第三章　机器学习概述

机器学习是对能通过经验自动改进的计算机算法的研究。

— ?

在介绍人工神经网络之前，我们先来了解下机器学习的基本概念。机器学习主要是研究如何使计算机从给定的数据中学习规律，即从观测数据（样本）中寻找规律，并利用学习到的规律（模型）对未知或无法观测的数据进行预测。

狭义地讲，机器学习是给定一些训练样本 $(x_i, y_i), 1 \leq i \leq N$ （其中，x_i 是输入，y_i 是需要预测的目标），让计算机自动寻找一个**决策函数** $f(\cdot)$ 来建立 x_i 和 y_i 之间的关系。这样对于一个新的输入 x，我们可以通过决策函数来预测目标 y。

$$y = f(\boldsymbol{\phi}(x), \theta), \tag{3.1}$$

这里，θ 表示决策函数 $f(\cdot)$ 的参数，$\boldsymbol{\phi}(x)$ 表示样本 x 对应的特征表示。如果 x 是已经经过特征提取后的变量，则 $\boldsymbol{\phi}(x) = x$。公式3.1也可以直接写为

$$y = f(x, \theta). \tag{3.2}$$

机器学习系统的示例见图3.1。首先，对一个预测任务，我们需要选择一个模型，即决策函数 $f(x, \theta)$，其中参数 θ 还没有确定。我们需要通过学习算法在一组训练样本上来得到一个最优的参数 θ^*。这个过程叫做机器学习的**训练**过程。有了模型 $f(x, \theta^*)$，我们就可以对任何输入 x 进行预测。

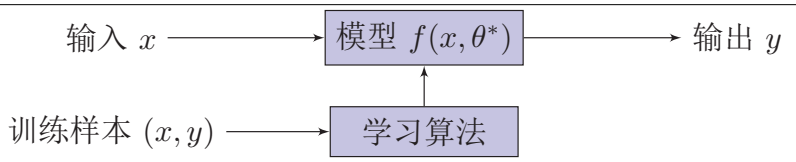


图 3.1: 机器学习系统示例

3.1 机器学习基本概念

3.1.1 机器学习算法类型

到目前为止已经非常多的机器学习模型。按照训练数据提供的信息以及反馈方式的不同，机器学习算法一般可以分为以下几类：

1. 有监督学习（Supervised Learning）是利用一组已知输入 x 和目标标签 y 的数据来学习模型的参数，使得模型预测的目标标签和真实标签尽可能的一致。

   根据目标标签的类型不同，有监督学习又可以分为**回归**和**分类**两类。

   (a) **回归**（Regression）问题：目标标签 y 是连续值（实数或连续整数），$f(x)$ 的输出也是连续值。对于所有已知或未知的 (x,y)，使得 $f(x,\theta)$ 和 y 尽可能地一致。

   (b) **分类**（Classification）问题：目标标签 y 是离散的类别（符号）。在分类问题中，通过学习得到的决策函数 $f(x,\theta)$ 也叫**分类器**。分类问题根据其类别数量可分为**两类**分类（binary classification）和**多类**分类（multiclass classification）。

2. **无监督学习**（Unsupervised Learning）是用来学习的数据不包含目标标签，需要学习算法自动学习到一些有价值的信息。一个典型的无监督学习问题就是**聚类**（Clustering）。

3. **增强学习**（Reinforcement Learning）也叫强化学习，强调如何基于环境做出一系列的动作，以取得最大化的累积收益。每做出一个动作，并不一定立刻得到收益。增强学习和有监督学习的不同在于增强学习不需要显式地以输入/输出对的方式给出训练样本，是一种在线的学习机制。

有监督的学习方法需要每个数据记录都有类标号，而无监督的学习方法则不考虑任何指导性信息。一般而言，一个监督学习模型需要大量的有标记数据集，而这些数据集是需要人工标注的。因此，也出现了很多**弱监督学习**和**半监督学习**的方法，希望从大规模的未标记数据中充分挖掘有用的信息，降低对标记数据数量的要求。

3.1.2 风险函数与损失函数

此外，我们还要建立一些准则来衡量决策函数的好坏。在很多机器学习算法中，一般是定义一个**损失函数** $\mathcal{L}(y, f(x,\theta))$，然后在所有的训练样本上来评价决策函数的风险。

损失函数

给定一个实例 (x, y)，真实目标是 y，机器学习模型的预测为 $f(x,\theta)$。如果预测错误时（$f(x,\theta) \neq y$），我们需要定义一个度量函数来定量地计算错误的程度。常见的损失函数有如下几类：

0-1 损失函数 0-1 损失函数（0-1 loss function）是

$$\mathcal{L}(y, f(x,\theta)) = \begin{cases} 0 & \textbf{if } y = f(x,\theta) \\ 1 & \textbf{if } y \neq f(x,\theta) \end{cases} \tag{3.3}$$

$$= I(y \neq f(x,\theta)), \tag{3.4}$$

这里 I 是指示函数。

平方损失函数 平方损失函数（quadratic loss function）是

$$\mathcal{L}(y, \hat{y}) = (y - f(x,\theta))^2 \tag{3.5}$$

交叉熵损失函数 对于分类问题，预测目标 y 为离散的类别，模型输出 $f(x,\theta)$ 为每个类的条件概率。

假设 $y \in \{1, \cdots C\}$，模型预测的第 i 个类的条件概率 $P(y = i|x) = f_i(x, \theta)$，则 $f(x, \theta)$ 满足

$$f_i(x, \theta) \in [0, 1], \qquad \sum_{i=1}^{C} f_i(x, \theta) = 1 \tag{3.6}$$

$f_y(x, \theta)$ 可以看作真实类别 y 的似然函数。参数可以直接用最大似然估计来优化。考虑到计算问题，我们经常使用最小化负对数似然，也就是**负对数似然损失函数**（Negative Log Likelihood function）。

$$\mathcal{L}(y, f(x, \theta)) = -\log f_y(x, \theta). \tag{3.7}$$

如果我们用 one-hot 向量 $\mathbf{y}$ 来表示目标类别 c，其中只有 $y_c = 1$，其余的向量元素都为 0。

负对数似然函数也可以写为：

$$\mathcal{L}(y, f(x, \theta)) = -\sum_{i=1}^{C} y_i \log f_i(x, \theta). \tag{3.8}$$

y_i 也也可以看成是真实类别的分布，这样公式3.8恰好是交叉熵的形式。因此，负对数似然损失函数也常叫做**交叉熵损失函数**（Cross Entropy Loss function）是负对数似然函数的一种改进。

Hinge损失函数 对于两类分类问题，假设 y 和 $f(x, \theta)$ 的取值为 $\{-1, +1\}$。Hinge 损失函数（Hinge Loss Function）的定义如下：

$$\mathcal{L}(y, f(x, \theta)) = \max(0, 1 - yf(x, \theta)) \tag{3.9}$$

$$= |1 - yf(x, \theta)|_+. \tag{3.10}$$

风险函数

$$\mathcal{R}_{emp}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, f(x^{(i)}, \theta)). \tag{3.11}$$

这里，风险函数 $\mathcal{R}_{emp}(\theta)$ 是在已知的训练样本（经验数据）上计算得来的，因此被称为**经验风险**。用对参数求经验风险来逐渐逼近理想的期望风险的最小

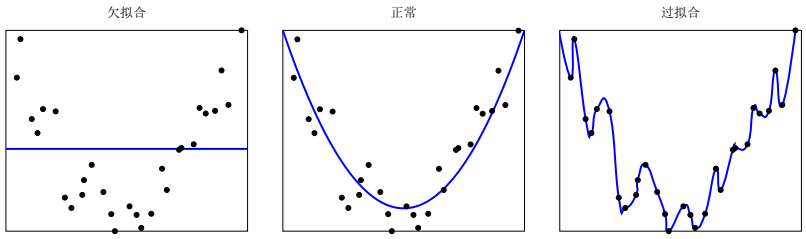


图 3.2: 欠拟合和过拟合示例

值，就是我们常说的**经验风险最小化原则**（Empirical Risk Minimization）。这样，我们的目标就是变成了找到一个参数 θ^* 使得经验风险最小。

$$\theta^* = \arg\min_{\theta} \mathcal{R}_{emp}(\theta). \tag{3.12}$$

因为用来训练的样本往往是真实数据的一个很小的子集或者包含一定的噪声数据，不能很好地反映全部数据的真实分布。经验风险最小化原则很容易导致模型在训练集上错误率很低，但是在未知数据上错误率很高。这就是所谓的**过拟合**。过拟合问题往往是由于训练数据少和噪声等原因造成的。过拟合的标准定义为：给定一个假设空间 H，一个假设 h 属于 H，如果存在其他的假设 h 属于 H, 使得在训练样例上 h 的损失比 h 小，但在整个实例分布上 h 比 h 的损失小，那么就说假设 h 过度拟合训练数据 [**?**]。

和过拟合相对应的一个概念是**泛化错误**。泛化错误是衡量一个机器学习模型是否可以很好地泛化到未知数据。泛化错误一般表现为一个模型在训练集和测试集上错误率的差距。

为了解决过拟合问题，一般在经验风险最小化的原则上上加参数的**正则化**（Regularization），也叫**结构风险最小化原则**（Structure Risk Minimization）。

$$\theta^* = \arg\min_{\theta} \mathcal{R}_{struct}(\theta) \tag{3.13}$$

$$= \arg\min_{\theta} \mathcal{R}_{emp}(\theta) + \lambda\|\theta\|_2^2 \tag{3.14}$$

$$= \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y^{(i)}, f(x^{(i)}, \theta)) + \lambda\|\theta\|^2. \tag{3.15}$$

这里，$\|\theta\|_2$ 是 L_2 范数的**正则化项**，用来减少参数空间，避免**过拟合**。λ 用来控制正则化的强度。

正则化项也可以使用其它函数，比如 L_1 范数。L_1 范数的引入通常会使得参数有一定稀疏性，因此在很多算法中也经常使用。在 Bayes 估计的角度来讲，正则化是假设了参数的先验分布，不完全依赖训练数据。

3.1.3 常见的基本概念

上述的关于机器学习的介绍中，提及了一些基本概念，比如“数据”，“样本”，“特征”，“数据集”等。我们首先来解释下这些概念。

数据

在计算机科学中，**数据**是指所有能计算机程序处理的对象的总称，可以是数字、字母和符号等。在不同的任务中，表现形式不一样，比如图像、声音、文字、传感器数据等。

特征

机器学习中很多算法的输入要求是数学上可计算的。而在现实世界中，原始数据通常是并不都以连续变量或离散变量的形式存在的。我们首先需要将抽取出一些可以表征这些数据的数值型特征。这些数值型特征一般可以表示为向量形式，也称为**特征向量**。

参数与超参数

机器学习可以归结为学习一个映射函数 $f : \mathbf{x} \to y$，将输入变量 $\mathbf{x}$ 映射为输出变量 y。一般我们可以假设映射函数为 $y = f(x, \theta)$。其中 θ 即为函数的**参数**。参数可以通过学习算法进行学习。

超参数可以理解为参数的参数。

特征学习

数据的原始表示转换为。原始数据的特征有很多，但是并不是所有的特征都是有用的。并且，很多特征通常是冗余并且易变的。我们需要抽取有效的、稳定的特征。传统的特征提取是通过人工方式进行的，这需要大量的人工和专家

知识。即使这样，人工总结的特征在很多任务上也不能满足需要。因此，如何自动地学习有效的特征也成为机器学习中一个重要的研究内容，也就是**特征学习**，也叫**表示学习**。特征学习分成两种，一种是**特征选择**，是在很多特征集合选取有效的子集；另一种是**特征提取**，是构造一个新的特征空间，并将原始特征投影在新的空间中。

样本

样本是按照一定的抽样规则从全部数据中取出的一部分数据，是实际观测得到的数据。在有监督学习中，需要提供一组有输出目标的样本用来学习模型以及检验模型的好坏。

训练集和测试集

一组样本集合就称为**数据集**。在很多领域，数据集也经常称为**语料库**。为了检验机器学习算法的好坏，一般将数据集分为两部分：训练集和测试集。训练集用来进行模型学习，测试集用来进行模型验证。通过学习算法，在训练集得到一个模型，这个模型可以对测试集上样本 x 预测一个类别标签 $\hat{y}$。假设测试集为 T, 模型的正确率为：

$$Acc = \frac{1}{|T|} \sum_{(x_i, y_i) \in T} |\hat{y}_i = y_i|, \tag{3.16}$$

其中 $|T|$ 为测试集的大小。第3.3节中会介绍更多的评价方法。

正例和负例

对于两类分类问题，类别可以表示为 $\{+1, -1\}$，或者直接用正负号表示。因此，常用**正例**和**负例**来分别表示属于不同类别的样本。

判别函数

经过特征抽取后，一个样本可以表示为 k 维特征空间中的一个点。为了对这个特征空间中的点进行区分，就需要寻找一些超平面来将这个特征空间分为一些互不重叠的子区域，使得不同类别的点分布在不同的子区域中，这些超平面就成为判别界面。

为了定义这些用来进行空间分割的超平面，就需要引入判别函数的概念。假设变量 $\mathbf{z} \in \mathbb{R}^m$ 为特征空间中的点，这个超平面由所有满足函数 $f(\mathbf{z}) = 0$ 的点组成。这里的 $f(\mathbf{z})$ 就称为**判别函数**。

有了判别函数，分类就变得很简单，就是看一个样本在特征空间中位于哪个区域，从而确定这个样本的类别。

3.1.4 参数学习算法

学习算法就是如何从训练集的样本中，自动学习决策函数的参数。不同机器学习算法的区别在于决策函数和学习算法的差异。相同的决策函数可以有不同的学习算法。比如，线性分类器有感知器，logistic 回归以及支持向量机等。感知器使用在线被动更新方法来学习参数，logistic 回归使用梯度下降法来学习参数，而支持向量机使用 SMO（Sequential minimal optimization）优化算法来学习参数。

通过一个学习算法进行自动学习参数的过程也叫作**训练过程**。

在选择合适的风险函数后，我们寻找一个参数 θ^*，使得风险函数最小化。

$$\theta^* = \arg\min_{\theta} \mathcal{R}(\theta) \tag{3.17}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, f(x^{(i)}, \theta)\big). \tag{3.18}$$

这样，我们就将机器学习问题转化成为一个最优化问题。

批量随机梯度下降法

在最优化问题中，最常用的方法就是梯度下降法。

梯度下降是求得所有样本上的风险函数最小值，也叫做**批量梯度下降法**。

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t} \tag{3.19}$$

$$= \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{R}\big(\theta_t; x^{(i)}, y^{(i)}\big)}{\partial \theta}. \tag{3.20}$$

在机器学习，搜索步长 α 中也叫作**学习率**（Learning Rate）。

梯度下降法见第??页第??节。

随机梯度下降法

若样本个数 N 很大，输入 $\mathbf{x}$ 的维数也很大时，那么批量梯度下降法每次迭代要处理所有的样本，效率会较低。为此，有一种改进的方法即**随机梯度下降法**。

随机梯度下降法（Stochastic Gradient Descent，SGD）也叫**增量梯度下降**，每个样本都进行更新

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{L}\big(\theta_t; x^{(t)}, y^{(t)}\big)}{\partial \theta}, \tag{3.21}$$

$x^{(t)}, y^{(t)}$ 是第 t 次迭代选取的样本。

批量梯度下降和随机梯度下降之间的区别在于每次迭代的风险是对所有样本汇总的风险还是单个样本的风险。随机梯度下降因为实现简单，收敛速度也非常快，因此使用非常广泛。

还有一种折中的方法就是小批量（Mini-Batach）随机梯度下降法，每次迭代时，只采用一小部分的训练样本，兼顾了批量梯度下降和随机梯度下降的优点。

提前停止

在梯度下降训练的过程中，由于过拟合的原因，在训练样本上收敛的参数，并不一定在测试集上最优。因此，我们使用一个**验证集**（Validation Dataset）（也叫**开发集**（Development Dataset））来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。这种策略叫**提前停止**（Early-Stop）。如果没有验证集，可以在训练集上进行**交叉验证**。

学习率设置

在梯度下降中，学习率的取值非常关键，如果过大就不会收敛，如果过小则收敛速度太慢。一般步长可以由线性搜索算法来确定。在机器学习中，经常使用自适应调整学习率的方法。

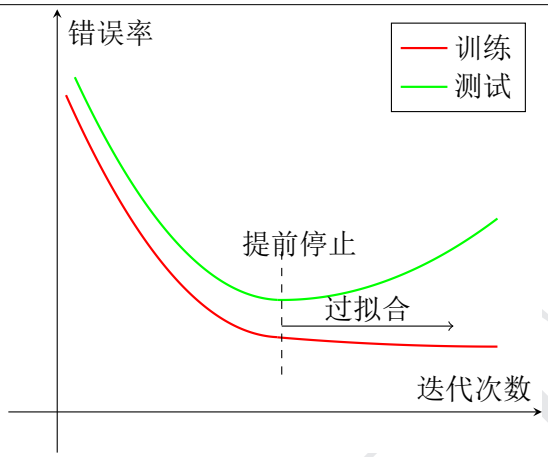


图 3.3: 前提停止

动量法 动量法（Momentum Method）[?] 是对当前迭代的更新中加入上一次迭代的更新。我们记 $\nabla\theta_t = \theta_t - \theta_{t-1}$。在第 t 迭代时，

$$\theta_t = \theta_{t-1} + (\rho\nabla\theta_{t-1} - \lambda g_t), \tag{3.22}$$

其中，ρ 为动量因子，通常设为 0.9。这样，在迭代初期，使用前一次的梯度进行加速。在迭代后期的收敛值附近，因为两次更新方向基本相反，增加稳定性。

从某种角度来说，当前梯度叠加上部分的上次梯度，一定程度上可以近似看作二阶梯度。

优化、二阶梯度

AdaGrad AdaGrad（Adaptive Gradient）算法 [?] 是借鉴 L2 正则化的思想。在第 t 迭代时，

$$\theta_t = \theta_{t-1} - \frac{\rho}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t, \tag{3.23}$$

其中，ρ 是初始的学习率，$g_\tau \in \mathbb{R}^{|\theta|}$ 是第 τ 次迭代时的梯度。

随着迭代次数的增加，梯度逐渐缩小。

AdaM

AdaDelta AdaDelta 算法 [?] 用指数衰减的移动平均来累积历史的梯度信息。第 t 次迭代的梯度的期望 $E(g^2)_t$ 为：

$$E(g^2)_t = \rho E(g^2)_{t-1} + (1-\rho) g_t^2, \tag{3.24}$$

其中，ρ 是衰减常数。

本次迭代的更新为

$$\nabla\theta_t = -\frac{\sqrt{E(\nabla\theta^2)_{t-1} + \epsilon}}{\sqrt{E(g^2)_t + \epsilon}} g_t \tag{3.25}$$

其中，$E(\nabla\theta^2)_t$ 为前一次迭代时 $\nabla\theta^2$ 的移动平均，ϵ 为常数。

累计更新本次迭代 $\nabla\theta^2$ 的移动平均

$$E(\nabla\theta^2)_t = \rho E(\nabla\theta^2)_{t-1} + (1-\rho)\nabla\theta_t^2. \tag{3.26}$$

最后更新参数

$$\theta_t = \theta_{t-1} + \nabla\theta_t. \tag{3.27}$$

3.2 常见机器学习算法

本节介绍一些常见的机器学习算法。

3.2.1 线性回归

如果输入 $\mathbf{x}$ 是列向量，目标 y 是连续值（实数或连续整数），预测函数 $f(\mathbf{x})$ 的输出也是连续值。这种机器学习问题是回归问题。

如果我们定义 $f(\mathbf{x})$ 是线性函数，

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \tag{3.28}$$

这就是**线性回归**问题（Linear Regression）。

为了简单起见，我们将公式3.28写为

$$f(\mathbf{x}) = \hat{\mathbf{w}}^{\mathrm{T}}\hat{\mathbf{x}}, \tag{3.29}$$

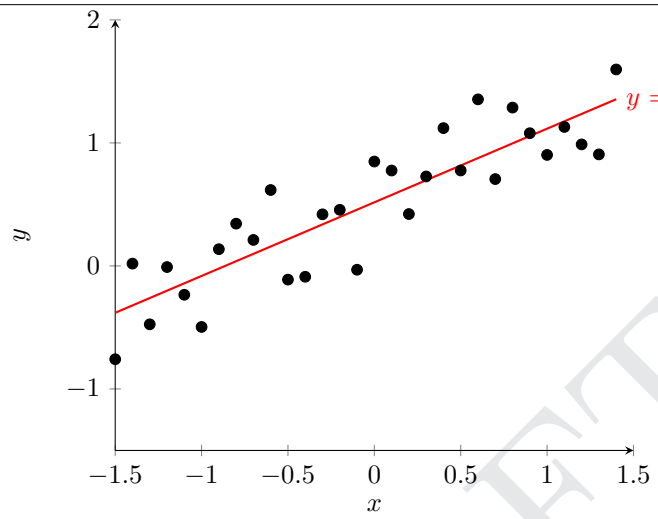


图 3.4: 线性回归示例

其中 $\hat{\mathbf{w}}$ 和 $\hat{\mathbf{x}}$ 分别称为**增广权重向量**和**增广特征向量**。

$$\hat{\mathbf{x}} = \mathbf{x} \oplus 1 \triangleq \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{3.30}$$

$$\hat{\mathbf{w}} = \mathbf{w} \oplus b \triangleq \begin{bmatrix} w_1 \\ \vdots \\ w_k \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}, \tag{3.31}$$

这里，$\oplus$ 定义为两个向量的拼接操作。

不失一般性，在本章后面的描述中我们采用简化的表示方法，直接用 $\mathbf{w}$ 和 $\mathbf{x}$ 来表示增广权重向量和增广特征向量。

线性回归的损失函数通常定义为平方损失函数。给定 N 给样本 $(\mathbf{x}^{(i)}, y^{(i)}), 1 \leq i \leq N$，模型的经验风险为

平方损失参见第3.1.2节

$$\mathcal{R}(\mathbf{y}, f(X, \mathbf{w})) = \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, f(\mathbf{x}^{(i)}, \mathbf{w})) \tag{3.32}$$

$$= \sum_{i=1}^{N} (\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} - y^{(i)})^2 \tag{3.33}$$

$$= \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2, \tag{3.34}$$

其中，$\mathbf{y} \in \mathbb{R}^N$ 是一个由目标值 $y^{(1)}, \cdots, y^{(N)}$ 组成的列向量，$X \in \mathbb{R}^{(k+1)\times N}$ 是所有输入 $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}$ 组成的矩阵

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_k^{(1)} & x_k^{(2)} & \cdots & x_k^{(N)} \\ 1 & 1 & \cdots & 1 \end{pmatrix}. \tag{3.35}$$

要最小化 $\mathcal{R}(\mathbf{y}, f(X, \mathbf{w}))$，我们要计算 $\mathcal{R}(\mathbf{y}, f(X, \mathbf{w}))$ 对 $\mathbf{w}$ 的导数

$$\frac{\partial \mathcal{R}\big(Y, f(X, \mathbf{w})\big)}{\partial \mathbf{w}} = \frac{\partial \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2}{\partial \mathbf{w}} \tag{3.36}$$

$$= X(X^{\mathrm{T}}\mathbf{w} - \mathbf{y}). \tag{3.37}$$

让 $\frac{\partial \mathcal{R}(\mathbf{y}, f(X, \mathbf{w}))}{\partial \mathbf{w}} = 0$，则可以得到

$$\mathbf{w} = (XX^{\mathrm{T}})^{-1}X\mathbf{y} \tag{3.38}$$

$$= \big(\sum_{i=1}^{N} \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^{\mathrm{T}}\big)^{-1}\big(\sum_{i=1}^{N} \mathbf{x}^{(i)}y^{(i)}\big). \tag{3.39}$$

这里要求 XX^T 是满秩的，存在逆矩阵，也就是要求 $\mathbf{x}$ 的每一维之间是线性不相关的。这样的参数求解方法也叫**最小二乘法估计**。

图3.4给出了用最小二乘法估计方法来进行参数学习的示例。

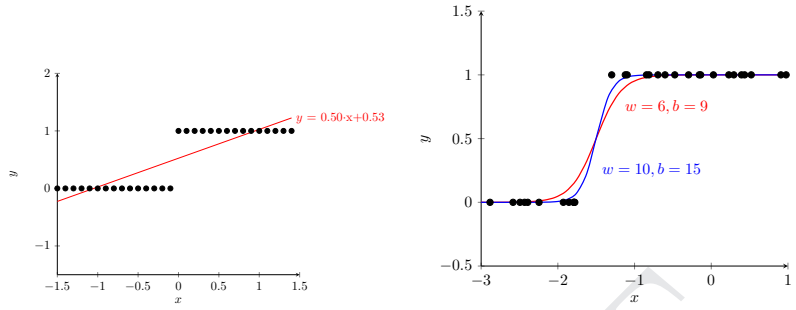
如果 XX^T 不可求逆矩阵，说明在训练数据上，输入的不同特征之间是线性相关的。我们可以预处理数据，先使用主成分分析等特征提取方法来消除不同特征之间的相关性，然后再使用最小二乘估计方法来求解。

另外一种方法是通过用梯度下降法来求解。初始化 $\mathbf{w}_0 = 0$，迭代公式为：

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \frac{\partial \mathcal{R}\big(Y, f(X, \mathbf{w})\big)}{\partial \mathbf{w}} \tag{3.40}$$

$$= \mathbf{w}_t + \alpha X(X^{\mathrm{T}}\mathbf{w} - \mathbf{y}), \tag{3.41}$$

其中 α 是学习率。



(a) 线性回归 (b) Logistic 回归

图 3.5: 一维两类问题示例

3.2.2 Logistic 回归

Logistic 回归（Logistic Regression）是一种两类分类问题的参数学习算法。和回归问题不同，分类问题中的目标标签 y 是离散的类别（符号）。因此，分类问题中的决策函数 $f(\mathbf{x}, \theta)$ 需要输出离散值，这一般可以通过一个连续判别函数加上一个阈值函数来实现。

我们先来看两类分类问题，其预测的目标标签 y 只有两种取值。一般可设置为 $y \in \{0, 1\}$ 或者 $y \in \{-1, +1\}$。

对于分类问题，使用线性回归算法来求解是不合适的。图3.5a使用线性回归算法来解决一维的两类分类问题示例。

线性分类器是机器学习中应用最广泛的一类分类器，使用线性函数来作为判别函数。

$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \\ 0 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} \leq 0 \end{cases} \tag{3.42}$$

$$= I(\mathbf{w}^{\mathrm{T}}\mathbf{x} > 0), \tag{3.43}$$

其中，$I()$ 是指示函数。

公式3.65定义了一个两类分类问题的线性判别函数。在高维的特征空间中，所有满足 $\mathbf{f}(z) = 0$ 的点组成用一个超平面，这个超平面将特征空间一分为二，划分成两个区域。这两个区域分别对应两个类别。

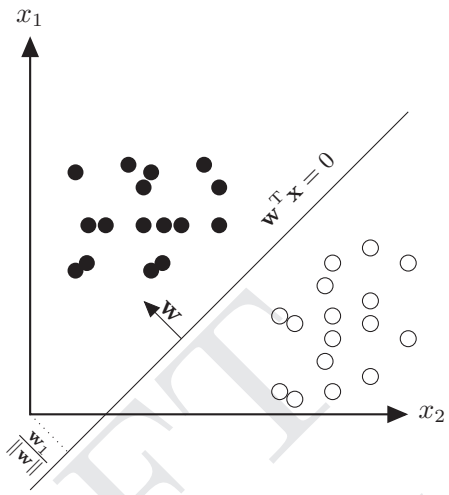


图 3.6: 两类分类线性判别函数

图3.6中给了一个两维数据的判别函数以及对应的判别界面。在二维空间中，分类界面为一个直线。在三维空间中，分类界面为一个平面。在高维空间中，分类界面为一个超平面。对于线性函数来说，权重向量在线性空间中垂直于分类界面的向量。

线性分类函数的参数 $\mathbf{w}$ 有很多种学习方式，比如第四章中介绍的感知器。这里使用另一种常用的学习算法：**Logistic 回归**。

我们定义目标类别 $y=1$ 的后验概率为：

$$P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}) \tag{3.44}$$

$$\triangleq \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})}, \tag{3.45}$$

其中，$\sigma(\cdot)$ 为 logistic 函数，$\mathbf{x}$ 和 $\mathbf{w}$ 为增广的输入向量和权重向量。

$y=0$ 的后验概率为

$$P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x}) \tag{3.46}$$

$$= \frac{\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})}. \tag{3.47}$$

根据公式3.45，

$$\mathbf{w}^{\top}\mathbf{x} = \log \frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})} \tag{3.48}$$

$$= \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}. \tag{3.49}$$

基于梯度下降法的训练算法

给定 N 给样本 $(x^{(i)}, y^{(i)}), 1 \leq i \leq N$，我们使用交叉熵损失函数，模型在训练集的风险函数为：

$$\mathcal{R}(\mathbf{w}) = -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right)\right) \tag{3.50}$$

$$= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) + (1-y^{(i)})\log\left(1-\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right)\right) \tag{3.51}$$

$$= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) + (1-y^{(i)})\log\left(\frac{1}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right)\right) \tag{3.52}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}y^{(i)} - \log\left(1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right)\right) \tag{3.53}$$

$$\tag{3.54}$$

采样梯度下降法，$\mathcal{R}(\mathbf{w})$ 关于 $\mathbf{w}$ 的梯度为：

$$\frac{\partial \mathcal{R}(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)}\cdot\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\cdot\frac{1}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{3.55}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)}\cdot\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{3.56}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}\cdot\left(y^{(i)} - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right)\right) \tag{3.57}$$

$$= \sum_{i=1}^{N} \left(\mathbf{x}^{(i)} \cdot \left(\sigma(\mathbf{w}^{\mathrm{T}} \mathbf{x}^{(i)}) - y^{(i)} \right) \right) \tag{3.58}$$

我们可以初始化 $\mathbf{w}_0 = 0$，然后用梯度下降法进行更新

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda \frac{\partial \mathcal{R}(\mathbf{w})}{\partial \mathbf{w}}, \tag{3.59}$$

其中 λ 是学习率。

3.2.3 Softmax 回归

Softmax 回归是 logistic 回归的多类推广。

多类分类问题

对于多类分类问题（假设类别数为 $C(C > 2)$），有很多种利用判别函数的方法。比较常用的是把多类分类问题转换为两类分类问题，在得到两类分类结果后，还需要通过投票方法进一步确定多类的分类结果。一般有两种多类转两类的转换方式：

1. 把多类分类问题转换为 C 个两类分类问题，构建 C 个一对多的分类器。每个两类分类问题都是把某一类和其他类用一个超平面分开。
2. 把多类分类问题转换为 $C(C-1)/2$ 个两类分类问题，构建 $C(C-1)/2$ 个两两分类器。每个两类分类问题都是把 C 类中某两类用一个超平面分开。

当对一个样本进行分类时，首先用多个两个分类器进行分类，然后进行投票，选择一个得分最高的类别。

但是上面两种转换方法都存在一个缺陷：空间中的存在一些区域，这些区域中点的类别是不能区分确定的。因为如果用多个两个分类器对这些点进行分类，然后进行投票时，会发现有两个或更多个类别的得分是一样的。

为了避免上述缺陷，可以使用一个更加有效的决策规则，直接建立多类线性分类器。假设 $y = \{1, \cdots, C\}$ 共 C 个类别，首先定义 C 个判别函数：

$$f_c(\mathbf{x}) = \mathbf{w}_c^{\mathrm{T}} \mathbf{x}, \qquad c = 1, \cdots, C, \tag{3.60}$$

这里 $\mathbf{w}_c$ 为类 c 的权重向量。

这样，对于空间中的一个点$\mathbf{x}$，如果存在类别c，对于所有的其他类别$\tilde{c}(\mathbf{w}_c^T\mathbf{x} \neq c)$都满足$\mathbf{f}_c(\mathbf{x}) > \mathbf{f}_{\tilde{c}}(\mathbf{x})$，那么$\mathbf{x}$属于类别$c$。相应的分类函数可以表示为：

$$\hat{y} = \arg\max_{c=1}^{C} \mathbf{w}_c^{\mathrm{T}}\mathbf{x} \tag{3.61}$$

当$C = 2$时，

$$\begin{aligned} \hat{y} &= \arg\max_{y\in\{0,1\}} f_y(\mathbf{x}) \\ &= I(f_1(\mathbf{x}) - f_0(\mathbf{x}) > 0) \\ &= I(\mathbf{w}_1^{\mathrm{T}}\mathbf{x} - \mathbf{w}_0^{\mathrm{T}}\mathbf{x} > 0) \\ &= I((\mathbf{w}_1 - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x} > 0) \end{aligned} \tag{3.62}$$

这里，$I()$是指示函数。对比公式3.65中的两类分类判别函数，可以发现$\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$。

多类线性分类函数的参数$\mathbf{w}$也有很多种学习方式。这里我们介绍一种常用的学习算法：**softmax 回归**。在softmax回归中，机器学习模型预测目标为每一个类别的后验概率。这就需要用到softmax函数。

利用softmax函数，我们定义目标类别$y = c$的后验概率为：

$$P(y = c|\mathbf{x}) = \mathrm{softmax}(\mathbf{w}_c^{\mathrm{T}}\mathbf{x}) \tag{3.63}$$

$$= \frac{\exp(\mathbf{w}_c^{\top}\mathbf{x})}{\sum_{i=1}^{C}\exp(\mathbf{w}_i^{\top}\mathbf{x})}. \tag{3.64}$$

对于样本$(\mathbf{x}, y)$，输出目标$y = \{1, \cdots, C\}$，我们用C维的one-hot向量$\mathbf{y}$来表示输出目标。对于类别c，

$$\mathbf{y} = [I(1 = c), I(2 = c), \cdots, I(C = c)]^{\mathrm{T}}, \tag{3.65}$$

这里，$I()$是指示函数。

同时，我们将公式3.63重新定义一下，直接输出k维向量。

$$\begin{aligned} \hat{\mathbf{y}} &= \mathrm{softmax}(W^{\mathrm{T}}\mathbf{x}) \\ &= \frac{\exp(W^{\mathrm{T}}\mathbf{x})}{\underline{1}^{\mathrm{T}}\exp\left((W^{\mathrm{T}}\mathbf{x})\right)} \\ &= \frac{\exp(\mathbf{z})}{\underline{1}^{\mathrm{T}}\exp\left(\mathbf{z}\right)}, \end{aligned} \tag{3.66}$$

其中，$W = [\mathbf{w}_1, \cdots, \mathbf{w}_C]$是$C$个类对应权重向量组成的矩阵。$\hat{\mathbf{y}}$的第$c$维的值是第$c$类的预测后验概率。其中，$\mathbf{z} = W^{\mathrm{T}}\mathbf{x}$，为softmax函数的输入向量。

基于梯度下降法的训练算法

给定N给样本$(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}), 1 \leq i \leq N$，我们使用交叉熵损失函数，模型在训练集的风险函数为：

$$\begin{aligned}\mathcal{R}(W) &= -\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{y}_c^{(i)}\log\hat{\mathbf{y}}_c^{(i)} \\ &= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\hat{\mathbf{y}}^{(i)} \\ &= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\Big(\mathrm{softmax}(\mathbf{z}^{(i)})\Big) \\ &= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\Big(\mathrm{softmax}(W^{\mathrm{T}}\mathbf{x}^{(i)})\Big). \end{aligned} \tag{3.67}$$

采样梯度下降法，我们要计算$\mathcal{R}(W)$关于W的梯度。首先，我们列下要用到的公式。

（1）softmax函数的导数为

$$\frac{\partial\mathbf{y}}{\partial\mathbf{x}} = \frac{\partial\,\mathrm{softmax}(\mathbf{x})}{\partial\mathbf{x}} \tag{3.68}$$

$$= \mathbf{diag}\left(\mathrm{softmax}(\mathbf{x})\right) - \mathrm{softmax}(\mathbf{x})\,\mathrm{softmax}(\mathbf{x})^{\top} \tag{3.69}$$

$$= \mathbf{diag}\left(\mathbf{y}\right) - \mathbf{y}\mathbf{y}^{\top}. \tag{3.70}$$

（2）$\mathbf{z} = W^{\mathrm{T}}\mathbf{x}$，则

$$\frac{\partial\mathbf{z}}{\partial\mathbf{w}_c} = M(\mathbf{x}, c), \tag{3.71}$$

$M(\mathbf{x}, c)$为第c列为$\mathbf{x}$，其余为0的矩阵。

（3）$\mathbf{x}^{\top}\,\mathbf{diag}(\mathbf{x})^{-1} = 1_C^{\top}$

（4）因为$\mathbf{y}$为onehot向量，所以$1_K^{\top}\mathbf{y} = 1$

采样梯度下降法，$\mathcal{R}(W)$关于$\mathbf{w}_c$的梯度为：

$$\begin{aligned}\frac{\partial\mathcal{R}(W)}{\partial\mathbf{w}_c} &= -\sum_{i=1}^{N}\frac{\partial\left((\mathbf{y}^{(i)})^{\mathrm{T}}\log\left(\hat{\mathbf{y}}^{(i)}\right)\right)}{\partial\mathbf{w}_c} \\ &= -\sum_{i=1}^{N}\frac{\partial\mathbf{z}^{(i)}}{\partial\mathbf{w}_c}\frac{\partial\hat{\mathbf{y}}^{(i)}}{\partial\mathbf{z}^{(i)}}\frac{\partial\log\hat{\mathbf{y}}^{(i)}}{\partial\hat{\mathbf{y}}^{(i)}}\mathbf{y}^{(i)}\end{aligned}$$

$$
\begin{aligned}
&= -\sum_{i=1}^{N} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{w}_c} \frac{\partial \operatorname{softmax}(\mathbf{z}^{(i)})}{\partial \mathbf{z}^{(i)}} \frac{\partial \log \hat{\mathbf{y}}^{(i)}}{\partial \hat{\mathbf{y}}^{(i)}} \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{diag}\left(\hat{\mathbf{y}}^{(i)}\right) - \hat{\mathbf{y}}^{(i)} (\hat{\mathbf{y}}^{(i)})^{\top} \right) \left(\mathbf{diag}(\hat{\mathbf{y}}^{(i)}) \right)^{-1} \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(I - \hat{\mathbf{y}}^{(i)} \underline{1}_C^{\top} \right) \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left((\mathbf{y}^{(i)}) - \hat{\mathbf{y}}^{(i)} \underline{1}_C^{\top} \mathbf{y}^{(i)} \right) \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right) \\
&= -\sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right)_c
\end{aligned} \tag{3.72}
$$

如果采用 $y \in \{1, \cdots, C\}$ 的离散表示形式，梯度公式也可以写为：

$$
\frac{\partial \mathcal{R}(W)}{\partial \mathbf{w}_c} = -\frac{1}{N} \sum_{i=1}^{C} \mathbf{x}^{(i)} \left(I(y^{(i)} = c) - p(y^{(i)} = c | \mathbf{x}^{(i)}; W) \right), \tag{3.73}
$$

其中，$I()$ 为指示函数。

根据公式3.2.3，我们可以得到

$$
\frac{\partial \mathcal{R}(W)}{\partial W} = -\sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right)^{\top} \tag{3.74}
$$

$$
= \sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)} \right)^{\top} \tag{3.75}
$$

我们可以初始化 $W_0 = 0$，然后用梯度下降法进行更新

$$
W_{t+1} = W_t + \alpha \sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)} \right)^{\top}, \tag{3.76}
$$

其中 α 是学习率。

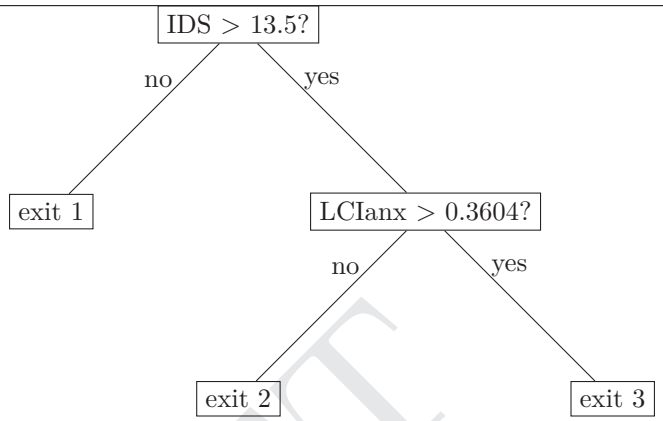


图 3.7: 决策树示例

3.2.4 决策树

决策树（Decision Tree）是一种简单但是广泛使用的分类器。通过训练数据构建决策树，可以高效的对未知的数据进行分类。决策树有两大优点：1）决策树模型可以读性好，具有描述性，有助于人工分析；2）效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

决策树的典型算法有ID3，C4.5，CART等。相对于其它算法，决策树易于理解和实现，人们在通过解释后都有能力去理解决策树所表达的意义。决策树可以同时处理不同类型的属性,并且在相对短的时间内能够对大型数据源做出可行且效果良好的结果。

3.2.5 朴素贝叶斯分类器

贝叶斯分类器根据在给定样本情况下不同类别的后验概率进行决策分类。

$$\hat{y} = \arg\max_{c=1}^{C} p(c|\mathbf{x}), \tag{3.77}$$

其中，$p(c|\mathbf{x})$ 为第 c 类的后验概率。

根据贝叶斯公式，上式可写为：

$$\hat{y} = \arg\max_{c=1}^{C} p(c|\mathbf{x}) \tag{3.78}$$

$$= \arg\max_{c=1}^{C} \frac{p(\mathbf{x}|c)p(c)}{\sum_c p(\mathbf{x}|c)p(c)} \tag{3.79}$$

$$\propto \arg\max_{c=1}^{C} p(\mathbf{x}|c)p(c). \tag{3.80}$$

这里需要估计样本$\mathbf{x}$的多维变量的联合条件概率$p(\mathbf{x}|c)$。当$\mathbf{x}$为高维向量时，这是比较困难的工作。因此在实际应用中，经常假设样本每一维之间是彼此独立的。$p(\mathbf{x}|c)$近似为多维向量$\mathbf{x}$中每一维变量的条件概率$p(\mathbf{x}_i|c)$的乘积。

$$p(\mathbf{x}|c) = \prod_{i=1}^{m} p(\mathbf{x}_i|c). \tag{3.81}$$

其中，m是样本$\mathbf{x}$的维数，$\mathbf{x}_i$是$\mathbf{x}$第i维的值。

这样，公式3.80就可以写为

$$\hat{y} \propto \arg\max_{c=1}^{C} \prod_{i=1}^{m} p(\mathbf{x}_i|c). \tag{3.82}$$

公式3.82就称为**朴素贝叶斯（Naïve Bayes，NB）分类器**。其主要假设就是样本的每一维特征之间是彼此独立的。但这种情况在实际情况中一般不成立，因此其分类准确率可能会下降。但在许多场合，朴素贝叶斯分类算法的假设依然取得很好的性能，并且十分简单，可以与很多复杂的分类算法相媲美，是机器学习中最为常用的算法之一。

3.2.6 k最近邻算法

k最近邻算法(k-Nearest Neighbor，kNN)是根据待分样本的最相似的几个样本的类别来决定其所属类别，是最简单有效的机器学习算法之一。

对于目标函数形式不作过多的假设的算法称为非参数机器学习算法。通过不做假设，算法可以自由的从训练数据中学习任意形式的函数。

当你拥有许多数据而先验知识很少时，非参数学习通常很有用，此时你不需要关注于参数的选取。

对于一个样本，我们可以在特征空间中找到和它最相似(即特征空间中最邻近)的k个样本，如果k个样本中的大多数属于某一个类别，则该样本也属于这个类别。当$k=1$时，也称为**最近邻（NN）算法**。

kNN方法相当于非参数密度估计方法，在决策时只与极少量的相邻样本有关。由于kNN方法主要靠周围有限的邻近的样本，因此对于类域的交叉或重叠较多的非线性可分数据来说，kNN方法较其他方法更为适合，可以很好的克服了线性不可分问题的缺陷。

在kNN算法中，所选择的邻居都是已经有类别标记的样本，这也意味着kNN算法不需要训练阶段。因此，kNN算法也很适用于分类标准随时会产生变化的需求。只要删除旧的标记样本，添加新的标记样本，就改变了分类准则。

但是，kNN算法也存在一些不足之处。

1. 在判断一个样本的类别时，需要把它与所有已知类别的样本都比较一遍，这样计算开销是相当大的。虽然可以通过对样本空间建立索引来提高找到最近邻的效率，但是效率依然低于其他分类器。
2. 当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的k个邻居中大容量类的样本占多数，导致分类错误。
3. kNN算法很容易受到噪声特征的影响，在进行kNN分类之前需要进行特征选择。

3.2.7 支持向量机

支持向量机（Support Vector Machine，SVM）是一个经典的监督学习算法，它在解决很多任务中表现很强的优势。

支持向量机引入**边际距离**（Margin）的概念，给定训练样本$\{(x_i, y_i)\}_{i=1}^n$，我们定义

$$\gamma_i(\mathbf{w}) = \mathbf{w}^\top \mathbf{x} - \arg\max_{\hat{y} \neq y} \boldsymbol{\theta}^T \boldsymbol{\phi}(x, \hat{y}). \tag{3.83}$$

支持向量机的目标是寻找一个权重向量$\boldsymbol{\theta}^*$使得，

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}:||\boldsymbol{\theta}||=1} \min_{i=1}^{n} \gamma_i(\boldsymbol{\theta}). \tag{3.84}$$

公式3.84可以写为：

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \tfrac{1}{2}||\boldsymbol{\theta}||^2 \\ &\text{s.t.} \gamma_i(w) \geq 1, (\forall i) \end{aligned} \tag{3.85}$$

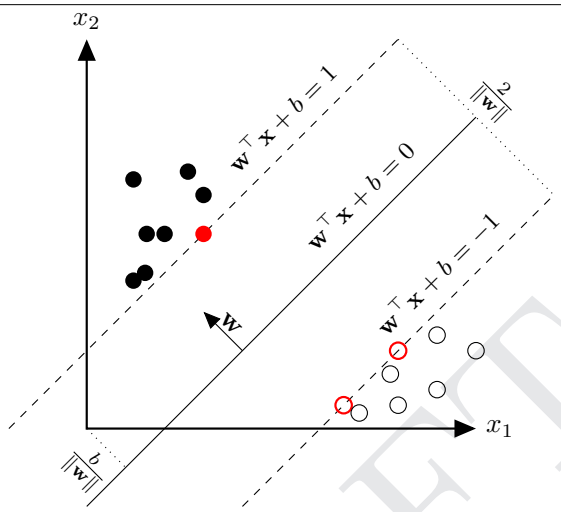


图 3.8: 支持向量机

公式3.85的约束条件比较强，为了能够容忍部分不满足约束的样本，可以引入松弛变量ξ：

$$\min_{\boldsymbol{\theta},\xi} \frac{1}{2}||\boldsymbol{\theta}||^2 + C\sum_{i=1}^{n}\xi_i \tag{3.86}$$
$$\text{s.t.}\gamma_i(w) \geq 1-\xi_i, \xi_i \geq 0, (\forall i)$$

支持向量机的求解可以通过二次优化方法得到全局最优解，这使它有着其他统计学习技术难以比拟的优越性。同时，还使用核函数将原始的样本空间向高维空间进行变换，能够解决原始样本线性不可分的问题。支持向量机的优点在于通用性较好，且分类精度高、分类速度快、分类速度与训练样本个数无关，是最常用的分类器之一。

3.3 评价方法

为了衡量一个分类算法好坏，需要给定一个测试集，用分类器对测试集中的每一个样本进行分类，并根据分类结果计算评价分数。常见的评价标准有正确率、准确率、召回率和F值等。

给定测试集$T=(x_1,y_1),\cdots,(x_N,y_N)$，对于所有的$y_i \in \{\omega_1,\cdots,\omega_C\}$。假

设分类结果为$Y = \hat{y_1}, \cdots, \hat{y_N}$。

则**正确率**（Accuracy，Correct Rate）为：

$$Acc = \frac{\sum_{i=1}^{N} |y_i = \hat{y}_i|}{N} \tag{3.87}$$

其中，$|\cdot|$为指示函数

和正确率相对应的就是**错误率**（Error Rate）。

$$Err = \frac{\sum_{i=1}^{N} |y_i \neq \hat{y}_i|}{N} \tag{3.88}$$

正确率是平均的整体性能。

在很多情况下，我们需要对每个类都进行性能估计，这就需要计算准确率和召回率。正确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值，在机器学习的评价中也被大量使用。

准确率（Precision，P），也叫查准率，精确率或精度，是识别出的个体总数中正确识别的个体总数的比例。对于类c来说，

$$P_c = \frac{\sum_{\substack{i=1 \\ \hat{y_i}=c}}^{N} |y_i = \hat{y}_i|}{\sum_{\substack{i=1 \\ \hat{y_i}=c}}^{N} 1} \tag{3.89}$$

召回率（Recall，R），也叫查全率，是测试集中存在的个体总数中正确识别的个体总数的比例。

$$R_c = \frac{\sum_{\substack{i=1 \\ y_i=c}}^{N} |y_i = \hat{y}_i|}{\sum_{\substack{i=1 \\ y_i=c}}^{N} 1} \tag{3.90}$$

F1 值是根据正确率和召回率二者给出的一个综合的评价指标，具体定义如下：

$$F1_c = \frac{P_c * R_c * 2}{(P_c + R_c)} \tag{3.91}$$

为了计算分类算法在整个数据集上的总体准确率、召回率和F1值，经常使用两种平均方法，分别称为**宏平均**（macro average）和**微平均**（micro average）。

宏平均是每一个类的性能指标的算术平均值，

$$R_{macro} = \sum_{i=1}^{C} R_c/C, \tag{3.92}$$

$$P_{macro} = \sum_{i=1}^{C} P_c/C, \tag{3.93}$$

$$F1_{macro} = \frac{P_{macro} * R_{macro} * 2}{(P_{macro} + R_{macro})}. \tag{3.94}$$

而微平均是每一个样本的性能指标的算术平均。对于单个样本而言，它的准确率和召回率是相同的（要么都是1，要么都是0）因此准确率和召回率的微平均是相同的，根据F1 值公式，对于同一个数据集它的准确率、召回率和F1的微平均指标是相同的。

3.4 定理

在机器学习中，有一些非常有名的定理。这些定理对理解机器学习的内在特性非常有帮助。

3.4.1 没有免费午餐定理

没有免费午餐定理(No Free Lunch Theorem，NFL) 是由Wolpert和Macerday在最优化理论中提出的。没有免费午餐定理证明：对于基于迭代的最优化算法，不存在某种算法对所有问题（有限的搜索空间内）都有效。如果一个算法对某些问题有效，那么它一定在另外一些问题上比纯随机搜索算法更差。也就是说，不能脱离具体问题来谈论算法的优劣，任何算法都有局限性。必须要"具体问题具体分析"。

没有免费午餐定理对于机器学习算法也同样适用。不存在一种机器学习算法适合于任何领域或任务。如果有人宣称自己的模型在所有问题上都好于其他模型，那么他肯定是在吹牛。

3.4.2 丑小鸭定理

渡边慧，模式识别研究的鼻祖之一。
这里的"丑小鸭"是指白天鹅的幼雏，而不是"丑陋的小鸭子"。

丑小鸭定理（Ugly Duckling）1960年代，，美籍日本学者渡边慧提出了丑小鸭定理："丑小鸭与白天鹅之间的区别和两只白天鹅之间的区别一样大"。这个定理初看好像不符合常识，但是仔细思考后是非常有道理的。因为世界上不存在相似性的客观标准，一切相似性的标准都是主观的。如果以体型大小的角度来看，丑小鸭和白天鹅的区别大于两只白天鹅的区别；但是如果以基因的角度来看，丑小鸭与它父母的差别要小于他父母和其他白天鹅之间的差别。

3.5 总结和深入阅读

本章简单地介绍了机器学习的理论知识，主要为后面讲解人工神经网络铺垫一些基础知识。如果需要快速全面地了解机器学习的基本概念可以阅读《Pattern Classification》[?]和《Pattern Recognition and Machine Learning》[Bishop, 2006]，进一步深入了解可以阅读《The Elements of Statistical Learning》[?] 以及《Learning in Graphical Models》[?]。

参考文献

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

James A Anderson and Edward Rosenfeld. *Talking nets: An oral history of neural networks*. MiT Press, 2000.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

C.M. Bishop. *Pattern recognition and machine learning*. Springer New York., 2006.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems*, pages 472–478, 2001.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. In *ICML*, volume 28, pages 1319–1327, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.