《机器学习》课程系列

基础知识*

武汉纺织大学数学与计算机学院

杜小勤

2019/02/02

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 21, 2020。

1 概述

1.1 机器学习

本质上讲，机器学习是一种自动或半自动的数据处理与分析方法，能够从数据中提取出有用的模式和模型，并利用它们完成各种预测、分类、标注以及决策任务。

机器学习所处理与研究的对象是各种形式的数据，例如图像、声音、视频、网页、基因等数据，不论这些数据是以交互式的、在线的方式提供，还是以非交互式的、离线的方式提供[1]。

机器学习是一个多学科交叉的研究领域，其主要的理论根基是概率论与统计学、信息论、最优化理论与计算理论等。

机器学习算法作为机器学习研究的目标与核心，如何给出它的定义呢？Herbert A. Simon 曾对“学习”给出了如下的定义：如果一个系统能够通过执行某个过程改进它的性能，那么这就是学习。可见，学习本身是一个广义的概念，从这个角度看，让（机器）系统拥有学习能力的算法就是一个机器学习算法。

显然，从不同的角度可以对机器学习算法进行分类。我们从学习方式的角度对机器学习算法进行如下的分类：

- 监督学习

  在监督学习下，机器学习算法需要一个数据集（Data Set）。该数据集的基本单位是样例（Sample），每个样例包括输入数据及其理想的输出（Supervisor）。机器学习算法的任务是从数据集中学习“正确的”输入输出映射关系，或建立“正确的”模型。其最终目标是，能够“正确地”将这种映射关系或模型泛化到未见的输入数据上，以产生有用的输出。

- 无监督学习

  在无监督学习中，每个样例中没有包括理想的输出，机器学习算法需要自己从数据集中发现有用的模式或模型。

- 半监督学习

[1]例如，在典型的监督式学习任务中，一般以非交互式离线的方式训练机器学习算法；而在强化学习任务中，一般以交互式在线的方式让智能体自己进行自主的学习。

在半监督学习中，并不是数据集中的每个样例都包括理想的输出，而是只有一部分样例包括[2]。机器学习算法需要有效地利用这些样例数据。

- 强化学习

  从学习反馈的角度来看，这种方式介于监督学习与无监督学习之间——监督学习提供正确的监督数据作为引导，无监督学习根本不提供监督数据。强化学习则提供某种“评价”信息作为引导：需要环境给学习算法的每一次执行给出一个评价或分数。

  从交互方式来看，监督学习、无监督学习在训练时不需要与环境进行交互，即以离线的方式进行训练；强化学习则是智能体（Agent）自主地通过与环境的交互而进行学习。

- 演化学习

  这是一种模拟生物进化进程的学习方法，通过适应度函数来评价种群中每个个体的优劣，让优秀的个体能够得到更多的繁衍机会。经过若干次迭代之后，种群中优秀个体的数量会越来越多，在满足一定的条件时，可以得到满足应用要求的解。

1.2 模型、策略和算法

任何一个学习系统包括如下三个要素，即模型、策略和算法。首先，需要确定学习的模型——确定包含所有可能模型的假设空间（Hypothesis Space），即学习模型的集合，本质上，它是函数的集合。然后，需要确定选择模型的某个评价准则（Evaluation Criterion）或策略——从假设空间中选取一个“最优的”模型，使得该模型对于已知数据及未知数据在给定的评价准则下具有“最优性”。最后，由算法实现最优模型的选取。

机器学习的模型可以是概率模型，也可以是非概率模型，前者可以由条件概率分布 $p(\mathbf{y}|\mathbf{x})$ 来表示，其中 $\mathbf{x}$ 和 $\mathbf{y}$ 分别表示输入空间与输出空间上的随机变量；后者可以由决策函数（Decision Function）$\mathbf{y} = f(\mathbf{x})$ 来表示，其中 $\mathbf{x}$ 和 $\mathbf{y}$ 分别表示输入空间与输出空间上的变量。

[2]考虑到数据收集的成本与困难等因素，或者数据本身的稀有性，这些应用场合下，无法提供充足的带有监督信息的数据。

1.2.1 基本概念

输入数据所在的空间被称为输入空间（Input Space），它也是输入变量所有可能取值的集合。相应地，输出数据所在的空间被称为输出空间（Output Space），它也是输出变量所有可能取值的集合。输入空间与输出空间可以是有限集合，也可以是无限集合；输入空间与输出空间可以位于同一个空间，也可以位于不同的空间；一般情况下，输出空间远小于输入空间。

每个具体的输入是一个实例（Instance），通常由特征向量（Feature Vector）表示，它的每一维对应一个特征。所有特征向量所在的空间为特征空间（Feature Space）。在一些情况下，输入空间与特征空间是一致的，即直接将输入空间作为特征空间进行处理。在大多数情况下，需要将输入空间变换到特征空间进行处理，这被称为预处理（Preprocessing）或特征提取（Feature Extraction）。我们期望，在新的特征空间中，机器学习问题要变得容易些。因此，实际上，模型都是定义在特征空间上的。

为了统一记号，我们做出如下约定。$\mathbf{x}$ 表示输入变量，它是列向量；x 表示输入变量，它是标量。$\mathbf{x}$ 的实例表示为 $\mathbf{x}_i = \left(x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(n)}\right)^{\mathrm{T}} \in \mathbf{R}^n$，其中 $x_i^{(j)}$ 表示输入实例 $\mathbf{x}_i$ 的第 j 个特征。标量变量 x 的实例表示为 x_i。$\mathbf{y}$、$\mathbf{y}_i$、y 和 y_i 的意义与上面的一样。为了讨论方便，除非另外说明，下面的讨论均假定变量为向量形式。

数据集表示为 $\mathbf{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$，其中 N 表示训练集中样例的数目。可以看出，每对样本（Sample）$(\mathbf{x}_i, \mathbf{y}_i)$ 由输入与输出对组成，其中 $\mathbf{y}_i$ 被称为标记（Label）或监督。一般情况下，数据集需要分成训练数据集、验证数据集与测试数据集，它们的作用将在后面进行说明。模型的训练结果可以被表示为一个函数 $f(\mathbf{x})$，它的精确形式在训练（Training）阶段就已经被确定，该阶段也可以被称为学习（Learning）阶段。以训练数据集为基础，一旦模型被训练出来，它就能够对新输入变量 $\mathbf{x}$ 的输出做出某种预测。在实际应用中，输入向量所在的输入空间相当大，训练数据只占其中相当小的一部分。因此，对未知的、未见的新输入变量 $\mathbf{x}$ 的预测能力是机器学习中的一个核心问题。我们把这种能力称为机器学习算法的泛化（Generalization）能力。

输入变量 $\mathbf{x}$ 和输出变量 $\mathbf{y}$ 可以是离散的，也可以是连续的。在监督学习中，根据输入、输出变量的不同类型，可以将预测任务分成以下三类：

- 回归（Regression）任务

  输入变量与输出变量均为连续变量的预测问题；

- 分类（Classification）任务

  输出变量为有限个离散变量的预测问题；

- 标注（Tagging）任务

  输入变量与输出变量均为变量序列的预测问题；

在监督学习中，学习算法总是要依赖于给定的、有限的、用于学习的训练数据集（Training Data Set），并且往往需要做出一些假设。第 1 个假设是，所处理的数据是以独立同分布（i.i.d）的形式产生的。第 2 个假设是，输入与输出随机变量 $\mathbf{x}$ 和 $\mathbf{y}$ 的联合概率分布 $p(\mathbf{x}, \mathbf{y})$（也被称为联合分布函数或联合分布密度函数）存在，即所处理的数据被看作是依该联合概率分布函数 $p(\mathbf{x}, \mathbf{y})$ 以独立同分布的形式产生的。但是，对于机器学习系统而言，该联合概率分布的具体定义是未知的。这意味着，监督学习假设待处理的数据存在一定的统计规律。

1.2.2 监督学习系统的组成

监督学习是机器学习的重要组成部分，占据着非常重要的地位，也是统计学习中内容最丰富、应用最广泛的部分。下面给出监督学习的形式化描述。

如图1-1所示，展示了监督学习系统的组成部分。

监督学习系统由 2 个模块组成：学习或训练系统、预测系统。从过程阶段来看，学习或训练是第一阶段，预测或应用是第二阶段。学习系统的任务是，从训练数据中生成预测模型或生成决策函数。预测系统则是对新数据进行预测，以完成分类、回归或标注任务。

1.2.3 模型

在监督学习中，模型就是所要学习的条件概率分布或决策函数，模型的假设空间包括了所有可能的条件概率分布或决策函数。假设空间中的模型一般有无穷多个。例如，假设决策函数是输入变量的线性函数，那么模型的假设空间就是所有由这些线性函数组成的函数集合。

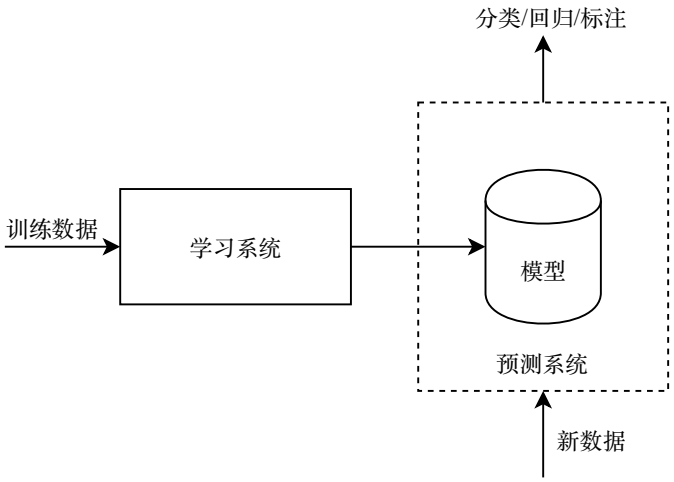


图 1-1: 监督学习系统

设假设空间为 $\mathcal{H}$，它可以被定义为决策函数的集合：

$$\mathcal{H} = \{f|\mathbf{y} = f(\mathbf{x})\} \tag{1}$$

其中，$\mathbf{x}$ 和 $\mathbf{y}$ 分别为输入空间和输出空间上的变量。$\mathcal{H}$ 通常是由一个参数向量确定的函数族，此时假设空间 $\mathcal{H}$ 可以表示为：

$$\mathcal{H} = \{f|\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{x}), \boldsymbol{\theta} \in \mathbf{R}^n\} \tag{2}$$

其中 $\boldsymbol{\theta} \in \mathbf{R}^n$ 是一个取值于 n 维参数空间（Parameter Space）的参数向量。

相应地，假设空间 $\mathcal{H}$ 也可以定义为条件概率的集合：

$$\mathcal{H} = \{p|p\left(\mathbf{y}|\mathbf{x}\right)\} \tag{3}$$

其中，$\mathbf{x}$ 和 $\mathbf{y}$ 分别为输入空间和输出空间上的随机变量。$\mathcal{H}$ 通常是由一个参数向量确定的条件概率分布族，此时假设空间 $\mathcal{H}$ 可以表示为：

$$\mathcal{H} = \{p|p_{\boldsymbol{\theta}}\left(\mathbf{y}|\mathbf{x}\right), \boldsymbol{\theta} \in \mathbf{R}^n\} \tag{4}$$

其中 $\boldsymbol{\theta} \in \mathbf{R}^n$ 是一个取值于 n 维参数空间（Parameter Space）的参数向量。

1.2.4 策略

有了模型的假设空间，需要定义选取“最优”模型的策略，即从假设空间中选取符合任务需求的“最优”模型。这需要我们定义某些选取模型的量化准则。

下面引入损失函数（Loss Function 或 Cost Function）与风险函数的概念，前者度量模型的单个预测值与理想值的接近程度，两者越接近，损失值越小，表明模型对该数据的预测效果好；后者是度量模型的整体预测值（通常是针对一个数据子集）与理想值的接近程度，风险值越小，表明模型对该数据子集的预测效果好。

损失函数是单个预测值与理想值的非负实值函数，记作 $L(\mathbf{y}, f(\mathbf{x}))$ 或 $L(\mathbf{y}, p(\mathbf{y}|\mathbf{x}))$。常见的损失函数有如下几种：

- 0-1 损失函数（0-1 Loss Function）

$$L(\mathbf{y}, f(\mathbf{x})) = \begin{cases} 1 & \mathbf{y} \neq f(\mathbf{x}) \\ 0 & \mathbf{y} = f(\mathbf{x}) \end{cases} \tag{5}$$

- 平方损失函数（Quadratic Loss Function）

$$L(\mathbf{y}, f(\mathbf{x})) = (\mathbf{y} - f(\mathbf{x}))^2 \tag{6}$$

- 绝对损失函数（Absolute Loss Function）

$$L(\mathbf{y}, f(\mathbf{x})) = |\mathbf{y} - f(\mathbf{x})| \tag{7}$$

- 对数损失函数（Logarithmic Loss Function）或对数似然损失函数（Log-Likelihood Loss Function）[3]

$$L(\mathbf{y}, p(\mathbf{y}|\mathbf{x})) = -\log p(\mathbf{y}|\mathbf{x}) \tag{8}$$

风险函数（Risk Function）是模型关于随机变量 $\mathbf{x}$ 和 $\mathbf{y}$ 的联合概率分布 $p(\mathbf{x}, \mathbf{y})$ 的期望损失（Expected Loss）[4]：

$$R_{\exp}(f) = E_p[L(\mathbf{y}, f(\mathbf{x}))] = \int_{\mathcal{X}\times\mathcal{Y}} L(\mathbf{y}, f(\mathbf{x}))p(\mathbf{x}, f(\mathbf{y}))\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y} \tag{9}$$

[3]从概率的角度看，$p(\mathbf{y}|\mathbf{x})$ 可以被称为条件概率。从函数的角度看，$p(\mathbf{y}|\mathbf{x})$ 可以被称为似然函数。$p(\mathbf{y}|\mathbf{x})$ 反映了当前模型下，输入数据 $\mathbf{x}$ 下得到输出数据 $\mathbf{y}$ 的概率，也反映了预测值与理想值的接近程度——概率越大，预测值与理想值越接近，损失越小，反之亦然。

[4]在离散情况下，可以定义相应的风险函数或期望损失。

其中，$\mathcal{X}$ 和 $\mathcal{Y}$ 分别为输入空间与输出空间。实际上，风险函数或期望损失只是理论上的模型，原因在于，联合概率分布 $p(\mathbf{x}, f(\mathbf{y}))$ 是未知的，没有办法直接计算 $R_{\text{exp}}(f)$。

因此，在监督学习中，需要针对特定的训练数据集定义模型的平均损失，我们把这种风险函数称为经验风险函数（Empirical Risk Function）或经验损失函数（Empirical Loss Function）：

$$R_{\text{emp}}(f) = \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{y}_i, f\left(\mathbf{x}_i\right)\right) \tag{10}$$

其中，N 表示训练数据集 $D = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ 中样例的个数。

期望风险 $R_{\text{exp}}(f)$ 是模型关于联合概率分布 $p(\mathbf{x}, f(\mathbf{y}))$ 的期望损失，而经验风险 $R_{\text{emp}}(f)$ 是模型关于训练数据集的平均损失。因此，根据大数定理，当样本容量 N 趋于无穷时，经验风险 $R_{\text{emp}}(f)$ 将趋于期望风险 $R_{\text{exp}}(f)$。一个很自然的想法是，在训练模型时，使用经验风险 $R_{\text{emp}}(f)$ 替换期望风险 $R_{\text{exp}}(f)$，并根据经验风险最小化来选取“最优的”模型。

然而，这样做是存在很大风险的。原因在于，在实际应用中，训练数据集的容量往往是有限的，甚至很小。因此，在大多数情况下，直接使用经验风险替换期望风险的效果并不是很理想，需要对经验风险函数进行一定的矫正。从数据拟合的角度来说，风险函数的直接替换是监督学习产生过拟合（Over Fitting）问题的根源。为了抑制过拟合现象，往往需要引入正则化技术对经验风险函数进行矫正。实际上，正则化技术是一种在已知数据的预测能力与未知数据的预测能力之间取得平衡的策略。这涉及到监督学习的两个基本策略：经验风险与结构风险及相应的最小化策略[5]。

下面，首先给出经验风险最小化（Empirical Risk Minimization，ERM）策略：

$$\min_{f\in\mathcal{H}} R_{\text{emp}}(f) = \min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{y}_i, f\left(\mathbf{x}_i\right)\right) \tag{11}$$

其中 $\mathcal{H}$ 为假设空间。显然，该策略认为，经验风险最小化的模型就是“最优”模型。

[5]从更广义的角度看，学习算法的目标函数都需要某种程度的平衡或折中。例如，在监督学习中，存在经验风险最小化与模型复杂度之间的平衡；在强化学习与 Monte Calor 树搜索算法中，存在探索与利用的平衡。实际上，从泛化能力的角度来说，各类学习算法都需要在已知数据的预测能力与未知数据的预测能力之间取得某种平衡。

当训练数据集的容量充分大时，该策略能够取得较好的学习效果。这种模型的典型代表是，最大似然估计（Maximum Likelihood Estimation）。在后文我们将看到，当模型使用条件概率分布，而损失函数使用对数似然损失函数时，经验风险最小化等价于最大似然估计。

从上述描述可以看出，当训练样本的容量较小时，经验风险最小化存在过拟合问题。最主要的原因在于，这种策略的训练目标是让训练后的模型精确地匹配训练数据，即最小化训练数据集上的误差。在实际应用中，由于如下 2 个因素：训练样本少，不能反映数据的真实特征；训练数据中本身存在随机噪声或误差。这 2 个因素将使得经验风险最小化策略的弱点更加明显——由于模型过渡精确地匹配了训练数据，导致模型泛化到其它未见数据上的能力较差。可见，拟合能力与泛化能力是一对矛盾，需要取得某种平衡。

从统计学的角度来看，如果模型反映的是数据中的随机噪声或误差，而不是以模型误差的形式反映数据中的内在关系，我们称模型的训练出现了过拟合现象。为了抑制经验风险最小化策略中的过拟合问题，有必要在经验风险最小化策略中添加能够平衡拟合能力与泛化能力的因素。

结构风险最小化（Structural Risk Minimization，SRM）正是基于此思路而提出的策略，它在经验风险的基础上添加了表示模型复杂度的正则化项（Regularizer）或惩罚项（Penalty Term）：

$$R_{\text{srm}}(f) = \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{y}_i, f\left(\mathbf{x}_i\right)\right) + \lambda J(f) \tag{12}$$

其中，$J(f)$ 表示模型的复杂度，是定义在假设空间 $\mathcal{H}$ 上的泛函，模型 f 越复杂，$J(f)$ 就越大。在结构风险最小化时，需要同时考虑经验风险项和惩罚项的最小化，$\lambda \geq 0$ 是权重系数，用来平衡经验风险和模型复杂度的最小化程度。理论分析与实践结果都表明，结构风险小的模型对训练数据及未知数据均有较好的预测能力。结构风险最小化可以表示为：

$$\min_{f\in\mathcal{H}} R_{\text{srm}}(f) = \min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} L\left(\mathbf{y}_i, f\left(\mathbf{x}_i\right)\right) + \lambda J(f) \tag{13}$$

最大后验估计（Maximum Posterior Estimation）是结构风险最小化的典型例子。前述的最大似然估计是最大后验估计的特例。当模型使用条件概率分布，损失函数使用对数似然损失函数，而模型复杂度由模型的先验概率表示时，结构风

险最小化就等价于最大后验概率估计[6]。

因此，在确定了模型的选择策略后，监督学习问题也就演变成了经验风险函数或结构风险函数的最优化问题，此时的经验风险函数或结构风险函数是监督学习的目标函数。

1.2.5 算法

算法指的是模型学习或训练所采用的具体计算方法。对于前述的监督学习而言，就是考虑采用何种最优化求解方法求解出最优模型。

因此，大多数机器学习问题可以归结为最优化问题。对于最优化问题，常用的方法有解析法与迭代优化法。在实际应用中，数据的维度相当高，前者一般不适用。迭代优化法采用数值计算的方式，迭代地计算出最优化问题的解，常用的方法有梯度下降法、牛顿法、拟牛顿法等，在一些情况下，需要开发针对特定问题的最优化求解方法。

1.3 实例：多项式曲线拟合

下面通过一个多项式曲线拟合的例子来说明一些重要的概念，以获得对机器学习模型、策略与算法等方面的直观理解。

假设数据集为 $\mathbf{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$，其中 $N = 10$。这些数据点满足：$x_i \in [0, 1]$，$y_i = sin(2\pi x_i) + \mathcal{N}(0, 0.3^2)$，即在函数值 $sin(2\pi x_i)$ 的基础上，添加了均值为 0，标准差为 0.3 的高斯随机噪声。数据点的分布如图1-2所示。

我们的目标是，从数据集 $\mathbf{D}$ 中学习一个模型，并利用该模型对新输入变量 x 进行预测，以获得某个对应的预测值 $f(x)$。

1.3.1 问题的模型、策略和算法

理论上讲，这涉及到学习算法需要隐式地发现潜在的函数 $sin(2\pi x)$ 或类似的映射关系。本质上，这是一个困难的问题，因为给定的数据样本非常有限，并且

[6]最大后验估计是依据贝叶斯定理的，它综合考虑了条件概率与先验概率，前者对应着最大似然估计，后者对应着模型复杂度。因此，两者的结合在使用对数似然损失函数时，具有如下关系：最大后验估计 = 最大似然估计 + 正则化项。在模型的先验概率为常数的情况下，例如均匀概率分布，最大后验估计就退化为最大似然估计。

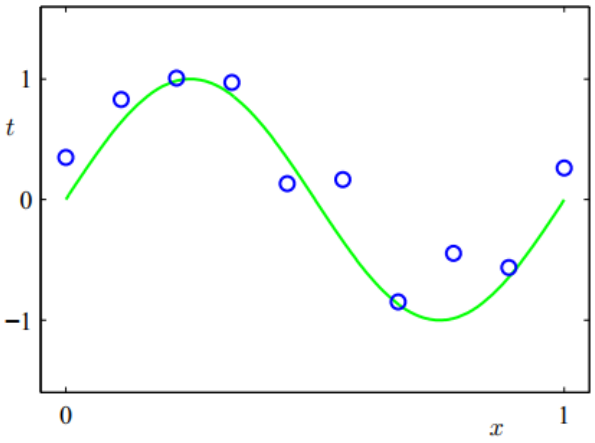


图 1-2: 训练数据点

这些数据样本点还包括噪声。因此，对于给定的新输入变量 x，其预测值具有某种不确定性。

下面给出一种解决该问题的思路。首先，确定具体的学习或训练模型。假设模型是一个关于 x 和 $\boldsymbol{w}$ 的多项式函数[7]，其中 $\boldsymbol{w}$ 表示模型的参数：

$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x \tag{14}$$

其中，M 是多项式的阶数，多项式函数 $f(x, \boldsymbol{w})$ 是 x 的一个非线性函数，却是系数 $\boldsymbol{w}$ 的一个线性函数[8]。

对于这个简单的单变量多项式模型，可以采用解析法求解。但是，此处，我们采用迭代的方式进行求解。具体而言，需要确定一个选择“最优”模型的策略。在此，我们定义经验风险函数[9]为[10]：

$$L(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \{f(x_n, \boldsymbol{w}) - y_n\}^2 \tag{15}$$

并求解优化问题：

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \min_{\boldsymbol{w}} \frac{1}{2} \sum_{n=1}^{N} \{f(x_n, \boldsymbol{w}) - y_n\}^2 \tag{16}$$

[7]这可以被称为多项式的曲线拟合问题。

[8]实际上，这是一个（关于多项式系数或参数的）线性模型。

[9]存在多个名称，例如目标函数、误差函数。在有些场合下，直接称为损失函数。在不引起歧义和混淆的情况下，可能会交替地使用这些名称。

[10]选择损失函数为平方损失函数是有原因的，后文将加以解释。另外，需要注意，所有的损失函数的值都是非负的，并且当做出正确预测时，误差将为 0。

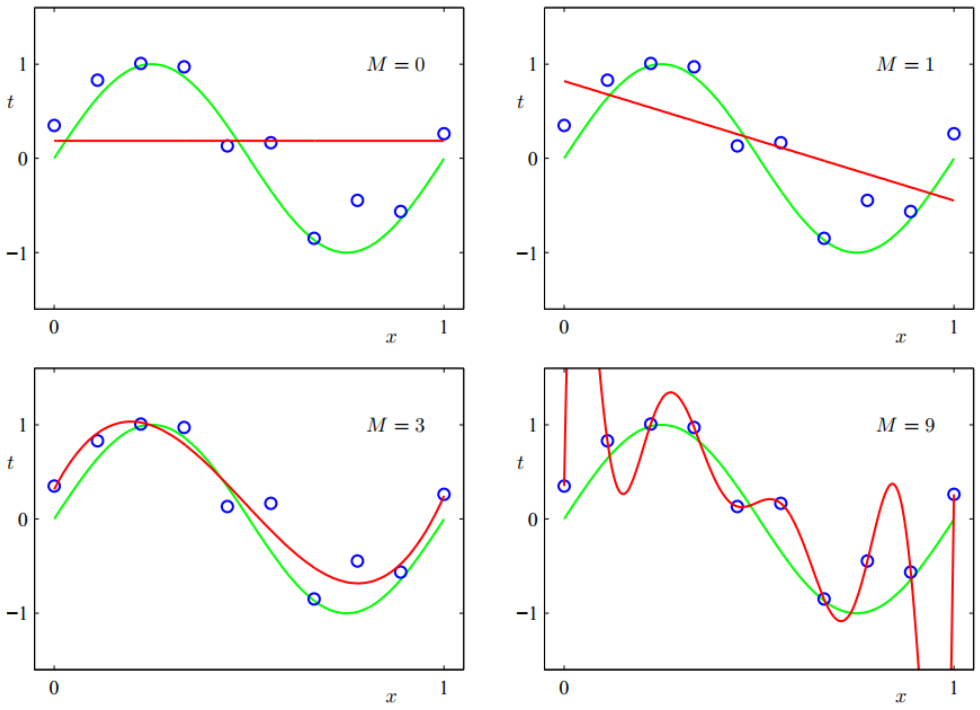


图 1-3: 不同阶数的多项式拟合的结果

最后，需要采用某种算法求解出最优解 $\boldsymbol{w}^*$。因为该多项式模型是关于参数 $\boldsymbol{w}$ 的线性模型，所以最优解 $\boldsymbol{w}^*$ 是唯一的。使用最优化求解方法，例如梯度下降法，迭代算法将会收敛。

到目前为止，我们已经确定了求解整个问题的框架与思路，唯一还需要确定的是该多项式模型的阶数 M。显然，模型的阶数 M 能够表示模型的复杂度，M 越大，多项式模型越复杂，能够拟合的曲线也越复杂。如图1-3所示，展示了阶数分别为 $M=0$、$M=1$、$M=3$、$M=9$ 的多边形模型的拟合结果。

在图1-3中，绿色曲线表示理想曲线，红色曲线表示拟合曲线。显然，阶数为 $M=0$ 和 $M=1$ 的模型的拟合效果很差，原因在于，模型太简单了，不足以表现或还原出真实的曲线数据点。而阶数 $M=9$ 的模型的拟合效果太“完美”了——精确地经过每个样本点，训练样本集上的整体误差为 0！它真的很完美吗？在我们已经确知隐藏模型为 $sin(2\pi x)$ 的情况下，阶数 $M=9$ 的多项式模型难以令人满意——其曲线与真实的曲线相差甚远，并且在区间 $[0,1]$ 的两端还存在剧烈的振荡。原因何在？阶数 $M=9$ 的多项式模型过于复杂，与样本数据点的真实模型的复杂度完全不匹配。从曲线拟合的角度看，这种现象就是过拟合。过拟合导致的

结果是，模型泛化到新数据的能力差，并且学习到了样本数据点中的噪声[11]，这是我们需要极力避免的。

在图1-3中，阶数 $M=3$ 的多项式模型，给出了最好的拟合效果，其曲线形状与真实的曲线形状也很相似。

1.3.2 测试数据集的引入

我们已经知道，对于模型而言，训练集上的误差为 0，也不一定能表明该模型是一个好模型，即我们不能单纯地使用训练集上的误差来衡量模型的好坏。模型泛化到新数据上的能力才是一个重要的衡量指标。

下面将使用定量的方式考察模型的泛化性能与模型复杂度 M 之间的关系，同时，也将看到，在训练样本点的个数有限的情况下，为什么阶数 $M=3$ 的模型要比阶数 $M=9$ 的模型要好。

为了定量地反映模型的泛化能力，需要引入一个测试数据集（Test Data Set），它的作用是反映模型在新数据上的误差。

关于误差的计算公式，除了前面引入的公式（15）外，还可以考虑均方根误差（Root Mean Square，RMS）公式[12]：

$$E_{RMS}=\sqrt{2L\left(\boldsymbol{w}^{*}\right)/N} \tag{17}$$

该公式是在公式（15）基础上经过变换所得的，它的意义非常直观：根式内的分母项为 N，便于比较不同大小的数据集；平方根确保了 E_{RMS} 与 $f(x)$ 都使用相同的单位进行度量。

图1-4反映了不同阶数下训练集与测试集的均方根误差曲线，测试集使用了 100 个样本点，数据的生成方式与训练集的生成方式完全一样。

从图1-4可以看出，训练集的误差曲线与测试集的误差曲线并不一致，这反映出不同阶数下模型的泛化能力不一样，也意味着在训练模型时，训练误差并非越低越好。从该图还可以看出，在当前的训练样本容量下，测试误差在某种程度上确实能够反映出模型的泛化能力：当 $3\leq M\leq 8$ 时，测试误差较小，拟合曲线与

[11]拟合曲线同样精确地经过了含有噪声的样本点。好的模型应该只学习样本数据点的内在特征。

[12]在不同的情况下，公式形式不同，但其意义是一样的。例如，对于标量类型变量，有：$x_{\mathrm{rms}}=\sqrt{\frac{1}{n}\left(x_1^2+x_2^2+\cdots+x_n^2\right)}$；对于区间上的连续型变量，有：$f_{\mathrm{rms}}=\sqrt{\frac{1}{T_2-T_1}\int_{T_1}^{T_2}[f(t)]^2dt}$；对于整个时间段上的连续型变量，有：$f_{\mathrm{rms}}=\lim\limits_{T\to\infty}\sqrt{\frac{1}{T}\int_0^T[f(t)]^2dt}$。

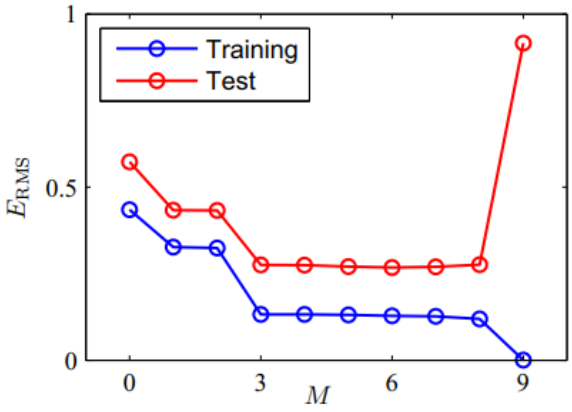


图 1-4: 不同阶数下训练集和测试集的均方根误差

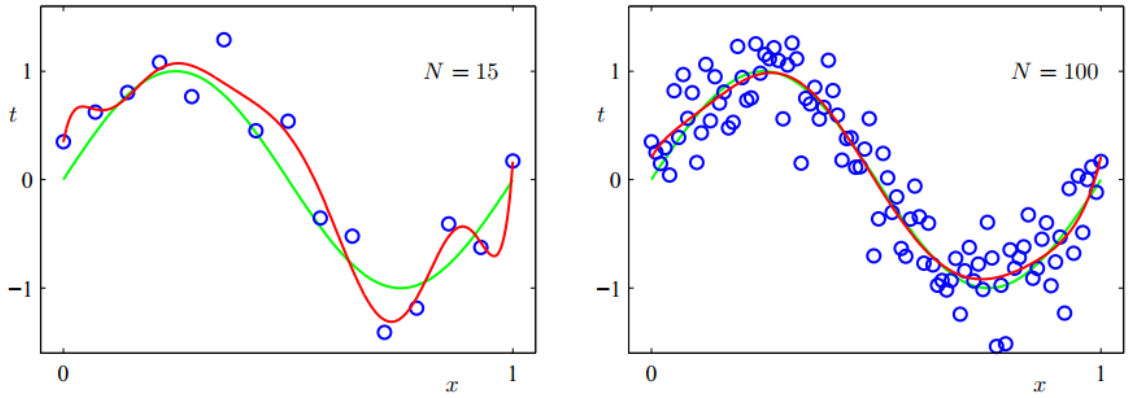


图 1-5: 训练样本点增多的情况下 M=9 模型的拟合曲线

真实曲线非常相似；对于 M 的其它取值，测试误差较大，拟合曲线与真实曲线相差较大。

但是，从纯粹数学的角度来看，上述结论似乎有点不对劲。真实函数 $sin(2\pi x)$ 的幂级数展开将会包含所有阶数的项——如果将真实函数按阶数 $M = 9$ 展开，其必定也包含阶数 $M = 3$ 的那些项，即 $M = 9$ 的模型至少应该与 $M = 3$ 的模型一样好，$M = 9$ 的模型应该更加精确才对。到底是哪里出了问题呢？实际上，在纯粹数学的视角下，我们忽略了一个非常重要的因素——训练数据集的容量。这是一个非常重要的因素，它能够改变待训练模型的泛化行为，如图1-5所示。

从图1-5可以看出，在训练样本点的容量分别增加到 $N = 15$ 和 $N = 100$ 的情况下，即便是 $M = 9$ 这样的“复杂”模型，也没有出现很严重的过拟合现象。在

	$M=0$	$M=1$	$M=3$	$M=9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

图 1-6: N=10 时不同阶数模型的参数比较

$N=100$ 时，$M=9$ 模型与真实模型已经非常接近了。因此，我们可以得到以下几个结论：

- 增加训练样本的容量可以抑制过拟合问题，从而提高泛化能力；
- 训练样本的容量越大，待训练模型的复杂度可以越复杂，或者说要相应地提高待训练模型的复杂度，使得两者的复杂度可以相称；
- 当训练样本的容量较大时，模型较不容易受到数据样本中噪声的干扰，较容易发现样本数据的内在规律，鲁棒性好；

1.3.3 正则项的意义

下面来看一下，不同阶数模型的参数值有什么特点，如图1-6所示。可以看出，随着阶数 M 的增大，参数的大小通常会变大，以适应模型的复杂度。例如，对于 $M=9$ 的模型，它的 10 个参数，除了参数 w_0^* 外，值都要大很多。这反映出，模型的复杂度与参数的个数和大小都是成正比的。这也意味着，有着更多参数的模型，参数值的自由度也更大，在过拟合的情况下，参数将会被过分地调节。在这种情况下，参数对数据点中的噪声也很敏感，模型的鲁棒性降低。有没有办法来抑制过拟合现象呢？

到目前为止，我们可以设想出 2 种解决方案。第 1 种方法是，限制参数的数量，例如让 M 取较小的值 $M=3$。第 2 种方法是，在模型复杂度不变的情况下，即模型参数个数不变的情况下，例如维持 $M=9$ 不变，限制参数值的大小，这就

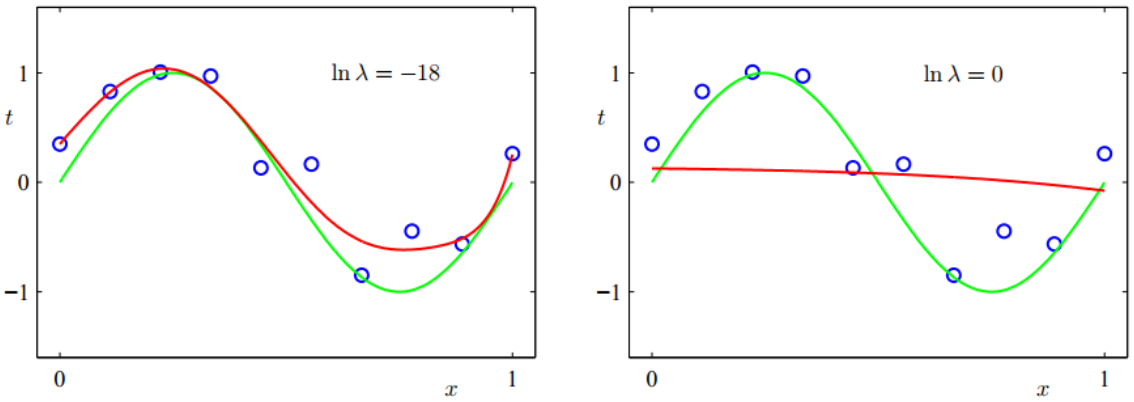


图 1-7: N=10 M=9 不同权重 λ 的拟合结果

是正则化技术，它相当于给误差函数增加一个惩罚项，使得参数不能达到太高的数值：

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{f(x_n, \boldsymbol{w}) - y_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \tag{18}$$

其中，$\|\boldsymbol{w}\|^2 \equiv \boldsymbol{w}^T\boldsymbol{w} = w_0^2 + w_1^2 + \ldots + w_M^2$，参数 λ 用于控制经验函数与正则化项的权重。通常，参数 w_0 要从正则项中去掉，否则会使得模型依赖于目标变量原点的选择。如果确实需要将参数 w_0 包括进来，那么可以给它单独配置一个正则化参数。在其它的领域，正则化还有另外几个名称，例如统计学中的收缩（Shrinkage）、神经网络中的权值衰减（Weight Decay）等。

上述对正则项作用的解释相当直观与感性。实际上，正则项有着含义非常明显的理论解释。我们将会看到，寻找模型参数的最小化平方损失函数的方法代表了应用广泛的最大似然估计的一种特殊情形，并且过拟合问题是最大似然估计的内在缺陷。而基于贝叶斯定理的最大后验估计可以“天然地”抑制过拟合问题，它本身就具有正则项。

如图1-7所示，展示了 $M = 9$ 模型在数据容量为 $N = 10$ 时的曲线拟合效果。需要注意的是，在 $\ln\lambda = 0$ 的情况下，参数值过于被限制在很小的数值范围内，模型的拟合能力与泛化能力都受到了很大的影响。

图1-8展示了 $N = 10$ 时不同正则项权重对参数值的影响，其中 $\ln\lambda = -\infty$ 对应于没有正则项的模型。可以看出，随着 λ 的增大，参数值的大小逐渐变小，模型的复杂度逐渐降低。

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

图 1-8: N=10 时不同正则项权重的参数比较

1.3.4 验证数据集的引入

在前面的讨论中，我们已经知道，如果误差函数使用平方损失函数，一般情况下，我们需要给它添加一个正则项，并且，在优化之前，还需要确定合适的模型参数个数。

为了确定模型参数的个数，就需要在训练模型的过程中，既要监测训练误差曲线，又要监测泛化误差曲线，以便在出现过拟合现象之后，可以及时调整模型的参数。

前面的做法是，使用训练数据集训练模型，使用测试数据集测试模型的泛化能力且同时要根据泛化误差曲线来调节模型的参数（模型本身的参数以及正则项的权重参数）。为了简化第 2 部分的工作，通常从测试数据集中分离出一部分数据作为验证数据集（Validation Data Set）[13]，专门用于优化模型的复杂度和正则项权重等参数。

下面给出三种数据集的作用：

- 训练数据集：用于训练模型；
- 验证数据集：用于过拟合的判断及早期停止（Early Stop），如果模型在训练数据集上的精度在提高，但是在验证数据集上的精度维持不变或降低，表明

[13]需要注意的是，在许多情况下，专门分离出验证数据集会浪费有价值的训练数据，可以考虑使用更好的方法进行处理，以便让数据集最大限度地发挥作用，例如采用各种交叉验证方法提高数据的利用率。

过拟合发生了，应该停止训练。本数据的另一个用途是，用于调整模型的超参数，例如学习率、模型本身的参数个数、正则项的权重参数等；

- 测试数据集：用于最终测试模型的泛化能力；

1.4 模型学习的概率解释

1.4.1 频率学派与贝叶斯学派

在现实世界中，存在着许多不确定的事件。例如，抛出一枚硬币，有时正面朝上，有时背面朝上；从一个盛满各种水果的箱子中取出水果，有时取出苹果，有时取出橘子等。这些事件有几个共同的特点：事件本身可以重复多次；事件发生具有一定的随机性；随着重复次数的增多，各个事件发生的频率似乎总是接近某个常数。

还有一类不确定的事件，它们无法重复，或者说，无法使用发生的频率来定量地描述。例如，2050 年之前北极冰盖是否会消失或消失的概率是多少；某个学生考上重点高中的概率是多少；在零下结冰的条件下，已经使用 10 年以上的某品牌汽车无法启动的概率是多少。在这些情况下，我们也希望能够定量地描述它们的不确定性，并且也允许根据少量的新证据对不确定性进行精确的定量修改，为最终的决策提供有效的依据。

实际上，上述两种不确定性的观点，代表了两类不同的概率学派：频率学派和贝叶斯学派，它们是统计学中的两大学派，在统计推断/推理（Inference）的方法上各有不同。

频率学派（Frequentist），也被称为经典概率（Classical Probability）。该学派认为概率是事件在长时间内发生的频率，是事件发生可能性的客观定量的表示。对许多事件来说，这种以频率解释概率的方式是有内在逻辑的，但对某些没有长期频率的事件来说，这种解释是难以理解的。

贝叶斯派（Bayesians），在另一方面，有更直观的解释方式，它把概率解释成对事件发生的信心或主观信念，或者认为概率是对某个知识或观点的定量描述。例如，某人把 0 赋给某个事件的发生概率，表明他完全确定此事不会发生；相反，如果赋予的概率值是 1，则表明他十分肯定此事一定会发生。

我们可以从频率学派和贝叶斯学派的长期争论历程中去了解两个学派的方法、观点以及统计推断的相关知识:

- 频率学派

  20 世纪初期建立，整个 20 世纪基本主宰了统计学，代表人 Fisher、Karl Pearson、Neyman，Fisher 提出最大似然估计方法（Maximum Likelihood Estimation，MLE）和多种抽样分布，Karl Pearson 提出 Pearson 卡方检验、Pearson 相关系数，Neyman 提出了置信区间的概念，同 Karl Pearson 的儿子 E.S.Pearson 一起提出了假设检验的 Neyman-Pearson 引理；

- 贝叶斯学派

  20 世纪 30 年代建立，快速发展于 20 世纪 50 年代（计算机诞生后），它的理论基础由 17 世纪的贝叶斯（Bayes）提出，他提出了贝叶斯公式，也称贝叶斯定理，贝叶斯规则。贝叶斯方法经过高斯（Gauss）和拉普拉斯（Laplace）的发展，在 19 世纪主宰了统计学。

  历史上，虽然贝叶斯学派起源于 18 世纪，但是贝叶斯学派一直沉寂的主要原因在于，后验概率的计算比较复杂，尤其是需要在整个参数空间求和或求积分。取样方法的发展（例如 MCMC 方法）以及计算机速度和存储容量的巨大提升，使得运用最大后验方法构建各种复杂模型成为可能。Monte Carlo 方法非常灵活，可以应用于许多种类的模型，但是，它们在计算上还是很复杂，主要应用于小规模问题。最近，许多高效的判别方法被提出来了（例如 Variational Bayes）和期望传播（Expectation Propagation）。这些方法作为补充的采样方法，允许贝叶斯方法应用于大规模的应用中。

具体来说，两种学派的主要差别在于对参数空间的认知上。频率学派认为，存在唯一的、真实的常数参数，观察数据都是在这个参数下产生的，因为不知道参数到底取何值，所以就引入了最大似然估计和置信区间（Confidence Interval），以便找出参数空间中最可能的参数值。而贝叶斯学派则认为，参数本身存在一个概率分布，并没有唯一真实的参数，参数空间里的每个值都可能是真实模型中使用的参数（概率不同），因而引入了先验分布（Prior Distribution）和后验分布（Posterior Distribution），以便找出参数空间中每个参数值的概率，并由此提出最大后验估计方法。最大后验估计会引入先验的（主观）知识，并且在得到新观察数据之后，也会更新先前得到的后验概率。因此，从认知的角度来说，这种方法更加符合人们对现实世界的认知过程。

关于频率学派和贝叶斯学派的相对优势，存在很多争论。事实上，并不存在所谓纯粹的频率学派观点或者贝叶斯学派观点。例如，针对贝叶斯学派的一种广泛批评是，先验的选择通常考虑的是计算的便利性而不是从问题的先验知识方面来考虑。某些人甚至把先验的依赖看作是求解问题困难的来源，减少对于先验的依赖性是所谓无信息（Noninformative）先验的一个研究动机。然而，这会导致模型比较上的困难，且当先验选择不好的时候，会导致贝叶斯方法有很大的可能性给出错误的结果。频率学派的估计方法在一定程度上避免了这一问题，并且诸如交叉验证这样的技术在模型比较等方面也很有用。

1.4.2 最大似然估计与最大后验估计的引入

前面的多项式拟合例子，是一个典型的不确定性问题。频率学派把样本数据 $\mathcal{D}$ 当作随机值，模型参数 $\boldsymbol{w}$ 当作常数，并应用最大似然估计求解出最可能生成这些数据的模型参数。而从贝叶斯学派的角度来看，模型参数 $\boldsymbol{w}$ 本身应该具有不确定性，在观察到数据 $\mathcal{D}$ 之前，我们可以有关于模型参数 $\boldsymbol{w}$ 的假设（以先验概率 $p(\boldsymbol{w})$ 的形式给出）；在观察到新数据 $\mathcal{D}$ 之后（以条件概率 $p(\mathcal{D}|\boldsymbol{w})$ 的形式给出），通过贝叶斯定理更新参数 $\boldsymbol{w}$ 的不确定性（以后验概率 $p(\boldsymbol{w}|\mathcal{D})$ 的形式表达），即贝叶斯学派每次在观察到新数据时需要完整地执行一次贝叶斯定理：

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})} \tag{19}$$

其中，$p(\mathcal{D}|\boldsymbol{w})$ 由观测数据集 $\mathcal{D}$ 来估计，可以被看作参数 $\boldsymbol{w}$ 的函数，被称为似然函数（Likelihood Function）[14]——它表达了在不同的参数 $\boldsymbol{w}$ 下，观测数据 $\mathcal{D}$ 出现的可能性大小。从频率学派的角度看，最大化求解似然函数 $p(\mathcal{D}|\boldsymbol{w})$ 并得到最符合观测数据 $\mathcal{D}$ 的模型参数是一件非常合理的事情。然而，前面已经描述的事实以及后文将要证明的结论表明，最大似然估计存在固有的缺陷——过拟合问题，而贝叶斯学派谋求最大化后验概率 $p(\boldsymbol{w}|\mathcal{D})$（最大后验估计），这种方法具有天然的抑制过拟合问题的结构——正则项。

显然，在频率学派和贝叶斯学派的观点中，似然函数 $p(\mathcal{D}|\boldsymbol{w})$ 都起着重要的作用。然而，它的使用方式有着本质的不同。在频率学派看来，$\boldsymbol{w}$ 被认为是固定的参数，它需要某种形式的损失函数来确定——可以通过考察（可能的）观测数据

[14]似然函数不是关于 $\boldsymbol{w}$ 的概率分布，并且它关于 $\boldsymbol{w}$ 的积分不一定等于 1。

集 $\mathcal{D}$ 的概率分布来得到[15]。相反，在贝叶斯学派看来，参数 $\boldsymbol{w}$ 是不确定的，需要通过 $\boldsymbol{w}$ 的概率分布来表达，并且在观测到新数据时需要更新。

下面，使用另一种方式来表达贝叶斯定理：

$$\text{posterior} \propto \text{ likelihood } \times \text{prior} \tag{20}$$

或者：

$$p(\boldsymbol{w}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) \tag{21}$$

其中，每个量都可以看作是 $\boldsymbol{w}$ 的函数。之所以采用这种表达方式，原因在于，公式（19）的分母是一个归一化常数。实际上，对公式（19）的两端关于 $\boldsymbol{w}$ 积分，可以使用先验概率分布和似然函数来表达该分母：

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})\mathrm{d}\boldsymbol{w} \tag{22}$$

贝叶斯学派将先验概率考虑进来，这是非常合理的。例如，在投掷硬币的实验中，假设每次都是正面朝上。对于频率学派来讲，最大似然估计将会得出硬币正面朝上的概率为 1，即在所有未来的投掷中，硬币总是正面朝上！而对于贝叶斯学派来讲，带有合理先验假设的最大后验估计不会得出如此极端的结论。

1.4.3 高斯分布及其性质

对于一元实值变量 x，高斯分布（Gaussian Distribution）或正态分布（Normal Distribution）被定义为：

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{23}$$

高斯分布由两个参数控制：均值（Mean）μ 和方差（Variance）σ^2。图1-9展示了一元高斯分布的曲线。

方差的平方根，即 σ，被称为标准差（Standard Deviation）；方差的倒数，记作 $\beta = \frac{1}{\sigma^2}$，被称为精度（Precision），它们都有一定的意义。

[15]可以使用常用的损失函数来度量，例如（没有正则项的）平方误差函数等。可以将似然函数的负对数作为误差函数（被称为对数似然误差函数），因而，最大化似然函数等价于最小化对数似然误差函数。后文将证明，在假设数据样本服从高斯分布的情况下，最小化对数似然误差函数就等价于最小化平方误差函数——这是一种常见的选取模型的误差函数或目标函数。

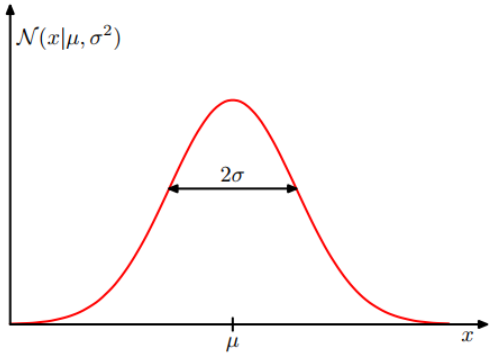


图 1-9: 一元高斯分布的曲线

高斯分布满足：

$$\begin{aligned}\mathcal{N}(x|\mu,\sigma^2) &> 0\\ \int_{-\infty}^{\infty}\mathcal{N}(x|\mu,\sigma^2)\mathrm{d}x &= 1\end{aligned} \tag{24}$$

因此，高斯分布满足概率密度函数的 2 个要求。

高斯分布关于 x 的平均值为：

$$\mathbb{E}[x] = \int_{-\infty}^{\infty}\mathcal{N}(x|\mu,\sigma^2)x\mathrm{d}x = \mu \tag{25}$$

也就是高斯分布的均值参数 μ。类似地，二阶矩为：

$$\mathbb{E}\left[x^2\right] = \int_{-\infty}^{\infty}\mathcal{N}(x|\mu,\sigma^2)x^2\mathrm{d}x = \mu^2+\sigma^2 \tag{26}$$

x 的方差为：

$$\mathrm{var}[x] = \mathbb{E}\left[x^2\right] - \mathbb{E}[x]^2 = \sigma^2 \tag{27}$$

也就是高斯分布的方差参数 σ^2。概率密度函数的最大值所对应的 x 值被称为众数（Mode），高斯分布的众数恰好等于均值[16]。

D 维向量 $\mathbf{x}$ 的高斯分布定义为：

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\} \tag{28}$$

其中 D 维向量 $\boldsymbol{\mu}$ 被称为均值向量，$D\times D$ 矩阵 $\boldsymbol{\Sigma}$ 被称为协方差矩阵，$|\boldsymbol{\Sigma}|$ 表示 $\boldsymbol{\Sigma}$ 的行列式。

[16]在给定的离散数据集中，众数是出现次数最多的那个值（可能有多个）。与众数相关的概念还有均值（Mean）和中位数（Median）。均值定义已经在上面给出了，关于中位数的定义，请阅读相关资料，例如Wikipedia: Median。

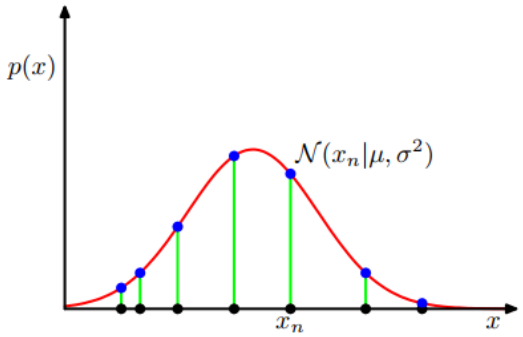


图 1-10: 一元高斯分布的似然函数

1.4.4 基于高斯分布的似然函数及最大化

现在，假设给定一个样本数据集 $\mathrm{X} = (x_1, \ldots, x_N)^T$，它是对标量变量 x 的 N 次观测，并且假定各次观测是从高斯分布中独立抽取的[17]，高斯分布的均值 μ 和方差 σ^2 未知。现在的任务是，根据所提供的数据样本点，来确定高斯分布的这 2 个参数。

此处，使用最大似然方法。由于数据集中的样本数据是独立同分布的，那么可以给出数据集的联合概率：

$$p(\mathrm{X}|\mu,\sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(x_n|\mu,\sigma^2\right) \tag{29}$$

如果把它看作是 μ 和 σ^2 的函数，那么它实际上就是基于高斯分布的似然函数，图像如图1-10所示。

从图中可以看出，红色曲线表示高斯分布的似然函数，黑点表示数据集 $\{x_i\}$ 的值，似然函数对应于蓝点的值乘积，该似然函数的变量 μ 和 σ^2 未知，需要求解——最大化似然函数，这涉及到调节高斯分布的参数 μ 和 σ^2，使得蓝点的值乘积最大。

显然，最大似然方法，是在已知概率分布类型（例如本例中的高斯分布）的情况下，基于提供的数据样本点（例如本例中的数据集 X），找出能够使似然函数的概率值（例如本例中的高斯联合概率）最大化的概率分布参数（例如本例中的 μ 和 σ^2）。

[17]独立地从相同的数据分布中抽取样本点，被称为独立同分布（Independent and Identically, i.i.d）。

如何最大化高斯分布的似然函数呢？在实际应用中，考虑似然函数的对数形式更加方便，原因在于：

- 对数函数本身是一个单调递增函数，最大化原函数的对数，等价于最大化原函数；
- 取对数不仅可以简化数学分析，将累乘形式变换为累加形式，而且也有助于数值计算的正确性，因为大量小概率的乘积很容易下溢。

于是，高斯分布的似然函数（29）可以写成：

$$\ln p(\mathrm{X}|\mu,\sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi) \tag{30}$$

首先，针对 μ，最大化函数（30），得到最大似然解：

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}x_n \tag{31}$$

实际上，它就是样本均值（Sample Mean），即观测值 $\{x_i\}$ 的均值，意义很明确。

然后，针对 σ^2，最大化函数（30），得到最大似然解：

$$\sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_{ML})^2 \tag{32}$$

实际上，它就是关于样本均值 μ_{ML} 的样本方差（Sample Variance），意义也很明确[18]。

一般情况下，在最大化函数的时候，需要同时考虑各个自变量。但是，在高斯分布的情况下，μ 与 σ^2 是无关的。因此，这 2 个阶段可以单独进行。

1.4.5 最大似然方法的局限性

值得注意的是，最大似然方法存在很大的局限性——它低估了分布的方差，对方差的估计产生了偏差（Bias）[19]。

[18]请注意，此处意指“样本方差”而不是高斯分布真正的方差，原因在于，样本方差是关于样本均值 μ_{ML} 的方差，而不是关于高斯分布的真实均值的方差。实际上，这是一种被称为偏差（Bias）的现象，该估计被称为有偏估计。而这正是最大似然方法的缺陷，它也是造成数据过拟合问题的直接原因。

[19]这种偏差与过拟合问题相关，即最大似然方法会导致过拟合问题。后文也将表明，为什么最大后验方法具有抑制过拟合的能力。

观察最大似然方法的求解结果，可以发现，它的解 μ_{ML} 和 σ^2_{ML} 都是数据集 x_1，$\cdots$，x_N 的函数。首先，给出 μ_{ML} 的数学期望：

$$\mathbb{E}\left(\mu_{ML}\right)=\mathbb{E}\left(\frac{1}{N}\sum_{n=1}^{N}x_n\right)=\frac{1}{N}\mathbb{E}\left(\sum_{n=1}^{N}x_n\right)=\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left(x_n\right)=\frac{1}{N}\cdot\left(N\cdot\mu\right)=\mu \tag{33}$$

然后，给出 σ^2_{ML} 的数学期望：

$$\begin{aligned}\mathbb{E}\left(\sigma^2_{ML}\right)&=\mathbb{E}\left(\frac{1}{N}\sum_{n=1}^{N}\left(x_n-\mu_{ML}\right)^2\right)=\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2-2x_n\mu_{ML}+\mu^2_{ML}\right)\right]\\&=\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2\right)-2\mu_{ML}\cdot\frac{1}{N}\sum_{n=1}^{N}x_n+\mu^2_{ML}\right]\\&=\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2\right)-2\mu_{ML}\cdot\mu_{ML}+\mu^2_{ML}\right]\\&=\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2\right)-\mu^2_{ML}\right]=\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left(x_n^2\right)-\mathbb{E}\left(\mu^2_{ML}\right)\end{aligned} \tag{34}$$

将如下两项代入上面的公式：

$$\begin{aligned}\mathbb{E}\left(x_n^2\right)&=\mu^2+\sigma^2\\\mathbb{E}\left(\mu^2_{ML}\right)=\left[\mathbb{E}\left(\mu_{ML}\right)\right]^2+\mathbb{D}\left(\mu_{ML}\right)&=\left[\mathbb{E}\left(\mu_{ML}\right)\right]^2+\mathbb{D}\left(\frac{1}{N}\sum_{n=1}^{N}x_n\right)\\&=\mu^2+\frac{1}{N^2}\sum_{n=1}^{N}\mathbb{D}\left(x_n\right)=\mu^2+\frac{1}{N}\sigma^2\end{aligned} \tag{35}$$

可得：

$$\mathbb{E}\left(\sigma^2_{ML}\right)=\frac{N-1}{N}\sigma^2 \tag{36}$$

可以看出，μ_{ML} 的均值是正确的均值，它是无偏估计，而 σ^2_{ML} 的均值被低估，它是有偏估计。

为什么 σ^2_{ML} 的均值（期望）会是有偏估计呢？原因在于，在公式（32）中，σ^2_{ML} 是关于样本均值 μ_{ML} 的方差，而不是关于高斯分布的真实均值 μ 的方差，因而就出现了偏差。为了消除这种偏差，可以使用下面的计算公式代替公式（32）：

$$\tilde{\sigma}^2=\frac{N}{N-1}\sigma^2_{ML}=\frac{1}{N-1}\sum_{n=1}^{N}\left(x_n-\mu_{ML}\right)^2 \tag{37}$$

当样本数据点的数量 N 增大时，这种偏差不是太严重，不使用矫正公式也是可以接受的。此外，当 $N\to\infty$ 时，方差的最大似然解与产生数据的高斯分布的真

实方差相等。但仍需注意的是，在有许多模型参数的复杂模型中，最大似然的偏差问题会变得更加严重，它也是导致多项式拟合中出现的过拟合问题的核心因素。

1.4.6 多项式曲线拟合问题的概率解释

在1.3节中，通过定义平方误差函数并进行优化求解，可以解决多项式曲线拟合问题。下面，我们将从概率的角度来解释其背后的原理，然后完全从贝叶斯的角度，使用最大后验方法进行方法上的改进——该方法中天然地隐藏着正则项。从概率的角度，这些解释能够让我们对优化中使用的误差函数以及正则化技术有一个更深刻更全面的认识。

多项式曲线拟合的目标是，在给定数据集 $\mathbf{D}=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$ 和多项式模型的情况下，从数据集 $\mathbf{D}$ 中学习该模型参数，并利用模型对新输入变量 x 进行预测，以获得某个对应的预测值 $f(x)$。

对目标变量 $f(x)$ 进行预测，预测值 $f(x)$ 本身具有一定的不确定性，我们可以使用某个概率分布来表达预测值的这种不确定性。假设目标变量 $f(x)$ 服从高斯分布，且分布的均值为公式（14）给出的多项式模型 $f(x,\boldsymbol{w})$，因此，可得：

$$p(f(x)|x,\boldsymbol{w},\beta)=\mathcal{N}(f(x)|f\left(x,\boldsymbol{w}\right),\beta^{-1}) \tag{38}$$

其中，$\beta=\frac{1}{\sigma^2}$，β 被称为精度参数。实际上，上式也是关于参数 $\boldsymbol{w}$ 和 β 的（单个数据样本点 x 的）似然函数。

现在，使用数据集 $\mathbf{D}$ 来确定未知参数 $\boldsymbol{w}$ 和 β 的最优值，定义似然函数如下：

$$p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)=\prod_{n=1}^{N}\mathcal{N}\left(f(x_n)|f\left(x_n,\boldsymbol{w}\right),\beta^{-1}\right) \tag{39}$$

为便于优化，使用对数似然函数，可得：

$$\ln p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)=-\frac{\beta}{2}\sum_{n=1}^{N}\left\{f\left(x_n,\boldsymbol{w}\right)-y_n\right\}^2+\frac{N}{2}\ln\beta-\frac{N}{2}\ln(2\pi) \tag{40}$$

观察到参数 $\boldsymbol{w}$ 和 β 之间是独立的，这意味着两个参数可以单独优化。首先，使用最大似然函数求解最优解 $\boldsymbol{w}_{ML}$。在公式（40）中，将 β 固定为常数：其右端最后两项与 $\boldsymbol{w}$ 无关，可以去掉；常数 $\frac{\beta}{2}$ 使用常数 $\frac{1}{2}$ 代替，不影响优化结果。然后考虑到，关于参数 $\boldsymbol{w}$ 最大化对数似然函数等价于最小化负对数似然函数。于是，将上述各种因素综合在一起，可得：

$$\min_{\boldsymbol{w}}-\ln p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)=\min_{\boldsymbol{w}}\frac{1}{2}\sum_{n=1}^{N}\left\{f\left(x_n,\boldsymbol{w}\right)-y_n\right\}^2 \tag{41}$$

上述公式似曾相识，它就是平方误差函数。这也解释了1.3节中使用平方误差函数获取模型最优参数的原因——在高斯噪声的假设下，使用平方误差函数训练或获取最优模型是一件非常自然的事情，它是最大化似然函数的结果！

1.4.7 多项式曲线拟合问题的改进

在参数 $\boldsymbol{w}_{ML}$ 确定之后，再对参数 β 进行优化，使用类似的方式可得：

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\left\{f\left(x_n, \boldsymbol{w}_{ML}\right) - y_n\right\}^2 \tag{42}$$

该公式与公式（32）一致。

现在，我们已经获得了 2 个模型：多项式模型 $f(x, \boldsymbol{w}_{ML})$，用来预测给定 x 处的函数值；高斯概率模型，均值是 $f(x, \boldsymbol{w}_{ML})$，方差是 $\frac{1}{\beta_{ML}}$，可以用来建模单个预测值 $f(x)$ 的预测分布（Predictive Distribution）[20]：

$$p(f(x)|x, \boldsymbol{w}_{ML}, \beta_{ML}) = \mathcal{N}(f(x)|f\left(x, \boldsymbol{w}_{ML}\right), \beta_{ML}^{-1}) \tag{43}$$

在1.3节中，多项式模型有直接的预测作用，而高斯概率模型究竟有什么作用呢？

在1.4.6节中，我们已经求解出了多项式模型的最优参数 $\boldsymbol{w}_{ML}$。这意味着，在多项式模型的阶数确定之后，多项式模型的参数是一个固定常数，这种近乎“武断地”确定模型参数的方式合理吗？难道我们按照过往的经验确定的多项式模型及基于该模型求解出的所谓“最优”模型参数，就一定符合给定数据样本点的内在规律吗？我们无法做出肯定的回答。为了能够反映出多项式模型参数 $\boldsymbol{w}$ 的这种不确定性，有必要构建 $\boldsymbol{w}$ 上的概率模型。因此，需要将 $\boldsymbol{w}$ 纳入概率分布的范畴加以考虑。这也与贝叶斯学派所倡导的概率原则是一致的。

再回顾一下贝叶斯定理，该定理描述了先验概率可以通过新证据[21]而转换成后验概率。从似然函数的角度来说，如果把先验概率看作模型参数的先验值，那么通过似然函数就可以将先验概率转换到后验概率——此时，后验概率可以看作观察到新数据后模型参数的后验值，这相当于更新了模型参数，使得概率模型越来越精确。

[20]这是一个带有高斯噪声的预测函数。

[21]可以理解为模型参数下观察到的新数据。

对于多项式曲线的拟合问题，可以按公式（20）确定贝叶斯定理，可得：

$$p(\boldsymbol{w}|\mathbf{D}, f(\mathbf{D}), \beta) \propto p(f(\mathbf{D})|\mathbf{D}, \boldsymbol{w}, \beta)p(\boldsymbol{w}) \tag{44}$$

其中 $p(f(\mathbf{D})|\mathbf{D}, \boldsymbol{w}, \beta)$ 是似然函数，$p(\boldsymbol{w})$ 是多项式的模型系数 $\boldsymbol{w}$ 的先验分布。

公式（44）非常明确地表明了多项式模型参数 $\boldsymbol{w}$ 也能够随着观察数据的更新而更新，这与现实情况完全相符。在上面的公式中，似然公式已经得到确定，只有多项式模型的参数 $\boldsymbol{w}$ 需要确定。简单起见，需要做出某种较为合理的假设，我们考虑下面的高斯分布：

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\mathbf{0}, \alpha^{-1}\boldsymbol{I}) = \left(\frac{\alpha}{2\pi}\right)^{\frac{M+1}{2}} \exp\left\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right\} \tag{45}$$

这是一个多元高斯分布函数，它的均值为 $\mathbf{0}$ 向量，协方差矩阵 $\boldsymbol{\Sigma} = \alpha^{-1}\boldsymbol{I}$，其中 α 表示分布的精度，$\boldsymbol{I}$ 是单位矩阵，M 表示多项式模型的阶数，其参数总共有 $M+1$ 个。

在引入多项式模型参数的高斯分布（45）后，多项式曲线拟合的后验概率公式（44）更新为：

$$p(\boldsymbol{w}|\mathbf{D}, f(\mathbf{D}), \alpha, \beta) \propto p(f(\mathbf{D})|\mathbf{D}, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha) \tag{46}$$

因此，在给定数据样本集的情况下，可以应用最大化后验概率的方法，来求解最优的多项式模型参数 $\boldsymbol{w}$（这种方法被称为最大后验方法）。我们设想，这样确定的模型参数 $\boldsymbol{w}$ 应该要比最大似然方法确定的模型参数合理些，事实证明，的确如此。取公式（46）的负对数，最大化后验概率变成最小化下面的公式：

$$\frac{\beta}{2}\sum_{n=1}^{N}\{f(x_n, \boldsymbol{w}) - y_n\}^2 + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{47}$$

对上式进行适当的变形，不会影响最优化结果：

$$\frac{1}{2}\sum_{n=1}^{N}\{f(x_n, \boldsymbol{w}) - y_n\}^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{48}$$

其中，$\lambda = \frac{\alpha}{\beta}$。上述公式也是似曾相识，它就是带正则项的平方误差函数。实际上，在高斯分布的假设下[22]，最大化后验概率等价于最小化具有正则项的平方误差函数，该正则项与生俱来！λ 是正则项的权重系数。

[22]总共有 2 个高斯分布，一个用于多项式模型的参数 $\boldsymbol{w}$，一个用于多项式模型的预测值 $f(x, \boldsymbol{w})$。

1.4.8 贝叶斯曲线拟合

虽然在上一节中使用了先验分布 $p(\boldsymbol{w}|\alpha)$，但是我们仍然在进行 $\boldsymbol{w}$ 的点估计，这并不是完整的贝叶斯方法——应该自始至终地应用概率求和规则和乘法规则。

在多项式曲线拟合的问题中，已知训练数据 $\mathbf{D}$，在提供新测试点 x 的情况下，需要预测 $f(x)$ 的值，即需要估计预测分布 $p(f(x)|x,\mathbf{D})$。注意，在此阶段，整个模型的 2 个参数 α 和 β 被认为是常数，从公式中省略掉。预测分布可以写成下面的形式：

$$\begin{aligned} p(f(x)|x,\mathbf{D}) &= \int p(f(x)|x,\boldsymbol{w},\mathbf{D})p(\boldsymbol{w}|x,\mathbf{D})\mathrm{d}\boldsymbol{w} \\ &= \int p(f(x)|x,\boldsymbol{w})p(\boldsymbol{w}|\mathbf{D})\mathrm{d}\boldsymbol{w} \end{aligned} \tag{49}$$

在上式中，第 1 项 $p(f(x)|x,\boldsymbol{w})$ 的化简，利用了 $\boldsymbol{w}$ 给定的情况下，$p(f(x)|x,\boldsymbol{w},\mathbf{D})$ 与数据集 $\mathbf{D}$ 之间的无关性；第 2 项 $p(\boldsymbol{w}|\mathbf{D})$ 的化简，利用了模型参数 $\boldsymbol{w}$ 与待预测数据点 x 之间的无关性。

在积分公式的第 1 项中，假设 $p(f(x)|x,\boldsymbol{w})$ 服从高斯分布，即：

$$p(f(x)|x,\boldsymbol{w}) = \mathcal{N}(f(x)|f\left(x,\boldsymbol{w}\right),\beta^{-1}) \tag{50}$$

积分公式的第 2 项是上一节中的后验概率。因此，对公式（49）进行变换，可得：

$$\begin{aligned} &p(f(x)|x,\mathbf{D}) = \int p(f(x)|x,\boldsymbol{w})p(\boldsymbol{w}|\mathbf{D})\mathrm{d}\boldsymbol{w} \\ &= \int \mathcal{N}(f(x)|f(x,\boldsymbol{w}),\beta^{-1})\frac{p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)p(\boldsymbol{w}|\alpha)}{\int p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)p(\boldsymbol{w}|\alpha)\mathrm{d}\mathbf{w}}\mathrm{d}\mathbf{w} \\ &= \int \mathcal{N}(f(x)|f(x,\boldsymbol{w}),\beta^{-1})\frac{\prod_{i:x_i\in\mathbf{D}}\mathcal{N}\left(f(x_i)|f\left(x_i,\boldsymbol{w}\right),\beta^{-1}\right)p(\boldsymbol{w}|\alpha)}{\int p(f(\mathbf{D})|\mathbf{D},\boldsymbol{w},\beta)p(\boldsymbol{w}|\alpha)\mathrm{d}\mathbf{w}}\mathrm{d}\mathbf{w} \\ &= c\int \prod_{i:x_i\in\overline{\mathbf{D}}}\mathcal{N}\left(f(x_i)|f\left(x_i,\boldsymbol{w}\right),\beta^{-1}\right)p(\mathbf{w}|\alpha)\mathrm{d}\mathbf{w} \end{aligned} \tag{51}$$

其中，$\overline{\mathbf{D}} = \mathbf{D}\cup\{x\}$。因此，预测分布由高斯分布的形式给出，可以通过解析的方式求解，最后得到：

$$p(f(x)|x,\mathbf{D}) = \mathcal{N}(f(x)|m(x),s^2(x)) \tag{52}$$

其中，均值和方差分别为：

$$\begin{aligned} m(x) &= \beta\boldsymbol{\phi}(x)^T\boldsymbol{S}\sum_{n=1}^{N}\boldsymbol{\phi}\left(x_n\right)y_n \\ s^2(x) &= \beta^{-1} + \boldsymbol{\phi}(x)^T\boldsymbol{S}\boldsymbol{\phi}(x) \end{aligned} \tag{53}$$

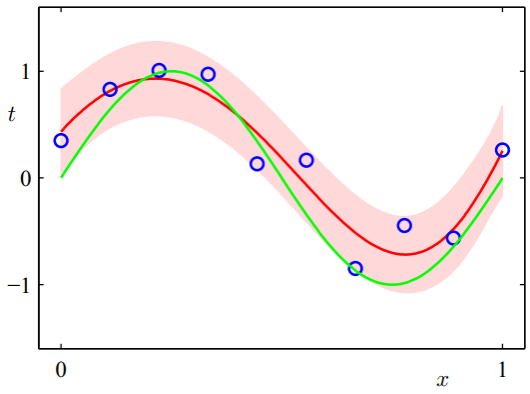


图 1-11: 贝叶斯曲线拟合的预测分布结果

其中，矩阵 $\boldsymbol{S}$ 由下式给出：

$$\boldsymbol{S}^{-1} = \alpha \boldsymbol{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}\left(x_n\right) \boldsymbol{\phi}\left(x_n\right)^T \tag{54}$$

其中，$\boldsymbol{I}$ 是 $(M+1) \times (M+1)$ 阶单位矩阵，M 是多项式的阶数，向量 $\boldsymbol{\phi}(x)$ 被定义为 $\phi_i(x) = x^i$，$i = 0, \cdots, M$。

从上面的公式可以看出，预测分布的均值与方差都依赖于待预测值 x。图1-11展示了贝叶斯曲线拟合的预测分布结果。其中，多项式阶数为 $M = 9$，超参数 $\alpha = 5 \times 10^{-3}$ 和 $\beta = 11.1$；红色曲线表示预测分布的均值，粉红色区域表示均值周围 ± 1 标准差的范围。

2 练习

1. 编程验证文中提到的多项式曲线拟合问题[23]：

   (a) 平方误差函数；

   (b) 带正则项的平方误差函数；

   (c) 贝叶斯曲线拟合；

[23]建议使用 Python 工具包：Scikit-Learn 和 Numpy。

3 附录

3.1 概率论的基础知识

求和规则（Sum Rule）：

$$p(X) = \sum_{Y} p(X, Y) \tag{55}$$

其中，$p(X)$ 被称为边缘概率（Marginal Probability），$p(X, Y)$ 被称为随机变量 X 和 Y 的联合概率（Joint Probability）。

乘积规则（Product Rule）：

$$p(X, Y) = p(Y|X)p(X) \tag{56}$$

其中，$p(Y|X)$ 被称为给定 X 的条件下 Y 的概率，也被简称为条件概率。

贝叶斯定理：

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{57}$$

再利用公式：

$$p(X) = \sum_{Y} p(X, Y) = \sum_{Y} p(X|Y)p(Y) \tag{58}$$

将贝叶斯定理变换为：

$$p(Y|X) = \frac{p(X|Y)p(Y)}{\sum_{Y} p(X|Y)p(Y)} \tag{59}$$

即贝叶斯定理中的分母部分可以使用分子中的项的和形式来表示，它的意义是确保贝叶斯定理左侧的条件概率对于所有的 Y 的取值之和为 1，我们也可以把贝叶斯定理的分母看作归一化常数。

随机变量 X 和 Y 具有相互独立性，如果下式成立：

$$p(X, Y) = p(X)p(Y) \tag{60}$$

如果 X 和 Y 相互独立，那么下式显然也成立：

$$p(Y|X) = p(Y) \tag{61}$$

如果 x 是连续型随机变量，那么 $x \in (a, b)$ 的概率由下式给出：

$$p(x \in (a, b)) = \int_{a}^{b} p(x)\mathrm{d}x \tag{62}$$

其中，$p(x)$ 被称为概率密度（Probability Density）函数。概率密度函数 $p(x)$ 必须满足以下 2 个条件：

$$\begin{aligned} p(x) &\geq 0 \\ \int_{-\infty}^{\infty} p(x)\mathrm{d}x &= 1 \end{aligned} \tag{63}$$

注意，如果 x 是一个离散变量，那么 $p(x)$ 有时也被叫做概率质量函数（Probability Mass Function），因为它可以被看作集中在合法的 x 值处的“概率质量”的集合。

位于区间 $(-\infty, z)$ 的 x 的概率由累积分布函数（Cumulative Distribution Function）表示：

$$P(z) = \int_{-\infty}^{z} p(x)\mathrm{d}x \tag{64}$$

累积分布函数与概率密度函数的关系如下：

$$P'(x) = p(x) \tag{65}$$

概率的求和规则和乘积规则以及贝叶斯规则，同样可以应用于概率密度函数，也可以应用于离散变量与连续变量相结合的情形。例如：

$$\begin{aligned} p(x) &= \int p(x, y)\mathrm{d}y \\ p(x, y) &= p(y|x)p(x) \end{aligned} \tag{66}$$

在概率分布 $p(x)$ 下，函数 $f(x)$ 的平均值被称为 $f(x)$ 的期望（Expectation），记作 $\mathbb{E}[f]$。对于离散型随机变量与连续型随机变量，它们的形式如下：

$$\begin{aligned} \mathbb{E}[f] &= \sum_{x} p(x)f(x) \\ \mathbb{E}[f] &= \int p(x)f(x)\mathrm{d}x \end{aligned} \tag{67}$$

在上述两种情形下，如果给定有限数量的 N 个采样点，这些点满足某个概率分布或概率密度函数，那么期望可以通过求和的方式进行估计：

$$\mathbb{E}[f] \simeq \frac{1}{N}\sum_{n=1}^{N} f\left(x_n\right) \tag{68}$$

当 $N \to \infty$ 时，该估计将会变得精确。在多变量的情形下，需要使用下标来表明被平均的是哪个变量，例如：

$$\mathbb{E}_x[f(x, y)] \tag{69}$$

注意，$\mathbb{E}_x[f(x,y)]$ 是关于 y 的函数。也可以考虑条件分布的条件期望（Conditional Expectation）：

$$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x) \tag{70}$$

$f(x)$ 的方差（Variance）被定义为：

$$\mathrm{var}[f] = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right] \tag{71}$$

它度量了 $f(x)$ 在均值 $\mathbb{E}[f(x)]$ 附近变化的大小。把平方项展开，它可以写成 $f(x)$ 和 $f(x)^2$ 的期望形式：

$$\mathrm{var}[f] = \mathbb{E}\left[f(x)^2\right] - \mathbb{E}[f(x)]^2 \tag{72}$$

特别地，可以考虑变量 x 自身的方差：

$$\mathrm{var}[x] = \mathbb{E}\left[x^2\right] - \mathbb{E}[x]^2 \tag{73}$$

对于两个随机变量 x 和 y，协方差（Covariance）被定义为：

$$\mathrm{cov}[x,y] = \mathbb{E}_{x,y}[\{x - \mathbb{E}[x]\}\{y - \mathbb{E}[y]\}] = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \tag{74}$$

协方差衡量了在多大程度上 x 和 y 会共同变化。如果 x 和 y 相互独立，那么它们的协方差为 0。

在两个随机向量 $\mathbf{x}$ 和 $\mathbf{y}$ 的情形下，协方差是一个矩阵：

$$\mathrm{cov}[\mathbf{x},\mathbf{y}] = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[\{\mathbf{x} - \mathbb{E}[\mathbf{x}]\}\left\{\mathbf{y}^T - \mathbb{E}\left[\mathbf{y}^T\right]\right\}\right] = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[\mathbf{x}\mathbf{y}^T\right] - \mathbb{E}[\mathbf{x}]\mathbb{E}\left[\mathbf{y}^T\right] \tag{75}$$

如果考虑的是向量 $\mathbf{x}$ 自身各个分量之间的协方差，那么符号可以简记为：$\mathrm{cov}[\mathbf{x}] \equiv \mathrm{cov}[\mathbf{x},\mathbf{x}]$。

3.2 内积、范数与角度

定义向量 $\mathbf{x},\mathbf{y} \in \mathbf{R}^n$ 的标准内积（Standard Inner Product）为：

$$\langle \mathbf{x},\mathbf{y} \rangle = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i \tag{76}$$

定义向量 $\mathbf{x} \in \mathbf{R}^n$ 的欧几里得范数（Euclidean Norm 或 ℓ_2-norm）为：

$$\|\mathbf{x}\|_2 = \left(\mathbf{x}^T\mathbf{x}\right)^{1/2} = \left(x_1^2 + \cdots + x_n^2\right)^{1/2} \tag{77}$$

对任意向量 $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$，Cauchy-Schwartz 不等式成立：

$$\left|\mathbf{x}^T\mathbf{y}\right| \leq \|\mathbf{x}\|_2\|\mathbf{y}\|_2 \tag{78}$$

对任意非零向量 $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$，它们之间的无符号角度定义为：

$$\angle(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}\right) \tag{79}$$

其中，$\cos^{-1}(u) \in [0, \pi]$；如果 $\mathbf{x}^T\mathbf{y} = 0$，那么称向量 $\mathbf{x}$ 与 $\mathbf{y}$ 是正交的。

定义矩阵 $X, Y \in \mathbf{R}^{m\times n}$ 的标准内积为：

$$\langle X, Y\rangle = \mathrm{tr}\left(X^TY\right) = \sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}Y_{ij} \tag{80}$$

其中，tr 表示矩阵的迹，即矩阵的主对角线上元素的和。实际上，2 个 $\mathbf{R}^{m\times n}$ 矩阵的内积等价于 2 个相应的 $\mathbf{R}^{mn}$ 向量的内积。

定义矩阵 $X \in \mathbf{R}^{m\times n}$ 的 Frobenius 范数为：

$$\|X\|_F = \left(\mathrm{tr}\left(X^TX\right)\right)^{1/2} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2\right)^{1/2} \tag{81}$$

实际上，Frobenius 范数等价于矩阵向量化后的（向量）欧几里得范数[24]。

设对称的 $n \times n$ 矩阵的集合为 $\mathbf{S}^n$，定义 $X, Y \in \mathbf{S}^n$ 的标准内积为：

$$\langle X, Y\rangle = \mathrm{tr}(XY) = \sum_{i=1}^{n}\sum_{j=1}^{n} X_{ij}Y_{ij} = \sum_{i=1}^{n} X_{ii}Y_{ii} + 2\sum_{i<j} X_{ij}Y_{ij} \tag{82}$$

3.3 矩阵的基本运算

设矩阵 $\mathbf{A}, \mathbf{B} \in \mathbf{R}^{m\times n}$，则下面的公式成立：

$$\begin{aligned}(\mathbf{A}+\mathbf{B})^{\mathrm{T}} &= \mathbf{A}^{\mathrm{T}} + \mathbf{B}^{\mathrm{T}} \\ (\mathbf{AB})^{\mathrm{T}} &= \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\end{aligned} \tag{83}$$

设方阵 $\mathbf{A}, \mathbf{B} \in \mathbf{R}^{n\times n}$，$\mathbf{I}_n$ 为 n 阶单位方阵，方阵 $\mathbf{A}$ 的逆矩阵（如果存在）为 $\mathbf{A}^{-1}$，则 $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ 成立，并且：

$$\begin{aligned}\left(\mathbf{A}^{\mathrm{T}}\right)^{-1} &= \left(\mathbf{A}^{-1}\right)^{\mathrm{T}} \\ (\mathbf{AB})^{-1} &= \mathbf{B}^{-1}\mathbf{A}^{-1}\end{aligned} \tag{84}$$

[24]注意，Frobenius 范数与矩阵的 ℓ_2 范数不同。

设 $\mathbf{A}$ 为 n 阶方阵，它的迹为 $\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} \mathbf{A}_{ii}$，迹有如下性质：

$$\begin{aligned} \mathrm{tr}\left(\mathbf{A}^{\mathrm{T}}\right) &= \mathrm{tr}(\mathbf{A}) \\ \mathrm{tr}(\mathbf{A}+\mathbf{B}) &= \mathrm{tr}(\mathbf{A}) + \mathrm{tr}(\mathbf{B}) \\ \mathrm{tr}(\mathbf{AB}) &= \mathrm{tr}(\mathbf{BA}) \\ \mathrm{tr}(\mathbf{ABC}) &= \mathrm{tr}(\mathbf{BCA}) = \mathrm{tr}(\mathbf{CAB}) \end{aligned} \tag{85}$$

n 阶方阵 $\mathbf{A}$ 的行列式（Determinant）具有如下性质：

$$\begin{aligned} \det(c\mathbf{A}) &= c^n \det(\mathbf{A}) \\ \det\left(\mathbf{A}^{\mathrm{T}}\right) &= \det(\mathbf{A}) \\ \det(\mathbf{AB}) &= \det(\mathbf{A})\det(\mathbf{B}) \\ \det\left(\mathbf{A}^{-1}\right) &= \det(\mathbf{A})^{-1} \\ \det\left(\mathbf{A}^{n}\right) &= \det(\mathbf{A})^{n} \end{aligned} \tag{86}$$

3.4 向量与矩阵的导数

设向量 $\mathbf{a}$ 和标量 x，它们之间的导数（Derivative）都是向量：

$$\begin{aligned} \left(\frac{\partial \boldsymbol{a}}{\partial x}\right)_i &= \frac{\partial a_i}{\partial x} \\ \left(\frac{\partial x}{\partial \boldsymbol{a}}\right)_i &= \frac{\partial x}{\partial a_i} \end{aligned} \tag{87}$$

设矩阵 $\mathbf{A}$ 和标量 x，它们之间的导数都是矩阵：

$$\begin{aligned} \left(\frac{\partial \mathbf{A}}{\partial x}\right)_{ij} &= \frac{\partial A_{ij}}{\partial x} \\ \left(\frac{\partial x}{\partial \mathbf{A}}\right)_{ij} &= \frac{\partial x}{\partial A_{ij}} \end{aligned} \tag{88}$$

设函数 $f(\mathbf{x})$ 和向量 $\mathbf{x}$，并假设函数 $f(\mathbf{x})$ 对向量 $\mathbf{x}$ 的元素可导，则函数 $f(\mathbf{x})$ 对向量 $\mathbf{x}$ 的一阶导数是向量：

$$(\nabla f(\boldsymbol{x}))_i = \frac{\partial f(\boldsymbol{x})}{\partial x_i} \tag{89}$$

如果相应的二阶导数存在，则二阶导数被称为 Hessian 矩阵或方阵：

$$\left(\nabla^2 f(\boldsymbol{x})\right)_{ij} = \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \tag{90}$$

设向量 $\mathbf{x}$ 和常向量 $\mathbf{a}$，矩阵 $\mathbf{A}$ 和矩阵 $\mathbf{B}$，向量和矩阵的导数满足乘法规则：

$$\begin{aligned} \frac{\partial \boldsymbol{x}^{\mathrm{T}}\boldsymbol{a}}{\partial \boldsymbol{x}} &= \frac{\partial \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a} \\ \frac{\partial \mathbf{AB}}{\partial \boldsymbol{x}} &= \frac{\partial \mathbf{A}}{\partial \boldsymbol{x}}\mathbf{B} + \mathbf{A}\frac{\partial \mathbf{B}}{\partial \boldsymbol{x}} \end{aligned} \tag{91}$$

设标量 x，由 $\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$ 及公式 (91)，逆矩阵 $\mathbf{A}^{-1}$ 关于标量 x 的导数为：

$$\frac{\partial \mathbf{A}^{-1}}{\partial x}=-\mathbf{A}^{-1}\frac{\partial \mathbf{A}}{\partial x}\mathbf{A}^{-1} \tag{92}$$

设矩阵 $\mathbf{A}$ 和矩阵 $\mathbf{B}$，有：

$$\begin{aligned}\frac{\partial \operatorname{tr}(\mathbf{AB})}{\partial A_{ij}}&=B_{ji}\\ \frac{\partial \operatorname{tr}(\mathbf{AB})}{\partial \mathbf{A}}&=\mathbf{B}^{\mathrm{T}}\end{aligned} \tag{93}$$

进一步有：

$$\begin{aligned}\frac{\partial \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{B}\right)}{\partial \mathbf{A}}&=\mathbf{B}\\ \frac{\partial \operatorname{tr}(\mathbf{A})}{\partial \mathbf{A}}&=\mathbf{I}\\ \frac{\partial \operatorname{tr}\left(\mathbf{ABA}^{\mathrm{T}}\right)}{\partial \mathbf{A}}&=\mathbf{A}\left(\mathbf{B}+\mathbf{B}^{\mathrm{T}}\right)\end{aligned} \tag{94}$$

由公式 (81) 和公式 (94) 可以得出：

$$\frac{\partial \|\mathbf{A}\|_F^2}{\partial \mathbf{A}}=\frac{\partial \operatorname{tr}\left(\mathbf{AA}^{\mathrm{T}}\right)}{\partial \mathbf{A}}=2\mathbf{A} \tag{95}$$

3.5 特征值分解

特征分解（Eigen Decomposition），又被称为谱分解（Spectral Decomposition），指的是将矩阵分解为特征值和特征向量表示的矩阵乘积的方法。

设方阵 $\mathbf{A}\in\mathbf{R}^{n\times n}$，其特征向量方程定义为：

$$\mathbf{A}\mathbf{u}_i=\lambda_i\mathbf{u}_i \tag{96}$$

其中，$i=1,\cdots,n$，$\mathbf{u}_i$ 被称为特征（列）向量（Eigenvector），标量 λ_i 被称为 $\mathbf{u}_i$ 的特征值（Eigenvalue）。

公式（96）也可以写成：

$$\left(\mathbf{A}-\lambda_i\mathbf{I}\right)\mathbf{u}_i=\mathbf{0} \tag{97}$$

它有非零特征向量解 $\mathbf{u}_i$ 的充分必要条件是系数行列式：

$$|\mathbf{A}-\lambda_i\mathbf{I}|=0 \tag{98}$$

上式是以 λ_i 为未知数的一元 n 次方程，被称为方阵 $\mathbf{A}$ 的特征方程（Characteristic Equation）。令 $f(\lambda_i)=|\mathbf{A}-\lambda_i\mathbf{I}|$，$f(\lambda_i)$ 被称为方阵 $\mathbf{A}$ 的特征多项式。显然，特

征方程有 n 个解（若 λ_i 为实数，则 $\mathbf{u}_i$ 为实数解；否则，若 λ_i 为复数，则 $\mathbf{u}_i$ 为复数解），这些解形成了特征值的集合，也被称为谱（Spectrum）。

公式（96）的意义在于，对特征向量 $\mathbf{u}_i$ 进行线性变换 $\mathbf{A}$，相当于对特征向量在原方向直线上进行一次伸缩变换[25]。

在一类方阵中，我们特别感兴趣的是对称方阵，因为协方差矩阵、核矩阵、Hessian 矩阵等都是对称方阵。对称方阵的性质为 $A_{ij} = A_{ji}$ 或者等价地，$A = A^T$。对称方阵的逆矩阵也是对称的。

一般情况下，方阵的特征值可能是复数。但是，对于实对称方阵，可以证明，它的特征值一定是实数，并且若 $\lambda_i \neq \lambda_j$，那么它们对应的特征向量 $\mathbf{u}_i$ 和 $\mathbf{u}_j$ 必定正交[26]。

如果 $\lambda_i = \lambda_j$，那么 $\mathbf{u}_i$ 和 $\mathbf{u}_j$ 的线性组合 $\alpha\mathbf{u}_i + \beta\mathbf{u}_j$ 也是一个有着相同特征值的特征向量。于是，我们按如下方式选择 2 个特征向量：$\eta_1 = \mathbf{u}_i$ 和 $\eta_2 = \mathbf{u}_i + \beta\mathbf{u}_j$，然后对它们进行正交归一化处理[27]。

这意味着，我们总是可以为实对称方阵找到一组正交归一化的特征向量。由于有 n 个特征值，对应的 n 个特征向量就组成了一个完备集。因此，任意一个 n 维向量都可以表示为这些正交归一化特征向量的线性组合。

令方阵 $\mathbf{U}$ 是由所有特征(列)向量 $\mathbf{u}_i$ 组成的单位正交方阵：$\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n)$。可以证明，如果 $\mathbf{A}$ 为 n 阶实对称矩阵，则必有正交矩阵 $\mathbf{U}$，使得下式成立：

$$\mathbf{U^{-1}AU} = \mathbf{\Lambda} = \begin{pmatrix} \boldsymbol{\lambda_1} & & & \\ & \boldsymbol{\lambda_2} & & \\ & & \ddots & \\ & & & \boldsymbol{\lambda_n} \end{pmatrix} \tag{99}$$

其中，$\lambda_1, \lambda_2, \cdots, \lambda_n$ 为 $\mathbf{A}$ 的特征值。上式表明，方阵 $\mathbf{A}$ 与其特征值组成的对角方阵 $\mathbf{\Lambda}$ 互为相似矩阵，$\mathbf{U}^{-1}\mathbf{AU}$ 被称为方阵 $\mathbf{A}$ 的相似变换，$\mathbf{U}$ 被称为相似变换

[25] $\lambda_i > 0$ 表示原方向上正的缩放，$\lambda_i < 0$ 表示反方向上的缩放，$\lambda_i = 0$ 表示将特征向量完全压缩为 $\mathbf{0}$ 向量。若方阵 $\mathbf{A}$ 的特征值都大于 0，则称方阵 $\mathbf{A}$ 是正定的；若方阵 $\mathbf{A}$ 的特征值都小于 0，则称方阵 $\mathbf{A}$ 是负定的；若方阵 $\mathbf{A}$ 的特征值都大于等于 0，则称方阵 $\mathbf{A}$ 是半正定的。方阵 A 的秩（Rank）等于非零特征值的个数。

[26] 详细的证明，请参考相关资料，例如《线性代数》，同济大学数学教研室，高等教育出版社，1982 年 3 月第 1 版，P103-104。

[27] 详细的过程，请参考《线性代数》（同济大学数学教研室，高等教育出版社，1982 年 3 月第 1 版）P108。

矩阵。

另一方面，如果方阵 $\mathbf{A}$ 与方阵 $\mathbf{B}$ 相似，则它们的特征多项式相同，从而有相同的特征值。

需要注意到一个有趣的事实，正交方阵 $\mathbf{U}$ 的行向量也是单位正交的，即 $\mathbf{U}^{-1}=\mathbf{U}^T$ 和 $|\mathbf{U}|=1$。特征向量方程（96）也可以使用 $\mathbf{U}$ 来表示：

$$\mathbf{AU}=\mathbf{U}\boldsymbol{\Lambda} \tag{100}$$

假设对（列）向量 $\mathbf{x}$ 进行正交变换：$\mathbf{y}=\mathbf{U}\mathbf{x}$，则：$\|\mathbf{y}\|_2=\left(\mathbf{y}^T\mathbf{y}\right)^{1/2}=\left(\mathbf{x}^T\mathbf{U}^T\mathbf{U}\mathbf{x}\right)^{1/2}=\left(\mathbf{x}^T\mathbf{x}\right)^{1/2}=\|\mathbf{x}\|_2$。因此，正交变换不会改变向量的长度。在实际应用中，正交变换相当于对向量进行旋转或反射操作。

根据公式(100)，可得：

$$\mathbf{U}^{\boldsymbol{T}}\mathbf{AU}=\boldsymbol{\Lambda} \tag{101}$$

实际上，它与公式(99)一致，这被称为实对称方阵 $\mathbf{A}$ 的对角化（Diagonalised）。同样，也可以得到：

$$\begin{aligned}\mathbf{A}&=\mathbf{U}\Lambda\mathbf{U}^T\\ \mathbf{A}^{-1}&=\mathbf{U}\Lambda^{-1}\mathbf{U}^T\end{aligned} \tag{102}$$

最后，上述方程也可以写成：

$$\begin{aligned}\mathbf{A}&=\sum_{i=1}^{n}\lambda_i\mathbf{u}_i\mathbf{u}_i^T\\ \mathbf{A}^{-1}&=\sum_{i=1}^{n}\frac{1}{\lambda_i}\mathbf{u}_i\mathbf{u}_i^T\end{aligned} \tag{103}$$

对于方阵 $\mathbf{A}$，还可以得出如下结论：

$$\begin{aligned}|\mathbf{A}|&=\prod_{i=1}^{n}\lambda_i\\ \mathrm{Tr}(\mathbf{A})&=\sum_{i=1}^{n}\lambda_i\end{aligned} \tag{104}$$

方阵 $\mathbf{A}$ 被称为正定的（Positive Definite），记作 $\mathbf{A}\succ 0$，如果对于向量 $\mathbf{u}$ 的所有非零值都有 $\mathbf{u}^T\mathbf{Au}>0$。等价地，一个正定方阵的所有特征值都有 $\lambda_i>0$。注意，正定不同于所有元素都为正。例如，矩阵：

$$\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix} \tag{105}$$

的特征值为 $\lambda_1\backsimeq 5.37$ 且 $\lambda_2\backsimeq -0.37$。方阵被称为半正定的（Positive Semidefinite），如果对于 $\mathbf{u}$ 的所有值都有 $\mathbf{u}^T\mathbf{Au}\geq 0$，记作 $\mathbf{A}\succeq 0$，这等价于 $\lambda_i\geq 0$。

3.6 奇异值分解

假设矩阵 $\mathbf{A} \in \mathbf{R}^{m \times n}$，其中的元素全部属于实数域或复数域，那么它可以被分解为：

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{106}$$

其中，$\mathbf{U} \in \mathbf{R}^{m \times m}$ 是满足 $\mathbf{U}^*\mathbf{U} = \mathbf{I}$ 的 m 阶酉矩阵（Unitary Matrix）[28]；$\mathbf{\Sigma} \in \mathbf{R}^{m \times n}$ 是非负实数对角矩阵，其中 $(\mathbf{\Sigma})_{ii} = \sigma_i$ 且其它位置的元素均为 0，σ_i 为非负实数且满足 $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant 0$[29]；$\mathbf{V}^* \in \mathbf{R}^{n \times n}$ 为 $\mathbf{V}$ 的共轭转置，也是满足 $\mathbf{V}^*\mathbf{V} = \mathbf{I}$ 的 n 阶酉矩阵。

这种分解被称为矩阵 $\mathbf{A}$ 的奇异值分解，$\mathbf{\Sigma}$ 对角线上的元素 $(\mathbf{\Sigma})_{ii} = \sigma_i$ 即为矩阵 $\mathbf{A}$ 的奇异值。当矩阵 $\mathbf{A}$ 为实对称正定矩阵时，奇异值分解与特征值分解的结果相同。

$\mathbf{U}$ 的列向量 $\mathbf{u}_i \in \mathbf{R}^m$ 被称为 $\mathbf{A}$ 的左奇异向量（Left Singular Vector）[30]，$\mathbf{V}$ 的列向量 $\mathbf{v}_i \in \mathbf{R}^n$ 被称为 $\mathbf{A}$ 的右奇异向量（Right Singular Vector）[31]。矩阵 $\mathbf{A}$ 的秩等于非零奇异值的个数。

3.7 梯度下降法

梯度下降（Gradient Descent）或最速下降（Steepest Descent）是一种一阶无约束最优化迭代算法，它按照梯度的反方向进行迭代搜索，用来寻找函数的局部极小值。与之对应的是梯度上升（Gradient Ascent），它按照梯度的正方向进行迭代搜索，用来寻找函数的局部极大值。

一般而言，梯度下降所求解的无约束优化问题具有如下形式：

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x}) \tag{107}$$

其中，$f(\mathbf{x})$（$\mathbf{x} \in \mathbf{R}^n$）是具有一阶连续偏导数的函数，它被称为待优化函数或目标函数。

[28]如果复数矩阵 $\mathbf{U} \in \mathbf{R}^{n \times n}$ 满足：$\mathbf{U}^*\mathbf{U} = \mathbf{U}\mathbf{U}^* = \mathbf{I}$ 且 $|det(\mathbf{U})| = 1$，其中 $\mathbf{I}$ 为 n 阶单位矩阵，$\mathbf{U}^*$ 为 $\mathbf{U}$ 的共轭转置，则称 $\mathbf{U}$ 为酉矩阵，即当且仅当其共轭转置与其逆矩阵相等时：$\mathbf{U}^* = \mathbf{U}^{-1}$。若酉矩阵的元素都是实数，则酉矩阵就是单位正交矩阵。

[29]常将奇异值按降序排列，以确保 $\mathbf{\Sigma}$ 的唯一性。

[30]$\mathbf{U}$ 的列向量组成一套正交基向量，它们是 $\mathbf{A}\mathbf{A}^*$ 的特征向量。

[31]$\mathbf{V}$ 的列向量组成一套正交基向量，它们是 $\mathbf{A}^*\mathbf{A}$ 的特征向量。

基于梯度的方法，其核心思想是：如果函数 $f(\mathbf{x})$ 沿着梯度的正向 $\nabla f(\mathbf{x})$ 或反向 $-\nabla f(\mathbf{x})$ 调整自变量 $\mathbf{x}$，那么就能够确保函数值上升或下降，而且是以最陡的方式上升或下降。

梯度下降法按如下过程进行工作：它从自变量的某个随机初值 $\mathbf{x}^{(0)}$ 开始进行迭代，每次需要重新计算目标函数 $f(\mathbf{x}^{(t)})$ 的梯度 $\nabla f(\mathbf{x}^{(t)})$，并以此梯度的反向（负梯度方向）更新 $\mathbf{x}^{(t)}$ 的值，从而达到减少函数值的目的；随着迭代的进行，最终达到目标函数 $f(\mathbf{x})$ 的极小值或最小值[32]。

下面讨论梯度下降法为什么起作用。假设函数 $f(\mathbf{x})$ 的自变量形式为 $\mathbf{x} = (x_1, x_2)$，即函数 $f(\mathbf{x})$ 具有 2 个自变量，并假设函数 $f(\mathbf{x})$ 具有一阶连续偏导数，现在需要找出该函数的某个极小值。

由于函数 $f(\mathbf{x})$ 具有一阶连续偏导数，则可将函数 $f(\mathbf{x})$ 在 $\mathbf{x}$ 附近进行一阶泰勒展开，即对于函数 $f(\mathbf{x})$，可以选取较小的 Δx_1 和 Δx_2，使下式成立[33]：

$$\Delta f \approx \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2 \tag{108}$$

其中，$\frac{\partial f}{\partial x_1}$ 为函数 f 对 x_1 的偏导数，$\frac{\partial f}{\partial x_2}$ 为函数 f 对 x_2 的偏导数。从上式可以看出，只要我们采取适当的方式调整 Δx_1 和 Δx_2，使得 $\Delta f \leq 0$ 总是成立，那么函数 $f(\mathbf{x})$ 的新值将满足下式：

$$f(x_1 + \Delta x_1, x_2 + \Delta x_2) = f(x_1, x_2) + \Delta f \leq f(x_1, x_2) \tag{109}$$

显然，如果按上述方式不断地调整 Δx_1 和 Δx_2，那么函数 $f(\mathbf{x})$ 的值将以非递增的方式进行变化，并最终收敛于某个极小值。

下面简化一下上述记法。设：

$$\Delta\mathbf{x} = (\Delta x_1, \Delta x_2)^T \tag{110}$$

$$\nabla\mathbf{f} = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2})^T \tag{111}$$

其中，$\nabla\mathbf{f}$ 就是函数 $f(\mathbf{x})$ 的梯度。这样，公式（108）可以重新写成：

$$\Delta f \approx \nabla\mathbf{f} \cdot \Delta\mathbf{x} \tag{112}$$

[32]如果目标函数 $f(\mathbf{x})$ 是凸函数，取得最小值；否则，取得极小值。

[33]函数 $f(\mathbf{x})$ 在 $\mathbf{x}_0$ 附近的一阶泰勒展开式为：$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla\mathbf{f}_{\mathbf{x}_0}^T(\mathbf{x} - \mathbf{x}_0)$，即 $\Delta f = f(\mathbf{x}) - f(\mathbf{x}_0) \approx \nabla\mathbf{f}_{\mathbf{x}_0}^T(\mathbf{x} - \mathbf{x}_0) = \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2$。

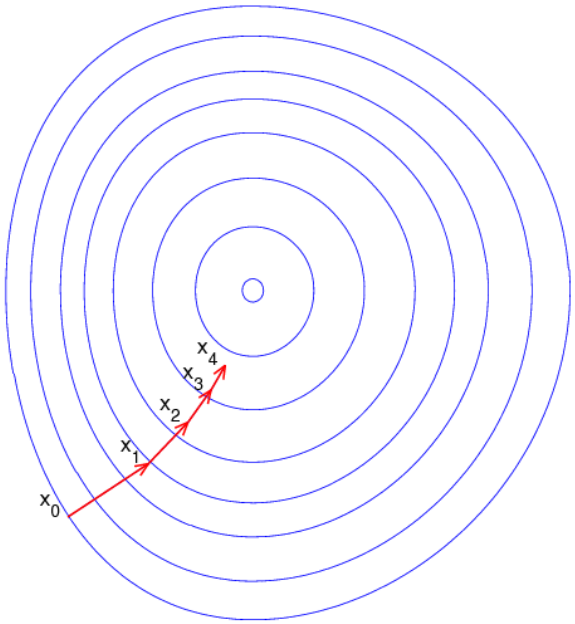


图 3-12: 梯度下降法示意图一

观察一下公式（112），可以发现，如果我们按如下方式设置 $\Delta\mathbf{x}$：

$$\Delta\mathbf{x} = -\eta\nabla\mathbf{f} \tag{113}$$

其中，η 是一个被称为学习率的小正数，那么再综合公式（112）和公式（113），可得：

$$\Delta f \approx -\eta\nabla\mathbf{f}\cdot\nabla\mathbf{f} = -\eta\|\nabla\mathbf{f}\|^2 \tag{114}$$

显然，上式确保了 $\Delta f \leq 0$。这意味着，如果连续地调整 Δx_1 和 Δx_2，函数 $f(\mathbf{x})$ 必将持续地以非递增的方式变化，最后将取得极小值。

实际上，我们得到了一个全新的方法——梯度下降法：

$$\Delta\mathbf{x} = -\eta\nabla\mathbf{f} \tag{115}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x} \tag{116}$$

上述调整过程一直迭代进行下去：计算梯度 $\nabla\mathbf{f}\rightarrow$ 计算 $\Delta\mathbf{x}\rightarrow$ 计算 $\mathbf{x}\rightarrow$ 计算函数值 f，直到迭代满足某个条件为止。上述公式被称为变量更新规则，自然地，它可以扩展到多变量情形。图3-12展示了梯度下降法的搜索示意图。

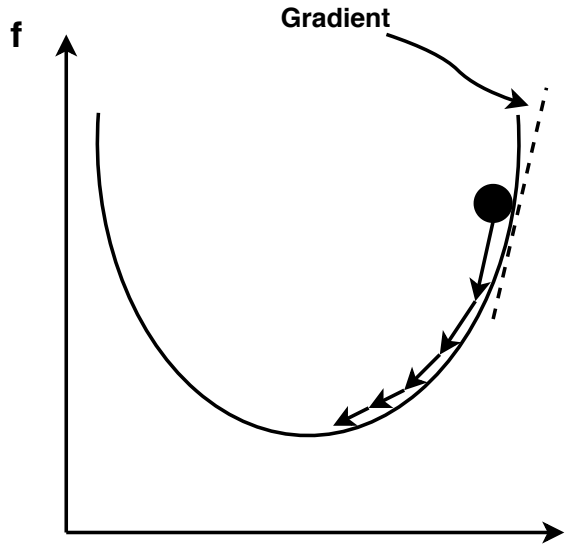


图 3-13: 梯度下降法示意图二

图3-12表示的是，将函数 $f(\mathbf{x})$（一个碗形曲面，具有 2 个自变量）投影到平面上（俯视图），蓝色曲线表示等高线，红色带箭头线段指向相应点处的梯度反方向[34]。可以看出，搜索沿着梯度下降的方向进行，并最终到达碗底，即函数 $f(\mathbf{x})$ 将取得全局最小值（因为该函数是一个凸函数）。图3-13展示了函数 $f(\mathbf{x})$ 的某个侧视图，并显示了该视角下梯度下降法的搜索示意图。

需要注意的是，如果函数 $f(\mathbf{x})$ 不是凸函数，则梯度下降法存在陷入局部极小值的问题，如图3-14所示，局部极小值或全局最小值的取得，与自变量的初始值有关。

在设计梯度下降算法时，还需要考虑学习率的设置问题——太大的学习率可能会导致搜索过程振荡，算法不收敛；太小的学习率可能会导致搜索时间过长，算法收敛慢，如图3-15和图3-16所示。

如果目标函数 $f(\mathbf{x})$ 满足一定的条件，则通过选取合适的学习率，就能够确保梯度下降算法收敛到极值。例如，如果函数 $f(\mathbf{x})$ 满足 L-Lipschitz 条件[35]，则将学习率设置为 $\frac{1}{2L}$，可确保算法收敛。

[34]梯度始终与相应点的等高线垂直。

[35]L-Lipschitz 条件指的是存在常数 L，使得 $\|\nabla f(\mathbf{x})\| \leq L, \forall \mathbf{x}$ 成立。

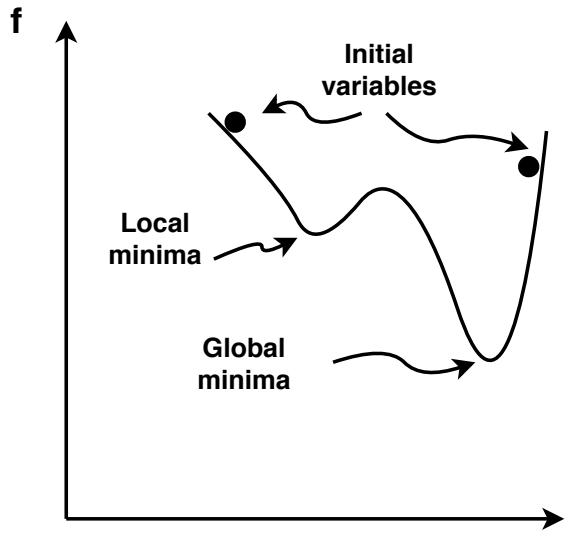


图 3-14: 局部极小值与全局最小值

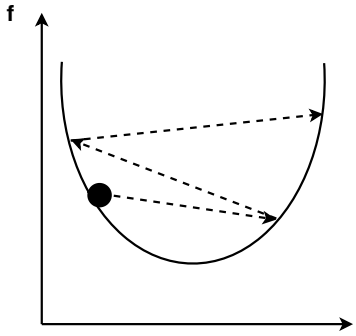


图 3-15: 学习率太大

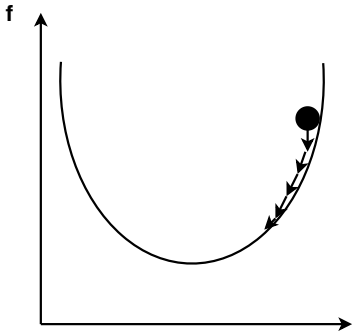


图 3-16: 学习率太小

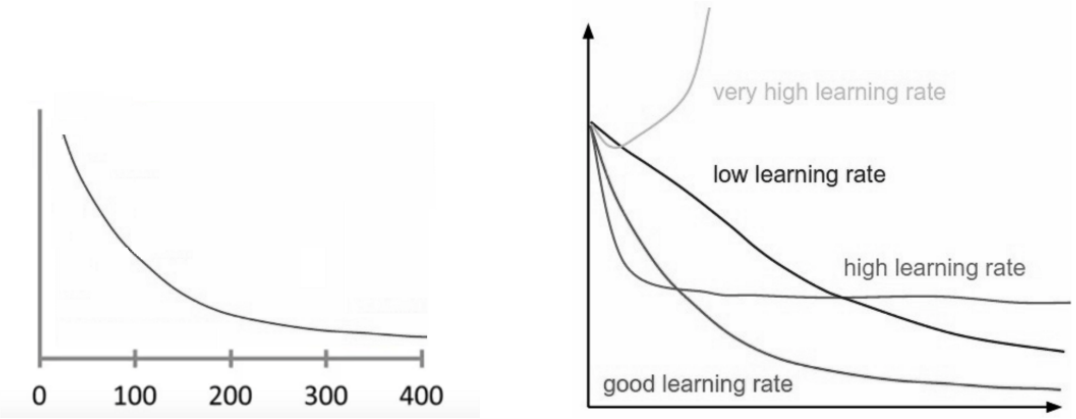


图 3-17: 根据函数值的下降曲线选取合适的学习率

在实际应用中，可以通过观察函数 $f(\mathbf{x})$ 值的下降曲线反复进行调整，以得到合适的学习率参数，如图3-17所示，横轴表示迭代次数，纵轴表示函数 $f(\mathbf{x})$ 值，左图表示合适的学习率产生的函数值下降曲线，右图将各种学习率产生的函数值下降曲线进行了对比。

在监督学习中，目标函数被定义为代价或误差函数。根据所使用的训练数据量，梯度下降法可以分为：

1. 批梯度下降（Batch Gradient Descent）

   它也被称为普通梯度下降（Vanilla Gradient Descent）——在为代价函数计算所有训练数据的误差梯度后，才更新自变量参数。

2. 随机梯度下降（Stochastic Gradient Descent，SGD）

   在计算每个训练数据的误差梯度后，立即更新自变量参数。

3. Mini-Batch 梯度下降（Mini-Batch Gradient Descent）

   这种方法是前面两种方法的折中与平衡，它将训练数据分割成若干份小批量数据，在每个小批量数据内执行批梯度下降算法。

上述算法分别具有不同的优缺点。批梯度下降算法在误差减少和收敛性方面具有稳定性，但是更新频率较低。随机梯度下降算法的更新频率高，但是对噪声敏感，

易于不稳定。Mini-Batch 梯度下降算法在它们之间取得平衡，兼具两者的优点，在实际应用中得到广泛的应用。

梯度下降算法具有如下几种变体形式：

1. 动量（Momentum）

   在进行本步更新时，将前一次更新也考虑进来。

2. AdaGrad

   Ada 代表自适应（Adaptive），表示学习率可以进行自适应的更新。

3. AdaDelta

   AdaDelta 对 AdaGrad 进行了改进，解决了全局学习率的定义及 AdaGrad 的早期停止问题。

4. RMSProp

   RMSProp 对 AdaGrad 进行了改进，通过滑动平均来统计梯度的平方，利用滑动平均值替换 AdaGrad 中梯度平方的累加和。

5. ADAM

   这种算法在 AdaGrad 基础上进行了改进——可以看作是在 AdaGrad 中引进了动量因素。

关于这几种算法的详细描述，请阅读相关资料。

从偏导数的阶次分类，可以分为以下 2 类：

1. 一阶方法

   上述方法都是一阶方法的典型代表。

2. 二阶方法

   二阶方法包括：牛顿法、拟牛顿法（存在多种算法，例如 DFP 算法、BFGS 和 L-BFGS 算法等）。

后文将详细描述相关二阶方法。

3.8 牛顿法

从原理上看，梯度下降法基于函数 $f(\mathbf{x})$ 的一阶泰勒展开式。而牛顿法（Newton Method）基于函数 $f(\mathbf{x})$ 的二阶泰勒展开式，它也是一种迭代算法，每一步需要计算目标函数的 Hessian 矩阵的逆矩阵，计算量大，但是收敛速度快，适用于自变量维度不高的应用场合。后文要介绍的拟牛顿法使用正定矩阵近似 Hessian 矩阵或逆矩阵，可以进行简化计算。

一般而言，所求解的无约束优化问题具有如下形式：

$$\min_{\mathbf{x}\in\mathbf{R}^n} f(\mathbf{x}) \tag{117}$$

其中，$f(\mathbf{x})$（$\mathbf{x}\in\mathbf{R}^n$）具有二阶连续偏导数。

无论是梯度下降法，还是牛顿法，它们都需要迭代计算自变量 $\mathbf{x}$。设 $\mathbf{x}^{(k)}$ 表示第 k 次迭代自变量的值。由于函数 $f(\mathbf{x})$ 具有二阶连续偏导数，则可将函数 $f(\mathbf{x})$ 在 $\mathbf{x}^{(k)}$ 附近进行二阶泰勒展开：

$$f(\mathbf{x}) \approx f\left(\mathbf{x}^{(k)}\right) + \nabla\mathbf{f}_{\mathbf{x}^{(k)}}^{\mathrm{T}}\left(\mathbf{x}-\mathbf{x}^{(k)}\right) + \frac{1}{2}\left(\mathbf{x}-\mathbf{x}^{(k)}\right)^{\mathrm{T}}\mathbf{H}_{\mathbf{x}^{(k)}}\left(\mathbf{x}-\mathbf{x}^{(k)}\right) \tag{118}$$

其中，$\nabla\mathbf{f}_{\mathbf{x}^{(k)}}$ 表示函数 $f(\mathbf{x})$ 在 $\mathbf{x}^{(k)}$ 处的梯度，$\mathbf{H}_{\mathbf{x}^{(k)}}$ 表示函数 $f(\mathbf{x})$ 在 $\mathbf{x}^{(k)}$ 处的 Hessian 矩阵。Hessian 矩阵被定义为：

$$\mathbf{H}_{\mathbf{x}} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{n\times n} \tag{119}$$

其中 x_i 和 x_j 分别表示 $\mathbf{x}$ 的第 i 和 j 个分量，向量 $\mathbf{x}$ 的维度为 n。可以看出，Hessian 矩阵是一个实对称方阵。

对于函数 $f(\mathbf{x})$ 而言，它取得极值的必要条件是在极值点处梯度向量为 $\mathbf{0}$ 向量，如果该点处的 Hessian 矩阵是正定矩阵，则函数 $f(\mathbf{x})$ 取得极小值。

下面推导出牛顿法的迭代公式。从公式（118）开始，设当前的迭代次数为第 $k+1$ 次，令 $\mathbf{x}=\mathbf{x}^{(k+1)}$，两端对 $\mathbf{x}^{(k+1)}$ 计算梯度，并令梯度为 $\mathbf{0}$[36]：

$$\nabla\mathbf{f}\left(\mathbf{x}^{(k+1)}\right) = 0 \tag{120}$$

可得（把 $\mathbf{x}^{(k)}$ 看作常量）：

$$\nabla\mathbf{f}\left(\mathbf{x}^{(k+1)}\right) = \nabla\mathbf{f}_{\mathbf{x}^{(k)}} + \mathbf{H}_{\mathbf{x}^{(k)}}\left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right) \tag{121}$$

[36]从下文可以看出，牛顿法实际上是在点 $\mathbf{x}^{(k)}$ 处使用一个二次函数（二阶泰勒展开式）来逼近函数 $f(\mathbf{x})$，并使用该二次近似函数的极值点作为下一次迭代的起点 $\mathbf{x}^{(k+1)}$。

最后得到：

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \nabla \mathbf{f}_{\mathbf{x}^{(k)}} \tag{122}$$

上式就是自变量 $\mathbf{x}$ 的迭代更新公式[37]，需要每次计算 Hessian 矩阵的逆矩阵。与一阶的梯度下降法相比，牛顿法属于二阶方法。显然，在公式（122）中，如果 $\mathbf{H}_{\mathbf{x}^{(k)}}^{-1} = \mathbf{I}$（单位矩阵），即算法没有有效地利用二阶偏导数信息，那么牛顿法退化为梯度下降法。

如果 Hessian 矩阵 $\mathbf{H}_{\mathbf{x}^{(k)}}$ 是正定的（其逆矩阵 $\mathbf{H}_{\mathbf{x}^{(k)}}^{-1}$ 也是正定的），那么可以确保牛顿法沿着函数值 $f(\mathbf{x})$ 下降的方向进行搜索，原因如下：

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \nabla \mathbf{f}_{\mathbf{x}^{(k)}} \tag{123}$$

其中，η 为学习率。这样，在公式（118）中，令 $\mathbf{x} = \mathbf{x}^{(k+1)}$，并将上式代入，可得近似值：

$$f\left(\mathbf{x}^{(k+1)}\right) \approx f\left(\mathbf{x}^{(k)}\right) - \eta \nabla \mathbf{f}_{\mathbf{x}^{(k)}}^{\mathrm{T}} \mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \nabla \mathbf{f}_{\mathbf{x}^{(k)}} \tag{124}$$

由于 $\mathbf{H}_{\mathbf{x}^{(k)}}$ 正定，故有 $\nabla \mathbf{f}_{\mathbf{x}^{(k)}}^{\mathrm{T}} \mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \nabla \mathbf{f}_{\mathbf{x}^{(k)}} > 0$。又由于学习率 η 是一个小的正数，故总有 $f\left(\mathbf{x}^{(k+1)}\right) < f\left(\mathbf{x}^{(k)}\right)$，即牛顿法总是沿着函数值 $f(\mathbf{x})$ 下降的方向进行搜索。

实际上，牛顿法与另一种近似求解方程解的方法——牛顿-拉弗森方法（Newton-Raphson method）有着非常密切的联系。

以单变量函数为例，设函数 $f(x)$ 为单变量函数，现在需要使用迭代法求解方程 $f(x) = 0$。迭代过程从某个初始值 x_0 开始。首先，计算相应的 $f(x_0)$ 和切线斜率 $f'(x_0)$，然后可以得到过点 $(x_0, f(x_0)$ 且斜率为 $f'(x_0)$ 的直线与 x 轴的交点 x_1：

$$0 = (x_1 - x_0) \cdot f'(x_0) + f(x_0) \tag{125}$$

解得：

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{126}$$

通常，x_1 比 x_0 更接近方程 $f(x) = 0$ 的解 x^*。类似地，再次以 x_1 为起点，开始下一轮迭代。实际上，x 的迭代公式为：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{127}$$

牛顿-拉弗森方法的迭代过程示意图如图3-18所示。

[37] 当然，还需要添加学习率因子 η，自变量 $\mathbf{x}$ 的迭代更新公式变为：$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \nabla \mathbf{f}_{\mathbf{x}^{(k)}}$。

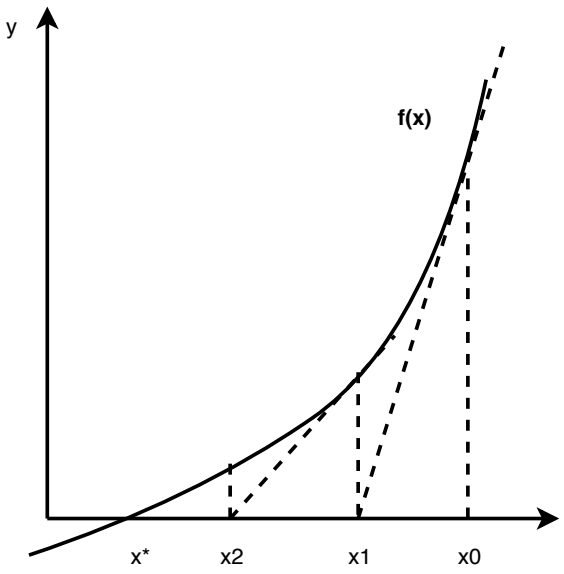


图 3-18: 牛顿-拉弗森方法求解方程根的示意图

牛顿-拉弗森方法也可以用于求函数的极值问题，这等价于求解方程 $f'(x)=0$，其迭代更新公式为：

$$x_{n+1}=x_n-\frac{f'(x_n)}{f''(x_n)} \tag{128}$$

如果函数 $f(\mathbf{x})$ 为多变量函数，那么上式等价于：

$$\mathbf{x}_{n+1}=\mathbf{x}_n-\mathbf{H}_{\mathbf{x}_n}^{-1}\nabla\mathbf{f}_{\mathbf{x}_n} \tag{129}$$

与前面推导出的牛顿法求解极值的迭代更新公式完全一样。

以单变量函数 $f(x)$ 为例，牛顿法的几何意义在于，它每次以一个二次函数在 x_n 附近去逼近函数 $f(x)$。设该二次函数的表达式为 ax^2+bx+c，则可依下列方程求解未知数 a、b 和 c：

$$\begin{aligned} ax_n^2+bx_n+c&=f(x_n)\\ 2ax_n+b&=f'(x_n)\\ 2a&=f''(x_n)\end{aligned} \tag{130}$$

然后，以二次函数 $ax_n^2+bx_n+c$ 的极值点作为下一次迭代的起点：

$$x_{n+1}=-\frac{b}{2a} \tag{131}$$

值得注意的是，如果函数 $f(x)$ 本身就是二次函数，那么只需要一次迭代就可以得到极值点。

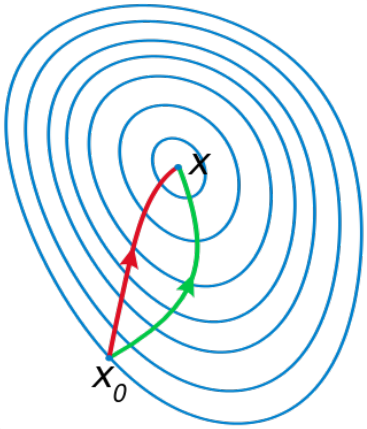


图 3-19: 牛顿法与梯度下降法的对比示意图

在实际应用中，对于低维函数 $f(\mathbf{x})$[38]，牛顿法的收敛速度要快于梯度下降法。图3-19展示了两种方法的对比示意图。图中，红色表示牛顿法的搜索路径，绿色表示梯度下降法的搜索路径。相比而言，牛顿法根据等高线的曲率选择了一条“捷径”。

需要注意的是，在实现牛顿法时，需要使用线搜索（Line Search）技术动态地确定学习率 η 的大小。常用的方法是，使用回溯线搜索（Backtracking Line Search）技术[39]。

3.9 拟牛顿法

在牛顿法中，需要计算 Hessian 矩阵的逆矩阵。当自变量的维度很大时，Hessian 矩阵也会很大，计算 Hessian 逆矩阵所消耗的时间与空间都很多。

为了减轻上述问题，避免直接计算 Hessian 逆矩阵，可以采取近似计算的方式进行处理。下面将要讨论的拟牛顿法（Quasi-Newton Method）正是为了解决上述问题而提出的。

拟牛顿法的基本思想是，从某个初始的正定矩阵开始，迭代地近似计算出 Hessian 矩阵或逆矩阵，然后应用于（自变量）参数的优化。

为了让优化过程能够按照预期的方向进行，例如，对于极小值问题，需要让搜索朝函数值减少的方向进行，必须满足牛顿法应用的几个基本条件：

[38]低维函数指的是自变量 $\mathbf{x}$ 的维度较低，或函数的自变量个数较少。

[39]请参考《Numerical Optimization》, Chapter 3, P35-p42. J. Nocedal, S.Wright, 1999。

- 拟牛顿条件

  从公式（121）可得：

$$\nabla \mathbf{f}\left(\mathbf{x}^{(k+1)}\right)-\nabla \mathbf{f}_{\mathbf{x}^{(k)}}=\mathbf{H}_{\mathbf{x}^{(k)}}\left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right) \tag{132}$$

  令 $\Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}=\nabla \mathbf{f}\left(\mathbf{x}^{(k+1)}\right)-\nabla \mathbf{f}_{\mathbf{x}^{(k)}}$ 和 $\Delta \mathbf{X}_{\mathbf{x}^{(k+1)}}=\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}$，则：

$$\Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}=\mathbf{H}_{\mathbf{x}^{(k)}} \Delta \mathbf{X}_{\mathbf{x}^{(k+1)}} \tag{133}$$

  或：

$$\mathbf{H}_{\mathbf{x}^{(k)}}^{-1} \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}=\Delta \mathbf{X}_{\mathbf{x}^{(k+1)}} \tag{134}$$

  上述 2 式，被称为拟牛顿条件，近似矩阵必须满足该条件。

- 近似矩阵必须是对称的

- 近似矩阵必须是正定的

最后两项条件是二阶方法规定的，以确保优化搜索过程能够朝着正确的方向进行。

根据拟牛顿条件，算法既可以选择近似 $\mathbf{H}_{\mathbf{x}^{(k)}}$，也可以选择近似 $\mathbf{H}_{\mathbf{x}^{(k)}}^{-1}$，这些算法统称为拟牛顿法。

3.9.1 DFP 算法

DFP（Davidon-Fletcher-Powell）算法对 $\mathbf{H}_{\mathbf{x}^{(k)}}^{-1}$ 进行近似计算。设近似矩阵为 $\mathbf{G}_k$，下面推导它的迭代更新公式：

$$\begin{aligned} \mathbf{G}_{k+1} &= \mathbf{G}_k+\Delta \mathbf{G}_k \\ \Delta \mathbf{G}_k &= m \mathbf{V} \mathbf{V}^T+n \mathbf{W} \mathbf{W}^T \end{aligned} \tag{135}$$

其中，m 和 n 均为实数，$\mathbf{V}$ 和 $\mathbf{W}$ 均为列向量。

将近似矩阵 $\mathbf{G}_{k+1}$ 代入拟牛顿条件公式（134），以取代 $\mathbf{H}_{\mathbf{x}^{(k)}}^{-1}$，可得：

$$\begin{aligned} (\mathbf{G}_k+m \mathbf{V} \mathbf{V}^T+n \mathbf{W} \mathbf{W}^T) \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}} &= \Delta \mathbf{X}_{\mathbf{x}^{(k+1)}} \\ \mathbf{G}_k \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}+\mathbf{V}\left(m \mathbf{V}^T \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}\right)+\mathbf{W}\left(n \mathbf{W}^T \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}\right) &= \Delta \mathbf{X}_{\mathbf{x}^{(k+1)}} \\ \mathbf{V}\left(m \mathbf{V}^T \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}\right)+\mathbf{W}\left(n \mathbf{W}^T \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}}\right) &= \Delta \mathbf{X}_{\mathbf{x}^{(k+1)}}-\mathbf{G}_k \Delta \mathbf{G}_{\mathbf{x}^{(k+1)}} \end{aligned} \tag{136}$$

观察最后一行公式，可得：

$$\begin{aligned}
&\mathbf{V} = \Delta\mathbf{X}_{\mathbf{x}^{(k+1)}} \\
&m\mathbf{V}^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}} = 1 \quad \Rightarrow \quad m = \frac{1}{\mathbf{V}^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}} \\
&\mathbf{W} = \mathbf{G}_k\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}} \\
&n\mathbf{W}^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}} = -1 \quad \Rightarrow \quad n = -\frac{1}{\mathbf{W}^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}}
\end{aligned} \tag{137}$$

代入近似矩阵的迭代公式（135），可得：

$$\mathbf{G}_{k+1} = \mathbf{G}_k + \frac{\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}^T}{\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}} - \frac{\mathbf{G}_k\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}\left(\mathbf{G}_k\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}\right)^T}{\left(\mathbf{G}_k\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}\right)^T\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}} \tag{138}$$

实际上，还可以利用转置公式以及 $\mathbf{G}_k$ 是对称矩阵的事实，进一步化简上面的公式。可以证明，如果初始矩阵 $\mathbf{G}_0$ 是正定的，那么每次迭代时生成的近似矩阵 $\mathbf{G}_k$ 都是正定的。一个可选的初始矩阵是单位矩阵 $\mathbf{I}$。

3.9.2 BFGS 算法

BFGS（Broyden-Fletcher-Goldfarb-Shanno）算法是最流行的拟牛顿算法之一，它对 $\mathbf{H}_{\mathbf{x}^{(k)}}$ 进行近似计算。设近似矩阵为 $\mathbf{B}_k$，下面推导它的迭代更新公式：

$$\begin{aligned}
\mathbf{B}_{k+1} &= \mathbf{B}_k + \Delta\mathbf{B}_k \\
\Delta\mathbf{B}_k &= m\mathbf{V}\mathbf{V}^T + n\mathbf{W}\mathbf{W}^T
\end{aligned} \tag{139}$$

其中，m 和 n 均为实数，$\mathbf{V}$ 和 $\mathbf{W}$ 均为列向量。

将近似矩阵 $\mathbf{B}_{k+1}$ 代入拟牛顿条件公式（133），以取代 $\mathbf{H}_{\mathbf{x}^{(k)}}$，并观察到与 DFP 算法中递推公式的相似性，直接交换公式（138）中的 $\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}$ 和 $\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}$ 即可得到 BFGS 的递推公式：

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}^T}{\Delta\mathbf{G}_{\mathbf{x}^{(k+1)}}^T\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}} - \frac{\mathbf{B}_k\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}\left(\mathbf{B}_k\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}\right)^T}{\left(\mathbf{B}_k\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}\right)^T\Delta\mathbf{X}_{\mathbf{x}^{(k+1)}}} \tag{140}$$

同样，可以证明，如果初始矩阵 $\mathbf{B}_0$ 是正定的，那么每次迭代时生成的近似矩阵 $\mathbf{B}_k$ 都是正定的。一个可选的初始矩阵是单位矩阵 $\mathbf{I}$。

注意，与牛顿法一样，在实现上述拟牛顿法时，也需要使用线搜索（Line Search）技术动态地确定学习率 η 的大小。

虽然 BFGS 算法对牛顿法中的 Hessian 逆矩阵进行了近似计算，但还存在空间消耗过大的问题。为解决该问题，提出了 L-BFGS（Limited-Memory BFGS）等算法。欲了解该算法，请阅读相关文献。

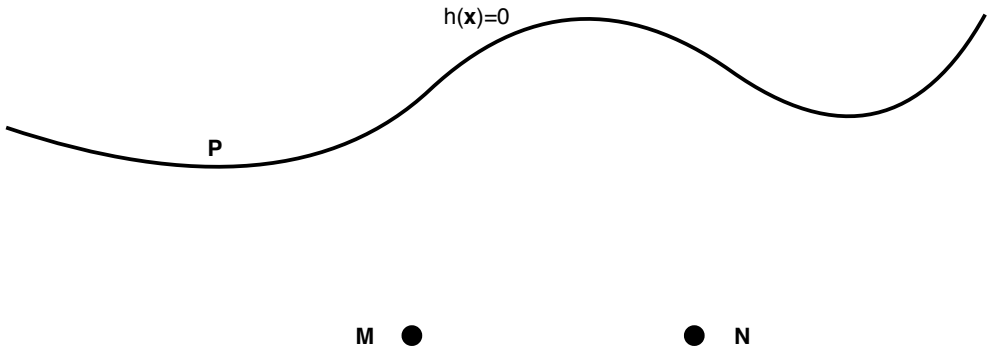


图 3-20: 一个简单的约束优化问题

3.10 拉格朗日优化方法——拉格朗日乘数与对偶性

对于无约束的优化问题，例如：

$$\min_{\mathbf{x}\in\mathbf{R}^n} f(\mathbf{x}) \tag{141}$$

其中，$f(\mathbf{x})$ 具有一阶和二阶偏导数，可以使用前面介绍的梯度下降法、牛顿法和拟牛顿法等来解决。

但是，对于如下约束优化问题：

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{R}^n} f(\mathbf{x}) \\ &\text{s.t. } h_i(\mathbf{x}) = 0, i = 1, 2, \ldots, s \\ &g_j(\mathbf{x}) \le 0, j = 1, 2, \ldots, t \end{aligned} \tag{142}$$

其中，$f(\mathbf{x}), h_i(\mathbf{x}), g_j(\mathbf{x})$ 是定义在 $\mathbf{R}^n$ 上的连续可微函数（上述优化问题被称为约束优化问题的原始问题），如何解决呢？

3.10.1 问题的引入

我们先从一个简单的问题开始，讨论如何进行优化，如图3-20所示。假设在平面上有两点，它们分别是 $\mathbf{M}$ 和 $\mathbf{N}$，且有一条平面曲线 $h(\mathbf{x}) = 0$，现在需要在该平面曲线上找出一点 $\mathbf{P}$ 使得 $|\mathbf{MP}| + |\mathbf{NP}|$ 的距离最短？

设 $f(\mathbf{x}) = dist(\mathbf{M}, \mathbf{x}) + dist(\mathbf{N}, \mathbf{x})$，则该问题可以形式化为：

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{R}^2} f(\mathbf{x}) \\ &\text{s.t. } h(\mathbf{x}) = 0 \end{aligned} \tag{143}$$

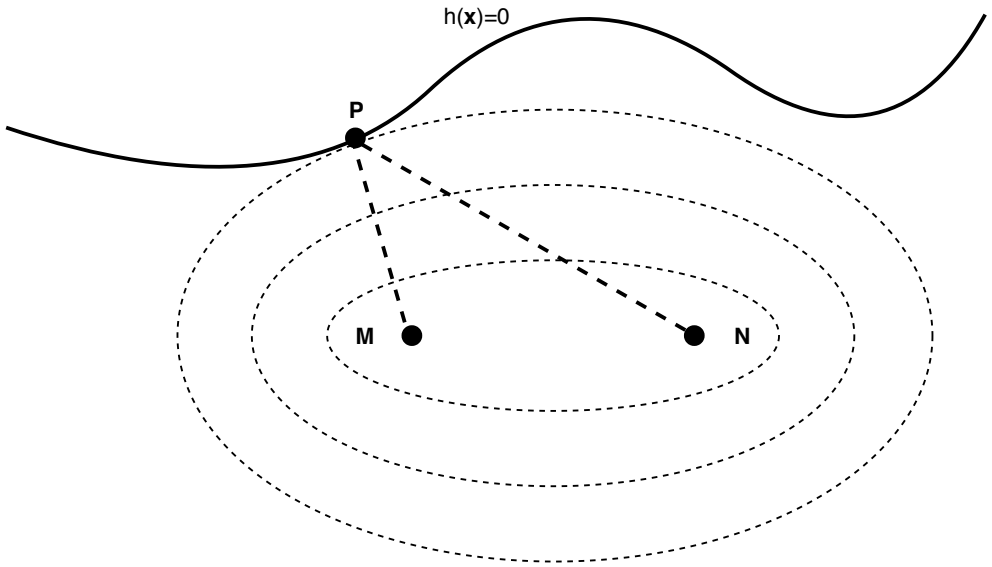


图 3-21: 一个非常直观的解决方案

显然，这是带一个等式约束的优化问题。我们先来看一种非常直观的解决方案，它有着很好的几何解释，如图3-21所示。

从该图可以看出，我们以 $\mathbf{M}$ 和 $\mathbf{N}$ 点为焦点分别做出不同的椭圆曲线，当不断扩大的椭圆曲线第 1 次与曲线 $h(\mathbf{x}) = 0$ 相切时，该相切点 $\mathbf{P}$ 就是我们要寻找的点[40]！它也是该约束优化问题的解。

下面从代数的角度来解决该约束优化问题。如图3-21所示，当椭圆曲线与曲线 $h(\mathbf{x})$ 相切时，它们在点 $\mathbf{P}$ 处的梯度存在如下关系：

$$\nabla F(\mathbf{P}) = \lambda \nabla h(\mathbf{P}) \tag{144}$$

其中，函数 F 为椭圆曲线方程，λ 是一个常量。上式表明，它们之间的梯度只相差一个常量[41]。

继续对公式（144）进行变换，可得：

$$\nabla \left(F(\mathbf{P}) - \lambda h(\mathbf{P}) \right) = 0 \tag{145}$$

此时，设 $L(\mathbf{x}, \lambda) = F(\mathbf{x}) - \lambda h(\mathbf{x})$，那么公式（145）所对应的就是函数 $L(\mathbf{x}, \lambda)$ 的优化问题——计算梯度，并令梯度等于 $\mathbf{0}$——一个典型的极值优化问题，但是已经变换成了一个无约束的优化问题！

[40]椭圆曲线上的点到两个焦点的距离和是一个常量。

[41]把每一条椭圆曲线看作是一条等值线或等高线，椭圆曲线上某点处的梯度始终垂直于该点处的等高线，并指向数值更高的等高线。

3.10.2 广义拉格朗日函数与拉格朗日乘数

实际上，上述的简单实例已经体现了拉格朗日优化方法的核心思想，即通过引入广义拉格朗日函数（Generalized Lagrange Function，例如函数 $L(\mathbf{x},\lambda)$）和拉格朗日乘数（Lagrange Multipliers，例如 λ，也被称为系数）将带约束的优化问题转换成无约束的优化问题。

上述的几何解释非常直观。实际上，还可以从偏导数的角度来解释广义拉格朗日函数。在广义拉格朗日函数 $L(\mathbf{x},\lambda)=F(\mathbf{x})-\lambda h(\mathbf{x})$ 中，如果对 λ 计算偏导数（把 $\mathbf{x}$ 当作常量），则可得 $\frac{\partial}{\partial\lambda}L(\mathbf{x},\lambda)=h(\mathbf{x})$，并令 $\frac{\partial}{\partial\lambda}L(\mathbf{x},\lambda)=h(\mathbf{x})=0$，这相当于恢复出了原问题中的等式约束部分。因此，广义拉格朗日将原问题中的待优化问题与约束部分，通过拉格朗日乘数叠加在一起了。

一般地，如果存在多个等式约束，则每个等式约束需要一个拉格朗日乘数，然后再将它们叠加在一起[42]，它的广义拉格朗日函数定义如下：

$$L(\mathbf{x},\boldsymbol{\lambda})=f(\mathbf{x})+\sum_{i=1}^{s}\lambda_i h_i(\mathbf{x}) \tag{146}$$

其中，$\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\cdots,\lambda_s)^T\in\mathbf{R}^s$ 是拉格朗日乘数。

从解析法的角度来看，为了求解上述优化问题，只需解下列方程组：

$$\begin{cases}\dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial x_1}=0\\ \dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial x_2}=0\\ \cdots\\ \dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial x_n}=0\\ \dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial \lambda_1}=0\\ \dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial \lambda_2}=0\\ \cdots\\ \dfrac{\partial L(\mathbf{x},\boldsymbol{\lambda})}{\partial \lambda_s}=0\end{cases} \tag{147}$$

[42]其几何与代数意义的解释是类似的，前者可以从梯度叠加的角度来解释——即待优化函数的梯度与各个等式约束函数的梯度之和相等，后者可以从各个等式约束函数对拉格朗日乘数的偏导数为 0 这一角度来解释——即分别获得相应的等式约束函数。

联立求解，总共 $n+s$ 个方程，$n+s$ 个未知数，方程组具有唯一最优解。在最优解不能使用解析法计算的情况下，可以使用各种优化迭代方法进行计算。实际上，通常的做法是，将拉格朗日优化原问题转化到相应的对偶问题进行求解，后面将进行详细的讨论。

现在考虑，在等式约束的基础上，新添加一个不等式约束 $g(\mathbf{x}) \leq 0$。此时，优化问题的最优值 $\mathbf{x}^*$，存在如下 2 种情形：

- 最优值 $\mathbf{x}^*$ 本身已满足 $g(\mathbf{x}) < 0$

  这意味着不等式约束 $g(\mathbf{x}) \leq 0$ 不起作用，可以直接让不等式约束 $g(\mathbf{x}) \leq 0$ 的拉格朗日乘数 $\mu = 0$；

- 最优值 $\mathbf{x}^*$ 落在边界 $g(\mathbf{x}) = 0$ 上

  这意味着不等式约束 $g(\mathbf{x}) \leq 0$ 变成了等式约束，即只需要新添加一个等式约束。但是考虑到两者的梯度是反向的因素，因此，必须对其拉格朗日乘数做出限制：$\mu > 0$，$\nabla f(\mathbf{x}^*) + \mu \nabla g(\mathbf{x}^*) = 0 \Rightarrow \nabla f(\mathbf{x}^*) = -\mu \nabla g(\mathbf{x}^*)$。

综合考虑上面 2 种情形，有：$\mu \geq 0$ 和 $\mu g(\mathbf{x}) = 0$，对于不等式约束 $g(\mathbf{x}) \leq 0$，可以定义如下的广义拉格朗日函数：

$$L(\mathbf{x}, \boldsymbol{\lambda}, \mu) = f(\mathbf{x}) + \sum_{i=1}^{s} \lambda_i h_i(\mathbf{x}) + \mu g(\mathbf{x}) \tag{148}$$

其中，$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_s)^T \in \mathbf{R}^s$ 是等式约束的拉格朗日乘数，μ 是不等式约束的拉格朗日乘数。但是，必须要附加如下的额外条件：

$$\begin{cases} g(\mathbf{x}) \leq 0 \\ \mu \geq 0 \\ \mu g(\mathbf{x}) = 0 \end{cases} \tag{149}$$

上述条件被称为 KKT（Karush-Kuhn-Tucker）条件。

类似地，如果存在多个不等式约束 $g_j(\mathbf{x}) \leq 0$，$j = 1, \cdots, t$，可以定义相应的广义拉格朗日函数：

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^{s} \lambda_i h_i(\mathbf{x}) + \sum_{j=1}^{t} \mu_j g_j(\mathbf{x}) \tag{150}$$

其中，$\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\cdots,\lambda_s)^T\in\mathbf{R}^s$ 是等式约束的拉格朗日乘数，$\boldsymbol{\mu}=(\mu_1,\mu_2,\cdots,\mu_t)^T\in\mathbf{R}^t$ 是不等式约束的拉格朗日乘数。但是，必须要附加如下的 KKT 条件 $(j=1,2,\cdots,t)$：

$$\begin{cases} g_j(\mathbf{x})\le 0 \\ \mu_j\ge 0 \\ \mu_j g_j(\mathbf{x})=0 \end{cases} \tag{151}$$

3.10.3 拉格朗日问题的对偶性

在广义拉格朗日函数中，除了原来的变量 $\mathbf{x}$ 外，还新引入了拉格朗日乘数，它们分别是等式约束的拉格朗日乘数 $\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\cdots,\lambda_s)^T\in\mathbf{R}^s$ 和不等式约束的拉格朗日乘数 $\boldsymbol{\mu}=(\mu_1,\mu_2,\cdots,\mu_t)^T\in\mathbf{R}^t$ 以及不等式约束引入的 KKT 条件。

为了简化问题，需要分步进行优化。在公式（150）定义的广义拉格朗日函数优化问题中，总共有 s 个等式约束和 t 个不等式约束，设自变量 $\mathbf{x}$ 的可行区域为 $\mathbf{O}\in\mathbf{R}^n$ 且非空。为了分步讨论方便，下面定义 2 个函数：$\theta_P(\mathbf{x})$ 和 $\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu})$。

如果 $\tilde{\mathbf{x}}\in\mathbf{O}$ 为可行区域中的点，则对于 $\forall\boldsymbol{\mu}\succeq 0,\boldsymbol{\lambda}$[43]，均有：

$$\sum_{i=1}^{s}\lambda_i h_i(\tilde{\mathbf{x}})+\sum_{j=1}^{t}\mu_j g_j(\tilde{\mathbf{x}})\le 0 \tag{152}$$

进而有：

$$L(\tilde{\mathbf{x}},\boldsymbol{\lambda},\boldsymbol{\mu})=f(\tilde{\mathbf{x}})+\sum_{i=1}^{s}\lambda_i h_i(\tilde{\mathbf{x}})+\sum_{j=1}^{t}\mu_j g_j(\tilde{\mathbf{x}})\le f(\tilde{\mathbf{x}}) \tag{153}$$

因此，如果 $\mathbf{x}^*$ 为优化问题的最优解，那么下式也成立：

$$L(\mathbf{x}^*,\boldsymbol{\lambda},\boldsymbol{\mu})=f(\mathbf{x}^*)+\sum_{i=1}^{s}\lambda_i h_i(\mathbf{x}^*)+\sum_{j=1}^{t}\mu_j g_j(\mathbf{x}^*)\le f(\mathbf{x}^*) \tag{154}$$

有了上面讨论的基础，下面首先定义函数 $\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu})$：

$$\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu})=\min_x L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{155}$$

上式表明，该函数是一个关于 $\boldsymbol{\lambda}$ 和 $\boldsymbol{\mu}$ 的函数。从公式（154）可知，为了尽可能地获得与原问题最接近的解，必须计算函数 $\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu})$ 的极大值：

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0}\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu})=\max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0}\min_x L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{156}$$

[43] $\boldsymbol{\mu}\succeq 0$ 表示 $\boldsymbol{\mu}$ 的每个分量都是非负的。

下面定义函数 $\theta_P(\mathbf{x})$：

$$\theta_P(\mathbf{x}) = \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{157}$$

现在，原问题演变为：

$$\min_{\mathbf{x}} \theta_P(\mathbf{x}) = \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{158}$$

为了维持它与最原始问题的一致性，需要考虑 $\mathbf{x}$ 满足原始问题的约束和不满足约束这 2 种情况：

- 如果 $\mathbf{x}$ 违反了原始问题的约束：

  即 $\forall j, g_j(\mathbf{x}) > 0$ 或 $\forall i, h_i(x) \neq 0$。对于前者，可令 $\mu_j \to +\infty$；对于后者，可令 $\lambda_i h_i(x) \to +\infty$。总之，使得 $\theta_P(\mathbf{x}) = +\infty$。

- 如果 $\mathbf{x}$ 满足原始问题的约束：

  直接可得 $\theta_P(\mathbf{x}) = \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = f(\mathbf{x})$，其中 $f(\mathbf{x})$ 对 $\boldsymbol{\lambda}$ 和 $\boldsymbol{\mu}$ 而言是常量，最大值就是其本身。

通过上面的分析，可得：

$$\theta_P(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \mathbf{x}\ s.t.\ conditions \\ +\infty & otherwise \end{cases} \tag{159}$$

因此，在满足约束的条件下，有：

$$\begin{aligned} &\min_{\mathbf{x}} \theta_P(\mathbf{x}) \\ &= \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \\ &= \min_{\mathbf{x}} f(\mathbf{x}) \end{aligned} \tag{160}$$

即 $\min_{\mathbf{x}} \theta_P(\mathbf{x})$ 与原始优化问题是等价的，所以常用 $\min_{\mathbf{x}} \theta_P(\mathbf{x})$ 代表原始问题，下标 P 表示原始问题，定义原始问题的最优值为：

$$p^* = \min_{\mathbf{x}} \theta_P(\mathbf{x}) \tag{161}$$

注意到，公式（158）和公式（156）之间的对偶性。因此把公式（156）称为原问题的对偶问题。

下面给出对偶问题的最优值：

$$d^* = \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} \theta_D(\boldsymbol{\lambda},\boldsymbol{\mu}) \tag{162}$$

定理一：若原始问题与对偶问题都有最优值，则：

$$d^* = \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} \min_{x} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = p^* \tag{163}$$

Proof. 对任意的 $\boldsymbol{\lambda}$、$\boldsymbol{\mu}$ 和 $\mathbf{x}$，有：

$$\begin{aligned} \theta_D(\boldsymbol{\lambda},\boldsymbol{\mu}) &= \min_{x} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \leq L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \\ &\leq \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = \theta_P(\mathbf{x}) \end{aligned} \tag{164}$$

即下式成立：

$$\theta_D(\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \theta_P(\mathbf{x}) \tag{165}$$

由于原始问题与对偶问题都有最优值，因此：

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} \theta_D(\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \min_{\mathbf{x}} \theta_P(\mathbf{x}) \tag{166}$$

即：

$$d^* = \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} \min_{x} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\mu};\boldsymbol{\mu}\succeq 0} L(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = p^*$$

□

推论：设 $\mathbf{x}^*$、$\boldsymbol{\lambda}^*$ 和 $\boldsymbol{\mu}^*$ 分别是原始问题和对偶问题的可行解，如果 $d^\star = p^\star$，那么 $\mathbf{x}^*$、$\boldsymbol{\lambda}^*$ 和 $\boldsymbol{\mu}^*$ 分别就是原始问题和对偶问题的最优解。

综上所述，当原始问题和对偶问题的最优值相等，即 $d^* = p^*$ 时，可以通过求解对偶问题来求解原始问题（当然，对偶问题的求解要比原始问题的求解简单才行）。但是，需要满足什么条件才能使得 $d^* = p^*$ 呢？下面的定理给出了答案。

定理：对于原始问题和对偶问题，假设函数 $f(\mathbf{x})$ 和 $g_j(\mathbf{x})$ 是凸函数[44]，$h_i(\mathbf{x})$ 是仿射函数[45]，并且假设不等式约束 $g_j(\mathbf{x})$ 是严格可行的，即存在 $\mathbf{x}$，对所有 j 有 $g_j(\mathbf{x}) < 0$，则存在 $\mathbf{x}^*$、$\boldsymbol{\lambda}^*$ 和 $\boldsymbol{\mu}^*$，使得 $\mathbf{x}^*$ 是原始问题的最优解，$\boldsymbol{\lambda}^*$ 和 $\boldsymbol{\mu}^*$ 是对偶问题的最优解，并且[46]：

$$d^* = p^* = L\left(\mathbf{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*\right) \tag{167}$$

在讨论了拉格朗日对偶性问题后，下面给出 2 个相关的 KKT 概念：

[44] 无论原始问题的凸性如何，对偶问题始终是凸优化问题。

[45] 由一阶多项式构成的函数，$h_i(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, $\mathbf{A}$ 是矩阵，$\mathbf{x}$、$\mathbf{b}$ 是向量。

[46] 若 $d^* \leq p^*$，则弱对偶性（Week Duality）成立；若 $d^* = p^*$，则强对偶性（Strong Duality）成立。

- 对偶互补条件或互补松弛条件：$\mu_j^* g_j\left(\mathbf{x}^*\right)=0, j=1,2, \cdots, t$；
- 对偶约束条件：$\mu_j^* \geq 0, j=1,2, \cdots, t$；

特别注意，当 $\mu_j^*>0$ 时，根据 KKT 对偶互补条件，有 $g_j(\mathbf{x}^*)=0$。这个知识点将在 SVM 的推导中得到应用。

总之，某些条件下，如果原始问题求解棘手，可以把原始约束优化问题通过广义拉格朗日函数转化为无约束优化问题，在满足强对偶性条件时，通过求解对偶问题来获得原始问题的解，使得问题的求解变得容易些。

3.11 最小二乘法：直线拟合实验

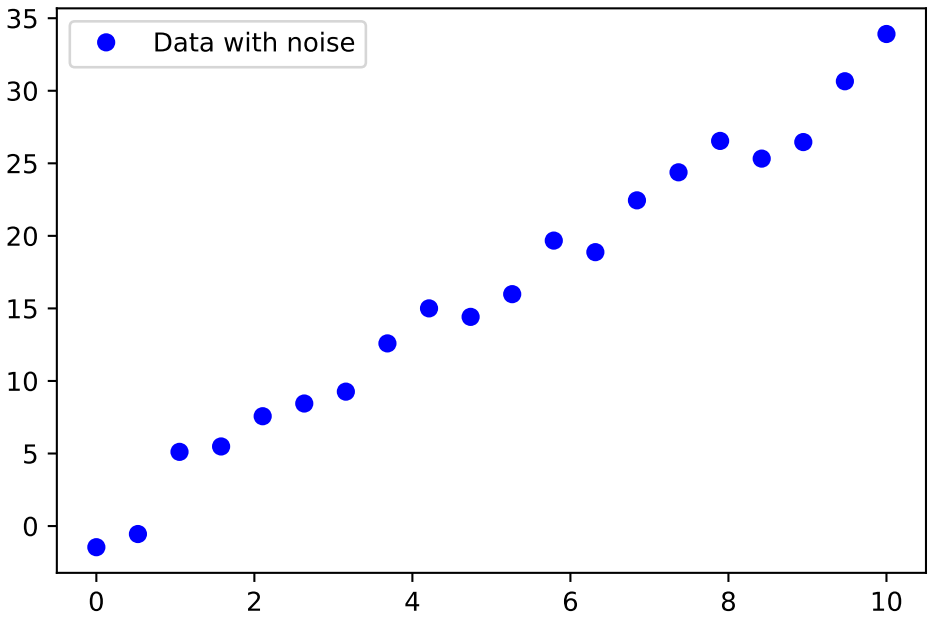
```
In [1]: import numpy as np
        from scipy.optimize import leastsq
        import matplotlib.pyplot as plt
        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'
```

生成数据点

```
In [2]: X = np.linspace(0, 10, 20)
        Y = 3 * X + 1 + np.random.normal(scale=1.5, size = X.shape)
```

绘制数据点

```
In [3]: plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.legend()
        plt.savefig('LINE_OUTPUT1.pdf', bbox_inches='tight')
```



定义直线函数

```
In [4]: def func(W, x):
            k, b = W
            return k * x + b
```

定义误差函数

```
In [5]: def error(W, x, y, step):
            step[0] = step[0] + 1
            print("Iteration: ", step[0])
            return func(W, x) - y
```

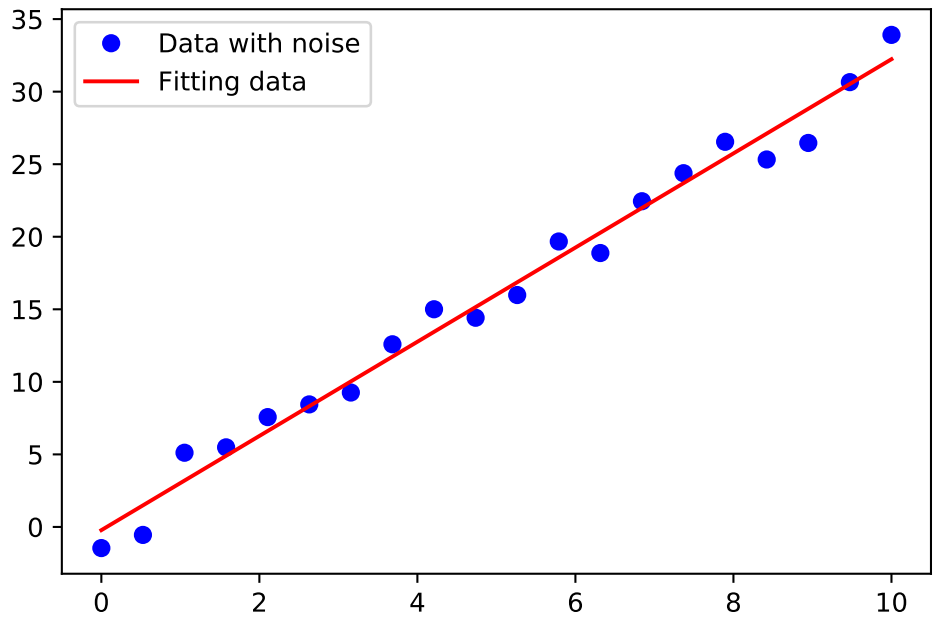
使用最小二乘法求解直线参数

```
In [6]: W0 = [100, 2]
        step = [0]
        # 把 error 函数中除了 W 以外的参数打包到 args 中
        lst = leastsq(error, W0, args=(X, Y, step))
        k, b = lst[0]
        print("k = ", k, " b = ", b)
```

```
Iteration:  1
Iteration:  2
Iteration:  3
Iteration:  4
Iteration:  5
Iteration:  6
Iteration:  7
Iteration:  8
Iteration:  9
k =  3.24575494124  b =  -0.221454728255
```

绘制拟合结果

```
In [7]: X_points = np.linspace(0, 10, 100)
        Y_points = k * X_points + b
        plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.plot(X_points, Y_points, 'r', label = 'Fitting data')
        plt.legend()
        plt.savefig('LINE_OUTPUT2.pdf', bbox_inches='tight')
```



3.12 最小二乘法：多项式拟合实验

高斯-马尔科夫定理表明，在满足一定条件时（随机噪声 ϵ 的均值 $\mathbb{E}[\epsilon|\boldsymbol{X}]=0$ 且独立于 $\boldsymbol{x}$；随机噪声 $\boldsymbol{\epsilon}$ 的方差 $\mathrm{Var}(\boldsymbol{\epsilon}|\boldsymbol{X})=\sigma^2\boldsymbol{I}$ 恒定不变），最小二乘法得到的线性回归参数在所有的无偏估计中具有最优的有效性，即方差最小。证明过程详见《机器学习算法背后的理论与优化》，史春奇等，清华大学出版社，2019 年 7 月第 1 版，1.2.2 节。

实例：$y=sin(2\pi x)+\epsilon$，其中 ϵ 为正态分布噪声，试用一个多项式去拟合该函数。

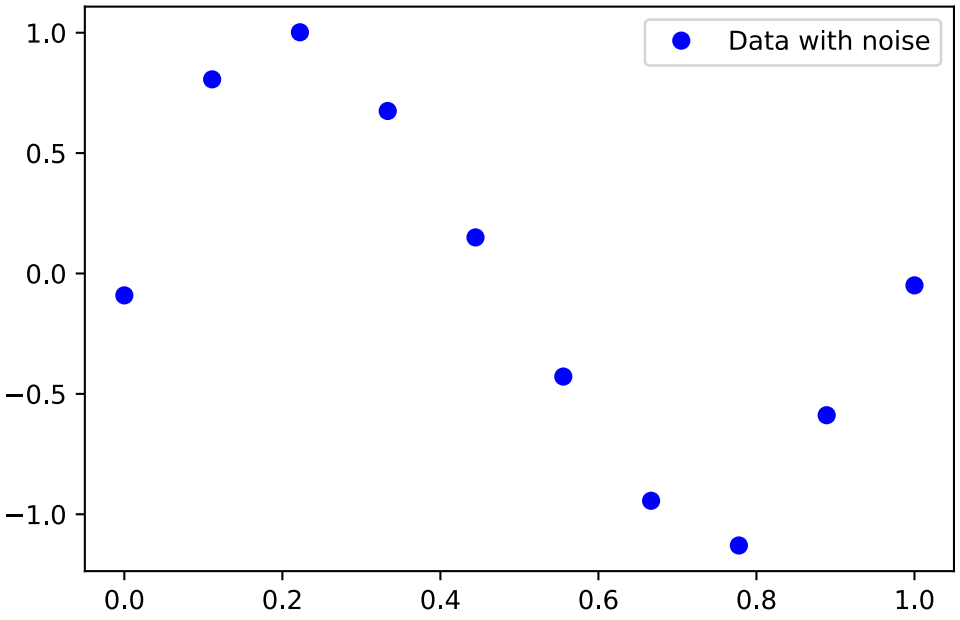
In [1]:
```
import numpy as np
import scipy as sp
from scipy.optimize import leastsq
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

np.random.seed()
```

生成待拟合的数据点（带噪声）并绘制

In [2]:
```
X = np.linspace(0, 1, 10) # 待拟合的点 x 坐标

Y = np.sin(2*np.pi*X) # 待拟合的点 y 坐标
epsilon = np.random.normal(0, 0.1, size = X.shape)
Y = Y + epsilon

print(X.shape)
print(Y.shape)
print(epsilon.shape)
```

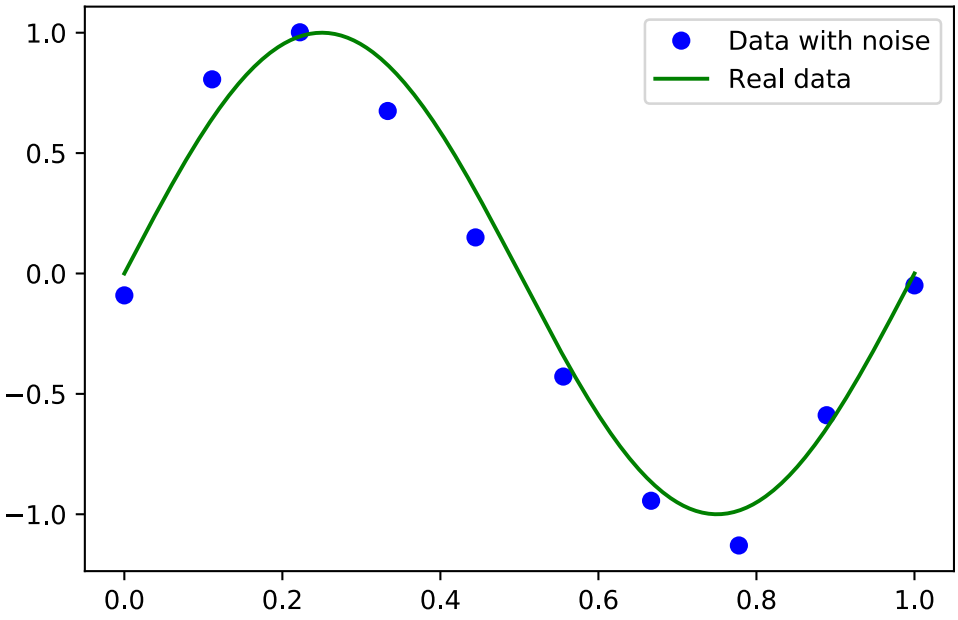
```
(10,)
(10,)
(10,)
```

In [3]:
```
plt.plot(X, Y, 'bo', label = 'Data with noise')
plt.legend()
plt.savefig('POLY_OUTPUT1.pdf', bbox_inches='tight')
```



生成原始数据点（不带噪声）并进行绘制

```
In [4]: X_points = np.linspace(0, 1, 500) # 待绘制的点 x 坐标
        Y_points = np.sin(2*np.pi*X_points) # 待绘制的点 y 坐标

In [5]: plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.plot(X_points, Y_points, 'g', label = 'Real data')
        plt.legend()
        plt.savefig('POLY_OUTPUT2.pdf', bbox_inches='tight')
```



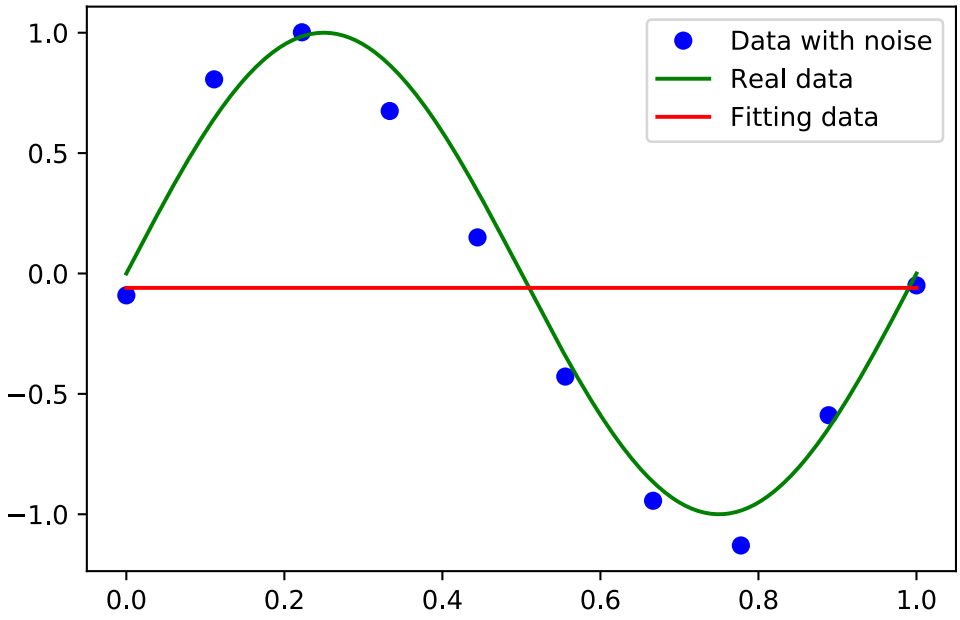
使用最小二乘法拟合
定义误差函数

```
In [6]: #W 是多项式的参数
        def error(W, x, y):
            #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
            f = np.poly1d(W)
            return f(x)-y
```

多项式次数 M=0

```
In [7]: M = 0 # 多项式次数
        W = np.random.rand(M+1) # 随机生成多项式参数
        lsq = leastsq(error, W, args = (X, Y))
        W = lsq[0]
        print('W: ', W)
        plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.plot(X_points, Y_points, 'g', label = 'Real data')
        plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
        plt.legend()
        plt.savefig('POLY_OUTPUT3.pdf', bbox_inches='tight')
```

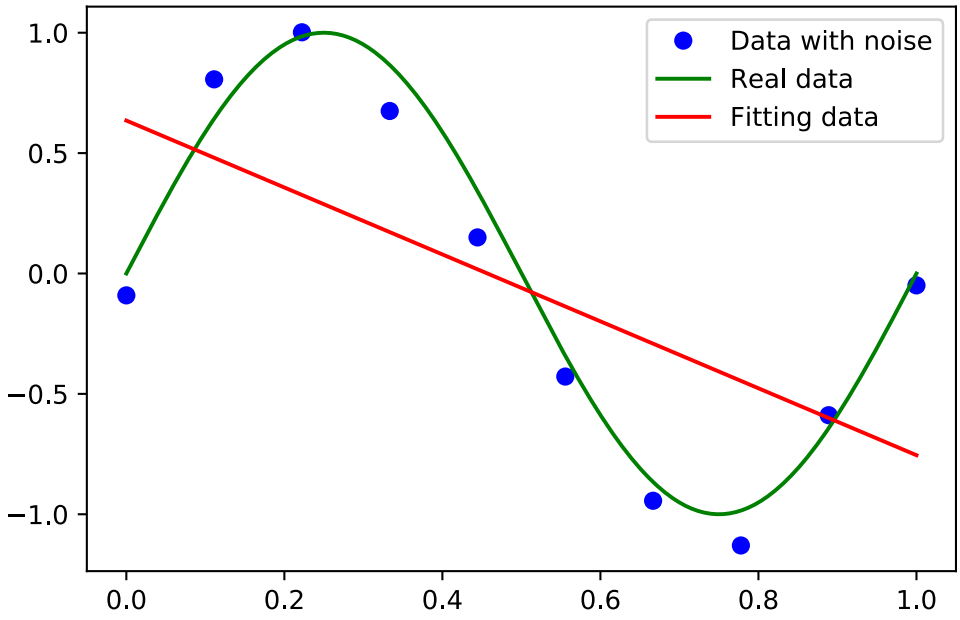
```
W:  [-0.05975641]
```



多项式次数 M=1

```
In [8]: M = 1 # 多项式次数
        W = np.random.rand(M+1) # 随机生成多项式参数
        lsq = leastsq(error, W, args = (X, Y))
        W = lsq[0]
        print('W: ', W)
        plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.plot(X_points, Y_points, 'g', label = 'Real data')
        plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
        plt.legend()
        plt.savefig('POLY_OUTPUT4.pdf', bbox_inches='tight')
```

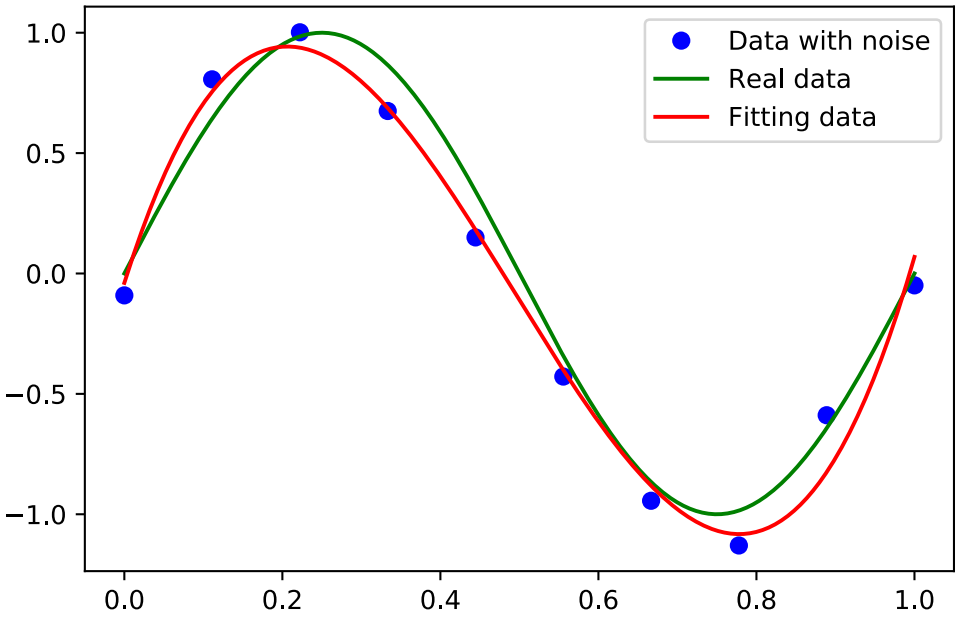
```
W:  [-1.38995301  0.63522009]
```



多项式次数 M=3

```
In [9]: M = 3 # 多项式次数
        W = np.random.rand(M+1) # 随机生成多项式参数
        lsq = leastsq(error, W, args = (X, Y))
        W = lsq[0]
        print('W: ', W)
        plt.plot(X, Y, 'bo', label = 'Data with noise')
        plt.plot(X_points, Y_points, 'g', label = 'Real data')
        plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
        plt.legend()
        plt.savefig('POLY_OUTPUT5.pdf', bbox_inches='tight')
```

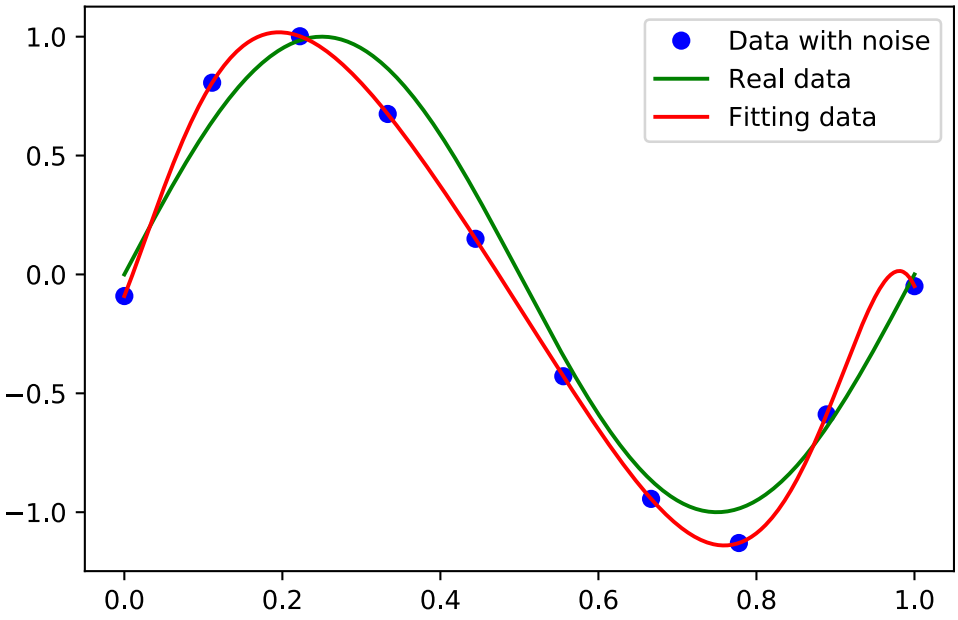
```
W:  [ 21.6654279  -31.99427083  10.43674666  -0.03906071]
```



多项式次数 M=9

```
In [10]: M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT6.pdf', bbox_inches='tight')
```

```
W:  [ -1.03820137e+03   4.36067832e+03  -8.06011405e+03   8.60351460e+03
  -5.74827113e+03   2.39828378e+03  -5.60711281e+02   3.64616941e+01
   8.40101803e+00  -9.08754331e-02]
```



正则化

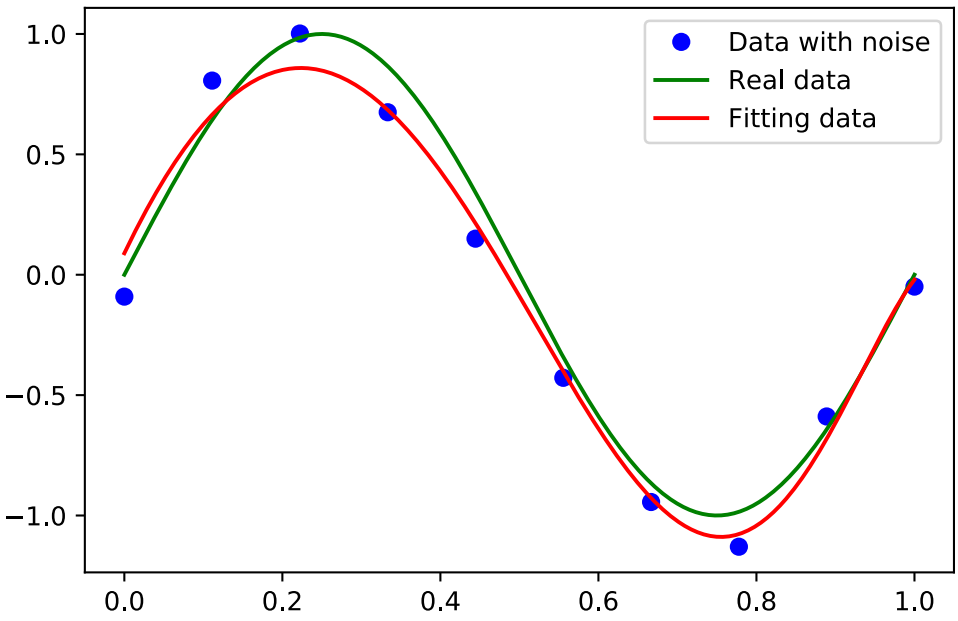
当 M=9 时，曲线拟合容易出现过拟合，可以引入正则化项 (regularizer)：

$$L(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{y}_i)^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

对于回归问题，正则化项可以是参数向量 $\boldsymbol{w}$ 的 L_2 范数，也可以是 L_1 范数。此处，使用 L_2 范数。

```
In [11]: #W 是多项式的参数
         weight = 0.001
         def error_L2(W, x, y):
             #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
             f = np.poly1d(W)
             return np.append(f(x)-y, np.sqrt(0.5*weight*np.square(W)))

         M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error_L2, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT7.pdf', bbox_inches='tight')

W:  [ -5.44869807  -1.27629572   2.49406481   5.33300197   6.3270061
```

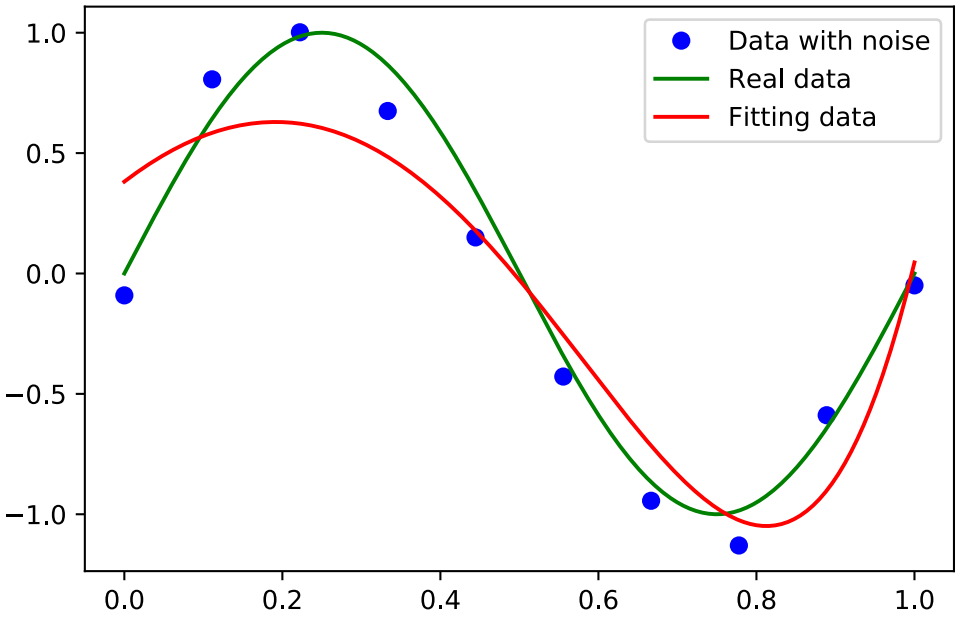
```
3.97415888  -3.61557397 -14.75558235   6.85824918   0.08918448]
```



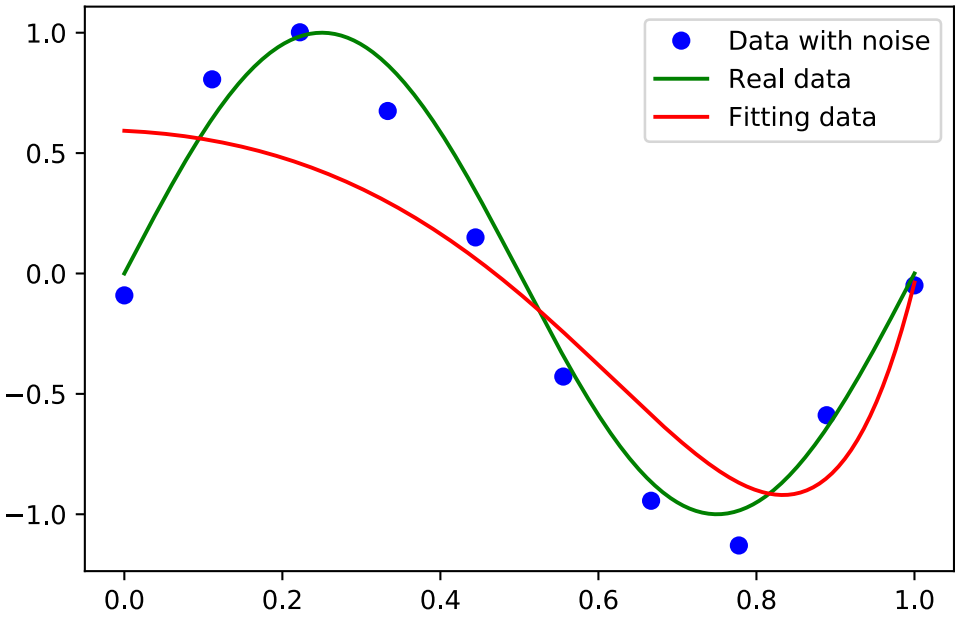
不同的权重参数 λ 对拟合结果的影响

```
In [12]: #W 是多项式的参数
         weight = 0.01
         def error_L2(W, x, y):
             #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
             f = np.poly1d(W)
             return np.append(f(x)-y, np.sqrt(0.5*weight*np.square(W)))

         M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error_L2, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT8.pdf', bbox_inches='tight')
```

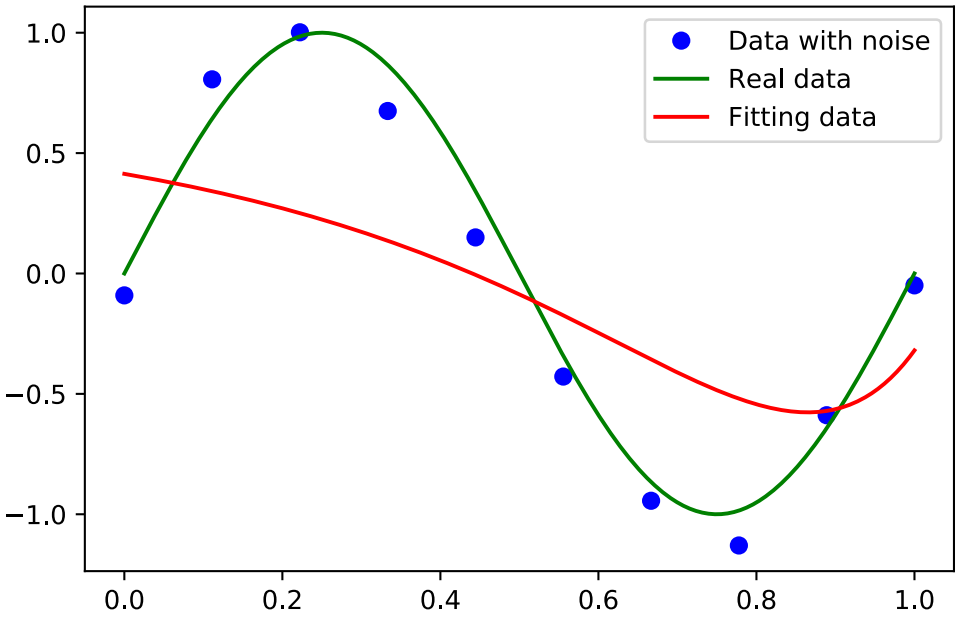
```
W:  [-0.91456068  0.43808918  1.54508144  2.17197005  1.96624088  0.46598583
 -2.63340642 -5.88602947  2.51110434  0.38152856]
```



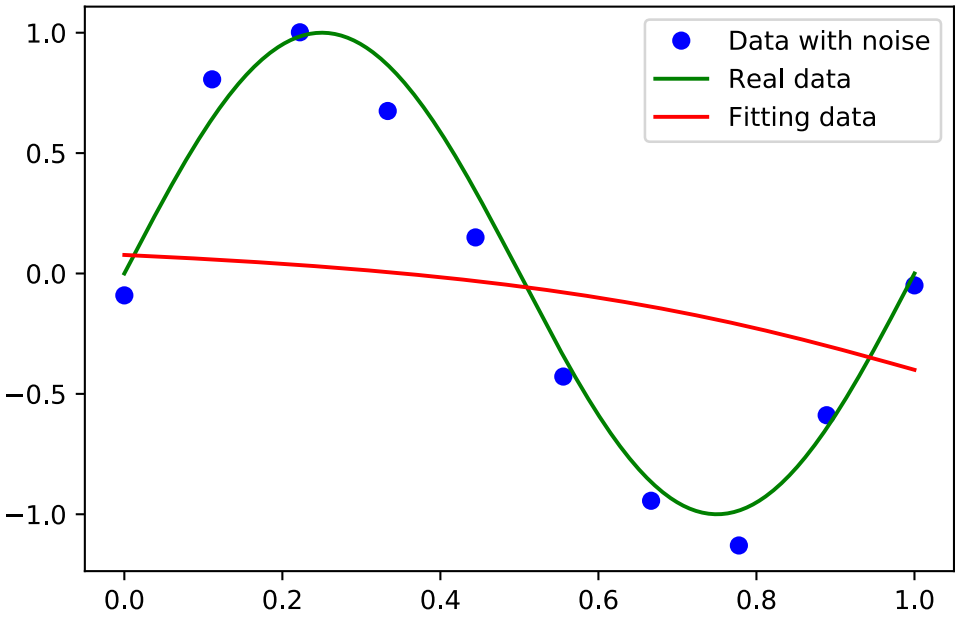

```
In [13]: #W 是多项式的参数
         weight = 0.1
         def error_L2(W, x, y):
             #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
             f = np.poly1d(W)
             return np.append(f(x)-y, np.sqrt(0.5*weight*np.square(W)))

         M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error_L2, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT9.pdf', bbox_inches='tight')

W:  [ 0.8351165   0.80098829  0.69613939  0.4847631   0.11993153 -0.44535836
 -1.19615234 -1.77296075 -0.15302984  0.59273811]
```




```
In [14]: #W 是多项式的参数
         weight = 1
         def error_L2(W, x, y):
             #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
             f = np.poly1d(W)
             return np.append(f(x)-y, np.sqrt(0.5*weight*np.square(W)))

         M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error_L2, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT10.pdf', bbox_inches='tight')

W:  [ 0.36797677  0.31321371  0.23919257  0.13873591  0.00255738 -0.17925908
 -0.40841186 -0.63715373 -0.57002881  0.41378759]
```




```
In [15]: #W 是多项式的参数
         weight = 10
         def error_L2(W, x, y):
             #poly1d 生成一个多项式。注意，它的 x 多项式次数由高到低的顺序排列
             f = np.poly1d(W)
             return np.append(f(x)-y, np.sqrt(0.5*weight*np.square(W)))

         M = 9 # 多项式次数
         W = np.random.rand(M+1) # 随机生成多项式参数
         lsq = leastsq(error_L2, W, args = (X, Y))
         W = lsq[0]
         print('W: ', W)
         plt.plot(X, Y, 'bo', label = 'Data with noise')
         plt.plot(X_points, Y_points, 'g', label = 'Real data')
         plt.plot(X_points, np.poly1d(W)(X_points), 'r', label = 'Fitting data')
         plt.legend()
         plt.savefig('POLY_OUTPUT11.pdf', bbox_inches='tight')

W:  [ 0.02654234  0.01624581  0.00286996 -0.01466183 -0.03775671 -0.06799672
 -0.10605245 -0.14633444 -0.15000933  0.07702366]
```



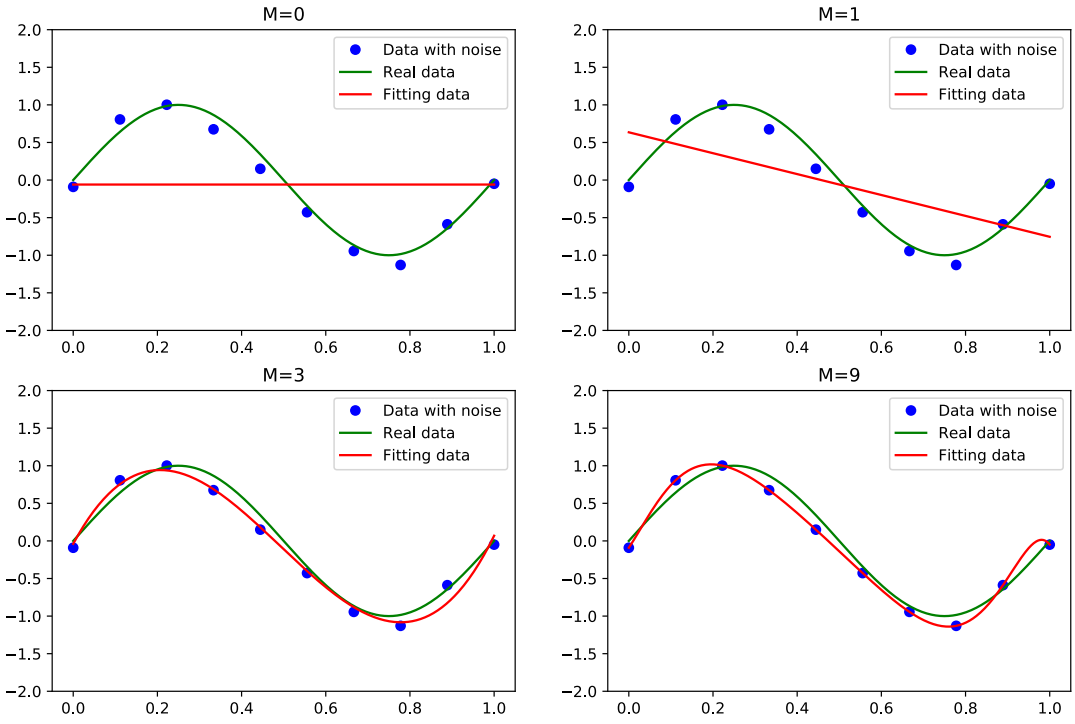
下面使用 sklearn 中的函数实现上面的多项式拟合实例

```
In [16]: import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
```

没有添加正则项

```
In [17]: X_TRAIN = X.reshape(-1, 1)
         Y_TRAIN = Y.reshape(-1, 1)

         fig = plt.figure(figsize=(12,8))
         for i, order in enumerate([0, 1, 3, 9]):

             plt.subplot(2, 2, i+1)

             poly = PolynomialFeatures(order)
             X_TRAIN_POLY = poly.fit_transform(X_TRAIN)
             lr = LinearRegression()
             lr.fit(X_TRAIN_POLY, Y_TRAIN)

             plt.ylim(-2, 2)
             plt.plot(X, Y, 'bo', label = 'Data with noise')
             plt.plot(X_points, Y_points, 'g', label = 'Real data')
             plt.plot(X_points, lr.predict(poly.fit_transform(X_points.
              reshape(-1, 1))), 'r', label = 'Fitting data')
```

```
    plt.title("M={}".format(order))
    plt.legend()
    plt.savefig('POLY_OUTPUT12.pdf', bbox_inches='tight')
```

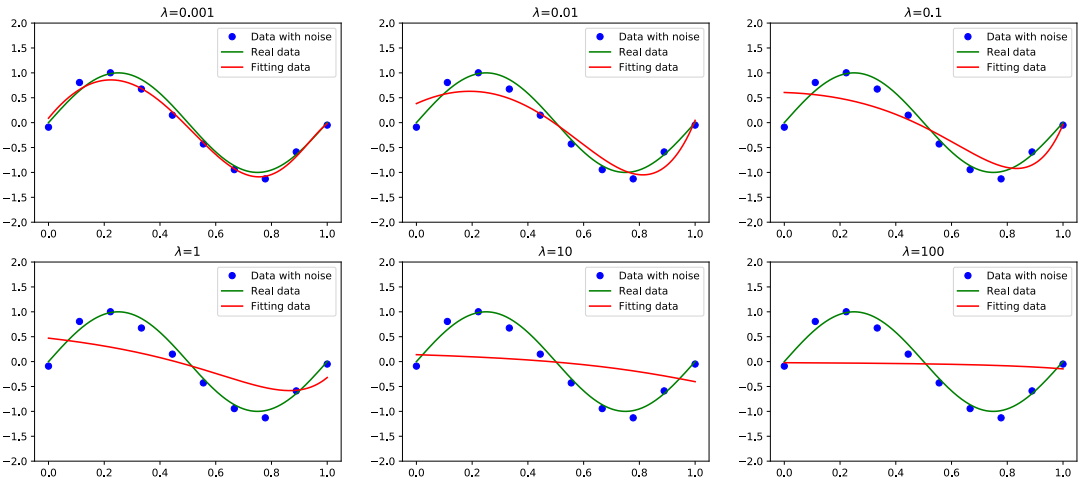


添加正则项

```
In [18]: from sklearn.linear_model import Ridge
```

```
In [19]: M = 9
         fig = plt.figure(figsize=(18, 12))
         for i, lamb in enumerate([0.001, 0.01, 0.1, 1, 10, 100]):

             plt.subplot(3, 3, i+1)

             poly = PolynomialFeatures(M)
             X_TRAIN_POLY = poly.fit_transform(X_TRAIN)
             lr = Ridge(alpha=lamb/2)
             lr.fit(X_TRAIN_POLY, Y_TRAIN)

             plt.ylim(-2, 2)
             plt.plot(X, Y, 'bo', label = 'Data with noise')
             plt.plot(X_points, Y_points, 'g', label = 'Real data')
             plt.plot(X_points, lr.predict(poly.fit_transform(X_points.
```

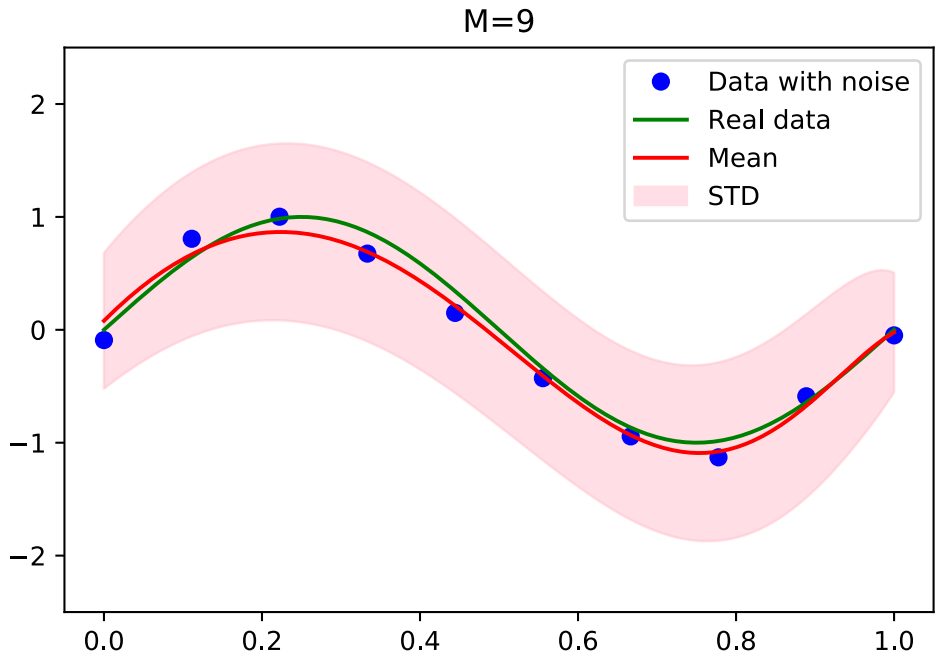
```
 reshape(-1, 1))), 'r', label = 'Fitting data')

plt.title("$\lambda$={}".format(lamb))
plt.legend()
plt.savefig('POLY_OUTPUT13.pdf', bbox_inches='tight')
```



贝叶斯曲线拟合，参见 1.4.8 节

In [20]:
```
M = 9
poly = PolynomialFeatures(M)
X_TRAIN_POLY = poly.fit_transform(X_TRAIN)

alpha=5e-3 # 模型参数精度
beta=11.1 # 数据集噪声精度

I = np.eye(np.size(X_TRAIN_POLY, 1)) # 单位矩阵
S_INV = alpha * I + beta * np.matmul(X_TRAIN_POLY.T, X_TRAIN_POLY)
S = np.linalg.inv(S_INV)
T = np.matmul(X_TRAIN_POLY.T, Y_TRAIN)

X_points_POLY = poly.fit_transform(X_points.reshape(-1, 1))
# 计算均值
MEAN = beta * np.matmul(X_points_POLY, np.matmul(S, T))
# 计算方差与标准差
SIGMA2 = 1/beta + np.sum(np.matmul(np.matmul(X_points_POLY, S),
 X_points_POLY.T), axis=1)/np.size(X_TRAIN, 0)#average
STD = np.sqrt(SIGMA2)

# 绘制均值曲线及 1 个标准差区域
plt.ylim(-2.5, 2.5)
```

```
plt.plot(X, Y, 'bo', label = 'Data with noise')
plt.plot(X_points, Y_points, 'g', label = 'Real data')
plt.plot(X_points, MEAN, c="r", label="Mean")
plt.fill_between(X_points, MEAN.flatten() - STD, MEAN.flatten() + STD,
 color="pink", label="STD", alpha=0.5)

plt.title("M={}".format(M))
plt.legend()
plt.savefig('POLY_OUTPUT14.pdf', bbox_inches='tight')
```



4 参考文献

1. Stephen Boyd, Lieven Vandenberghe. Convex Optimization. Cambridge University Press 2004.
2. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。
3. 李航。《统计学习方法》，清华大学出版社，2012 年 3 月第 1 版。
4. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
5. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
6. https://zh.wikipedia.org/wiki/梯度下降法.
7. https://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix.
8. https://en.wikipedia.org/wiki/Singular_value_decomposition.
9. https://zh.wikipedia.org/wiki/酉矩阵.
10. https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0.
11. https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/.
12. https://zh.wikipedia.org/wiki/牛顿法.
13. https://zh.wikipedia.org/wiki/应用于最优化的牛顿法.
14. https://blog.csdn.net/acdreamers/article/details/44664941.
15. http://www.slimy.com/ steuard/teaching/tutorials/Lagrange.html.
16. http://www.cnblogs.com/90zeng/p/Lagrange_duality.html.
17. https://zhuanlan.zhihu.com/p/48521073.
18. https://www.zhihu.com/question/20587681.

19. http://smallmargin.uk/bayesianCurveFitting.html.

20. https://github.com/wzyonggege/statistical-learning-method.