第七章 支持向量机

支持向量机（support vector machine，简称SVM）于1964年由Vapnik和Chervonenkis建立，在上世纪90年代获得快速发展并衍生出一系列改进和扩展算法，在人像识别、文本分类、手写字识别及生物信息学等领域获得广泛应用。

1 间隔与支持向量

以二分类问题为例，两个类别分别以“+1”和“-1”表示，对于训练样本集$D=\{(x_0,y_0),(x_1,y_1),\cdots,(x_{n-1},y_{n-1})\}$,$y_i\in\{-1,+1\}$。要实现正确分类，就是要基于训练集D在样本空间中找到一个划分超平面，将不同类别的样本分开。对于样本有两个特征的情况，如图7-1所示，此时划分超平面为直线。能将训练样本分开的直线有很多，我们应该取哪一条呢？

显然应该去找位于两类训练样本“正中间”的直线，即图7-1中的粗线。因为该直线对训练样本局部扰动的容忍性最好。

例如，由于训练集的局限性或噪声的因素，训练集外的样本可能比图7-1中的样本更接近两个类的分隔界，这将使许多划分直线出现错误，而粗线受影响最小。换言之，这条直线对未见样本的泛化能力最强。

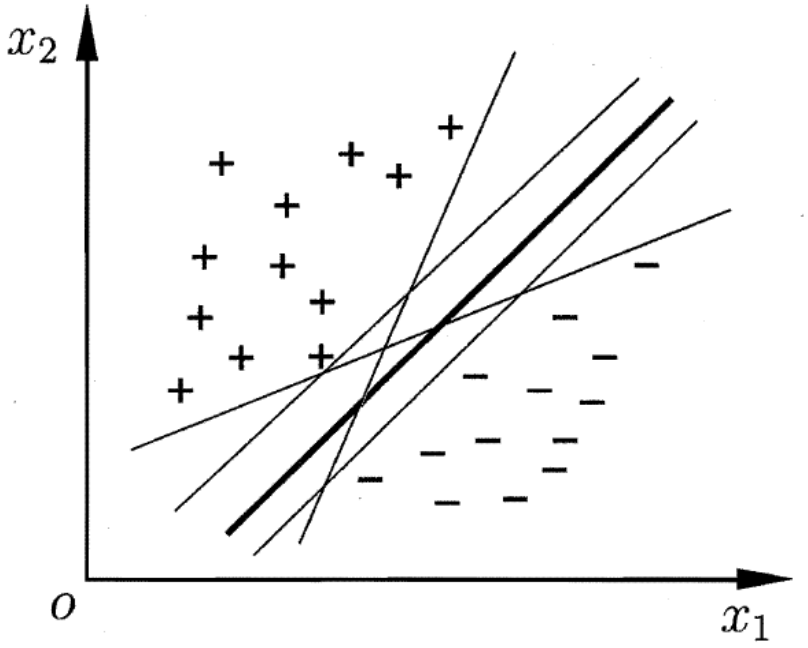


图7-1 将两类训练样本正确分类的多个超平面

在样本空间中，划分超平面可通过如下线性方程来描述：

$$\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x} + b' = w_0' x_0 + w_1' x_1 + \cdots + w_{m-1}' x_{m-1} + b' = 0$$

其中**w'**为法向量，b'为位移项。显然划分超平面可被法向量和位移确定，我们记该超平面为（**w'**, b'）。样本空间中任意点**x**到超平面（**w'**, b'）的距离可写为：

$$r = \frac{\left|\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x} + b'\right|}{\|\boldsymbol{w}'\|}$$

假设超平面（**w'**, b'）能将训练样本正确分类，过正类样本点中距离分隔超平面最近的点，作与超平面相平行的超平面，其方程为：

$$\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x} + b' = d$$

相应地，由于超平面位于样本点中间，过负类样本点中距离分隔超平面最近的点，与超平面相平行的超平面方程为：

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b' = -d$$

这三个超平面方程，都同时除以一个数d，不会改变它们的位置，因此，三个超平面方程转变为：

分隔超平面　　$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$

正类样本一侧　$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 1$

负类样本一侧　$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = -1$

对于任意样本点$(x_i, y_i) \in D$，有：

$$\begin{cases} \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b \geq 1, y_i = +1 \\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b \leq -1, y_i = -1 \end{cases}$$

距离分隔超平面最近的训练样本点使上式中等号成立，它们被称为“支持向量”，两个异类支持向量到超平面的距离之和为：

$$\gamma = \frac{2}{\|\boldsymbol{w}\|}$$

γ称为间隔（margin）。

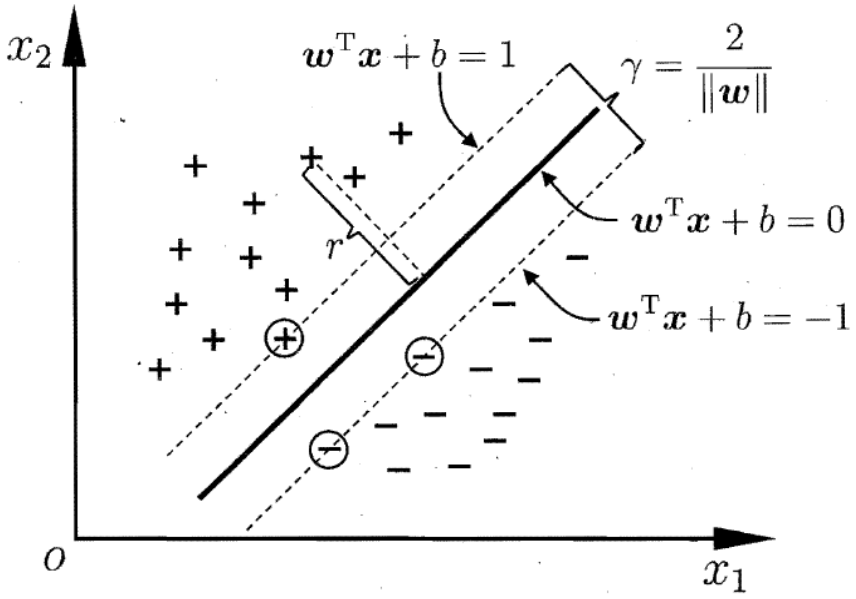


图7.2 支持向量与间隔

要找到间隔最大的划分超平面，也就是要找到能满足约束的参数**w**和b，使得γ最大，即：

$$\max_{\boldsymbol{w},b} \frac{2}{\|\boldsymbol{w}\|}$$

$$\text{s.t.}\, y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1, i = 0,1,\cdots,n-1$$

$$\max_{\boldsymbol{w},b} \frac{2}{\|\boldsymbol{w}\|}$$

显然，最大化 $\|\boldsymbol{w}\|^{-1}$ 等价于最小化 $\|\boldsymbol{w}\|$ 或 $\|\boldsymbol{w}\|^2$

因此，优化问题可重写为：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{s.t. } y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1, i = 0,1,\cdots,n-1$$

这就是支持向量机的基本形式。

可以看出，支持向量机最终转化为一个有约束的优化问题，虽然可以利用一些现成的优化程序包求解，但通常采用拉格朗日乘子法将其转化为对偶问题求解。

为此，我们先来介绍有约束优化问题的KKT条件。

2 KKT条件

我们知道，对于无约束优化问题minf(x)，令函数对所有自变量的偏导为0，即函数梯度$\nabla_x f(x)=0$，求解即可。

对于具有等式约束条件的优化问题：

$$\min f(\boldsymbol{x})$$
$$\text{s.t.}\, h_m(\boldsymbol{x}) = 0, m = 0, 1, \cdots, M-1$$

如果约束方程足够简单，可通过消元将其转变为无约束问题。但如果约束方程较为复杂，可以利用拉格朗日乘子法解决。定义拉格朗日函数：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x})$$

$$L(\boldsymbol{x},\boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x})$$

然后令*L*对x及λ的梯度为0：

$$\begin{cases} \nabla_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{\lambda}) = 0 \\ \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x},\boldsymbol{\lambda}) = 0 \end{cases}$$

解上式求得x及λ的值。

为了解释这样做的原因，我们以二维问题f(x,y)的优化为例，假设只有一个等式约束条件h(x,y)=0。

图7-3画出了f的等高线（虚线）及约束条件线（实线）。

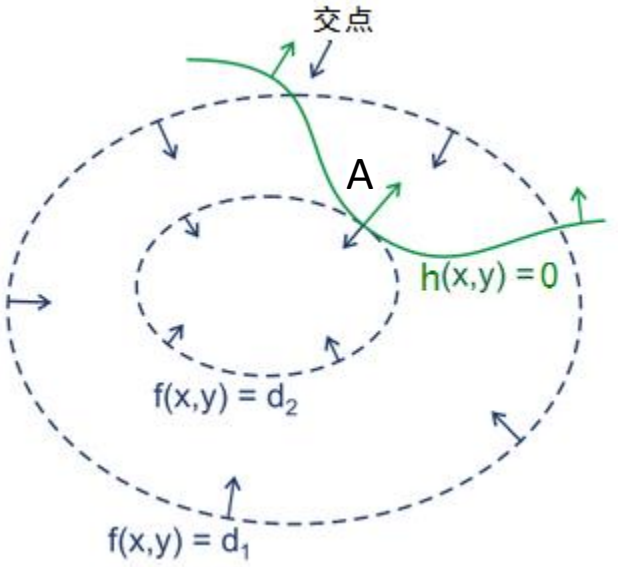
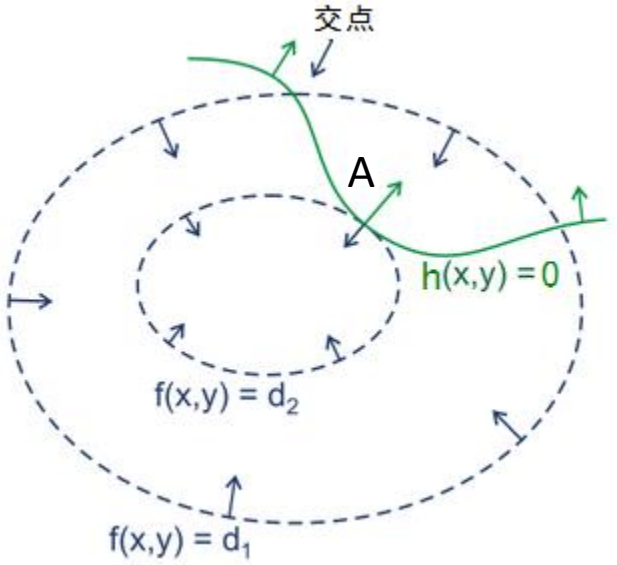


图7-3 具有等式约束条件的优化问题示意图

显然，可行解一定在约束线上并使得f最小的点处，这个点一定在约束线与某条等高线相切的位置，即图中A点处，此点处约束线梯度∇h与f的梯度∇f方向相同或相反，即有：



$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \lambda \nabla_{\boldsymbol{x}} h(\boldsymbol{x}) = 0$$

满足上式且满足约束条件h(x)=0的点即为可能的解，联立起来，正好就是拉格朗日乘子法的结果：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x})$$

$$\begin{cases} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = 0 \\ \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = 0 \end{cases}$$

对于即包含等式约束又包含不等式约束的优化问题：

$$\min f(\boldsymbol{x})$$

$$\text{s.t. } h_m(\boldsymbol{x}) = 0, m = 0, 1, \cdots, M-1$$
$$g_k(\boldsymbol{x}) \leq 0, k = 0, 1, \cdots, K-1$$

KKT（Karush-Kuhn-Tucher）条件给出了最优解的必要条件，KKT条件是拉格朗日乘子法的推广。先定义拉格朗日函数：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x}) + \sum_{k=0}^{K-1} \alpha_k g_k(\boldsymbol{x})$$

KKT条件如下：

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = 0$$

$$h_m(\boldsymbol{x}) = 0, m = 0, 1, \cdots, M-1$$

$$\alpha_k g_k(\boldsymbol{x}) = 0, k = 0, 1, \cdots, K-1$$

$$\alpha_k \geq 0$$

可以看到，KKT条件中对等式约束的处理与上面的拉格朗日乘子法完全相同，下面我们以二维问题$f(x_1,x_2)$的优化为例，对不等式约束条件的处理进行解释，为简单起见，考虑只有一个不等式约束的情况。如图7-4所示，依据$f(x_1,x_2)$无约束极小值点x*的位置分两种情况

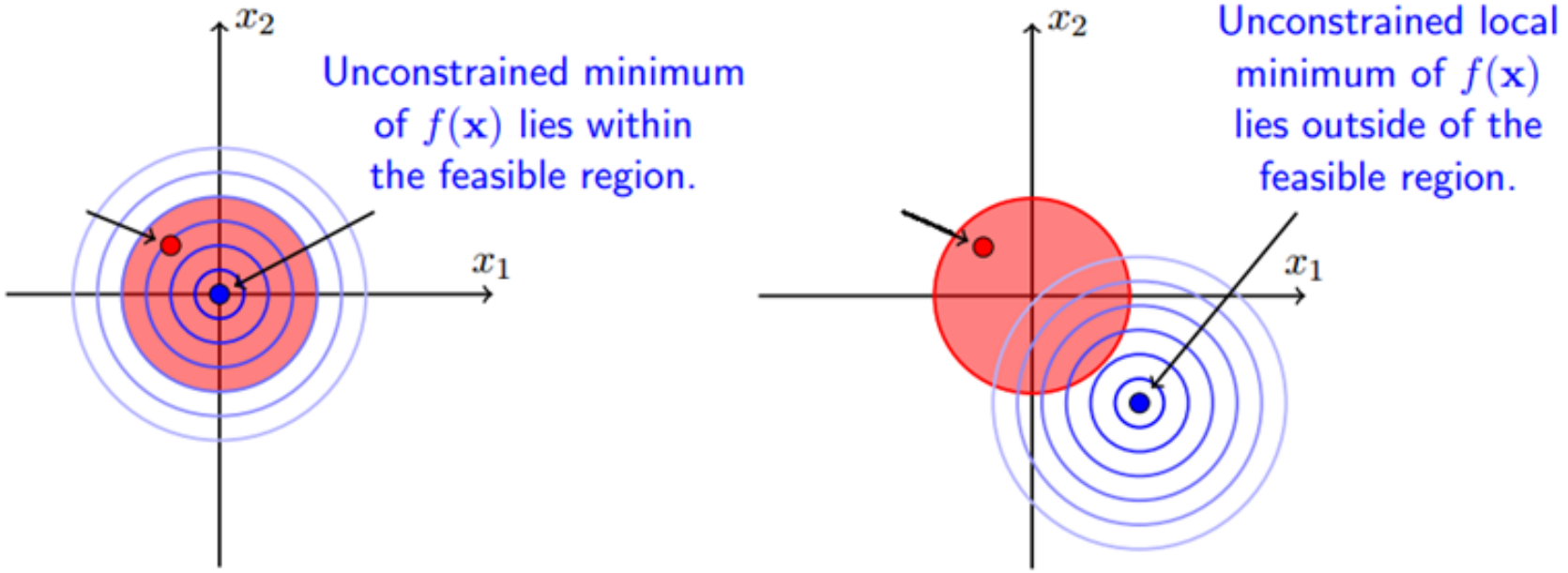


图7-4 不等式约束条件优化问题图示

(1) x*位于g(x)<0的区域内（图7-4a），此时必然存在x*的一个邻域全部位于g(x)≤0的约束区域之内，这时按无约束问题求解即可，约束条件g(x)≤0不起作用。

(2) x*位于 g(x)<0的区域之外（图7-4b），此时可行的极小值点必然在g(x)≤0区域内且使得f(x_1,x_2)最小的位置，即处于g(x)=0且与f的某条等高线相切的位置，这与等式约束的情况相同。

综合以上两种情况，可得：

$$\alpha g(\boldsymbol{x}) = 0$$

α=0对应于第(1)种情况，此时约束条件不起作用，令α=0消去约束条件即可；

g(x)=0对应于第(2)种情况，此时可行解落在约束边界上。

最后解释为什么要求α≥0。对于第(1)种情况，α=0显然满足。对于第(2)种情况，极小值出现在g(x)=0与f的某条等高线相切的位置，该点满足：

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \alpha \nabla_{\boldsymbol{x}} g(\boldsymbol{x}) = 0$$

将该式重写为：

$$-\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \alpha \nabla_{\boldsymbol{x}} g(\boldsymbol{x})$$

由于梯度表示一个函数增速最大的方向，相应地，$-\nabla_x f(x)$表示f减小最快的方向。如果α<0，则$\alpha\nabla_x g(x)$也指向g减小最快的方向，此方向指向符合约束条件的区域，从切点沿此方向前进，必然存在一点即在可行区域，又使得f减小，该点f值小于切点，这与切点为极小值点的假设矛盾。因此必须α>0。总之，对(1)、(2)两种情况，都有α≥0。

3 对偶问题

一个优化问题可以从两个角度考察，即”主问题” (primal problem)和”对偶问题” (dual problem)。对于上节的主问题：

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s.t. } h_m(\boldsymbol{x}) = 0, m = 0, 1, \cdots, M-1 \\ &\quad\ g_k(\boldsymbol{x}) \le 0, k = 0, 1, \cdots, K-1 \end{aligned}$$

基于其拉格朗日函数：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x}) + \sum_{k=0}^{K-1} \alpha_k g_k(\boldsymbol{x})$$

定义其对偶函数为：

$$\Gamma(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \inf_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x}) + \sum_{k=0}^{K-1} \alpha_k g_k(\boldsymbol{x})$$

$$\Gamma(\boldsymbol{\lambda},\boldsymbol{\alpha}) = \inf_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{m=0}^{M-1} \lambda_m h_m(\boldsymbol{x}) + \sum_{k=0}^{K-1} \alpha_k g_k(\boldsymbol{x})$$

其中inf表示下确界，即最大下界。若 $\tilde{\boldsymbol{x}}$ 为主问题可行域中的点，则对任意α≥0和λ都有

$$\sum_{m=0}^{M-1} \lambda_m h_m(\tilde{\boldsymbol{x}}) + \sum_{k=0}^{K-1} \alpha_k g_k(\tilde{\boldsymbol{x}}) \leq 0$$

进而有：

$$\Gamma(\boldsymbol{\lambda},\boldsymbol{\alpha}) = \inf_{\boldsymbol{x}} L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\alpha}) \leq L(\tilde{\boldsymbol{x}},\boldsymbol{\lambda},\boldsymbol{\alpha}) \leq f(\tilde{\boldsymbol{x}})$$

若主问题的最小值为p*，则对任意α≥0和λ都有：

$$\Gamma(\boldsymbol{\lambda},\boldsymbol{\alpha}) \leq p^*$$

$$\Gamma(\boldsymbol{\lambda}, \boldsymbol{\alpha}) \leq p^*$$

因此对偶函数给出了主问题最小值的下界。显然这个下界取决于α和λ的值。于是一个很自然的问题是：基于对偶函数能获得的最大下界是什么？这就引出了优化问题：

$$\max_{\boldsymbol{\lambda},\boldsymbol{\alpha}} \Gamma(\boldsymbol{\lambda}, \boldsymbol{\alpha}), \text{s.t.} \boldsymbol{\alpha} \geq 0$$

上式就是前述主问题的对偶问题，其中λ和α称为”对偶变量”。

针对于支持向量机的基本形式:

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{s.t.} \, y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1, i = 0,1,\cdots,n-1$$

将约束条件重写为g(x)≤0的形式：

$$1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \leq 0$$

拉格朗日函数为：

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=0}^{n-1} \alpha_i [1 - y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i + b)]$$

令L(w,b,α)对w和b的偏导为0：

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial w_j} = w_j - \sum_{i=0}^{n-1} \alpha_i y_i x_{ij} = 0$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = -\sum_{i=0}^{n-1} \alpha_i y_i = 0$$

因而有：

$$\boldsymbol{w} = \sum_{i=0}^{n-1} \alpha_i y_i \boldsymbol{x}_i \qquad \sum_{i=0}^{n-1} \alpha_i y_i = 0$$

$$\boldsymbol{w} = \sum_{i=0}^{n-1} \alpha_i y_i \boldsymbol{x}_i \qquad \sum_{i=0}^{n-1} \alpha_i y_i = 0$$

将上面两式代回拉格朗日函数：

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=0}^{n-1} \alpha_i [1 - y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i + b)]$$

$$= \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=0}^{n-1} \alpha_i - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - b\sum_{i=0}^{n-1} \alpha_i y_i$$

$$= \sum_{i=0}^{n-1} \alpha_i - \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

再考虑到 $\sum_{i=0}^{n-1} \alpha_i y_i = 0$ 的约束，从而得到对偶问题：

$$\max_{\boldsymbol{\alpha}} \sum_{i=0}^{n-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$\text{s.t.} \sum_{i=0}^{n-1} \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 0,1,\cdots,n-1$$

解对偶问题得到α后，将其代入 $\boldsymbol{w} = \sum_{i=0}^{n-1} \alpha_i y_i \boldsymbol{x}_i$ 即可求出w。

b可通过任一支持向量（x_s,y_s）求出，因为在支持向量处满足$y_s(w^T x_s+b)=1$。现实任务中常采用一种更稳健的做法：使用所有支持向量求解的平均值：

$$b = \frac{1}{|S|} \sum_{s \in S} \left(\frac{1}{y_s} - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_s \right) = \frac{1}{|S|} \sum_{s \in S} \left(y_s - \sum_{i \in S} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_s \right)$$

$$b = \frac{1}{|S|}\sum_{s\in S}\left(\frac{1}{y_s} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_s\right) = \frac{1}{|S|}\sum_{s\in S}\left(y_s - \sum_{i\in S}\alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_s\right)$$

其中，S={i|α_i>0, i=0,1,⋯,n-1}为所有支持向量的下标集，|S|表示S集合的长度。

解出w和b即可得到模型：

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = \sum_{i=0}^{n-1}\alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j + b$$

解得的任何α_i都对应一个样本（x_i,y_i），根据KKT条件，如果α_i=0，则该样本不会对f(x)有任何影响；若α_i>0，则必有$y_i f(x_i)$=1，所对应的样本是一个支持向量。这是支持向量机的一个重要性质：训练完成后，大部分的训练样本都不需保留，最终模型仅与支持向量有关。

4. 软间隔支持向量机

前面的讨论中假定训练样本是线性可分的，即存在一个超平面能将不同类的样本完全分开。但现实问题也许不是线性可分的，解决该问题的办法之一是允许支持向量机在一些样本上出错，为此引入“软间隔”（soft margin）的概念，即允许某些样本不满足$y_i(w^Tx_i+b)\geq 1$的约束条件。相应地，把前面要求所有样本都满足该约束条件即所有样本都必须分类正确的情况称为“硬间隔”（hard margin）。

对软间隔的支持向量机，在最大化间隔的同时，希望不满足约束的样本应尽可能少，于是优化目标可写为：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=0}^{n-1} \ell_{0/1}(y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) - 1)$$

其中C>0是一个常数，$\ell_{0/1}$ 是"0/1损失函数"：

$$\ell_{0/1}(\mathrm{z}) = \begin{cases} 1, & \mathrm{z} < 0 \\ 0, & \mathrm{z} \geq 0 \end{cases}$$

显然，当C为无穷大时，优化目标函数将迫使所有样本均满足约束，于是软间隔等价于硬间隔，当C取有限值时，软间隔允许一些样本不满足约束。

引入“松驰变量”$\xi_i \geq 0$，可将上式重写为：

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=0}^{n-1}\xi_i$$

$$\text{s.t. } y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0, i = 0,1,\cdots,n-1$$

这就是常用的“软间隔支持向量机”。显然，每个样本都有一个对应的松驰变量，用以表征该样本不满足约束的程度。

对软间隔支持向量机，定义拉格朗日函数：

$$L(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\xi},\boldsymbol{\mu})=\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=0}^{n-1}\xi_i+\sum_{i=0}^{n-1}\alpha_i\left(1-\xi_i-y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)\right)-\sum_{i=0}^{n-1}\mu_i\xi_i$$

令L(w,b,α,ξ,μ)对w, b, ξ_i的偏导为零可得：

$$\boldsymbol{w}=\sum_{i=0}^{n-1}\alpha_i y_i\boldsymbol{x}_i \qquad \sum_{i=0}^{n-1}\alpha_i y_i=0 \qquad C=\alpha_i+\mu_i$$

将上面三式代入拉格朗日函数得到对偶问题：

$$\max_{\boldsymbol{\alpha}}\sum_{i=0}^{n-1}\alpha_i-\frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\alpha_i\alpha_j y_i y_j\boldsymbol{x}_i^T\boldsymbol{x}_j$$

$$\text{s.t.}\ \sum_{i=0}^{n-1}\alpha_i y_i=0$$

$$0\le\alpha_i\le C, i=0,1,\cdots,n-1$$

将软间隔的对偶问题与硬间隔比较，两者唯一的差别就在于对偶变量的约束不同：软间隔是$0\leq\alpha_i\leq C$，而硬间隔是$0\leq\alpha_i$。

对于软间隔支持向量机，KKT条件要求：

$$\begin{cases} \alpha_i \geq 0, \mu_i \geq 0 \\ y_i f(\boldsymbol{x}_i) - 1 + \xi_i \geq 0 \\ \alpha_i (y_i f(\boldsymbol{x}_i) - 1 + \xi_i) = 0 \\ \xi_i \geq 0, \mu_i \xi_i = 0 \end{cases}$$

对于任意训练样本（x_i,y_i），总有$\alpha_i=0$或$y_i f(x_i)=1-\xi_i$。

- 若$\alpha_i=0$，则该样本不会对f(x)有任何影响；
- 若$\alpha_i>0$，则必有$y_i f(x_i)=1-\xi_i$，即该样本是支持向量：又由于$C=\alpha_i+\mu_i$，若$\alpha_i<C$，则$\mu_i>0$，由于$\mu_i\xi_i=0$则$\xi_i=0$，即该样本恰好落在最大间隔边界上；若$\alpha_i=C$，则有$\mu_i=0$，此时若$\xi_i\leq 1$则该样本落在最大间隔内部，若$\xi_i>1$则该样本被错误分类。

$$\max_{\boldsymbol{\alpha}} \sum_{i=0}^{n-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

5. SMO算法

对偶问题是一个二次规划问题，可使用通用的二次规划算法来求解，但对于实际任务会有很大的计算开销，因此，人们根据问题本身的特性，开发了很多高效算法，应用最广泛的是SMO（sequential minimal optimization）算法。

SMO算法的核心思想是将一系列α的优化问题转化为一个个小的优化问题求解，即每一步选择两个α进行优化，其它α值暂时固定不变。之所以选择两个α，是因为要符合$\sum \alpha_i y_i = 0$的约束，无法只选择一个α进行优化。然后对两个α的二次优化问题可以得到其解析解，从而加速整个优化过程。

不失一般性，假设选出α_0和α_1来进行优化，为了方便，我们暂时以$K_{ij}=x_i^T x_j$表示两个向量x_i与x_j的内积。将对偶问题写为极小值问题，并展开为与α_0、α_1相关的部分和无关的部分（在本次优化中可视为常数，用c_1表示）：

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\alpha_i\alpha_j y_i y_j K_{ij} - \sum_{i=0}^{n-1}\alpha_i$$

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}K_{00}\alpha_0^2 + \frac{1}{2}K_{11}\alpha_1^2 + y_0 y_1 K_{01}\alpha_0\alpha_1$$

$$+y_0\alpha_0\sum_{i=2}^{n-1}\alpha_i y_i K_{0i} + y_1\alpha_1\sum_{i=2}^{n-1}\alpha_i y_i K_{1i} - (\alpha_0+\alpha_1) + c_1$$

利用约束条件可消去一个α：

$$y_0\alpha_0 + y_1\alpha_1 = c_2 \quad \Longrightarrow \quad \alpha_0 = y_0(c_2 - y_1\alpha_1)$$

其中c_2为常数，在本次优化中固定，另外用到了$y_i^2=1$。

将α_0的表达式代入优化问题消去α_0得到单变量优化问题：

$$\min \Psi(\alpha_1) = \frac{1}{2}K_{00}(c_2 - y_1\alpha_1)^2 + \frac{1}{2}K_{11}\alpha_1^2 + y_1K_{01}\alpha_1(c_2 - y_1\alpha_1)$$

$$+(c_2 - y_1\alpha_1)\sum_{i=2}^{n-1}\alpha_i y_i K_{0i} + y_1\alpha_1\sum_{i=2}^{n-1}\alpha_i y_i K_{1i} - y_0(c_2 - y_1\alpha_1) - \alpha_1 + c_1$$

令$d\Psi/(d\alpha_1)=0$，得到：

$$(K_{00} + K_{11} - 2K_{01})\alpha_1 = y_1K_{00}c_2 - y_1K_{01}c_2 + y_1\sum_{i=2}^{n-1}\alpha_i y_i K_{0i} - y_1\sum_{i=2}^{n-1}\alpha_i y_i K_{1i} - y_0y_1 + 1$$

这样就得到了α_1的更新公式，其右侧可通过当前值（old）来计算：

$$c_2 = y_0\alpha_0^{\text{old}} + y_1\alpha_1^{\text{old}}$$

$$f(\boldsymbol{x}_0) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_0 + b = \sum_{i=0}^{n-1} \alpha_i y_i \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_0 + b = \sum_{i=0}^{n-1} \alpha_i y_i K_{0i} + b \quad \xrightarrow{\text{yields}}$$

$$\sum_{i=2}^{n-1} \alpha_i y_i K_{0i} = f(\boldsymbol{x}_0) - \alpha_0^{\mathrm{old}} y_0 K_{00} - \alpha_1^{\mathrm{old}} y_1 K_{01} - b$$

$$f(\boldsymbol{x}_1) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_1 + b = \sum_{i=0}^{n-1} \alpha_i y_i \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_1 + b = \sum_{i=0}^{n-1} \alpha_i y_i K_{1i} + b \quad \xrightarrow{\text{yields}}$$

$$\sum_{i=2}^{n-1} \alpha_i y_i K_{1i} = f(\boldsymbol{x}_1) - \alpha_0^{\mathrm{old}} y_0 K_{01} - \alpha_1^{\mathrm{old}} y_1 K_{11} - b$$

将c_2、$\sum\alpha_i y_i K_{0i}$、$\sum\alpha_i y_i K_{1i}$表达式代入迭代式右端得到：

$$(K_{00} + K_{11} - 2K_{01})\alpha_1^{\mathrm{new}} = (K_{00} + K_{11} - 2K_{01})\alpha_1^{\mathrm{old}} + y_1[(f(\boldsymbol{x}_0) - y_0) - (f(\boldsymbol{x}_1) - y_1)]$$

令：　$E_i = f(\boldsymbol{x}_i) - y_i$　　$\eta = K_{00} + K_{11} - 2K_{01}$

则：

$$\alpha_1^{\mathrm{new}} = \alpha_1^{\mathrm{old}} + \frac{y_1(E_0 - E_1)}{\eta}$$

利用上式计算得到α_1^{new}还需要检验它是否满足约束条件，因为它应该介于0到C之间，但对于某一次迭代中选出的特定的α_0和α_1，还可以更精确地限定其取值范围。首先，新产生的α_0^{new}和α_1^{new}应该处于图7-6中[0,C]×[0,C]所限定的“盒子”内。

如果$y_0 \neq y_1$（图7.6a），则有：$\alpha_0 - \alpha_1 = k$ (k为常数)

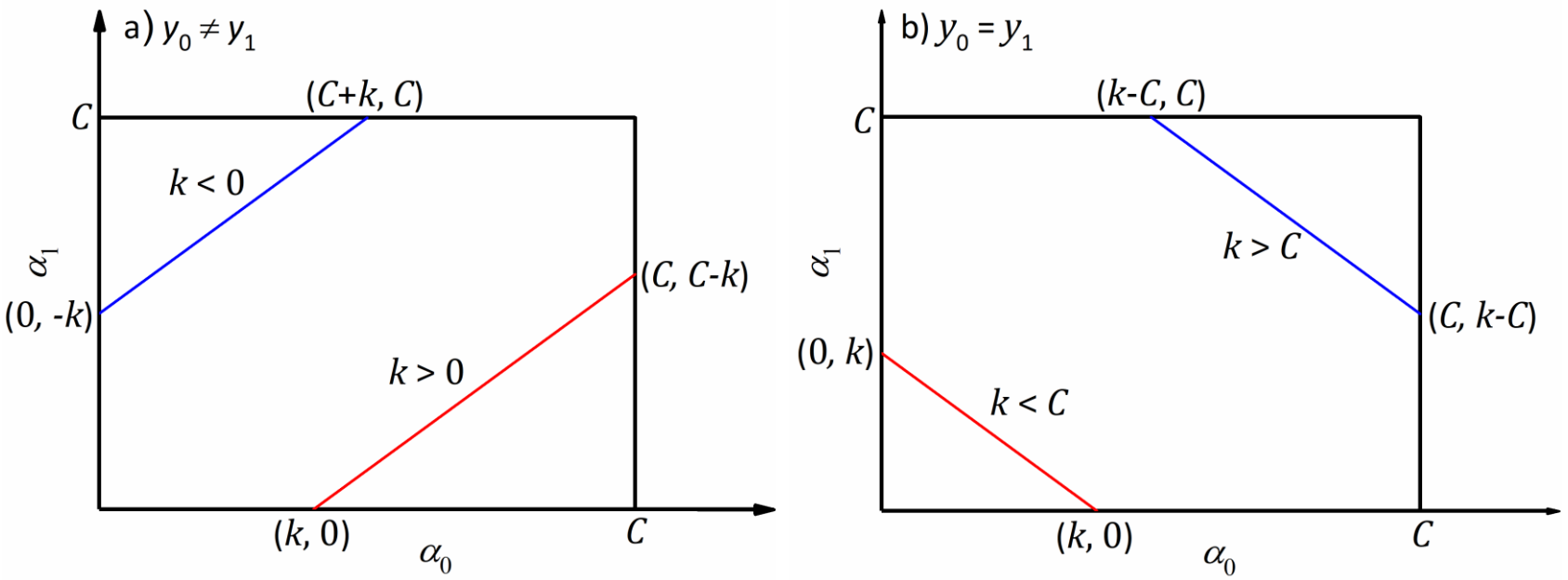
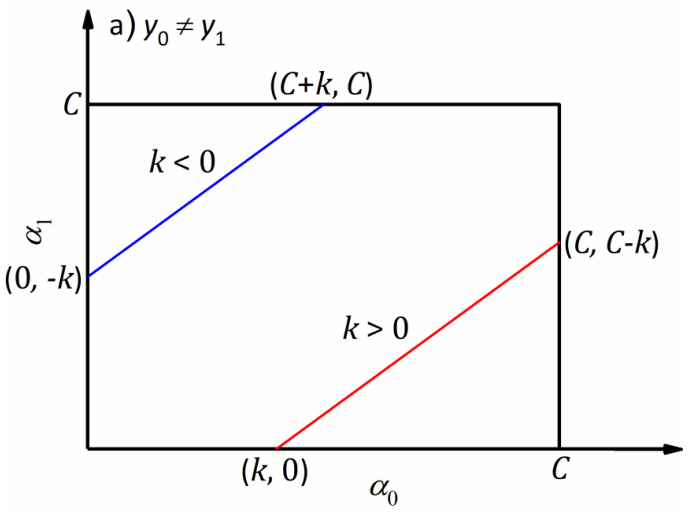


图7-6 拉格朗日乘子更新时的可行域

$$\alpha_0 - \alpha_1 = k \Longrightarrow \alpha_1 = \alpha_0 - k$$

上式为一直线方程，当k>0时该直线位于对角线下侧，如图7-6a中红线所示，该直线与“盒子”的交点在(k, 0)和(C,C-k)，显然α_1只能在0到C-k之间变化。



由于当前值(α_0^{old},α_1^{old})也在这条线上，即α_0^{old}-α_1^{old}=k，得到α_1的上下限分别为：

$$\mathrm{L} = 0, \mathrm{H} = C - k = C + \alpha_1^{\mathrm{old}} - \alpha_0^{\mathrm{old}}$$

当k<0时该直线位于对角线上侧，如图7-6a中蓝线所示，该直线与“盒子”的交点在 (0,-k)和(C+k,C)，此时α_1介于-k到C之间变化，α_1的上下限分别为：

$$\mathrm{L} = -k = \alpha_1^{\mathrm{old}} - \alpha_0^{\mathrm{old}}, \mathrm{H} = C$$

$$\mathrm{L}=0,\mathrm{H}=C-k=C+\alpha_1^{\mathrm{old}}-\alpha_0^{\mathrm{old}}$$
$$\mathrm{L}=-k=\alpha_1^{\mathrm{old}}-\alpha_0^{\mathrm{old}},\mathrm{H}=C$$

综合两种情况，得到$y_0 \neq y_1$时α_1的上下限：

$$\begin{cases}\mathrm{L}=\max\left(0,\alpha_1^{\mathrm{old}}-\alpha_0^{\mathrm{old}}\right)\\ \mathrm{H}=\min\left(C,C+\alpha_1^{\mathrm{old}}-\alpha_0^{\mathrm{old}}\right)\end{cases}$$

类似地可以得到$y_0=y_1$时（参见图7-6b）α_1的上下限：

$$\begin{cases}\mathrm{L}=\max\left(0,\alpha_0^{\mathrm{old}}+\alpha_1^{\mathrm{old}}-C\right)\\ \mathrm{H}=\min\left(C,\alpha_0^{\mathrm{old}}+\alpha_1^{\mathrm{old}}\right)\end{cases}$$

因此，计算得到的α_1^{new}需要根据其取值范围进行修剪，得到本轮优化的结果：

$$\begin{cases}\alpha_1^{\mathrm{new,clip}}=\mathrm{H},\text{while }\alpha_1^{\mathrm{new}}\geq\mathrm{H}\\ \alpha_1^{\mathrm{new,clip}}=\mathrm{L},\text{while }\alpha_1^{\mathrm{new}}\leq\mathrm{L}\\ \alpha_1^{\mathrm{new,clip}}=\alpha_1^{\mathrm{new}},\text{while L}<\alpha_1^{\mathrm{new}}<H\end{cases}$$

然后根据$\alpha_1^{new,clip}$再计算α_0^{new}：

$$y_0\alpha_0^{\text{new}} + y_1\alpha_1^{\text{new,clip}} = y_0\alpha_0^{\text{old}} + y_1\alpha_1^{\text{old}} \Longrightarrow$$

$$\alpha_0^{\text{new}} = \alpha_0^{\text{old}} + y_0y_1\left(\alpha_1^{\text{old}} - \alpha_1^{\text{new,clip}}\right)$$

在每次完成两个α的优化之后，需要重新计算位移项b。当$0<\alpha_0^{new}<C$时，则样本x_0为支持向量且落在分隔边界上，满足$y_0f(x_0)-1=0$，即：

$$y_0 = f(\boldsymbol{x}_0) = \sum_{i=0}^{n-1}\alpha_i y_i \boldsymbol{x}_i\boldsymbol{x}_0 + b = \sum_{i=0}^{n-1}\alpha_i y_i K_{0i} + b \Longrightarrow$$

$$b_0^{\text{new}} = y_0 - \sum_{i=0}^{n-1}\alpha_i y_i K_{0i} = y_0 - \sum_{i=2}^{n-1}\alpha_i y_i K_{0i} - \alpha_0^{\text{new}} y_0 K_{00} - \alpha_1^{\text{new,clip}} y_1 K_{01}$$

上式的加和项只含有未改变的，可由当前值得出：

$$E_0 = f(\boldsymbol{x}_0) - y_0 = \sum_{i=2}^{n-1} \alpha_i y_i K_{0i} + \alpha_0^{\text{old}} y_0 K_{00} + \alpha_1^{\text{old}} y_1 K_{01} + b^{\text{old}} - y_0$$

$$b_0^{\text{new}} = y_0 - \sum_{i=0}^{n-1} \alpha_i y_i K_{0i} = y_0 - \sum_{i=2}^{n-1} \alpha_i y_i K_{0i} - \alpha_0^{\text{new}} y_0 K_{00} - \alpha_1^{\text{new,clip}} y_1 K_{01}$$

上面两式相加得到b_0^{new}的计算式：

$$b_0^{\text{new}} = -E_0 - y_0 K_{00}\left(\alpha_0^{\text{new}} - \alpha_0^{\text{old}}\right) - y_1 K_{01}\left(\alpha_1^{\text{new,clip}} - \alpha_1^{\text{old}}\right) + b^{\text{old}}$$

类似地，如果$0<\alpha_1^{\text{new,clip}}<C$，得到位移项更新公式，记为$b_1^{\text{new}}$：

$$b_1^{\text{new}} = -E_1 - y_0 K_{01}\left(\alpha_0^{\text{new}} - \alpha_0^{\text{old}}\right) - y_1 K_{11}\left(\alpha_1^{\text{new,clip}} - \alpha_1^{\text{old}}\right) + b^{\text{old}}$$

如果$0<\alpha_0^{\text{new}}, \alpha_1^{\text{new,clip}}<C$，即两个拉格朗日乘子都落在边界上，这时$b_0^{\text{new}}$与$b_1^{\text{new}}$是一致的，取前者即可。

如果α_0^{new}和$\alpha_1^{new,clip}$都没有落在(0,C)区间，仍然按上面两式计算b_0^{new}和b_1^{new}，取二者的均值作为新的位移项。因此位移项更新规则为：

$$b^{\text{new}} = \begin{cases} b_0^{\text{new}}, \text{while } 0 < \alpha_0^{\text{new}} < C \\ b_1^{\text{new}}, \text{while } 0 < \alpha_1^{\text{new,clip}} < C \\ \dfrac{b_0^{\text{new}} + b_1^{\text{new}}}{2}, \text{otherwise} \end{cases}$$

最后，还有一个问题需要解决，即如何选择要优化的两个拉格朗日乘子。在SMO算法中利用一个外层循环选择第一个要优化的拉格朗日乘子（比如α_1），在内层循环中选择第二个拉格朗日乘子（比如α_0）并完成优化。外层循环中，先遍历所有数据，如果某一个α违反KKT条件则将它选为第一个待优化的拉格朗日乘子。

遍历一次以后，外循环中会继续遍历那些非边界处的α即0<α<C，这是由于边界处的α（α=0或α=C）在优化中的改动通常不大，选择非边界处α则可加速收敛过程。经过一系列的循环，当所有非边界α都满足KKT条件后，再对整个数据集进行遍历。这样在单次遍历整个数据集及多次遍历非边界α之间进行切换，当所有α都满足KKT条件后终止程序。

判断某个α_i是否满足KKT条件的标准包括：

（1）如果$\alpha_i<C$，则对应样本为正确分类点且不会落在间隔内部，因而$y_i f(x_i)-1\geq 0$，考虑到允许误差tol，如果$y_i E_i=y_i f(x_i)-1<-tol$则视为违反KKT条件；

（2）如果$\alpha_i>0$，则对应样本为支持向量，$y_i f(x_i)-1=0$，考虑到允许误差，如果$y_i E_i=y_i f(x_i)-1>tol$，则视为违反KKT条件。

依据外循环中选出的第一个拉格朗日乘子α_1，在内层循环中进行第二个拉格朗日乘子α_0的选择，选择原则是尽量让优化中产生的变化更大，从而“大步”走向最优点。根据更新式：

$$\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + \frac{y_1(E_0 - E_1)}{\eta}$$

可以看出，优化后拉格朗日乘子的变化正比于两个样本的误差，即$|E_0-E_1|$，因此，应选择与E_1距离最大的E所对应的α，如果$E_1>0$则选择最小的E为E_0，如果$E_1<0$则选择最大的E为E_0，相应地可以确定α_0。

6. SMO算法python实现

下面利用python实现SMO算法，测试数据由scikit-learn中datasets.samples_generator模块中的make_blobs函数产生。

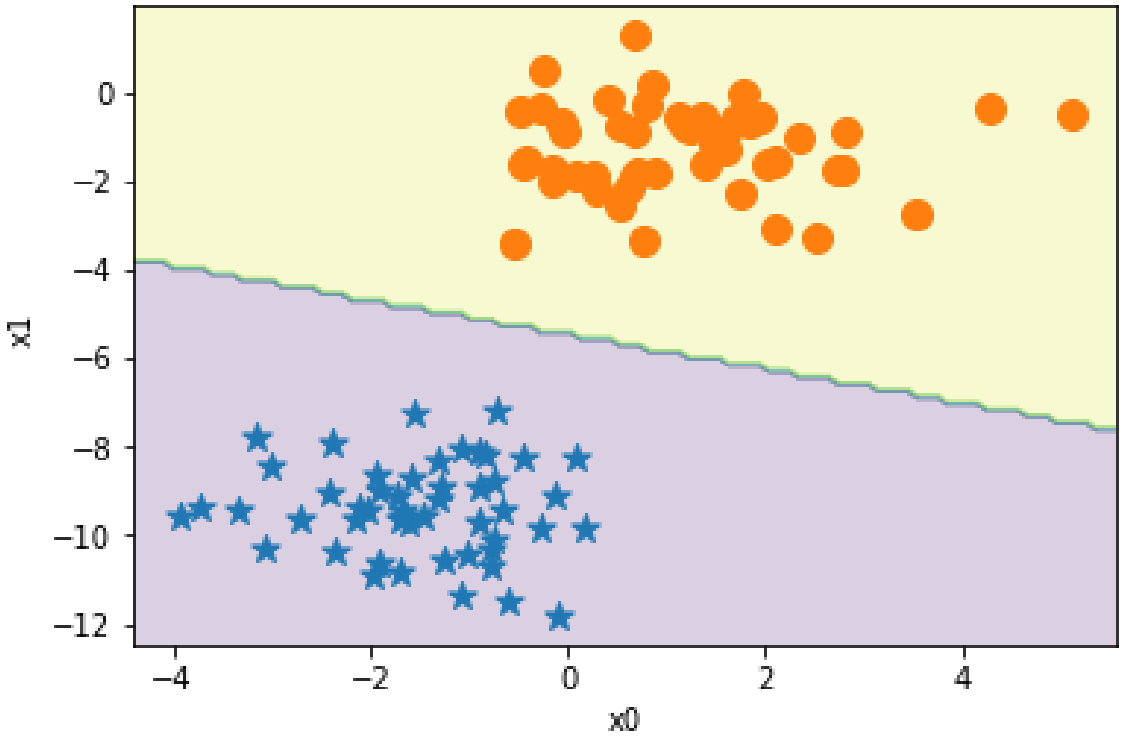


图7-7 SMO算法产生的分隔线

7. 核函数

前面的讨论都假设数据是线性可分的，然而在现实任务中也许并不存在一个能正确划分两类样本的超平面，如图7-8所示的两类数据，该数据由scikit-learn中的make_moons函数产生。

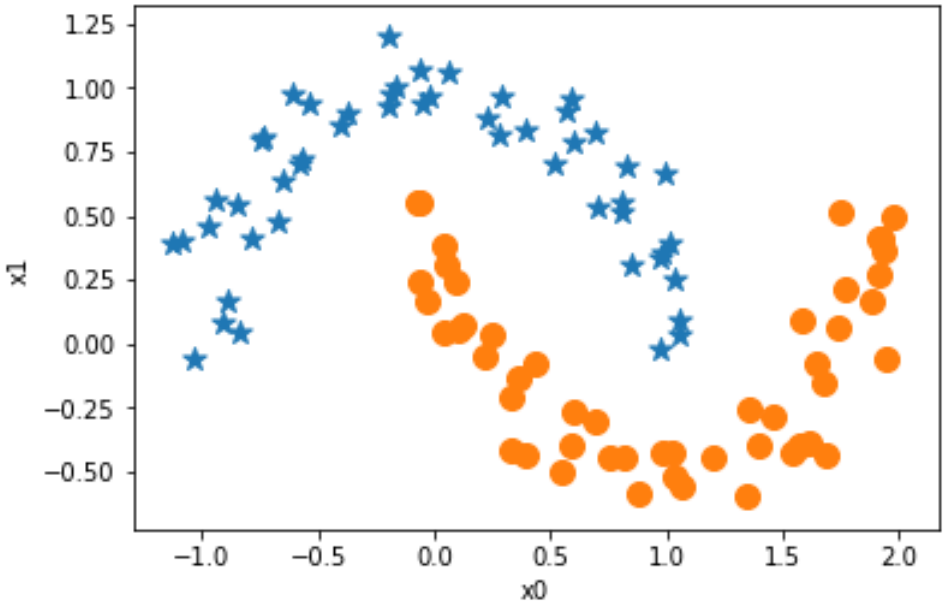


图7-8 用make_moons产生的数据集

图7-8中的数据集，人眼很容易进行划分，只需要一条曲线就可以将所有样本正确分割。对这样的问题，可将样本从原始空间映射到一个更高维的空间，使得样本在新空间中线性可分。

幸运的是，如果原始样本空间是有限维，即特征数目有限，那么一定存在一个高维空间使样本可分。

令ϕ(x)表示将x映射后的特征向量，于是，在新的特征空间中划分超平面所对应的模型可以表示为：

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b$$

与前面的推导过程类似地，也可以得到其对偶问题为：

$$\max_{\boldsymbol{\alpha}} \sum_{i=0}^{n-1} \alpha_i - \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}_j)$$

$$\text{s.t.} \sum_{i=0}^{n-1} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, i = 0,1,\cdots,n-1$$

上式与原始空间的优化问题几乎相同，唯一的区别是将x_i与x_j在原始空间中的内积$x_i^T x_j$改变为映射空间中新特征向量的内积$\phi(x_i)^T\phi(x_j)$。由于新的特征空间维数可能很高，甚至是无穷维的，因此直接计算$\phi(x_i)^T\phi(x_j)$通常是困难的。为了避开这个障碍，人们采用核函数（kernel function）。

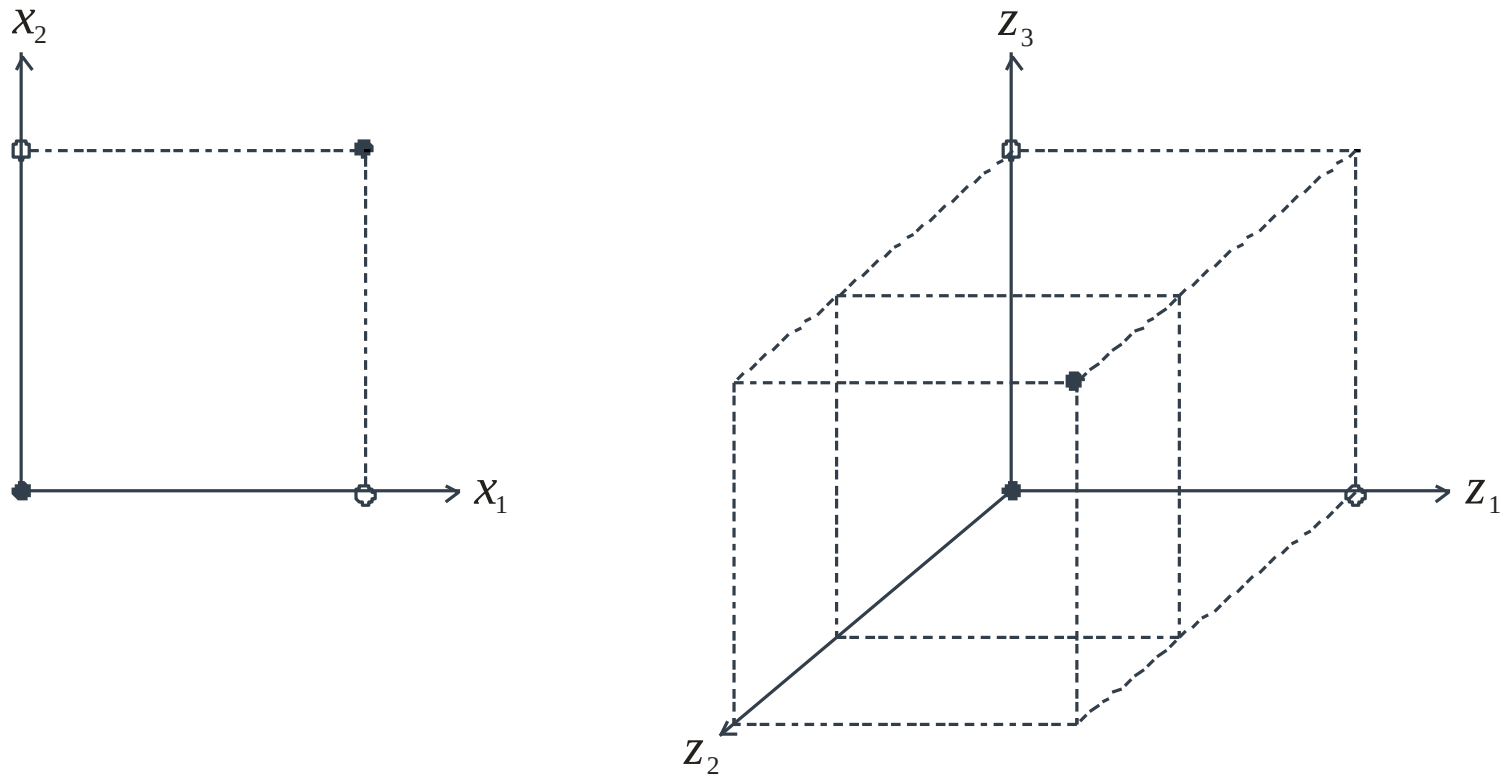
以二维空间中的异或问题为例，由于它在二维空间无法线性分隔，可以将其映射到更高维空间，比如三维空间，作如下映射：

$$\boldsymbol{x} = (x_1, x_2) \Longrightarrow \varphi(\boldsymbol{x}) = \boldsymbol{z} = (z_1, z_2, z_3) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

(0, 0) ⟹ (0, 0, 0), (0, 1) ⟹ (0, 0, 1),

(1, 0) ⟹ (1, 0, 0), (1, 1) ⟹ (1, 1.4, 1)

$$(z_1, z_2, z_3) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$



在新特征空间中，易于实现线性分割。比如(0.5, 0, 0), (0, 0, 0.5), (1, 1, 1)这三个点构成的平面即可将异或问题正确分类。

在支持向量机的学习过程，会大量计算两个向量的内积，在新的高维特征空间中的内积最终都可以用原始空间的坐标来计算，比如上面的映射：

$$(z_1, z_2, z_3) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

$$\boldsymbol{z} \cdot \boldsymbol{z}' = z_1z_1' + z_2z_2' + z_3z_3' = x_1^2x_1'^2 + 2x_1x_1'x_2x_2' + x_2^2x_2'^2$$

$$= (x_1x_1' + x_2x_2')^2 = (\boldsymbol{x} \cdot \boldsymbol{x}')^2 = K(\boldsymbol{x}, \boldsymbol{x}')$$

其中的K即为核函数，它是原始样本空间中的一个函数，它以两个样本（即向量）作为输入，其结果就是这两个样本映射后新特征向量的内积，即：

$$K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi\left(\boldsymbol{x}_j\right)$$

这样就把高维空间中两个向量的内积转换为原始样本空间中的一个函数，一旦有了这样的函数，就避免了在高维空间中进行计算。

利用核函数，优化问题转化为：

$$\max_{\boldsymbol{\alpha}} \sum_{i=0}^{n-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

求解后得到：

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}) + b = \sum_{i=0}^{n-1} \alpha_i y_i \phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}) + b = \sum_{i=0}^{n-1} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

显然，如果已知映射ϕ则容易写出其对应的核函数，但在现实任务中，构造合适的映射ϕ往往是很困难的，那如何得到其核函数呢？

我们有下面的定理：

核函数定理：令χ为输入空间，K(·,·)是定义在χ×χ上的对称函数，K是核函数的充分必要条件是对于任意数据集$D=\{x_0,x_1,\cdots,x_{n-1}\}$，核矩阵（kernel matrix）**K**总是半正定的：

$$\boldsymbol{K}=\begin{bmatrix} K(\boldsymbol{x}_0,\boldsymbol{x}_0) & \cdots & K(\boldsymbol{x}_0,\boldsymbol{x}_{n-1}) \\ \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_{n-1},\boldsymbol{x}_0) & \cdots & K(\boldsymbol{x}_{n-1},\boldsymbol{x}_{n-1}) \end{bmatrix}$$

核函数定理表明，只要一个对称函数所对应的核矩阵是半正定的，它就能作为核函数使用，或者说，就一定存在一个与它对应的映射ϕ，因此，每一个核函数都隐式地定义了一个映射。既然如此，只要找到合适的核函数，也就不必关心相应的映射是什么样子了。但是，我们建立映射的目的是为了让样本在新的特征空间中线性可分，既然不知道映射的具体形式，也就无法确切知道映射后是否是线性可分的。

通常的做法是可以比较几个核函数，看哪一个性能最佳。因而，核函数的选择成为支持向量机的一大变数，若核函数选择不合适，必然导致分类器的性能不佳。

表7-1 常用核函数

名称	表达式	参数
线性核	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$	
多项式核	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j)^d$	d≥1为多项式次数, d=1即为线性核
高斯核	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$	σ>0高斯核带宽
拉普拉斯核	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma}\right)$	σ>0
Sigmoid核	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + \theta)$	β>0, θ<0

在非线性核中，应用最多的是高斯核函数，也称为径向基函数（Radial basis function，简称RBF）：

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right)$$

其中 $\gamma=1/(2\sigma^2)$，是一个由用户设定的参数，它与方差成反比。

利用 python 实现RBF支持向量机分类图7-8的数据集。程序输出结果如图7-9所示。

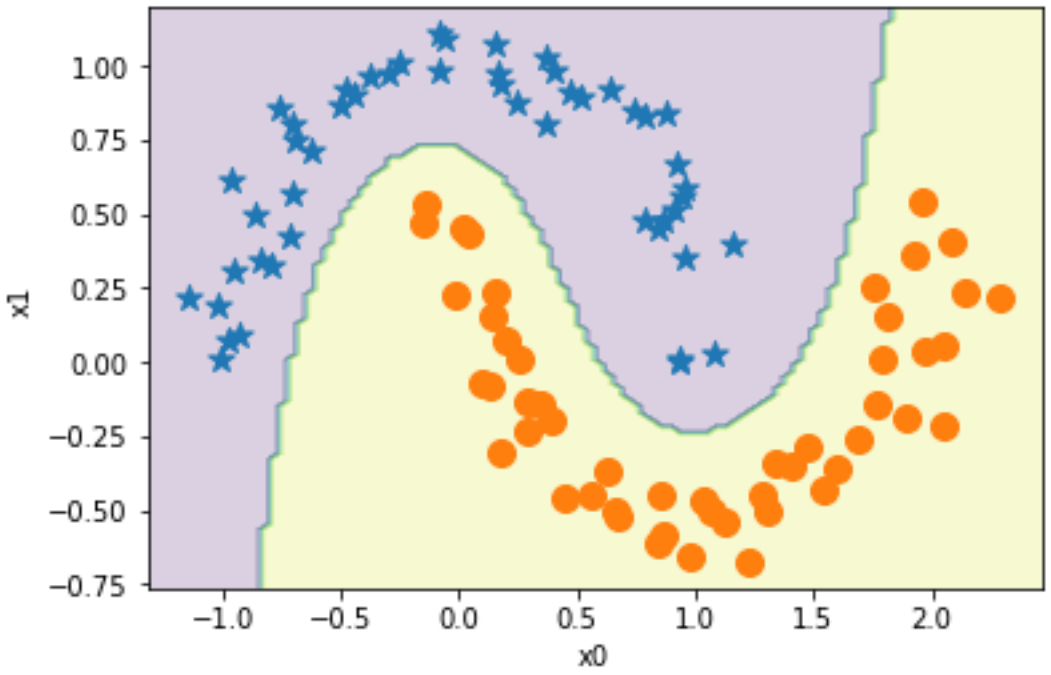


图7-9 RBF支持向量机产生的分隔线

8. 利用scikit-learn实现支持向量机

在scikit-learn中的svm模块实现了多个支持向量机相关的类：

- LinearSVC用于线性支持向量分类，
- LinearSVR用于线性支持向量回归，
- NuSVC用于核函数支持向量分类，
- NuSVR用于核函数支持向量回归。

LinearSVC的用法如下：

LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)

利用LinearSVC类对图7-7中的线性分类问题进行求解。

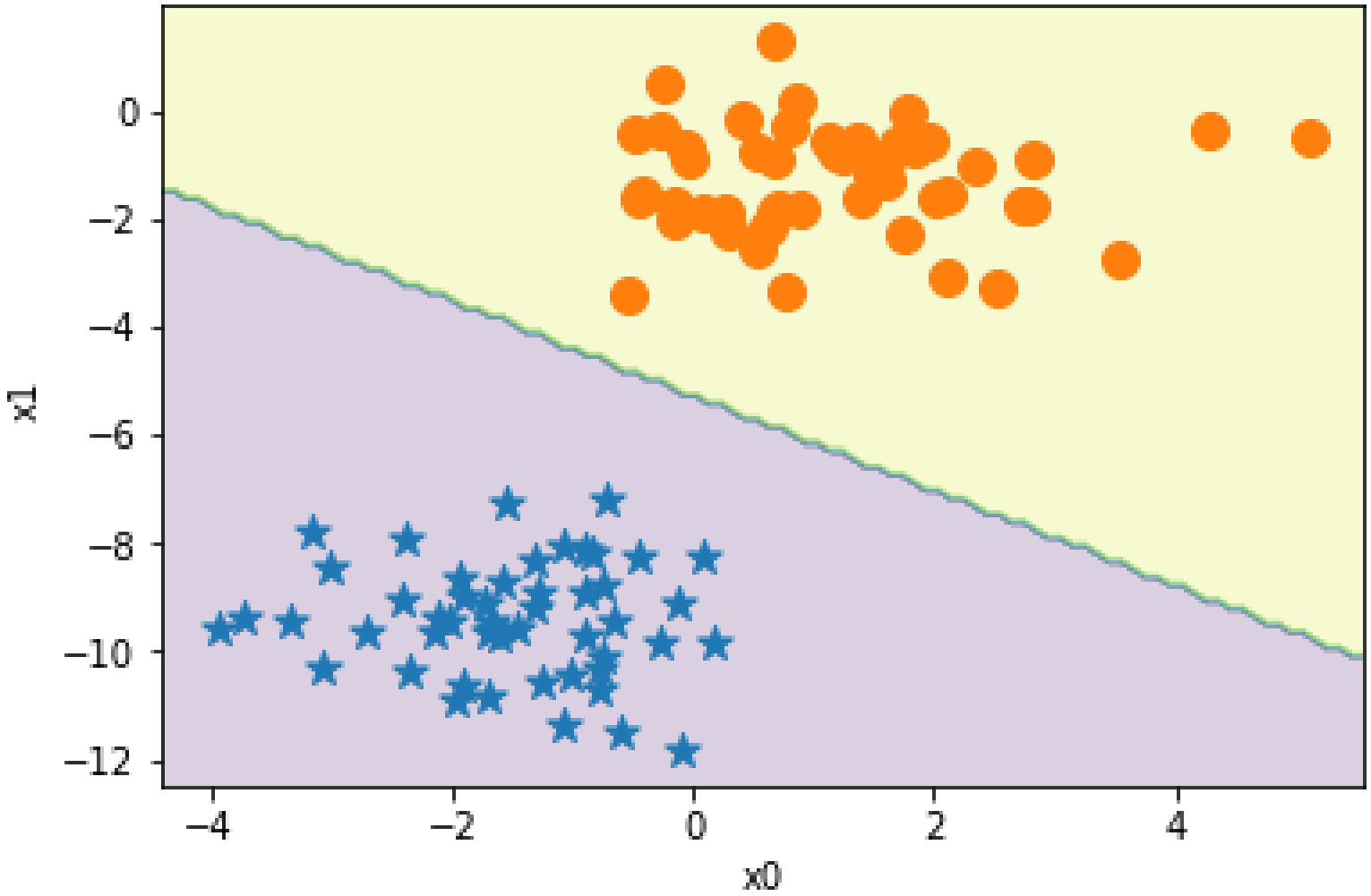


图7-10 利用LinearSVC分类的分隔线

NuSVC类用法如下：

NuSVC(nu=0.5, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

利用NuSVC类来对make_moons产生的数据进行分类。

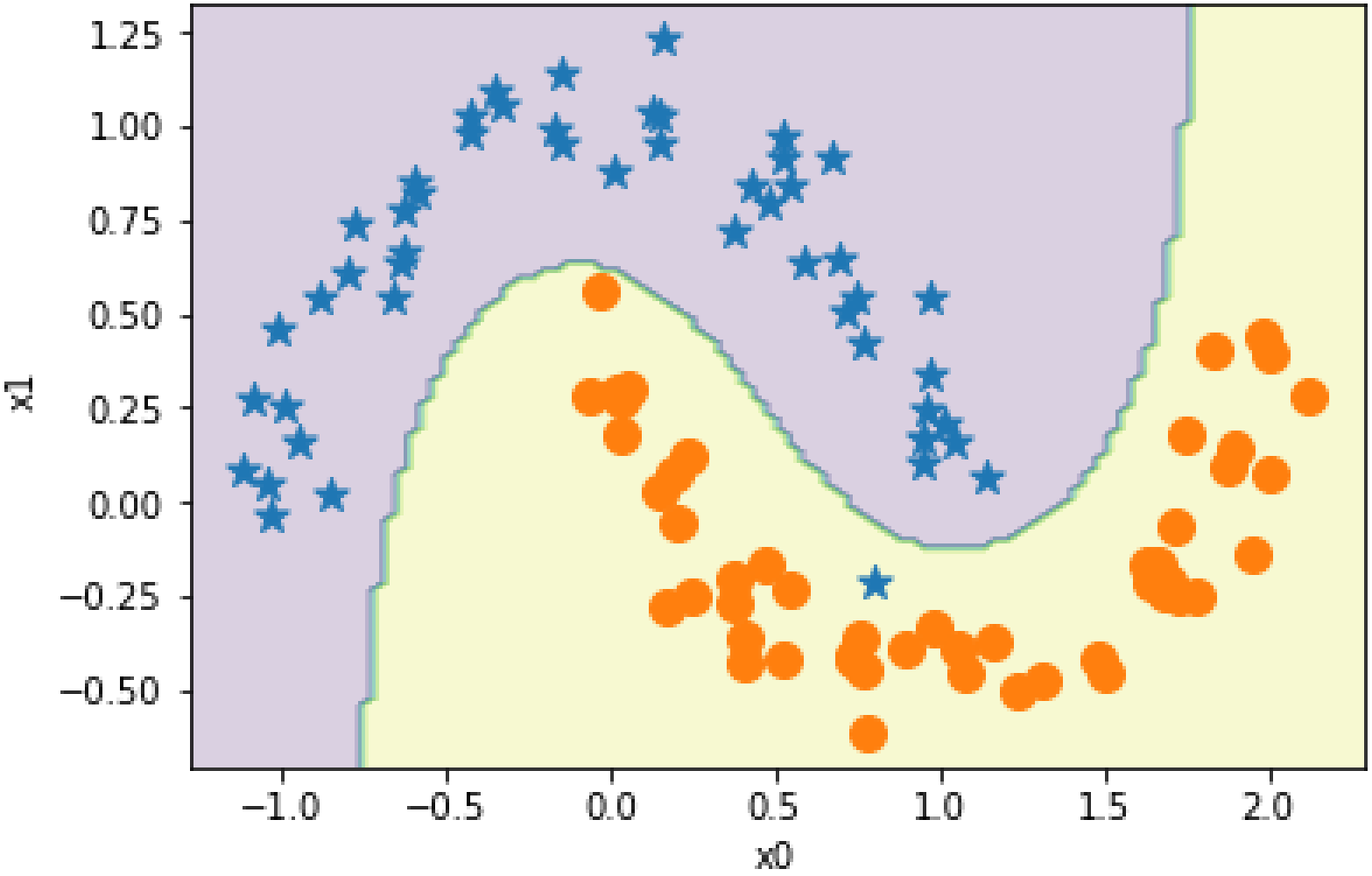


图7-11 利用NuSVC类对make_moons数据分类结果