第3章

机器学习算法

机器学习是计算机科学与统计学结合的产物，主要研究如何选择统计学习模型，从大量已有数据中学习特定经验。机器学习中的经验称为模型，机器学习的过程即根据一定的性能度量准则对模型参数进行近似求解，使得模型在面对新数据时能够给出相应的经验指导。对于机器学习的准确定义，目前学术界尚未有统一的描述，比较常见的是Mitchell教授于1997年对机器学习的定义："对于某类任务T和性能度量P，一个计算机程序被认为可以从经验E中学习是指：通过经验E改进后，它在任务T上的性能度量P有所提升。"

通过本章的学习，读者可以了解机器学习的常用算法及其实现细节。

3.1 算法概述

3.1.1 线性回归

线性回归是最基本的回归分析方法，有着广泛的应用。线性回归研究的是自变量与因变量之间的线性关系。对于特征 $\boldsymbol{x}=(x^1,x^2,\cdots,x^n)$ 及其对应的标签 y，线性回归假设二者之间存在线性映射：

$$y \approx f(x)=\omega_1 x^1+\omega_2 x^2+\cdots+\omega_n x^n+b=\sum_{i=1}^{n}\omega_i x^i+b=\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b \qquad (3\text{-}1)$$

其中，$\boldsymbol{\omega}=(\omega_1,\omega_2,\cdots,\omega_n)$ 和 b 分别表示待学习的权重及偏置。直观上，权重 $\boldsymbol{\omega}$ 的各个分量反映每个特征变量的重要程度。权重越大，对应的随机变量的重要程度越大，反之则越小。

线性回归的目标是求解 $\boldsymbol{\omega}$ 和 b，使得 $f(\boldsymbol{x})$ 与 y 尽可能接近。求解线性回归模型的基本方法是最小二乘法。最小二乘法是一个不带条件的最优化问题，优化目标是让整个样

本集合上的预测值与真实值之间的欧氏距离之和最小。

1. 一元线性回归

式(3-1)描述的是多元线性回归。为简化讨论,首先以一元线性回归为例进行说明:

$$y \approx f(x) = \omega x + b \tag{3-2}$$

给定空间中的一组样本点 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$,目标函数为

$$\min J(\omega,b) = \min\sum_{i=1}^{m}(y_i - f(x_i))^2 = \min\sum_{i=1}^{m}(y_i - \omega x_i - b)^2 \tag{3-3}$$

令目标函数对 ω 和 b 的偏导数为 0:

$$\begin{cases}\dfrac{\partial J(\omega,b)}{\partial \omega} = \displaystyle\sum_{i=1}^{m}2\omega x_i^2 + \sum_{i=1}^{m}2(b-y_i)x_i \\ \dfrac{\partial J(\omega,b)}{\partial b} = \displaystyle\sum_{i=1}^{m}2(\omega x_i - y_i) + 2mb\end{cases} \tag{3-4}$$

则可得到 ω 和 b 的估计值:

$$\begin{cases}\omega = \dfrac{m\displaystyle\sum_{i=1}^{m}x_i y_i - \sum_{i=1}^{m}x_i\sum_{i=1}^{m}y_i}{m\displaystyle\sum_{i=1}^{m}x_i^2 - \left(\sum_{i=1}^{m}x_i\right)^2} = \dfrac{\overline{xy} - \bar{x}\cdot\bar{y}}{\overline{x^2} - \bar{x}^2} \\ b = \dfrac{1}{m}\left(\displaystyle\sum_{i=1}^{m}y_i - \omega\sum_{i=1}^{m}x_i\right) = \bar{y} - \omega\bar{x}\end{cases} \tag{3-5}$$

其中,短横线"-"表示求均值运算。

2. 多元线性回归

对于多元线性回归,本书仅做简单介绍。为了简化说明,可以将 b 同样看作权重,即令

$$\begin{cases}\boldsymbol{\omega} = (\omega_1,\omega_2,\cdots,\omega_n,b) \\ \boldsymbol{x} = (x^1,x^2,\cdots,x^n,l)\end{cases} \tag{3-6}$$

此时式(3-1)可表示为

$$y \approx f(x) = \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} \tag{3-7}$$

给定空间中的一组样本点 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$,优化目标为

$$\min J(\boldsymbol{\omega}) = \min(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\omega})^{\mathrm{T}}(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\omega}) \tag{3-8}$$

其中,$\boldsymbol{X}$ 为样本矩阵的增广矩阵:

$$\boldsymbol{X} = \begin{bmatrix}x_1^1 & x_1^2 & \cdots & x_1^n & 1 \\ x_2^1 & x_2^2 & \cdots & x_2^n & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_m^1 & x_m^2 & \cdots & x_m^n & 1\end{bmatrix} \tag{3-9}$$

$\boldsymbol{Y}$ 为对应的标签向量:

$$\boldsymbol{Y} = (y_1,y_2,\cdots,y_n)^{\mathrm{T}} \tag{3-10}$$

求解式(3-8)可得

$$\boldsymbol{\omega}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} \tag{3-11}$$

当 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 可逆时，线性回归模型存在唯一解。当样本集合中的样本太少或者存在大量线性相关的维度，则可能会出现多个解的情况。奥卡姆剃刀原则指出，当模型存在多个解时，选择最简单的那个。因此可以在原始线性回归模型的基础上增加正则化项以降低模型的复杂度，使得模型变得简单。若加入 L2 正则化，则优化目标可写作

$$\min J(\boldsymbol{\omega})=\min(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\omega})^{\mathrm{T}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\omega})+\lambda\|\boldsymbol{\omega}\|_2 \tag{3-12}$$

此时，线性回归又称为岭(Ridge)回归。求解式(3-12)有

$$\boldsymbol{\omega}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} \tag{3-13}$$

$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}$ 在 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的基础上增加了一个扰动项 $\lambda\boldsymbol{I}$。此时不仅能够降低模型的复杂度，防止过拟合，而且能够使 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}+\lambda\boldsymbol{I}$ 可逆，$\boldsymbol{\omega}$ 有唯一解。

当正则化项为 L1 正则化时，线性回归模型又称为 Lasso(Least Absolute Shrinkage and Selection Operator)回归，此时优化目标可写作

$$\min J(\boldsymbol{\omega})=\min(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\omega})^{\mathrm{T}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\omega})+\lambda|\boldsymbol{\omega}| \tag{3-14}$$

L1 正则化能够得到比 L2 正则化更为稀疏的解。所谓稀疏是指 $\boldsymbol{\omega}=(\omega_1,\omega_2,\cdots,\omega_n)$ 中会存在多个值为 0 的元素，从而起到特征选择的作用。由于 L1 范数使用绝对值表示，所以目标函数 $J(\boldsymbol{\omega})$ 不是连续可导，此时不能再使用最小二乘法进行求解。

3.1.2　逻辑回归

逻辑回归是一种广义线性回归，通过回归对数概率(Logits)的方式将线性回归应用于分类任务。对于一个二分类问题，令 $Y\in\{0,1\}$ 表示样本 x 对应的类别变量。设 x 属于类别 1 的概率为 $P(Y=1|x)=p$，则自然有 $P(Y=0|x)=1-p$。比值 $\dfrac{p}{1-p}$ 称为概率(Odds)，概率的对数即为对数概率：

$$\ln\frac{p}{1-p} \tag{3-15}$$

逻辑回归通过回归式(3-15)间接得到 p 的值，即

$$\ln\frac{p}{1-p}=\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b \tag{3-16}$$

解得

$$p=\frac{1}{1+\mathrm{e}^{-(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b)}} \tag{3-17}$$

为方便描述，令

$$\begin{cases}\boldsymbol{\omega}=(\omega_1,\omega_2,\cdots,\omega_n,b)^{\mathrm{T}}\\ \boldsymbol{x}=(x^1,x^2,\cdots,x^n,1)^{\mathrm{T}}\end{cases} \tag{3-18}$$

则有

$$p=\frac{1}{1+\mathrm{e}^{-\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}}} \tag{3-19}$$

由于样本集合给定的样本属于类别 1 的概率非 0 即 1,所以式(3-19)无法用最小二乘法求解。此时可以考虑使用极大似然估计进行求解。

给定样本集合 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_m,y_m)\}$,似然函数为

$$L(\boldsymbol{\omega})=\prod_{i=1}^{m}p^{y_i}(1-p)^{(1-y_i)} \tag{3-20}$$

对数似然函数为

$$\begin{aligned} l(\boldsymbol{\omega}) &= \sum_{i=1}^{m}(y_i\ln p+(1-y_i)\ln(1-p)) \\ &= \sum_{i=1}^{m}(y_i\,\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_i-\ln(1+\mathrm{e}^{\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_i})) \end{aligned} \tag{3-21}$$

之后可用经典的启发式最优化算法梯度下降法求解式(3-21)。

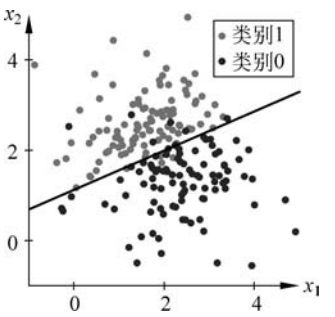

图 3-1 逻辑回归示例(见彩插)

图 3-1 是二维空间中使用逻辑回归进行二分类的示例。图中样本存在一定的噪声(正类中混合有部分负类样本、负类中混合有部分正类样本)。可以看到逻辑回归能够抵御一定的噪声干扰。

3.1.3 线性判别分析

线性判别分析(Linear Discriminant Analysis,LDA)是对 Fisher 线性判别方法的归纳,这种方法使用统计学、模式识别和机器学习方法,试图找到两类物体或事件的特征的一个线性组合,以能够特征化或区分它们。

1. 二类线性判别分析原理

首先从比较简单的二类 LDA 入手,严谨地分析 LDA 的原理。假设数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$,其中任意样本 $\boldsymbol{x}_i$ 为 n 维向量,$y_i\in\{0,1\}$,定义 $N_j(j=0,1)$为第 j 类样本的个数,$X_j(j=0,1)$为第 j 类样本的集合,$\boldsymbol{\mu}_j(j=0,1)$为第 j 类样本的均值向量,$\boldsymbol{\Sigma}_j(j=0,1)$为第 j 类样本的协方差矩阵。$\boldsymbol{\mu}_j$的表达式为

$$\boldsymbol{\mu}_j=\frac{1}{N_j}\Sigma_{x\in X_j}x,\quad j=0,1 \tag{3-22}$$

$\boldsymbol{\Sigma}_j$的表达式为

$$\boldsymbol{\Sigma}_j=\Sigma_{x\in X_j}(x-\boldsymbol{\mu}_j)(x-\boldsymbol{\mu}_j)^{\mathrm{T}},\quad j=0,1 \tag{3-23}$$

由于有两种类型的数据,所以只需要将数据投影到一条直线上即可。假设投影直线是向量 $\boldsymbol{w}$,则对任意一个样本 x_i,它在直线 $\boldsymbol{w}$ 的投影为 $\boldsymbol{w}^{\mathrm{T}}x_i$,对于两个类别的中心点 μ_0 和 μ_1,在直线 $\boldsymbol{w}$ 的投影为 $\boldsymbol{w}^{\mathrm{T}}\mu_0$ 和 $\boldsymbol{w}^{\mathrm{T}}\mu_1$。因为 LDA 需要使不同类别数据的类别中心之间的距离尽可能大,而我们希望同一类别数据的投影点尽可能接近,即投影点的协方差 $\boldsymbol{w}^{\mathrm{T}}\Sigma_0\boldsymbol{w}$ 和 $\boldsymbol{w}^{\mathrm{T}}\Sigma_1\boldsymbol{w}$ 尽可能小,即最小化 $\boldsymbol{w}^{\mathrm{T}}\Sigma_0\boldsymbol{w}+\boldsymbol{w}^{\mathrm{T}}\Sigma_1\boldsymbol{w}$。定义类内散度矩阵 $\boldsymbol{S}_w$ 和类间散度矩阵 $\boldsymbol{S}_b$。这样优化重写的目标为

$$\underbrace{\operatorname{argmax}}_{w} J(\boldsymbol{w}) = \frac{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{S}_b \boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{S}_w \boldsymbol{w}} \tag{3-24}$$

由式(3-24)可知，只要求出原始二类样本的均值和方差就可以确定最佳的投影方向 w。

2. 多类 LDA 原理

以二类 LDA 的原理为基础，下面介绍多类别 LDA 的原理。假设数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$，其中，任意样本 $\boldsymbol{x}_i$ 为 n 维向量，$y_i \in \{C_1,C_2,\cdots,C_k\}$，定义 $N_j(j=1,2,\cdots,k)$为第 j 类样本的个数，$X_j(j=1,2,\cdots,k)$为第 j 类样本的集合，$\boldsymbol{\mu}_j(j=1,2,\cdots,k)$为第 j 类样本的均值向量，$\boldsymbol{\Sigma}_j(j=1,2,\cdots,k)$为第 j 类样本的协方差矩阵。在二类 LDA 中定义的公式可以很容易地类推到多类 LDA。由于是多类向低维投影，所以此时投影到的低维空间就不是一条直线，而是一个超平面。假设投影到的低维空间的维度为 d，对应的基向量为$(\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_d)$，基向量组成的矩阵为 $\boldsymbol{W}$，是一个 $n\times d$ 的矩阵。优化目标变为

$$\frac{\boldsymbol{W}^{\mathrm{T}} \boldsymbol{S}_b \boldsymbol{W}}{\boldsymbol{W}^{\mathrm{T}} \boldsymbol{S}_w \boldsymbol{W}} \tag{3-25}$$

3.1.4　分类与回归树分析

分类与回归树(Classification And Regression Tree，CART)是在给定输入随机变量 $\boldsymbol{X}$ 的条件下输出随机变量 $\boldsymbol{Y}$ 的条件概率分布的学习方法。CART 假设决策树是二叉树，内部节点特征的取值为“是”和“否”，左分支是取值为“是”的分支，右分支是取值为“否”的分支。这样的决策树等价于递归地二分每个特征，将输入空间即特征空间划分为有限个单元，并在这些单元上确定预测的概率分布，也就是在输入给定的条件下输出的条件概率分布。

1. 回归树生成

假设 $\boldsymbol{X}$ 与 $\boldsymbol{Y}$ 分别是输入和输出向量，并且 $\boldsymbol{Y}$ 是连续变量。给定训练数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，考虑如何生成回归树。

一个回归树对应输入空间(即特征空间)的一个划分以及在划分的单元上的输出值。假设已将输入空间划分为 M 个单元 $R_1,R_2,\cdots,R_m$，并且在每个单元 R_m 上有一个固定的输出值 c_m，则回归树模型可表示为

$$f(x)=\sum_{m=1}^{M} c_m I, \quad x \in R_m \tag{3-26}$$

当输入空间的划分确定时，可以用平方误差表示回归树对于训练数据的预测误差，用平方误差最小的准则求解每个单元上的最优输出值。易知，单元 R_m 上的 c_m 的最优值是 R_m 上所有输入实例 x_i 对应的输出 y_i 的均值，即 $\mathrm{ave}(y_i \mid x_i \in R_m)$。之后采用启发式的方法对输入空间进行划分。选择第 j 个变量 x_j 和它的取值 s 作为切分变量和切分点定义两个区域，以此寻找最优切分变量 k 和最优切分点 s。遍历所有输入变量，找到最优切

分变量 j 和最优切分点 s，构成一个对(j,s) 依次将输入空间划分为两个区域。接着，对每个区域重复上述划分过程，直到满足停止条件为止。这样就生成一棵回归树，这样的回归树通常被称为最小二乘树。

2. 分类树生成

在分类过程中，假设有 k 个类，样本点属于第 k 个类的概率为 p_k，则概率分布的基尼指数定义为

$$\mathrm{Gini}(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2 \tag{3-27}$$

对于二类分类问题，若样本点属于第 1 个类的概率是 p，则概率分布的基尼指数为

$$\mathrm{Gini}(p)=2p(1-p) \tag{3-28}$$

对于给定的样本集合 D，其基尼指数为

$$\mathrm{Gini}(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2 \tag{3-29}$$

其中，C_k 是 D 中属于第 k 类的样本子集，K 是类的个数。

如果样本集合 D 根据特征 A 是否取某个可能值 a 被分割成 D_1 和 D_2 两部分，即

$$D_1=\{(x,y)\in D\mid A(x)=a\},\quad D_2=D-D_1 \tag{3-30}$$

则在特征 A 的条件下，集合 D 的基尼指数定义为

$$\mathrm{Gini}(D,A)=\frac{|D_1|}{|D|}\mathrm{Gini}(D_1)+\frac{|D_2|}{|D|}\mathrm{Gini}(D_2) \tag{3-31}$$

基尼指数 $\mathrm{Gini}(D)$表示集合 D 的不确定性，基尼指数 $\mathrm{Gini}(D,A)$表示经 $A=a$ 分割后集合 D 的不确定性。基尼指数越大，样本集合的不确定性越大，这一点与熵相似。

3.1.5 朴素贝叶斯

朴素贝叶斯分类器(Naive Bayes Classifier，NBC)发源于古典数学理论，有着坚实的数学基础以及稳定的分类效率。同时，朴素贝叶斯分类器模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。

设有样本数据集 $D=\{d_1,d_2,\cdots,d_n\}$，对应样本数据的特征属性集为 $X=\{x_1,x_2,\cdots,x_d\}$，类变量为 $Y=\{y_1,y_2,\cdots,y_m\}$，即 D 可以分为 y_m 类别。其中 $x_1,x_2,\cdots,x_d$ 相互独立且随机，则 Y 的先验概率 $P_{\mathrm{prior}}=P(Y)$，Y 的后验概率 $P_{\mathrm{post}}=P(Y|X)$，由朴素贝叶斯算法可得，后验概率可以由先验概率 $P_{\mathrm{prior}}=P(Y)$、证据 $P(X)$和类条件概率 $P(X|Y)$计算得

$$P(Y\mid X)=\frac{P(Y)P(X\mid Y)}{P(X)} \tag{3-32}$$

朴素贝叶斯基于各特征之间相互独立，在给定类别为 y 的情况下，式(3-32)可以进一步表示为式(3-33)：

$$P(X\mid Y=y)=\prod_{i=1}^{d}P(x_i\mid Y=y) \tag{3-33}$$

由式(3-32)、式(3-33)可以计算出后验概率为

$$P_{\text{post}}=P(Y\mid X)=\frac{P(Y)\prod_{i=1}^{d}P(x_i\mid Y)}{P(X)} \tag{3-34}$$

由于 $P(X)$的大小是固定不变的,因此在比较后验概率时,只比较式(3-34)的分子部分即可。可以得到一个样本数据属于类别 y_i 的朴素贝叶斯计算:

$$P(y_i\mid x_1,x_2,\cdots,x_d)=\frac{P(y_i)\prod_{j=1}^{d}P(x_j\mid y_i)}{\prod_{j=1}^{d}P(x_j)} \tag{3-35}$$

朴素贝叶斯算法假设数据集的属性是相互独立的,因此算法逻辑非常简单,算法相对稳定。当数据呈现不同的特征时,朴素贝叶斯的分类性能不会有太大区别。换句话说,朴素贝叶斯算法的鲁棒性更好,对于不同类型的数据集不会表现出太大的差异。当数据集的属性之间的关系相对独立时,朴素贝叶斯分类算法会有更好的结果。属性独立的条件也是朴素贝叶斯分类器的缺点。数据集属性的独立性在很多情况下很难满足,因为数据集的属性往往是相互关联的。如果在分类过程中出现这类问题,则分类的效果就会大打折扣。

3.1.6　*k* 最近邻算法

k 最近邻(*k*-Nearest Neighbor,KNN)算法是最简单的机器学习算法之一。该方法的思路是在特征空间中,如果一个样本附近的 *k* 个最近(即特征空间中最邻近)样本的大多数属于某一个类别,则该样本也属于这个类别。

3.1.7　学习矢量量化

学习矢量量化(Learning Vector Quantization,LVQ)是 1988 年由 Kohonen 提出的一类用于模式分类的有监督学习算法,是一种结构简单、功能强大的有监督式神经网络分类方法。根据训练样本是否有监督,学习矢量量化算法可分为两种:一种是有监督学习矢量量化,如 LVQ1、LVQ2.1 和 LVQ3,它是对有类别属性的样本进行聚类;另一种是无监督学习矢量量化,如序贯硬 *C* 均值,它是对无类别属性的样本进行聚类。

传统的学习矢量量化算法虽然性能优越且应用广泛,但仍存在一些不足。首先,权重向量在训练过程中可能不会收敛。原因是在寻找最优贝叶斯边界时,没有充分考虑权向量更新的趋势。同时,学习矢量量化算法并没有充分利用输入样本各个维度属性的信息,也没有体现出各个维度属性在分类过程中重要性的差异。原因是在寻找获胜神经元的过程中使用的欧氏距离测量方法没有考虑输入样本每个维度属性的重要性差异,即假设每个维度属性的对分类的"贡献"是一样的。

3.1.8　支持向量机

支持向量机(Support Vector Machine,SVM)是一类按监督学习方式对数据进行二

元分类的广义线性分类器,其决策边界是对学习样本求解的最大边距超平面。支持向量机使用铰链损失函数计算经验风险并在求解系统中加入正则化项以优化结构风险,是一个具有稀疏性和稳健性的分类器。

支持向量机于1964年被提出,在20世纪90年代后得到快速发展并衍生出一系列改进和扩展算法,在人像识别、文本分类等模式识别问题中应用广泛。

3.1.9 Bagging和随机森林

随机森林和梯度提升树(GBDT)是集成学习中应用得最多的算法,尤其是随机森林可以很方便地进行并行训练,在如今大数据大样本的时代很有诱惑力。

Bagging的特点是随机抽样。随机抽样就是从训练集中采集固定数量的样本,但是每采集完一个样本,就返回样本。也就是说,之前采集的样本放回后可以继续采集。Bagging算法一般会随机采集与训练集样本数 m 相同数量的样本。这样得到的样本集与训练集的样本数相同,但样本内容不同。如果我们用 m 个样本随机抽取训练集 T 次,则由于随机性 T 个样本集是不同的。这与GBDT子采样不同GBDT的子抽样是无放回抽样,而Bagging的子抽样是有放回抽样。由于Bagging算法每次都采样训练模型,因此它的泛化能力很强,对于降低模型的方差非常有用。当然,训练集的拟合会更差,也就是模型的偏差会更大。

随机森林(Random Forest)算法是Bagging算法的进化版本。随机森林算法使用CART决策树作为弱学习器。随机森林在使用决策树的基础上改进了决策树的建立。对于普通的决策树,我们会使用节点上的所有 n 个样本特征,选择一个最优特征划分决策树的左右子树。但是随机森林随机选择节点上的一部分样本特征,这个数小于 n,然后从这些随机选择的样本特征中选择一个最优特征做决策树的左右子树划分,进一步增强了模型的泛化能力。

3.1.10 Boosting和AdaBoost

Boosting是一种组合弱分离器形成强分类器的算法框架。它把很多分类准确率很低的分类器通过更新对数据的权重,集成起来形成一个分类效果好的分类器,所以也是集成方法(Ensemble Method)的一种。一般而言,Boosting算法有以下三个要素。

(1) 函数模型:Boosting的函数模型是叠加型的。

(2) 目标函数:选定某种损失函数作为优化目标。

(3) 优化算法:逐步优化。

AdaBoost是Boosting算法框架中的一种实现。它在Boosting的基础上,每一个样本给定相同的初始权重,分类错误的样本权重上升,分类正确的样本权重下降,即它是在前一个分类器训练的基础上训练得到新的分类器,分类器的权重由其分类的准确率决定,组合弱分类器形成强分类器,不容易过拟合。AdaBoost通过更改训练样本的权重,让每个弱分类器之间能互补从而使不能很好分类的数据得到重视。

3.2　支持向量机算法

3.2.1　线性支持向量机

在支持向量机的最小优化目标函数中加入松弛因子可以实现软间隔，松弛系数 C 越小，间隔越宽，分割超平面越硬。相反松弛系数 C 越大，间隔越窄，分割超平面越软，会更多地拟合训练样本。需要注意当 C 太大时，容易过拟合。

给定输入数据和学习目标 $\boldsymbol{X}_i=(X_1,X_2,\cdots,X_N)$，$y_i=(y_1,y_2,\cdots,y_N)$，硬边界支持向量机是一种求解线性可分问题中最大边距超平面的算法。约束条件是采样点到决策边界的距离大于或等于 1，可以转化为等价的二次凸优化问题求解，即

$$\begin{aligned}&\max_{w,b}\quad \frac{2}{\|\boldsymbol{w}\|} &&\Leftrightarrow &&\min_{w,b}\quad \frac{1}{2}\|\boldsymbol{w}\|^2\\ &\text{s.t.}\quad y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)\geqslant 1 &&&&\text{s.t.}\quad y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)\geqslant 1\end{aligned}\tag{3-36}$$

由式(3-36)得到的决策边界可以对任意样本进行分类：

$$\operatorname{sign}[y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)]\tag{3-37}$$

在线性不可分问题中使用硬边界支持向量机会产生分类错误。因此，可以在最大化边际的基础上引入损失函数，构造新的优化问题。支持向量机使用铰链损失函数，遵循硬边界支持向量机的优化问题形式。软间隔支持向量机的优化问题表示如下：

$$\begin{aligned}&\min_{w,b}\quad \frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}L_i,\quad L_i=\max[0,1-y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)]\\ &\text{s.t.}\quad y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)\geqslant 1-L_i,\quad L_i\geqslant 0\end{aligned}\tag{3-38}$$

由式(3-38)可知，软边距 SVM 是一个 L_2 正则化分类器，其中，L_i 表示铰链损失函数。使用松弛变量 $\zeta\geqslant 0$ 处理铰链损失函数的分段取值后，式(3-38)可化为

$$\begin{aligned}&\min_{w,b}\quad \frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\xi_i\\ &\text{s.t.}\quad y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)\geqslant 1-\xi_i,\quad \xi_i\geqslant 0\end{aligned}\tag{3-39}$$

求解上述软边距支持向量机通常利用其优化问题的对偶性(Duality)。定义软边距支持向量机的优化问题为原问题，可得到其拉格朗日函数：

$$\mathcal{L}(\boldsymbol{w},b,\xi,\alpha,\mu)=\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\xi_i+\sum_{i=1}^{N}\alpha_i[1-\xi_i-y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)]-\sum_{i=1}^{N}\mu_i\xi_i\tag{3-40}$$

令拉格朗日函数对优化目标 ω,b,ξ 的偏导数为 0，可得到一系列包含拉格朗日乘子的表达式：

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}=0\rightarrow\boldsymbol{w}=\sum_{i=1}^{N}\alpha_i y_i\boldsymbol{X}_i,\quad \frac{\partial\mathcal{L}}{\partial b}=0\rightarrow\sum_{i=1}^{N}\alpha_i y_i=0,\quad \frac{\partial\mathcal{L}}{\partial\xi}=0\rightarrow C=\alpha_i+\mu_i\tag{3-41}$$

将式(3-41)代入拉格朗日函数后可得原问题的对偶问题：

$$\begin{aligned}\max_{\alpha} \quad & \sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}[\alpha_i y_i(\boldsymbol{X}_i)^{\mathrm{T}}(\boldsymbol{X}_j)y_j\alpha_j]\\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i=0,\quad 0\leqslant\alpha_i\leqslant C\end{aligned} \tag{3-42}$$

对偶问题的约束条件中包含不等关系,因此其存在局部最优的条件是拉格朗日乘子满足 KTT 条件(Karush-Kuhn-Tucker Condition):

$$\begin{cases}\alpha_i\geqslant 0,\quad \mu_i\geqslant 0\\ \xi_i\geqslant 0,\quad \mu_i\xi_i=0\\ y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)-1+L_i\geqslant 0\\ \alpha_i[y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i+b)-1+L_i]=0\end{cases} \tag{3-43}$$

由式(3-43)的条件可知,决策边界的确定仅与支持向量有关,软边距支持向量机利用铰链损失函数使得支持向量机具有稀疏性。

3.2.2 非线性支持向量机

使用非线性函数将输入数据映射至高维空间后应用线性支持向量机可得到非线性支持向量机。非线性支持向量机可以类比线性支持向量机的算法,有如下优化问题:

$$\begin{aligned}\min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\xi_i\\ \text{s.t.} \quad & y_i[\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{X}_i)+b]\geqslant 1-\xi_i,\quad \xi_i\geqslant 0\end{aligned} \tag{3-44}$$

类比软边距支持向量机,非线性支持向量机有如下对偶问题:

$$\begin{aligned}\max_{\alpha} \quad & \sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}[\alpha_i y_i\phi(\boldsymbol{X}_i)^{\mathrm{T}}\phi(\boldsymbol{X}_j)y_j\alpha_j]\\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i=0,\quad 0\leqslant\alpha_i\leqslant C\end{aligned} \tag{3-45}$$

注意到式(3-45)中存在映射函数内积,因此可以使用核方法,即直接选取核函数。

3.2.3 支持向量机算法求解

支持向量机算法的求解可以使用二次凸优化问题的数值方法,例如内点法和序列最小优化算法。

(1) 内点法(Interior Point Method,IPM):以软边距支持向量机为例,内点法使用对数阻挡函数将支持向量机的对偶问题由极大值问题转化为极小值问题,并将其优化目标和约束条件近似表示。

(2) 序列最小优化(Sequential Minimal Optimization,SMO)是一种坐标下降法,以迭代方式求解支持向量机的对偶问题,其设计是在每个迭代步选择拉格朗日乘子中的两个变量并固定其他参数,将原优化问题化简至一维子可行域。

(3) 随机梯度下降(Stochastic Gradient Descent,SGD)是机器学习问题中常见的优化算法,适用于样本充足的学习问题。随机梯度下降每次迭代都随机选择学习样本更新模型参数,以减少一次性处理所有样本带来的内存开销。

3.3　逻辑回归算法

3.3.1　线性回归算法

线性回归模型指 $f(\cdot)$采用线性组合形式的回归模型，在线性回归问题中，因变量和自变量之间是线性关系。对于第 i 个因变量 x_i，乘以权重系数 w_i，取 y 为因变量的线性组合：

$$y = f(\boldsymbol{x}) = w_1 x_1 + \cdots + w_n x_n + b \tag{3-46}$$

其中，b 为常数项。若令 $\boldsymbol{w} = (w_1, w_2, \cdots, w_n)$，则式(3-46)可以写成向量形式：

$$y = f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{3-47}$$

可以看到 $\boldsymbol{w}$ 和 b 决定了回归模型 $f(\cdot)$的行为。由数据样本得到 $\boldsymbol{w}$ 和 b 有许多方法，例如最小二乘法和梯度下降法。在这里我们介绍最小二乘法求解线性回归中参数估计的问题。

直觉上，希望找到这样的 $\boldsymbol{w}$ 和 b，使得对于训练数据中每个样本点$(\boldsymbol{x}^{(n)}, y^{(n)})$，预测值 $f(\boldsymbol{x}^{(n)})$与真实值 $y^{(n)}$尽可能接近。于是我们需要定义一种“接近”程度的度量，即误差函数。在这里我们采用平均平方误差(Mean Square Error)作为误差函数：

$$E = \Sigma_n [y^{(n)} - (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(n)} + b)]^2 \tag{3-48}$$

为什么要选择这样一个误差函数呢？这是因为我们做出了这样的假设：给定 $\boldsymbol{x}$，则 y 的分布服从如下高斯分布：

$$p(y \mid \boldsymbol{x}) \sim N(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b, \sigma^2) \tag{3-49}$$

具体分布如图 3-2 所示。直观上，这意味着在自变量 $\boldsymbol{x}$ 取某个确定值时，数据样本点以回归模型预测的因变量 y 为中心，以 σ^2 为方差呈高斯分布。

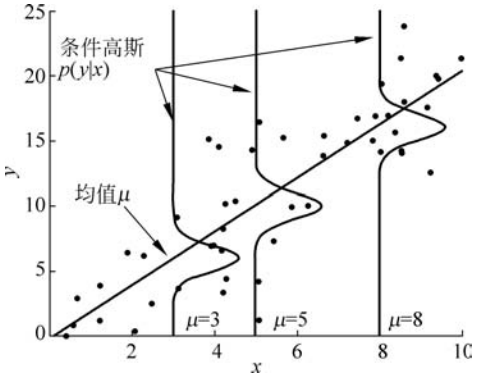

图 3-2　条件概率服从高斯分布

基于高斯分布的假设，我们得到条件概率 $p(y \mid \boldsymbol{x})$的对数似然函数：

$$\boldsymbol{L}(\boldsymbol{w}, b) = \log\left(\prod\nolimits_n \exp\left(-\frac{1}{2\sigma^2}(y^{(n)} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(n)} - b)^2\right)\right) \tag{3-50}$$

即

$$\boldsymbol{L}(\boldsymbol{w}, b) = -\frac{1}{2\sigma^2}\Sigma_n (y^{(n)} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(n)} - b)^2 \tag{3-51}$$

做极大似然估计：

$$\boldsymbol{w},b=\underset{\boldsymbol{w},b}{\operatorname{argmax}}\boldsymbol{L}(\boldsymbol{w},b) \tag{3-52}$$

由于对数似然函数中 σ 为常数，因此极大似然估计可以转化为

$$\boldsymbol{w},b=\underset{\boldsymbol{w},b}{\operatorname{argmin}}\Sigma_n(y^{(n)}-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(n)}-b)^2 \tag{3-53}$$

这就是选择平方平均误差函数作为误差函数的概率解释。

我们的目标就是要最小化这样一个误差函数 E，具体做法可以令 E 对于参数 $\boldsymbol{w}$ 和 b 的偏导数为 0。由于问题变成了最小化平均平方误差，因此这种通过解析方法直接求解参数的做法习惯上被称为最小二乘法。

为了方便矩阵运算，我们将 E 表示成向量形式。令

$$\boldsymbol{Y}=\begin{bmatrix}y^{(1)}\\y^{(2)}\\\vdots\\y^{(n)}\end{bmatrix} \tag{3-54}$$

$$\boldsymbol{X}=\begin{bmatrix}x^{(1)}\\x^{(2)}\\\vdots\\x^{(n)}\end{bmatrix}=\begin{bmatrix}x_1^{(1)}&\cdots&x_m^{(1)}\\x_1^{(2)}&\cdots&x_m^{(2)}\\\vdots&&\vdots\\x_1^{(n)}&\cdots&x_m^{(n)}\end{bmatrix} \tag{3-55}$$

$$\boldsymbol{b}=\begin{bmatrix}b_1\\b_2\\\vdots\\b_n\end{bmatrix},\quad b_1=b_2=\cdots=b_n \tag{3-56}$$

则 E 可表示为

$$E=(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}-\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}-\boldsymbol{b}) \tag{3-57}$$

由于 $\boldsymbol{b}$ 的表示较为烦琐，我们不妨更改一下 $\boldsymbol{w}$ 的表示，将 b 视为常数 1 的权重，令

$$\boldsymbol{w}=(w_1,w_2,\cdots,w_n,b) \tag{3-58}$$

相应地，对 $\boldsymbol{X}$ 做如下更改：

$$\boldsymbol{X}=\begin{bmatrix}\boldsymbol{x}^{(1)};1\\\boldsymbol{x}^{(2)};1\\\vdots\\\boldsymbol{x}^{(n)};1\end{bmatrix}=\begin{bmatrix}x_1^{(1)}&\cdots&x_m^{(1)}&1\\x_1^{(2)}&\cdots&x_m^{(2)}&1\\\vdots&&\vdots&\\x_1^{(n)}&\cdots&x_m^{(n)}&1\end{bmatrix} \tag{3-59}$$

则 E 可表示为

$$E=(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}})^{\mathrm{T}}(\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}) \tag{3-60}$$

对误差函数 E 求参数 $\boldsymbol{w}$ 的偏导数得

$$\frac{\partial E}{\partial \boldsymbol{w}}=2\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}-\boldsymbol{Y}) \tag{3-61}$$

令偏导为 0，得

$$\boldsymbol{w}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} \tag{3-62}$$

因此，对于测试向量 $\boldsymbol{x}$，根据线性回归模型预测的结果为

$$y=\boldsymbol{x}((\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y})^{\mathrm{T}} \tag{3-63}$$

3.3.2　逻辑回归

在 3.3.1 节中，我们假设随机变量 $x_1,x_2,\cdots,x_n$ 与 y 之间的关系是线性的。但在实际中，我们通常会遇到非线性关系。此时，可以利用非线性变换 $g(\cdot)$ 使得线性回归模型 $f(\cdot)$ 实际上对 $g(y)$ 而非 y 进行拟合，即

$$y=g^{-1}(f(\boldsymbol{x})) \tag{3-64}$$

其中，$f(\cdot)$ 仍为

$$f(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b \tag{3-65}$$

因此，这样的回归模型称为广义线性回归模型。

广义线性回归模型使用非常广泛。例如在二元分类任务中，我们的目标是拟合一个分离超平面 $f(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$，使得目标分类 y 可表示为以下阶跃函数：

$$y=\begin{cases}0, & f(\boldsymbol{x})<0\\ 1, & f(\boldsymbol{x})>0\end{cases} \tag{3-66}$$

在分类问题中，由于 y 取离散值，因此这个阶跃判别函数是不可导的。不可导的性质使得许多数学方法不能使用。我们考虑使用一个函数 $\sigma(\cdot)$ 近似这个离散的阶跃函数，通常可以使用 Logistic 函数或 tanh 函数。Logistic 函数如图 3-3 所示。这里就 Logistic 函数情况进行讨论。令

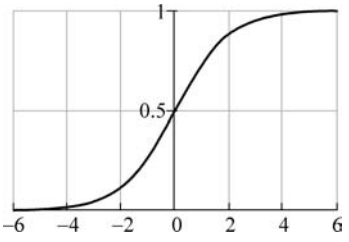

图 3-3　Logistic 函数

$$\sigma(\boldsymbol{x})=\frac{1}{1+\exp(-x)} \tag{3-67}$$

使用 Logistic 函数代替阶跃函数：

$$\sigma(f(\boldsymbol{x}))=\frac{1}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-b)} \tag{3-68}$$

并定义条件概率：

$$\begin{cases}p(y=1\mid\boldsymbol{x})=\sigma(f(\boldsymbol{x}))\\ p(y=0\mid\boldsymbol{x})=1-\sigma(f(\boldsymbol{x}))\end{cases} \tag{3-69}$$

这样就可以把离散取值的分类问题近似地表示为连续取值的回归问题，这样的回归模型称为逻辑回归模型。

在 Logistic 函数中 $g^{-1}(x)=\sigma(x)$，若将 $g(\cdot)$ 还原为 $g(y)=\log\frac{y}{1-y}$ 的形式并移到等式一侧，则可以得到

$$\log\frac{p(y=1\mid\boldsymbol{x})}{p(y=0\mid\boldsymbol{x})}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b \tag{3-70}$$

为了求得逻辑回归模型中的参数 $\boldsymbol{w}$ 和 b，下面对条件概率 $p(y\mid\boldsymbol{x};\boldsymbol{w},b)$ 做极大似然估计。

$p(y\mid\boldsymbol{x};\boldsymbol{w},b)$ 的对数似然函数为

$$L(\boldsymbol{w},b)=\log\left(\prod_n\left[\sigma(f(\boldsymbol{x}^{(n)}))\right]^{y^{(n)}}\left[1-\sigma(f(\boldsymbol{x}^{(n)}))\right]^{1-y^{(n)}}\right) \tag{3-71}$$

即

$$\boldsymbol{L}(\boldsymbol{w},b)=\Sigma_n\left[y^{(n)}\log(\sigma(f(\boldsymbol{x}^{(n)})))+(1-y^{(n)})\log(1-\sigma(f(\boldsymbol{x}^{(n)})))\right] \tag{3-72}$$

这就是常用的交叉熵误差函数的二元形式。

直接求解似然函数$\boldsymbol{L}(\boldsymbol{w},b)$的最大化问题比较困难,我们可以采用数值方法。常用的方法有牛顿迭代法、梯度下降法等。

3.3.3 用 PyTorch 实现逻辑回归算法

后文给出的代码依赖例 3-1。

【例 3-1】 代码依赖模块。

```
1  import torch
2  from torch import nn
3  from matplotlib import pyplot as plt
4  %matplotlib inline
```

1. 数据准备

逻辑回归常用于解决二分类问题,为了便于描述,我们分别从两个多元高斯分布$\mathcal{N}_1(\mu_1,\Sigma_1)$和$\mathcal{N}_2(\mu_2,\Sigma_2)$中生成数据$X_1$和$X_2$,这两个多元高斯分布表示两个类别,分别设置其标签为y_1和y_2。

如例 3-2 所示,PyTorch 的 torch.distributions 提供了 MultivariateNormal 构建多元高斯分布。第 5～8 行设置两组不同的均值向量和协方差矩阵,$\boldsymbol{\mu}_1$和$\boldsymbol{\mu}_2$是二维均值向量,$\boldsymbol{\Sigma}_1$和$\boldsymbol{\Sigma}_2$是 2×2 维的协方差矩阵。第 11～12 行前面定义的均值向量和协方差矩阵作为参数传入 MultivariateNormal,就实例化了两个二元高斯分布m_1和m_2。第 13～14 行调用m_1和m_2的 sample 方法分别生成 100 个样本。第 17～18 行设置样本对应的标签y,分别用 0 和 1 表示不同高斯分布的数据,也就是正样本和负样本。第 21 行使用 cat 函数将x_1和x_2组合在一起,第 22～24 行打乱样本和标签的顺序,将数据重新随机排列是十分重要的步骤,否则算法的每次迭代只会学习同一个类别的信息,容易造成模型过拟合。

【例 3-2】 构建多元高斯分布。

```
1  import numpy as np
2  from torch.distributions import MultivariateNormal
3
4  # 设置两个高斯分布的均值向量和协方差矩阵
5  mu1 = -3 * torch.ones(2)
6  mu2 = 3 * torch.ones(2)
7  sigma1 = torch.eye(2) * 0.5
8  sigma2 = torch.eye(2) * 2
9
10 # 各从两个多元高斯分布中生成 100 个样本
11 m1 = MultivariateNormal(mu1, sigma1)
```

```
m2 = MultivariateNormal(mu2, sigma2)
x1 = m1.sample((100,))
x2 = m2.sample((100,))

# 设置正负样本的标签
y = torch.zeros((200, 1))
y[100:] = 1

# 组合、打乱样本
x = torch.cat([x1, x2], dim = 0)
idx = np.random.permutation(len(x))
x = x[idx]
y = y[idx]

# 绘制样本
plt.scatter(x1.numpy()[:,0], x1.numpy()[:,1])
plt.scatter(x2.numpy()[:,0], x2.numpy()[:,1])
```

例 3-2 的第 27～28 行将生成的样本用 plt.scatter 绘制出来，绘制的结果如图 3-4 所示。可以很明显地看出多元高斯分布生成的样本聚成两个簇，并且簇的中心分别处于不同的位置（多元高斯分布的均值向量决定其位置）。右上角簇的样本分布更加稀疏而左下角簇的样本分布紧凑（多元高斯分布的协方差矩阵决定分布形状）。读者可自行调整例 3-2 第 5～6 行的参数，观察其变化。

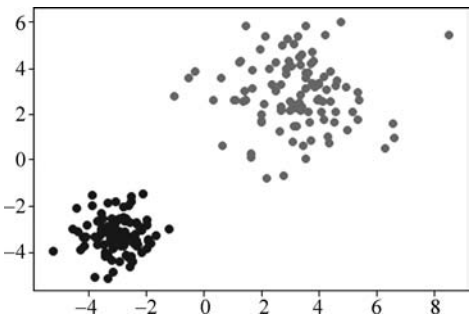

图 3-4　多元高斯分布生成的数据

2. 线性方程

逻辑回归用输入变量 X 的线性函数表示样本为正类的对数概率。torch.nn 中的 Linear 实现了 $y=x\boldsymbol{A}^{\mathrm{T}}+b$，我们可以直接调用它实现逻辑回归的线性部分。具体代码如例 3-3 所示。

【例 3-3】 Linear。

```
D_in, D_out = 2, 1
linear = nn.Linear(D_in, D_out, bias = True)
```

```
output = linear(x)

print(x.shape, linear.weight.shape, linear.bias.shape, output.shape)

def my_linear(x, w, b):
return torch.mm(x, w.t()) + b

torch.sum((output - my_linear(x, linear.weight, linear.bias)))
>>> torch.Size([200, 2]) torch.Size([1, 2]) torch.Size([1]) torch.Size([200, 1])
```

例 3-3 第 1 行定义线性模型的输入维度 D_in 和输出维度 D_out,因为前面定义的二维高斯分布 m_1 和 m_2 产生的变量是二维的,所以线性模型的输入维度应该定义为 D_in=2,而 logistic regression 是二分类模型,预测的是变量为正类的概率,所以输出的维度应该定义为 D_in=1。第 2～3 行实例化了 nn.Linear,将线性模型应用到数据 x 上,得到计算结果 output。

Linear 的初始参数是随机设置的,可以调用 Linear.weight 和 Linear.bias 获取线性模型的参数,第 5 行打印了输入变量 x、模型参数 weight 和 bias,计算结果 output 的维度。第 7～8 行定义我们实现的线性模型 my_linear,第 10 行将 my_linear 的计算结果和 PyTorch 的计算结果 output 做比较,可以发现其结果一致。

3. 激活函数

前文介绍了 torch.nn.Linear 可用于实现线性模型,除此之外,它还提供了机器学习中常用的激活函数,逻辑回归用于二分类问题时,使用 Sigmoid 函数将线性模型的计算结果映射到 0 和 1 之间,得到的计算结果作为样本为正类的置信概率。torch.nn.Sigmoid()提供了这一函数的计算,在使用时,将 Sigmoid 类实例化,再将需要计算的变量作为参数传递给实例化的对象。具体代码如例 3-4 所示。

【例 3-4】 Sigmoid 函数。

```
sigmoid = nn.Sigmoid()
scores = sigmoid(output)

def my_sigmoid(x):
    x = 1 / (1 + torch.exp(-x))
    return x

torch.sum(sigmoid(output) - sigmoid_(output))
>>> tensor(1.1190e-08, grad_fn=<SumBackward0>)
```

作为练习,第 4～6 行手动实现 Sigmoid 函数,第 8 行通过 PyTorch 验证我们的实现结果,其结果一致。

4. 损失函数

逻辑回归使用交叉熵作为损失函数。PyTorch 的 torch.nn 提供了许多标准的损失

函数，我们可以直接使用 torch.nn.BCELoss 计算二值交叉熵损失。第 1～2 行调用了 BCELoss 计算 logistic regression 模型的输出结果 sigmoid(output)和数据的标签 y，同样地，第 4～6 行自定义二值交叉熵函数，第 8 行将 my_loss 和 PyTorch 的 BCELoss 做比较，发现结果无差异。损失函数部分具体代码如例 3-5 所示。

【例 3-5】 损失函数。

```
1   loss = nn.BCELoss()
2   loss(sigmoid(output), y)
3
4   def my_loss(x, y):
5       loss = - torch.mean(torch.log(x) * y + torch.log(1 - x) * (1 - y))
6       return loss
7
8   loss(sigmoid(output), y) - my_loss(sigmoid_(output), y)
    >>> tensor(5.9605e-08, grad_fn=<SubBackward0>)
```

在前面的代码中，我们使用了 torch.nn 包中的线性模型 nn.Linear、激活函数 nn.Softmax()、损失函数 nn.BCELoss，它们都继承于 nn.Module 类，在 PyTorch 中，我们通过继承 nn.Module 构建自己的模型。接下来的例 3-6 用 nn.Module 实现逻辑回归。

【例 3-6】 用 nn.Module 实现逻辑回归。

```
1   import torch.nn as nn
2
3   class LogisticRegression(nn.Module):
4       def __init__(self, D_in):
5           super(LogisticRegression, self).__init__()
6           self.linear = nn.Linear(D_in, 1)
7           self.sigmoid = nn.Sigmoid()
8       def forward(self, x):
9           x = self.linear(x)
10          output = self.sigmoid(x)
11          return output
12
13  lr_model = LogisticRegression(2)
14  loss = nn.BCELoss()
15  loss(lr_model(x), y)
    >>> tensor(0.8890, grad_fn=<BinaryCrossEntropyBackward>)
```

通过继承 nn.Module 实现自己的模型时，forward()方法必须被子类覆写，在 forward 内部应当定义每次调用模型时执行的计算。从前面的应用可以看出，nn.Module 类的主要作用就是接收 Tensor 然后计算并返回结果。

在一个 Module 中，还可以嵌套其他的 Module，被嵌套的 Module 的属性就可以被自动获取。如例 3-7 所示，调用 nn.Module.parameters()方法获取 Module 所有保留的参数，调用 nn.Module.to()方法将模型的参数放置到 GPU 上等。

【例 3-7】 嵌套 Module。

```
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.linear1 = nn.Linear(1, 1, bias = False)
        self.linear2 = nn.Linear(1, 1, bias = False)
    def forward(self):
        pass

for param in MyModel().parameters():
    print(param)
>>> Parameter containing:
    tensor([[0.3908]], requires_grad = True)
    Parameter containing:
    tensor([[ - 0.8967]], requires_grad = True)
```

5. 优化算法

逻辑回归通常采用梯度下降法优化目标函数。PyTorch 的 torch.optim 包实现了大多数常用的优化算法,使用起来非常简单。具体步骤如下。

(1) 构建一个优化器。在构建时,需要将待学习的参数传入,然后传入优化器需要的参数,例如例 3-8 的学习率。

【例 3-8】 构建优化器。

```
from torch import optim
optimizer = optim.SGD(lr_model.parameters(), lr = 0.03)
```

(2) 迭代地对模型进行训练。首先调用损失函数的 backward()方法计算模型的梯度,然后再调用优化器的 step()方法更新模型的参数。需要注意的是,应当调用优化器的 zero_grad()方法清空参数的梯度。具体代码如例 3-9 所示。

【例 3-9】 模型训练。

```
batch_size = 10
iters = 10
#for input, target in dataset:
for _ in range(iters):
    for i in range(int(len(x)/batch_size)):
        input = x[i * batch_size:(i + 1) * batch_size]
        target = y[i * batch_size:(i + 1) * batch_size]
        optimizer.zero_grad()
        output = lr_model(input)
        l = loss(output, target)
        l.backward()
        optimizer.step()
>>> 模型准确率为: 1.0
```

6. 模型可视化

逻辑回归模型的判决边界在高维空间是一个超平面，而我们的数据集是二维的，所以判决边界只是平面内的一条直线，在线的一侧被预测为正类，另一侧则被预测为负类。下面的例 3-10 实现了 draw_decision_boundary 函数。

【例 3-10】 draw_decision_boundary 函数。

```
1   pred_neg = (output <= 0.5).view(-1)
2   pred_pos = (output > 0.5).view(-1)
3   plt.scatter(x[pred_neg, 0], x[pred_neg, 1])
4   plt.scatter(x[pred_pos, 0], x[pred_pos, 1])
5
6   w = lr_model.linear.weight[0]
7   b = lr_model.linear.bias[0]
8
9   def draw_decision_boundary(w, b, x0):
10      x1 = (-b - w[0] * x0) / w[1]
11      plt.plot(x0.detach().numpy(), x1.detach().numpy(), 'r')
12
13  draw_decision_boundary(w, b, torch.linspace(x.min(), x.max(), 50))
```

例 3-10 的 draw_decision_boundary 函数接收线性模型的参数 w、b 以及数据集 x，绘制判决边界的方法十分简单，如第 10 行，只需要计算一些数据在线性模型的映射值，即 $x_1=(-b-w_0x_0)/w_1$，然后调用 plt.plot 绘制线条即可。绘制的结果如图 3-5 所示。

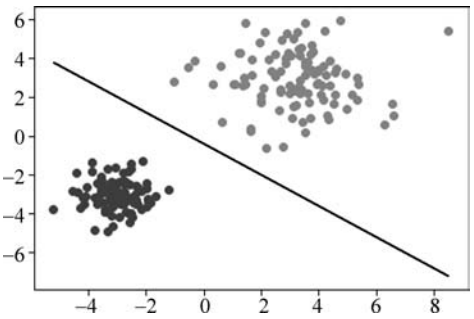

图 3-5　带有判决边界的分类结果

3.4　聚类算法

3.4.1　*K*-Means 聚类

聚类是将数据集中在某些方面相似的数据成员进行分类组织的过程，聚类就是一种发现这种内在结构的技术，聚类技术经常被称为无监督学习。*K*-Means 聚类是最著名的划分聚类算法，由于简洁和效率高使得它成为所有聚类算法中最广泛使用的。给定一个数据点集合和需要的聚类数目 K，K 由用户指定，*K*-Means 聚类算法根据某个距离函数

反复把数据分入 K 个聚类中。

K-Means 聚类算法(K-Means Clustering Algorithm)是一种迭代求解的聚类分析算法,其步骤如下:

(1) 预将数据分为 K 组,则随机选取 K 个对象作为初始的聚类中心。

(2) 计算每个对象与各个种子聚类中心之间的距离,把每个对象分配给距离它最近的聚类中心,聚类中心以及分配给它们的对象就代表一个聚类。

(3) 每分配一个样本,聚类的聚类中心会根据聚类中现有的对象被重新计算。

(4) 重复步骤(2)、(3)直到满足某个终止条件。终止条件可以是没有(或最小数目)对象被重新分配给不同的聚类,没有(或最小数目)聚类中心再发生变化,误差平方和局部最小。

3.4.2 均值漂移聚类

均值漂移(Meanshift)的基本思想是沿着密度上升方向寻找聚簇点。

假设在一个有 N 个样本点的特征空间,初始确定一个中心点 center,计算在设置的半径为 D 的圆形空间内所有的点与中心点 center 的向量。这时想要计算整个圆形空间内所有向量的平均值,得到一个偏移均值,就需要将中心点 center 移动到偏移均值位置。之后重复移动,直到满足一定条件结束。

均值漂移运算包括以下几步。

(1) 在未被分类的数据点中随机选择一个点作为中心点。

(2) 找出离中心点距离在带宽内的所有点,记作集合 M,认为这些点属于簇 c。

(3) 计算从中心点开始到集合 M 中每个元素的向量,将这些向量相加,得到偏移向量。

(4) 中心点沿着 shift 的方向移动,移动距离是偏移向量的模。

(5) 重复步骤(2)、(3)、(4),直到偏移向量的大小满足设定的阈值要求,记录此时的中心点。

(6) 重复步骤(1)、(2)、(3)、(4)、(5),直到所有的点都被归类。

(7) 分类:根据每个类对每个点的访问频率,取访问频率最高的那个类作为当前点集的所属类。

3.4.3 基于密度的聚类方法

与其他聚类方法相比,基于密度的聚类方法可以在嘈杂的数据中找到不同形状、不同大小的聚类。DBSCAN(Ester 等,1996) 是这类方法中最典型的代表算法之一(2014 年 DBSCAN 获得了 SIGKDD Test of Time Award)。核心思想是先找到密度更高的点,然后逐渐将相似的高密度点连接成一块,最后生成各种簇。算法实现的一般过程:首先,以每个数据点为中心,以 eps 为半径画一个圆(称为 eps-neighborhood,即邻域),统计这个圆中有多少个点,这个数字就是点密度值。其次,我们可以选择一个密度阈值 MinPts。例如,小于 MinPts 的圆心点为低密度点,大于或等于 MinPts 的圆心点为高密度点(Core Point)。如果在另一个高密度点的圆中有一个高密度点,我们将这两个点连接起来,这样

就可以连续串联许多点。最后，如果低密度点也在高密度点的圆内，则将其连接到最近的高密度点，称为边界点。这样，所有可以连接在一起的点就形成了一个簇，不在任何高密度点圆内的低密度点都是异常点。由于 DBSCAN 通过不断连接邻域内的高密度点发现簇，因此只需要定义邻域大小和密度阈值，就可以发现不同形状、不同大小的簇。

3.5　机器学习算法总结

3.5.1　逻辑回归和朴素贝叶斯

逻辑回归和朴素贝叶斯本质上都是线性分类模型。不同点在于逻辑回归基于损失函数最小化，而朴素贝叶斯基于贝叶斯定理和条件独立性假设；逻辑回归是判别模型而朴素贝叶斯是生成模型；朴素贝叶斯可应用于垃圾邮件过滤、文本分类等。

3.5.2　逻辑回归和支持向量机

逻辑回归和支持向量机都是有监督学习，本质都是线性分类判别模型。但它们的原理不同，主要有以下几点。

(1) 逻辑回归基于损失函数最小化，也可以说是经验风险最小化，而支持向量机基于最大化间隔，也就是结构风险最小化。

(2) 逻辑回归的分类决策面由所有样本决定，而支持向量机的决策面即分割超平面只由少数样本即支撑向量决定。

(3) 支持向量机算法在使用过程中涉及核函数，而逻辑回归一般不使用核函数。

(4) 逻辑回归使用正则化抑制过拟合，而支持向量机自带正则化。

(5) 支持向量机算法只给出结果属于哪一类，而逻辑回归算法在给出类别的同时，还给出了后验概率，使得逻辑回归可解释性更强，特征可控性更高。因此，其适用范围更广泛，可以用于医疗诊断、CTR 点击率预估和推荐系统等多种场合。

3.5.3　Bagging、随机森林和 Boosting

Bagging 和随机森林算法都是在所有样本中有放回地随机选取 n 个样本。不同的是，Bagging 算法使用所有样本特征训练得到一个分类器，重复 m 次得到 m 个分类器，最后采用投票的方式得到决策结果，各分类器权重一样。而随机森林算法在所有样本特征中随机选取 k 个特征，训练得到一个分类器，重复 m 次得到 m 个分类器，最后采用投票的方式得到决策结果，各分类器权重一样。

相对于 Bagging 算法而言，Boosting 算法各分类器的权重由其分类的准确率决定。