1 梯度下降法

2018年3月12日 16:49

目录

1. 梯度

在微积分中，对多元函数的参数求偏导数，把求得的各个参数偏导数以向量的形式写出来，就是梯度。比如函数$f(x,y)$，分别对x，y求偏导数，求得的梯度向量就是$(\partial f/\partial x, \partial f/\partial y)^T$，简称grad$f(x,y)$或者$\nabla f(x,y)$。对于在点$(x_0,y_0)$的具体梯度向量就是$(\partial f/\partial x, \partial f/\partial y)^T$，或者$\nabla f(x_0,y_0)$，如果是3个参数的向量梯度，就是$(\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)^T$，以此类推。

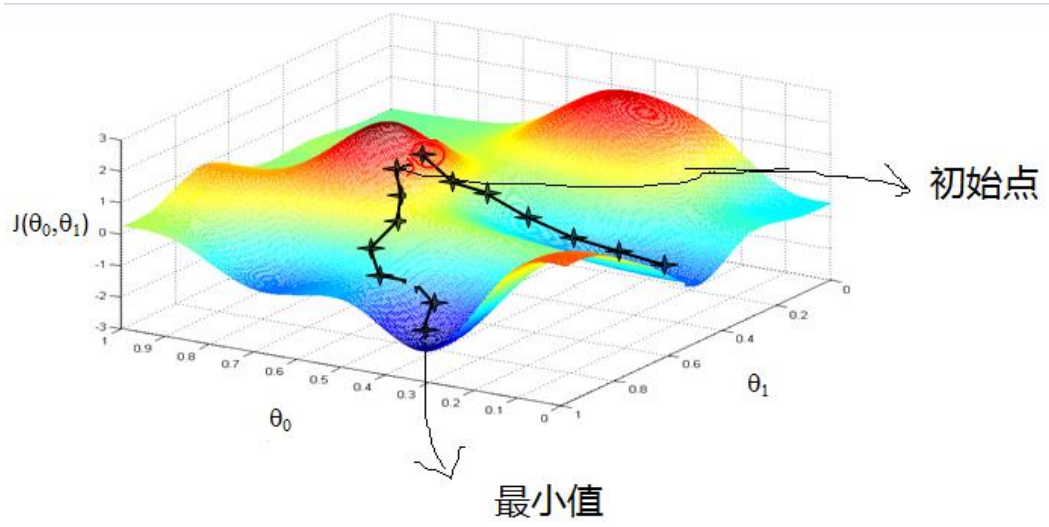
梯度的意义就是函数变化增加最快的方向。具体来说，对于函数$f(x,y)$，在点(x_0,y_0)沿梯度向量的方向就是$(\partial f/\partial x, \partial f/\partial y)^T$的方向是$f(x,y)$增加最快的方向，更加容易找到函数的最大值。反过来，沿着梯度向量相反的方向，也就是$-(\partial f/\partial x, \partial f/\partial y)^T$的方向，$f(x,y)$减小最快，更容易找到函数的最小值。

2. 梯度下降算法详解

2.1 梯度下降的直观解释

比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处。

从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。



2.2 梯度下降的相关概念

1、步长（Learning rate）：步长决定了在梯度下降迭代的过程中，每一步沿梯度负方向前进的长度。用上面下山的例子，步长就是在当前这一步所在位置沿着最陡峭最易下山的位置走的那一步的长度。

2、特征（feature）：指的是样本中输入部分，比如2个单特征的样本$(x^{(0)},y^{(0)})$，$(x^{(1)},y^{(1)})$，则第一个样本特征为$x^{(0)}$，第一个样本输出为$y^{(0)}$。

3、假设函数（hypothesis function）：在监督学习中，为了拟合输入样本，而使用的假设函数，记为$h_\theta(x)$。比如对于单个特征的m个样本$(x^{(i)},y^{(i)})(i=1,2,\cdots,m)$，可以采用拟合函数如下：$h_\theta(x)=\theta_0+\theta_1 x$。

4、损失函数（loss function）：为了评估模型拟合的好坏，通常用损失函数来度量拟合的程度。损失函数极小化，意味着拟合程度最好，对应的模型参数即为最优参数。在线性回归中，损失函数通常为样本输出和假设函数的差取平方。比如对于m个样本$(x^{(i)},y^{(i)})(i=1,2,\cdots,m)$，采用线性回归，损失函数为：

$$J(\theta_0,\theta_1)=\sum_{i=1}^{m}\left(h_\theta(x_i)-y_i\right)^2$$

其中，x_i表示第i个样本特征，y_i表示第i个样本对应的输出，$h_\theta(x_i)$为假函数。

2.3 梯度下降的详细算法

列出样本总体和输出表格：

特征 样本数	x_0	x_1	.....	x_n	真实 输出
0	$x_0^{(0)}=1$	$x_1^{(0)}=0$	.....	$x_n^{(0)}=0$	$y_0=\theta_0$
1	$x_0^{(1)}=1$	$x_1^{(1)}$	.....	$x_n^{(1)}$	y_1
2	$x_0^{(2)}=1$	$x_1^{(2)}$	.....	$x_n^{(2)}$	y_2
.....	.....	.....	.....	.....	.....
i	$x_0^{(i)}=1$	$x_1^{(i)}$	.....	$x_n^{(i)}$	y_i
.....	.....	.....	.....	.....	.....
m	$x_0^{(m)}=1$	$x_1^{(m)}$	.....	$x_n^{(m)}$	y_m

假设函数表示为$h_\theta(x_1,x_2,\cdots,x_n)=\theta_0+\theta_1x_1+\cdots+\theta_nx_n$，其中$\theta_i(i=0,1,2,\cdots,n)$为模型参数，$x_i=1,2,\cdots,n$为样本的n个特征。简化表示，增加一个特征$x_0=1$，这样：

$h_\theta(x_0,x_1,\cdots,x_n)=\sum_{i=0}^{n}\theta_ix_i$。

损失函数即为：

$$J(\theta_0,\theta_1,\cdots,\theta_n)=\frac{1}{2m}\sum_{j=0}^{m}\left(h_\theta(x_0^{(j)},x_1^{(j)},\cdots,x_n^{(j)})-y_j\right)^2$$

在没有任何先验知识的时候，可将θ都初始化为0，将步长（学习率）初始化为1。在调优的时候再优化。

下面求损失函数的梯度，对θ_i的偏导数计算如下：

$$\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1,\cdots,\theta_n)=\frac{1}{m}\sum_{j=0}^{m}\left(h_\theta(x_0^{(j)},x_1^{(j)},\cdots,x_n^{(j)})-y_j\right)x_i^{(j)}$$

确定是否所有的θ_i梯度下降距离都小于设定的终止距离ε，如果小于ε则算法终止，否则就不停的更新θ_i：

$$\theta_i=\theta_i-\alpha\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1,\cdots,\theta_n)$$

$$=\theta_i-\alpha\frac{1}{m}\sum_{j=0}^{m}\left(h_\theta(x_0^{(j)},x_1^{(j)},\cdots,x_n^{(j)})-y_j\right)x_i^{(j)}$$

这里1/m只是为了方便理解，其为一常数。

2.4 梯度下降法的矩阵描述

首先模型函数可以写为：

$$h_\theta(x)=X_{(m+1)\times(n+1)}\boldsymbol{\theta}_{(n+1)\times1}$$

其中$X_{(m+1)\times(n+1)}$就是上面列出的样本矩阵，之所以加一是为了包含x_0和θ_0。

损失函数表达式为：

$$J(\boldsymbol{\theta})=\frac{1}{2}(X\boldsymbol{\theta}-\boldsymbol{Y})^T(X\boldsymbol{\theta}-\boldsymbol{Y})$$

对损失函数求偏导，为：

$$\frac{\partial}{\partial\theta}J(\theta)|_{(n+1)\times1}=X^T(X\boldsymbol{\theta}-\boldsymbol{Y})$$

更新如下：

$$\boldsymbol{\theta}=\boldsymbol{\theta}-\alpha X^T(X\boldsymbol{\theta}-\boldsymbol{Y})$$

注：矩阵的求导法则如下

公式1：$\frac{\partial}{\partial X}(XX^T)=2X$

公式2：$\frac{\partial}{\partial\theta}(X\theta)=X^T$

- 算法的步长选择取决于样本，要多尝试一些值，看运行效果，如果损失函数在变小，说明取值有效。步长太大，可能会错过最优解，步长太小，迭代速度太慢，很长时间都不能迭代结束。
- 算法的初始值选择不同，也会导致获得的最小值不同，因此梯度下降获得的是局部最小值。当然如果损失函数是凸函数，那么一定能得到最优解。由于局部最优解的风险，需要多次用不同的初始值运行算法。
- 归一化，由于样本不同特征值的取值范围不一样，可能导致迭代很慢，为了减小特征取值的影响，可以对特征数据归一化。

3. 梯度下降法家族（BGD、SGD、MBGD）

3.1 批量梯度下降法（BGD）

批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新，前面介绍的方法即是批量梯度下降法，有m个样本，就用所有m个样本的梯度数据。

3.2 随机梯度下降法（SGD）

随机梯度下降法，其实和批量梯度下降法原理类似，区别在于求梯度时没有用所有的m个样本的数据，而是仅仅选取一个样本j来求梯度。对应的更新公式是：

$$\theta_i = \theta_i - \alpha\left(h_\theta(x_0^{(j)}, x_1^{(j)}, \cdots, x_n^{(j)}) - y_j\right)x_i^{(j)}$$

随机梯度下降法，和批量梯度下降法是两个极端，一个采用所有数据来梯度下降，一个用一个样本来梯度下降。自然各自的优缺点都非常突出。对于训练速度来说，随机梯度下降法由于每次仅仅采用一个样本来迭代，训练速度很快，而批量梯度下降法在样本量很大的时候，训练速度不能让人满意。对于准确度来说，随机梯度下降法用于仅仅用一个样本决定梯度方向，导致解很有可能不是最优。对于收敛速度来说，由于随机梯度下降法一次迭代一个样本，导致迭代方向变化很大，不能很快的收敛到局部最优解。

3.3 小批量梯度下降法（MBGD）

小批量梯度下降法是批量梯度下降法和随机梯度下降法的折衷，也就是对于m个样本，我们采用x个样子来迭代，1<x<m。一般可以取x=10，当然根据样本的数据，可以调整这个x的值。对应的更新公式是：

$$\theta_i = \theta_i - \alpha\sum_{j=t}^{t+x-1}\left(h_\theta(x_0^{(j)}, x_1^{(j)}, \cdots, x_n^{(j)}) - y_j\right)x_i^{(j)}$$

4. 梯度下降法和其它无约束优化算法的比较

在机器学习中的无约束优化算法，除了梯度下降以外，还有前面提到的最小二乘法，此外还有牛顿法和拟牛顿法。

梯度下降法和最小二乘法相比，梯度下降法需要选择步长，而最小二乘法不需要。梯度下降法是迭代求解，最小二乘法是计算解析解。如果样本量不算很大，且存在解析解，最小二乘法比起梯度下降法要有优势，计算速度很快。但是如果样本量很大，用最小二乘法由于需要求一个超级大的逆矩阵，这时就很难或者很慢才能求解解析解了，使用迭代的梯度下降法比较有优势。

梯度下降法和牛顿法/拟牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法/拟牛顿法是用二阶的海森矩阵的逆矩阵或伪逆矩阵求解。相对而言，使用牛顿法/拟牛顿法收敛更快。但是每次迭代的时间比梯度下降法长。

5. 梯度下降法程序

构造一个X向量和一个Y向量，使其方程为：Y=10+5X，这是一个最简单的线性函数，尝试用梯度下降法求出系数。
程序如下：

```
"""
    gradient descent method

"""

import numpy as np


def main():
```

x = np.random.rand(100)	随机生成100个0~1之间的数
y = 10 + 5 * x	构造一个线性函数

```
    print(len(x), len(y))
```

epsilon = 0.01 # iterative threshold	最终迭代的损失函数精度

```
    cnt = 0
```

alpha = 0.001	梯度每走一步的步长，在递归中结合了1/len(x)

```
    theta0 = 0
    theta1 = 0

    error1 = 0
    error0 = 0

    while True:
        sum0 = 0
        sum1 = 0
        cnt = cnt + 1
        for i in range(0, len(x)):
            diff = y[i] - (theta0 + theta1 * x[i])
```

sum0 = sum0 + diff	系数theta0的梯度，此时x0=1
sum1 = sum1 + diff*x[i]	系数theta1的梯度，此时x1=np.random.rand(100)

theta0 = theta0 + alpha * sum0/len(x)	系数theta0走一步
theta1 = theta1 + alpha * sum1/len(x)	系数theta1走一步

```
        error1 = 0
```

```
        for i in range(0, len(x)):
            error1 = error1 + (y[i] - (theta0 + theta1 * x[i]))**2      损失函数

            if abs(error1) < epsilon:      判断递归的条件，如果要找到全局最优点，可让损失函数减小到一个很小的值。
                print('error1-0:', abs(error1 - error0))
                break

            else:
                print('error1:', error1)

            print('theta0:', theta0, 'theta1:', theta1)


if __name__ == '__main__':
    main()
```

结果：

theta0: 9.98084025904 theta1: 5.03490615569

error1-0: 0.00999883195471

上面的程序是让损失函数减小小一个很小的值，也就是找到全局最优点。因为本例很简单，可以找到该点，但是有很多时候无法寻找全局最优点。此时，可以把判断部分的代码改写如下：

```
            if abs(error1 - error0) < epsilon:      利用最近两次的递归结果，判断梯度是否已经达到了局部最优点
                print('error1-error0:', abs(error1 - error0))
                break

            else:
                error0 = error1      损失赋值，以便求出局部最优点
                print('error1:', error1)
```

结果：

theta0: 9.72286645217 theta1: 5.14547998578

error1-error0: 0.00998931157493

可见，同样在精度0.01的情况下，局部最优得到的结果是不如全局最优的。

因为这是一个一元线性回归问题，所以可直接用最小二乘法得到系数，根据公式：

$$\begin{cases} \hat{\beta}_1 = \dfrac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \\ \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \end{cases}$$

直接求出系数。代码如下：

```
"""
```

```
    least square method

"""
import numpy as np


def main():

    x = np.random.rand(100)
    y = 10 + 5 * x
    sum1 = 0
    sum2_x = 0
    sum2_y = 0
    sum3 = 0

    for i in range(0, len(x)):
        sum1 = sum1 + x[i]*y[i]
        sum2_x = sum2_x + x[i]
        sum2_y = sum2_y + y[i]
        sum3 = sum3 + x[i]*x[i]

    a = (len(x)*sum1) - (sum2_x*sum2_y)
    b = (len(x)*sum3) - (sum2_x*sum2_x)

    theta1 = a/b
    theta0 = (sum2_y/len(x))-theta1*(sum2_x/len(x))

    print('y = {} + {}*x'.format(theta0, theta1))


if __name__ == '__main__':
    main()
```

结果：

y = 9.999999999999984 + 5.000000000000029*x