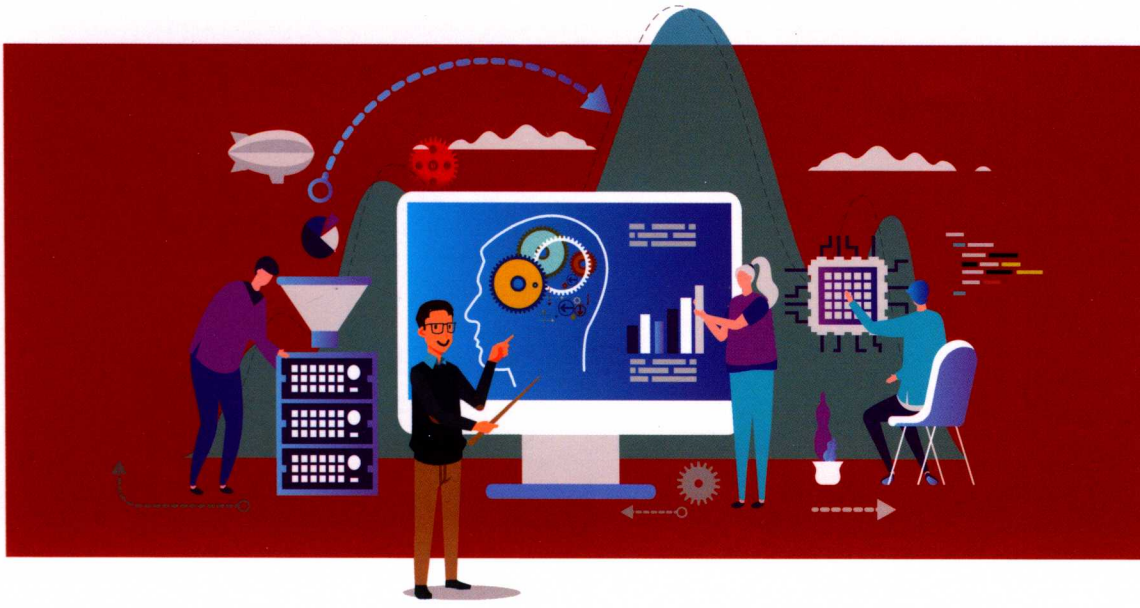


Python

机器学习算法

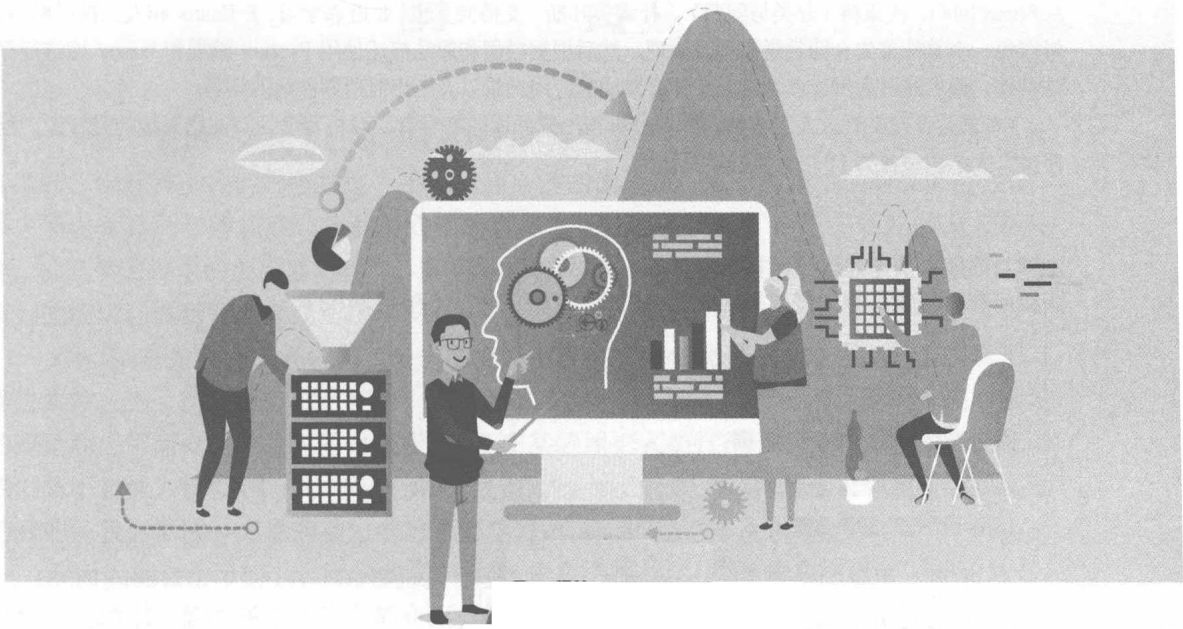
原理、实现与案例

刘硕 著

算法原理·数学推导·代码实现·项目实战

写给初学者的机器学习算法入门书

清華大學出版社



Python

机器学习算法

原理、实现与案例

刘硕 著

清华大学出版社
北京

内 容 简 介

本书用平实的语言深入浅出地介绍当前热门的机器学习经典算法，包括线性回归、Logistic 回归与 Softmax 回归、决策树（分类与回归）、朴素贝叶斯、支持向量机、K 近邻学习、K-Means 和人工神经网络，针对每一个算法首先介绍数学模型及原理，然后根据模型和算法描述使用 Python 编程和 Numpy 库进行算法实现，最后通过案例让读者进一步体会算法的应用场景以及应用时所需注意的问题。

本书适合准备进入人工智能和数据分析与挖掘领域的初学者，对机器学习算法感兴趣的爱好者、程序员、大学生和各类 IT 培训班的学员使用。

图书在版编目（CIP）数据

Python 机器学习算法：原理、实现与案例/刘硕著. —北京：清华大学出版社，2019
ISBN 978-7-302-53650-5

Ⅰ. ①P… Ⅱ. ①刘… Ⅲ. ①软件工具—程序设计 ②机器学习 Ⅳ. ①TP311.561②TP181

中国版本图书馆 CIP 数据核字（2019）第 186637 号

责任编辑：王金柱
封面设计：王　翔
责任校对：闫秀华
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：13.5　　**字　　数**：303 千字
版　　次：2019 年 11 月第 1 版　　**印　　次**：2019 年 11 月第 1 次印刷
定　　价：69.00 元

产品编号：083701-01

前　言

近年来，机器学习技术已经渗透到我们日常生活的各个方面，比如网上购物时的商品推荐、浏览网页时的广告推送、手机拍照后的图像处理、电子邮箱中的垃圾邮件过滤、停车场出入口的车牌识别、各种游戏中的机器人玩家以及汽车厂商正在研发的无人驾驶等，机器学习技术的应用随处可见，并且它的发展极其迅猛，在更多领域令人兴奋（或恐惧）的应用已被研发出来或正在研发中。

尤瓦尔•赫拉利在其畅销书《未来简史》中表明了一个观点：未来的世界由机器学习算法掌控。当了解到像谷歌、苹果、亚马逊、IBM 这样的大公司投入巨资用于机器学习的理论和应用研究，并且时不时就听到 AI 在某领域把人类打得一败涂地的新闻时，或许我们就不会认为赫拉利的观点是离谱的异端邪说，或出于好奇，或出于恐惧，或出于实际的目的，我们都应有充足的动力学习机器学习。

市面上机器学习的书已经很多了，大体上分为两类：一类是偏重机器学习理论的书，这种类型的书，算法理论部分大都介绍得很详细，但对于算法仅给出粗略的伪代码，而没有详尽的编码实现，也没有提供案例应用，在初学者对机器学习了解甚少的情况下，直接面对枯燥烦琐的数学推导，难免痛苦与沮丧。另外，由于初学者很难直接根据理论自己实现算法以及恰当地运用算法进行项目实践，因此无法验证学习成果。另一类是偏重机器学习应用的书，这类书对算法的理论进行了简单的提及，省略了有助于理解的重要数学推导，且大多数不会带领读者编码实现一个算法，而是直接使用开源库（如sklearn）中实现的算法，这类书算法的案例应用部分介绍得很详细，初学者会对机器学习应用有所了解，但由于理论匮乏且没有亲自动手实现算法，故导致无法深入理解算法，学习一段时间后大部分内容便忘记了。

本书是一本写给初学者的机器学习算法入门书，试图填补以上两类书的不足。本书在讲解算法时，首先详细介绍数学模型及原理，然后带领读者根据模型和算法描述进行算法实现，最后通过案例让读者进一步体会算法的应用场景以及应用时所需注意的问题。其中，算法实现部分是本书的重点，这部分所有算法的实现都基于 Numpy 这样一个非常底层的数学库，这就意味着需自己手工实现更多的细节，例如在计算损失函数的梯度时，需要手工推导计算梯度的数学公式，然后对照公式编码实现计算梯度的函数，相信本书的这种做法对初学者来说是一个有益的训练。另外，书中几乎每一行代码都给出了详尽的注释，通过代码注释来讲解算法的实现能给读者带来更好的体验，也便于读者理解编码的思想。

本书精选了经典的机器学习算法进行讲解，主要包括：

- 线性回归
- Logistic 回归与 Softmax 回归
- 决策树（分类与回归）
- 朴素贝叶斯
- 支持向量机
- k 近邻学习
- K-Means
- 人工神经网络

以上算法涵盖机器学习领域重要的思想，建议初学者在学完本书后能深入理解并自己实现以上算法，它们是日后在机器学习领域继续深入学习的基础。

本书还提供了算法示例的源代码，读者可以扫描以下二维码下载：

如何在下载过程中遇到问题，请联系 booksaga@163.com，邮件主题为“Python 机器学习算法：原理、实现与案例下载资源”。

由于笔者水平有限，书中难免出现纰漏，恳请读者不吝赐教。

最后，感谢清华大学出版社的王金柱编辑对笔者的信任和在写作方面的指点，我们的第二次合作非常愉快。

刘　硕

2019 年 7 月

目　录

第1章

线性回归

线性回归是最简单的机器学习模型，其形式简单，易于实现，同时也是很多机器学习模型的基础。第一个机器学习算法我们便从线性回归讲起。

1.1 线性回归模型

对于一个给定的训练集数据，线性回归的目标是找到一个与这些数据最为吻合的线性函数。举一个例子，中学物理我们学过的胡克定律指出：弹簧在发生弹性形变时，弹簧所受拉力F和弹簧的形变量 x 成正比，即$F=kx$。假设我们拿到一个新弹簧，测得了一组包含弹簧所受拉力F和形变量x的实验数据，如图 1-1 所示，根据实验数据估计弹簧倔强系数k的过程就是线性回归。

一般情况下，线性回归模型假设函数为：

$$h_{w,b}(x)=\sum_{i=1}^{n} w_i x_i + b = w^{\mathrm{T}} x + b$$

其中，$w \in \mathbf{R}^n$与 $b \in \mathbf{R}$ 为模型参数，也称为回归系数。为了方便，通常将 b 纳入权向量 w，作为 w_0，同时为输入向量 x 添加一个常数 1 作为 x_0：

$$w = (b, w_1, w_2, \ldots w_n)^{\mathrm{T}}$$
$$x = (1, x_1, x_2, \ldots x_n)^{\mathrm{T}}$$

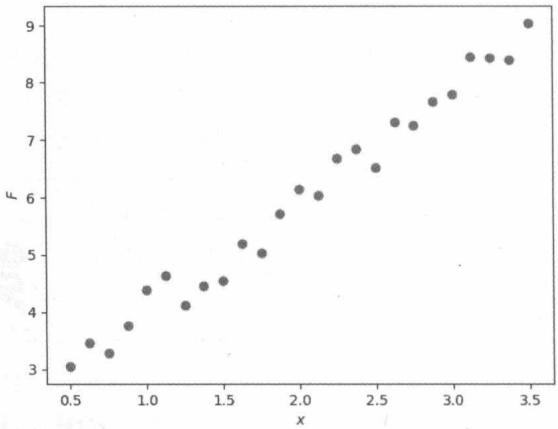


图 1-1

此时，假设函数为：

$$h_w(x) = \sum_{i=0}^{n} w_i x_i = w^{\mathrm{T}} x$$

其中，$w \in \mathbf{R}^{n+1}$，通过训练确定模型参数 w 后，便可使用模型对新的输入实例进行预测。

1.2 最小二乘法

线性回归模型通常使用均方误差（MSE）作为损失函数，假设训练集 $\boldsymbol{D}$ 有 m 个样本，均方误差损失函数定义为：

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$$
$$= \frac{1}{2m} \sum_{i=1}^{m} (w^{\mathrm{T}} x - y_i)^2$$

均方误差的含义很容易理解，即所有实例预测值与实际值误差平方的均值，模型的训练目标是找到使得损失函数最小化的w。式中的常数 $\frac{1}{2}$ 并没有什么特殊的数学含义，仅是为了优化时求导方便。

损失函数 $J(w)$ 最小值点是其极值点，可先求 $J(w)$ 对 w 的梯度并令其为 0，再通过解方程求得。

计算 $J(w)$ 的梯度：

$$\begin{aligned}\nabla J(w) &= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial w}(w^{\mathrm{T}}x_i - y_i)^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}2(w^{\mathrm{T}}x_i - y_i)\frac{\partial}{\partial w}(w^{\mathrm{T}}x_i - y_i) \\ &= \frac{1}{m}\sum_{i=1}^{m}(w^{\mathrm{T}}x_i - y_i)x_i\end{aligned}$$

以上公式使用矩阵运算描述形式更为简洁，设：

$$X = \begin{bmatrix} 1, & x_{11}, & x_{12} & \cdots & x_{1n} \\ 1, & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1, & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} x_1^{\mathrm{T}} \\ x_2^{\mathrm{T}} \\ \vdots \\ x_m^{\mathrm{T}} \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$w = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

那么，梯度计算公式可写为：

$$\nabla J(w) = \frac{1}{m}X^{\mathrm{T}}(Xw - y)$$

令梯度为 0，解得：

$$\hat{w} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y$$

式中，$\hat{w}$ 即为使得损失函数（均方误差）最小的 w。需要注意的是，式中对 $X^{\mathrm{T}}X$ 求了逆矩阵，这要求 $X^{\mathrm{T}}X$ 是满秩的。然而实际应用中，$X^{\mathrm{T}}X$不总是满秩的（例如特

征数大于样本数），此时可解出多个 $\hat{\boldsymbol{w}}$，选择哪一个由学习算法的归纳偏好决定，常见做法是引入正则化项。

以上求解最优 $\boldsymbol{w}$ 的方法被称为普通最小二乘法（Ordinary Least Squares，OLS）。

1.3 梯度下降

1.3.1 梯度下降算法

有很多机器学习模型的最优化参数不能像普通最小二乘法那样通过“闭式”方程直接计算，此时需要求助于迭代优化方法。通俗地讲，迭代优化方法就是每次根据当前情况做出一点点微调，反复迭代调整，直到达到或接近最优时停止，应用最为广泛的迭代优化方法是梯度下降（Gradient Descent）。图 1-2 所示为梯度下降算法逐步调整参数，从而使损失函数最小化的过程示意图。

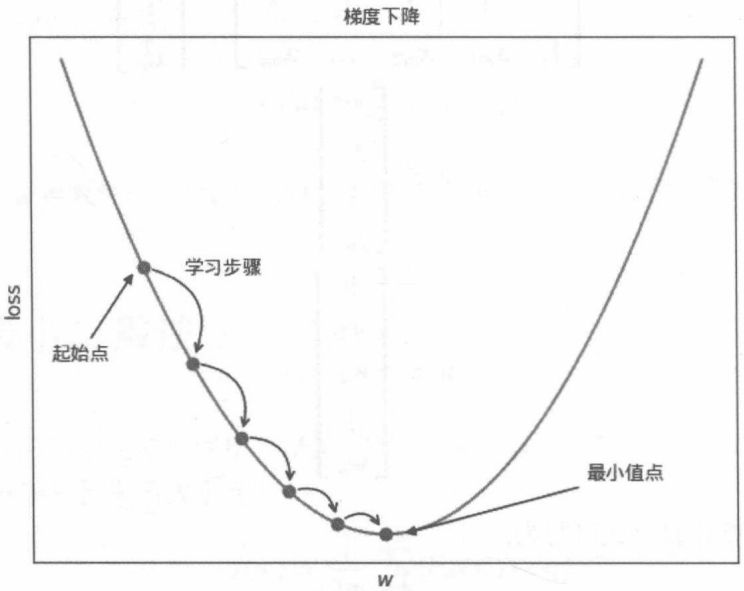


图 1-2

梯度下降算法常被形象地比喻为“下山”。如果你想尽快走下一座山，那么每迈一步的方向应选择当前山坡最陡峭的方向，迈一步调整一下方向，一直走到发现脚下已是平地。对于函数而言，梯度向量的反方向是其函数值下降最快的方向，即最陡峭的方向。梯度下降算法可描述为：

（1）根据当前参数 w计算损失函数梯度 $\nabla J(w)$。

（2）沿着梯度反方向 $-\nabla J(w)$ 调整 w，调整的大小称为步长，由学习率 η 控制。使用公式表述为：

$$w := w - \eta \nabla J(w)$$

（3）反复执行上述过程，直到梯度为 0 或损失函数降低小于阈值，此时称算法已收敛。

应用梯度下降算法时，超参数学习率 η 的选择十分重要。如果 η 过大，则有可能出现走到接近山谷的地方又一大步迈到了山另一边的山坡上，即越过了最小值点；如果 η 过小，下山的速度就会很慢，需要算法更多次的迭代才能收敛，这会导致训练时间过长。以上两种情形如图 1-3 所示。

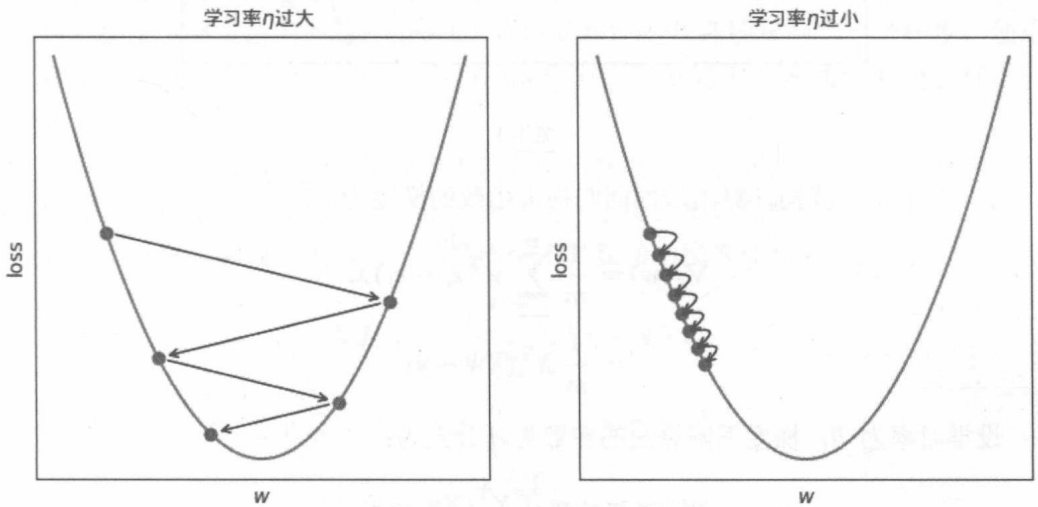


图 1-3

另外，还需知道的是，图 1-3 中的损失函数对于梯度下降算法是很理想的，它仅有一个全局最小值点，算法最终将收敛于该点。但也有很多机器学习模型的损失函数存在局部最小值，其曲线如绵延起伏的山脉，如图 1-4 所示。

对于图 1-4 中的损失函数，假设梯度下降算法的起始点位于局部最小值点左侧，算法则有可能收敛于局部最小值，而非全局最小值。此例子表明，梯度下降算法并不总收敛于全局最小值。

本节我们讨论的线性回归的损失函数是一个凸函数，不存在局部最小值，即只有一个全局最小值，因此梯度下降算法可收敛于全局最小值。

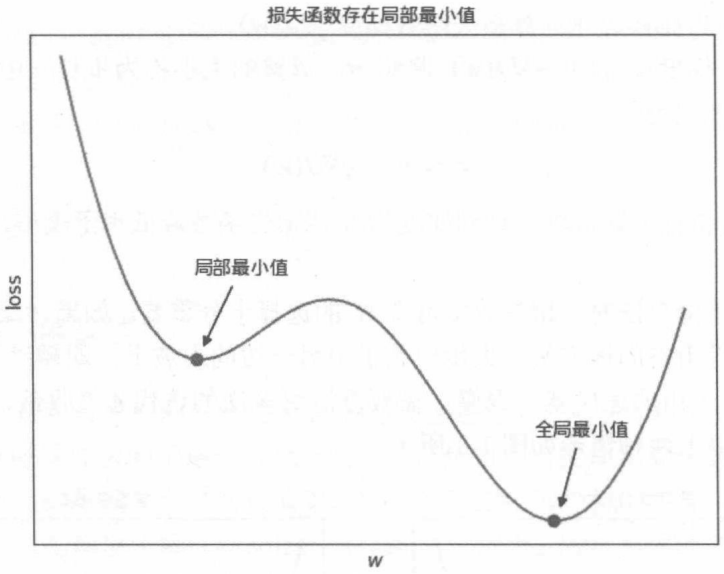


图 1-4

在 1.2 节中，我们计算出线性回归损失函数的梯度为：

$$\nabla J(w) = \frac{1}{m}\sum_{i=1}^{m}(w^{\mathrm{T}}x_i - y_i)x_i$$
$$= \frac{1}{m}X^{\mathrm{T}}(Xw - y)$$

设学习率为 η，梯度下降算法的参数更新公式为：

$$w := w - \eta\frac{1}{m}X^{\mathrm{T}}(Xw - y)$$

可以看出，执行梯度下降算法的每一步都是基于整个训练集 $\boldsymbol{X}$ 计算梯度的，因此梯度下降也被称为批量梯度下降：每次使用整批训练样本计算梯度，在训练集非常大时，批量梯度下降算法会运行得极慢。1.3.2 小节将介绍的随机梯度下降和小批量梯度下降可以解决该问题。

1.3.2 随机梯度下降和小批量梯度下降

随机梯度下降和小批量梯度下降可以看成是对批量梯度下降的近似，算法流程基本相同，只是每步使用少量的训练样本计算梯度。

随机梯度下降是与批量随机下降相反的极端情况，每一步只使用一个样本来计算梯度。

随机梯度下降算法的梯度计算公式为：

$$\nabla J(w) = (w^{\mathrm{T}} x_i - y_i) x_i$$

设学习率为 η，随机梯度下降算法的参数更新公式为：

$$w := w - \eta (w^{\mathrm{T}} x_i - y_i) x_i$$

因为每次只使用一个样本来计算梯度，所以随机梯度下降运行速度很快，并且内存开销很小，这使得随机梯度下降算法可以支持使用海量数据集进行训练。随机梯度下降过程中，损失函数的下降不像批量梯度下降那样缓缓降低，而是不断上下起伏，但总体上趋于降低，逐渐接近最小值。通常随机梯度下降收敛时，参数 w 是足够优的，但不是最优的。随机梯度下降算法的另一个优势是，当损失函数很不规则时（存在多个局部最小值），它更有可能跳过局部最小值，最终接近全局最小值。

随机梯度下降算法的一轮（Epoch）训练是指：迭代训练集中每一个样本，使用单个样本计算梯度并更新参数（一轮即 m 步），在每轮训练前通常要随机打乱训练集。

小批量梯度下降是介于批量梯度下降和随机梯度下降之间的折中方案，每一步既不使用整个训练集又不使用单个样本，而使用一小批样本计算梯度。

设一小批样本的数量为 N，小批量梯度下降算法的梯度计算公式为：

$$\nabla J(w) = \frac{1}{N} \sum_{i=k}^{k+N} (w^{\mathrm{T}} x_i - y_i) x_i$$

设学习率为 η，小批量梯度下降算法的参数更新公式为：

$$w := w - \eta \frac{1}{N} \sum_{i=k}^{k+N} (w^{\mathrm{T}} x_i - y_i) x_i$$

小批量梯度下降同时具备批量梯度下降和随机梯度下降二者的优缺点，应用时可视具体情况指定 N 值。

1.4　算法实现

1.4.1　最小二乘法

首先，我们基于最小二乘法实现线性回归，代码如下：

```
import numpy as np

class OLSLinearRegression:

    def _ols(self, X, y):
        '''最小二乘法估算 w'''
        tmp = np.linalg.inv(np.matmul(X.T, X))
        tmp = np.matmul(tmp, X.T)
        return np.matmul(tmp, y)

        # 若使用较新的 Python 和 Numpy 版本，可使用如下实现
        # return np.linalg.inv(X.T @ X) @ X.T @ y

    def _preprocess_data_X(self, X):
        '''数据预处理'''

        # 扩展 X，添加 x_0 列并设置为 1
        m, n = X.shape
        X_ = np.empty((m, n + 1))
        X_[:, 0] = 1
        X_[:, 1:] = X

        return X_

    def train(self, X_train, y_train):
        '''训练模型'''

        # 预处理 X_train(添加 x0=1)
        X_train = self._preprocess_data_X(X_train)

        # 使用最小二乘法估算 w
        self.w = self._ols(X_train, y_train)

    def predict(self, X):
        '''预测'''
        # 预处理 X_train(添加 x0=1)
        X = self._preprocess_data_X(X)
        return np.matmul(X, self.w)
```