

# 线性回归 (Linear Regression)

Guangyao Zhao

2022-01-13

## Contents

数学表示符号	1
非矩阵形式	1
矩阵形式	2
代码	4
非第三方库	4
Sklearn	5
正则项	6

## 数学表示符号

给定由  $m$  个样本,  $n$  个属性描述的示例  $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_m, y_m)\}$ ,  $j$  表示第  $j$  个样本值。其中  $x^{(j)} \in \mathbb{R}^n$ , 每个样本又包含  $n$  个维度, 即  $x^{(j)} = \{x_1^{(j)}, x_2^{(j)}, \dots, x_m^{(j)}\}$ ,  $y^{(j)} \in \mathbb{R}$ 。

## 非矩阵形式

线性模型试图学习到一个通过属性的线性组合来预测的函数, 即:

$$\begin{aligned} f(x) &= w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b \\ &= w^T x + b \end{aligned} \tag{1}$$

其中:  $w = \{w_1; w_2; w_3; \dots; w_n\}$ ,  $x = \{x_1; x_2; x_3; \dots; x_n\}$ 。

线性回归试图寻求能使损失函数 (Cost function):

$$J(w, b) = \frac{1}{2m} \sum_{j=1}^m (f(x^{(j)}) - y^{(j)})^2 \quad (2)$$

最小的  $w$  和  $b$ , 对 Eq. 2 求导可得:

$$\frac{\partial J_{(w,b)}}{\partial w_i} = \frac{1}{m} \sum_{j=1}^m (w^T x^{(j)} + b - y^{(j)}) x_i^{(j)} \quad (3)$$

$$\frac{\partial J_{(w,b)}}{\partial b} = \frac{1}{m} \sum_{j=1}^m (w^T x^{(j)} + b - y^{(j)}) \quad (4)$$

更新  $w, b$  :

$$w_i := w_i - \alpha \frac{\partial J_{(w,b)}}{\partial w_i}$$

$$b := b - \alpha \frac{\partial J_{(w,b)}}{\partial b}$$

其中  $\alpha$  为学习率。

## 矩阵形式

$w$  的表示如下:

$$w = \{w_1; w_2; w_3; \dots; w_n\}$$

所有样本  $\mathbf{X}$  的表示如下:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} & 1 \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} & 1 \end{pmatrix} = \begin{pmatrix} x^{(1)} & 1 \\ x^{(2)} & 1 \\ \vdots & \vdots \\ x^{(m)} & 1 \end{pmatrix}$$

$y$  的的标记如下:

$$y = \{y^{(1)}; y^{(2)}; y^{(3)}; \dots; y^{(m)}\}$$

目标函数的推导如下:

$$\begin{aligned}
L(w) &= \frac{1}{2m} \sum_{j=1}^m \|w^T x^{(j)} - y^{(j)}\|^2 \\
&= \frac{1}{2m} (w^T x^{(1)} - y^{(1)}, w^T x^{(2)} - y^{(2)}, \dots, w^T x^{(m)} - y^{(m)}) \begin{pmatrix} w^T x^{(1)} - y^{(1)} \\ w^T x^{(2)} - y^{(2)} \\ \vdots \\ w^T x^{(m)} - y^{(m)} \end{pmatrix} \\
&= \frac{1}{2m} [(w^T x^{(1)}, w^T x^{(2)}, \dots, w^T x^{(m)}) - (y^{(1)}, y^{(2)}, \dots, y^{(m)})] \left[ \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix} w - \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \right] \quad (5) \\
&= \frac{1}{2m} (w^T \mathbf{X}^T - y^T) (\mathbf{X}w - y) \\
&= \frac{1}{2m} (\mathbf{X}w - y)^T (\mathbf{X}w - y)
\end{aligned}$$

则，目标函数为：

$$\operatorname{argmin}_w \frac{1}{2m} (\mathbf{X}w - y)^T (\mathbf{X}w - y) \quad (6)$$

对于目标函数化简：

$$\begin{aligned}
J(w) &= \frac{1}{2m} (\mathbf{X}w - y)^T (\mathbf{X}w - y) \\
&= \frac{1}{2m} w^T \mathbf{X}^T \mathbf{X} w - w^T \mathbf{X}^T y - y^T \mathbf{X} w + y^T y \\
&= \frac{1}{2m} w^T \mathbf{X}^T \mathbf{X} w - 2w^T \mathbf{X}^T y + y^T y
\end{aligned} \quad (7)$$

对上式（即目标函数）求导：

$$\frac{\partial L(w)}{\partial w} = \frac{1}{2m} (2\mathbf{X}^T \mathbf{X} w - 2\mathbf{X}^T y) = 0 \quad (8)$$

求解可得到解析解： $w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y$ 。在进行梯度下降的时候也是利用该公式， $w$  的更新公式为：

$$w := w - \frac{\alpha}{m} \mathbf{X}^T (\mathbf{X}w - y) \quad (9)$$

## 代码

### 非第三方库

```
1
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from sklearn.datasets import load_boston
5
6
7 def load_dataset():
8     X, y = load_boston(return_X_y=True)
9     return X, y
10
11
12 def normalize(X):
13     return (X - np.mean(X, axis=0)) / np.std(X, axis=0)
14
15
16 def cost_function(X, y, w):
17     m = X.shape[0] # samples
18     y_pre = f(X, w) # (m, 1)
19     return float(np.matmul((y - y_pre).T, (y - y_pre)) / (2 * m))
20
21
22 def f(X, w):
23     return np.matmul(X, w)
24
25
26 def gradient(X, y, w, learning_rate=0.01, epoch=100):
27     J_all = [] # 存放 loss
28     m = X.shape[0]
29     for i in range(epoch):
30         y_pre = f(X, w)
31         cost_ = np.matmul(X.T, y_pre - y) / m # 重点, 以矩阵形式更新参数
32         w = w - learning_rate * cost_
```

```

33     J_all.append(cost_function(X, y, w))
34     return w, J_all
35
36
37 def plot_cost(J_all):
38     fig = plt.figure(figsize=(8, 6),
39                       facecolor='pink',
40                       frameon=True,
41                       edgecolor='green',
42                       linewidth=2)
43     ax1 = fig.add_subplot(111, xlabel='Epoch', ylabel='Loss')
44     ax1.plot(J_all)
45
46
47 # =====
48 X, y = load_dataset() # 加载数据集
49 X = np.array(X)
50 y = np.array(y).reshape((y.size, 1))
51
52 X = normalize(X) # 标准化
53 X = np.hstack((X, np.ones((X.shape[0], 1)))) # 变化 X
54 w = np.ones((X.shape[1], 1)) # 初始化 w
55 learning_rate = 0.1
56 epoch = 100
57
58 w, J_all = gradient(X, y, w, learning_rate, epoch)
59 plot_cost(J_all)

```

## Sklearn

一般情况下是不需要自己造轮子的，而且自己写的代码并不一定在运行速度上有优势，所以大多数情况下是直接调用第三方库，比如参考[sklearn 官方文档](#)：

```

1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import StandardScaler
3

```

```

4 def load_dataset():
5     X, y = load_boston(return_X_y=True)
6     return X, y
7
8 X, y = load_dataset() # 加载数据集
9 X = np.array(X)
10 y = np.array(y).reshape((y.size, 1))
11
12 X = StandardScaler().fit_transform(X) # 归一化
13 reg = LinearRegression(fit_intercept=True).fit(X, y)
14
15 score = reg.score(X,y) # R2
16 print('score: ', score)

```

## 正则项

当样本数较小的时候，模型容易过拟合，此时一般有三种办法解决：

1. 增加样本量
2. 提取出有效特征，减少维度
3. 正则化

其中正则化是一种最常用且最方便的方法，正则化的思想是使参数  $w$  的尽量小，因为当其很大的时候容易造成不稳定，比如即使  $x$  稍微有点波动，由于  $w$  太大，也会造成  $y$  剧烈波动。

线性回归中， $L_2$  正则的代价函数如下：

$$J(w) = \sum_{j=1}^m \|w^T x^{(j)} - y^{(j)}\|^2 + \lambda w^T w \quad (10)$$

将 Eq. 10 和 Eq. 7 相结合可得  $w$  的更新公式：

$$w := w - (\alpha \mathbf{X}^T (\mathbf{X}w - y) - \lambda w) \quad (11)$$

非矩阵形式的更新公式如下：

$$w_i := w_i - \left( \alpha \frac{1}{m} \sum_{j=1}^m (w^T x^{(j)} - y^{(j)}) x_i^{(j)} - \lambda w_i \right)$$

```

1 def load_dataset():
2     X, y = load_boston(return_X_y=True)
3     return X, y
4
5
6 def normalize(X):
7     return (X - np.mean(X, axis=0)) / np.std(X, axis=0)
8
9
10 def cost_function(X, y, w, alpha=0.1):
11     m = X.shape[0] # samples
12     y_pre = f(X, w) # (m, 1)
13     return float(np.matmul((y - y_pre).T, (y - y_pre)) /
14                 (2 * m)) + alpha / 2 + float(alpha / 2 * np.matmul(w.T, w))
15
16
17 def f(X, w):
18     return np.matmul(X, w)
19
20
21 def gradient(X, y, w, learning_rate=0.01, epoch=100, alpha=0.1):
22     J_all = [] # 存放 loss
23     m = X.shape[0]
24     for i in range(epoch):
25         y_pre = f(X, w)
26         cost_ = np.matmul(X.T, y_pre - y) / m + alpha * w # 重点, 以矩阵形式更新参数
27         w = w - learning_rate * cost_
28         J_all.append(cost_function(X, y, w, alpha))
29     return w, J_all
30
31
32 def plot_cost(J_all):
33     fig = plt.figure(figsize=(8, 6),
34                       facecolor='pink',
35                       frameon=True,
36                       edgecolor='green',

```

```

37         linewidth=2)
38     ax1 = fig.add_subplot(111, xlabel='Epoch', ylabel='Loss')
39     ax1.plot(J_all)
40     fig.savefig('test.pdf')
41
42
43 # =====
44 X, y = load_dataset() # 加载数据集
45 X = np.array(X)
46 y = np.array(y).reshape((y.size, 1))
47
48 X = normalize(X) # 标准化
49 X = np.hstack((X, np.ones((X.shape[0], 1)))) # 变化 X
50 w = np.ones((X.shape[1], 1)) # 初始化 w
51 learning_rate = 0.1
52 epoch = 1000
53
54 w, J_all = gradient(X, y, w, learning_rate, epoch, alpha=1)
55 print('w: ', w)
56 plot_cost(J_all)

```