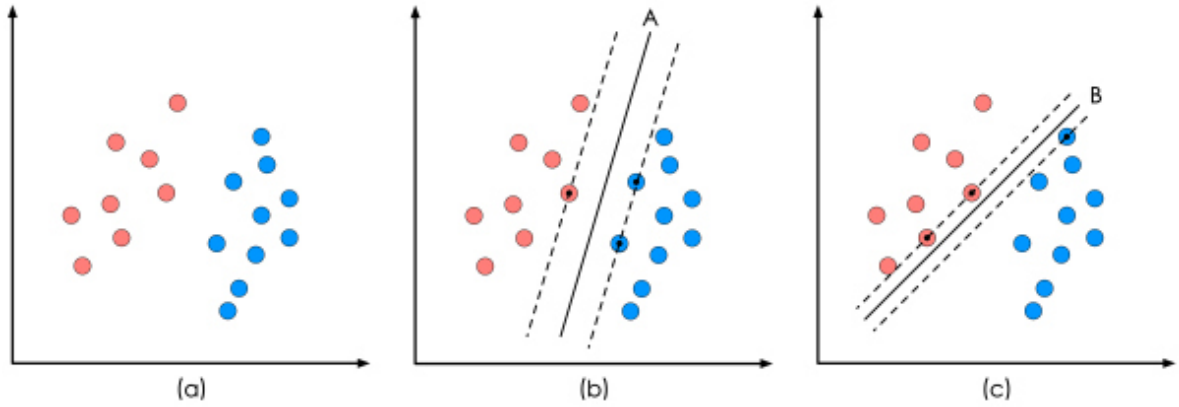
机器学习——支持向量机

算法思想：

当我们面对一个分类问题（假设是一堆小球），而且数据是线性可分的、可以将两类小球分开的时候，只要把分界线放在让小球离分界线距离最大化的位置就好了，在寻找这个间隔的过程，就叫做最优化。但是现实中，情况经常是线性不可分的，我们需要一个核函数，用于切分小球的界面就叫超平面。我们首先从线性可分开始着手，延申到线性不可分。

线性支持向量机：

对于线性可分的二分类情况：



可以看到，数据是可以明显做出分类的，但是分类方案不止一个，这些可能的分界面叫作“决策面”，它们代表了对应的线性分类器，虽然都能达到相同的分类效果，但是性能是有差距的可以通过添加测数据点验证分类正确性来衡量）。这里涉及到一个支持向量机的概念“分类间隔”，在保证决策面方向不变且不会出现错分样本的情况下移动决策面，在原来决策面两侧找到两个极限位置（越过这个位置就会产生错分现象），如虚线所示，虚线的位置由决策面的方向和距离原决策面最近的几个样本的位置决定，而这两条平行虚线正中间的分界线就是在保持当前决策面方向不变的前提下的最优决策面，两条虚线之间的垂直距离就是分类间隔，显然每个分类器都有自己的分类间隔，具有最大分类间隔的决策面就是我们要找的最优解，而位于虚线上的数据点就是我们一直在说的“支持向量”。

数学推导过程：

决策面

在支持向量机中，最终就是要使得“决策面”最优化，这个最优化有两个基本元素：目标函数、优化对象；因此我们需要对这两个因素进行深一步的探究，显然目标函数就是分类间隔的最大化，优化对象就是决策面，我们来看一下决策面：
在二维空间里，一条直线可以表示为：

$$ax_1 - x_2 + b = 0$$

向量化：

$$\begin{bmatrix} a & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = 0$$

用ω列向量和x列向量和标量γ进一步向量化：

$$\omega^T x + \gamma = 0$$

其中有：

$$\omega = [w_1, w_2]^T, x = [x_1, x_2]^T$$

对于线性代数还有数学有一定基础的人肯定可以直接看出来w向量和直线x_2向量的关系，它们是垂直关系，所以我们称w为直线的法向量。

推广至n维空间：

$$w = [w_1, w_2, w_3, \ldots, w_n]^T, x = [x_1, x_2, x_3, \ldots, x_n]^T$$

这就是我们推导出来的"决策面"方程，它就是我们的超平面方程。

分类间隔方程

对于间隔的表达式，我们直接给出空间内的表达式，大家可以由二维空间去推导：

$$d = \frac{|w^T x + \gamma|}{||w||}$$

这个d就是"分类间隔"，其中$||w||$表示w的二范数，求所有元素的平方和，然后开方；因此我们对于分类器的性能评估依据就是分类间隔的大小，最大化$W = 2d$就是我们的目的。

约束条件

对于二维平面上的两种点，我们把正样本标记为1，把负样本标记为-1，因此我们有分类方程：

$$\begin{cases} w^T x_i + \gamma > 0 & y_i = 1 \\ w^T x_i + \gamma < 0 & y_i = -1 \end{cases}$$

如果要求更加高一点，相应支持向量样本点到决策面的距离为d，那么公式可以写成：

$$\begin{cases} \frac{w^T x_i + \gamma}{||w||} \geq d & \forall y_i = 1 \\ \frac{w^T x_i + \gamma}{||w||} \leq -d & \forall y_i = -1 \end{cases}$$

对以上公式都除以d，可以写成：

$$\begin{cases} w_d^T x_i + \gamma_d \geq 1 & \forall y_i = 1 \\ w_d^T x_i + \gamma_d \leq -1 & \forall y_i = -1 \end{cases}$$
$$w_d = \frac{w}{||w||d}, \gamma_d = \frac{\gamma}{||w||d}$$

因为$||w||$和d都是标量，所以上面的式子依然表示一条直线，并且他们所表达的意义与原式子是一样的，同时我们去掉下标之后依然不变，表达式在原理上并没有改变模型，所以我们可进一步把式子归纳为：

$$y_i(w^T x_i + \gamma) \geq 1 \qquad \forall x_i$$

线性支持向量机优化问题

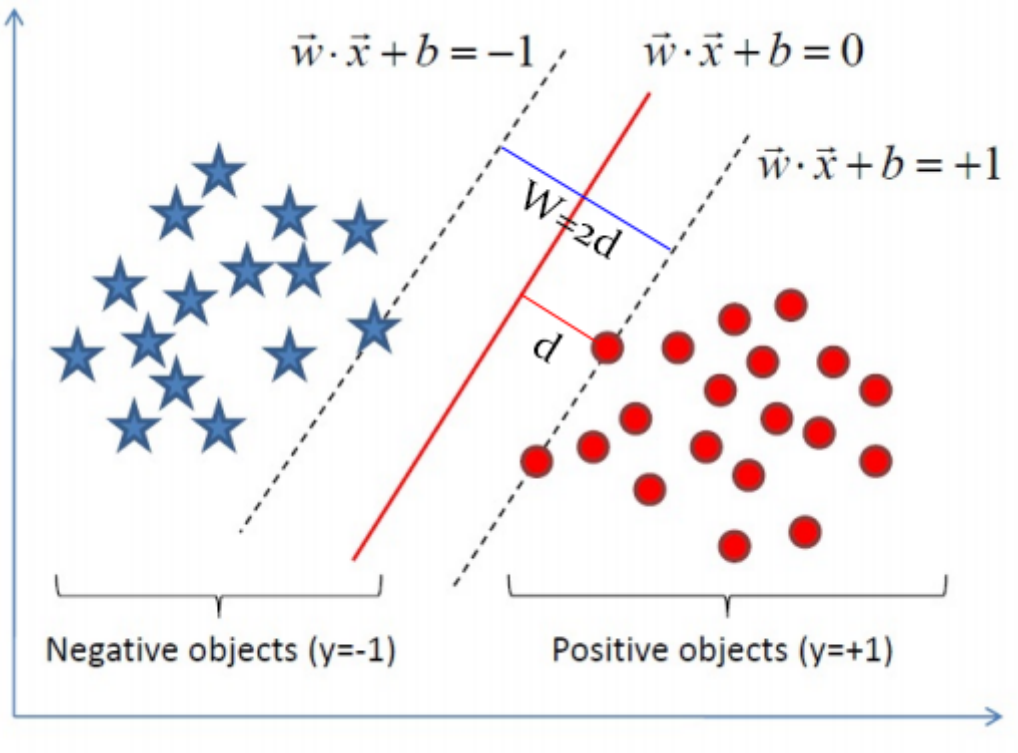
综上所述，我们的目标函数就是：

$$d = \frac{|w^T x + \gamma|}{||w||}$$

而在支持向量上所有的样本点都满足：

$$|w^T + \gamma| = 1 \qquad \forall \text{支持向量上的样本点}$$

如果还不懂上式子，请参考下图并重看上面的推导：



进一步化简上面的式子，就可以得到：

$$d=\frac{1}{||w||}$$

因此问题就由最大化d转化为最小化w，等效于：

$$min\frac{1}{2}||w||^2$$
$$s.t.\quad y_i(w^Tx_i+b)\geq 1, i=1,2,\ldots,n$$

求解准备：

对于这种优化问题，通常我们有三种求解方法：
（1）凸函数：直接使用高中的求导得到极值就可以了

（2）有等式约束的非凸函数：对约束等式用一个系数与非凸函数写成一个式子，这个式子叫作拉格朗日函数，我们对各个变量进行求导，令其为零，求导各个局部最优解，然后验证得到最优解。

（3）对于非等式非凸函数：需要用到了KKT条件，我们需要构造拉格朗日函数，通过这种方法求出最优值的必要条件就叫KKT条件。

在使用拉格朗日函数是，步骤分为：

1、将有约束的原始目标函数转为无约束的新构造的拉个朗日目标函数

2、使用拉格朗日对偶性，将不易求解的优化问题转化为易求解的优化

第一步：转化为拉格朗日函数：

$$L(w,b,\alpha)=\frac{1}{2}||w||^2-\sum_1^n\alpha_i(y_i(w^Tx_i+b)-1)$$

其中α_i是拉格朗日乘子，α_i大于等于0，我们令

$$\theta(w)=\max_{\alpha_i\geq 0}L(w,b,\alpha)$$

当不满足约束条件时，也就是在可行解区域外：

$$y_i(w^T x_i + b) < 1$$

我们设置α_i设置为正无穷，此时$\theta(w)$显然也是正无穷。

因此我们现在的目的就是最小拉格朗日函数：

$$\min_{w,b} \theta(w) = \min_{w,b} \max_{\alpha_i \geq 0} L(w,b,a) = p^*$$

第二步：转化求解：

根据拉格朗日的对偶性，交换最大最小位置：

$$\max_{\alpha \geq 0} \min_{w,b} L(w,b,\alpha) = d^*$$

根据性质，交换之后问题是原来问题的对偶问题，最优值为$d^* \leq p^*$，我们需要找到$d = q$的时候，才是最终的解。这需要：

这是个凸优化问题，同时满足KKT条件。我们可以看到我们想要求解的函数是凸函数，要满足KKT条件则需要：

对于优化问题的标准形式：

$$\begin{gathered} minf(x) \\ s.t.\, h_j(x) = 0, j = 1, 2, \ldots n \\ g_k(x) \leq 0, k = 1, 2, \ldots n \\ x \in X \subset \Re^n \end{gathered}$$

KKT条件对应的就是：

（1）拉格朗日处理后对x求导为零

（2）h（x)=0；

（3）$\alpha * g(x) = 0$

很明显我们的问题的都满足。

因此接下来就是求解对偶问题，对于对偶问题：

$$\max_{\alpha \geq 0} \min_{w,b} L(w,b,\alpha) = d^*$$

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_1^n \alpha_i (y_i(w^T x_i + b) - 1)$$

先求最小化，首先对w、b求偏导数，并且令其为零：

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_1^n \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_1^n \alpha_i y_i = 0$$

代入原式整理得：

$$L(w,b,a) = \sum_1^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j$$

总的来说，我们最终的优化问题变成：

$$\max_{\alpha} \sum_{1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$
$$s.t. \alpha_i \geq 0, i = 1, 2, \dots n$$
$$\sum_{1}^{n} \alpha_i y_i = 0$$

对于上面的式子求解，我们有更加有效的优化算法，就是SMO算法，通过这个算法能得到α，根据α求解w和b，将找到超平面。

SMO算法

SMO算法思想：通过循环选择两个alpha优化，找到一对合适的alpha，就增大其中一个，减小另一个，这里的合适是指：两个alpha在间隔边界外，同时两个alpha没有进行过区间化处理或者不在边界上。对于输出函数：

$$u = w^T x + b$$
$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{1}^{n} \alpha_i y_i x_i$$
$$u = \sum_{1}^{n} \alpha_i y_i x_i^T x + b$$

同时对上述的目标函数进行变形，前面增加一个符号，将最大值问题转换成最小值问题：

$$\min_{\alpha} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{1}^{n} \alpha_i$$
$$s.t. \alpha_i \geq 0, i = 1, 2, \dots n$$
$$\sum_{1}^{n} \alpha_i y_i = 0$$

由于线性完全可分是难以实现的，需要添加松弛变量和惩罚参数，经过一系列变化：

$$\text{目标函数：} \frac{1}{2} ||w||^2 + C \sum_{1}^{N} \xi_i$$

$$\min_{\alpha} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{1}^{n} \alpha_i$$
$$s.t. 0 \leq \alpha_i \leq C, i = 1, 2, \dots n$$
$$\sum_{1}^{n} \alpha_i y_i = 0$$

满足KKT条件：

$$a_i (y_i (w x_i + b) - 1 + \xi_i) = 0$$
$$\mu_i \xi_i = 0$$

对于样本点，输出：

$$u_i = w x_i + b$$

对于正定二次优化问题的优化点充要条件是KKT条件：

$$\alpha_i = 0 \leftrightarrow y_i u_i \geq 1$$
$$0 < \alpha_i < C \leftrightarrow y_i u_i = 1$$
$$\alpha_i = C \leftrightarrow y_i u_i \leq 1$$

对于上面几个式子的推导如下：

（1）$\alpha_i = 0$

由$C - \alpha_i - u_i = 0$得：$u_i = C$

因此：$\xi_i = 0 \rightarrow$ 约束条件：$y_i(wx_i + b) \geq 1 - \xi_i \rightarrow y_i u_i \geq 1$

（2）$0 < \alpha_i < C$

将u_i乘到上面的条件：$u_i\alpha_i y_i u_i - u_i\alpha_i + u_i\alpha_i\xi_i = 0$

$$u_i\alpha_i(y_i u_i - 1) = 0$$

由$C - \alpha_i - u_i = 0$得：$(C - \alpha_i)\alpha_i(y_i u_i - 1) = 0$

所以$y_i u_i = 1 \rightarrow \xi_i = 0$

（3）$\alpha_i = C$

$y_i u_i = 1 - \xi_i$，又因为$\xi_i \geq 0 \rightarrow y_i u_i \leq 1$

因此：

在间隔边界上，当$0 < \alpha_i^* < C$时，$y_i u_i = 1$时分类正确，且$u_i = \pm 1$，也就是在间隔边界上。

当$\alpha_i^* = C, 0 < \xi_i < 1$:

$$0 < y_i u_i < 1$$

则当y和u同号，分类正确，间隔下于1，在间隔边界内。

当$\alpha_i^* = C, \xi_i = 1$:

$$y_i u_i = 0 \rightarrow u_i = 0$$

也就是说在分离超平面上。

当$\alpha_i = C, \xi_i > 1$:

$$y_i u_i < 0$$

分类错误，在分离超平面误分的一侧。

只有在满足KKT条件的alpha只能被保存，如果存在不满足KKT条件的α_i那么我们需要更新这些α_i，这是第一个约束条件，同时更新需要受到第二个约束条件限制：

$$\sum_i^n a_i y_i = 0$$

因为这个条件，我们同时更新两个alpha值，成对更新，才能保证更新后和为0的约束，假设我们选择两个乘子：

$$\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \xi$$

因为两个因子不好同时求解，所以可以先求第二乘子的解，再求第一个乘子的解，为了求得第二个解α_2，需要先确定它的取值范围，假设其上下界：

$$L \leq \alpha_2^{new} \leq H$$

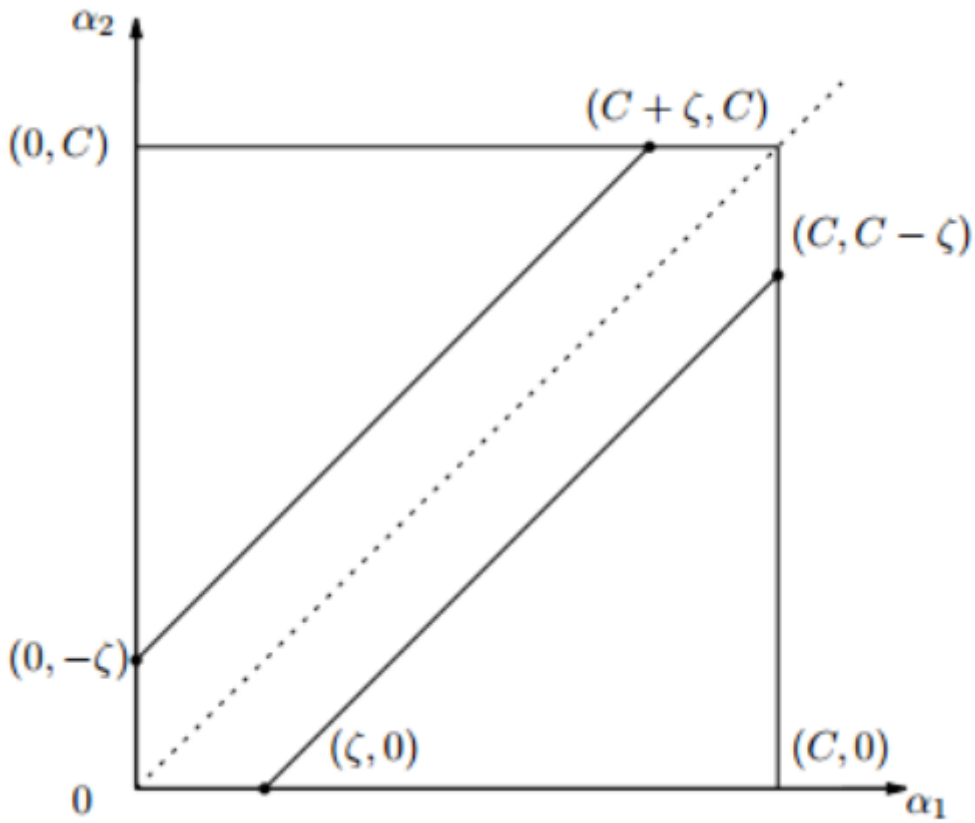
我们上面的式子，当y1不等于y2时，一个为正1，另一个负1，得到：

$$\alpha_1^{old} - \alpha_2^{old} = \xi$$

所以有：

$$L = max(0, -\xi), H = min(C, C - \xi)$$

如下图：

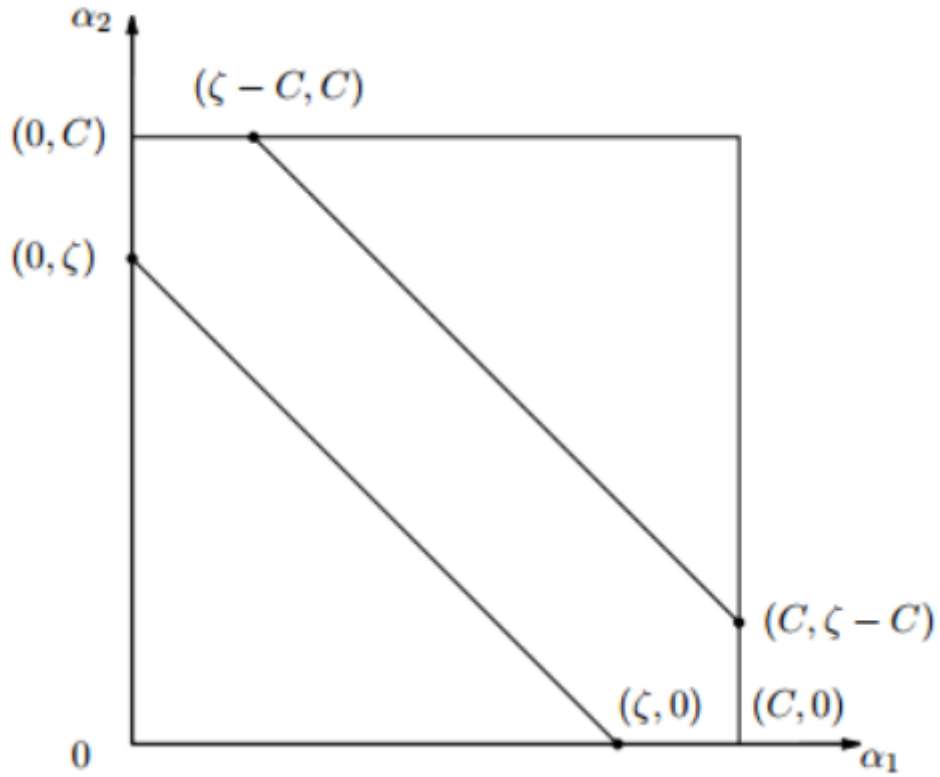


当y1=y2时，可以得到：

$$\alpha_1^{old} + \alpha_2^{old} = \xi$$

有：

$$L = max(0, \xi - C), H = min(C, \xi)$$



因此，我们确定了边界后，就要推导迭代式，用于更新alpha值。我们首先固定两个乘子，进行推导，得到目标函数：

$$W(\alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2}\alpha_1^2 x_1^T x_1 - \frac{1}{2}\alpha_2^2 x_2^T x_2 - y_1 y_2 \alpha_1 \alpha_2 x_1^T x_2 - y_1 a_1 v_1 - y_2 a_2 v_2 + constant$$

$$v_i = \sum_{j=3}^{n} \alpha_j y_j x_i^T x_j = f(x_i) - \sum_{j=1}^{2} \alpha_j y_j x_i^T x_j - b$$

$$f(x_i) = \sum_{j=1}^{n} \alpha_j y_j x_i^T x_j + b$$

现在需要推导出两个乘子之间的关系，才可以求导。

$$\text{因为：} \sum_{1}^{n} \alpha_i y_i = 0$$

$$\text{有：} \alpha_1 y_1 + \alpha_2 y_2 = B$$

$$\text{同乘于} y1\text{：} \alpha_1 = \gamma - s\alpha_2$$

代入目标函数：

$$W(\alpha_2) = \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}(\gamma - s\alpha_2)^2 x_1^T x_1 - \frac{1}{2}\alpha_2^2 x_2^T x_2 - s(\gamma - s\alpha_2)\alpha_2 x_1^T x_2 - y_1(\gamma - s\alpha_2)v_1 - y_2\alpha_2 v_2 + constant$$

对其求偏导令其为零，（s=y1y2，s的平方为1）：

$$\alpha 2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

$$E_i = f(x_i) - y_i$$

$$\eta = x_1^T x_1 + x_2^T x_2 - 2x_1^T x_2$$

到那时没有剪辑，需要考虑约束条件：

$$0 < \alpha_i < C$$

所以最后的解：

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

又因为：

$$\alpha_1^{old} = \gamma - s\alpha_2^{old}$$
$$\alpha_1^{new} = \gamma - s\alpha_2^{new,clipped}$$

有：

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

由此可得：

$$b = \begin{cases} b_1 & 0 < \alpha_1^{new} < C \\ b_2 & 0 < \alpha_2^{new} < C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

总结以上，我们得到SMO算法的步骤：

首先计算误差：$E_i = f(x_i) - y_i = \sum_{j=1}^{n} \alpha_j y_j x_i^T x_j + b - y_i$

然后计算上下界：

$$\begin{cases} L = max(0, \alpha_j^{old} - \alpha_i^{old}), H = min(C, C + \alpha_j^{old} - \alpha_i^{old}) & if \quad y_i \neq y_j \\ L = max(0, \alpha_j^{old} + \alpha_i^{old} - C), H = min(C, \alpha_j^{old} + \alpha_i^{old}) & if \quad y_i = y_j \end{cases}$$

计算η：

$$\eta = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$$

更新α_j：

$$\alpha_j^{new} = \alpha_j^{old} + \frac{yi(E_i - E_j)}{\eta}$$

经过修剪：

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

更新α_i：

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

更新b1和b2：

$$b_1^{new} = b^{old} - E_i - y_(\alpha_i^{new} - \alpha_i^{old}) x_i^T x_i - y_j (\alpha_j^{new} - \alpha_j^{old}) x_j^T x_i$$
$$b_2^{new} = b_{old} - E_j - y_i (\alpha_i^{new} - \alpha_i^{old}) x_i^T x_j - y_j (\alpha_j^{new} - \alpha_j^{old}) x_j^T x_j$$

之更新b：

$$b = \begin{cases} b_1 & 0 < \alpha_1^{new} < C \\ b_2 & 0 < \alpha_2^{new} < C \\ (b_1 + b_2)/2 & otherwise \end{cases}$$

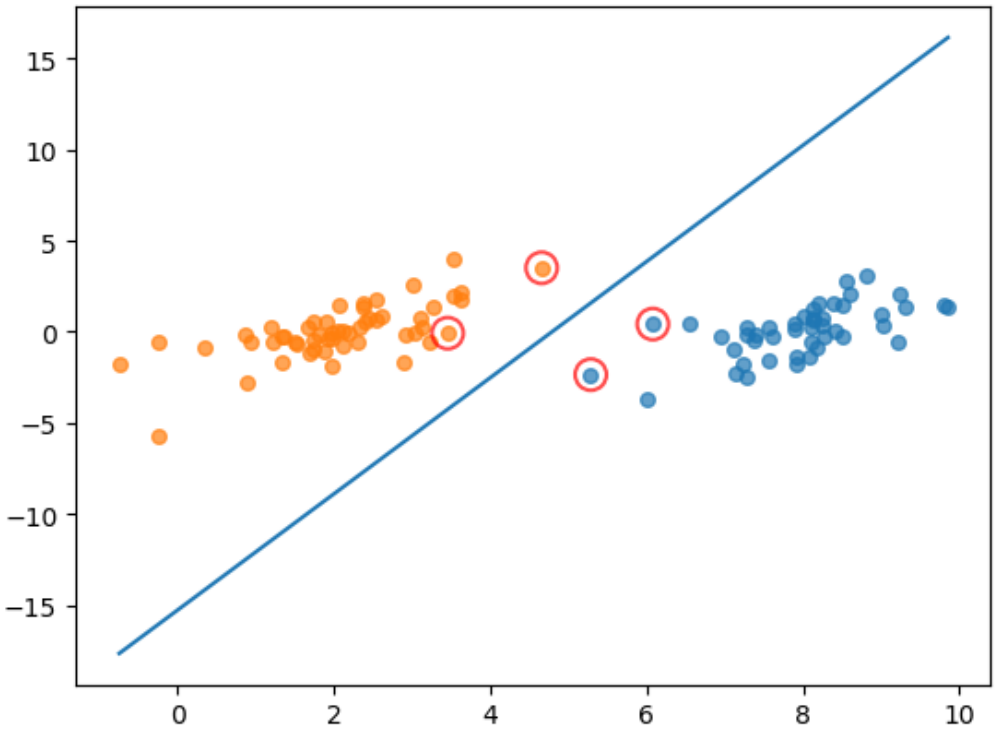
对于上述原理的代码实现：

```
from time import sleep
import matplotlib.pyplot as plt
import numpy as np
import random
import types

def loadDataSet(filename):
    dataMat=[]
    lableMat=[]
    fr=open(filename)
    for line in fr.readlines():
        lineArr=line.strip().split('\t')
        dataMat.append([float(lineArr[0]),float(lineArr[1])])
        lableMat.append(float(lineArr[2]))
    return dataMat,lableMat

#随机选择alpha值
def selectJrand(i,m):
    j=i
    while(j==i) :
        j=int(random.uniform(0,m))
    return j

#修剪alpha
def clipAlpha(aj,H,L):
    if aj>H:
        aj=H
    if L>aj:
        aj=L
    return aj

#简化版smo算法:
def smoSimple(dataMatIn,classLabels,C,toler,maxIter):
    dataMatrix =np.mat(dataMatIn)
    lableMat=np.mat(classLabels).transpose()
    b=0
    m,n=np.shape(dataMatrix)
    alphas=np.mat(np.zeros((m,1)))
    iter_num=0
    while(iter_num<maxIter):
        alphaPairsChanged=0
        for i in range(m):
            fXi=float(np.multiply(alphas,lableMat).T*
(dataMatrix*dataMatrix[i,:].T))+b
            Ei=fXi-float(lableMat[i])
            #优化alpha
            if((lableMat[i]*Ei< -toler) and (alphas[i]<C)) or
((lableMat[i]*Ei>toler) and (alphas[i]>0)):
                j=selectJrand(i,m)
                fXj=float(np.multiply(alphas,lableMat).T*
(dataMatrix*dataMatrix[j,:].T))+b
                Ej=fXj-float(lableMat[j])
                alphaIold =alphas[i].copy()
                alphaJold=alphas[j].copy()
                #计算上下界
                if(lableMat[i]!=lableMat[j]):
                    L=max(0,alphas[j]-alphas[i])
                    H=min(C,C+alphas[j]-alphas[i])
```

```
                else:
                    L=max(0,alphas[j]+alphas[i]-C)
                    H=min(C,alphas[j]+alphas[i])
                if L==H:
                    print("L==H")
                    continue
                #步骤3：计算eta
                eta=2.0*dataMatrix[i,:]*dataMatrix[j,:].T-
dataMatrix[i,:]*dataMatrix[i,:].T-dataMatrix[j,:]*dataMatrix[j,:].T
                if eta>=0:
                    print("eta>=0")
                    continue
                #更新alpha_j
                alphas[j] -= lableMat[j]*(Ei-Ej)/eta
                alphas[j]=clipAlpha(alphas[j],H,L)
                if(abs(alphas[j]-alphaJold)<0.00001):
                    print("alpha_j变化太小")
                    continue
                alphas[i]+=lableMat[j]*lableMat[i]*(alphaJold-alphas[j])
                b1=b-Ei-lableMat[i]*(alphas[i]-
alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T-lableMat[j]*(alphas[j]-
alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
                b2=b-Ej-lableMat[i]*(alphas[i]-
alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T-lableMat[j]*(alphas[j]-
alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
                #更新b
                if(0<alphas[i]) and (C>alphas[i]):
                    b=b1
                elif (0<alphas[j]) and (C>alphas[j]):
                    b=b2
                else:
                    b=(b1+b2)/2.0
                alphaPairsChanged+=1
                print("第%d次迭代 样本：%d， alpha优化次数：%d" %
(iter_num,i,alphaPairsChanged))
        if(alphaPairsChanged==0):
            iter_num+=1
        else:
            iter_num=0
        print("迭代次数： %d" % iter_num)
    return b,alphas

def showClassifer(dataMat,w,b):
    data_plus=[]
    data_minus=[]
    for i in range(len(dataMat)):
        if lableMat[i]>0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np=np.array(data_plus)
    data_minus_np=np.array(data_minus)
    plt.scatter(np.transpose(data_plus_np)[0],np.transpose(data_plus_np)
[1],s=30,alpha=0.7)
    plt.scatter(np.transpose(data_minus_np)[0],np.transpose(data_minus_np)
[1],s=30,alpha=0.7)
    x1=max(dataMat)[0]
    x2=min(dataMat)[0]
```

```
    a1,a2=w
    b=float(b)
    a1=float(a1[0])
    a2=float(a2[0])
    y1=(-b-a1*x1)/a2
    y2= (-b-a1*x2)/a2
    plt.plot([x1,x2],[y1,y2])
    for i,alpha in enumerate(alphas):
        if alpha>0:
            x,y=dataMat[i]
            plt.scatter([x],
[y],s=150,c='none',alpha=0.7,linewidth=1.5,edgecolor='red')
    plt.show()

def get_w(dataMat,lableMat,alphas):

alphas,dataMat,lableMat=np.array(alphas),np.array(dataMat),np.array(lableMat)
    w=np.dot((np.tile(lableMat.reshape(1,-1).T,(1,2))*dataMat).T,alphas)
    return w.tolist()

dataMat,lableMat=loadDataSet('testSet.txt')
b,alphas=smoSimple(dataMat,lableMat,0.6,0.001,40)

w=get_w(dataMat,lableMat,alphas)

showClassifer(dataMat,w,b)
```

结果:



终于写完了推导，其实还是挺麻烦的，同时对于SMO算法的实现还没有使用启发式，没有优化，下一次的更新会深入研究，同时对于支持向量机的算法推导将做更进一步的理解。