

梯度下降法做线性回归：

估计函数：

$$h(x) = h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

误差函数：

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

在选定线性回归模型后，只需要确定参数 $\theta$ ，就可以将模型用来预测。然而 $\theta$ 需要在 $J(\theta)$ 最小的情况下才能确定。因此问题归结为求极小值问题，使用梯度下降法。梯度下降法最大的问题是求得有可能是全局极小值，这与初始点的选取有关。

梯度下降法是按面的流程进行：

- 1) 首先对  $\theta$  初始化，这个可以是随机的也让  $\theta$  是一个全零的向量。
- 2) 改变  $\theta$  的值，使得  $J(\theta)$  按梯度下降的方向减少

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

单变量实现：

```
1. # coding: utf-8
2. import pandas as pd
3. import numpy as np
4. import matplotlib.pyplot as plt
5.
6. def plotData(x, y):
7.     plt.title('mytest')
8.     plt.xlabel('Population of City in 10,000s')
9.     plt.ylabel('Profit in $10,000s')
10.    plt.plot(x, y, 'rx')
11.    plt.show()
12.
13. def gradientDescent(x, y, theta, alpha, num_iters):
14.     m = len(y)
15.     J_history = np.zeros((num_iters, 2))
16.     for i in range(num_iters):
17.         theta -= (alpha/m)*(np.dot(x.T, (np.dot(x, theta) - y)))
18.         J_history[i] = computeCost(x, y, theta)
19.     return theta, J_history
20.
21. def computeCost(x, y, theta):
22.     m = len(y)
23.     return sum(np.square(np.dot(x, theta)-y))/(2*m)
24.
25. def main():
26.     # 读入数据
27.     mydata = pd.read_csv("ex1data1.txt", header=None)
28.     x = np.array(mydata[0])
29.     y = np.array(mydata[1])
30.     m = len(y)
31.     # 画出数据图像
32.     plotData(x,y)
33.     theta = np.zeros((2, 1))
34.     x = np.c_[np.ones((m, 1)), x]
35.     y = y.reshape((len(y), 1))
36.     J = computeCost(x, y, theta)
37.     iterations = 1500
38.     alpha = 0.01
39.     theta, J_history = gradientDescent(x, y, theta, alpha, iterations)
40.     print theta
```

```

41.     plt.plot(J_history, 'r')
42.     plt.show()
43.
44.
45.
46. if __name__ == "__main__":
47.     main()

```

多变量实现：

```

1. # coding: utf-8
2. import pandas as pd
3. import numpy as np
4. import matplotlib.pyplot as plt
5.
6. def plotData(x, y):
7.     plt.title('mytest')
8.     plt.xlabel('Population of City in 10,000s')
9.     plt.ylabel('Profit in $10,000s')
10.    plt.plot(x, y, 'rx')
11.    plt.show()
12.
13. def gradientDescent(x, y, theta, alpha, num_iters):
14.     m = len(y)
15.     # 用来存储历史损失
16.     J_history = np.zeros((num_iters, 2))
17.     for i in range(num_iters):
18.         # 梯度下降法求解
19.         theta -
20.         = (alpha / m) * (np.dot(x.T, (np.dot(x, theta) - y)))
21.         J_history[i] = computeCost(x, y, theta)
22.     return theta, J_history
23.
24. def computeCost(x, y, theta):
25.     m = len(y)
26.     # 计算损失均方误差
27.     return sum(np.square(np.dot(x, theta) - y)) / (2 * m)
28.
29. def featureNormalize(x):
30.     # 求均值与方差，注意 axis 的设置，控制求均值的方向（维度），如果不设置则为整体的均值/方差

```

```
30.     mu = x.mean(axis=0)
31.     sigma = x.std(axis=0)
32.     return (x - mu)/sigma
33.
34.
35. def main():
36.     # 读入数据
37.     mydata = pd.read_csv("ex1data2.txt", header=None)
38.     # 选取前两列 (Dataframe 如何切片)
39.     x = np.array(mydata.ix[:, 0:1])
40.     y = np.array(mydata[2])
41.     m = len(y)
42.     # 画出数据图像
43.     # plotData(x, y)
44.     theta = np.zeros((3, 1))
45.     x = featureNormalize(x)
46.     # numpy 添加一列
47.     x = np.c_[np.ones((m, 1)), x]
48.     # 变成列向量
49.     y = y.reshape((len(y), 1))
50.     J = computeCost(x, y, theta)
51.     iterations = 400
52.     alpha = 0.01
53.     theta, J_history = gradientDescent(x, y, theta, alpha, iterations)
54.     print theta
55.     plt.plot(J_history, 'r')
56.     plt.show()
57.
58.
59. if __name__ == "__main__":
60.     main()
```