
(http://lib.csdn.net/base/python)

Python (http://lib.csdn.net/base/python) - 数据科学 (http://lib.csdn.net/python/node/88) - 数据分析 (http://lib.csdn.net/python/knowledge/183)

315 2

梯度下降法(BGD,SGD,MSGD)python+numpy具体实现

作者：pengjian444 (http://my.csdn.net/pengjian444)

梯度下降是一阶迭代优化算法。为了使用梯度下降找到函数的局部最小值，一个步骤与当前位置的函数的梯度（或近似梯度）的负值成正比。如果相反，一个步骤与梯度的正数成比例，则接近该函数的局部最大值;该程序随后被称为梯度上升。梯度下降也被称为最陡峭的下降，或最快下降的方法。（from wikipad)

首先，大家要明白它的本质：这是一个优化算法！！！它是可以用来解决很多问题的，一般学习机器学习的朋友都会在线性回归的遇到这个名词，但是要声明的是，它和最小二乘法类似，是用于求解线性回归问题的一种方法。同时它的功能又不仅于此，它在线性回归中的意义在于通过寻找梯度最大的方向下降（或上升）来找到损失函数最小时候对应的参数值。

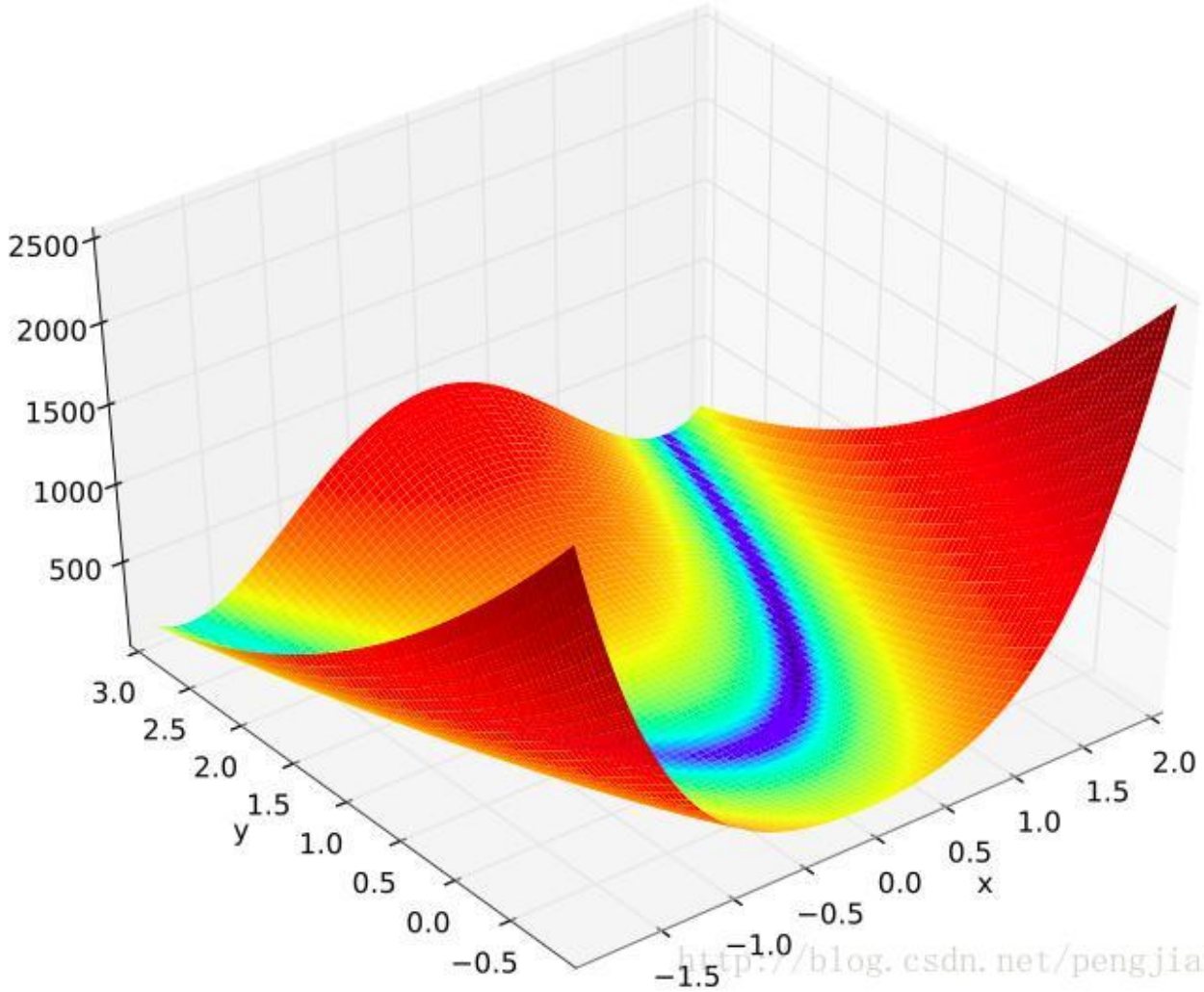
好了，绕来绕去的就拿线性回归的例子来和大家讲讲吧。

梯度下降方法

本质是每次迭代的时候都沿着梯度最大的地方更新参数。现在假设有函数（Rosenbrock函数：是一个用来测试最优化算法性能的非凸函数，由Howard Harry Rosenbrock在1960年提出[1]。也称为Rosenbrock山谷或Rosenbrock香蕉函数，也简称为香蕉函数）如下定义：

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

很明显，其最小最为$f(1,1) = 0$，其三维图片如下：



函数f分别对 x，y求导得到

$$\frac{\partial f(x,y)}{\partial x} = -2(1-x) - 2*100(y-x^2)*2x$$

$$\frac{\partial \mathrm{f}(\mathrm{x},\mathrm{y})}{\partial \mathrm{y}} = 2 * 100(\mathrm{y} - \mathrm{x}^2)\Big|$$

在实现的过程中可以给出x, y初始值(例如设置为 0, 0) 然后计算函数在这个点的梯度，并按照梯度方向更新x, y的值。

这里给出通过梯度下降法计算上述函数的最小值对应的x 和 y

```
import numpy as np


def cal_rosenbrock(x1, x2):
    """
    计算rosenbrock函数的值
    :param x1:
    :param x2:
    :return:
    """
    return (1 - x1) ** 2 + 100 * (x2 - x1 ** 2) ** 2


def cal_rosenbrock_prax(x1, x2):
    """
    对x1求偏导
    """
    return -2 + 2 * x1 - 400 * (x2 - x1 ** 2) * x1

def cal_rosenbrock_pray(x1, x2):
    """
    对x2求偏导
    """
    return 200 * (x2 - x1 ** 2)

def for_rosenbrock_func(max_iter_count=100000, step_size=0.001):
    pre_x = np.zeros((2,), dtype=np.float32)
    loss = 10
    iter_count = 0
    while loss > 0.001 and iter_count < max_iter_count:
        error = np.zeros((2,), dtype=np.float32)
        error[0] = cal_rosenbrock_prax(pre_x[0], pre_x[1])
        error[1] = cal_rosenbrock_pray(pre_x[0], pre_x[1])
```

```
34
35          for j in range(2):
36              pre_x[j] -= step_size * error[j]
37
38          loss = cal_rosenbrock(pre_x[0], pre_x[1])  # 最小值为0
39
40          print("iter_count: ", iter_count, "the loss:", loss)
41          iter_count += 1
42      return pre_x
43
44  if __name__ == '__main__':
45      w = for_rosenbrock_func()
46      print(w)
```

如果大家想运行这个算法，建议使用默认的参数，效果还不错。不要把step_size设置过大，会出问题的（可能是实现过程有问题，请指正）。

线性回归问题

这里关于回归的前导介绍我建议大家取看周志华老师的西瓜书，介绍得通透明亮，但是周老师对线性回归问题给出的解决方法是通过最小二乘法来做的，而我们在这里要用梯度下降。

这里给出一般的定义吧~

一般的线性回归方程如下：

$$y = \theta_1 * x_1 + \theta_2 * x_2 + \cdots + \theta_n * x_n + b$$

转换为：

$$y = \theta_1 * x_1 + \theta_2 * x_2 + \cdots + \theta_n * x_n + \theta_0 * b$$

这里$\theta_0 = 1$转换为向量的形式$y = \theta^T * x$，θ，x，均为为行向量。

现在需要定义损函数，用于判断最后得到的预测参数的预测效果。常用的损失函数是均方误差：

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

i是维度索引 j是样本索引，接下来对θ求导得到

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{j=1}^{m}(h(\theta)^i - y^i)x_j^i$$

更新公式为：

$$\theta_i = \theta_i - \alpha\frac{1}{m}\sum_{j=1}^{m}(h(\theta)^i - y^i)x_j^i$$

α就是学习的步长。

BGM（批量梯度下降法）

```
import numpy as np

def gen_line_data(sample_num=100):
    """
    y = 3*x1 + 4*x2
    :return:
    """
    x1 = np.linspace(0, 9, sample_num)
    x2 = np.linspace(4, 13, sample_num)
    x = np.concatenate(([x1], [x2]), axis=0).T
    y = np.dot(x, np.array([3, 4]).T)  # y 列向量
    return x, y

def bgd(samples, y, step_size=0.01, max_iter_count=10000):
    sample_num, dim = samples.shape
    y = y.flatten()
    w = np.ones((dim,), dtype=np.float32)
    loss = 10
    iter_count = 0
    while loss > 0.001 and iter_count < max_iter_count:
        loss = 0
        error = np.zeros((dim,), dtype=np.float32)
        for i in range(sample_num):
            predict_y = np.dot(w.T, samples[i])
            for j in range(dim):
                error[j] += (y[i] - predict_y) * samples[i][j]

        for j in range(dim):
            w[j] += step_size * error[j] / sample_num

        for i in range(sample_num):
            predict_y = np.dot(w.T, samples[i])
            error = (1 / (sample_num * dim)) * np.power((predict_y - y[i]), 2)
```

```
            loss += error

        print("iter_count: ", iter_count, "the loss:", loss)
        iter_count += 1
    return w

if __name__ == '__main__':
    samples, y = gen_line_data()
    w = bgd(samples, y)
    print(w)  # 会很接近[3, 4]
```

SGB（随机梯度下降法）

```
import numpy as np

def gen_line_data(sample_num=100):
    """
    y = 3*x1 + 4*x2
    :return:
    """
    x1 = np.linspace(0, 9, sample_num)
    x2 = np.linspace(4, 13, sample_num)
    x = np.concatenate(([x1], [x2]), axis=0).T
    y = np.dot(x, np.array([3, 4]).T)  # y 列向量
    return x, y

def sgd(samples, y, step_size=0.01, max_iter_count=10000):
    """
    随机梯度下降法
    :param samples: 样本
    :param y: 结果value
    :param step_size: 每一接迭代的步长
    :param max_iter_count: 最大的迭代次数
    :param batch_size: 随机选取的相对于总样本的大小
    :return:
    """
    sample_num, dim = samples.shape
    y = y.flatten()
    w = np.ones((dim,), dtype=np.float32)
    loss = 10
    iter_count = 0
    while loss > 0.001 and iter_count < max_iter_count:
        loss = 0
        error = np.zeros((dim,), dtype=np.float32)
        for i in range(sample_num):
            predict_y = np.dot(w.T, samples[i])
```

```
            for j in range(dim):
                error[j] += (y[i] - predict_y) * samples[i][j]
                w[j] += step_size * error[j] / sample_num

        # for j in range(dim):
        #     w[j] += step_size * error[j] / sample_num

        for i in range(sample_num):
            predict_y = np.dot(w.T, samples[i])
            error = (1 / (sample_num * dim)) * np.power((predict_y - y[i]), 2)
            loss += error

        print("iter_count: ", iter_count, "the loss:", loss)
        iter_count += 1
    return w

if __name__ == '__main__':
    samples, y = gen_line_data()
    w = sgd(samples, y)
    print(w)  # 会很接近[3, 4]
```

MBGB(小批量梯度下降法)

```
import numpy as np
import random

def gen_line_data(sample_num=100):
    """
    y = 3*x1 + 4*x2
    :return:
    """
    x1 = np.linspace(0, 9, sample_num)
    x2 = np.linspace(4, 13, sample_num)
    x = np.concatenate(([x1], [x2]), axis=0).T
    y = np.dot(x, np.array([3, 4]).T)  # y 列向量
    return x, y

def mbgd(samples, y, step_size=0.01, max_iter_count=10000, batch_size=0.2):
    """
    MBGD（Mini-batch gradient descent）小批量梯度下降：每次迭代使用b组样本
    :param samples:
    :param y:
    :param step_size:
    :param max_iter_count:
    :param batch_size:
    :return:
    """
    sample_num, dim = samples.shape
    y = y.flatten()
    w = np.ones((dim,), dtype=np.float32)
    # batch_size = np.ceil(sample_num * batch_size)
    loss = 10
    iter_count = 0
    while loss > 0.001 and iter_count < max_iter_count:
        loss = 0
        error = np.zeros((dim,), dtype=np.float32)
```

```
34
35          # batch_samples, batch_y = select_random_samples(samples, y,
36          # batch_size)
37
38          index = random.sample(range(sample_num),
39                                int(np.ceil(sample_num * batch_size)))
40          batch_samples = samples[index]
41          batch_y = y[index]
42
43          for i in range(len(batch_samples)):
44              predict_y = np.dot(w.T, batch_samples[i])
45              for j in range(dim):
46                  error[j] += (batch_y[i] - predict_y) * batch_samples[i][j]
47
48          for j in range(dim):
49              w[j] += step_size * error[j] / sample_num
50
51          for i in range(sample_num):
52              predict_y = np.dot(w.T, samples[i])
53              error = (1 / (sample_num * dim)) * np.power((predict_y - y[i]), 2)
54              loss += error
55
56          print("iter_count: ", iter_count, "the loss:", loss)
57          iter_count += 1
58      return w
59
60  if __name__ == '__main__':
61      samples, y = gen_line_data()
62      w = mbgd(samples, y)
63      print(w)  # 会很接近[3, 4]
```

查看原文>> (http://blog.csdn.net/pengjian444/article/details/71075544)

0

看过本文的人也看了：

- Python知识结构图 (http://lib.csdn.net/base/python/structure)
- Python聚类工具scipy cluster (http://lib.csdn.net/article/python/47326)
- python networkx学习 (http://lib.csdn.net/article/python/64704)
- windows环境下python2.7、pycharm、numpy_m... (http://lib.csdn.net/article/python/64679)
- python数据分析pandas包入门学习（二）... (http://lib.csdn.net/article/python/66549)
- 量化分析师的Python日记【第6天：数据... (http://lib.csdn.net/article/python/44487)

发表评论

输入评论内容

发表

2个评论


(http://my.csdn.net/pengjian444)

pengjian444 (http://my.csdn.net/pengjian444)

[quote=sunhuaqiang1]CSDN博友你好，我是孙华强，现将此篇博文收录进“Python知识库”。CSDN现在在做CSDN博客...[/quote]已投，谢谢收录 ~

2017-05-23 11:38:06 回复


(http://my.csdn.net/sunhuaqiang1)

sunhuaqiang1 (http://my.csdn.net/sunhuaqiang1)

CSDN博友你好，我是孙华强，现将此篇博文收录进“Python知识库”。CSDN现在在做CSDN博客专栏评选活动，若你感觉我写的博文内容还可以，对你有帮助，请投上你宝贵的一票，愿与君共勉，谢谢。技术问题，可与君探讨~请投票给《Python3进阶》和《JVM进阶》两大专栏，谢谢~（投票地址：https://wj.qq.com/s/1375474/9b8e）

2017-05-23 09:08:59 回复

公司简介 (http://www.csdn.net/company/about.html) | 招贤纳士 (http://www.csdn.net/company/recruit.html) | 广告服务 (http://www.csdn.net/company/marketing.html) |
联系方式 (http://www.csdn.net/company/contact.html) | 版权声明 (http://www.csdn.net/company/statement.html) | 法律顾问 (http://www.csdn.net/company/layer.html) |
问题报告 (mailto:webmaster@csdn.net) | 合作伙伴 (http://www.csdn.net/friendlink.html) | 论坛反馈 (http://bbs.csdn.net/forums/Service)

网站客服 杂志客服 (http://wpa.qq.com/msgrd?v=3&uin=2251809102&site=qq&menu=yes) 微博客服 (http://e.weibo.com/csdnsupport/profile) webmaster@csdn.net (mailto:webmaster@csdn.net)

400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved (http://www.hd315.gov.cn/beian/view.asp?bianhao=010202001032100010)