第3章

回　　归

与分簇、分类和标注任务不同，回归(Regression)任务预测的不是有限的离散的标签值，而是无限的连续值。回归任务的目标是通过对训练样本的学习，得到从样本特征集到连续值之间的映射。如天气预测任务中，预测天气是冷还是热是分类问题，而预测精确的温度值则是回归问题。

本章从较容易理解的线性回归入手，分别讨论了线性回归、多项式回归、岭回归和局部回归等算法。

本章引入了最优化计算、过拟合处理、向量相关性度量等机器学习基础知识。

某些神经网络也可完成回归任务，有关神经网络的算法将在后文有关章节中统一讨论。

视频

3.1　回归任务、评价与线性回归模型

3.1.1　回归任务

设样本集 $\boldsymbol{S}=\{\boldsymbol{s}_1,\boldsymbol{s}_2,\cdots,\boldsymbol{s}_m\}$ 包含 m 个样本，样本 $\boldsymbol{s}_i=(\boldsymbol{x}_i,y_i)$ 包括一个实例 $\boldsymbol{x}_i$ 和一个实数标签值 y_i，实例由 n 维特征向量表示，即 $\boldsymbol{x}_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})$。回归任务可分为学习过程和预测过程，如图 3-1 所示。

在学习过程，基于损失函数最小的思想，学习得到一个模型，该模型是从实例特征向量到实数的映射，用决策函数 $Y=f(X)$ 来表示，X 是定义域，它是所有实例特征向量的集合，Y 是值域 R。

记测试样本为 $\boldsymbol{x}=(x^{(1)},x^{(2)},\cdots,x^{(n)})$。在预测过程，利用学习到的模型来得到测试样本 $\boldsymbol{x}$ 的预测值 $\hat{y}$。

误差和误差平方是回归模型的评价指标，常作为损失函数，在下文结合线性回归进行讨论。

回归常表现为用曲线或曲面（二维或高维）去逼近分布于空间中的各样本点，因此也称为拟合。直线和平面可视为特殊的曲线和曲面。

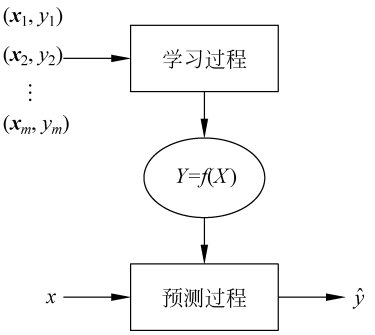


图 3-1　回归任务的模型

3.1.2　线性回归模型与回归评价指标

当用输入样本的特征的线性组合作为预测值时，就是线性回归（Linear Regression）。

记样本为 $\boldsymbol{s}=(\boldsymbol{x},y)$，其中 $\boldsymbol{x}$ 为样本的实例，$\boldsymbol{x}=(x^{(1)},x^{(2)},\cdots,x^{(n)})$，$x^{(j)}$ 为实例 $\boldsymbol{x}$ 的第 j 维特征，也直接称为该样本的第 j 维特征，y 为样本的标签，在回归问题中，y 是一个无限的连续值。

定义一个包含 n 个实数变量的集合 $\{w^{(1)},w^{(2)},\cdots,w^{(n)}\}$ 和一个实数变量 b，将样本的特征进行线性组合：

$$f(\boldsymbol{x})=w^{(1)}\cdot x^{(1)}+w^{(2)}\cdot x^{(2)}+\cdots+w^{(n)}\cdot x^{(n)}+b \tag{3-1}$$

就得到了线性回归模型，用向量表示为

$$f(\boldsymbol{x})=\boldsymbol{W}\cdot\boldsymbol{x}^{\mathrm{T}}+b \tag{3-2}$$

其中，向量 $\boldsymbol{W}=(w^{(1)}\quad w^{(2)}\quad\cdots\quad w^{(n)})$ 称为回归系数，负责调节各特征的权重，标量 b 称为偏置，负责调节总体的偏差。显然，在线性回归模型中，回归系数和偏置就是要学习的知识。

当只有 1 个特征时：

$$f(x)=w^{(1)}\cdot x^{(1)}+b \tag{3-3}$$

式(3-3)中，只有一个自变量，一个因变量，因此它可看作是二维平面上的直线。

下面介绍一个二维平面上的线性回归模型的例子。当温度处于 15～40℃时，某块草地上小花的数量和温度值的数据如表 3-1 所示。现在要来找出这些数据中蕴含的规律，用来预测其他未测温度时的小花的数量。

表 3-1　线性回归示例温度值和小花数量

温度/℃	15	20	25	30	35	40
小花数量/朵	136	140	155	160	157	175

以温度为横坐标，小花数量为纵坐标作出如图 3-2 所示的点和折线图。容易看出可以用一条直线来近似该折线。在二维平面上，用直线来逼近数据点，就是线性回归的思想，类似可以推广到高维空间中，如在三维空间中，用平面来逼近数据点。那么，如何求出线性回归模型中的回归系数 $\boldsymbol{W}$ 和偏置 b 呢？在此例中，也就是如何求出该直线的斜率和截距。要求出回归系数和偏置，首先要解决评价的问题，也就是哪条线才是最逼近所

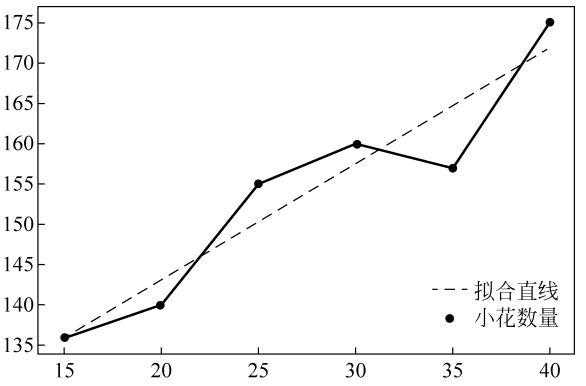


图 3-2　线性回归示例(见彩插)

有数据点的最佳直线。只有确定了标准才能有目的地寻找回归系数和偏置。

对于二维平面上的直线,有两个不重合的点即可确定,仅有一个点无法确定。现在的问题是,点不是少了,而是多了,那怎么解决此问题?一个思路是,让这条直线尽可能地贴近所有点。那怎么来衡量这个“贴近”呢?

在二维平面上,让一条线去尽可能地贴近所有点,直接的想法是使所有点到该直线的距离和最小,使之最小的直线被认为是最“好”的。

距离 l 计算起来比较麻烦,一般采用更容易计算的残差 s:

$$s_i = | y_i - f(x_i) | \tag{3-4}$$

式中,$f(x)$是拟采用的直线,如图 3-3 所示。容易理解,残差 s 与距离 l 之间存在等比例关系。因此,可以用所有点与该直线的残差和 $\sum s_i$ 代替距离和 $\sum l_i$ 作为衡量“贴近”程度的标准。

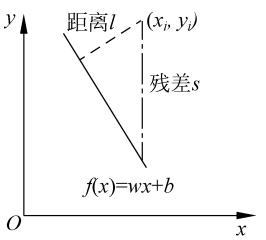


图 3-3　距离和残差

式(3-4)中,$f(x_i)$即为预测值 $\hat{y}_i$。因为残差需要求绝对值,后续计算时比较麻烦,尤其是在一些需要求导的场合,因此常采用残差的平方作为衡量“贴近”程度的指标:

$$s_i^2 = (y_i - \hat{y}_i)^2 \tag{3-5}$$

以上分析过程是基于线性回归模型的。非线性回归模型也采用式(3-5)所示的评价指标。

如同轮廓系数和 DB 指数等是分簇模型的评价指标,残差和残差平方是回归模型的常用评价指标。

残差称为绝对误差(Absolute Loss),残差平方称为误差平方(Squared Loss)。误差平方对后续计算比较方便,因此常采用所有点的误差平方和(Sum of Squared Error, SSE)作为损失函数来评价回归算法,此种命名方式与聚类的误差平方和损失函数相同。

还经常采用均方误差(Mean of Squared Error, MSE)作为损失函数,它的计算方法是误差平方和除以样本总数。在样本集确定的情况下,样本总数是常数,因此,在求极值时,均方误差和误差平方和作为目标函数并没有区别。但是,均方误差体现单个样本的平均误差,因此可用来比较不同容量样本集上的误差。

3.1.3 最小二乘法求解线性回归模型

最小二乘法是解析法，即用矩阵等数学知识直接求解线性回归模型的方法。

式(3-2)不便于矩阵推导，线性回归的模型常用式(3-6)的表示方法。

$$f(\boldsymbol{x})=\boldsymbol{W}\cdot\boldsymbol{x}=\sum_{i=0}^{n}w^{(i)}\cdot x^{(i)} \tag{3-6}$$

其中，$\boldsymbol{x}=(x^{(0)}\quad x^{(1)}\quad\cdots\quad x^{(n)})^{\mathrm{T}}$ 为特征向量，并指定 $x^{(0)}=1$，$\boldsymbol{W}=(w^{(0)}\quad w^{(1)}\quad\cdots\quad w^{(n)})$ 为系数向量，并指定 $w^{(0)}=b$。

如果求出 $\boldsymbol{W}$，那么问题就解决了。如前文所述，$\boldsymbol{W}$ 就是使训练集所有样本的误差平方和最小的那组系数。对于第 i 个样本来说，其误差平方 $s_i^2(\boldsymbol{W})$ 为实际值 y_i 与预测值 $f(\boldsymbol{x}_i)$ 之差的平方：

$$s_i^2(\boldsymbol{W})=(y_i-f(\boldsymbol{x}_i))^2=(y_i-\boldsymbol{W}\cdot\boldsymbol{x}_i)^2 \tag{3-7}$$

假设有 m 个训练样本时，误差平方和 $L(\boldsymbol{W})$ 为

$$\begin{aligned}L(\boldsymbol{W})&=s_1^2(\boldsymbol{W})+s_2^2(\boldsymbol{W})+\cdots+s_m^2(\boldsymbol{W})\\&=(y_1-\boldsymbol{W}\cdot\boldsymbol{x}_1)^2+(y_2-\boldsymbol{W}\cdot\boldsymbol{x}_2)^2+\cdots+(y_m-\boldsymbol{W}\cdot\boldsymbol{x}_m)^2\\&=(y_1-\boldsymbol{W}\cdot\boldsymbol{x}_1\quad y_2-\boldsymbol{W}\cdot\boldsymbol{x}_2\quad\cdots\quad y_m-\boldsymbol{W}\cdot\boldsymbol{x}_m)\begin{pmatrix}y_1-\boldsymbol{W}\cdot\boldsymbol{x}_1\\y_2-\boldsymbol{W}\cdot\boldsymbol{x}_2\\\vdots\\y_m-\boldsymbol{W}\cdot\boldsymbol{x}_m\end{pmatrix}\end{aligned} \tag{3-8}$$

令 $\boldsymbol{Y}=(y_1\quad\cdots\quad y_m)$，$\bar{\boldsymbol{X}}=(\boldsymbol{x}_1\quad\cdots\quad\boldsymbol{x}_m)=\begin{pmatrix}x_1^{(0)}&\cdots&x_m^{(0)}\\\vdots&\ddots&\vdots\\x_1^{(n)}&\cdots&x_m^{(n)}\end{pmatrix}$，上式可表示为

$$L(\boldsymbol{W})=(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}} \tag{3-9}$$

$L(\boldsymbol{W})$ 是要优化的目标，当样本 $\bar{\boldsymbol{X}}$ 确定后，它的取值只与 $\boldsymbol{W}$ 有关，要使它达到最小值，即

$$\hat{\boldsymbol{W}}=\arg\min_{\boldsymbol{W}}L(\boldsymbol{W})=\arg\min_{\boldsymbol{W}}(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}} \tag{3-10}$$

求 $\boldsymbol{W}$ 使得 $L(\boldsymbol{W})$ 最小化的过程，称为线性回归模型的最小二乘“参数估计”。对 $\boldsymbol{W}^{\mathrm{T}}$ 求导，即

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}}&=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}[(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}^{\mathrm{T}}-\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}})]\\&=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}[\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}-\boldsymbol{W}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}}-\boldsymbol{Y}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}+\boldsymbol{W}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}]\\&=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}-\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{W}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}}-\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{Y}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}\\&\quad+\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{W}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}\end{aligned} \tag{3-11}$$

其中,第一项是常量求导,所以

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}=0 \tag{3-12}$$

第二项和第三项互为转置,且是标量,所以相等,由矩阵求导法则[①]可知:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{W}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}}=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{Y}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}=\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}} \tag{3-13}$$

第四项也是标量对向量求导,同样由矩阵求导法则:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}\boldsymbol{W}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}=2\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}} \tag{3-14}$$

所以

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}}=2\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}-2\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}}=2\bar{\boldsymbol{X}}(\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}-\boldsymbol{Y}^{\mathrm{T}}) \tag{3-15}$$

令式(3-15)为 0,在 $\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}$ 可逆时,可得 $\boldsymbol{W}$ 的估计:

$$\hat{\boldsymbol{W}}^{\mathrm{T}}=(\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}})^{-1}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}} \tag{3-16}$$

对表 3-1 所示数据用最小二乘法回归分析的代码见代码 3-1。

代码 3-1　最小二乘法求解线性回归(线性回归.ipynb)

```
temperatures = [10, 15, 20, 25, 30,35]
insects = [39, 42, 50, 57, 58, 65]

import numpy as np
def least_square(X, Y):
    '''
    para X:矩阵,样本特征矩阵
    para Y:矩阵,标签向量
    return:矩阵,回归系数
    '''
    W = (X * X.T).I * X * Y.T
    return W

X = np.mat([[1,1,1,1,1,1], temperatures])
Y = np.mat(insects)

W = least_square(X, Y)
'''
matrix([[114.39047619],
        [  1.43428571]])
'''
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

① 程云鹏,等.矩阵论.3 版.西安:西北工业大学出版社,2006.

```
plt.scatter(temperatures, flowers, color = "green", label = "小花数量", linewidth = 2)
plt.plot(temperatures,flowers,linewidth = 1)
x1 = np.linspace(15, 40, 100)
y1 = W[1,0] * x1 + W[0,0]
plt.plot(x1, y1, color = "red", label = "拟合直线", linewidth = 2,linestyle = ':')
plt.legend(loc = 'lower right')
plt.show()
new_tempera = [18, 22, 33]
new_tempera = (np.mat(new_tempera)).T
pro_num = W[1,0] * new_tempera + W[0,0]
print(pro_num)
'''
[[140.20761905]
 [145.9447619 ]
 [161.72190476]]
'''
```

第1、2行是样本数据。第5～12行是最小二乘法的函数，它是式(3-16)的实现，numpy包提供了很简便的方法来完成矩阵运算。第19、20行是求解后得到的模型参数。第31行输出的图如图3-2所示。

第32行开始是用训练好的模型来预测18℃、22℃、33℃时的昆虫数量，结果为140、146和162。

在scipy包中提供了最小二乘法算法scipy.optimize.leastsq，可以直接调用。

除了最小二乘法外，线性回归问题还有一些其他解析求解方法，如正规方程组法等。

sklearn.linear_model包中提供了线性回归模型：LinearRegression。

3.2 机器学习中的最优化方法


视频

最小二乘法等解析法在面临大数据量时，存在效率低的问题，而且大部分机器学习问题非常复杂，难以用数学模型来表达，因此，更多的机器学习模型要采用最优化方法来求解。在k-means算法的进一步讨论中已经提到最优化计算，本节集中讨论机器学习中的最优化方法。最优化计算在机器学习中具有十分重要的作用，大部分机器学习任务最后都可归结为最优化问题。最优化问题在军事、工程、管理等领域有着广泛的应用。

3.2.1 最优化模型

最优化理论是以矩阵论、数值分析和计算机技术为基础发展起来的。基于最优化理论发展而来的常用最优化方法有单纯型法、惩罚函数法等，在机器学习中应用最多的是导数方法，如梯度下降法、牛顿法、拟牛顿法、共轭梯度法等。

最优化方法是研究在多元变量系统中，如何科学配置各元的值，使系统达到最佳的方法。系统最佳用某一指标来衡量，通常是达到最小值(求最大值的问题可通过加负号转化为求最小值问题)。最优化问题的基本数学模型见式(3-17)。

$$\min_{x\in R} f(\boldsymbol{x}) \quad \text{s.t.} \begin{cases} h_i(\boldsymbol{x})=0 \\ g_j(\boldsymbol{x})\leqslant 0 \end{cases} \tag{3-17}$$

其中，$\boldsymbol{x}$ 是一个位于实数域 R 的 n 维向量；s.t. 为英文 subject to 的缩写，表示“受限于”；$f(\boldsymbol{x})$称为目标函数或代价函数；$h(\boldsymbol{x})$为等式约束；$g(\boldsymbol{x})$为不等式约束。在各种约束条件下，使 $f(\boldsymbol{x})$达到最小的 $\boldsymbol{x}$ 被称为问题的解。

无约束的最优化问题简化表述为

$$\arg\min_{x} f(\boldsymbol{x}) \tag{3-18}$$

式(3-10)就是线性回归问题的最优化表述方式。

3.2.2 迭代法

对于大部分最优化问题来说，很难像线性回归问题那样能求得解析解，一般需要利用计算机快速运算的特点，采用迭代法(Iteration)求解。迭代法又称辗转法或逐次逼近法。

迭代法与其说是一种算法，更是一种思想，它不像传统数学方法那样一步到位得到精确解，而是步步为营，逐次推进，逐步接近。迭代法是现代计算机求解问题的一种基本形式。

迭代法的核心是建立迭代关系式。迭代关系式指明了前进的方式，只有正确的迭代关系式才能取得正确解。

下面用解方程的例子来说明它在数值计算领域的应用。在迭代法求解中，每次迭代都得到一个新的 x 值，将每次迭代得到的 x 值依序排列就可得到数列$\{x_k\}$，x_0 为选定的初值。在用迭代法求解方程时有个常用的迭代关系式建立方法，先将方程 $f(x)=0$ 变换为 $x=\varphi(x)$，然后建立起迭代关系式：

$$x_{k+1}=\varphi(x_k) \tag{3-19}$$

如果$\{x_k\}$收敛于 x^*，那么 x^* 就是方程的根，因为：

$$x^*=\lim_{k\to\infty} x_{k+1}=\lim_{k\to\infty}\varphi(x_k)=\varphi(\lim_{k\to\infty} x_k)=\varphi(x^*) \tag{3-20}$$

即，当 $x=x^*$ 时，有 $f(x)=x-\varphi(x)=0$。

用迭代法求下列方程的解：

$$x^3+\frac{e^x}{2}+5x-6=0 \tag{3-21}$$

该方程很难用解析的方法求解。建立迭代关系式为

$$x=\frac{\left(6-x^3-\frac{e^x}{2}\right)}{5} \tag{3-22}$$

迭代的结束条件是实际应用时需要考虑的问题。在无法预估时，可采用控制总的迭代次数的办法。也可以根据数列$\{x_k\}$的变化情况来判断，如将$|x_{k+1}-x_k|$的值小于某个阈值作为结束的标准。

用迭代法求解方程的示例代码见代码 3-2。

代码 3-2 迭代法求解方程示例(迭代法.ipynb)

```
1. import math
2. x = 0
3. for i in range(100):
4.     x = (6 - x**3 - (math.e**x)/2.0)/5.0
5.     print(str(i) + ":" + str(x))
```

运行结果显示从 28 次迭代开始,收敛于 0.84592。

3.2.3 梯度下降法

梯度下降(Gradient Descent)法是迭代法中利用导数进行优化的算法。在求解机器学习模型参数时,梯度下降法是最常用的方法。

1. 基本思想

如前文所述,迭代关系式是迭代法应用时的关键问题,而梯度下降法正是用梯度来建立迭代关系式的迭代法。

对于无约束优化问题$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$,其梯度下降法求解的迭代关系式为

$$\boldsymbol{x}_{i+1}=\boldsymbol{x}_i+\alpha\cdot\left(-\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}\right)\Bigg|_{\boldsymbol{x}=\boldsymbol{x}_i}=\boldsymbol{x}_i-\alpha\cdot\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}\Bigg|_{\boldsymbol{x}=\boldsymbol{x}_i} \tag{3-23}$$

式中,$\boldsymbol{x}$ 为多维向量,记为 $\boldsymbol{x}=(x^{(1)},x^{(2)},\cdots,x^{(n)})$;$\alpha$ 为正实数,称为步长,也称为学习率;$\dfrac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}=\left(\dfrac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}^{(1)}}\quad\dfrac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}^{(2)}}\quad\cdots\quad\dfrac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}^{(n)}}\right)$是 $f(\boldsymbol{x})$的梯度函数。

式(3-23)的含义可用将向量 $\boldsymbol{x}$ 的函数简化为一元变量 x 的函数来示意,如图 3-4 所示。

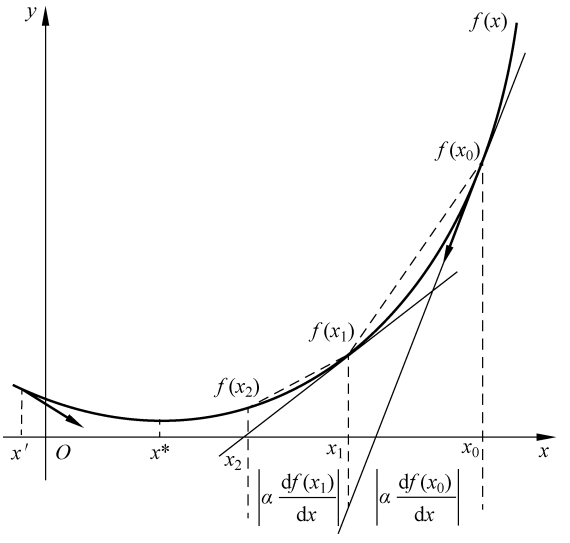


图 3-4 梯度下降法示意(见彩插)

先来看 x 前进的方向。迭代关系式(3-23)是当前的 x 加上步长 α 与负梯度的乘积。负梯度的方向可以确保 x 始终向函数极小值的方向前进。在图中的点 x_0,函数 $f(x)$的负梯度方向指向左,而在点 x',函数 $f(x)$的负梯度方向指向右,分别如图 3-4 中粗箭头所示。

再来看 x 前进的量。一元函数 $f(x)$在点 x_0 上的导数定义为:$\dfrac{\mathrm{d}f(x_0)}{\mathrm{d}x}=\lim\limits_{x\to x_0}\dfrac{f(x)-f(x_0)}{x-x_0}$,它在几何上指的是 $f(x)$在点 x_0 处的切线方向,也就是斜率。切线方向是该点函数值增长最快的方向,切线相反的方向是函数降低最快的方向。可以看到,在“陡峭”的地方,$\left|\dfrac{\mathrm{d}f(x_i)}{\mathrm{d}x}\right|$值要大,而“平缓”的地方,$\left|\dfrac{\mathrm{d}f(x_i)}{\mathrm{d}x}\right|$值要小。因此,$x$ 前进的量 $\alpha\left|\dfrac{\mathrm{d}f(x_i)}{\mathrm{d}x}\right|$会随着“陡峭”程度而变化,越“陡”的地方前进越多。

图 3-4 示意的梯度下降法的迭代过程中,第一次迭代是从点 x_0 开始,沿 $f(x)$在该点的梯度反方向(图中右侧粗箭头所示)前进了 $\alpha\left|\dfrac{\mathrm{d}f(x_0)}{\mathrm{d}x}\right|$长度到达 x_1 点,函数值则从 $f(x_0)$变为 $f(x_1)$。第二次迭代是从点 x_1 开始,沿该点梯度反方向再一次前进了 $\left|\alpha\dfrac{\mathrm{d}f(x_1)}{\mathrm{d}x}\right|$长度到达 x_2 点,函数值则从 $f(x_1)$变为 $f(x_2)$。如此多次迭代,逐次逼近使 $f(x)$取得最小值的 x^*。

该过程可以推广到多元变量函数中。多元变量函数的梯度 $\dfrac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}=\left(\dfrac{\partial f(\boldsymbol{x})}{\partial x^{(1)}}\quad\dfrac{\partial f(\boldsymbol{x})}{\partial x^{(2)}}\quad\cdots\quad\dfrac{\partial f(\boldsymbol{x})}{\partial x^{(n)}}\right)$是该函数增长最快的方向。二元变量函数沿梯度反方向下降的迭代过程可以在三维空间中形象地显示出来,如图 3-5 所示。从初始点出发,沿下降最快的方向前进,直到到达极低点。

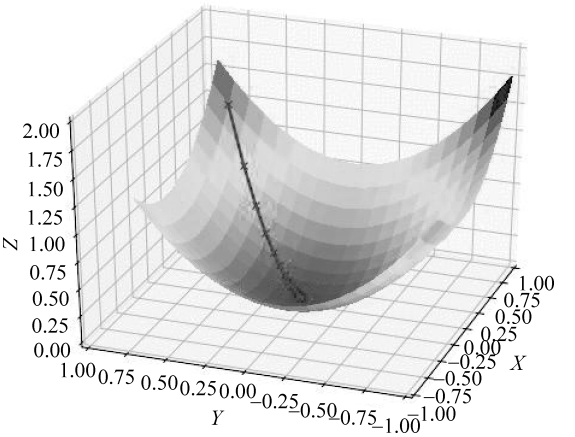


图 3-5 二元变量函数沿梯度下降的迭代过程示意图[①](见彩插)

① https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

下面讨论梯度下降法的几个问题。

(1) 梯度下降法的结束条件,一般采用以下方法:①迭代次数达到了最大设定;②损失函数降低幅度低于设定的阈值。

(2) 关于步长 α,过大时,初期下降的速度很快,但有可能越过最低点,如果"洼地"够大,会再折回并反复振荡(见图 3-6)。如果步长过小,则收敛的速度会很慢。因此,可以采取先大后小的策略调整步长,具体大小的调节可根据 $f(\boldsymbol{x})$降低的幅度或者 $\boldsymbol{x}$ 前进的幅度进行。

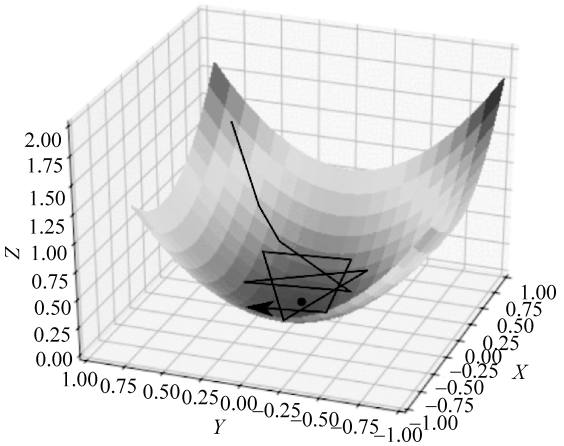


图 3-6 梯度下降法中步长过大时振荡示意图①(见彩插)

(3) 关于特征归一化问题,梯度下降法应用于机器学习模型求解时,对特征的取值范围也是敏感的,当不同的特征值取值范围不一样时,相同的步长会导致尺度小的特征前进比较慢,从而走之字形路线,影响迭代的速度,甚至不收敛。

下面用梯度下降法来求解式(3-21)所示的方程作为简单示例。

令 $f(x)=x^3+\frac{e^x}{2}+5x-6$。求方程的根并不是求函数的极值,因此,并不能直接套用梯度下降法来求解。为了迭代到取值为 0 的点,可采取对原函数取绝对值或者求平方作为损失函数,这样损失函数取得最小值的点,也就是原函数为 0 的点。但是绝对值函数不便于求梯度,因此,一般采用对原函数求平方的方法来得到损失函数。求解的代码见代码 3-3。

代码 3-3 梯度下降法求解方程示例(迭代法.ipynb)

```
1. import numpy as np
2. import math
3.
4. def f(x):
```

① https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

```
5.      return x**3 + (math.e**x)/2.0 + 5.0*x - 6
6.
7.  def loss_fun(x):
8.      return (f(x))**2
9.
10. def calcu_grad(x):
11.     delta = 0.0000001
12.     return(loss_fun(x+delta) - loss_fun(x-delta))/(2.0*delta)
13.
14. alpha = 0.01
15. maxTimes = 100
16. x = 0.0
17.
18. for i in range(maxTimes):
19.     x = x - alpha * calcu_grad(x)
20.     print(str(i)+":"+str(x))
```

第10～12行是计算导数(即梯度),采用了类似导数的定义式的近似计算方法。第14行是步长设为0.01。第15行是结束条件,简单设为最大次数100。运行结果显示从17次迭代开始,稳定收敛于0.84592,比纯粹的迭代法收敛要快。

2. 梯度下降法求解线性回归问题

由前面分析,将线性回归问题中 m 个样本的损失函数表示为

$$L(\boldsymbol{W})=\frac{1}{2}[s_1^2(\boldsymbol{W})+s_2^2(\boldsymbol{W})+\cdots+s_m^2(\boldsymbol{W})]=\frac{1}{2}\sum_{i=1}^{m}s_i^2(\boldsymbol{W}) \tag{3-24}$$

这里乘$\frac{1}{2}$,是为了求导后去掉常数系数,不影响用梯度下降法求解最小值。

由式(3-23)可知回归系数的更新过程如下:

$$w_{l+1}^{(j)}=w_l^{(j)}-\alpha\frac{\partial L(\boldsymbol{W})}{\partial w^{(j)}},\quad \text{对每一个特征 } j \tag{3-25}$$

其中:

$$\begin{aligned}\frac{\partial L(\boldsymbol{W})}{\partial w^{(j)}}&=\frac{1}{2}\sum_{i=1}^{m}\frac{\partial s_i^2(\boldsymbol{W})}{\partial w^{(j)}}=\frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w^{(j)}}(y_i-f(x_i))^2\\&=-\sum_{i=1}^{m}(y_i-f(x_i))\frac{\partial f(x_i)}{\partial w^{(j)}}=-\sum_{i=1}^{m}(y_i-f(x_i))x_i^{(j)}\\&=-\sum_{i=1}^{m}\left(y_i-\sum_{k=0}^{n}x_i^{(k)}\cdot w^{(k)}\right)x_i^{(j)}\end{aligned} \tag{3-26}$$

用表3-1的例子来示例梯度下降法。计算梯度的代码见代码3-4,第12行代码对应式(3-26)中括号内式子 $y_i-\sum_{k=0}^{n}x_i^{(k)}\cdot w_l^{(k)}$ 的计算。

代码 3-4 梯度的计算(迭代法.ipynb)

```
import numpy as np
def gradient(x, y, w):
    '''计算一阶导函数的值
    para x:矩阵, 样本集
    para y:矩阵, 标签
    para w:矩阵, 线性回归模型的参数
    return:矩阵, 一阶导数值
    '''
    m, n = np.shape(x)
    g = np.mat(np.zeros((n, 1)))
    for i in range(m):
        err = y[i, 0] - x[i, ] * w
        for j in range(n):
            g[j, ] -= err * x[i, j]
    return g
```

计算损失函数值的代码见代码 3-5,通过将误差矩阵转置再自乘,以误差平方和作为损失函数。

代码 3-5 损失函数值的计算(迭代法.ipynb)

```
def lossValue(x, y, w):
    '''计算损失函数
    para x:矩阵, 样本集
    para y:矩阵, 标签
    para w:矩阵, 线性回归模型的参数
    return:损失函数值'''
    k = y - x*w
    return k.T * k / 2
```

主程序见代码 3-6。

代码 3-6 梯度下降法求解线性回归问题示例(迭代法.ipynb)

```
temperatures = [15, 20, 25, 30, 35,40]
flowers = [136, 140, 155, 160, 157, 175]
X = (np.mat([[1,1,1,1,1,1], temperatures])).T
y = (np.mat(flowers)).T

W = (np.mat([0.0,0.0])).T
print(W)
# alpha = 0.0005 步长太大,来回振荡,无法收敛
alpha = 0.00025
loss_change = 0.000001
loss = lossValue(X, y, W)
for i in range(30000):
    W = W - alpha * gradient(X, y, W)
```

```
    newloss = lossValue(X, y, W)
    print(str(i) + ":" + str(W[0]) + ':' + str(W[1]))
    print(newloss)
    if abs(loss - newloss) < loss_change:
        break
    loss = newloss

new_tempera = [18, 22, 33]
new_tempera = (np.mat([[1,1,1], new_tempera])).T
pro_num = new_tempera * W
print(pro_num)
```

当循环达到最大次数 30 000,或者损失函数值的变化小于 0.000 001 时,程序终止。对 3 个实例预测的结果为:139,145 和 161。

第 8 行中,当把步长设为 0.0005 时,则会因为步长太大而直接越过洼地,无法收敛。

在随书资源中的"迭代法.ipynb"文件中,还给出了特征归一化的例子,供参考。

sklearn 的 linear_model 包中实现了梯度下降回归,类名为:SGDRegressor。

3. 随机梯度下降和批梯度下降

从梯度下降算法的处理过程,可知梯度下降法在每次计算梯度时,都涉及全部样本。在样本数量特别大时,算法的效率会很低。随机梯度下降法(Stochastic Gradient Descent,SGD),试图改正这个问题,它不是通过计算全部样本来得到梯度,而是随机选择一个样本来计算梯度。随机梯度下降法不需要计算大量的数据,所以速度快,但得到的并不是真正的梯度,可能会造成不收敛的问题。

批梯度下降法(Batch Gradient Descent,BGD)是一个折中方法,每次在计算梯度时,选择小批量样本进行计算,既考虑了效率问题,又考虑了收敛问题。

3.2.4 全局最优与凸优化

前文提到过聚类算法中存在局部最优和全局最优问题。该问题在机器学习领域是常见问题。本小节讨论凸函数在解决局部最优问题中的作用。在损失函数为凸函数的优化中,不存在局部最优的问题,因此,如果能将损失函数转化为凸函数,就可以解决此问题。

1. Hessian 矩阵

Hessian 矩阵(Hessian Matrix),常翻译为海森矩阵、黑塞矩阵、海瑟矩阵、海塞矩阵等,是一个多元函数的二阶偏导数构成的方阵,它描述了函数的局部曲率。海森矩阵最早于 19 世纪由德国数学家 Ludwig Otto Hesse 提出,并以其名字命名。利用 Hessian 矩阵可判定多元函数的极值问题,在凸函数判定和牛顿法解优化问题中常用。

若一元函数 $f(x)$在 $x=x_0$ 点的某个邻域具有任意阶导数,则可以将 $f(x)$在 x_0 处展开成泰勒级数:

$$f(x)=f(x_0)+f'(x_0)\Delta x+f''(x_0)\Delta x^2+\cdots \tag{3-27}$$

其中，$\Delta x=(x-x_0)$，$\Delta x^2=(x-x_0)^2$。

二元函数 $f(x^{(1)},x^{(2)})$ 在 $x_0=(x_0^{(1)},x_0^{(2)})$ 点处的泰勒级数展开式为

$$\begin{aligned}f(x^{(1)},x^{(2)})=&\ f(x_0^{(1)},x_0^{(2)})+\frac{\partial f}{\partial x^{(1)}}\bigg|_{x_0}\Delta x^{(1)}+\frac{\partial f}{\partial x^{(2)}}\bigg|_{x_0}\Delta x^{(2)}\\&+\frac{1}{2}\left[\frac{\partial^2 f}{\partial x^{(1)^2}}\bigg|_{x_0}\Delta x^{(1)^2}+\frac{\partial^2 f}{\partial x^{(1)}\partial x^{(2)}}\bigg|_{x_0}\Delta x^{(1)}\Delta x^{(2)}\right.\\&\left.+\frac{\partial^2 f}{\partial x^{(2)}\partial x^{(1)}}\bigg|_{x_0}\Delta x^{(1)}\Delta x^{(2)}+\frac{\partial^2 f}{\partial x^{(2)^2}}\bigg|_{x_0}\Delta x^{(2)^2}\right]+\cdots\end{aligned} \tag{3-28}$$

其中，$\Delta x^{(1)}=(x^{(1)}-x_0^{(1)})$，$\Delta x^{(2)}=(x^{(2)}-x_0^{(2)})$。

将式(3-28)写成矩阵形式：

$$\begin{aligned}f(\boldsymbol{x})=&f(\boldsymbol{x}_0)+\left(\frac{\partial f}{\partial x^{(1)}},\frac{\partial f}{\partial x^{(2)}}\right)_{\boldsymbol{x}_0}\begin{pmatrix}\Delta x^{(1)}\\ \Delta x^{(2)}\end{pmatrix}+\\&\frac{1}{2}(\Delta x^{(1)},\Delta x^{(2)})\begin{bmatrix}\dfrac{\partial^2 f}{\partial x^{(1)^2}} & \dfrac{\partial^2 f}{\partial x^{(1)}\partial x^{(2)}}\\ \dfrac{\partial^2 f}{\partial x^{(2)}\partial x^{(1)}} & \dfrac{\partial^2 f}{\partial x^{(2)^2}}\end{bmatrix}_{x_0}\begin{pmatrix}\Delta x^{(1)}\\ \Delta x^{(2)}\end{pmatrix}+\cdots\end{aligned} \tag{3-29}$$

记：

$$\nabla f(\boldsymbol{x}_0)=\begin{pmatrix}\dfrac{\partial f}{\partial x^{(1)}}\\ \dfrac{\partial f}{\partial x^{(2)}}\end{pmatrix}_{\boldsymbol{x}_0},\quad G(\boldsymbol{x}_0)=\begin{bmatrix}\dfrac{\partial^2 f}{\partial x^{(1)^2}} & \dfrac{\partial^2 f}{\partial x^{(1)}\partial x^{(2)}}\\ \dfrac{\partial^2 f}{\partial x^{(2)}\partial x^{(1)}} & \dfrac{\partial^2 f}{\partial x^{(2)^2}}\end{bmatrix}_{\boldsymbol{x}_0},\quad \Delta\boldsymbol{x}=\begin{pmatrix}\Delta x^{(1)}\\ \Delta x^{(2)}\end{pmatrix} \tag{3-30}$$

得到式(3-31)：

$$f(\boldsymbol{x})=f(\boldsymbol{x}_0)+\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\Delta\boldsymbol{x}+\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}G(\boldsymbol{x}_0)\Delta\boldsymbol{x}+\cdots \tag{3-31}$$

其中，$G(\boldsymbol{x}_0)$ 是 $f(\boldsymbol{x})$ 在 $\boldsymbol{x}_0$ 点处的 Hessian 矩阵。它是函数 $f(x^{(1)},x^{(2)})$ 在 $\boldsymbol{x}_0=(x_0^{(1)},x_0^{(2)})$ 点处的二阶偏导数所组成的方阵。

将二元函数的泰勒展开式进一步推广到多元函数，则函数 $f(x^{(1)},x^{(2)},\cdots,x^{(n)})$ 在 $\boldsymbol{x}_0=(x_0^{(1)},x_0^{(2)},\cdots,x_0^{(n)})$ 点处的泰勒展开式的矩阵形式为

$$f(\boldsymbol{x})=f(\boldsymbol{x}_0)+\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\Delta\boldsymbol{x}+\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}G(\boldsymbol{x}_0)\Delta\boldsymbol{x}+\cdots \tag{3-32}$$

其中：

$$\nabla f(\boldsymbol{x}_0)=\left[\frac{\partial f}{\partial x^{(1)}},\frac{\partial f}{\partial x^{(2)}},\cdots,\frac{\partial f}{\partial x^{(n)}}\right]^{\mathrm{T}}_{\boldsymbol{x}_0} \tag{3-33}$$

是 $f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处的梯度。

$$G(\boldsymbol{x}_0)=\begin{bmatrix} \dfrac{\partial^2 f}{\partial x^{(1)^2}} & \dfrac{\partial^2 f}{\partial x^{(1)}\partial x^{(2)}} & \cdots & \dfrac{\partial^2 f}{\partial x^{(1)}\partial x^{(n)}} \\ \dfrac{\partial^2 f}{\partial x^{(2)}\partial x^{(1)}} & \dfrac{\partial^2 f}{\partial x^{(2)^2}} & \cdots & \dfrac{\partial^2 f}{\partial x^{(2)}\partial x^{(n)}} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x^{(n)}\partial x^{(1)}} & \dfrac{\partial^2 f}{\partial x^{(n)}\partial x^{(2)}} & \cdots & \dfrac{\partial^2 f}{\partial x^{(n)^2}} \end{bmatrix}_{\boldsymbol{x}_0} \tag{3-34}$$

是 $f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处的 Hessian 矩阵。

如果 $f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处二阶连续可导,那么$\dfrac{\partial^2 f}{\partial x^{(1)}\partial x^{(2)}}=\dfrac{\partial^2 f}{\partial x^{(2)}\partial x^{(1)}}$,Hessian 矩阵为对称矩阵。Hessian 矩阵可用来判断多元函数的极值。

设 $f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处二阶连续可导,且有$\nabla f(\boldsymbol{x}_0)=0$,那么:①当 $G(\boldsymbol{x}_0)$是正定矩阵时,$f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处是极小值;②当 $G(\boldsymbol{x}_0)$是负定矩阵时,$f(\boldsymbol{x})$在 $\boldsymbol{x}_0$ 点处是极大值;③当 $G(\boldsymbol{x}_0)$是不定矩阵时,$\boldsymbol{x}_0$ 不是极值点;④当 $G(\boldsymbol{x}_0)$是半正定矩阵或半负定矩阵时,$\boldsymbol{x}_0$ 点是可疑极值点。

用 Hession 矩阵求极值:求三元函数 $f(x,y,z)=x^2+2xy+2y^2+z^2+6x$ 的极值。

解:令各变量的梯度为 0:

$$\frac{\partial f}{\partial x}=2x+2y+6=0$$

$$\frac{\partial f}{\partial y}=2x+4y=0$$

$$\frac{\partial f}{\partial z}=2z=0$$

可得驻点(−6,3,0)。

因此,Hessian 矩阵为 $\begin{bmatrix}2&2&0\\2&4&0\\0&0&2\end{bmatrix}$,是正定矩阵,故该驻点是极小值点,极小值为 $f(-6,3,0)=-18$。

在随书资源“迭代法.ipynb”文件中,给出了验证矩阵 $\begin{bmatrix}2&2&0\\2&4&0\\0&0&2\end{bmatrix}$ 为正定矩阵的代码和用梯度下降法迭代求解该函数极值的代码,供读者参考。

2. 凸集与凸函数

凸集的定义:在实数域 R 上的向量空间中,如果集合 $\boldsymbol{S}$ 中任意两点的连线上的点都在 $\boldsymbol{S}$ 内,则称集合 $\boldsymbol{S}$ 为凸集。凸集和非凸集如图 3-7 所示。

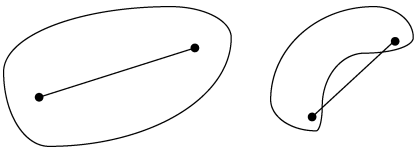


图 3-7　凸集和非凸集示意

设 $\boldsymbol{X}\in R^n$ 是一个凸集，当且仅当：

$$\alpha\boldsymbol{x}_1+(1-\alpha)\boldsymbol{x}_2\in\boldsymbol{X},\quad \forall\boldsymbol{x}_1,\boldsymbol{x}_2\in\boldsymbol{X},\quad \forall\alpha\in[0,1] \tag{3-35}$$

其中，$\boldsymbol{X}$ 是一个向量集合；$\boldsymbol{x}_1,\boldsymbol{x}_2$ 是集合 $\boldsymbol{X}$ 中的两个向量；α 位于$[0,1]$。如果一个集合 $\boldsymbol{X}$ 是凸集，则该集合中的任意两个点连成的线段上的任意一点也位于集合 $\boldsymbol{X}$ 中。

在欧氏空间中，凸集在直观上就是一个向四周凸起的图形。在一维空间中，凸集是一个点，或者一条连续的非曲线(线段、射线和直线)；在二维空间中，就是上凸的图形，如锥形扇面、圆、椭圆、凸多边形等；在三维空间中，凸集可以是一个实心的球体等。总之，凸集就是由向周边凸起的点构成的集合。

凸函数在凸子集上的定义[①]：凸函数是定义在某个向量空间的凸子集 C 上的实值函数 f，它在定义域 C 上的任意两点 x_1,x_2，以及任意 $\alpha\in[0,1]$，都有

$$f(\alpha x_1+(1-\alpha)x_2)\leqslant\alpha f(x_1)+(1-\alpha)f(x_2) \tag{3-36}$$

与凸函数定义相对的是凹函数，它是同样条件下满足下式的函数：

$$f(\alpha x_1+(1-\alpha)x_2)>\alpha f(x_1)+(1-\alpha)f(x_2) \tag{3-37}$$

直观上，凸函数曲线上任意两点连线上的点都在曲线的上方，即两个点的线性组合的函数值要小于等于两点函数值的线性组合，如图 3-8 所示。

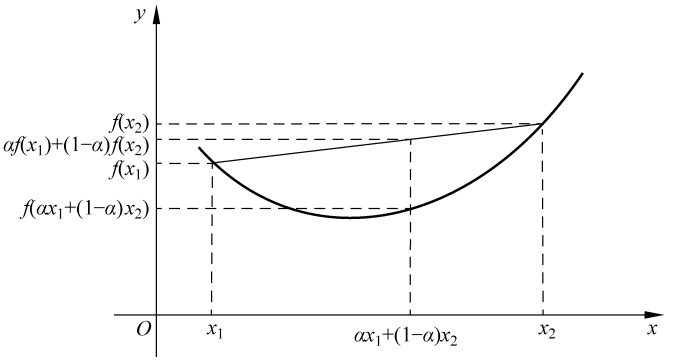


图 3-8　凸函数示意

3. 凸优化

在机器学习领域，凸函数最有价值的性质是它的局部最优点就是全局最优点。

假设凸函数上有一局部最优点 x^* 不是全局最优点，那么一定存在另一点 x'，使得

① 本书采用的定义是国际定义，与国内一些教材给的定义方向正相反，读者在阅读资料时要特别注意。

$f(x')\leqslant f(x^*)$。按式(3-36),令 $x=\alpha x'+(1-\alpha)x^*$,则有 $f(x)\leqslant\alpha f(x')+(1-\alpha)f(x^*)<f(x^*)$。当 $\alpha\to 0$ 时,$x\to x^*$,即 x 无限趋近于 x^*,即 x 是 x^* 邻域中的一点,但 $f(x)<f(x^*)$,这与 x^* 是局部最优点矛盾。因此,在凸函数中,局部最优点就是全局最优点。

针对凸函数的优化问题称为凸优化。机器学习中尽量使用凸函数作为损失函数。对于那些无法转换为凸函数的优化问题,要想找到全局最优解,就只能采用穷举法。

凸函数的判定方法:

(1) 设在凸集 $D\subseteq R^n$ 上 $f(x)$可微,则 $f(x)$在 D 上为凸函数的充要条件是对任意的 $x,y\in D$,都有:$f(y)\geqslant f(x)+\nabla f'(x)(y-x)$。该充要条件称为凸函数的一阶微分条件,对照图 3-8,实际上就是要求切线位于凸函数曲线的下方。

(2) 设在开凸集 $D\subseteq R^n$ 内,$f(x)$二阶可微,则 $f(x)$在 D 内为凸函数的充要条件是对任意的 $x\in D$,$f(x)$的 Hessian 矩阵半正定。该条件称为凸函数的二阶微分条件。对照图 3-8,实际上就是要求凸函数曲线的二阶导数大于等于 0。

仿射函数(最高次数为 1 的多项式函数)和线性函数(常数项为零的仿射函数)的二阶导数为 0,按判定方法,它们是凸函数。指数函数 e^x 的二阶导数大于 0,因此也为凸函数。在正实数域 R^+ 上的幂函数 x^α,当 $\alpha\geqslant 1$ 时为凸函数,$0<\alpha<1$ 时为凹函数。在 R^+ 上的对数函数 $\log_a x$,当 $a>1$ 时为凹函数,$0<a<1$ 时为凸函数。几何平均函数 $f(x)=\left(\prod_{i=1}^{n}x_i\right)^{\frac{1}{n}}$ 是 R^+ 上的凸函数。

对于严格的凹函数来说,只要取负值即可转化为凸函数。对于凸函数的函数,是否为凸函数的判定方法:

(1) 凸函数的非负线性组合是凸函数:$f_1,f_2,\cdots,f_k$ 是凸集 S 上的凸函数,那么 $\phi(x)=\sum_{i=1}^{k}\alpha_i f_i(x),\forall\alpha_i\geqslant 0(i=1,2,\cdots,k)$ 是凸函数。

(2) 凸函数的最大值函数是凸函数:$f_1,f_2,\cdots,f_k$ 是凸集 S 上的凸函数,那么 $\varphi(x)=\max f_i(x)(i=1,2,\cdots,k)$是凸函数。

更复杂的复合函数的凸性,读者可以在需要时查阅相关文献。

3.2.5 牛顿法

牛顿法最初是用来在实数域和复数域上近似求解方程的迭代方法,在机器学习领域,它用来求解使损失函数取得最小值时的参数。

1. 牛顿法的迭代与近似

前面讨论过,梯度下降法是用梯度来建立迭代关系式的迭代法,而牛顿法则是用切线来建立迭代关系的迭代法。

图 3-4 示意了梯度下降法的迭代过程,牛顿法的迭代过程与之相似,只不过是用切线与 x 轴的交点来作为下一轮迭代的起点,如图 3-9 所示。第一次迭代是从点 x_0 的值

$f(x_0)$开始，沿切线的相反方向一直前进到与 x 轴的交点 x_1 处。第二次迭代从点 x_1 的值 $f(x_1)$开始，前进到 $f(x_1)$处的切线与 x 轴的交点 x_2 处。如此持续进行，逐步逼近 x^* 点。牛顿法又叫切线法。

需要注意的是，牛顿法是用来求解方程的，因此在图 3-9 中，$f(x)$与 x 轴有交点 x^*，即存在使 $f(x)=0$ 的 x^*，这是应用牛顿法的前提。

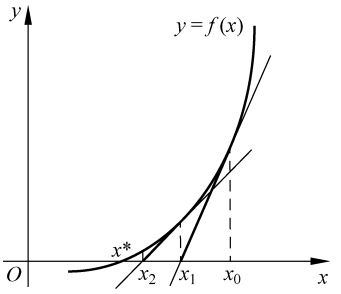


图 3-9　牛顿法示意(见彩插)

设非线性函数 $f(x)$在 x_0 点的导数为 $f'(x_0)$，则 $f(x)$在该点的切线为 $\hat{f}(x)=f(x_0)+(x-x_0)f'(x_0)$，设 $f'(x_0)\neq 0$，令 $\hat{f}(x)=0$，可解得它与 x 轴的交点 $x_1=x_0-f(x_0)/f'(x_0)$。

因此，牛顿法求解方程 $f(x)=0$ 的迭代关系式为

$$x_{i+1}=x_i-f(x_i)/f'(x_i) \tag{3-38}$$

上述求切线交点的过程，也可看作近似的过程。把 $f(x)$在 x_0 处展开成泰勒级数 $f(x)=f(x_0)+(x-x_0)f'(x_0)+\cdots$。在 x_0 附近取其线性部分 $\hat{f}(x)=f(x_0)+(x-x_0)f'(x_0)$作为 $f(x)$的近似，将 $\hat{f}(x)$与 x 轴的交点近似为 $f(x)$与 x 轴的交点。所以，牛顿法也是通过一次次近似来逼近方程的根的过程。

2. 牛顿法在机器学习中的应用

在机器学习领域，牛顿法常用来求解极值问题。牛顿法最初是为了求方程的根，所以并不能直接用来求极值。但是，函数极值的一阶导数为 0，因此，可以用牛顿法来求函数一阶导数为 0 的方程的根，得到极值点。对一元函数来说，迭代关系式为

$$x_{n+1}=x_n-f'(x_n)/f''(x_n) \tag{3-39}$$

下面来看如何得到上式。

对式(3-27)所示的 $f(x)$的泰勒展开式求一阶导数：

$$\begin{aligned}\frac{\mathrm{d}f(x)}{\mathrm{d}x}&=\frac{\mathrm{d}f(x_0)}{\mathrm{d}x}+\frac{\mathrm{d}}{\mathrm{d}x}f'(x_0)\Delta x+\frac{\mathrm{d}}{\mathrm{d}x}f''(x_0)\Delta x^2+\cdots\\&=f'(x_0)+2\Delta x f''(x_0)+\cdots\end{aligned} \tag{3-40}$$

对一阶导函数来说，同样取它的线性部分来近似，线性部分为：$\hat{f}(x)=f'(x_0)+2\Delta x f''(x_0)$，令其为 0，可得下一个迭代点：

$$x_1=x_0-f'(x_0)/f''(x_0) \tag{3-41}$$

对于二阶连续可导的二元函数 $f(\boldsymbol{x})$，其泰勒级数展开式的矩阵表示式为式(3-31)，对向量 $\boldsymbol{x}=(x^{(1)},x^{(2)})$求一阶导数：

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}f(\boldsymbol{x})&=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}f(\boldsymbol{x}_0)+\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\nabla f(\boldsymbol{x}_0)^{\mathrm{T}}\Delta\boldsymbol{x}+\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\left(\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}G(\boldsymbol{x}_0)\Delta\boldsymbol{x}\right)+\cdots\\&=\nabla f(\boldsymbol{x}_0)+\frac{1}{2}(G(\boldsymbol{x}_0)+G(\boldsymbol{x}_0)^{\mathrm{T}})\Delta\boldsymbol{x}+\cdots\\&=\nabla f(\boldsymbol{x}_0)+G(\boldsymbol{x}_0)\Delta\boldsymbol{x}+\cdots\end{aligned} \tag{3-42}$$

取线性部分近似，并令其为 0，可得下一个迭代点：

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \frac{\nabla f(\boldsymbol{x}_0)}{G(\boldsymbol{x}_0)} = \boldsymbol{x}_0 - G(\boldsymbol{x}_0)^{-1} \nabla f(\boldsymbol{x}_0) \tag{3-43}$$

对于高维函数，可类似推导。

牛顿法不仅利用了损失函数的一阶偏导数，还利用了损失函数的二阶偏导数，即梯度变化的趋势，因而比梯度下降法更全面地确定合适的搜索方向，具有二阶收敛速度。但牛顿法具有两个主要缺点，一是损失函数必须具有连续的一、二阶偏导数，Hessian 矩阵必须正定；二是计算更为复杂，不仅需要计算一阶偏导数，还需要计算二阶偏导数矩阵和它的逆矩阵，计算复杂度高。

3. 牛顿法求解线性回归问题

线性回归问题的损失函数的梯度如式(3-26)所示，计算代码见代码 3-4。

线性回归问题的损失函数的 Hessian 矩阵为

$$\begin{aligned} G(\boldsymbol{x}) &= \begin{bmatrix} \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(1)^2}} & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(1)} \partial w^{(2)}} & \cdots & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(1)} \partial w^{(n)}} \\ \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(2)} \partial w^{(1)}} & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(2)^2}} & \cdots & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(2)} \partial w^{(n)}} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(n)} \partial w^{(1)}} & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(n)} \partial w^{(2)}} & \cdots & \dfrac{\partial^2 L(\boldsymbol{W})}{\partial w^{(n)^2}} \end{bmatrix} = \left[\frac{\partial^2 L(\boldsymbol{W})}{\partial w^{(i)} \partial w^{(j)}} \right]_{n \times n} \\ &= \left[\frac{1}{2} \cdot \frac{\partial^2}{\partial w^{(i)} \partial w^{(j)}} \sum_{i=1}^{m} s_i^2(\boldsymbol{W}) \right]_{n \times n} \\ &= \left[\frac{1}{2} \sum_{i=1}^{m} \frac{\partial^2}{\partial w^{(i)} \partial w^{(j)}} s_i^2(\boldsymbol{W}) \right]_{n \times n} = \left[\sum_{i=1}^{m} x_i^{(k)} \cdot x_i^{(j)} \right]_{n \times n} \end{aligned} \tag{3-44}$$

可见线性回归损失函数的 Hessian 矩阵与标签值无关。

计算 Hessian 矩阵的代码如代码 3-7 所示。

代码 3-7　线性回归损失函数的 Hessian 矩阵(牛顿法.ipynb)

```
def hessian(x):
    '''计算 Hessian 矩阵
    para x:矩阵, 样本集
    return:矩阵, Hessian 矩阵
    '''
    m, n = np.shape(x)
    a = np.mat(np.zeros((n, n)))
    for i in range(m):
        xi_T = x[i, ].T
        xi = x[i, ]
        a += xi_T * xi
    return a
```

计算损失函数值的代码见代码 3-5。

牛顿法求解如表 3-1 所示的示例及输出如代码 3-8 所示。

代码 3-8 牛顿法主函数及应用示例(牛顿法.ipynb)

```
1. def newton(x, y, iterMax, delta):
2.     '''牛顿法
3.     para x:矩阵, 样本集
4.     para y:矩阵, 标签
5.     para iterMax: int,最大迭代次数
6.     para delta: float,函数值变化阈值,如迭代后函数值小于该值,则退出
7.     return: mat,回归系数'''
8.     n = np.shape(x)[1]
9.     w = np.mat(np.zeros((n,1)))
10.    step = 0
11.    loss = lossValue(x, y, w)
12.    print(str(step) + ":" + str(loss))
13.    while step <= iterMax:
14.        g = gradient(x, y, w)
15.        G = hessian(x)
16.        w = w - G.I * g
17.        newloss = lossValue(x, y, w)
18.        print(str(step + 1) + ":" + str(newloss))
19.        if loss - newloss < delta:
20.            break
21.        else:
22.            loss = newloss
23.        step += 1
24.    return w
25.
26. temperatures = [15, 20, 25, 30, 35,40]
27. flowers = [136, 140, 155, 160, 157, 175]
28. X = (np.mat([[1,1,1,1,1,1], temperatures])).T
29. Y = (np.mat(flowers)).T
30. w = newton(X, Y, 1000, 0.01)
31. print(w)
32. >>> 0:[[71497.5]]
33. >>> 1:[[53.40952381]]
34. >>> 2:[[53.40952381]]
35. >>>[[114.39047619]
36. >>>[  1.43428571]]
```

第 16 行完成迭代关系式(3-43)。

可见只需要 1 次迭代就得到解,这是因为损失函数为二次函数时,其泰勒级数的二次以上项都为 0,取其二次项与原目标函数不是近似,而是完全相同,Hessian 矩阵退化成一个常数矩阵。因此,只需要一步迭代即可达到极小点。

在 Scipy 的 optimize 包中包含了常用的优化计算工具。


视频

3.3 多项式回归

线性回归是用一条直线或者一个平面(超平面)去近似原始样本在空间中的分布。显然这种近似能力是有限的。非线性回归是用一条曲线或者曲面去逼近原始样本在空间中的分布,它"贴近"原始分布的能力一般较线性回归更强。

多项式是代数学中的基础概念,是由称为不定元的变量和称为系数的常数通过有限次加减法、乘法以及自然数幂次的乘方运算得到的代数表达式。

多项式回归(Polynomial Regression)是研究一个因变量与一个或多个自变量间多项式关系的回归分析方法。多项式回归模型是非线性回归模型中的一种。

由泰勒级数可知,在某点附近,如果函数 n 次可导,那么它可以用一个 n 次的多项式来近似。这种近似可以达到很高的精度。

进行多项式回归分析,首先要确定多项式的次数。次数一般是根据经验和实验确定。假设确定了用一个一元 n 次多项式来拟合训练样本集,模型可表示如下:

$$h(x)=\theta_0+\theta_1 x+\theta_2 x^2+\cdots+\theta_n x^n \tag{3-45}$$

那么多项式回归的任务就是估计出各 θ 值。可以采用均方误差作为损失函数,用梯度下降法求解,但难度较大,也难以确保得到全局解。

包括多项式回归问题在内的一些非线性回归问题可以转化为线性回归问题来求解,具体思路是将式中的每一项看作一个独立的特征(或者说生成新的特征),令 $y_1=x$,$y_2=x^2$,…,$y_n=x^n$,那么一个一元 n 次多项式 $\theta_0+\theta_1 x+\theta_2 x^2+\cdots+\theta_n x^n$ 就变成了一个 n 元一次多项式 $\theta_0+\theta_1 y_1+\theta_2 y_2+\cdots+\theta_n y_n$,就可以采用线性回归的方法来求解。

下面给出一个示例,该例子的基本过程是:先拟定一个一元三次多项式作为目标函数,然后再加上一些噪声产生样本集,再用转化的线性回归模型来完成拟合,最后对测试集进行预测。这个例子在随书资源的"多项式回归与欠拟合、过拟合.ipynb"文件中实现,采用 sklearn.linear_model 包中的 LinearRegression 函数来完成。

目标函数代码见代码 3-9。

代码 3-9 多项式回归示例中的目标函数代码(多项式回归与欠拟合、过拟合.ipynb)

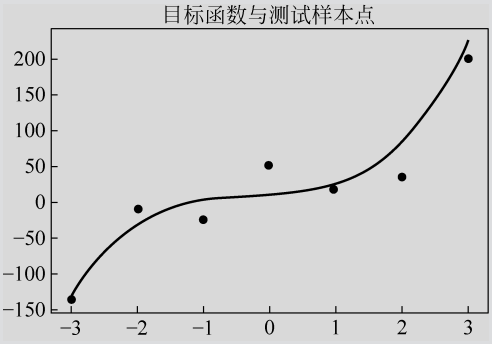
```
1. def myfun(x):
2.     '''目标函数
3.     input:x(float):自变量
4.     output:函数值'''
5.     return 10 + 5 * x + 4 * x**2 + 6 * x**3
```

产生样本集与测试集,并画出目标函数与样本点,见代码 3-10。

代码 3-10 多项式回归示例产生样本集与测试集(多项式回归与欠拟合、过拟合.ipynb)

```
1. import numpy as np
2. x = np.linspace(-3,3, 7)
3. x
```

```
>>> array([ -3., -2., -1.,  0.,  1.,  2.,  3.])
x_p = (np.linspace( -2.5, 2.5, 6)).reshape( -1,1)   #预测点
import random
y = myfun(x) + np.random.random(size = len(x)) * 100 - 50
y
>>> array([ -136.49570384,    -8.98763646,   -23.33764477,    50.97656894,
          20.19888523,    35.76052266,   199.48378741])
 %matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['axes.unicode_minus'] = False
plt.rc('font', family = 'SimHei', size = 13)
plt.title(u'目标函数与测试样本点')
plt.scatter(x, y, color = "green", linewidth = 2)
x1 = np.linspace( -3, 3, 100)
y0 = myfun(x1)
plt.plot(x1, y0, color = "red", linewidth = 1)
plt.show()
```

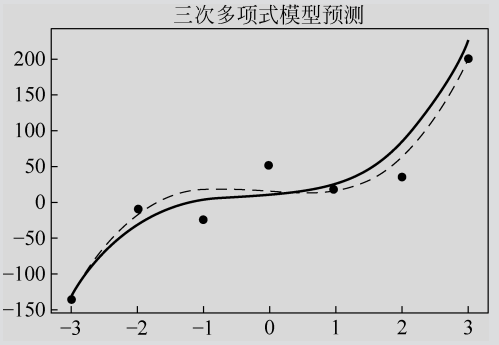


现在用三次多项式来拟合，见代码 3-11。

代码 3-11　三次多项式拟合示例（多项式回归与欠拟合、过拟合.ipynb）

```
from sklearn.preprocessing import PolynomialFeatures
featurizer_3 = PolynomialFeatures(degree = 3)
x_3 = featurizer_3.fit_transform(x)
x_3
>>> array([[  1.,   -3.,    9., -27.],
        [  1.,   -2.,    4.,   -8.],
        [  1.,   -1.,    1.,   -1.],
        [  1.,    0.,    0.,    0.],
        [  1.,    1.,    1.,    1.],
        [  1.,    2.,    4.,    8.],
        [  1.,    3.,    9.,   27.]])
x_p_3 = featurizer_3.transform(x_p)
x_p_3
>>> array([[  1.    ,   -2.5  ,    6.25 , -15.625],
        [  1.    ,   -1.5  ,    2.25 ,   -3.375],
```

```
16.         [  1.    ,  -0.5   ,   0.25 ,  -0.125],
17.         [  1.    ,   0.5   ,   0.25 ,   0.125],
18.         [  1.    ,   1.5   ,   2.25 ,   3.375],
19.         [  1.    ,   2.5   ,   6.25 ,  15.625]])
20. model_3 = LinearRegression()
21. model_3.fit(x_3, y)
22. print('-- 三次多项式模型 -- ')
23. print('训练集预测值与样本的误差均方值: ' +
    str(np.mean((model_3.predict(x_3) - y) ** 2)))
24. print('测试集预测值与目标函数值的误差均方值: ' +
    str(np.mean((model_3.predict(x_p_3) - myfun(x_p)) ** 2)))
25. print('系数: ' + str(model_3.coef_))
26. >>> -- 三次多项式模型 --
27. >>>训练集预测值与样本的误差均方值: 534.1920527426208
28. >>>测试集预测值与目标函数值的误差均方值: 247.2068856878784
29. >>>系数: [[ 0.          -7.4139024   1.43393358  6.88041117]]
30.
31. plt.title(u'三次多项式模型预测')
32. plt.scatter(x, y, color = "green", linewidth = 2)
33. plt.plot(x1, y0, color = "red", linewidth = 1)
34. #y1 = model.predict(x1)
35. #plt.plot(x1, y1, color = "black", linewidth = 1)
36. y3 = model_3.predict(featurizer_3.fit_transform(x1))
37. plt.plot(x1, y3, "b--", linewidth = 1)
38. plt.show()
```



39.

第 2～11 行用来生成样本的新特征，使用 PolynomialFeatures 类按 $y_0=x^0$，$y_1=x^1$，$y_2=x^2$，$y_3=x^3$ 生成新的特征值，第 12～19 行是生成预测点的新特征。第 20、21 行是用 LinearRegression 对新特征集进行线性回归，第 29 行给出了新的一元三次多项式的系数。预测图中，实线为目标函数，虚线表示学习得到的模型在连续各点的预测值。

转化为线性问题来求解，是处理非线性问题的常用方法，如指数函数 $h(t)=\alpha \cdot e^{\beta t}$ 通过两边取自然对数，得到 $\ln h(t)=\beta t+\ln\alpha$，可转化为线性回归问题。

3.4 过拟合与泛化

过拟合与泛化是机器学习中非常重要的概念，也是必须面对的基本问题。本小节先从多项式回归的讨论中引入过拟合、欠拟合和泛化的概念，然后从工程角度和算法角度讨论常用处理方法。

3.4.1 欠拟合、过拟合与泛化能力

在多项式回归的示例中，训练样本集是以一元三次多项式为基础加上噪声产生的，然后以一个待定系数的一元三次多项式去逼近。

能够求解问题的模型往往不止一个，不同模型往往有复杂度上的区别。多项式回归示例中，还可以分别用一元一次线性式、一元五次多项式和一元九次多项式去逼近，它们的复杂度越来越高，效果如图 3-10 所示。实现代码见随书资源的“多项式回归与欠拟合、过拟合.ipynb”文件。

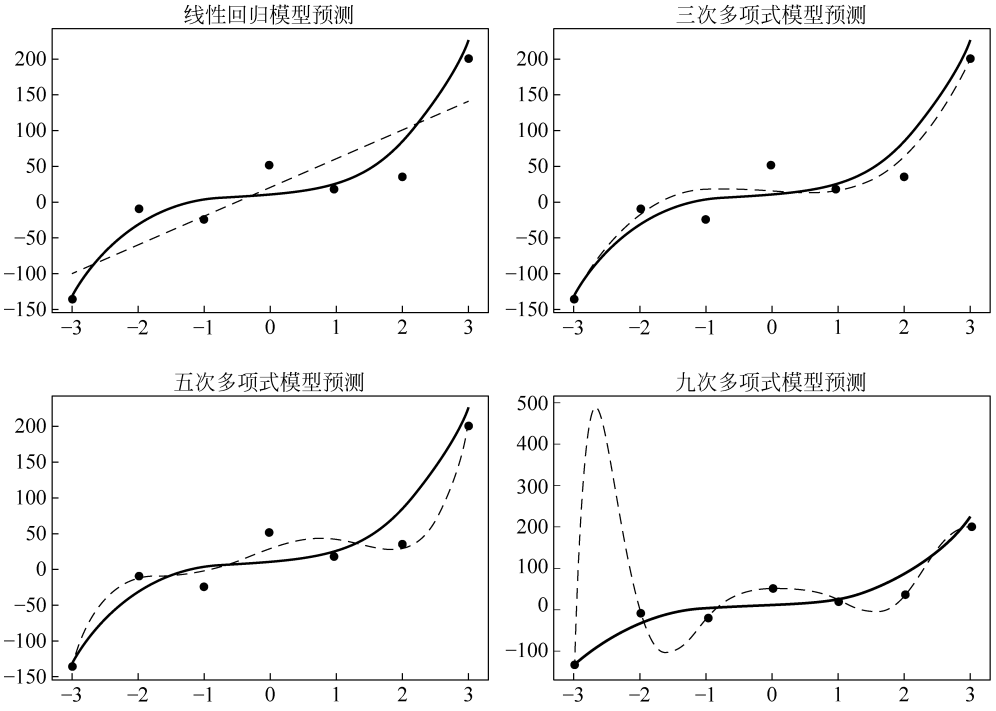


图 3-10 不同次多项式拟合效果示意(见彩插)

结果显示以三次多项式来逼近样本，可以取得最好的效果。

最简单的线性模型，它是用一条直线来逼近各个样本点，显然是力不从心，这种现象称为“欠拟合”(Under-fitting)。欠拟合模型是由于模型复杂度不够、训练样本集容量不够、特征数量不够、抽样分布不均衡等原因引起的不能学习出样本集中蕴含知识的模型。

欠拟合问题较容易处理，如增加模型复杂度、增加训练样本、提取更多特征等。

五次多项式的逼近，它比三次多项式更加接近样本点，但是与实线表示的目标函数已经产生背离。九次多项式能一一穿过所有样本点，可是它已经严重背离目标函数了，虚线与实线的变化趋势显得面目全非。这说明在某些情况下，越复杂的模型越能逼近样本点，但也越背离作为目标的三次多项式函数。这样的模型在训练集上表现很好，而在测试集上表现很差，这种现象称为“过拟合”(Over-fitting)。产生过拟合的原因是模型过于复杂，以至于学习太过了，把噪声的特征也学习进去了。

模型在训练样本上产生的误差叫训练误差(Training Error)，它是模型对训练样本的预测值与样本标签之间的误差。同样，在测试样本上产生的误差叫测试误差(Test Error)。在示例中，采用均方误差作为损失函数，因此，样本误差就是所有训练样本的误差平方的均值。同样，测试误差是所有测试样本的误差平方的均值。表 3-2 展示了例子中各模型的训练误差和测试误差及它们的和。

表 3-2　不同次多项式拟合的训练误差和测试误差

	线性回归模型	三次多项式模型	五次多项式模型	九次多项式模型
训练误差	2019	534	209	4
测试误差	578	247	1232	38 492
和	2597	781	1441	38 496

可以看出，随着次数的增加，拟合模型越来越复杂，训练误差越来越小，而测试误差先是减少，但随后会急剧增加。

衡量模型好坏的是测试误差，它标志了模型对测试样本的预测能力，因此一般追求的是测试误差最小的那个模型。模型对测试样本的预测能力称为泛化能力(Generalization Ability)，模型在测试样本上的误差称为泛化误差(Generalization Error)。“泛化”一词源于心理学，它是指某种刺激产生一定条件反应后，其他类似的刺激也能产生某种程度的同样反应。

关于泛化能力和模型复杂程度之间的经验关系如图 3-11 所示。

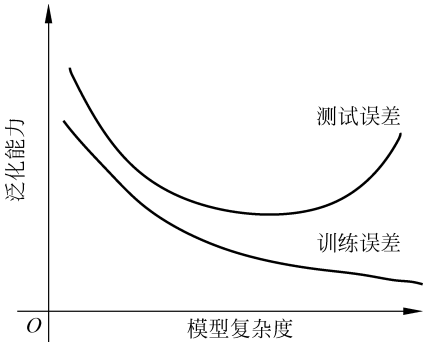


图 3-11　泛化能力与模型复杂度之间的关系示意

一般来说，只有合适复杂程度的模型才能最好地反映出训练集中蕴含的规律，取得最好的泛化能力。

3.4.2　泛化能力评估方法

如何来证明训练好的模型具有好的泛化能力呢？容易想到，可以在实际应用中通过观测模型对测试样本的预测效果来判断，但这种办法无法及时得到反馈结果用于分析改进，不符合实用要求。

在监督学习任务中，工程上经常采用将已有样本集划分为训练集和验证集的方法，用训练集来训练模型，用验证集来检验模型，达到足够好的效果后，再提交实际应用。

这么做的依据是什么呢？

机器学习是基于这样一个假设：已有训练数据和未知测试数据蕴含着相同规律。如果两者的规律不同，那么就不能从前者的数据中找到适合后者的规律，那么机器学习是无能为力的。同样的，将训练数据划分为训练集和验证集，也是基于这样的假设，即训练集蕴含的规律与验证集中蕴含的规律也是一致的，因此，可以用训练集来训练模型，用验证集来验证模型，达到希望的效果后，再用来预测测试集。如果把这个规律简化成二维平面上的分布区域，则可以将该过程形象示意如图 3-12 所示。图中圆点表示正样本，三角形表示负样本，正方形表示噪声。

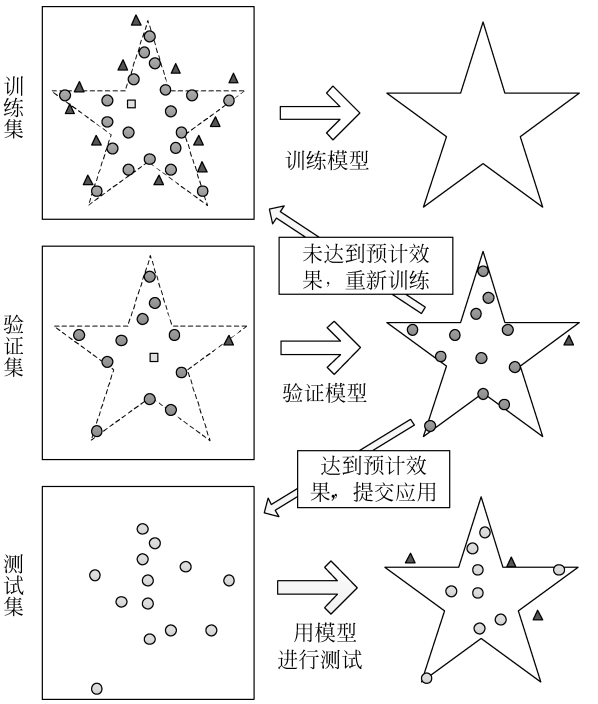


图 3-12　划分数据集的训练过程(见彩插)

好的算法模型能够学习出训练集蕴含的规律，在图 3-12 中表示为五角星的区域分布。验证集中的样本用来验证这个五角星的区域分布是否合理。验证集中的样本的真实分布是已知的，因此，可以用真实分布情况来比对预测分布情况，这样就可以判断出训练出来的模型的效果。达到要求后，才能将该模型投入实际应用。

对训练集和验证集的样本有什么要求呢?

首先,训练集的数据要尽可能充分且分布平衡,并符合一定的清洁度要求(即噪声不能过多)。不充分或者分布不平衡的样本集,训练不出一个完整的模型,如图 3-13 所示。而噪声过多的样本,则可能训练不出反映原来规律的模型,如图 3-14 所示。

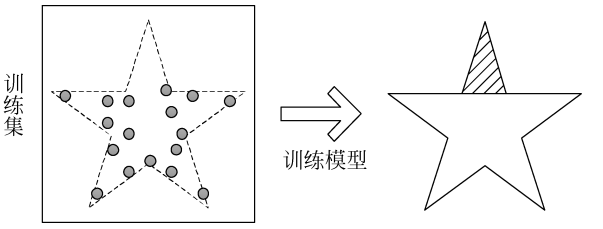


图 3-13 不充分或分布不平衡的样本集训练效果(见彩插)

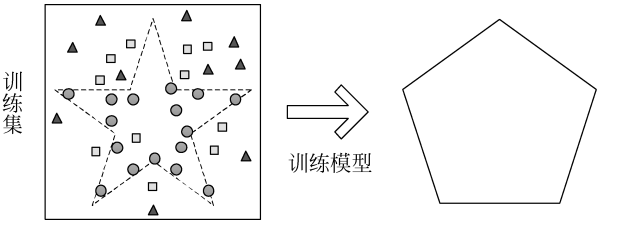


图 3-14 噪声过多的样本集训练效果(见彩插)

其次,验证集的样本也需要符合一定的分布平衡和清洁度要求,否则将无法验证出一个真实的模型。在验证集样本过少,或者分布不平衡的情况下,有可能无法验证出如图 3-13 所示的模型的缺陷。

此外,训练模型和验证模型的样本不能相同,否则训练出的模型的验证结果非常完美,而实际上并不一定可用。

将训练数据划分为训练集和验证集的方法称为保持法(Holdout Method),一般保留已知样本的 20%~30%作为验证集。如果数据分布合理,验证集产生的验证误差通常会接近测试集产生的测试误差。

除了保持法,还经常采用一种称为 k-折交叉验证(k-fold Cross-validation)的评估模型预测效果的方法。k-折交叉验证是将总样本集随机地划分为 k 个互不相交的子集。对于每个子集,将所有其他样本集作为训练集训练出模型,将该子集作为验证集,并记录验证集每一个样本的预测结果。每个子集都这样处理完后,所有样本都有一个预测值。然后与真实值进行比对,从而评估模型的效果。这个方法将每一个样本都用来进行了验证,其评估的准确性一般要高于保持法。

一般来说,划分的子集越多,k-折交叉验证评估的效果就越好,但训练耗费的时间就很长。如果训练耗时不是问题时,可以采用单一保留(Leave-one-out)交叉验证,即每个验证集只有一个样本,其余全是训练集。

对于有时间顺序的样本集,即样本产生有时间先后关系,则不能随机划分验证集和训练集,不能用后产生的样本集来预测先发生的样本。

3.4.3 过拟合抑制

在算法研究中，解决过拟合时，常提到“奥卡姆剃刀(Occam's Razor)定律”，它是由14世纪逻辑学家奥卡姆提出的。这个定律称为“如无必要，勿增实体”，即“简单有效原理”。在模型选择中，就是在所有可以选择的模型中，能够很好地解释已知数据并且简单的模型才是最好的模型。基于这个思路，在算法研究中，人们常采用正则化(Regularization)、早停(Early Stopping)、随机失活(Dropout)等方法来抑制过拟合。

1. 正则化方法

正则化方法是在样本集的损失函数中增加一个正则化项(Regularizer)，或者称罚项(Penalty Term)，来对冲模型的复杂度。正则化项一般是模型复杂度的单调递增函数，模型越复杂，正则化值就越大。

正则化方法的优化目标为

$$\min_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^{m}L(y_i,f(x_i))+\lambda J(f) \tag{3-46}$$

其中，f 代表某一模型；$\mathcal{F}$是可选模型的集合；L 是损失函数。第一项 $\frac{1}{m}\sum_{i=1}^{m}L(y_i,f(x_i))$ 是训练集上的平均损失，即训练误差，又称为经验风险(Empirical Risk)。第二项 $J(f)$是正则化项，$\lambda\geqslant 0$ 为正则化项的权重系数。正则化项可以取不同的形式的函数，但其值必须满足模型越复杂值越大的要求。

经验风险加上正则化项，称为结构风险(Structural Risk)。显然经验风险只刻画了模型对样本集的适应能力，而结构风险不仅考虑了对样本集的适应能力，还考虑了模型的复杂度，因此，正则化方法追求的是结构风险最小化，而不仅仅是经验风险最小化[12]。

常用的正则化方法有L1、L2正则化方法。L1、L2正则化方法指的是正则化项是模型参数向量 W 的L1和L2范数。

(1) L2正则化方法。

设原始损失函数是 L_0，给它加一个参数向量 W 的L2范数，得到新的损失函数为

$$L=L_0+\frac{\lambda}{2k}\sum_j(w^{(j)})^2 \tag{3-47}$$

其中，λ 是正则化项的权重系数。k 是参数总数，它在一个模型中是一个常量，也可以不除，乘1/2是为了求导后消除常数2。

这个L2正则项是怎么来抑制过拟合的呢？来看看在梯度下降法中的作用。对式(3-47)求特征 $w^{(j)}$ 的导数：

$$\frac{\partial L}{\partial w^{(j)}}=\frac{\partial L_0}{\partial w^{(j)}}+\frac{\partial}{\partial w^{(j)}}\left(\frac{\lambda}{2k}\sum_j(w^{(j)})^2\right)=\frac{\partial L_0}{\partial w^{(j)}}+\frac{\lambda}{k}w^{(j)} \tag{3-48}$$

迭代公式(3-23)的分量变为

$$w_{i+1}^{(j)}=w_i^{(j)}-\alpha\frac{\partial L}{\partial w_i^{(j)}}=w_i^{(j)}-\alpha\frac{\partial L_0}{\partial w_i^{(j)}}-\alpha\frac{\lambda}{k}w_i^{(j)}$$

$$=\left(1-\frac{\alpha\lambda}{k}\right)w_i^{(j)}-\alpha\frac{\partial L_0}{\partial w_i^{(j)}} \tag{3-49}$$

可见,使用 L2 正则项后,$w_i^{(j)}$ 的系数小于 1 了,因此,将使得 $w_{i+1}^{(j)}$ 较原来的变化要小一些,这个方法也叫权重衰减(Weight Decay)。来看看多项式回归例子中 $w^{(j)}$ 小而拟合好的情况。代码 3-11 中,第 25 行是打印出模型的系数,将线性模型、三次模型、五次模型和九次模型的系数都列出来:

线性模型系数:40.74897579;

三次模型系数:0,-7.4139024,1.43393358,6.88041117;

五次模型系数:0,31.53983182,-9.59767085,-11.33268976,1.15255569,1.56112294;

九次模型系数:-1.55175872e-12,9.86092386e+00,-3.85815674e+01,8.93592424e+00,-2.49195458e+01,5.70545419e+00,1.19222564e+01,-2.99067031e+00,-9.67091926e-01,2.56633014e-01。

可以发现,三次多项式模型的系数最小,九次多项式模型的系数出现了很大的值。从二者的拟合图 3-10 中,可以进一步理解这种情况。因为样本点是带噪声的,因此要完美穿过所有点,那么拟合后的曲线必定要有很多急剧变化的弯才行。急剧变化则意味某些斜率很大,也就是导数很大,因此系数也必然变大。

(2) L1 正则化方法。

L1 正则化方法为

$$L=L_0+\frac{\lambda}{k}\sum_j|w^{(j)}| \tag{3-50}$$

上式对 $w^{(j)}$ 求导:

$$\frac{\partial L}{\partial w^{(j)}}=\frac{\partial L_0}{\partial w^{(j)}}+\frac{\lambda}{k}\mathrm{sgn}(w^{(j)}) \tag{3-51}$$

sgn(·)是符号函数:

$$\mathrm{sgn}(x)=\begin{cases}+1, & x\geqslant 0\\ -1, & x<0\end{cases} \tag{3-52}$$

于是梯度下降法的迭代式为

$$w_{i+1}^{(j)}=w_i^{(j)}-\frac{\alpha\lambda}{k}\mathrm{sgn}(w_i^{(j)})-\alpha\frac{\partial L_0}{\partial w_i^{(j)}} \tag{3-53}$$

第二项中符号函数的作用是不管 $w_i^{(j)}$ 是正还是负,都使之往 0 靠近,也就相当于减小了模型的复杂度,防止过拟合。在实际应用时,可令 sgn(0)=0 解决不可导的问题。

采用 L2 范数和 L1 范数正则项的线性回归,分别称为岭回归和 lasso 回归。后文将详细讨论岭回归。

2. 早停法

早停法是在模型迭代训练中，在模型对训练样本集收敛之前就停止迭代以防止过拟合的方法。

前面讨论过，模型泛化能力评估的思路是将样本集划分为训练集和验证集，用训练集来训练模型，训练完成后，用验证集来验证模型的泛化能力。而早停法提前引入验证集来验证模型的泛化能力，即在每一轮训练（一轮是指遍历所有训练样本一次）完后，就用验证集来验证泛化能力，如果 n 轮训练都没有使泛化能力得到提高，就停止训练。n 是根据经验提前设定的参数，常取 10、20、30 等值。这种策略称为“No-improvement-in-n”。

3. 随机失活

随机失活只应用于人工神经网络的过拟合抑制，它通过随机使一部分神经元临时失效来达到目的。关于随机失活的内容将在后文结合具体神经网络进行讨论。

在工程方面，可以从样本集数据方面采取措施来防止过拟合，包括数据清洗（Data Cleaning）和数据扩增（Data Augmentation）等。数据清洗是指尽量清除掉噪声，以减少对模型的影响。数据扩增是指增加训练样本来抵消噪声的影响，从而抑制过拟合。增加训练样本包括从数据源采集更多的样本和人工制造训练样本两种方法。在人工制造训练样本时，要注意制造的样本要和已有样本是近似独立同分布的。

3.5 向量相关性与岭回归


视频

最小二乘法求解线性回归模型时，其基本公式为 $\hat{\boldsymbol{W}}^{\mathrm{T}}=(\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}})^{-1}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}}$。式中有一个矩阵逆运算 $(\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}})^{-1}$，也就是说 $\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}$ 必须可逆，否则将无法求解。本小节讨论用岭回归算法来处理此类情况。岭回归算法还可以处理特征相关的问题。岭回归算法实际上是加了 L2 正则项的线性回归。

岭回归涉及特征向量之间的相关性，所以，先介绍机器学习中非常重要的向量的相关性度量。

3.5.1 向量的相关性

向量的相关性度量在机器学习中有重要的应用，比如，可以用验证集的预测值组成的向量与实际标签值组成的向量之间的相关性来衡量算法的有效性。样本可表示为由特征依序组成的向量，因此可以用向量的相关性比较两个样本的相似程度。如果把所有样本的指定特征依序排列成向量，就能用向量的相关性来比较两个特征的相似程度。

1. 协方差

向量 $\boldsymbol{X}$ 和向量 $\boldsymbol{Y}$ 的协方差为

$$\mathrm{Cov}(\boldsymbol{X},\boldsymbol{Y})=E\{[\boldsymbol{X}-E(\boldsymbol{X})][\boldsymbol{Y}-E(\boldsymbol{Y})]\} \tag{3-54}$$

协方差反映的是两个向量的变化趋势。当变化趋势相近时,协方差大于0;如果相反,则小于0;如果完全无关,即独立时,为0。在样本集中,如果某特征向量与标签向量协方差为0,则意味着该特征对预测没有帮助,可以去掉,以减少计算量。

2. 相关系数

向量 $\boldsymbol{X}$ 和向量 $\boldsymbol{Y}$ 的相关系数为

$$\rho_{XY}=\frac{\mathrm{Cov}(\boldsymbol{X},\boldsymbol{Y})}{\sqrt{D(\boldsymbol{X})}\sqrt{D(\boldsymbol{Y})}}=\frac{E\{[\boldsymbol{X}-E(\boldsymbol{X})][\boldsymbol{Y}-E(\boldsymbol{Y})]\}}{\sqrt{D(\boldsymbol{X})}\sqrt{D(\boldsymbol{Y})}} \tag{3-55}$$

相关系数也反映了两个向量的变化趋势。由于它做了标准化处理,消除了两个变量变化幅度的影响,因此更适合实际应用。

3. 相关距离

向量 $\boldsymbol{X}$ 和向量 $\boldsymbol{Y}$ 的相关距离为

$$D_{\boldsymbol{XY}}=1-\rho_{\boldsymbol{XY}} \tag{3-56}$$

表3-1所示的温度和小花数量所组成的温度向量和小花数量向量的协方差为125.5,相关系数0.945,说明两者之间有很强的相关性。如果表3-1中再增加湿度值,如表3-3所示。

表3-3 线性回归示例温度、湿度和小花数量

温度/℃	15	20	25	30	35	40
湿度/%	25	35	45	55	65	75
小花数量/朵	136	140	155	160	157	175

计算得到温度向量与湿度向量的相关系数为1.0,湿度向量与小花数量向量的相关系数为0.945。可见湿度向量与温度向量是完全相关的,因此,它与小花数量向量的相关系数等于温度向量与小花数量向量的相关系数。以上值的计算如代码3-12所示。

代码3-12 向量相关性示例(向量相关性度量.ipynb)

```
import numpy as np
temperatures = [15, 20, 25, 30, 35,40]
t = np.array(temperatures)
flowers = [136, 140, 155, 160, 157, 175]
f = np.array(flowers)
np.cov(t,f)
>>> array([[  87.5        ,  125.5        ],
        [ 125.5        ,  201.36666667]])'''
np.corrcoef(t,f)
>>> array([[ 1.         ,  0.94546598],
        [ 0.94546598,  1.         ]])
humiditys = [0.25, 0.35, 0.45, 0.55, 0.65, 0.75]
h = np.array(humiditys)
np.cov(t,h)'''
```

```
>>> array([[  8.75000000e+01,   1.75000000e+00],
        [  1.75000000e+00,   3.50000000e-02]])
np.corrcoef(t,h)
>>> array([[ 1.,  1.],
        [ 1.,  1.]])
np.corrcoef(h,f)
>>> array([[ 1.        ,  0.94546598],
        [ 0.94546598,  1.        ]])
```

协方差和相关系数的计算使用了 numpy 包中的 cov 和 corrcoef 函数。

3.5.2 岭回归算法

如前文所述，在线性回归求解中，$\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}$ 必须可逆才能应用最小二乘法，即样本矩阵的行列式不能为 0，或者说是列满秩。因此，样本矩阵的各个特征向量之间不能有强烈的线性相关性，即相关系数最好是接近于 0。如果两个特征列有强烈的相关性，那么会出现两列系数不固定的情况，即在稳定标签值的情况下，其中一列的系数的升高可以通过另一列系数的降低来弥补，因此会出现差异较大的模型，也称为模型的方差较大。代码 3-12 计算了表 3-3 中湿度特征向量和温度特征向量的相关系数为 1，是强相关的。应用最小二乘法来计算表 3-3 所代表的线性回归模型：

$$f(\boldsymbol{x})=w^{(2)}\cdot x^{(2)}+w^{(1)}\cdot x^{(1)}+b \tag{3-57}$$

其中，$x^{(2)}$ 表示湿度值；$x^{(1)}$ 表示温度值。

计算得到 $w^{(2)}$，$w^{(1)}$ 和 b 分别为 325.21368214，−24.40422285，−232.07635727，对温度和湿度组成的测试样本(18，0.31)，(22，0.39)，(33，0.61)进行预测得到小花数量为：−186，−302，−621。该结果显然是不合理的。计算过程见附属资源中的文件“向量相关性度量.ipynb”。

岭回归算法是在原线性回归的损失函数 $L(\boldsymbol{W})$[式(3-9)]上增加 L2 正则项 $\lambda\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}$（$\lambda$ 称为岭系数，是事先指定的参数）得到新损失函数 L'：

$$L'=L(\boldsymbol{W})+\lambda\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}=(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}}+\lambda\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}} \tag{3-58}$$

此时，要求出的参数是：

$$\hat{\boldsymbol{W}}=\arg\min_{\boldsymbol{W}}L'(\boldsymbol{W})=\arg\min_{\boldsymbol{W}}[(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}}+\lambda\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}] \tag{3-59}$$

最小二乘法求解，对 $\boldsymbol{W}^{\mathrm{T}}$ 求导：

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}L'&=\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{W}^{\mathrm{T}}}[(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})(\boldsymbol{Y}-\boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}}+\lambda\boldsymbol{W}\boldsymbol{W}^{\mathrm{T}}]\\&=2\bar{\boldsymbol{X}}(\bar{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}-\boldsymbol{Y}^{\mathrm{T}})+2\lambda\boldsymbol{W}^{\mathrm{T}}\end{aligned} \tag{3-60}$$

然后令其为 0，可得

$$\hat{\boldsymbol{W}}^{\mathrm{T}}=(\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}+\lambda\boldsymbol{I})^{-1}\bar{\boldsymbol{X}}\boldsymbol{Y}^{\mathrm{T}} \tag{3-61}$$

岭回归算法在最小二乘法中，实际上是给 $\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}$ 加一个非负因子 $\lambda\boldsymbol{I}$（$\boldsymbol{I}$ 为单位矩阵），使得 $\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^{\mathrm{T}}+\lambda\boldsymbol{I}$ 列满秩。这样，通过加入一个人为的干扰，虽然使估计成了有偏的了（即

新模型预测值与真实值之间有差异,也称为偏差),但是成了列满秩,可以求得逆矩阵,从而提高了稳定性,即减少了方差。因为单位矩阵的对角线上有一条由 1 组成的"岭",其他元素为 0,所以把这个方法称为岭回归。

用 Python 的 numpy 包,容易实现式(3-61),并对表 3-3 所示的样本数据进行求解,令 $\lambda=0.5$ 求得 $w^{(2)}$,$w^{(1)}$ 和 b 分别为:95.97627991,2.13220123,−4.75616997。对温度和湿度组成的测试(18,0.31),(22,0.39),(33,0.61)进行预测得到小花数量为 132,141,163。该结果虽然产生了一定的偏差,但显然要合理得多。

容易得到岭回归在梯度下降法中的迭代关系式:

$$w_{l+1}^{(j)}=(1-2\lambda)w_l^{(j)}-\alpha\sum_{i=1}^{m}\left(y_i-\sum_{k=0}^{n}x_i^{(k)}\cdot w_l^{(k)}\right)x_i^{(j)},\quad \text{对每一个特征 } j \tag{3-62}$$

通过权重对系数进行了衰减,抑制了相关特征的系数发生过大变化,迫使它们向 0 趋近,与简单的线性回归相比,能取得更好的预测效果。

关于岭参数 λ 的确定,需要使用试错法,通过验证集的预测误差最小化来得到,即对不同的 λ 进行多次实验,取使验证集预测误差最小的那个 λ 作为最终的参数值。

3.6 局部回归

前述的回归模型,假设所有样本之间都存在相同程度的影响,这类模型称为全局模型。在机器学习中,还有另一种思想:认为相近的样本相互影响更大,离得远的样本相互影响很小,甚至可以不计。这种以"远亲不如近邻"思想为指导得到的模型称为局部模型。局部思想在聚类、回归、分类等机器学习任务中都有应用,聚类算法中的 DBSCAN 算法就是以这种思想为指导的模型。

用于回归的局部模型有局部加权线性回归模型、K 近邻模型和树回归模型等。树模型主要用于分类,因此树回归模型将与树分类模型一并在后面第 4 章中讨论。

3.6.1 局部加权线性回归

局部加权线性回归(Locally Weighted Linear Regression,LWLR)模型根据训练样本点与预测点的远近设立权重,离预测点越近的点的权重就越大。

设 q_i 为给样本 $\boldsymbol{S}_i$ 设立的权值,那么样本 $\boldsymbol{S}_i$ 的误差平方 $e_i^2(W)$ 为

$$e_i^2(\boldsymbol{W})=q_i(y_i-\boldsymbol{W}\cdot\boldsymbol{x}_i)^2=(y_i-\boldsymbol{W}\cdot\boldsymbol{x}_i)q_i(y_i-\boldsymbol{W}\cdot\boldsymbol{x}_i)^{\mathrm{T}} \tag{3-63}$$

其中,$\boldsymbol{x}_i$ 是样本 i 的实例;y_i 是样本 i 的标签值。

所有样本的误差平方和,即损失函数为

$$\begin{aligned}L(\boldsymbol{W})&=e_1^2+e_2^2+\cdots+e_m^2\\&=(y_1-\boldsymbol{W}\cdot\boldsymbol{x}_1\quad y_2-\boldsymbol{W}\cdot\boldsymbol{x}_2\quad\cdots\quad y_m-\boldsymbol{W}\cdot\boldsymbol{x}_m)\begin{pmatrix}q_1&\cdots&0\\\vdots&\ddots&\vdots\\0&\cdots&q_m\end{pmatrix}\end{aligned}$$

$$\begin{pmatrix} y_1 - \boldsymbol{W} \cdot \boldsymbol{x}_1 \\ y_2 - \boldsymbol{W} \cdot \boldsymbol{x}_2 \\ \vdots \\ y_m - \boldsymbol{W} \cdot \boldsymbol{x}_m \end{pmatrix} = (\boldsymbol{Y} - \boldsymbol{W}\bar{\boldsymbol{X}})\boldsymbol{Q}(\boldsymbol{Y} - \boldsymbol{W}\bar{\boldsymbol{X}})^{\mathrm{T}} \tag{3-64}$$

其中，$\boldsymbol{Q} = \begin{pmatrix} q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & q_m \end{pmatrix}$。

用最小二乘法求解，可得

$$\hat{\boldsymbol{W}}^{\mathrm{T}} = (\bar{\boldsymbol{X}}\boldsymbol{Q}\bar{\boldsymbol{X}}^{\mathrm{T}})^{-1}\bar{\boldsymbol{X}}\boldsymbol{Q}\boldsymbol{Y}^{\mathrm{T}} \tag{3-65}$$

那么 q_i 是怎么设立的呢？LWLR 采用核函数来设立 q_i 的值，下面介绍一种常用的高斯核函数，其设立权重的公式为

$$q_i = \mathrm{e}^{\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x})^{\mathrm{T}}(\boldsymbol{x}_i - \boldsymbol{x})}{2\tau^2}\right)} \tag{3-66}$$

其中，$\boldsymbol{x}_i$ 为第 i 个样本的实例；$\boldsymbol{x}$ 为测试样本的实例；τ 为预设的参数。因为指数部分为非正数，所以权值最大为 1.0，只有在点 $\boldsymbol{x}_i$ 与预测点 $\boldsymbol{x}$ 重合时才出现。点 $\boldsymbol{x}_i$ 与点 $\boldsymbol{x}$ 距离越大，权值越小，向 0 趋近。参数 τ 控制了权值变化的速率，取值越大，权值从中心点向两边降低的速率越慢。

实现式(3-65)的最小二乘法解局部加权线性回归的函数如代码 3-13 所示。

代码 3-13　最小二乘法求解局部加权线性回归(局部加权线性回归.ipynb)

```
def lwlr(t,X,Y,k = 1.0):
    '''最小二乘法求解局部加权线性回归
    para t: 矩阵,测试样本
    para X:矩阵,样本特征矩阵
    para Y:矩阵,标签
    para k: 核函数系数
    return: 预测值
    '''
    m = np.shape(X)[1]
    Q = np.mat(np.eye((m)))
    for i in range(m):
        diffX = X[:,i] - t
        Q[i,i] = np.exp( - diffX.T * diffX/(2.0 * k ** 2))
    w = (X * Q * X.T).I * X * Q * Y.T
    return w * t
```

用 LWLR 来拟合多项式回归(3.3 节)中的示例，参数 τ 分别取 5.0，1.0，0.5，0.05，画出预测曲线如图 3-15 所示。可见随着核函数的参数从大到小，模型由欠拟合变为过拟合。

局部加权线性回归方法不形成固定的模型，对每一个新的预测点，都需要计算每个样本点的权值，在样本集非常大的时候，预测效率较低。

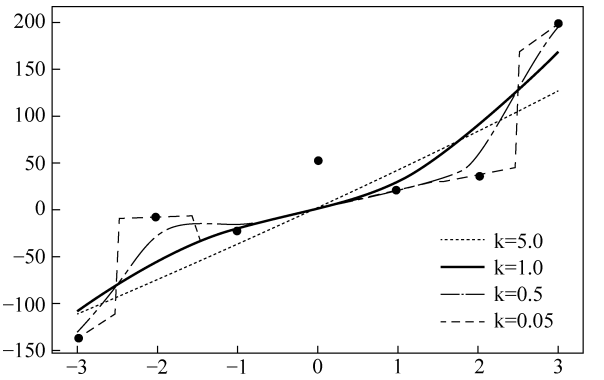


图 3-15　不同核函数参数时的局部加权线性回归预测曲线(见彩插)

3.6.2　K 近邻法

K 近邻法(K-Nearest Neighbor, KNN)是一种简单而基本的机器学习方法,可用于求解分类和回归问题。K 近邻法于 1968 年由 Cover 和 Hart 的提出。

应用 K 近邻法求解回归问题,需要先指定三个要素:样本间距离度量方法 $d(\cdot)$、邻居样本个数 k 和根据 k 个邻居样本计算标签值方法 $v(\cdot)$。

设样本集为 $\boldsymbol{S}=\{\boldsymbol{s}_1,\boldsymbol{s}_2,\cdots,\boldsymbol{s}_m\}$ 包含 m 个样本,每个样本 $\boldsymbol{s}_i=(\boldsymbol{x}_i,y_i)$ 包括一个实例 $\boldsymbol{x}_i$ 和一个实数标签值 y_i。测试样本记为 $\boldsymbol{x}$。

K 近邻法用于回归分为以下两步:

(1) 根据 $d(\cdot)$,从 $\boldsymbol{S}$ 中找出 k 个距离 $\boldsymbol{x}$ 最近的样本,即得到 $\boldsymbol{x}$ 的邻域 $N_k(\boldsymbol{x})$。

(2) 计算 $v(N_k(\boldsymbol{x}))$ 得到 $\boldsymbol{x}$ 的标签值。

$d(\cdot)$ 常用欧氏距离。$v(\cdot)$ 常用求均值函数、线性回归模型和局部加权线性回归模型。

k 值的大小对算法有重大影响。过小的 k 值,结果对噪声更敏感,容易发生过拟合;过大的 k 值,较远的节点也会影响结果,近似误差(Approximation Error)会增大。

K 近邻法也不形成固定模型,预测时计算量相对较大。

应用 K 近邻法求解分类问题,只要将三要素中的计算标签值的方法改为计算分类标签的方法即可。计算分类标签的方法常采用投票法。

sklearn 中实现 K 近邻回归的类是 neighbors 包中的 KNeighborsRegressor,实现 K 近邻分类的类是 KNeighborsClassifier。

3.7　练习题

1. 用 sklearn.linear_model 包中的 LinearRegression 对表 3-1 所示的示例进行线性回归实验,比较结果。

2. 查阅资料,研究梯度下降法中步长的动态调整方法,试将代码 3-6 中固定步长改

为动态步长，并对比两者运行结果。

3. 试修改代码 3-6 实现批梯度下降和随机梯度下降算法，并从时间和结果两方面与原算法进行比较。

4. 实现岭回归的最小二乘法求解算法，并进行实验。

5. 实现岭回归的梯度下降法求解算法，进行实验，并与最小二乘法求解结果进行比较。