神经计算的数学原理

“海棠”系列丛书

作者：刘俊宏

组织：哈尔滨工业大学（深圳）

目录

第一章 回归模型 **1**
1.1 线性模型 1
1.2 量化评价 1
1.3 模型求解 2
1.4 线性模型求解二分类问题 3
1.5 多项式回归 4
1.6 正则化处理 5
1.7 算法实验 6

第二章 梯度下降 **7**
2.1 算法原理 7
2.2 随机梯度下降 7
2.2.1 小批量随机梯度下降 9
2.2.2 求解线性二分类问题 9

第三章 感知器和神经网络 **10**
3.1 感知器 10
3.2 单层神经网络 10

第一章 回归模型

内容提要

- 线性模型
- 量化评价
- 模型求解
- 多项式回归
- 正则化处理
- 算法实验

让我们以一个**线性回归**（Linear Regression）的例子开始我们的神经计算之旅。假设我们要预测去哈工大的通勤时间，并有如下数据：

- 距离 (Km): [2.7, 4.1, 1.0, 5.2, 2.8]
- 时段 (1: if weekday, 0: if weekend): [1, 1, 0, 1, 0]
- 通勤时间 (min): [25, 33, 15, 45, 22]

我们希望找到一个函数 $f : R^2 \to R$ 满足 $f(D, T_d) \approx T_c$，其中 D 表示距离，T_d 表示时段，T_c 表示通勤时间。

1.1 线性模型

我们首先尝试构建一个线性回归模型来预测通勤时间，其一般表达式如 (1.1) 所示，其中 d 为变量的个数。

$$f(x) = w_0 + w_1 x_1 + \ldots + w_d x_d \tag{1.1}$$

为了后续方便，我们将其转化为矩阵的形式：$f(x) = W^{\mathsf{T}} X$，其中

$$\left[w_0, w_1, w_2, \ldots, w_d\right] = W^{\mathsf{T}}, \begin{bmatrix} x_0 = 1 \\ x_1 \\ \ldots \\ x_d \end{bmatrix} = X$$

笔记 我们约定所有初始向量都是列向量。在上述通勤的例子中，线性回归模型中的参数 $d = 2$，x_1 和 x_2 分别对应于 D 和 T_d 中的元素，w_0 是一个偏置项（Bias Term）。

1.2 量化评价

一旦我们初始了一个线性回归模型，我们就需要一种方法来量化地评价我们模型效果的好坏，即在本例子中我们需要一种方法来评价我们构建的模型是否可以准确地预测通勤时间。在这里，我们使用**残差**（Residual）(1.2) 来衡量第 i 个数据点的期望输出（真实的通勤时间）和模型输出（预测的通勤时间）之间的差距。

$$e^{(i)} = y^{(i)} - W^{\mathsf{T}} x^{(i)} \tag{1.2}$$

同时，我们使用**均方误差** (Mean Square Error, MSE) (1.3) 来综合考虑所有数据点的残差情况。根据**一阶必要最优性条件**（First-Order Necessary Optimality Condition），当均方误差 $C(W)$ 的一阶导数 $C'(W) = 0$ 时 $C(W)$ 取得极值。又根据**二阶充分最优性条件**（Second-Order Sufficiency Conditions），$C(W)$ 的二阶导数 $C''(W) = \frac{1}{n} X^{\mathsf{T}} X$ 恒大于零，所以 $C'(W) = 0$ 时 $C(W)$ 取得最小值。

$$C(W) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - W^{\mathsf{T}} x^{(i)}\right)^2 \tag{1.3}$$

笔记 通常，标准的均方误差表达式为：$C(W) = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - W^{\mathsf{T}} x^{(i)}\right)^2$。但在机器学习，特别是在使用梯度下降

法进行优化时，将标准的均方误差缩放为 $C(W)=\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)}-W^{\mathsf{T}}x^{(i)}\right)^2$ 有利于计算的便利，这在下一小节的计算演示中大家会感受到。

1.3 模型求解

这里使用了一个巧妙地方法求解 $C'(W^*)=0$ 的解析解 W^*，首先我们假设第 i 个数据点的特征向量为 $x^{(i)}$，其中第 i 行表示第 i 个数据点的特征向量，第 j 列表示特征向量中的第 j 个特征。接着我们可以假设一个维度为 $n\times d$ 的特征矩阵 X 和目标输出向量 Y。

$$x^{(i)}=\begin{bmatrix}x_0^{(i)}=1\\ \cdots\\ x_j^{(i)}\\ \cdots\\ x_d^{(i)}\end{bmatrix},X=\begin{bmatrix}x^{(1)\mathsf{T}}\\ \cdots\\ x^{(n)\mathsf{T}}\end{bmatrix},Y=\begin{bmatrix}y^{(1)}\\ \cdots\\ y^{(n)}\end{bmatrix}$$

接着，我们首先推导 $XW-Y$ 这个公式看看会得到什么：

$$XW-Y=\begin{bmatrix}x^{(1)\mathsf{T}}\\ \cdots\\ x^{(n)\mathsf{T}}\end{bmatrix}W-\begin{bmatrix}y^{(1)}\\ \cdots\\ y^{(n)}\end{bmatrix}=\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)}\\ \cdots\\ x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}$$

由此，我们可以得到下述等式：

$$\begin{aligned}(XW-Y)^{\mathsf{T}}(XW-Y)&=\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)},&\cdots&,x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}\begin{bmatrix}x^{(1)\mathsf{T}}W-y^{(1)}\\ \cdots\\ x^{(n)\mathsf{T}}W-y^{(n)}\end{bmatrix}\\&=\sum_{i=1}^{n}\left(X^{(i)\mathsf{T}}W-y^{(i)}\right)^2\\&=2nC(W)\end{aligned}$$

上述等式 $2nC(W)=(XW-Y)^{\mathsf{T}}(XW-Y)$ 描述了 $C(W)$ 与 X，W，Y 之间的关系，于是我们可以进一步推理得到一个更明确的等式 (1.4)：

$$\begin{aligned}C(W)&=\frac{1}{2n}(XW-Y)^{\mathsf{T}}(XW-Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}-Y^{\mathsf{T}})(XW-Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-W^{\mathsf{T}}X^{\mathsf{T}}Y-Y^{\mathsf{T}}XW+Y^{\mathsf{T}}Y)\\&=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-2W^{\mathsf{T}}X^{\mathsf{T}}Y+Y^{\mathsf{T}}Y)\end{aligned}\tag{1.4}$$

笔记 在推导等式 (1.4) 的时候可能会对最后两步推导过程感到疑惑：

$$\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-W^{\mathsf{T}}X^{\mathsf{T}}Y-Y^{\mathsf{T}}XW+Y^{\mathsf{T}}Y)=\frac{1}{2n}(W^{\mathsf{T}}X^{\mathsf{T}}XW-2W^{\mathsf{T}}X^{\mathsf{T}}Y+Y^{\mathsf{T}}Y)$$

这里解释一下，这是因为 $W^{\mathsf{T}}X^{\mathsf{T}}Y$ 实际上会得到一个实数，并且 $W^{\mathsf{T}}X^{\mathsf{T}}=(XW)^{\mathsf{T}}$。于是我们可以获得等价关系：

$$W^{\mathsf{T}}X^{\mathsf{T}}Y=(W^{\mathsf{T}}X^{\mathsf{T}}Y)^{\mathsf{T}}=Y^{\mathsf{T}}XW$$

笔记 同时，请注意看 X，W，Y 的组织形式，所以这里写成 $XW-Y$ 的形式才是正确并方便我们的推导，而不是 $Y-W^{\mathsf{T}}X$ 的形式。

通过等式 (1.4) 我们可以进一步求解 $C'(W)$ 与 X,W,Y 之间的关系。我们首先通过观察可以发现：1. $W^{\mathsf{T}}X^{\mathsf{T}}XW$

是二次项，2. $W^\mathsf{T}X^\mathsf{T}Y$ 是一次项，3. $Y^\mathsf{T}Y$ 是常数项。于是我们便可以快速得到等式 (1.5):

$$C'(W) = \frac{1}{2n}(2X^\mathsf{T}XW - 2X^\mathsf{T}Y) \tag{1.5}$$

根据上一小节的推论，当 $C'(W^*) = 0$ 时取得最优的 W^*，此时 $C(W)$ 取得最小值。于是根据等式 (1.5) 可以得到 $X^\mathsf{T}XW^* = X^\mathsf{T}Y$。当 $X^\mathsf{T}X$ 可逆时便可以获得 W^* 的解析解 (1.6) 和模型的输出 Y^* (1.6)：

$$\begin{cases} W^* = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \\ Y^* = XW^* = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \end{cases} \tag{1.6}$$

证明 看完了 W^* 的解析解的整个推导过程，有些读者可能会有这样一个疑问：为什么 W 作为一个向量却可以将其视为一个“整体”进行求导操作？这涉及到多元微积分和矩阵求导的核心概念，下面开始阐述其数学原理。

1. 标量对向量的导数：梯度的定义

当函数 $f(X)$ 的输出是一个标量，而输入 X 是一个向量时，其导数被称为梯度（Gradient），记作 $\nabla_X f(X)$。梯度本身是一个向量，其每个分量是函数 f 对 X 的每个分量的偏导数。

例如，如果 $W = [w_0, w_1, ..., w_d]^\mathsf{T}$，那么：

$$\nabla_W C(W) = \frac{\partial C(W)}{\partial W} = \left[\frac{\partial C(W)}{\partial w_0}, \frac{\partial C(W)}{\partial w_1}, \cdots, \frac{\partial C(W)}{\partial w_d}\right]^\mathsf{T}$$

2. 为什么推导过程看起来像是在对“一维未知量”求导？

要回答这个问题，我们需要理解的一个关键问题是：为什么 XW（或 $W^\mathsf{T}X^\mathsf{T}$）对 W 的梯度是 X（或 X^T）。

我们假设

$$Z(W) = XW$$

其中：

$$X = \begin{bmatrix} x_0^{(1)}, x_1^{(1)}, ..., x_d^{(1)} \\ ... \\ x_0^{(n)}, x_1^{(n)}, ..., x_d^{(n)} \end{bmatrix}, W = \begin{bmatrix} w_0 \\ ... \\ w_d \end{bmatrix}$$

接着我们对 $Z(W)$ 求一阶导，即求 $Z(W)$ 的雅可比矩阵：

$$\nabla_W Z(W) = \frac{\partial Z_i}{\partial W_k} = \frac{\partial(\sum_{j=0}^{d} X_{ij}W_j)}{\partial W_k} = X_{ik}$$

即

$$J = \begin{bmatrix} \frac{\partial Z_1}{\partial w_0} & \frac{\partial Z_1}{\partial w_1} & \cdots & \frac{\partial Z_1}{\partial w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Z_n}{\partial w_0} & \frac{\partial Z_n}{\partial w_1} & \cdots & \frac{\partial Z_n}{\partial w_d} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(n)} & x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} = X$$

所以雅可比矩阵的第 i 行就是 X 的第 i 行，即证：

$$\frac{\partial XW}{\partial W} = X$$

1.4 线性模型求解二分类问题

假设我们有下述期望的输入输出对 S：

$$S = \left\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\right\},\ y^{(i)} \in \{-1, +1\}$$

模型在上述分类问题的单样本表现可以用 0-1 损失函数 L 来衡量：

$$L(\alpha, \beta) = \mathbb{I}[\alpha \neq \beta] = \begin{cases} 1, & \text{if } \alpha \neq \beta \\ 0, & otherwise. \end{cases}$$

从而模型的整体效果我们可以用下面的函数 $C(W)$ 衡量：

$$C(W) = \frac{1}{n}\sum_{i=1}^{n} L(\text{sgn}(\hat{y}^{(i)}), y^{(i)}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{sgn}(W^{\mathsf{T}}x^{(i)}) \neq y^{(i)}]$$

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0. \end{cases}$$

但上述公式有一个不方便的地方就是目前的 $C(W)$ 是一个不连续的分段函数，如果我们可以找到一个可微分的减函数 G 来代替 L 函数用于评估单个样本上的模型效果，那么我们就可以得到一个易于求解的优化问题。

这里我们引入**间隔**（Margin）的概念，并定义间隔为模型的预测值与期望值的乘积，数学表达式为 $y\hat{y}$，即 $yW^{\mathsf{T}}x$。我们现在考虑基于间隔的函数 G 来衡量模型在单个样本上的效果，下面列举了三个可选的 G 函数形式：

$$G(y\hat{y}) = \max\{0, 1 - y\hat{y}\}$$

$$G(y\hat{y}) = \frac{1}{2}(\max\{0, 1 - y\hat{y}\})^2$$

$$G(y\hat{y}) = \log(1 + \exp(-y\hat{y}))$$

这里我们考虑 G 函数的第二种形式：

$$G(y\hat{y}) = \frac{1}{2}(\max\{0, 1 - y\hat{y}\})^2 = \frac{1}{2}\left(\max\left\{0, 1 - yW^{\mathsf{T}}x\right\}\right)^2 = \begin{cases} 0, & \text{if } yW^{\mathsf{T}}x \geq 1 \\ \frac{1}{2}(1 - yW^{\mathsf{T}}x)^2, & otherwise. \end{cases}$$

于是我们有如下形式的 $C(W)$ 函数：

$$C(W) = \frac{1}{2n}\sum_{i=1}^{n}\left(\max\left\{0, 1 - y^{(i)}W^{\mathsf{T}}x^{(i)}\right\}\right)^2$$

其中 $C_i(W) = G(y^{(i)}\hat{y}^{(i)})$，接着我们可以求得 $C_i(W)$ 的一阶导表达式：

$$\nabla C_i(W) = \begin{cases} 0, & \text{if } y^{(i)}W^{\mathsf{T}}x^{(i)} \geq 1 \\ -(1 - y^{(i)}W^{\mathsf{T}}x^{(i)})y^{(i)}x^{(i)}, & otherwise. \end{cases}$$

又因为 $y^2(i) = 1$，所以 $\nabla C_i(W)$ 的表达式 (1.7) 可以简化为：

$$\nabla C_i(W) = \begin{cases} 0, & \text{if } y^{(i)}W^{\mathsf{T}}x^{(i)} \geq 1 \\ (W^{\mathsf{T}}x^{(i)} - y^{(i)})x^{(i)}, & otherwise. \end{cases} \tag{1.7}$$

通过上面的学习我们知道最优的模型权重 W^* 必须满足 $\frac{1}{n}\sum_{i=1}^{n}\nabla C_i(W^*) = 0$，但即使我们知道了等式左右两边的完整表达式，我们仍然缺乏一种有效的手段来求解上面的等式，即难以求出 W^* 的解析解。所以退而求其次，我们将在下面章节介绍利用随机梯度下降来求 W^* 的数值解（近似解）的方法。

1.5 多项式回归

在推导出线性回归模型的解后，我们其实也可以用上述公式求解多项式回归的 W^* 的解析解。多项式回归是一种基础的非线性回归模型，其基本形式如公式 (1.8) 所示：

$$f(x) = w_0 + w_1x_1 + w_2x^2 + \ldots + w_mx^m = W^{\mathsf{T}}\phi(x) \tag{1.8}$$

其中 $\phi(x) = \left[1, x, x^2, \cdots, x^m\right]^{\mathsf{T}}$，我们同样设：

$$X = \begin{bmatrix} \phi(x)^{(1)\mathsf{T}} \\ \ldots \\ \phi(x)^{(n)\mathsf{T}} \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ \ldots \\ y^{(n)} \end{bmatrix}$$

同样可以得到相同的解析解形式：

$$\begin{cases} W^* = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \\ Y^* = XW^* = X(X^\mathsf{T}X)^{-1}X^\mathsf{T}Y \end{cases}$$

1.6 正则化处理

理想情况下，我们希望模型在训练样本上表现良好同时又不至于太复杂，从而实现良好的泛化（既不过拟合也不过欠拟合）。解决这个问题的一种方法是鼓励较小的权重（这样任何特征都不会对预测产生太大的影响）。这被称为**正则化**（Regularization）。

给定一个数据集 $S = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$ 和一个正则化强度 $\lambda > 0$，我们同样希望找到下列**正则化最小二乘回归**函数（Regularized Least Square Regression） (1.9)，也被称为岭回归（Ridge Regression），的最小值：

$$C(W) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - W^\mathsf{T}X^{(i)}\right)^2 + \frac{\lambda}{2}||W||_2^2 \tag{1.9}$$

其中 $||W||_2$ 表示 W 的二范数（2-norm）。现在函数 (1.9) 有两个优化目标：函数右边第一项 $\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - W^\mathsf{T}X^{(i)}\right)^2$ 是误差项（经验风险），用来衡量模型对训练数据的拟合程度；函数右边第二项 $\frac{\lambda}{2}||W||_2^2$ 是正则化项（结构风险），用来对模型的复杂度进行惩罚，当模型的某些权重变得过大，正则化项会变大并超过误差项的影响，从而导致总成本 $C(W)$ 变大。

定义 1.1 (范数)

对于一个向量 $v = (v_1, ..., v_d) \in R^d$，范数被定义为：

$$||V||_n = (|v_1|^n + |v_2|^n + ... + |v_d|^n)^{\frac{1}{n}}$$

二范数也被称为欧几里得范数（Euclidean Norm）。

♣

接下来我们开始求解上述函数的 W^* 值，前面我们已经证明过 $2nC(W) = (XW - Y)^\mathsf{T}(XW - Y)$，于是我们便可以得知：

$$C(W) = \frac{1}{2n}(XW - Y)^\mathsf{T}(XW - Y) + \frac{\lambda}{2}||W||_2^2$$

同时通过公式 (1.5)，我们可知 $C(W)$ 的一阶导为：

$$\begin{aligned} C'(W) &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + (||W||_2^2)' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + ((|w_0|^2 + |w_1|^2 + ... + |w_d|^2)^{\frac{1}{2}\times 2})' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + (|w_0|^2 + |w_1|^2 + ... + |w_d|^2)' \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + 2(|w_0| + |w_1| + ... + |w_d|) \\ &= \frac{1}{n}(X^\mathsf{T}XW - X^\mathsf{T}Y) + 2W \end{aligned}$$

将 $C'(W^*) = 0$ 代入上式可得：

$$\begin{aligned} \frac{1}{n}(X^\mathsf{T}XW^* - X^\mathsf{T}Y) + \lambda W^* &= 0 \\ \frac{1}{n}X^\mathsf{T}XW^* + \lambda W^* &= \frac{1}{n}X^\mathsf{T}Y \\ (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})W^* &= \frac{1}{n}X^\mathsf{T}Y \\ W^* &= (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})^{-1}(\frac{1}{n}X^\mathsf{T}Y) \end{aligned}$$

于是我们得到了公式 (1.9) 的解析解 $W^* = (\frac{1}{n}X^\mathsf{T}X + \lambda\mathbb{I})^{-1}(\frac{1}{n}X^\mathsf{T}Y)$。这个正则化后的解析解有一个非常好的性质

那就是不要求 $X^\mathsf{T}X$ 是可逆的（invertible），因为 $\frac{1}{n}X^\mathsf{T}X+\lambda\mathbb{I}$ 几乎总是可逆的（一个矩阵可逆的充要条件是它的所有特征值都不为零，而 $\frac{1}{n}X^\mathsf{T}X+\lambda\mathbb{I}$ 的所有特征值都严格大于零。）

1.7 算法实验

现在让我们回到最开头的案例，我们想预估我们上学的通勤时间，现在我们有下列数据：

- 距离 (Km): [2.7, 4.1, 1.0, 5.2, 2.8]
- 时段 (1: if weekday, 0: if weekend): [1, 1, 0, 1, 0]
- 通勤时间 (min): [25, 33, 15, 45, 22]

代如上述公式 (1.6) 求解出的输出应该为：

$$W^* = \begin{bmatrix} 6.08613445378149 \\ 6.53361344537816 \\ 2.11274509803922 \end{bmatrix}, Y^* = \begin{bmatrix} 25.8396358543418 \\ 34.9866946778712 \\ 12.6197478991597 \\ 42.1736694677872 \\ 24.3802521008403 \end{bmatrix}, Y = \begin{bmatrix} 25 \\ 33 \\ 15 \\ 45 \\ 22 \end{bmatrix}$$

其中 W^* 为模型的最优权重，Y^* 为模型的最优预测输出，Y 为模型的期望输出（即真实数据）。上述例子在 GitHub 仓库提供了 Matlab 代码（Linear_Regression.m）实现，其可视化的结果如图 1.1 所示。

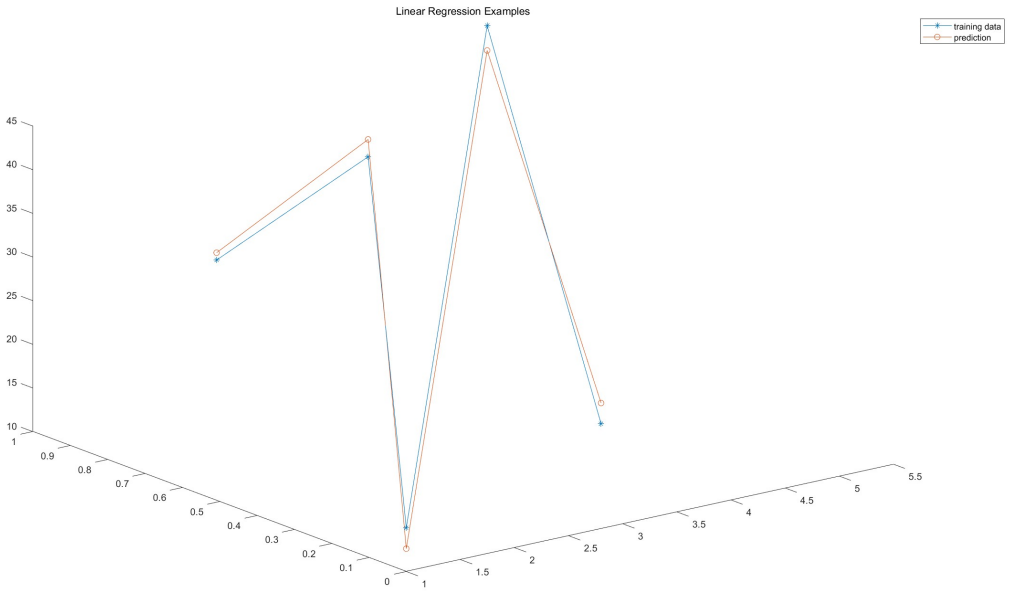

图 1.1: 通勤时间的回归建模结果

第二章 梯度下降

内容提要

- 算法原理
- 随机梯度下降

2.1 算法原理

梯度下降（Gradient Descent）是最小化目标函数 $C(W)$ 的通用算法之一（由柯西于 1847 年首次提出）。它是一种迭代算法，从 $W^{(t)}$ 开始，每次迭代都会产生一个新的 $W^{(t+1)}$，其公式 (2.1) 如下：

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C(W^{(t)}),\ (t = 0, 1, ..., n) \tag{2.1}$$

其中 η_t 称为**学习率**（Learning Rate）或**步长**（Step Size）。因此梯度下降 (2.1) 使用负梯度作为搜索方向，并沿着该方向移动 η_t 的距离。

接下来我们证明梯度下降 (2.1) 的合理性与可行性。根据一阶泰勒近似 $f(x) \approx f(a) + f'(a)(x-a)$ 的形式，那么存在下列近似关系：

$$C(W^{(t+1)}) \approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(W^{(t+1)} - W^{(t)})$$

接着将公式 (2.1) 代入上述式子中可以得到：

$$\begin{aligned} C(W^{(t+1)}) &\approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(W^{(t)} - \eta \nabla C(W^{(t)}) - W^{(t)}),\ (\eta \to 0) \\ &\approx C(W^{(t)}) + \nabla C(W^{(t)})^{\mathsf{T}}(-\eta \nabla C(W^{(t)})) \\ &\approx C(W^{(t)}) - \eta \nabla C(W^{(t)})^{\mathsf{T}} \nabla C(W^{(t)}) \end{aligned}$$

因为 $\nabla C(W)^{\mathsf{T}} \nabla C(W) = ||\nabla C(W)||_2^2 > 0$，所以即证 $C(W^{(t+1)}) < C(W^{(t)})$。

笔记 我们现在可以尝试利用梯度下降法来求解线性回归问题。

对于最小二乘回归 (1.4)，我们已经得到了 $C(W)$ 的一阶导形式 (1.5)，套用梯度下降公式 (2.1) 可以直接得到 W^* 数值解的求解方式：

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \eta_t \nabla C(W^{(t)}) \\ &= W^{(t)} - \frac{\eta}{n}(X^{\mathsf{T}} X W^{(t)} - X^{\mathsf{T}} Y) \end{aligned}$$

继续套用通勤时间计算的例子，GitHub 仓库提供了 Matlab 代码（Gradient_Descent_for_Linear_Regression.m）实现。

2.2 随机梯度下降

通过上面的学习，我们可以知道 $\nabla C(W^{(t)})$ 的一般形式：

$$\nabla C(W^{(t)}) = \frac{1}{n} \sum_{i=1}^{n} \nabla C_i(W^{(t)})$$

其中 $C_i(W^{(t)})$ 表示的是模型对数据点 i 的预测值与期望值之间的差距。通过上述式子我们不难发现，模型需要遍历所有的数据才可以计算出梯度，如果数据特别多这种方式的计算消耗将会特别大。为了解决这个问题，人们发明了一种梯度下降法的变体叫做**随机梯度下降**（Stochastic Gradient Descent，SGD）(2.2)。

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C_i(W^{(t)}),\ (\eta_t = C/\sqrt{t}) \tag{2.2}$$

其中 $\nabla C_i(W^{(t)})$ 在每次迭代时将从数据集中进行随机且均匀地抽样。随机梯度是真实梯度的带有噪声的估计，如果学习率太大且固定不变，那么噪声的影响会被放大从而导致权重的更新步伐会非常不稳定，甚至可能无法收

敛到最小值点，而是被噪声推着越跑越远。所以我们在训练过程中一般会逐步降低学习率，一个典型的选择是：

$$\eta_t = \frac{C}{\sqrt{t}}$$

其中 C 是一个需要根据实际情况调整的超参数，t 是迭代次数。也就是说，随着迭代次数的增加，我们使用的学习率会越来越小。对于梯度下降与随机梯度下降之间的差异，我们可以总结以下几点：

- 如果我们想要一个高精度的解决方案，并且样本量较小，那么梯度下降是好的选择。但如果我们想要一个精度适中的解决方案，并且问题规模较大，那么使用随机梯度下降可能是更好的选择。
- 随机梯度下降的计算成本与样本大小无关，因此 SGD 对于大规模问题尤其有效。
- 梯度下降是平滑的，因为负梯度是函数值减小的方向。随机梯度下降是不稳定的，因为随机梯度是对真实梯度的带有噪声的估计。

笔记 为什么要采用 $\eta_t = \frac{C}{\sqrt{t}}$ 形式的学习率设计？

为了确保 SGD 最终能够收敛到最优解（或其附近），我们必须克服噪声的影响。这里我们遵从 Robbins-Monro 算法收敛条件，其为随机近似问题提供了几乎必然收敛（Almost Surely Convergence）的充分条件。应用于 SGD 的语境下，我们希望学习率序列满足下列条件：

1. $\sum_{t=1}^{\infty} \eta_t \to \infty$
2. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$

第一个条件确保算法拥有足够的“能量”从任意初始点到达最优解，即使初始点离得很远。第二个条件确保梯度估计中的噪声（方差）能够被逐渐衰减至零，从而保证算法最终稳定在最优解附近。接下来我们证明 $\eta_t = \frac{C}{\sqrt{t}}$（其中 $C > 0$ 是一个常数）的形式满足上述两个条件。

条件一：$\sum_{t=1}^{\infty} \eta_t = \sum_{t=1}^{\infty} \frac{C}{\sqrt{t}} \to \infty$。这个我们可以通过将其与一个著名的发散级数（调和级数）进行比较来证明这个级数是发散的。对于任意 $t \geq 1$，我们有：

$$\frac{1}{\sqrt{t}} \geq \frac{1}{t}$$

我们知道调和级数 $\sum_{t=1}^{\infty} \frac{1}{t}$ 是发散的，一个简便的证明是：

$$\begin{aligned}\sum_{t=1}^{\infty} \frac{1}{t} &\geq \int_1^n \frac{1}{x} dx \\ &= \ln x \Big|_1^n \\ &= (\ln x - \ln 1) \to \infty\end{aligned}$$

由于 $\frac{1}{\sqrt{t}}$ 中每一项都大于或等于 $\frac{1}{t}$ 的对应项，所以 $\sum_{t=1}^{\infty} \frac{1}{\sqrt{t}}$ 也发散，即 $\sum_{t=1}^{\infty} \eta_t = \sum_{t=1}^{\infty} \frac{C}{\sqrt{t}} \to \infty$

条件二：$\sum_{t=1}^{\infty} \eta_t^2 = \sum_{t=1}^{\infty} \left(\frac{C}{\sqrt{t}}\right)^2 = \sum_{t=1}^{\infty} \frac{C^2}{t} < \infty$。现在我们来看这个级数是否收敛，我们首先进行如下变换：

$$\sum_{t=1}^{\infty} \eta_t^2 = C^2 \sum_{t=1}^{\infty} \frac{1}{t}$$

调和级数（Harmonic Series）$\sum_{t=1}^{\infty} \frac{1}{t}$ 在前面我们已经证明是发散的，所以实际上 $\eta_t = \frac{C}{\sqrt{t}}$ 并不严格满足 Robbins-Monro 的第二个条件。

接下来让我们重新审视 $\eta_t = \frac{C}{\sqrt{t}}$ 的设计。尽管 η_t 发散，但其发散速度是对数级的（类似于 $\ln(t)$）。在有限步（例如 $T = 10^6$）的迭代中，累积的 $\sum_{t=1}^{T} \eta_t^2$ 仍然是一个可控的有限值。但在实践中如果采用 $\eta_t = \frac{C}{t}$ 的形式，其确实满足 Robbins-Monro 的两个条件，但其衰减速度过快，在实践中性能往往不如 $\frac{C}{\sqrt{t}}$。所以，$\frac{C}{\sqrt{t}}$ 的衰减速度在“保证充分搜索”($\sum \eta_t = \infty$) 和“抑制噪声”之间提供了一个非常好的实践折衷。它比 $\frac{C}{t}$ 衰减得慢，允许算法在后期仍然保持一定的探索能力，从而在实践中通常获得更好的性能。

2.2.1 小批量随机梯度下降

在了解了基本的随机梯度下降算法之后，接下来给大家介绍随机梯度下降的衍生算法，**小批量随机梯度下降** (Minibatch Stochastic Gradient Descent)。小批量随机梯度下降不再每次只抽取一个样本计算梯度，而是抽取多个样本，其数学表达式为 (2.3)：

$$W^{(t+1)} = W^{(t)} - \frac{\eta_t}{b} \sum_{i \in B} \nabla C_i(W^{(t)}) \tag{2.3}$$

其中 B 为随机抽取的样本的集合，$b = \mathsf{length}(B)$。

这里我们可以采用小批量随机梯度下降的方法继续再次求解上述的通勤时间预测问题。我们设定 $C = 0.06$，$b = 2$，且当 $|W^{(t+1)} - W^{(t)}| < 0.02$ 时停止迭代。GitHub 仓库提供了 Matlab 代码（MiniSGD_for_Linear_Regression.m）实现。

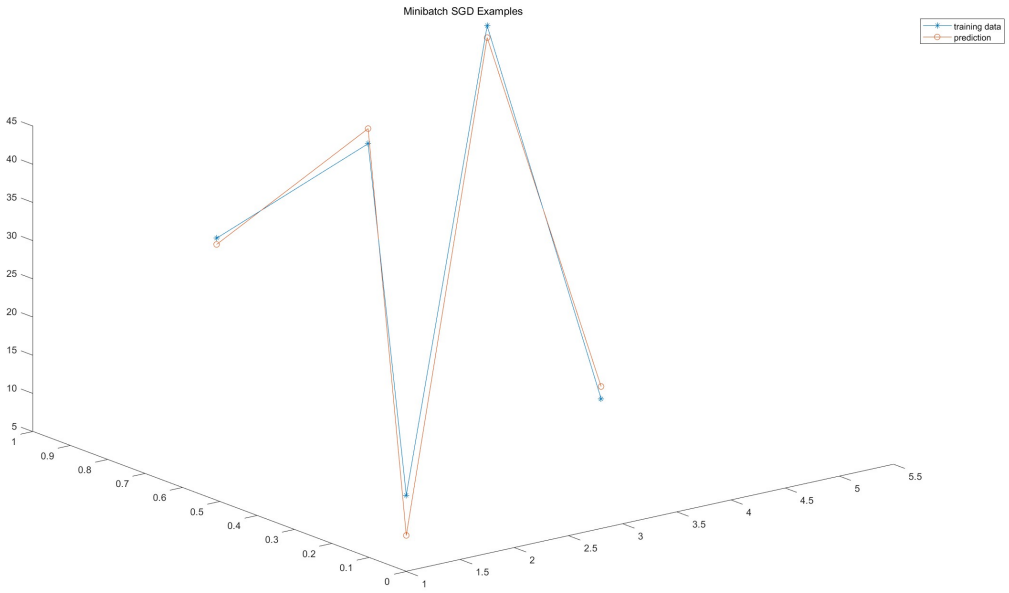

图 2.1: 小批量随机梯度下降的回归建模结果

笔记 这里有两种采样方式：重复采样和不重复采样。这两个概念的核心区别在于从总体中抽取样本后是否将样本个体放回总体中再进行下一轮采样。一般我们采用不重复采样。

2.2.2 求解线性二分类问题

在 1.4 小节中我们已经推导出了 $\nabla C_i(W)$ 的表达式 (1.7)，但我们难以直接求解 W^* 的解析解，这时通过随机梯度下降法我们就可以方便快速地迭代出 W^* 的数值解：

$$W^{(t+1)} = W^{(t)} - \eta_t \nabla C_i(W^{(t)}) = \begin{cases} W^{(t)}, & \text{if } y^{(i)} {W^{(t)}}^{\mathsf{T}} X^{(i)} \geq 1 \\ W^{(t)} - \eta_t (W^{\mathsf{T}} x^{(i)} - y^{(i)}) x^{(i)}, & otherwise. \end{cases}$$

我们在 GitHub 仓库提供了利用随机梯度下降（及小批量随机梯度下降）求解花朵品种预测的二分类问题的 Matlab 代码（MiniSGD_for_Linear_Classification.m）实现。

第三章 感知器和神经网络

3.1 感知器

感知器（Perceptron）算法是一种二分类的线性分类模型，也是最早最简单的人工神经网络之一。它在 1958 年由 Frank Rosenblatt 提出，是深度学习领域的重要基石。

让我们继续考虑上一章节的二分类问题，假设存在输入输出对 S：

$$S = \left\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\right\},\ y^{(i)} \in \{-1, +1\}$$

这里 y 的取值只是一种形式化的表示，目的在于方便模型进行二分类预测。同时，我们需要一个**激活函数**将感知机的线性计算（加权求和）转换为一个非线性的决策（分类结果），从而让感知机能够执行分类任务而不仅仅是做线性回归。此外，激活函数的另一个关键意义在于：它使得感知器学习算法得以实现。在这里感知器使用的是一种叫单位阶跃函数的激活函数，结合线性模型的表达式 $y = W^{\mathsf{T}}x + b$ 我们可以得到感知器模型的输出 $\hat{y}$ 的一个完整表达式：

$$\hat{y} = \begin{cases} 1, & \text{if } (W^{\mathsf{T}}x + b) > 0 \\ -1, & otherwise. \end{cases}$$

接着我们继续使用间隔的概念，当 $y\hat{y} = yW^{\mathsf{T}}x < 0$ 时，这意味着模型没有正确预测该样本的类别，那么我们就需要更新模型的权重来让模型能够进行正确的预测。所以下面我们开始介绍感知器的核心算法步骤：

1. 初始化：将权重 W 和偏置 b 初始化为 0 或很小的随机数。
2. 判断是否所欲样本都被正确分类。如果都被正确分类则程序终止，否则跳转到步骤 3。
3. 对所有样本进行遍历，如果 $yW^{\mathsf{T}}x < 0$，则更新权重 $W = W + \eta(y - \hat{y})x$ 和偏置 $b = b + \eta(y - \hat{y})$。所有样本遍历完后跳转到步骤 2。

证明 这里我们来证明为什么 $W = W + \eta(y - \hat{y})x$ 能够有效更新权重并优化模型的输出。首先我们直接对 $yW_{new}^{\mathsf{T}}x$ 进行展开，有如下推导过程：

$$\begin{aligned} yW_{new}^{\mathsf{T}}x &= y(W_{old} + \eta(y - \hat{y})x)^{\mathsf{T}}x \\ &= yW_{old}^{\mathsf{T}}x + \eta(y^2 - y\hat{y})x^{\mathsf{T}}x \\ &= yW_{old}^{\mathsf{T}}x + C \end{aligned}$$

接着我们开始分析 $C = \eta(y^2 - y\hat{y})x^{\mathsf{T}}x$ 这一项，其中 $\eta > 0$，$y^2 > 0$，$x^{\mathsf{T}}x > 0$，$y\hat{y} < 0$，所以我们可以知道 C 这个常数项恒大于 0，即 $yW_{new}^{\mathsf{T}}X > yW_{old}^{\mathsf{T}}X$。所以感知器的更新迭代策略是有效的。

3.2 单层神经网络

感知机的一个局限性在于它只能构建线性分类器。具体而言，它根据样本位于超平面某一侧的位置对其进行分类。同时感知机还具有以下缺陷：

1. 感知机仅在数据线性可分时才能收敛，对于线性不可分的数据（即不存在能够完全区分正负样本的超平面）则无法停止迭代。
2. 权值 W 仅针对误分类样本进行调整（正确分类的样本完全不被考虑）。
3. 多个感知机可以形成复杂的决策边界（分段线性：例如通过两个感知机将空间划分为四个区域，从而实现正负样本的分离），但训练难度较大。

所以我们得想办法改进感知机，一个思路就是用可微的非线性函数替换掉感知机原始的激活函数，这里我们使用 Sigmoid 函数 (3.1)：

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

Sigmoid 函数可以将区间 $(-\infty, +\infty)$ 映射到 $(0, 1)$，且 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$。单层神经网络与感知机的唯一区别在于：前者使用高度非线性的 sigmoid 函数替代了感知机中的单位阶跃函数。此外 Sigmoid 函数可以输出任意实数值，这意味着单层神经网络能够应用于回归问题。

证明 这里我们来证明 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$：

$$
\begin{aligned}
\sigma(x) &= \frac{1}{1 + e^{-x}} \\
\sigma'(x) &= \frac{-(-e^{-x})}{(1 + e^{-x})^2} \\
\sigma'(x) &= \frac{1}{1 + e^{-x}} \bullet \frac{e^{-x} + 1 - 1}{1 + e^{-x}} \\
\sigma'(x) &= \frac{1}{1 + e^{-x}}(1 - \frac{1}{1 + e^{-x}}) \\
\sigma'(x) &= \sigma(x)(1 - \sigma(x))
\end{aligned}
$$

现在让我们重新考虑非线性回归问题，套用公式 (1.3) 我们可以快速得知单层神经网络的均方误差表达式为 (3.2)：

$$
C(W) = \frac{1}{2n} \sum_{i=1}^{n} (\sigma(W^{\mathsf{T}} x^{(i)} + b) - y^{(i)})^2 \tag{3.2}
$$

不同于线性回归问题，我们难以直接求解单层神经网络的 $\nabla C(W^*) = 0$ 的解析解，但我们任然可以通过梯度下降法求解 W^* 的近似数值解：

$$
W^* = W - \eta \nabla C(W)
$$

即求解 (3.3)

$$
w_i^* = w_i - \eta \frac{\partial C(W)}{\partial w_i} \tag{3.3}
$$