

# 线性回归

np.ndarray 可以有 1 ~ n 维, np.matrix 的矩阵是 2 维. 在线性回归中主要涉及到一维数组和二维数组/矩阵的运算.

## 线性回归的实现过程

- Step1: hypothesis

$$\begin{aligned} h_{\theta}(x) &= X\theta^T \\ h_{\theta}(x^{(i)}) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \\ \theta &= (\theta_0, \theta_1, \theta_2, \dots, \theta_n)^T \\ X &= (x^{(0)}, x^{(1)}, x^{(2)}, \dots, x^{(m)})^T \\ x^{(i)} &= (x_0, x_1, x_2, \dots, x_n) \end{aligned}$$

The notes show a handwritten table for m samples, where each sample has n features plus a bias term  $x_0$ . The columns are labeled  $x_0, x_1, x_2, \dots, x_n$  and  $y$  (目标). The rows are labeled  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ . Below the table, the matrix equation  $X = [x^{(1)} \ x^{(2)} \ \dots \ x^{(m)}] = [x_0^{(1)} \ x_1^{(1)} \ x_2^{(1)} \ \dots \ x_n^{(1)} \ x_0^{(2)} \ x_1^{(2)} \ x_2^{(2)} \ \dots \ x_n^{(2)} \ \vdots \ \vdots \ \vdots \ x_0^{(m)} \ x_1^{(m)} \ x_2^{(m)} \ \dots \ x_n^{(m)}]$  is written. The parameter vector  $\theta = (\theta_0, \theta_1, \theta_2, \dots, \theta_n)$  is also defined. The hypothesis function  $h_{\theta}(x) = X \cdot \theta^T$  is shown at the bottom.

- Step2: Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

- Step3: Gradient Descent

$$\theta_j = \theta_j - \alpha \frac{\partial J_{\theta}}{\partial \theta_j}$$

$$\frac{\partial J_{\theta}}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m np.multiply((h_{\theta}(x^{(i)}) - y^{(i)}), X[:, j]), \text{ for } j \text{ in range}(n)$$

## 如何用 NumPy 实现公式的 Code

- 方法一: 用数组的方式, 在算矩阵乘法时用 np.dot(array1, array2). 如计算  $h_{\theta}(x)$
- 方法二: 用矩阵的方式, 定义为矩阵, 直接用 \* 计算矩阵乘法

```

x = np.array([[1, 2, 3],           # 2d.array (4, 3)
              [1, 3, 4],
              [1, 4, 5],
              [1, 5, 6]])
theta = np.array([0.1, 0.2, 0.3])      # 1d.array (3,)

X_mat = np.mat(x)                   # matrix (4, 3)
theta_mat = np.mat(theta)          # matrix (1,3)

"""

np.dot(a, b):
若都为一维向量，则结果为向量点积；  

若都为二维向量，则结果为矩阵乘法，当然要保持行列对应才能进行矩阵乘法；  

若前者为二维向量（也就是矩阵），后者为一维向量，则后者会被当做一维矩阵进行计算
"""

h1 = np.dot(x, theta)            # 1d.array (4,)
h2 = X_mat * theta_mat.T        # matrix (4,1)
h3 = np.dot(X_mat, theta_mat.T)  # matrix (4,1)

"""

np.multiply(a, b) 对应位置相乘，a b互换位置结果一样
a = [[1, 2, 3],                 b = [1, 2, 3] 或 b = [[1,
              [1, 3, 4],                  [2],
              [1, 4, 5],                  [3],
              [1, 5, 6]]                  [4]]
"""

"""

```

*np.dot()*

$a = \begin{bmatrix} 1 & 2 \end{bmatrix}$

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

**计算方式：**  
 $np.dot(A \cdot a)$

$\Rightarrow (2 \cdot 2) \quad (2 \cdot 1) = (2 \cdot 1)$

$np.dot(a, A)$

$\Rightarrow (1 \cdot 2) \quad (2 \cdot 2) = (1 \cdot 2)$

**但结果还是：**

$array([5, 11])$

$array([7, 10])$