型，则可以表示为若干非线性基函数 $\boldsymbol{\phi}(\boldsymbol{x})$ 的线性组合：

$$f(\boldsymbol{x},\theta)=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})+b$$

其中，$\boldsymbol{\phi}(\boldsymbol{x})$ 代表由若干非线性基函数拼接成的向量，参数向量 $\theta$ 由权重向量 $\boldsymbol{w}$ 和偏置 $b$ 组成。如果该非线性基函数组成的向量本身也是带参数、可学习的，即

$$\boldsymbol{\phi}(\boldsymbol{x})=h(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}'(\boldsymbol{x})+b)$$

其中，$h(\cdot)$ 代表一个非线性函数，那么该模型 $f(\boldsymbol{x},\theta)$ 就是一个**多层感知机**(**Multi-Layer Perceptron，MLP**)。

(2) 学习准则(策略)。

在明确了模型的假设空间之后，接下来需要做的是：如何从假设空间中选出最优的模型，即学习准则或学习策略问题。如果选出的模型不是最优的，那么这个模型函数的预测值 $f(\boldsymbol{X})$ 和样本的真实标签值 $\boldsymbol{Y}$ 会出现不一致的情况，这时通常用**损失函数**(**Loss Function**)或者**代价函数**(**Cost Function**)来衡量它们不一致的大小，损失函数是一个非负值的实值函数，记作 $L(\boldsymbol{Y},f(\boldsymbol{X}))$。

下面来介绍几种常见的损失函数。

**0-1 损失函数**(**0-1 Loss Function**)：0-1 损失函数是最直接地反映正确与错误的损失函数，对于正确的预测，损失值就为 0；对于错误的预测，损失值就为 1。虽然 0-1 损失函数能够直观地反映模型的错误情况，但是它的数学性质并不是很好——不连续也不可导，因此在优化时很困难。通常，我们会选择其他相似的连续可导函数来替代它。

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

**平方损失函数**(**Quadratic Loss Function**)：平方损失函数就是预测值和标签值差的平方，经常用于需要预测连续实值的任务中，适用于回归任务，一般不用于分类任务。该函数拥有良好的数学性质——连续、可微且为凸函数。通常，为了保证其导数前的系数为 1，我们对原函数乘以 $\frac{1}{2}$ 的系数。

$$L(\boldsymbol{Y}, f(\boldsymbol{X})) = \frac{1}{2}(\boldsymbol{Y} - f(\boldsymbol{X}))^2$$

**绝对损失函数(Absolute Loss Function)**：绝对损失函数就是预测值和标签值差的绝对值，与平方损失函数类似，经常用于预测连续实值的回归任务。不同的是，绝对损失函数的导函数值只可能为+1和−1，避免了平方损失函数在偏差很大的情况下梯度太大的问题。它对每个样本的重视程度一视同仁，不会过于偏向误差更大的样本。这是它的优点，同时也是缺点，因此需要仔细考虑、合理利用。

$$L(\boldsymbol{Y}, f(\boldsymbol{X})) = |\boldsymbol{Y} - f(\boldsymbol{X})|$$

**对数损失函数(Logarithmic Loss Function)或负对数似然损失函数(Negative Log-Likelihood Loss Function)**：这个损失函数源于极大似然原理——极大化对数似然函数，而我们通常习惯于最小化损失函数，因此将它转变为最小化负对数似然函数。究其根本，这个损失函数是为了最大化预测条件概率的正确率。

$$L(\boldsymbol{Y}, f(\boldsymbol{X})) = -\log P(\boldsymbol{Y} \mid \boldsymbol{X})$$

**交叉熵损失函数(Cross-Entropy Loss Function)**：交叉熵损失一般用于分类任务。对于一个多分类任务，共有 $C$ 个类别可供选择。我们通常将分类的标签写作一个 one-hot 向量，仅有目标类别的元素为1，其余元素都为0。针对分类的预测值，我们通常也会写作一个向量，它的 $L_1$ 范数为1，每个元素代表对应类别的概率值。为了衡量两个概率分布，我们就需要用交叉熵来衡量他们的差异：

$$L(\boldsymbol{Y}, f(\boldsymbol{X})) = -\sum_{c=1}^{C} \boldsymbol{Y}_c \log f(\boldsymbol{X}_c)$$

这里我们再回顾对数损失函数和交叉熵损失函数，会发现它们其实是等价的。因为这里的标签 $\boldsymbol{Y}$ 是一个 one-hot 向量，因此交叉熵损失函数的目标就是使目标类别的条件概率极大化，即最小化负对数似然函数。

**Hinge 损失函数(Hinge Loss Function)**：对于一个两分类的问题，数据集的标签取值是 $\{+1, -1\}$，模型的预测值是一个连续的实值函数，那么 hinge 损失的定义为：

$$L(\boldsymbol{Y}, f(\boldsymbol{X})) = \max(0, 1 - \boldsymbol{Y} f(\boldsymbol{X}))$$

**Huber 损失函数(Huber Loss Function)**：Huber 损失函数通常用于回归问题。它结合了平方损失函数和绝对损失函数的优点，针对特定的问题进行了优化。在预测值与标签值偏差小的时候选择用平方损失计算，而偏差大的时候选择用绝对损失计算。这样设计的主要目的是减少数据集中离群点的影响，这部分离群点通常为噪声或者是错误标定的点，因此在计算损失的时候不需要关注太多。

$$L_\delta(\boldsymbol{Y}, f(\boldsymbol{X})) = \begin{cases} \frac{1}{2}(\boldsymbol{Y} - f(\boldsymbol{X}))^2, & |\boldsymbol{Y} - f(\boldsymbol{X})| < \delta \\ \delta \cdot \left(|\boldsymbol{Y} - f(\boldsymbol{X})| - \frac{1}{2}\delta\right), & \text{其他} \end{cases}$$

**BerHu 损失函数(BerHu Loss Function)**：我们知道 Huber 损失是为了减弱外点对模型的影响，但是当我们确定外点不多或者急切地想减小大的偏差时，我们会选择另一种相反组合的函数——BerHu 函数。该函数更偏向于偏差大的那些样本，而对于偏差小的样本，我们也可以利用绝对损失函数的导数恒定的优点，来保证学习步长足够大，不至于像平方损失

那样学习缓慢。

$$L_{\delta}(\boldsymbol{Y},f(\boldsymbol{X}))=\begin{cases}|\boldsymbol{Y}-f(\boldsymbol{X})|, & \text{当}|\boldsymbol{Y}-f(\boldsymbol{X})|<\delta\\ \dfrac{(\boldsymbol{Y}-f(\boldsymbol{X}))^2+\delta^2}{2\delta}, & \text{其他}\end{cases}$$

除了上述的几种损失函数外，还有很多其他对特定问题实用的损失函数。总而言之，损失函数的设计是以能够更好地解决具体问题为目的的。

损失函数的作用就类似于机器学习的形式化定义中的性能 $E$，损失函数越小，模型的性能 $E$ 就越大。模型的输入 $\boldsymbol{X}$ 和输出 $\boldsymbol{Y}$ 都可以看作是输入和输出联合空间的随机变量，遵循着联合分布 $P(\boldsymbol{X},\boldsymbol{Y})$，我们称损失函数在该联合分布上的期望为**风险函数**(**Risk Function**)或**期望损失**(**Expected Loss**)。

$$R_{\exp}(f)=E_{P(\boldsymbol{X},\boldsymbol{Y})}[L(\boldsymbol{Y},f(\boldsymbol{X}))]=\iint L(\boldsymbol{y},f(\boldsymbol{x}))P(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}x\,\mathrm{d}y$$

一个好的模型应当有较小的期望损失，但是实际上，我们无法得知真实的数据分布情况，因此也没有办法真的去计算期望风险。事实上，如果我们知道数据的联合分布 $P(\boldsymbol{X},\boldsymbol{Y})$，我们就可以直接利用贝叶斯公式求得条件概率分布 $P(\boldsymbol{Y}|\boldsymbol{X})$，也就不用学习的过程了。所以，这样的循环依赖的问题是一个**病态问题**(**Ill-Formed Problem**)。

然而，从另一个方面来看，我们可以近似地求得期望风险。给定一个数据集，可以很容易计算出模型的**经验风险**(**Empirical Risk**)或**经验损失**(**Empirical Loss**)，即在训练集上的平均误差：

$$R_{\mathrm{emp}}(f)=\frac{1}{N}\sum_{i=1}^{N}L(\boldsymbol{y}_i,f(\boldsymbol{x}_i))$$

如此，一个可以具体实施的学习策略就诞生了，那就是在假设空间中找到一个最优模型 $f^*$ 使得经验风险最小，这就是**经验风险最小化**(**Empirical Risk Minimization，ERM**)准则。

$$f^*=\underset{f\in\boldsymbol{F}}{\operatorname{argmin}}R_{\mathrm{emp}}(f)$$

根据大数定律，当训练集的数据量趋向于无穷大时，经验风险能够保证收敛于期望风险。但是通常情况下，我们无法获得无穷大量的训练集，并且实际中训练集的样本包含了各种噪声，因此实际所用的训练集不能很好地反映数据的真实分布。在这种情况下，如果利用经验风险最小化很容易导致训练集上的损失很低，但是对于未知的数据预测误差很大。这种训练误差不断降低、测试误差反而提高的现象称为**过拟合**(**Overfitting**)。

过拟合发生的因素有很多，最主要的两点是因为训练数据量不足以及模型能力过强或模型函数过于复杂。为了解决这一问题，我们将经验风险函数进行修改，增加了**正则化**(**Regularization**)项或**惩罚**(**Penalty**)项，得到了结构风险函数：

$$R_{\mathrm{str}}(f)=\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i,f(\boldsymbol{x}_i))+\lambda J(f)$$

其中，$J(f)$代表模型函数的复杂度，是定义在假设空间 $\boldsymbol{F}$ 上的泛函，简单来说就是函数的函数。模型函数的复杂度越大，$J(f)$也就越大。一般我们使用模型参数向量的 $L_2$ 范数来近似模型的复杂度。因此，该风险函数强制使模型的复杂度不应过高，这种学习策略称为**结构风险最小化**(**Structural Risk Minimization，SRM**)。

从数学优化的角度来看结构风险函数，可以将模型的复杂度项看作是引入拉格朗日乘

子的带约束优化问题，约束条件为模型函数的复杂度为 0，目标为经验风险最小。而从贝叶斯学习的角度来看，正则化项可以看作人为给定的先验分布，即在不确定目标的分布时，选择最“模糊”的分布。

此外，还有一种与过拟合相反的极端就是**欠拟合**（**Underfitting**），即模型过于简单而导致训练误差一直很大。

（3）优化算法。

在获得了数据集、确定了假设空间以及选定了合适的学习准则之后，最后一步就是要解决一个**最优化**（**Optimization**）问题。机器学习的训练和学习的过程，实际上就是求解最优化问题的过程。

如果最优化问题存在显式的解析解，那么我们就可以很容易求取它的闭式解；但是如果不存在解析解，我们就只能通过数值方法来不断逼近。并且在机器学习中，很多优化函数不是凸函数，因此如何寻找全局最优解就成了一个很重要的问题。

最简单也最常用的优化算法就是**梯度下降法**（**Gradient Descent，GD**）。梯度下降法通过不断迭代的方式来降低风险函数的值：

$$\theta_{t+1}=\theta_t-\alpha\frac{\partial R(\theta)}{\partial\theta}$$

其中，$\theta_t$ 为第 $t$ 次迭代时的参数值，$\alpha$ 代表优化的步长，又称为学习率。学习率过小，会导致学习速度太慢，还有可能会导致陷入局部最优；学习率过大又会出现震荡，严重时会导致发散。

针对梯度下降法，后续还有很多的改进。例如为了优化它的收敛速度以及越过局部“平缓”区域，可以加入“**冲量项**”（**Momentum**）来使优化保持一定的速度；为了优化迭代的速度以及质量，采用**随机梯度下降**（**Stochastic Gradient Descent，SGD**）和**小批量梯度下降**（**Mini-Batch Gradient Descent，MBGD**）等。这些“小技巧”在接下来的章节中会一一介绍。

### 3.1.3 机器学习方法概述

机器学习按照学习方法来分类，可以分为有监督学习、非监督学习、半监督学习、深度学习和强化学习等内容。需要注意的是，这几种方法并不是非此即彼的关系，而是可以相互交叉的。例如深度学习中的任务有监督学习的方法，也有非监督学习的方法；深度学习和强化学习可以相互结合，称为深度强化学习。

首先，**有监督学习**（**Supervised Learning**）又称有教师学习，是指利用带标签的样本来优化算法的参数，使其性能提高的过程。监督学习利用的数据集都不仅包含特征，还包含标签。根据这些标签，我们可以设计一种学习策略（损失函数）来优化模型。监督学习也是目前使用最广、效果最好的一种学习方式。监督学习的优点是模型性能往往较好，精度高；缺点是需要人为的参与，对数据集的标定工作耗时耗力，获取大量标记的数据成本很高。

表 3.2 列出了传统机器学习方法中的一些代表性的监督学习方法以及它们的学习策略、优化方法。

表 3.2 传统机器学习方法中的部分监督学习方法

| 方 法 | 适用任务 | 学习策略 | 损失函数 | 优化算法 |
|---|---|---|---|---|
| 感知机（线性分类器） | 二分类 | 最小化误分点到分类超平面的距离 | 平方损失 | 随机梯度下降 |
| 朴素贝叶斯（NB） | 多分类 | 极大后验概率估计 | 对数似然损失 | 概率计算公式 |
| 决策树（DT） | 多分类或回归 | 正则化的极大似然估计 | 对数似然损失 | 特征选择，生成，剪枝 |
| 最大熵模型（ME） | 多分类 | 正则化的极大似然估计 | 逻辑斯蒂损失 | 改进的迭代尺度法等 |
| 支持向量机（SVM） | 二分类 | 最小正则化合页损失，软间隔最大化 | 合页损失 | 序列最小最优化算法 |
| 提升方法（Boosting） | 二分类 | 极小化加法模型的指数损失 | 指数损失 | 前向分步加法算法 |
| 隐马尔可夫模型（HMM） | 标注问题 | 极大后验概率估计 | 对数似然损失 | EM 算法等 |
| 条件随机场（CRF） | 标注问题 | 正则化极大似然估计 | 对数似然损失 | 改进的迭代尺度法等 |

**无监督学习**（**Unsupervised Learning**）又称为无导师学习，是指算法根据没有标签的样本来解决各种问题的过程。现实生活中经常会出现如下情况：（1）缺乏足够的先验知识，有些数据难以标注；（2）人工标注的成本太高；（3）有无穷多的可行解，无法确定哪一种最优或者这些解都是可接受的。因此，我们希望机器或者算法能够脱离人为的标签，来完成这些任务，或者是辅助完成这样的任务。在无监督学习中，数据集中的样本只有特征，没有标签，这些标签是模型根据特征按某种规则归纳得出的。近年来，无监督学习受到越来越多的关注。无监督学习因为不需要人参与，所以训练数据量可以更大。在传统机器学习算法中，聚类算法和主成分分析 PCA 是两个最有代表性的无监督学习算法。

图 3.2 展示了 $K$-means 聚类算法的结果。对于同样的特征点，使用不同的迭代初始值可能会得到完全不同的结果。这也反映了无监督学习算法的一个问题——难以衡量高维数据的相似度，直观的评价标准还是具有人为的主观性。

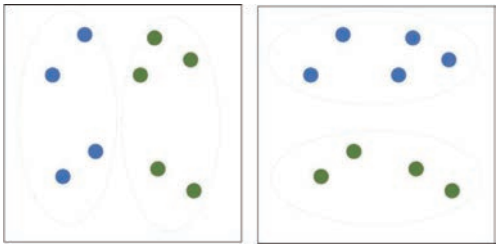

图 3.2 $K$-means 聚类的结果

**半监督学习**（**Semi-Supervised Learning**）是有监督学习和无监督学习相结合的一种学习方法，它既利用了有标签的数据，也利用了没有标签的数据，因此人为的参与度较少，同时准确度也比较高。半监督学习有三个常用的基本假设：**平滑假设**（**Smoothness Assumption**）、

**聚类假设**(**Cluster Assumption**)和**流形假设**(**Manifold Assumption**)。究其根本,这三个假设说的都是一回事——对于两个样本,如果它们在稠密区域距离很近或者位于同一簇中时,它们的标签有很大概率相同。通过这样的假设设计出的算法,我们可以利用半监督学习得到比只用带标签数据的监督学习更好的结果。

监督学习、非监督学习和半监督学习都是以是否有数据的标签来分类,而深度学习和强化学习是按照算法模型的结构和功能取得名字。

**深度学习**(**Deep Learning**)是指深层的人工神经网络结构,这种结构通过深度的网络不断提取高层次的特征来达到非常优秀的结果。深度学习是一种基于数据表征学习的方法,动机在于模拟人脑的分析过程,从底层特征到高层特征一一建模。深度学习有几种比较有代表性的网络结构,比如前馈神经网络——**多层感知机**(**Multi-Layer Perceptron, MLP**)和**卷积神经网络**(**Convolutional Neural Network, CNN**),反馈神经网络——**循环神经网络**(**Recurrent Neural Network, RNN**)。这些结构根据其特点有不同的功能,同时也可以组合起来使用,来更好地利用它们各自的特点。

深度学习也有监督学习、无监督学习和半监督学习之分。在众多深度学习模型中,监督学习方法占大多数(虽然有些方法称为非监督深度学习,实际是通过一些其他的方法间接地获得标签,而不是人为标定标签),包括卷积神经网络、循环神经网络和多层感知机完成的大多数分类、回归等任务;无监督学习方法也是近期的研究热点,包括**编码解码器**(**Encoder-Decoder**)、**生成对抗网络**(**Generative Adversarial Network, GAN**)、**深度信念网络**(**Deep Belief Network, DBN**)和**深度玻尔兹曼机**(**Deep Boltzmann Machine, DBM**)等结构完成的特征降维和概率分布估计等任务。

目前深度学习在很多任务上都达到了非常好的结果。例如基于卷积神经网络的图像识别、物体检测、语义分割等任务,基于循环神经网络的动作检测、语音识别、机器翻译等任务。然而,深度学习也有很多的缺点:网络的可解释性不强——大多数的网络结构并不能让人清楚知道每一层学习到的是什么;训练代价太大——动辄几天的训练时间和大量的运算加速器让人难以接受;优化困难——深层的结构让基于梯度的优化变得艰难……当然这些问题目前也有大量研究人员尝试解决,例如人工推演网络的参数、使用预训练的网络参数来加速训练等。

深度学习是本书的重点内容,我们会在后续的章节中介绍常用的深度学习模型以及它们的特点和应用。

**强化学习**(**Reinforcement Learning**)又称为再励学习,是通过**智能体**(**Agent**)以试错的方式进行的学习。不同于之前所讲的监督学习与非监督学习,强化学习并不需要真实的标签来指导模型的修改。它是通过智能体不断与环境进行交互来获得奖励或者惩罚,目标是使智能体最终获得的奖励值最大。

强化学习非常适合于那些没有绝对的正确标准的任务,如棋牌类对弈、公路自动驾驶策略、游戏中的人机对战等。这里以自动驾驶中的决策过程为例,汽车从出发地驶往目的地的路线并没有绝对的正确标准,因此很难人为地规定学习的标签,我们只能给予它一些正确和错误的规则,比如"与其他车发生碰撞"会得到惩罚而"安全无事故地到达了目的地"则会得到奖励。通过这些奖励和惩罚的措施,我们想让智能体"自发"地决定应该如何行驶才能获得最大的奖励。因此,强化学习和一句老话很像——"不管黑猫白猫,抓住老鼠的就是好猫"。

强化学习是一种思想，它突破了传统“问题只有唯一答案”的想法，成为目前火热的解决决策问题的一种机器学习算法。在解决问题的过程中，强化学习经常会与深度学习相结合，从而使模型获取强大的特征提取和综合能力，这种模型被称为**深度强化学习**（**Deep Reinforcement Learning**）。在后续的章节中，我们会对强化学习算法的设计和用途作进一步的说明。

## 3.2 数据预处理

数据对于机器学习算法的重要性就如同空气对于人的重要性。数据的质量直接影响模型的效果，在这一节中，我们将会介绍几种常用的数据预处理方法与流程。

### 3.2.1 数据清洗

数据清洗，顾名思义就是将数据集中的“脏”数据去除。在这个大数据的时代，我们在获取海量数据的同时，肯定会遇到很多的“脏”数据，这些“脏”数据主要包括残缺的数据、错误的数据和重复的数据等。这些数据显然是我们不想要的，因此我们就需要根据某种规则将它“清洗”掉，这就是数据清洗。注意，数据清洗的工作一般是由计算机完成，而不是人工去除。

数据清洗的步骤主要包括：分析数据、残缺数据处理、错误数据处理和重复数据处理等。

首先，当我们得知任务需求之后，我们为了满足需求，就会去寻找相应的数据。获得数据后，就需要对数据进行**统计分析**，观察合理的数据大概是什么样的，看看哪些数据是不合理的，同时了解基本的数据情况。既然是统计分析，我们可以利用一些数学上的统计工具来协助完成，例如直方图、散点图等，通过观察图像我们可以很容易地找到那些不合理的数据样本。

**缺失数据**是不可避免的。我们在网上爬取的数据不一定包含所有我们需要的属性，每一个独立的数据样本可能会包含不同的缺失值。有些人对于缺失值就直接删去，而有些人则是将它们赋予 0 或者其他特殊的值。那么究竟应该怎么做呢？我们应当根据实际情况选择不同的处理方式。就如同 10 000 个样本中，有缺失属性的样本只有 5 个，我们可以直接删除，因为删除对整体的数据影响不大；而如果 10 000 个样本中有 90%的样本都存在缺失属性的问题，或者这些存在缺失数据的特征维度非常重要，我们就需要仔细考虑解决方式了。

通常，对于缺失数据我们有以下几种处理方式。

（1）直接删去。这种方法适合于缺失数据少，并且缺失数据随机出现，删除对结果影响不大的情况。

（2）赋予一个常量。例如我们可以将缺失的属性赋予 0 值或者 Unknown 值，但是这样处理的效果不一定好，因为算法可能会将赋予的常量当作数据本身的属性值，因此该方法使用较少。

（3）赋予均值或中位数。与赋予常量不同，该方法赋予的是这一属性维度的统计特征，处理简单，与直接删去相比也不会减少样本数量。但是赋予的缺失数据可能会存在偏差。

因此，对于数据分布正常的数据，我们可以用均值来赋予；而对于数据分布不对称或倾斜的情况，用中位数可能比均值更好。

(4) 插补法。使用现有未缺失数据通过某种方法来生成该缺失数据。

① 随机插补法：随机选取一个未缺失的值来填充该缺失的部分。

② 热平台插补法：在非缺失的数据中找到一个与缺失样本最相似的样本，使用该样本对缺失的部分进行填充。

③ 拉格朗日插值法或牛顿插值法。

(5) 建模法。可以使用机器学习中的方法对数据进行建模，然后进行推理预测。比如可以构造一棵决策树来预测缺失的值。

以上的几种方法各有优缺点，具体使用时我们需要根据数据的分布情况和缺失情况来综合考虑。一般而言，建模法是使用较多的方法，因为它能建模未缺省数据来预测缺失值，准确率较高。

**错误数据**又称为异常值，也叫作离群点。在分析数据时，我们可以通过画图的方法很容易找到离群点，但是画图的目的毕竟是通过人来判断离群点，并且数据量大时，画图的效率很低。在这里，我们继续介绍一些分析错误数据的方法。

(1) 通过简单的数据分析。对于收集的数据，我们一般会对其中的属性值有大概的先验感受。我们可以利用这种先验来制定某种规则，从而筛选出错误的数据。例如，人的身高体重不可能存在负值，人的年龄不可能超过 200 岁等。

(2) 3-sigma 原则。对于服从正态分布的数据，异常值是那些观测值与均值的偏差超过三维标准差的数据。对于正态分布，我们很清楚地知道 $P(|x-\mu|>3\sigma)\approx 0.003$，因此这部分数据属于小概率情况。

(3) 箱型图。通过寻找数据的上四分位值 $P$ 和下四分位值 $Q$ 来估计数据大概的上界和下界，那些超过上、下界的数据就被称为异常值。因为有 25% 的数据可以变得任意远而不会影响四分位值，因此四分位值具有很强的稳定性。利用箱型图来判断错误数据是一种非常常见的方法。

$$\text{upperbound} = P + 1.5(P - Q)$$
$$\text{lowerbound} = Q - 1.5(P - Q)$$

(4) 建模法。在分析异常数据时同样可以通过建模的方法来判断，对于那些不能很好拟合模型的数据，就可以判断为异常值。对于聚类的模型，那些不属于任何一类的数据被称为离群点；对于回归模型，那些偏离预测值的数据被称为离群点。在了解数据分布的时候建模的方法效果通常比较好，但是对于高维数据效果可能很差。

(5) 基于距离。比较任意两个样本的空间距离，对于那个远离其他样本的样本可以视为离群点。该方法操作简单，但是计算复杂度很高，并且对于那种多簇分布、数据密度不均的情况适用度不高。

(6) 基于密度。如果一个样本的局部密度低于它的大部分临近样本的密度，我们可以视它为离群点。

在识别出错误数据之后，对于错误数据的处理也类似于缺省值。

(1) 直接删去。对于错误数据少的情况比较适用。

(2) 不作处理。如果算法对于异常值很鲁棒可以采用这种方式，但是如果算法对于离

群点很敏感，尤其是基于度量距离的算法，如 K-means 和最近邻等，不建议使用。

(3) 将异常值删去，视为缺省值，并且按照缺省值的处理方法来处理。

缺省数据和错误数据是两种比较严重的问题，在解决了这两个问题之后，数据集中的样本基本“正常”。下一步需要做的就是去除**重复数据**，保证数据集中且相同特征的数据只有一份。去重的方法有很多，包括直接比较、排序后删除相邻重复数据，使用哈希函数映射后再进行匹配可以提高比较效率，利用**“集合”(Set)**这种数据结构可以很方便快捷地去重等。

### 3.2.2 数据集拆分

在清洗数据之后，我们就可以正式使用数据集了。根据之前的讲解，我们知道模型应当在训练集上进行训练，训练结束后在测试集上进行评价。但是由于训练迭代次数太多很有可能引起过拟合，而训练迭代次数不足会导致欠拟合，我们不知道训练的次数应该选取多少为优。因此，我们还需要另外一个与它们都不相交的集合——**验证集(Validation Dataset)**。

在机器学习中，我们通常将获得的数据集分为三份。

(1) **训练集(Training Dataset)**：用来模型迭代训练的数据集。

(2) **验证集(Validation Dataset)**：用来预防过拟合的发生，辅助训练过程的数据集。

(3) **测试集(Test Dataset)**：用来评估最终训练好的模型性能的数据集。

这里需要注意的一点是测试集在训练的过程中是不可见的，我们在训练的时候能评估模型好坏的数据集只有验证集，而不应该在训练时直接使用测试集来评估训练，更不应该将测试集加入到训练集中参与模型的训练。在训练过程中，参与模型训练的只有测试集中的数据，验证集可以辅助我们调整网络的超参数等工作。通常，我们会选择在验证集中评估最好的模型作为最终的输出模型。

通常有以下三种划分数据集的方法。

(1) **留出法(Hold-Out)**：留出法将数据集分为两个互斥的集合，通常选择 70%的样本作为训练集，剩下 30%的样本作为测试集，没有验证集。使用留出法对数据集进行划分时，需要注意训练集和测试集的数据分布应当相同，不能引入额外的偏差导致对最终模型的训练和评价产生影响。为达此目的，通常我们需要进行多次划分，然后重复训练和评估，最后取平均作为最终留出法的评估结果。

(2) ***K*-折交叉验证法(*K*-Fold Cross Validation)**：将原始数据均分为 $K$ 个互斥的集合，并且尽量保证每个集合的数据分布一致。如此，就可以获取 $K$ 组训练集-测试集对，从而可以进行 $K$ 次训练和测试，最终再通过 $K$ 次交叉验证取平均得到评估结果。通常，$K$ 的取值为 5、10、20 等。

(3) **自助法(Bootstrap)**：自助法主要通过自助采样的方式进行：初始数据集大小为 $m$ 个样本，每次从数据集中选出一个样本放入训练集中(初始训练集为空，选出的数据并不从原始数据集中删除)，这样的操作重复 $m$ 次，那么我们就得到了包含 $m$ 个样本的训练集，最后从原始数据集中选出不在训练集中的那部分样本作为测试集。如此，一个样本不被选入训练集的概率为

$$p=\left(1-\frac{1}{m}\right)^{m}$$

当数据集很大时，该概率为

$$p=\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^{m}=\frac{1}{\mathrm{e}}\approx 0.368$$

自助法在多次实验下的性能评估变化小，适合于数据集小且难以划分的情况。此外，自助法在集成学习中应用非常广泛——可以通过多次自助采样来生成多个弱分类器，然后集合为一个强分类器。但是自助法在采样的过程中会引入重复的数据，因此会改变数据分布，引入偏差，在数据量足够大的情况下，使用留出法和交叉验证法的效果会更好。

### 3.2.3 数据集不平衡

数据集不平衡的问题很常见，而对于此类任务，如果不对不平衡的数据集作调整，那么机器学习算法的效果会非常差，例如判断罪犯任务。我们知道，生活中大部分人都是良民，罪犯占极少数。对于一个 10 000 人的样本，其中的罪犯可能不超过 50 个，那么如果不对这样的数据集作处理，算法会更倾向于将一个人分类为非罪犯。一个更极端的情况是将所有人都分类为非罪犯，这样的正确率能打到 99.5%，但是这显然不是我们想要的模型，因为它没有做任何事。如果要预测的事件比例小于 5%，那么这样的事件我们称之为罕见事件(Rare Event)。

处理数据集不平衡的问题有很多方法，包括数据层面的重采样、集成算法等方法，这里只介绍对数据集的处理。

数据层面的**重采样(Resampling)**主要目标大多都是为了增加少数类的样本数量或者减少多数类的样本数量，从而达成基本平衡。常用的几种重采样技术如下。

(1) **随机欠采样(Random Under-Sampling)**：随机欠采样的目的是降低多数类参与训练的样本数量，从而使多数类和少数类的样本数量趋于平衡。在上例中，我们可以从非罪犯的群体中抽取 1%的个体作为训练的负样本，从而达到平衡。随机欠采样的优点是减少了训练样本，提高训练的速度；缺点是丢弃了很多可能有用的信息，严重时会导致欠采样的数据分布改变。

(2) **随机过采样(Random Over-Sampling)**：随机过采样与随机欠采样的目的相反，它的目标在于通过复制少数类的样本来增加少数类的数量。在上例中，我们可以将罪犯的群体复制 100 倍从而达到平衡。随机过采样不会丢失那些有用的信息，但是单纯地复制少数类的样本很容易导致过拟合。

(3) **基于聚类的过采样(Cluster-Based Over-Sampling)**：该方法将 $K$-means 聚类算法分别用于多数类和少数类样本，然后每一个聚类都被过采样使得所有相同类的聚类都拥有相同数量的样本。在上例中，我们可以将罪犯的群体和非罪犯群体分别聚类，得到了如下结果：

罪犯群体：2 个聚类(20，30)

非罪犯群体：4 个聚类(4000，3000，2000，500)

然后我们进行过采样，使得每个群体中的所有聚类的样本数相同：

罪犯群体：2 个聚类(2000，2000)

非罪犯群体：4 个聚类(4000，4000，4000，4000)

通过基于聚类的过采样方法，克服了不同聚类的类间不平衡性，同时也克服了多数类与

少数类之间的不平衡性；但是与其他过采样方法相同，该方法也没有逃脱过拟合的可能。

(4) **合成少数类过采样技术(Synthetic Minority Over-Sampling Technique，SMOTE)**：该过采样方法并不是完全相同的复制少数类样本，而是将新产生样本与原来的样本作一些改变再加入数据集中。通常做法是，选取少数类中的若干点，然后对于点 $A$，寻找距离它最近的 $m$ 个少数类的点，然后从中随机选出一个点 $B$，最后将线段 $AB$ 连线上的任意一点加入数据集中作为新的少数类点。这种过采样方式可以缓解过拟合问题，且不会有信息损失，但是由于没有考虑每个少数类点周围的多数类点的分布，可能会增加多数类与少数类样本的空间重叠，从而引入额外的噪声，且 SMOTE 算法通常对高维数据并不是那么有效。

## 3.3 特征工程

### 3.3.1 特征编码

现实应用中，我们使用的数据的种类是多种多样的，例如图像、视频、音频、文本等。而不同类型的数据的**原始特征(Raw Feature)**也是不同的，因此我们需要将这些原始特征编码为机器学习算法可使用的类型。

**图像(Image)**。对于图像特征，我们比较熟悉的一种形式是将其表示为三维张量的结构，其中前两个维度是图像的高和宽，最后一个维度与图像的颜色空间有关。对于彩色图像，它的颜色空间一般为红-绿-蓝(RGB)、色调-饱和度-亮度(HSI)等；对于灰度图像，它的颜色空间仅仅只有灰度值一个维度。一般地，考虑到图像特征提取过程中产生的中间特征图(Feature Map)，我们通常称它的第三维为**通道(Channel)**，它反映了图像中每个像素点的特征向量。因此，图像的原始特征空间大小为 $[0,255]^{m\times n\times c}$，其中 $m$ 为图像的高、$n$ 为图像的宽、$c$ 为图像的通道数。

在传统机器学习算法中，我们一般会将图像的原始特征进行进一步的特征提取，得到高层次的特征后再进行特征的学习、分类等工作。一个典型例子是基于图像的行人检测任务，具有代表性的做法是首先对图像提取**梯度直方图特征(Histogram of Gradient，HOG)**，然后再利用**支持向量机(Support Vector Machine，SVM)**对其中的候选区域分类。

在当今的深度学习时代，由于深度神经网络拥有强大的提取特征的能力，我们经常会直接对图像的原始特征进行处理。

**文本(Text)**。机器学习算法能够直接利用的特征大多都是数值量化后的特征(也有例外，如决策树等)，而文本特征就是这一类需要进行特征编码才能使用的特征。以中文文本为例，我们需要将“我爱你”进行编码。

最简单的做法就是将每个字按顺序编码，如“我”：00，“爱”：01，“你”：10。对于每个文字，我们都可以从字库 $V$ 中找到它对应的编码，编码的长度为 $\lceil\log|V|\rceil$。这种编码方式简单快捷、编码长度短，但是编码的可解释性较差，不利于数值计算中的特征提取过程。你可能会发现“我”和“你”的编码平均值是“爱”，这其实是不合理的。因此，该编码方式应用场景不大。

另一种比较简单的编码方式是 One-Hot 编码，如“我”：001，“爱”：010，“你”：100。对于每个文字，我们都通过向量的某一位设置 1 来编码，编码的长度为字库的大小 $|V|$。这种编码方式能得到稀疏编码，很快捷，文字特征之间相互垂直，但是当字库非常庞大时特征向量的维度会非常大，会导致维度灾难。

考虑到文字之间的相关性，我们可以将这种 One-Hot 向量压缩为较低维的向量。这个过程叫作**词嵌入**(**Word Embedding**)，即将高维的特征向量嵌入到低维空间中，并且映射前后的信息应当不被损失，一个典型的词嵌入方法就是 word2vec。对于两个意思相近的词，它们的“距离”也应当近；对于意思不同的词，它们的“距离”应当远。这种考虑词语语义上下文的编码方式目前使用最广泛。

### 3.3.2 特征选择与特征降维

特征选择就是选择出对模型的预测有用的特征，将那些无用的、有干扰的特征去除的过程。在实际应用中，我们通常能得到海量的数据和它们的特征。一方面，特征的数量很多可能会导致学习到了一些实际不相关的特征，模型的性能从而有所下降；另一方面，大量的特征对于模型的学习本就是一种负担，最终导致模型需要花费大量时间来训练，同时模型也会变得很复杂。严重时则会导致**维度灾难**(**Curse of Dimensionality**)，即维度增加会导致计算量以指数速度增长。

为了避免维度灾难的问题，我们就需要从全部的特征中选择出一个最优的特征子集，使得在某个评价指标下，训练数据和测试数据的评估效果最好。因此，特征选择通常有三种做法。

(1) 从大量特征中选出固定数量的特征，并且使得模型效果最好。这是一个无约束的组合优化问题。

(2) 对于给定的目标性能，找到数量最小的特征子集。这是一个有约束的最优化问题。

(3) 在模型性能和特征数量之间找到一个折中点。

不幸的是，这三个问题都是 NP 难问题，当可选特征数量很大时，寻找最优解变得不可能。所以，我们的目标就变为寻找一个较优的特征子集。

一种简单直接的特征选择方法是**子集搜索**(**Subset Search**)。原始特征数量为 $d$ 的非空子集数量为 $2^d-1$ 个。我们可以通过穷举法尝试所有的特征子集，然后选择最优的结果，这种暴力搜索的方法耗时最多，但是理论上可以找到最优子集。为了权衡搜索速度和特征子集的质量，我们可以加入贪心的策略。常用的两种贪心策略为：从空集合开始不断选择当前最优特征的**前向搜索法**(**Forward Search**)和从全集开始不断删去无用特征的**反向搜索法**(**Backward Search**)。

此外，子集搜索的方法也能分为过滤式和包裹式两种。

(1) **过滤式**(**Filter**)方法不依赖于将要使用的算法模型，通过信息量或信息增益来衡量特征的有用与无用的程度，然后向空集中加入有用特征或从全集中删除无用特征。

(2) **包裹式**(**Wrapper**)方法依赖于将要使用的算法模型，它通过后续算法模型的评价指标来衡量当前特征的有用与否，然后向空集中加入有用特征或从全集中删除无用特征。

另一种获得较优特征子集的过程借助了一些随机算法，比较有代表性的例如**模拟退火**

**算法**(**Simulated Annealing**)和**遗传算法**(**Genetic Algorithm**)。这些算法都是通过某种规则随机地寻找优化函数的最优点,但不一定保证是全局最优。

最后,我们介绍一种常用的无监督数据降维方法——**主成分分析**(**Principal Component Analysis**, **PCA**)。一方面,PCA 可以通过线性变换,降低特征之间的相关性;另一方面,在线性变换之后,我们可以找到数据差异性最大的方向,称之为主方向,通常主方向的维度对模型的分类是有帮助的,而那些数据差异性较小的方向(那些维度的数据方差通常接近 0)对于后续任务帮助不大。因此,PCA 的目的就在于通过一个线性变换(或者找到一组新的基底),使得变换后的数据方差最大,协方差最小。

对于一组特征 $\{x_1, x_2, \cdots, x_n\}$,我们先求出这组特征的中心

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

然后对所有特征进行去中心化,并且按行组织成矩阵的形式

$$\boldsymbol{X} = [x_1 - \bar{x}, x_2 - \bar{x}, \cdots, x_n - \bar{x}]$$

再求特征的协方差矩阵

$$\boldsymbol{C} = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$$

该协方差矩阵是一个对称矩阵,其对角线元素为特征的方差,其他元素为特征间的协方差。对此,我们需要找的线性变换矩阵就是使得协方差矩阵对角化的特征向量,即对协方差矩阵 $\boldsymbol{C}$ 进行特征分解,得到

$$\boldsymbol{C} = \boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^{\mathrm{T}}$$

对特征值按从大到小排列,最后取前 $K$ 个最大的特征值对应的特征向量为新的基底,就组成了所求线性变换的矩阵:

$$\boldsymbol{P} = [\boldsymbol{V}^{\mathrm{T}}]_{1:K}$$

其中,$\boldsymbol{P} = [\boldsymbol{V}^{\mathrm{T}}]_{1:K}$ 代表特征向量组成的矩阵的转置的前 $K$ 行。

PCA 的本质就是将方差最大的方向作为新的主基底(方向),并且在其各个正交的方向上去相关性。但是 PCA 降维也有一定的限制,诸如 PCA 虽然能够解除线性相关性,但是对于高阶的非线性相关,传统的 PCA 算法就无能为力了,这时可以考虑 Kernel PCA。此外,PCA 作为一种无监督的特征选取器,没有调参的过程,这就意味着谁来做 PCA 都可以得到相同的结果,没有个性化。

### 3.3.3 特征标准化

我们搜集到的特征一般是有某种含义的,例如判断一个人是否身体健康,我们可能提取身高、体重、血压、红细胞计数等特征作为参考。但是如果这些特征不作任何处理,机器学习算法可能不能得到很好的效果。以此为例,大部分成人的身高在 150~200cm,极差大概为 50 厘米;但是每个人的红细胞计数可能相差很大,每立方毫米的计数从 4 000 000~5 500 000 都是正常的,极差大概为 1 500 000。由于特征之间的量纲不同,导致每一个特征如果没有归一化完全没有可比性。如果不经归一化就进行 PCA 降维,那么就会筛除掉很多可能有用的特征。

特征归一化又称为特征标准化，其目标就在于使不同特征的量纲一致，并且数据变为 0 至 1 之间的小数。常用的标准化算法如下。

（1）线性标准化：

$$y=\frac{x-MinValue}{MaxValue-MinValue}$$

（2）标准差标准化：

$$y=\frac{x-\bar{x}}{\sigma}$$

（3）Logistic 标准化：利用逻辑斯蒂函数进行非线性映射。

$$y=\frac{1}{1+e^{-x}}$$

（4）反正切函数标准化：利用反正切函数进行非线性映射。

$$y=\frac{\arctan(x)\times 2}{\pi}$$

（5）小数定标标准化：直接移动小数点的位置来实现标准化，其中 $j$ 是使得最大的 $y$ 小于 1 的最小值。

$$y=\frac{x}{10^{j}}$$

特征的标准化是一种定制的操作，需要根据实际数据的特点进行设计。除上述方法外，还有很多其他方法。

综上所述，线性标准化适用于样本分布比较均匀的情况；标准差标准化适用于样本近似于正态分布，或者当最大值最小值未知，以及在最大最小处存在孤立点的情况；而非线性映射的方法通常用于数据分化比较大的情况，即有的数据很大、有的数据很小，通过非线性函数映射，使得数据变得尽量均匀或“有特点”。

## 3.4 模型评估

获得了训练数据，做了数据清洗并选择了备选特征，训练了模型参数，最后一步便是评估该模型的好坏。在训练的时候，模型评估的工作在验证集上进行；在测试的时候，模型评估的工作在测试集上进行。

对于回归问题，可供选择的评价指标较少，大部分情况下会选择平均平方误差、平均绝对误差、平均对数误差等指标进行评价，或者针对特定问题设计特定的评价指标。这里不作过多的介绍。

对于分类问题，我们总能得到类别 $c$ 预测结果的混淆矩阵，如表 3.3 所示。

表 3.3 类别预测结果的混淆矩阵

| 真实类别 | $y$ 值 | 预测类别 | |
|---|---|---|---|
| | | $\hat{y}=c$ | $\hat{y}\neq c$ |
| | $y=c$ | TP | FN |
| | $y\neq c$ | FP | TN |

在混淆矩阵中，有四个常用的概念，分别是：

**真正例**(**True Positive**，**TP**)：真实类别为 $c$ 且预测结果也为 $c$ 的样本。

**假负例**(**False Negative**，**FN**)：又称为假阴性样本，真实类别为 $c$ 却预测为其他类别的样本。

**假正例**(**False Positive**，**FP**)：又称为假阳性样本，真实类别不为 $c$ 却预测为类别 $c$ 的样本。

**真负例**(**True Negative**，**TN**)：真实类别不为 $c$ 且预测结果也不为 $c$ 的样本。

根据这四个概念，我们可以得到如下常用的评价标准。

**准确率**(**Accuracy**)：最常用，衡量分类结果的正确性。

$$\text{Accuracy}=\frac{1}{N}\sum_{i=1}^{N}I(y_i=\hat{y}_i)$$

其中，$I(\cdot)$为指示函数，相等为 1，不等为 0。对于二分类问题，准确率也等于

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

**错误率**(**Error Rate**)：与准确率对应的就是错误率。

$$\text{ErrorRate}=1-\text{Accuracy}=\frac{1}{N}\sum_{i=1}^{N}I(y_i\neq\hat{y}_i)$$

对于二分类问题，错误率也等于

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

准确率和错误率对于评价模型是远远不够的，现在考虑这样一种场景：一个任务的真实类别中，有 99 个是负样本，只有 1 个是正样本；或者有 99 个正样本，仅有 1 个是负样本。那么模型有可能会将结果全部预测为负样本或者正样本，这其实是我们不想看到的结果，因为模型相当于什么都没有做！因此，我们还需要考虑其他的一些性能指标，用于区别这种实际中常见的情况。

**查准率**(**Precision**)：又称为精确率或精度，代表的是预测为类别 $c$ 的样本中有多少预测正确了。

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

**查全率**(**Recall**)：也称为召回率，代表真实标签为类别 $c$ 的众多样本中，有多少被真正检测出来了。

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

查准率与查全率往往不能得兼，例如在物体检测的应用中，随着最终置信度的阈值调整，查准率与查全率此消彼长。但是如果一个算法能够同时拥有很高的查准率与查全率，这一定是极好的。

***F* 值**(***F* Measure**)：综合查准率与查全率两者关系的一个综合性指标。

$$F=\frac{(1+\beta^2)\cdot\text{Precision}\cdot\text{Recall}}{\beta^2\cdot\text{Precision}+\text{Recall}}$$

其中，$\beta$ 值用于平衡准确率与召回率，一般取值为 1，称为 F1 值。

**宏平均(Macro Average)**和**微平均(Micro Average)**是用于计算所有类别整体的精确率、召回率和 F1 值的方法。宏平均计算的是每一类的精确率、召回率和 F1 值的算术平均值：

$$\text{Precision}_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C}\text{Precision}_c$$

$$\text{Recall}_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C}\text{Recall}_c$$

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

微平均计算的是每一个样本的精确率、召回率和 F1 值的算术平均值。由于对单个样本而言，准确率和召回率是相同的，要么是 0，要么是 1，因此准确率的微平均和召回率的微平均是相同的。对于不同类别样本数量不均衡的情况，使用宏平均更合理，宏平均更关注于少数类的评价指标。

最后，我们又回到数据集划分时谈到的**交叉验证(Cross Validation)**。交叉验证可以避免因划分训练集和测试集带来的破坏数据分布的问题。因此，通过交叉验证，我们可以缓解不同划分带来的性能评估不准的问题。

## 3.5 实践：鸢尾花分类

分类模型是机器学习非常重要的一个组成部分，它的目标是根据已知训练集提供的样本，通过计算选择特征参数，创建判别函数，以此对新的样本进行分类，属于监督学习的一个实例。本节以鸢尾花分类为例，根据鸢尾花的花萼和花瓣大小将其分为三种不同的品种，如图 3.3 所示。

图 3.3 鸢尾花分类

本实践代码已在 AI Studio 上公开，通过扫描上方二维码或访问 https://aistudio.baidu.com/aistudio/projectDetail/101810，可在页面中找到本章节对应实践代码。

### 3.5.1 数据准备

鸢尾花(Iris)数据集是机器学习领域一个非常经典的数据集，鸢尾花数据集共收集三类鸢尾花，即 Iris Setosa、Iris Versicolour 和 Iris Virginica，每类鸢尾花包含 50 条样本记录，共计 150 条。数据集包含 4 个属性，分别为花萼长度、花萼宽度、花瓣长度和花瓣宽度，单位

cm。数据示例如图 3.4 所示。

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
```

图 3.4 鸢尾花分类数据集示例

首先导入必要的 Python 包，如代码清单 3.1 所示，其中 numpy 是 Python 提供的数据处理和分析工具，sklearn 是 Python 提供的非常强力的机器学习库，灵活使用这些库可以极大地节省编写代码的时间。Matplotlib 为 Python 绘图库，可方便绘制折线图、散点图等图形。

**代码清单 3.1 导入必要的包**

```
import numpy as np
from matplotlib import colors
from sklearn import svm
from sklearn.svm import SVC
from sklearn import model_selection
import matplotlib.pyplot as plt
import matplotlib as mpl
```

如代码清单 3.2 所示。完成数据集的加载，并随机挑选出训练集和测试集作为模型训练使用，其中测试集占比 30%。

**代码清单 3.2 加载数据**

```
#加载数据
data = np.loadtxt('/home/aistudio/data/data5423/iris.data',
                  dtype = float,
                  delimiter = ',',
                  converters = {4:iris_type})
#数据分割
x, y = np.split(data,4,),axis = 1)
x = x[:, :3]
x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y,random_state = 1,test_
size = 0.3)
```

其中，使用 iris_type()方法将目标值 Iris-setosa、Iris-versicolor、Iris-virginica 分别转换为 0、1、2。如代码清单 3.3 所示。

代码清单 3.3　目标值转换

```
def iris_type(s):
    it = {b'Iris-setosa':0, b'Iris-versicolor':1, b'Iris-virginica':2}
    return it[s]
```

### 3.5.2　配置模型

**支持向量机(Support Vector Machine,SVM)**是机器学习中解决分类问题的一种常见模型,本节采用 sklearn 提供的 SVM 函数进行计算。模型定义如代码清单 3.4 所示。

代码清单 3.4　SVM 模型定义

```
def classifier():
    clf = svm.SVC(C=0.5,
                  kernel='linear',
                  decision_function_shape='ovr')
    return clf
```

其中,$C$ 表示错误项的惩罚系数,是 0～1 的浮点数,默认 1.0。$C$ 越大,即对分错样本的惩罚程度越大,趋向于对数据全部分类正确,这样在训练集中准确率越高,但是泛化能力就会降低;相反,$C$ 越小,惩罚力度减小,允许分类错误,将错误分类的样本当作噪声处理,泛化能力就会较强。

kernel 参数表示核函数,核函数可以简化 SVM 中的运算。常用的核函数包括以下几种:线性核函数 linear、高斯核函数 rbf、多项式核函数 poly 等。

decision_function_shape 参数表示决策函数(样本到分离超平面的距离)的类型,取值'ovo','ovr'或 None,默认 None。

### 3.5.3　模型训练

在定义好模型后,就可以使用训练数据进行模型训练。训练过程如代码清单 3.5 所示。

代码清单 3.5　模型训练

```
clf = classifier()
clf.fit(x_train,
        y_train.ravel())
```

在训练好模型后,需要加载模型对结果进行预测,并对预测结果的准确度进行评估。预测和评估如代码清单 3.6 所示。

代码清单 3.6 准确度评估

```
def show_accuracy(a, b, tip):
    acc = a.ravel() == b.ravel()
    print('%s Accuracy: %.3f' %(tip, np.mean(acc)))
def print_accuracy(clf, x_train, y_train, x_test, y_test):
    print('traing prediction: %.3f' % clf.score(x_train, y_train))
    print('test data prediction: %.3f' % clf.score(x_test, y_test))
    show_accuracy(clf.predict(x_train), y_train, 'traing data')
    show_accuracy(clf.predict(x_test), y_test, 'testing data')
    print('decision_function:\n', clf.decision_function(x_train))
```

### 3.5.4 数据可视化

模型训练完成后，使用训练好的分类器模型对鸢尾花分布进行可视化展示，如代码清单 3.7 所示。

代码清单 3.7 数据可视化

```
def draw(clf,x):
    iris_feature = 'sepal length', 'sepal width', 'petal lenght', 'petal width'
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max()
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max()
    x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j]
    grid_test = np.stack((x1.flat, x2.flat), axis = 1)
    z = clf.decision_function(grid_test)
    grid_hat = clf.predict(grid_test)
    grid_hat = grid_hat.reshape(x1.shape)
    cm_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
    cm_dark = mpl.colors.ListedColormap(['g', 'b', 'r'])
    plt.pcolormesh(x1, x2, grid_hat, cmap = cm_light)
    plt.scatter(x[:, 0], x[:, 1], c = np.squeeze(y), edgecolor = 'k', s = 50, cmap = cm_dark)
    plt.scatter(x_test[:, 0], x_test[:, 1], s = 120, facecolor = 'none', zorder = 10)
    plt.xlabel(iris_feature[0], fontsize = 20)
    plt.ylabel(iris_feature[1], fontsize = 20)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.title('svm in iris data classification', fontsize = 30)
    plt.grid()
    plt.show()
```

最终鸢尾花分布如图 3.5 所示。

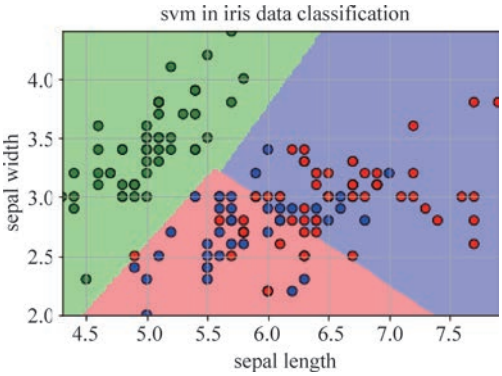


图 3.5 鸢尾花分类效果

## 3.6 习题

1. 对于分类问题能不能用平方误差？试分析其中原因。

2. 回顾第一章的习题 4(线性可分支持向量机)，回答：求解带约束的优化问题是如何变为最小化结构风险的？为什么代价函数是 Hinge 损失？

3. 如果有 $N$ 个样本服从正态分布，其中分布的均值 $\mu$ 未知。

(1) 请使用极大似然估计求解均值 $\mu$ 的最优值。

(2) 如果均值 $\mu$ 也为随机变量，并且服从正态分布 $N(\mu_0,\sigma_0^2)$，请使用极大后验估计求解均值 $\mu$ 的最优值。

(3) 试证明，为什么当样本数量足够大的时候，最大后验估计等于极大似然估计。

4. 对于一个四分类问题，模型的预测结果以及真实标签如下所示：

| 预测值 | 1 | 2 | 3 | 4 | 1 | 3 | 4 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|
| 真实值 | 1 | 2 | 3 | 4 | 4 | 2 | 4 | 2 | 1 |

请求出模型的精确率、召回率、F1 值以及它们的宏平均与微平均。