

机器学习算法实践-SVM中的SMO算法



PytLab酱

On the way to be a hardcore programmer

118 人赞了该文章

前言

前两篇关于SVM的文章分别总结了SVM基本原理和核函数以及软间隔原理，本文我们就针对前面推导出的SVM对偶问题的一种高效的优化方法-序列最小优化算法(Sequential Minimal Optimization, SMO)的原理进行总结并进行相应的Python实现。

坐标上升算法(Coordinate Ascent)

在SMO算法之前，还是需要总结下坐标上升算法，因为SMO算法的思想与坐标上升算法的思想类似。

坐标上升算法每次通过更新多元函数中的一维，经过多次迭代直到收敛来达到优化函数的目的。简单的讲就是不断地选中一个变量做一维最优化直到函数达到局部最优。

假设我们需要求解的问题形式为(类似我们SVM的对偶形式):

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_N)$$

算法过程伪码:

▲ 赞同 118 ▼ 16 条评论 分享 收藏 ...



```

Loop until convergence: {
    For  $i = 1, \dots, m$ , {
         $\alpha_i := \arg \max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$ .
    }
}

```

例子

若我们的优化目标为一个二元函数:

$$\arg \min_{x_1, x_2} f(x_1, x_2) = -x_1^2 - 3x_2^2 + 2x_1x_2 + 6$$

我们先给一个 (x_1, x_2) 的初值然后开始迭代。

1. 先固定 x_1 ，把 f 看做 x_2 的一元函数求最优值，可以简单的进行求导获取解析解:

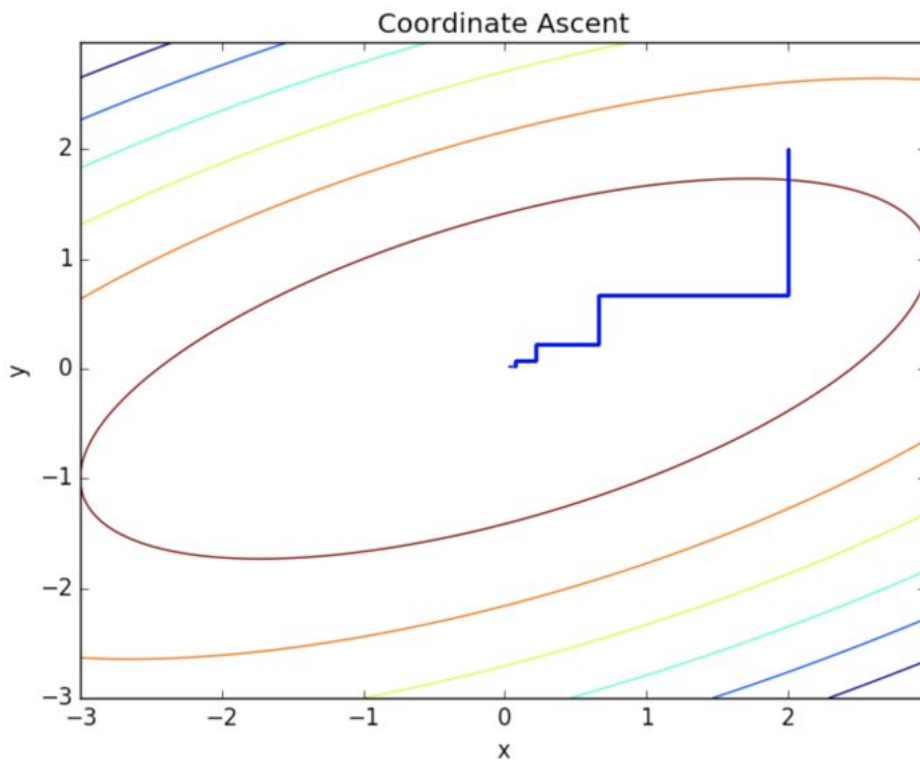
$$\frac{\partial f}{\partial x_1} = -2x_1 + 2x_2 = 0 \rightarrow x_1 = x_2$$

2. 在固定 x_2 下, 把 f 看成 x_1 的一元函数求最优值, 得到 x_1 的解析解:

$$\frac{\partial f}{\partial x_2} = -6x_2 + 2x_1 \rightarrow x_2 = \frac{1}{3}x_1$$

按照上面两个过程不断交替的优化 x_1 和 x_2 ，直到函数收敛。

通过下面的图就可以看出，优化的过程，因为每次只优化一个变量，每次迭代的方向都是沿着坐标轴方向的。



因为每次只是做一维优化，所以每个循环中的优化过程的效率是很高的，但是迭代的次数会比较多。

序列最小优化算法(SMO)

SMO算法介绍

SMO的思想类似坐标上升算法，我们需要优化一系列的 α 的值，我们每次选择尽量少的 α 来优化，不断迭代直到函数收敛到最优值。

来到SVM的对偶问题上，对偶形式：

$$\arg \max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^N \alpha_i y_i = 0$$

其中我们需要对 $(\alpha_1, \alpha_2, \dots, \alpha_N)$ 进行优化，但是这个凸二次优化问题的其他求解算法的复杂度很高，但是Platt提出的SMO算法可以高效的求解上述对偶问题，他把原始问题的求解 N 个参数二次规划问题分解成多个二次规划问题求解，每个子问题只需要求解2各参数，节省了时间成本和内存需求。

与坐标上升算法不同的是，我们在SMO算法中我们每次需要选择一对变量 (α_i, α_j) ，因为在SVM中，我们的 α 并不是完全独立的，而是具有约束的：

$$\sum_{i=1}^N \alpha_i y_i = 0$$

因此一个 α 改变，另一个也要随之变化以满足条件。

SMO算法原理

获得没有修剪的原始解

假设我们选取的两个需要优化的参数为 α_1, α_2 ，剩下的 $\alpha_3, \alpha_4, \dots, \alpha_N$ 则固定，作为常数处理。将SVM优化问题进行展开就可以得到(把与 α_1, α_2 无关的项合并成常数项 C)：

$$W(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2} K_{1,1} y_1^2 \alpha_1^2 - \frac{1}{2} K_{2,2} y_2^2 \alpha_2^2 - K_{1,2} y_1 y_2 \alpha_1 \alpha_2 - y_1 \alpha_1 \sum_{i=3}^N \alpha_i y_i K_{i,1} - y_2 \alpha_2 \sum_{i=3}^N \alpha_i y_i K_{i,2} + C$$

于是就是一个二元函数的优化：

$$\arg \max_{\alpha_1, \alpha_2} W(\alpha_1, \alpha_2)$$

根据约束条件 $\sum_{i=1}^N \alpha_i y_i = 0$ 可以得到 α_1 与 α_2 的关系：

$$\alpha_1 y_1 + \alpha_2 y_2 = - \sum_{i=3}^N \alpha_i y_i = \zeta$$

两边同时乘上 y_1 ，由于 $y_i y_i = 1$ 得到：

$$\alpha_1 = \zeta y_1 - \alpha_2 y_1 y_2$$

令 $v_1 = \sum_{i=3}^N \alpha_i y_i K_{i,1}$ ， $v_2 = \sum_{i=3}^N \alpha_i y_i K_{i,2}$ ， α_1 的表达式代入得到：

$$W(\alpha_2) = -\frac{1}{2} K_{1,1} (\zeta - \alpha_2 y_2)^2 - \frac{1}{2} K_{2,2} \alpha_2^2 - y_2 (\zeta - \alpha_2 y_2) \alpha_2 K_{1,2} -$$

后面我们需要对这个一元函数进行求极值， W 对 α 的一阶导数为0得到:

$$\frac{\partial W(\alpha_2)}{\partial \alpha_2} = -(K_{1,1} + K_{2,2} - 2K_{1,2})\alpha_2 + K_{1,1}\zeta y_2 - K_{1,2}\zeta y_2 + v_1 y_2 - v_2 y_2 - y_1 y_2 + y_2^2 = 0$$

下面我们稍微对上式进行下变形，使得 α_2^{new} 能够用更新前的 α_2^{old} 表示，而不是使用不方便计算的 ζ 。

$$\text{因为SVM对数据点的预测值为: } f(x) = \sum_{i=1}^N \alpha_i y_i K(x_i, x) + b$$

则 v_1 以及 v_2 的值可以表示成:

$$v_1 = \sum_{i=3}^N \alpha_i y_i K_{1,i} = f(x_1) - \alpha_1 y_1 K_{1,1} - \alpha_2 y_2 K_{1,2} - b$$

$$v_2 = \sum_{i=3}^N \alpha_i y_i K_{2,i} = f(x_2) - \alpha_1 y_1 K_{1,2} - \alpha_2 y_2 K_{2,2} - b$$

已知 $\alpha_1 = (\zeta - \alpha_2 y_2) y_2$ ，可得到:

$$v_1 - v_2 = f(x_1) - f(x_2) - K_{1,1}\zeta + K_{1,2}\zeta + (K_{1,1} + K_{2,2} - 2K_{1,2})\alpha_2 y_2$$

将 $v_1 - v_2$ 的表达式代入到 $\frac{\partial W(\alpha_2)}{\partial \alpha_2}$ 中可以得到:

$$\frac{\partial W(\alpha_2)}{\partial \alpha_2} = -(K_{1,1}) + K_{2,2} - 2K_{1,2})\alpha_2^{new} + (K_{1,1}) + K_{2,2} - 2K_{1,2})\alpha_2^{old} + y_2 [y_2 - y_1 + f(x_1) - f(x_2)]$$

我们记 E_i 为SVM预测值与真实值的误差: $E_i = f(x_i) - y_i$

令 $\eta = K_{1,1} + K_{2,2} - 2K_{1,2}$ 得到最终的一阶导数表达式:

$$\frac{\partial W(\alpha_2)}{\partial \alpha_2} = -\eta \alpha_2^{new} + \eta \alpha_2^{old} + y_2 (E_1 - E_2) = 0$$

得到:

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2 (E_1 - E_2)}{\eta}$$

这样我们就得到了通过旧的 α_2 获取新的 α_2 的表达式, α_1^{new} 便可以通过 α_2^{new} 得到。

对原始解进行修剪

上面我们通过对一元函数求极值的方式得到的最优 α_i, α_j 是未考虑约束条件下的最优解，我们便更正我们上部分得到的 α_2^{new} 为 $\alpha_2^{new, unclipped}$ ，即:

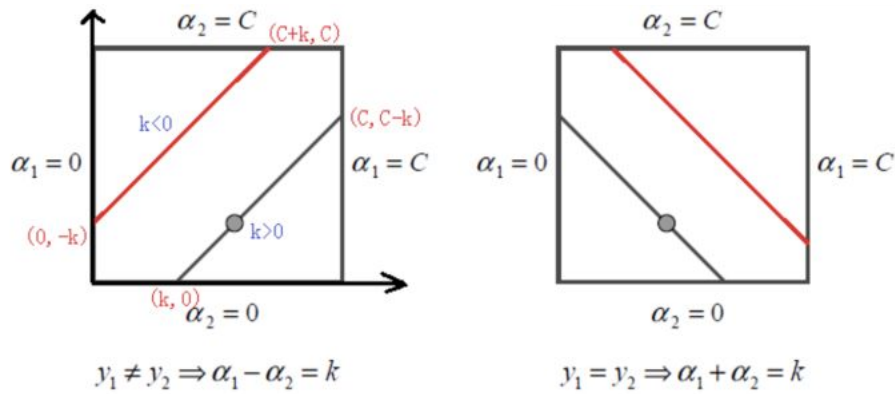
$$\alpha_2^{new, unclipped} = \alpha_2^{old} + \frac{y_2 (E_1 - E_2)}{\eta}$$

但是在SVM中我们的 α_i 是有约束的，即:

$$\alpha_1 y_1 + \alpha_2 y_2 = - \sum_{i=3}^N \alpha_i y_i = \zeta$$

$$0 \leq \alpha_i \leq C$$

此约束为方形约束(Bosk constraint), 在二维平面中我们可以看到这是个限制在方形区域中的直线 (见下图)。



(如左图) 当 $y_1 \neq y_2$ 时, 线性限制条件可以写成: $\alpha_1 - \alpha_2 = k$, 根据 k 的正负可以得到不同的上下界, 因此统一表示成:

- 下界: $L = \max(0, \alpha_2^{old} - \alpha_1^{old})$
- 上界: $H = \min(C, C + \alpha_2^{old} - \alpha_1^{old})$

(如右图) 当 $y_1 = y_2$ 时, 限制条件可写成: $\alpha_1 + \alpha_2 = k$, 上下界表示成:

- 下界: $L = \max(0, \alpha_1^{old} + \alpha_2^{old} - C)$
- 上界: $H = \min(C, \alpha_2^{old} + \alpha_1^{old})$

根据得到的上下界, 我们可以得到修剪后的 α_2^{new} :

$$\alpha_2^{new} = \begin{cases} H & \alpha_2^{new, unclipped} > H \\ \alpha_2^{new, unclipped} & L \leq \alpha_2^{new, unclipped} \leq H \\ L & \alpha_2^{new, unclipped} < L \end{cases}$$

得到了 α_2^{new} 我们便可以根据 $\alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \alpha_1^{new} y_1 + \alpha_2^{new} y_2$ 得到 α_1^{new} :

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new})$$

OK, 这样我们就知道如何将选取的一对 α_i, α_j 进行优化更新了。

更新阈值b

当我们更新了一对 α_i, α_j 之后都需要重新计算阈值 b , 因为 b 关系到我们 $f(x)$ 的计算, 关系到下次优化的时候误差 E_i 的计算。

为了使得被优化的样本都满足KKT条件,

当 α_1^{new} 不在边界, 即 $0 < \alpha_1^{new} < C$, 根据KKT条件可知相应的数据点为支持向量, 满足

$y_1(w^T + b) = 1$, 两边同时乘上 y_1 得到 $\sum_{i=1}^N \alpha_i y_i K_{i,1} + b = y_1$, 进而得到 b_1^{new} 的值:

$$b_1^{new} = y_1 - \sum_{i=3}^N \alpha_i y_i K_{i,1} - \alpha_1^{new} y_1 K_{1,1} - \alpha_2^{new} y_2 K_{2,1}$$

其中上式的前两项可以写成:

$$y_1 - \sum_{i=3}^N \alpha_i y_i K_{i,1} = -E_1 + \alpha_1^{old} y_1 K_{1,1} + \alpha_2^{old} y_2 K_{2,1} + b^{old}$$

当 $0 < \alpha_2^{new} < C$, 可以得到 b_{new2} 的表达式(推导同上):

$$b_2^{new} = -E_2 - y_1 K_{1,2} (\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{2,2} (\alpha_2^{new} - \alpha_2^{old}) + b^{old}$$

当 b_1 和 b_2 都有效的时候他们是相等的, 即 $b^{new} = b_1^{new}$

赞同 118

16 条评论

分享

收藏

...

当两个乘子 α_1, α_2 都在边界上, 且 $L \neq H$ 时, b_1, b_2 之间的值就是和KKT条件一致的阈值。SMO选择他们的中点作为新的阈值:

$$b^{new} = \frac{b_1^{new} + b_2^{new}}{2}$$

简化版SMO算法实现

这里我主要针对SMO中已选取的一对 α_i, α_j 值的优化过程进行下Python实现, 其中 α_i, α_j 的选取直接使用傻瓜的遍历方式, 并使用100数据点进行训练。

首先是一些辅助函数, 用来帮助加载数据, 修剪 α 的值以及随机选取 α

```
def load_data(filename):
    dataset, labels = [], []
    with open(filename, 'r') as f:
        for line in f:
            x, y, label = [float(i) for i in line.strip().split()]
            dataset.append([x, y])
            labels.append(label)
    return dataset, labels

def clip(alpha, L, H):
    ''' 修建alpha的值到L和H之间.
    ...

    if alpha < L:
        return L
    elif alpha > H:
        return H
    else:
        return alpha

def select_j(i, m):
    ''' 在m中随机选择除了i之外剩余的数
    ...

    l = list(range(m))
    seq = l[: i] + l[i+1:]
    return random.choice(seq)
```

为了能在最后绘制SVM分割线, 我们需要根据获取的 α , 数据点以及标签来获取 w 的值:

```
def get_w(alphas, dataset, labels):
    ''' 通过已知数据点和拉格朗日乘子获得分割超平面参数w
    ...

    alphas, dataset, labels = np.array(alphas), np.array(dataset), np.array(labels)
    yx = labels.reshape(1, -1).T*np.array([1, 1])*dataset
    w = np.dot(yx.T, alphas)
    return w.tolist()
```

简化版SMO算法的实现,即便没有添加启发式的 α 选取, SMO算法仍然有比较多的公式需要实现, 我本人按照上文的推导进行实现的时候就因为写错了一个下标算法一直跑不出想要的结果。

此实现主要包含两重循环, 外层循环是控制最大迭代步数, 此迭代步数是在每次有优化一对 α 之后进行判断所选取的 α 是否已被优化, 如果没有则进行加一, 如果连续max_iter步数之后仍然没有 α 被优化, 则我们就认为所有的 α 基本已经被优化, 优化便可以终止了。

```
def simple_smo(dataset, labels, C, max_iter):
    ''' 简化版SMO算法实现, 未使用启发式方法对alpha对进行选择.
    :param dataset: 所有特征数据向量
    :param labels: 所有的数据标签
    :param C: 软间隔常数, 0 <= alpha_i <= C
    :param max_iter: 外层循环最大迭代次数
    ...

    dataset = np.array(dataset)
    m, n = dataset.shape
```

```

labels = np.array(labels)
# 初始化参数
alphas = np.zeros(m)
b = 0
it = 0
def f(x):
    "SVM分类器函数  $y = w^T x + b$ "
    # Kernel function vector.
    x = np.matrix(x).T
    data = np.matrix(dataset)
    ks = data*x
    # Predictive value.
    wx = np.matrix(alphas*labels)*ks
    fx = wx + b
    return fx[0, 0]

while it < max_iter:
    pair_changed = 0
    for i in range(m):
        a_i, x_i, y_i = alphas[i], dataset[i], labels[i]
        fx_i = f(x_i)
        E_i = fx_i - y_i
        j = select_j(i, m)
        a_j, x_j, y_j = alphas[j], dataset[j], labels[j]
        fx_j = f(x_j)
        E_j = fx_j - y_j
        K_ii, K_jj, K_ij = np.dot(x_i, x_i), np.dot(x_j, x_j), np.dot(x_i, x_j)
        eta = K_ii + K_jj - 2*K_ij
        if eta <= 0:
            print('WARNING eta <= 0')
            continue
        # 获取更新的alpha对
        a_i_old, a_j_old = a_i, a_j
        a_j_new = a_j_old + y_j*(E_i - E_j)/eta
        # 对alpha进行修剪
        if y_i != y_j:
            L = max(0, a_j_old - a_i_old)
            H = min(C, C + a_j_old - a_i_old)
        else:
            L = max(0, a_i_old + a_j_old - C)
            H = min(C, a_j_old + a_i_old)
        a_j_new = clip(a_j_new, L, H)
        a_i_new = a_i_old + y_i*y_j*(a_j_old - a_j_new)
        if abs(a_j_new - a_j_old) < 0.00001:
            #print('WARNING alpha_j not moving enough')
            continue
        alphas[i], alphas[j] = a_i_new, a_j_new
        # 更新阈值b
        b_i = -E_i - y_i*K_ii*(a_i_new - a_i_old) - y_j*K_ij*(a_j_new - a_j_old) +
        b_j = -E_j - y_i*K_ij*(a_i_new - a_i_old) - y_j*K_jj*(a_j_new - a_j_old) +
        if 0 < a_i_new < C:
            b = b_i
        elif 0 < a_j_new < C:
            b = b_j
        else:
            b = (b_i + b_j)/2
        pair_changed += 1
        print('INFO iteration:{} i:{} pair_changed:{}'.format(it, i, pair_chan
    if pair_changed == 0:
        it += 1
    else:
        it = 0
    print('iteration number: {}'.format(it))
return alphas, b

```

Ok, 下面我们就用训练数据对SVM进行优化, 并对最后优化的分割线以及数据点进行可视化

```
if '__main__' == __name__:
    # 加载训练数据
    dataset, labels = load_data('testSet.txt')
    # 使用简化版SMO算法优化SVM
    alphas, b = simple_smo(dataset, labels, 0.6, 40)
    # 分类数据点
    classified_pts = {'+1': [], '-1': []}
    for point, label in zip(dataset, labels):
        if label == 1.0:
            classified_pts['+1'].append(point)
        else:
            classified_pts['-1'].append(point)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    # 绘制数据点
    for label, pts in classified_pts.items():
        pts = np.array(pts)
        ax.scatter(pts[:, 0], pts[:, 1], label=label)
    # 绘制分割线
    w = get_w(alphas, dataset, labels)
    x1, _ = max(dataset, key=lambda x: x[0])
    x2, _ = min(dataset, key=lambda x: x[0])
    a1, a2 = w
    y1, y2 = (-b - a1*x1)/a2, (-b - a1*x2)/a2
    ax.plot([x1, x2], [y1, y2])
    # 绘制支持向量
    for i, alpha in enumerate(alphas):
        if abs(alpha) > 1e-3:
            x, y = dataset[i]
            ax.scatter([x], [y], s=150, c='none', alpha=0.7,
                      linewidth=1.5, edgecolor='#AB3319')
    plt.show()
```

优化最后我们可以看到针对100个数据的 α 只有少部分是大于零的, 即对应的数据点就是支持向量:

为了能直观的显示支持向量，我将其标注了出来，最终可视化的效果如下图:

总结

本文从坐标上升算法开始介绍，并对SMO算法的原理进行了简单的推导，针对SMO算法中对 α 对的优化并使用了Python进行了简化版的SMO实现，并针对小数据集进行了训练得到了对应优化后的SVM。

实现代码以及训练数据链接: github.com/PytLab/MLBox...


参考

- 1. 优化算法——坐标上升法
- 2. [支持向量机系列（5）——SMO算法解对偶问题](#)
- 3. 《Machine Learning in Action》

编辑于 2017-09-13


SVM 机器学习

文章被以下专栏收录



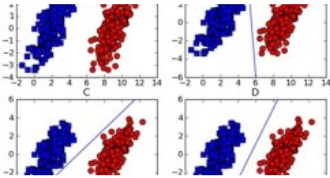
化学狗码砖的日常

进入专栏



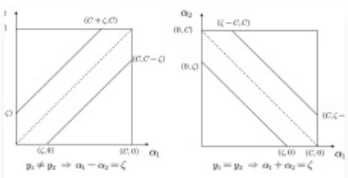
Python中文社区
微信公众号同名，欢迎投稿。全平台约20万开发者关注，会员来自全球十多个国家和...

进入专栏



机器学习算法实践-支持向量机(SVM)算法原理

PytLab酱



支持向量机(SVM)——SMO算法

余帅



机器学习

两仪式

16 条评论

切换为时间排序

写下你的评论...

知乎用户 1 年前

求助:那个分隔线是怎么和样本点画一块的? ? 弄半天没弄出来

赞

知乎用户 (作者) 回复 知乎用户 1 年前

你可以看一下我的代码, 就是需要先将w和b求出来, 然后找数据点中最大的x和最小的x, 代入分割线方程wx+b=0求出相应的y, 然后画直线

2 查看对话

知乎用户 回复 知乎用户 (作者) 1 年前

会了! 谢谢!

赞 查看对话

Xavier Li 1 年前

前两天才用c++写了个, 确实代码量没有想象的少

赞

Menothing 1 年前

你那个eta的判断和机器学习实战的里面不一样啊

赞

zgctmac 1 年前

为什么alpha1和alpha2在边界上时, b1和b2之间的任何数都满足要求, 能解释一下吗?

1

知乎用户 (作者) 回复 Menothing 1 年前

只是符号不同啦, 在alpha更新的时候还有一个负号, 实际是一样的

赞 查看对话

余泽 11 个月前

为什么可以区分a2 new和old。然后都出现在同一个公式中。那不应该讲两个a2都看成是一样的吗? 那就应该不要区分old和new才对。? ? ?

2

glxxx 10 个月前

求助: 我根据您的代码跑出来的alpha值是有小于0的, 但是alpha不应是大于零小于C的吗; 另外再按照及其学习实战里的代码跑出来, alpha值只有三到四个

赞同 118 16 条评论 分享 收藏 ...



zzy zzy 回复 余泽

9 个月前

同问啊！我也是没有搞明白这一点，你现在知道了吗？



赞



查看对话



陈秋远

4 个月前

终于找到个不是抄机器学习实战的了。。。



赞



我要一桶浆糊

4 个月前

作者你好，你举的坐标上升算法的例子那个函数应该是求max才对吧？



赞



Timtoy

3 个月前

大神好，github链接好像不能访问了，能否更新一下



赞



4Recommend

2 个月前

666



赞



Codex 回复 余泽

1 个月前

可以参考一下作者的第二个链接里面的。根据式子求出来的a2是新的a2，对v1和v2初始化的a2是old的。



1



查看对话



Codex

1 个月前

膜拜膜拜



赞