机器学习与数据挖掘 Homework1

21307035 邓栩瀛

用梯度下降的方法找出下列函数的最小值$f(x,y)=4x^2-4xy+2y^2$

初始点是$(x_0,y_0)=(2,3)$

a.思考：步长要怎么设计？（用最贪心？太大或太小会怎么样？）不用梯度，用其他方向会怎么样？

b.对以上问题完成回归代码：

——实现用梯度下降来找出回归的解

——尝试一下不同的学习率与迭代次数

思考

计算梯度：计算函数 $f(x,y)$ 对于 x 和 y 的偏导数，得到梯度向量 $\nabla f(x,y)=\begin{bmatrix}\frac{\partial f}{\partial x}\\ \frac{\partial f}{\partial y}\end{bmatrix}=\begin{bmatrix}8x-4y\\ -4x+4y\end{bmatrix}$

更新规则：

$$\begin{aligned} x_{n+1} &= x_n - learning\_rate*(8x_n-4y_n) \\ y_{n+1} &= y_n - learning\_rate*(-4x_n+4y_n) \end{aligned} \tag{1}$$

根据收敛条件，达到一定的迭代次数或梯度变化很小的时候，判断是否停止迭代

步长（学习率）的选择：如果步长太小，梯度下降的收敛速度可能会很慢，需要更多的迭代次数才能达到最小值。如果步长太大，可能导致在最小值附近发生振荡或无法收敛的情况，需要具体情况具体分析

选择其他方向，可能无法高效地找到函数的最小值，梯度下降可以推动搜索方向朝着最优的方向进行，从而更快地到达最小值。

代码实现

1、使用`numpy`库

```
import numpy as np
```

2、定义函数`f(x,y)`

```
def f(x, y):
    return 4 * x ** 2 - 4 * x * y + 2 * y ** 2
```

3、定义梯度函数`grad_func`

```
def grad_func(point):
    x, y = point
    grad_x = 8 * x - 4 * y # df/dx
    grad_y = -4 * x + 4 * y # df/dy
    return np.array([grad_x, grad_y])
```

4、定义梯度下降函数`gradient_descent`

```python
def gradient_descent(starting_point=None, iterations=1000, learning_rate=0.01):
    if starting_point is None: # 如果没有初始点，则随机生成
        point = np.random.uniform(-10, 10, size=2)
    else:
        point = starting_point
    trajectory = [point] # 存储迭代后的点
    for i in range(iterations): # 循环迭代
        grad = grad_f(point) # 计算当前点的梯度
        point = point - learning_rate * grad # 根据梯度和学习率更新当前的点
        trajectory.append(point) # 将更新后的点添加到列表中
    return np.array(trajectory) # 将列表返回
```

5、计算并输出结果

```python
traj = gradient_descent(iterations=1000, starting_point=[2, 3], learning_rate=0.01)

print("Minimum point: ", traj[-1])
print("Minimum value:", f(traj[-1][0], traj[-1][1]))
```

6、运行结果（默认参数，`iterations=1000`，`learning_rate=0.01`）

```
Minimum point:  [3.89794946e-07 6.30701471e-07]
Minimum value: 4.199521074915956e-13
```

学习率与迭代次数的调整

学习率调整

迭代次数不变，为`iterations=1000`

`learning_rate=0.2`

```
Minimum point:  [ 1.62340402e+38 -1.00331886e+38]
Minimum value: 1.9070227348321904e+77
```

`learning_rate=0.1`

```
Minimum point:  [1.86377865e-72 3.01565720e-72]
Minimum value: 9.600990115659 59e-144
```

`learning_rate=0.01`

```
Minimum point:  [3.89794946e-07 6.30701471e-07]
Minimum value: 4.199521074915956e-13
```

`learning_rate=0.001`

```
Minimum point:  [0.41061099 0.66437621]
Minimum value: 0.4659963445337194
```

如果学习率太小，在每次迭代中，点的更新步长很小，算法将收敛缓慢，可能需要更多的迭代次数才能达到最小值点。

如果学习率太大，在每次迭代中，点的更新步长太大，导致可能会跳过最小值点，因而算法可能无法收敛或会在最小值周围震荡，导致无法达到最优解。

迭代次数调整

学习率不变，为`learning_rate=0.01`

`iterations=100`

```
Minimum point:  [0.40627072 0.65735612]
Minimum value: 0.45619954921099315
```

`iterations=1000`

```
Minimum point:  [3.89794946e-07 6.30701471e-07]
Minimum value: 4.199521074915956e-13
```

`iterations=10000`

```
Minimum point:  [2.57668168e-67 4.16915854e-67]
Minimum value: 1.8350542081000036e-133
```

`iterations=24247`

```
Iteration is: 24247
Minimum point:  [1.37971573e-162 2.23242695e-162]
Minimum value: 0.0
```

`iterations=100000`

```
Minimum point:  [1.2e-322 1.8e-322]
Minimum value: 0.0
```

如果迭代次数太少，算法可能无法收敛到最小值点。在每次迭代中，点的位置更新不足以达到最优解，但由于迭代次数限制，导致算法提前终止，未能找到全局最小值。

如果迭代次数太多，算法可能会过度拟合或浪费计算资源。在每次迭代中，点的位置更新可能会越过最小值点并继续朝着梯度的反方向移动，导致算法在最小值附近来回震荡。

最终结果

`learning_rate=0.01`,`iterations=24247`时，达到最小值点

```
Minimum point:  [1.37971573e-162 2.23242695e-162]
Minimum value: 0.0
```