机器学习-线性回归

线性回归

回归问题是非常常见的一类问题，目的是寻找变量之间的关系。比如要从数据中寻找房屋面积与价格的关系，年龄和身高的关系，气体压力和体积的关系等等。而机器学习要做的正是要让机器自己来学习这些关系，并为对未知的情况做出预测。

对于线性回归，假设变量之间的关系是线性的，即：

$$h_\theta(x) = \theta_0 + \theta_1 x$$

其中 $\boldsymbol{\theta}$ 就是学习算法需要学习的参数，在线性回归的问题上，就是θ_1和θ_0，而 x 是我们对于问题所选取的特征，也即输入。h表示算法得到的映射。

代价函数的表示

为了找到这个算法中合适的参数，我们需要制定一个标准。一般而言算法拟合出来的结果与真实的结果误差越小越好，试想一下如果算法拟合出来的结果与真实值的误差为零，那么就是说算法完美地拟合了数据。所以可以根据“真实值与算法拟合值的误差”来表示算法的“合适程度”。在线性回归中，我们经常使用最小二乘的思路构建代价函数：

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

这里 $h_\theta(x^{(i)})$ 由假设模型得出。对线性回归任务，代价函数可以展开为：

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

误差函数的值越小，则代表算法拟合结果与真实结果越接近。

梯度下降

梯度下降算法沿着误差函数的反向更新θ的值，知道代价函数收敛到最小值。梯度下降算法更新θ_i的方法为：

$$\theta_i = \theta_i - \alpha\frac{\partial}{\partial\theta_i}J(\boldsymbol{\theta})$$

其中 α表示学习率。对于线性回归的的参数，可以根据代价函数求出其参数更新公式：

$$\frac{\partial J}{\partial\theta_0} = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot 1,$$

$$\frac{\partial J}{\partial\theta_1} = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}.$$

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
def plot_data(x, y):
    """绘制给定数据x与y的图像"""
    plt.figure()
    # ==========================================================
    # 绘制x与y的图
    # 使用 matplotlib.pyplt 的命令 plot, xlabel, ylabel 等。
    # 提示：可以使用 'rx' 选项使数据点显示为红色的 "x"，
    #       使用 "markersize=8, markeredgewidth=2" 使标记更大
    # 给制数据
    # 设置y轴标题为 'Profit in $10,000s'
    plt.plot(x, y, 'rx', label='Data', markersize=8, markeredgewidth=3);
    plt.ylabel('Profit in $10,000s');
    # 设置x轴标题为 'Population of City in 10,000s'
    plt.xlabel('Population of City in 10,000s');
    plt.legend("loc='lower right'")
    # ==========================================================
    plt.show()


# 从txt中加载数据
print('Plotting Data ...\n')
#根据数据的实际位置进行下载，这个网页notebook存放数据的位置
data = np.loadtxt('./data/data5984/PRML_LR_data.txt', delimiter=',')
x, y = data[:, 0], data[:, 1]
# 绘图
plot_data(x, y)
plt.show()


# Add a column of ones to x
m = len(y)
X = np.ones((m, 2))
X[:, 1] = data[:, 0]
# initialize fitting parameters
theta = np.zeros((2, 1))
# Some gradient descent settings
iterations = 1500
alpha = 0.01
```

```
def compute_cost(X, y, theta):
    """计算线性回归的代价。"""
    m = len(y)
    J = 0.0
    # ============================================================
    # 计算给定 theta 参数下线性回归的代价
    # 请将正确的代价赋值给 J
    h_theta = theta[0] * X[:, 0].T + theta[1] * X[:, 1].T
    J = (1.0 / (2 * m)) * sum((h_theta - y) ** 2)
    # ============================================================
    return J
# compute and display initial cost
# Expected value 32.07
J0 = compute_cost(X, y, theta)
print(J0)



def gradient_descent(X, y, theta, alpha, num_iters):
    """执行梯度下降算法来学习参数 theta。"""
    m = len(y)
    J_history = np.zeros((num_iters,))
    for iter in range(num_iters):
        # ============================================================
        # 计算给定 theta 参数下线性回归的梯度，实现梯度下降算法
        h_theta = theta[0] * X[:, 0].T + theta[1] * X[:, 1].T
        temp0 = theta[0] - (alpha / m) * np.dot((h_theta - y), (X[:, 0]))
        temp1 = theta[1] - (alpha / m) * np.dot((h_theta - y), (X[:, 1]))
        theta[0] = temp0
        theta[1] = temp1
        # ============================================================
        # 将各次迭代后的代价进行记录
        J_history[iter] = compute_cost(X, y, theta)
    return theta, J_history
# run gradient descent
# Expected value: theta = [-3.630291, 1.166362]
theta, J_history = gradient_descent(X, y, theta,
                                    alpha, iterations)
print(theta)



plt.figure()
plt.plot(x, y, 'rx',markersize=8, markeredgewidth=2)
plt.ylabel('Profit in $10,000s')
```

```
plt.xlabel('Population of City in 10,000s')
y_predict = theta[1]*x+theta[0]
plt.plot(x,y_predict)
plt.show()

def plot_visualize_cost(X, y, theta_best):
    """可视化代价函数"""
    # 生成参数网格
    theta0_vals = np.linspace(-10, 10, 101)
    theta1_vals = np.linspace(-1, 4, 101)
    t = np.zeros((2, 1))
    J_vals = np.zeros((101, 101))
    for i in range(101):
        for j in range(101):
            # ===========================================
            # 加入代码，计算 J_vals 的值
            J_vals[i][j] = compute_cost(X,y,[theta0_vals[i],theta1_vals[j]])
            # ===========================================
    plt.figure()
    plt.contour(theta0_vals, theta1_vals, J_vals,
                levels=np.logspace(-2, 3, 21))
    plt.plot(theta_best[0], theta_best[1], 'rx',
             markersize=8, markeredgewidth=2)
    plt.xlabel(r'$\theta_0$')
    plt.ylabel(r'$\theta_1$')
    plt.title(r'$J(\theta)$')
plot_visualize_cost(X, y, theta)
plt.show



def plot_visual_history(X, y, theta, J_history):
    """可视化线性回归模型迭代优化过程"""
    theta0_vals = np.linspace(-10, 10, 101)
    theta1_vals = np.linspace(-1, 4, 101)
    T0, T1 = np.meshgrid(theta0_vals, theta1_vals)
    J_vals = np.zeros((101, 101))
    for i in range(101):
        for j in range(101):
            theta_test = [theta0_vals[i], theta1_vals[j]]
            J_vals[i][j] = compute_cost(X, y, theta_test)
    plt.figure(figsize=(12, 4))
    # 画代价函数的等高线图
    plt.subplot(121)
```

```python
    plt.contour(T0, T1, J_vals, levels=np.logspace(-2, 3, 21))
    plt.plot(theta[0], theta[1], 'rx', markersize=8, markeredgewidth=2)
    plt.xlabel(r'$\theta_0$')
    plt.ylabel(r'$\theta_1$')
    plt.title(r'$J(\theta)$ - Contour Plot')
    # 画代价函数的迭代过程
    plt.subplot(122)
    plt.plot(range(len(J_history)), J_history, 'b-')
    plt.xlabel('Iterations')
    plt.ylabel(r'$J(\theta)$')
    plt.title('Cost Function History')
    plt.show()
# 调用函数
plot_visual_history(X, y, theta, J_history)
```