《机器学习》课程系列

支持向量机*

武汉纺织大学数学与计算机学院

杜小勤

2020/05/24

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 6, 2020。

1 线性可分支持向量机

在二分类问题中，一般地，当训练数据集线性可分[1]时，存在无穷多个分离超平面，将两类样本点正确地分开。

在前面的章节中，我们已经介绍了感知机模型，它是误分类样本驱动的分类模型。当一个样本点被误分类时，通过梯度下降方法，调整权值参数 $\boldsymbol{w}$ 和偏置参数 b，使得分离超平面向误分类点一侧移动，以减少该误分类点与分离超平面的距离，直至分离超平面越过该误分类点使其被正确分类。因此，只要感知机模型能够对可分离数据集中的所有样本点进行正确的分离，那么算法就即刻终止，这意味着感知机模型学习得到的分离超平面不是唯一的 (与分离超平面的初始方位有关)，因而也不一定是最优的 (属于没有约束条件的最优化)。

为了解决上述问题，需要对学习算法施加某些约束，使其得到的结果是唯一且最优的。支持向量机 (Support Vector Machine, SVM) 就是这样一种重要的分类模型。

首先，考虑简单的二分类情形：训练数据集是线性可分的。此时，相应的支持向量机被称为“线性可分支持向量机”。

定义 1.1（线性可分支持向量机）对于给定的线性可分训练数据集，使用最大间隔法得到的分离超平面[2]：

$$\boldsymbol{x}^T\boldsymbol{w}^* + b^* = 0 \tag{1}$$

及其相应的分类决策函数：

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{x}^T\boldsymbol{w}^* + b^*) \tag{2}$$

它们被称为线性可分支持向量机。

[1] 关于数据集的线性可分性，请阅读“判别函数的线性分类基础”一章。

[2] 注意，$\boldsymbol{x}^T\boldsymbol{w} = \boldsymbol{w}^T\boldsymbol{x}$。此处，仅为了方便使用上标，对 $\boldsymbol{x}$ 与 $\boldsymbol{w}$ 进行了交换。

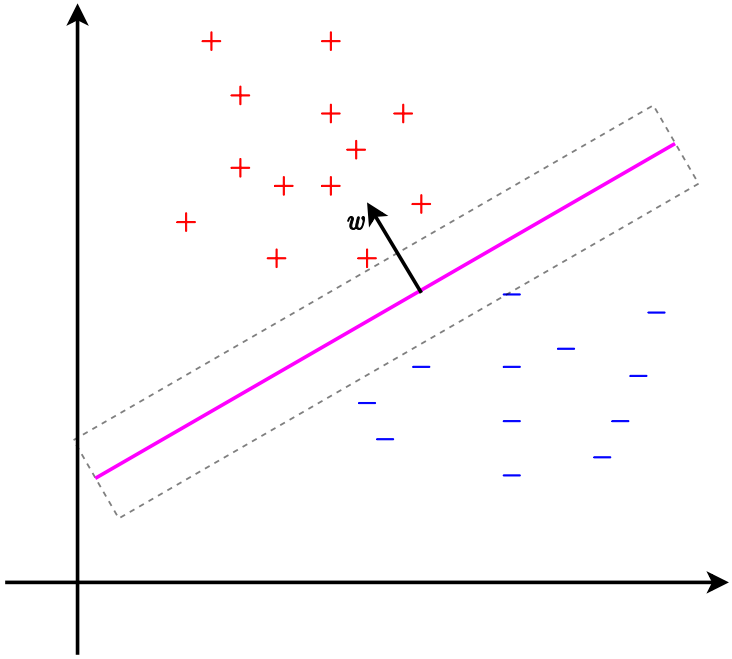


图 1-1: 线性可分支持向量机的二分类示意图

设给定的训练数据集为 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$，其中，$\boldsymbol{x}_i\in\mathbb{R}^d$，$y_i\in\{+1,-1\}$，$i=1,2,\ldots,N$，$N$ 为数据集中样本点的个数。一般地，将 $y_i=+1$ 的点称为正例，将 $y_i=-1$ 的点称为负例。

图1-1展示了二维特征空间中线性可分支持向量机的二分类示意图。

从图中可以看出，紫色直线为分离超平面，向量 $\boldsymbol{w}$ 所指向的一侧为正例点，另一侧为负例点[3]。平行虚线表示与分离超平面相对应的 2 个间隔超平面，它们都被边界上的正例点与负例点唯一地确定，且它们之间具有最大的间隔；分离超平面位于 2 个间隔超平面的正中间对称位置上。因此，该分离超平面满足唯一性与最优性。

以上，我们建立了对线性可分支持向量机的感性认识。在下文中，首先，我们将详细描述它的约束最优化求解过程。随后，对其进行适当的扩展，以处理数据集的线性不可分及非线性情形。

[3]一般情况下，在 2D 和 3D 空间中，直线方程为 $ax+by+c=0$，平面方程为 $ax+by+cz+d=0$。直线与平面的方向分别由向量 $\boldsymbol{w}=(a,b)$ 和向量 $\boldsymbol{w}=(a,b,c)$ 确定。当向量为单位向量时，即 $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$，才被称为法向量。直线方程和平面方程都可以法向量化。例如，直线方程法向量化为：$\frac{ax+by+c}{\sqrt{a^2+b^2}}=0$。此时，法向量为 $\frac{(a,b)}{\sqrt{a^2+b^2}}$，截距为 $\frac{c}{\sqrt{a^2+b^2}}$。平面方程情形，依此类推。

1.1 几何间隔与函数间隔

在“判别函数的线性分类基础”一章中，我们引入了任意点 $\boldsymbol{x}_i$ 到分离超平面或决策面 $\boldsymbol{w}^T\boldsymbol{x}+b=0$ 的带符号距离：

$$l_i = \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \tag{3}$$

如果结合考虑 y 的取值 $\{+1,-1\}$ 特点以及正负例与向量 $\boldsymbol{w}$ 的关系[4]，那么如下公式：

$$y_i \cdot l_i = y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \tag{4}$$

将能够提供以下的有效信息：

- 分类的正确性

  显然，如果 $y_i \cdot l_i > 0$，那么实例点 $\boldsymbol{x}_i$ 被正确分类；否则，如果 $y_i \cdot l_i < 0$，则实例点 $\boldsymbol{x}_i$ 被误分类。

- 分类的确信度与敏感度

  在分类正确的情况下，$y_i \cdot l_i$ 越大，表明实例点 $\boldsymbol{x}_i$ 离分离超平面越远，对 $\boldsymbol{x}_i$ 分类的确信度就越高。

  在误分类的情况下，$y_i \cdot l_i$ 越大，表明实例点 $\boldsymbol{x}_i$ 离分离超平面越近，$\boldsymbol{x}_i$ 对分离超平面就越敏感。

下面，给出几何间隔的定义。

定义 1.2（几何间隔）对于给定的训练数据集 T 和分离超平面 $\boldsymbol{w}^T\boldsymbol{x}+b$，定义任意实例点 $\boldsymbol{x}_i$ 的几何间隔(*Geometric Margin*) 为：

$$\gamma_i = y_i \cdot l_i = y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \tag{5}$$

定义训练数据集 T 的几何间隔为：

$$\gamma = \min_{i=1\ldots N} \gamma_i \tag{6}$$

[4]即规定向量 $\boldsymbol{w}$ 所指向的一侧为正例点 (取值 $+1$)，另一侧为负例点 (取值 -1)。这种取值惯例，也符合我们的常识，原因在于：如果点 $\boldsymbol{x}_i$ 在法向量 $\boldsymbol{w}$ 指向的一侧，那么 $\boldsymbol{w}^T\boldsymbol{x}_i + b > 0$；否则，$\boldsymbol{w}^T\boldsymbol{x}_i + b < 0$。

在一些情况下，我们并不关心几何间隔的真实大小，为简化计算，可以考虑将 $\|\boldsymbol{w}\|$ 从公式中去掉[5]，从而得到了函数间隔。

定义 **1.3**（函数间隔）对于给定的训练数据集 T 和分离超平面 $\boldsymbol{w}^T\boldsymbol{x}+b$，定义任意实例点 $\boldsymbol{x}_i$ 的函数间隔(*Functional Margin*)为：

$$\hat{\gamma}_i = y_i \cdot \left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \tag{7}$$

定义训练数据集 T 的函数间隔为：

$$\hat{\gamma} = \min_{i=1\ldots N} \hat{\gamma}_i \tag{8}$$

几何间隔与函数间隔，都可以提供关于实例点的有效信息，即分类的正确性、分类的确信度与敏感度。但是，几何间隔具有绝对的可比性，而函数间隔只具有相对的可比性[6]。原因在于，如果给 $\boldsymbol{w}$ 和 b 分别添上相同的缩放因子 $s(\neq 0)$，那么分离超平面本身并不会发生任何变化：$s\left(\boldsymbol{w}^T\boldsymbol{x}_i+b=0\right)$ 与 $\boldsymbol{w}^T\boldsymbol{x}_i+b=0$ 等价，因而，几何间隔也将维持不变；然而，函数间隔将随缩放因子 s 变化而变化，且为原函数间隔的 s 倍。根据上述定义，几何间隔与函数间隔具有如下关系：

$$\begin{aligned} \gamma_i &= \frac{\hat{\gamma}_i}{\|\boldsymbol{w}\|} \\ \gamma &= \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \end{aligned} \tag{9}$$

1.2 学习策略：硬间隔最大化

对于线性可分数据集，支持向量机的学习策略是，求解能够正确划分数据集且确保几何间隔最大的分离超平面。由于假设给定的数据集是 (严格) 线性可分的，因而此处的“间隔”被称为“硬间隔”[7]。

为什么要确保硬间隔最大化，其意义何在？这与前面介绍的“分类确信度”概念相关。假设支持向量机已经能够对所有的样本实例点进行正确的分类，那么，

[5]将 $\|\boldsymbol{w}\|$ 去掉，相当于对权值向量 $\boldsymbol{w}$ 进行缩放或去单位化，因而并不会改变分离超平面。原因是，就分离超平面而言，$\frac{\boldsymbol{w}^T\boldsymbol{x}_i+b}{\|\boldsymbol{w}\|}=0$ 与 $\boldsymbol{w}^T\boldsymbol{x}_i+b=0$，均表示同一个分离超平面。因此，分离超平面不会受到任何影响。

[6]在一些情况下，利用相对可比性，可以简化问题的求解。例如，在即将介绍的“硬间隔最大化”问题中，就利用了这一点，设置 $\hat{\gamma}=1$，从而达到了简化问题的目的。

[7]对于数据集为非严格线性可分情形，相应的“间隔”被称为“软间隔”。

显然，数据集的几何间隔或函数间隔越大[8]，实例点越远离分离超平面，被误分的可能性就越小，实例点分类的确信度就越大，分类也就越可靠。特别地，对于那些未知的新实例点，这种分类器也应该有更好的分类预测能力 (即泛化能力)。

下面，对硬间隔最大化问题进行形式化。这是一个带约束的最优化问题：

$$\begin{aligned} &\max_{\boldsymbol{w},b} \quad \gamma \\ &s.t. \quad y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \geqslant \gamma \quad i = 1, 2, \ldots, N \end{aligned} \tag{10}$$

其中，γ 是数据集的几何间隔。上式中，第 2 个表达式 (约束条件) 表达了每个实例点的几何间隔与数据集的几何间隔之间的关系：后者是所有实例点几何间隔的最小值；第 1 个表达式，表示最优化目标：最大化该最小值。具体地，它蕴含着 2 层含义：

- 在数据集严格线性可分的情况下，对数据集的几何间隔最大化，将确保所有的实例点都能够被正确地分类，即：

$$\max_{\boldsymbol{w},b} \quad \gamma \quad \Rightarrow \quad \gamma > 0 \quad \Rightarrow \quad y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} > 0 \tag{11}$$

- 具有最大的几何间隔：

$$\left.\begin{aligned} &\max_{\boldsymbol{w},b} \quad \gamma \\ &s.t. \quad y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \geqslant \gamma \end{aligned}\right\} \quad \Rightarrow \quad y_i \cdot \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|} \geqslant \gamma^* \quad i = 1, 2, \ldots, N \tag{12}$$

其中，γ^* 表示最佳几何间隔。

在上述带约束的优化目标 (公式 (10)) 中，除了需要对权值参数 $\boldsymbol{w}$ 和偏置参数 b 进行最优化之外，还需要对具体的几何间隔 γ 进行最优化，不是很方便。然而，仔细分析，观察到几何间隔 γ 实际上是依赖于权值 $\boldsymbol{w}$ 和偏置 b 的——它们之间具有耦合性，虽然有 3 个变量，但实际上只有 2 个自由变量。因此，可以考虑去掉几何间隔 γ。利用几何间隔与函数间隔之间的关系：$\gamma = \frac{\hat{\gamma}}{\|\boldsymbol{w}\|}$，可以做到这一点：

$$\begin{aligned} &\max_{\boldsymbol{w},b} \quad \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \\ &s.t. \quad y_i \cdot \left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geqslant \|\boldsymbol{w}\|\gamma = \hat{\gamma} \quad i = 1, 2, \ldots, N \end{aligned} \tag{13}$$

[8]数据集的几何间隔或函数间隔，被定义为所有实例点的几何间隔或函数间隔的最小值，参见前文。

其中，$\hat{\gamma}$ 是数据集的函数间隔。

到目前为止，我们仅仅对公式 (10) 做了等价变换：将它转换为公式 (13)。然而，正如前面所分析的那样，函数间隔是相对的，即当权值 $\boldsymbol{w}$ 和偏置 b 同时缩放一个因子 s 时，函数间隔将是原函数间隔的 s 倍，而几何间隔将维持不变。这一点很重要，它表明，函数间隔的具体值，对我们来说并不重要，因为我们关注的是其比值 $\gamma = \frac{\hat{\gamma}}{\|\boldsymbol{w}\|}$，即几何间隔，它才是我们的最优化目标。既然如此，$\hat{\gamma} = 1$ 时得到的分离超平面，与 $\hat{\gamma}$ 为其它数值时得到的分离超平面，并没有本质上的不同——对不同的 $\hat{\gamma}$ 而言，权值向量 $\boldsymbol{w}$ 的大小在发生变化，但其方向不会发生任何变化，即分离超平面都是同一个。于是，令 $\hat{\gamma} = 1$，从而优化目标变为：

$$\begin{aligned} \max_{\boldsymbol{w},b} \quad & \frac{1}{\|\boldsymbol{w}\|} \\ s.t. \quad & y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{14}$$

按照惯例，将最大化问题转换为等价的最小化问题，得到：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \|\boldsymbol{w}\| \\ s.t. \quad & y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{15}$$

进一步地，考虑到 $\|\boldsymbol{w}\|$ 中含有根号，不便于求解，因此，将其继续转换为等价的最小化问题：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ s.t. \quad & y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{16}$$

注意，虽然待优化的目标函数由 $\|\boldsymbol{w}\|$ 变成了 $\frac{1}{2}\|\boldsymbol{w}\|^2$，即优化函数发生了变化，但是，最优点或极值点 (权值参数 $\boldsymbol{w}^*$ 和偏置 b^*) 的位置不会发生变化。由此，我们得到了一个凸优化问题，具体地，这是一个凸二次规划 (Convex Quadratic Programming) 问题[9]。

[9]凸优化问题表示为：

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & f(\boldsymbol{w}) \\ s.t. \quad & g_i(\boldsymbol{w}) \leqslant 0, \quad i = 1, 2, \ldots, K \\ & h_j(\boldsymbol{w}) = 0, \quad j = 1, 2, \ldots, L \end{aligned} \tag{17}$$

其中，目标函数 $f(\boldsymbol{w})$ 和不等式约束函数 $g_i(\boldsymbol{w})$ 都是 $\mathbb{R}^n$ 上的连续可微凸函数，等式约束函数 $h_j(\boldsymbol{w})$ 是 $\mathbb{R}^n$ 上的仿射函数。仿射函数被定义为：$h_j(\boldsymbol{w}) = \boldsymbol{a}^T\boldsymbol{w} + b$，$a \in \mathbb{R}^n$，$b \in \mathbb{R}$，$\boldsymbol{w} \in \mathbb{R}^n$。当目标函数 $f(\boldsymbol{w})$ 为二次函数，且不等式约束函数 $g_i(\boldsymbol{w})$ 为仿射函数时，凸优化问题成为凸二次规划问题。

对上述优化问题，直接进行求解，可以得到最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^*。该方法被称为线性可分支持向量机的原始学习算法。下面，给出算法描述。

算法 **1.1**（线性可分支持向量机的原始学习算法）

Input:

Linear Separable Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Best Model Parameters: $\boldsymbol{w}^*$, b^*

Algorithm:

Get $\boldsymbol{w}^*$ and b^* by:

$$\begin{aligned} &\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &s.t. \quad y_i \cdot \left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{18}$$

1.3 最优分离超平面的存在性与唯一性

定理 **1.1**（最优分离超平面的存在性与唯一性）如果训练数据集 T 线性可分，那么使用上述原始学习算法得到的最优分离超平面存在且唯一。

证明：

1. 最优分离超平面的存在性

   从训练数据集的角度而言，由于训练数据集线性可分，那么一定存在 (无穷多个) 分离超平面能够将数据集的正负实例点区分开，即约束条件 $y_i \cdot \left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geqslant 1$ 一定成立，其中 $i = 1, 2, \ldots, N$。另一方面，对于凸二次优化问题 (公式 (16))，目标函数 $\frac{1}{2}\|\boldsymbol{w}\|^2$ 存在下界，这意味着，它的最小化求解必定能够得到某个最优解 $\boldsymbol{w}^*$ 和 b^*，且 $\boldsymbol{w}^*$ 不可能是 $\mathbf{0}$ 向量。原因在于，假设 $\boldsymbol{w}^* = \mathbf{0}$，那么约束条件变为 $y \cdot b^* \geqslant 1$，又由于训练数据集中既有正实例点，又有负实例点——对于正实例点 $\boldsymbol{x}_i$，$y_i = +1$，那么由 $y_i \cdot b^* \geqslant 1$，可以得到 $b^* > 0$；对于负实例点 $\boldsymbol{x}_j$，$y_j = -1$，那么由 $y_j \cdot b^* \geqslant 1$，可以得到 $b^* < 0$。然而，b^* 是一个常数，这就产生了矛盾。因此，假设 $\boldsymbol{w}^* = \mathbf{0}$ 不成立。由此可知，由上述算法得到的最优分离超平面 $\boldsymbol{x}^T\boldsymbol{w}^* + b^* = 0$ 必定存在。

2. 最优分离超平面的唯一性

首先，证明最优权值向量 $\boldsymbol{w}^*$ 的唯一性。

假设存在 2 个最优解，并将它们分别记为 $(\boldsymbol{w}_1^*, b_1^*)$ 和 $(\boldsymbol{w}_2^*, b_2^*)$。根据原始学习算法的目标函数可知，$\|\boldsymbol{w}_1^*\| = \|\boldsymbol{w}_2^*\|$。于是，令 $\|\boldsymbol{w}_1^*\| = \|\boldsymbol{w}_2^*\| = c$，其中 c 为某个大于 0 的常数。

下面，利用这 2 个最优解，计算出 1 个新的超平面：

$$\begin{aligned} \boldsymbol{w} &= \frac{\boldsymbol{w}_1^* + \boldsymbol{w}_2^*}{2} \\ b &= \frac{b_1^* + b_2^*}{2} \end{aligned} \tag{19}$$

容易验证，这个新超平面满足约束条件 $y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1$，其中 $i = 1, 2, \ldots, N$：

$$\begin{aligned} y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) &= y_i \cdot \left(\boldsymbol{x}_i^T \frac{\boldsymbol{w}_1^* + \boldsymbol{w}_2^*}{2} + \frac{b_1^* + b_2^*}{2} \right) \\ &= \frac{1}{2} \left[y_i \cdot \left(\boldsymbol{x}_i^T \boldsymbol{w}_1^* + b_1^* \right) + y_i \cdot \left(\boldsymbol{x}_i^T \boldsymbol{w}_2^* + b_2^* \right) \right] \\ &\geqslant 1 \end{aligned} \tag{20}$$

这表明，$(\boldsymbol{w}, b)$ 也是一个分离超平面。于是，$\|\boldsymbol{w}\| \geqslant c$ 成立。另一方面，利用向量的三角不等式 $\|\boldsymbol{w}_1^* + \boldsymbol{w}_2^*\| \leqslant \|\boldsymbol{w}_1^*\| + \|\boldsymbol{w}_2^*\|$[10]：

$$\boldsymbol{w} = \frac{\boldsymbol{w}_1^* + \boldsymbol{w}_2^*}{2} \quad \Rightarrow \quad \|\boldsymbol{w}\| = \frac{1}{2} \|\boldsymbol{w}_1^* + \boldsymbol{w}_2^*\| \leqslant \frac{1}{2} \left(\|\boldsymbol{w}_1^*\| + \|\boldsymbol{w}_2^*\| \right) = c \tag{21}$$

综合 $\|\boldsymbol{w}\| \geqslant c$，得到：

$$\begin{aligned} & c \leqslant \|\boldsymbol{w}\| \leqslant c \quad \Rightarrow \\ & \|\boldsymbol{w}\| = \frac{1}{2} \|\boldsymbol{w}_1^* + \boldsymbol{w}_2^*\| = c \quad \Rightarrow \\ & \|\boldsymbol{w}_1^* + \boldsymbol{w}_2^*\| = \|\boldsymbol{w}_1^*\| + \|\boldsymbol{w}_2^*\| \end{aligned} \tag{22}$$

这表明，向量 $\boldsymbol{w}_1^*$ 与 $\boldsymbol{w}_2^*$ 共线。于是，令 $\boldsymbol{w}_1^* = \lambda \boldsymbol{w}_2^*$，根据 $\|\boldsymbol{w}_1^*\| = \|\boldsymbol{w}_2^*\| = c$ 可知，$|\lambda| = 1$。如果 $\lambda = -1$，那么向量 $\boldsymbol{w}_1^*$ 与 $\boldsymbol{w}_2^*$ 反向，于是：

$$\boldsymbol{w} = \frac{\boldsymbol{w}_1^* + \boldsymbol{w}_2^*}{2} \quad \Rightarrow \quad \boldsymbol{w} = 0 \tag{23}$$

这与 $(\boldsymbol{w}, b)$ 为分离超平面的事实相矛盾。因此，$\lambda = 1$ 成立，即 $\boldsymbol{w}_1^* = \boldsymbol{w}_2^*$ 成立，这意味着最优权值参数 $\boldsymbol{w}^*$ 具有唯一性。

[10] 仅当向量 $\boldsymbol{w}_1^*$ 与向量 $\boldsymbol{w}_2^*$ 共线时，等式成立。

于是，上述 2 个最优解 $(\boldsymbol{w}_1^*, b_1^*)$ 和 $(\boldsymbol{w}_2^*, b_2^*)$ 可以分别改写为 $(\boldsymbol{w}^*, b_1^*)$ 和 $(\boldsymbol{w}^*, b_2^*)$。下面，再证明最优偏置 b^* 具有唯一性。

考虑间隔超平面上的点，即满足约束条件 $y_i \cdot \left(\boldsymbol{x}_i^T \boldsymbol{w}^* + b^*\right) = 1$ 的实例点，将约束条件变形为：

$$y_i \cdot \left(\boldsymbol{x}_i^T \boldsymbol{w}^* + b^*\right) = 1 \quad \Rightarrow \quad y_i^2 \cdot \left(\boldsymbol{x}_i^T \boldsymbol{w}^* + b^*\right) = y_i \quad \Rightarrow \quad \boldsymbol{x}_i^T \boldsymbol{w}^* + b^* = y_i \tag{24}$$

于是，设 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 分别是最优解 $(\boldsymbol{w}^*, b_1^*)$ 和最优解 $(\boldsymbol{w}^*, b_2^*)$ 对应的正间隔超平面上的实例点，则：

$$\left.\begin{aligned} \boldsymbol{x}_1^T \boldsymbol{w}^* + b_1^* = 1 \\ \boldsymbol{x}_2^T \boldsymbol{w}^* + b_2^* = 1 \end{aligned}\right\} \quad \Rightarrow \quad b_1^* - b_2^* = \boldsymbol{x}_2^T \boldsymbol{w}^* - \boldsymbol{x}_1^T \boldsymbol{w}^* \tag{25}$$

对于最优解 $(\boldsymbol{w}^*, b_1^*)$ 有：

$$\boldsymbol{x}_2^T \boldsymbol{w}^* + b_1^* \geqslant \boldsymbol{x}_1^T \boldsymbol{w}^* + b_1^* = 1 \quad \Rightarrow \quad \boldsymbol{x}_2^T \boldsymbol{w}^* \geqslant \boldsymbol{x}_1^T \boldsymbol{w}^* \tag{26}$$

而对于最优解 $(\boldsymbol{w}^*, b_2^*)$ 有：

$$\boldsymbol{x}_1^T \boldsymbol{w}^* + b_2^* \geqslant \boldsymbol{x}_2^T \boldsymbol{w}^* + b_2^* = 1 \quad \Rightarrow \quad \boldsymbol{x}_1^T \boldsymbol{w}^* \geqslant \boldsymbol{x}_2^T \boldsymbol{w}^* \tag{27}$$

于是，得到：

$$\boldsymbol{x}_1^T \boldsymbol{w}^* = \boldsymbol{x}_2^T \boldsymbol{w}^* \quad \Rightarrow \quad b_1^* - b_2^* = \boldsymbol{x}_2^T \boldsymbol{w}^* - \boldsymbol{x}_1^T \boldsymbol{w}^* = 0 \quad \Rightarrow \quad b_1^* = b_2^* \tag{28}$$

由此可知，最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^* 都具有唯一性，最优解的唯一性得证。

□

1.4 原始学习算法的求解实例

例 **1.1** 如图1-2所示，正实例点为 $\boldsymbol{x}_1 = (3,3)^T$ 和 $\boldsymbol{x}_2 = (4,3)^T$，负实例点为 $\boldsymbol{x}_3 = (1,1)^T$，试用线性可分支持向量机的原始学习算法求解最优分离超平面。

解：根据线性可分支持向量机的原始最优化问题 (公式 (16))：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ s.t. \quad & y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{29}$$

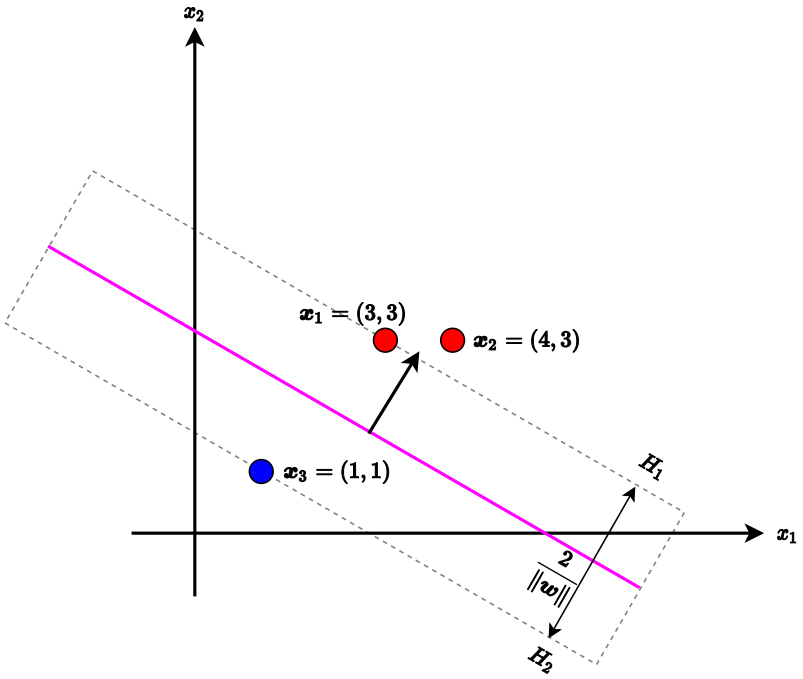


图 1-2: 硬间隔最大分离超平面示例

由所给定的数据集构造如下的约束最优化问题，且根据问题性质，令 $\boldsymbol{w} = (w_1, w_2)$：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\left(w_1^2 + w_2^2\right) \\ s.t. \quad & 3w_1 + 3w_2 + b \geqslant 1 \\ & 4w_1 + 3w_2 + b \geqslant 1 \\ & -w_1 - w_2 - b \geqslant 1 \end{aligned} \tag{30}$$

下面，将应用拉格朗日乘数法：首先，写出拉格朗日优化函数，将约束最优化问题转换为无约束最优化问题；然后，求解各变量的偏导数，并令它们等于 0，得到方程组；最后，求解方程组。需要注意的是，在求解方程组的过程中，需要排除不满足 KKT 条件的解[11]。

[11] 关于拉格朗日优化问题中的 KKT 条件，详见《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

首先，写出拉格朗日优化函数：

$$L(\boldsymbol{w},b,\boldsymbol{\alpha})=\frac{1}{2}\left(w_1^2+w_2^2\right)+\alpha_1\left(1-3w_1-3w_2-b\right)+\alpha_2\left(1-4w_1-3w_2-b\right)+\\ \alpha_3\left(1+w_1+w_2+b\right) \tag{31}$$

其中，$\boldsymbol{\alpha}$ 表示拉格朗日乘数 $(\alpha_1,\alpha_2,\alpha_3)$，需要满足的 KKT 条件为[12]：

$$\begin{cases}\alpha_i\geqslant 0 \quad i=1,2,3\\ \alpha_1\left(1-3w_1-3w_2-b\right)=0\\ \alpha_2\left(1-4w_1-3w_2-b\right)=0\\ \alpha_3\left(1+w_1+w_2+b\right)=0\end{cases} \tag{32}$$

其中，第 1 个条件被称为不等式约束条件；后面的条件被称为不等式互补条件，这是此例中 2 种主要的 KKT 条件。特别地，当 $\alpha_i>0$ 时，其对应的不等式约束将等于 0，表示相应的实例点有效或激活；当 $\alpha_i=0$ 时，其对应的不等式约束将不起作用，表示相应的实例点失效。在下面的求解过程中，可以看到这 2 种类型的实例点。然后，求解各变量的偏导数，并令它们等于 0，得到如下的方程组：

$$\begin{cases}(1)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial w_1}=w_1-3\alpha_1-4\alpha_2+\alpha_3=0\\ (2)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial w_2}=w_2-3\alpha_1-3\alpha_2+\alpha_3=0\\ (3)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial b}=-\alpha_1-\alpha_2+\alpha_3=0\\ (4)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial \alpha_1}=1-3w_1-3w_2-b=0\\ (5)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial \alpha_2}=1-4w_1-3w_2-b=0\\ (6)\dfrac{\partial L(\boldsymbol{w},b,\boldsymbol{\alpha})}{\partial \alpha_3}=1+w_1+w_2+b=0\end{cases} \tag{33}$$

解方程组：$(4)+3*(6)$，解得 $b=-2$；$(5)+3*(6)$，解得 $w_1=0$；将 $w_1=0$ 代入 (6)，解得 $w_2=1$；将 (3) 整理成 $\alpha_3=\alpha_1+\alpha_2$，代入 (1)，同时将 w_1=0 代入 (1)，解得 $2\alpha_1+3\alpha_2=0$；将 $\alpha_3=\alpha_1+\alpha_2$，代入 (2)，同时将 w_2=1 代入 (2)，解得 $2\alpha_1+2\alpha_2=1$；解上述 2 个中间式，可得 $\alpha_1=\frac{3}{2}$、$\alpha_2=-1$，最后解得 $\alpha_3=\frac{1}{2}$。

[12]KKT 条件还包括 $L(\boldsymbol{w},b,\boldsymbol{\alpha})$ 关于 $\boldsymbol{w}^*$ 和 b^* 的梯度为 0 以及原始的不等式约束条件，例如 $3w_1+3w_2+b\geqslant 1$ 等。其中，“梯度为 0”条件体现在随后的方程组求解过程中；“原始不等式约束”体现在拉格朗日优化函数以及拉格朗日乘数约束条件与互补条件中。

但是，$\alpha_2 = -1$ 不符合 KKT 条件，令 $\alpha_2 = 0$，将 (5) 从方程组中去掉。实际上，α_2 对应的样本点为 $\boldsymbol{x}_2 = (4, 3)$，它在正例类别的内部，与决策面的形成没有关系。解剩余方程组，可得：$w_1 = w_2 = \frac{1}{2}$、$b = -2$、$\alpha_1 = \alpha_3 = \frac{1}{4}$。

因此，最大间隔分离超平面为：$\frac{1}{2}x_1 + \frac{1}{2}x_2 - 2 = 0$ □

下面，使用 Python 相关的工具包，重新计算例1.1：

```
import numpy as np
from scipy.linalg import solve
```

首先，写出拉格朗日优化函数，求解该函数对 w_1、w_2、b、α_1、α_2、α_3 的偏导数，得到如下的线性方程组 (假定 α_1、α_2、α_3 都是活跃的)。

```
a = np.array([[1,0,0,-3,-4,1],\
              [0,1,0,-3,-3,1],\
              [0,0,0,-1,-1,1],\
              [-3,-3,-1,0,0,0],\
              [-4,-3,-1,0,0,0],\
              [1,1,1,0,0,0]])
b = np.array([0,0,0,-1,-1,-1])

x = solve(a, b)
print(x)
```

```
[  1.66533454e-16   1.00000000e+00  -2.00000000e+00   1.50000000e+00
  -1.00000000e+00   5.00000000e-01]
```

上面的解分别对应：w_1、w_2、b、α_1、α_2、α_3。可以看出，α_2 为负数，不符合 KKT 条件：$\alpha_2 \geqslant 0$。因此，设置 $\alpha_2 = 0$，使得它对应的方程变得不活跃（不起作用）。重新列出线性方程组如下：

```
a = np.array([[1,0,0,-3,1],\
              [0,1,0,-3,1],\
              [0,0,0,-1,1],\
              [-3,-3,-1,0,0],\
              [1,1,1,0,0]])
b = np.array([0,0,0,-1,-1])

x = solve(a, b)
print(x)
```

```
[ 0.5   0.5  -2.    0.25  0.25]
```

再次检查解是否符合 KKT 条件：全部满足。于是，解：$w_1 = \frac{1}{2}$、$w_2 = \frac{1}{2}$、$b = -2$、$\alpha_1 = \frac{1}{4}$、$\alpha_2 = 0$、$\alpha_3 = \frac{1}{4}$ 即为所求。可以看出，程序求解的结果与手工计算的结果一致。

从图1-2可以看出，$\boldsymbol{x}_1$ 和 $\boldsymbol{x}_3$ 在间隔超平面上，决定了分离超平面；而 $\boldsymbol{x}_2$ 为非活跃点，对分离超平面的形成不起作用。从拉格朗日乘数的角度看，实例点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_3$ 的乘数大于 0，根据不等式互补条件有：

$$\begin{aligned} 1 - 3w_1 - 3w_2 - b = 0 \\ 1 + w_1 + w_2 + b = 0 \end{aligned} \tag{34}$$

其对应的不等式约束起作用，这 2 个实例点刚好在正负间隔超平面上。对于实例点 $\boldsymbol{x}_2$ 而言，其乘数不符合 KKT 条件，被设置为 0，其对应的不等式约束不起作用，该实例点失效，对分离超平面的形成没有作用。

实际上，从约束条件 $y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1$ 的角度看，我们可以将训练数据集分成如下的两类：

1. $y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) = 1$

   满足此类条件的实例点，都在间隔超平面上：或者在正间隔超平面上，或者在负间隔超平面上，它们对分离超平面的形成起到了决定性作用，属于活跃点。从拉格朗日优化函数的角度看，它们的拉格朗日乘数不等于 0。这意味着，这些乘数所对应的约束条件，对硬间隔最大化的优化起到了约束作用。我们将此类实例点称为“支持向量”(Support Vector)，支持向量机由此得名。一般来说，支持向量的数量很少。因此，支持向量机是由少数重要的训练实例点确定的。

2. $y_j \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_j + b\right) > 1$

   满足此类条件的实例点，都“远离”间隔超平面，它们对分离超平面的形成没有作用，属于失效点或非活跃点。从拉格朗日优化函数的角度看，它们的拉格朗日乘数等于 0，因而导致这些乘数所对应的约束条件不起作用。

在图1-2中，H_1 和 H_2 分别为正间隔超平面与负间隔超平面，紫色的分离超平面与它们平行且位于它们的正中间位置。H_1 与 H_2 之间的距离被称为间隔 (Margin)，其长度为 $\frac{2}{\|\boldsymbol{w}\|}$。

1.5 学习的对偶算法

在前面，我们已经讨论了线性可分支持向量机的原始最优化问题 (公式 (16))。为方便，将其复述如下：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ s.t. \quad & y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geqslant 1 \quad i = 1, 2, \ldots, N \end{aligned} \tag{35}$$

为得到最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^*，需要执行如下的求解过程：首先，列出其拉格朗日优化函数，然后求解优化函数关于各参数 (除了上述参数外，还包括拉格朗日乘数) 的偏导数，并令它们等于 0，得到方程组；最后，结合 KKT 条件，求解方程组，就得到了最优解。

上述问题之所以被称为原始最优化问题，主要原因有如下 2 点：

1. 直接以问题的求解目标，即 $\boldsymbol{w}$ 和 b，作为最终的优化对象，虽然在求解的过程中，也将拉格朗日乘数作为 (中间过渡的) 求解目标。一般地，可以采用 2 阶段来表示该原始最优化问题[13]：

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{36}$$

   其中，$\boldsymbol{\alpha}$ 表示拉格朗日乘数，为中间过渡目标；而 $\boldsymbol{w}$ 和 b 是最终优化目标。

2. 还存在一种所谓的对偶最优化问题，它与原始最优化问题相对应。对偶最优化问题，也是采用 2 阶段来表示：

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{37}$$

   其参数意义同上。可以看出，在对偶最优化问题中，以拉格朗日乘数 $\boldsymbol{\alpha}$ 作为最终的优化对象，而以真正的优化对象 $\boldsymbol{w}$ 和 b 作为中间过渡的求解目标。尽管如此，对于强对偶性问题，原始最优化问题的解与对偶最优化问题的解是等价的[14]。

形式上看，原始最优化问题为极小-极大问题，对偶最优化问题为极大-极小问题。

[13] 关于拉格朗日的 2 阶段优化以及原始优化问题与对偶优化问题的讨论，详见《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

[14] 关于强对偶性与弱对偶性的讨论，详见《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

线性可分支持向量机满足强对偶性条件，因而可以通过求解对偶最优化问题，而得到原始最优化问题的解，这就是所谓的对偶学习算法。它的优点是：对偶问题往往更容易求解；可以很自然地引入核函数，进而推广到非线性分类问题上。

下面，将针对任意线性可分数据集，利用拉格朗日优化函数与对偶性，将线性可分支持向量机的原始最优化问题转换为等价的对偶最优化问题。

首先，针对线性可分支持向量机的原始最优化问题 (公式 (35))，构建它的拉格朗日优化函数：

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \alpha_i \left(1 - y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right)\right) \\ &= \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} + \sum_{i=1}^{N} \alpha_i \left(1 - y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right)\right) \end{aligned} \tag{38}$$

其中，N 为数据集中样本点的个数；α_i 为不等式约束的拉格朗日乘数，相应的 KKT 条件为[15]：

$$\begin{cases} \alpha_i \geqslant 0 \\ \alpha_i \left(1 - y_i \cdot \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right)\right) = 0 \quad i = 1, 2, \ldots, N \end{cases} \tag{39}$$

其中，第 1 个条件称为拉格朗日乘数约束条件，第 2 个条件称为对偶互补条件，这是线性可分支持向量机中的 2 个主要 KKT 条件。

根据拉格朗日对偶性，对偶问题为极大极小问题：

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{40}$$

因此，为了得到对偶问题的解，需要先求解内部的 $\min\limits_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\alpha})$，然后再求解外层关于 $\boldsymbol{\alpha}$ 的极大化。

于是，求解 $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ 关于 $\boldsymbol{w}$ 和 b 的偏导数，并令其为 0：

$$\begin{aligned} &\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \\ &\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{41}$$

[15]在前一节中，我们已经讨论过，KKT 条件还包括 $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ 关于 $\boldsymbol{w}^*$ 和 b^* 的梯度为 0 以及原始的不等式约束条件。其中，“梯度为 0”条件体现在随后的方程组求解过程中；“原始不等式约束”体现在拉格朗日优化函数以及拉格朗日乘数约束条件与互补条件中。

将求解结果 $\boldsymbol{w}=\sum\limits_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i$ 代入拉格朗日优化函数 (公式 (38)) 中，并利用另一个求解结果 $\sum\limits_{i=1}^{N}\alpha_i y_i=0$，得到：

$$\begin{aligned}\min_{\boldsymbol{w},b} L(\boldsymbol{w},b,\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+\sum_{i=1}^{N}\alpha_i\left(1-y_i\cdot\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right)\\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j+\sum_{i=1}^{N}\alpha_i\left(1-y_i\cdot\left(\sum_{j=1}^{N}\alpha_j y_j\boldsymbol{x}_j^T\boldsymbol{x}_i+b\right)\right)\\ &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j+\sum_{i=1}^{N}\alpha_i-\sum_{i=1}^{N}\alpha_i y_i b\\ &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j+\sum_{i=1}^{N}\alpha_i\end{aligned} \tag{42}$$

于是，对偶问题 (公式 (40)) 变换为：

$$\max_{\boldsymbol{\alpha}}\min_{\boldsymbol{w},b} L(\boldsymbol{w},b,\boldsymbol{\alpha})=\max_{\boldsymbol{\alpha}}-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j+\sum_{i=1}^{N}\alpha_i \tag{43}$$

按照惯例，将最大化问题转换为等价的最小化问题，并写出约束条件，就得到与原始最优化问题对应的对偶最优化问题：

$$\begin{aligned}&\min_{\boldsymbol{\alpha}}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j-\sum_{i=1}^{N}\alpha_i\\ &s.t.\quad \alpha_i\geqslant 0\quad i=1,2,\ldots,N\\ &\quad\sum_{i=1}^{N}\alpha_i y_i=0\end{aligned} \tag{44}$$

其中，第 1 个条件为与不等式约束对应的拉格朗日乘数约束；第 2 个条件为内层最优化求解产生的中间结果。

线性可分支持向量机符合强对偶性条件，其原始最优化问题与对偶最优化问题等价。这意味着，从对偶问题的最优解 $\boldsymbol{\alpha}^*$ 出发，使用公式 $\boldsymbol{w}^*=\sum\limits_{i=1}^{N}\alpha_i^* y_i\boldsymbol{x}_i$ 计算得到的最优权值参数 $\boldsymbol{w}^*$ 与偏置参数 b^*，与直接求解原始最优化问题得到的最优权值参数 $\boldsymbol{w}^*$ 与偏置参数 b^* 是一致的。

定理 1.2 设 $\boldsymbol{\alpha}^*=(\alpha_1^*,\alpha_2^*,\ldots,\alpha_N^*)^T$ 是对偶最优化问题(公式(44)) 的解，则必定存在某个实例点 j，其相应的拉格朗日乘数 $\alpha_j^*>0$ 成立，且根据下式可以计算出

原始最优化问题(公式(16)) 的解 $\boldsymbol{w}^*$ 和 b^*:

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\ b^* &= y_j - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \end{aligned} \tag{45}$$

证明: 实际上, 在前文, 我们已经将原始最优化问题转换为对偶最优化问题 (结合 KKT 条件), 并得到了最优权值参数 $\boldsymbol{w}^*$ 的计算公式:

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \tag{46}$$

因此, 在求解得到 $\boldsymbol{\alpha}^*$ 之后, 可以根据上式, 计算出最优权值参数 $\boldsymbol{w}^*$。

由公式 (46) 可知, 至少存在 1 个实例点 (设它为 $\boldsymbol{x}_j$), 其对应的拉格朗日乘数 $\alpha_j > 0$; 否则, 如果不存在这样的实例点, 那么 $\boldsymbol{\alpha}^* = \mathbf{0}$, 则 $\boldsymbol{w}^* = \mathbf{0}$, 这与 $\boldsymbol{w}^*$ 为原始最优化问题的解相矛盾。于是, 根据 KKT 的对偶互补条件, 有 $1 - y_j(\boldsymbol{x}_j^T \boldsymbol{w}^* + b^*) = 0$, 即 $y_j(\boldsymbol{x}_j^T \boldsymbol{w}^* + b^*) = 1$。这表明, 实例点 $\boldsymbol{x}_j$ 为间隔超平面上的支持向量, 则:

$$\begin{aligned} & y_j(\boldsymbol{x}_j^T \boldsymbol{w}^* + b^*) = 1 \quad \Rightarrow \\ & \boldsymbol{x}_j^T \boldsymbol{w}^* + b^* = y_j \quad \Rightarrow \\ & b^* = y_j - \boldsymbol{x}_j^T \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \quad \Rightarrow \\ & b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \end{aligned} \tag{47}$$

□

由此可知, 在对偶最优化问题中, 最优权值参数 $\boldsymbol{w}^*$ 表现为数据集中所有实例点及其拉格朗日乘数和类别的线性组合。具体而言, $\boldsymbol{w}^*$ 是那些拉格朗日乘数大于 0 的实例点 (支持向量) 及其乘数和类别的线性组合。于是, 分离超平面和分类决策函数分别为:

$$\begin{aligned} & \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x} + b^* = 0 \\ & f(\boldsymbol{x}) = \operatorname{sign}(\sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x} + b^*) \end{aligned} \tag{48}$$

综上所述, 基于对偶最优化方法得到的算法简称为对偶算法, 它的主要优点如下:

- 对偶问题往往更容易求解；
- 将 $\boldsymbol{w}^*$ 和 b^* 表示成样本对 $(\boldsymbol{x}_i, y_i)$ 的线性组合形式，可以很自然地引入核函数，进而推广到非线性分类问题上；
- 在最优参数的迭代求解过程中，数据样本点之间仅以向量内积 $\boldsymbol{x}_i^T\boldsymbol{x}_j$ 的形式出现：可以预先计算它们之间的内积，从而形成了所谓的 Gram 矩阵：

$$\boldsymbol{G} = \left[\boldsymbol{x}_i^T\boldsymbol{x}_j\right]_{N\times N} \tag{49}$$

在使用核函数的情况下，相应的 Gram 矩阵为：

$$\boldsymbol{G} = \left[K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right]_{N\times N} \tag{50}$$

下面给出对偶学习算法的描述。

算法 **1.2**（线性可分支持向量机的对偶学习算法）

Input:

Linear Separable Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Best Model Parameters: $\boldsymbol{w}^*$, b^*

Algorithm:

Get $\boldsymbol{\alpha}^*$ by：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \alpha_i \geqslant 0 \quad i = 1, 2, \ldots, N \\ &\sum_{i=1}^{N}\alpha_i y_i = 0 \end{aligned} \tag{51}$$

Get $\boldsymbol{w}^*$ and b^* by：

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{N}\alpha_i^* y_i \boldsymbol{x}_i \\ b^* &= y_j - \sum_{i=1}^{N}\alpha_i^* y_i \boldsymbol{x}_i^T\boldsymbol{x}_j \quad \exists\alpha_j^* > 0, (\boldsymbol{x}_j, y_j) \end{aligned} \tag{52}$$

在引入对偶最优化方法之后，可以给支持向量下一个正式的定义。

定义 **1.4**（支持向量）无论是原始最优化问题，还是对偶最优化问题，将训练数据集中拉格朗日乘数 $\alpha_j > 0$ 的实例点 $\boldsymbol{x}_j$ 称为支持向量，且根据 KKT 的对偶互补条件 $\alpha_j \left(1 - y_j(\boldsymbol{x}_j^T \boldsymbol{w}^* + b^*)\right) = 0$，得到 $y_j(\boldsymbol{x}_j^T \boldsymbol{w}^* + b^*) = 1$，即实例点 $\boldsymbol{x}_j$ 一定在(正或负）间隔超平面上。

1.6 对偶学习算法的求解实例

例 **1.2** 训练数据与例1.1相同，正实例点为 $\boldsymbol{x}_1 = (3,3)^T$ 和 $\boldsymbol{x}_2 = (4,3)^T$，负实例点为 $\boldsymbol{x}_3 = (1,1)^T$，试用线性可分支持向量机的对偶学习算法求解最优分离超平面。

解：根据对偶最优化问题 (公式 (44))：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \alpha_i \geqslant 0 \quad i = 1,2,\ldots,N \\ &\qquad \sum_{i=1}^{N}\alpha_i y_i = 0 \end{aligned} \tag{53}$$

在给定的训练数据集下，上述对偶最优化问题转换为：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\left(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_3 - 14\alpha_2\alpha_3\right) - (\alpha_1 + \alpha_2 + \alpha_3) \\ &s.t. \quad \alpha_i \geqslant 0 \quad i = 1,2,3 \\ &\qquad \alpha_1 + \alpha_2 - \alpha_3 = 0 \end{aligned} \tag{54}$$

将上式的 min 内部项记为 $f(\alpha_1, \alpha_2)$，并作简单的变形：

$$\begin{aligned} f(\alpha_1,\alpha_2) &= \frac{1}{2}\left(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_3 - 14\alpha_2\alpha_3\right) - (\alpha_1 + \alpha_2 + \alpha_3) \\ &= 9\alpha_1^2 + \frac{25}{2}\alpha_2^2 + \alpha_3^2 + 21\alpha_1\alpha_2 - 6\alpha_1\alpha_3 - 7\alpha_2\alpha_3 - \alpha_1 - \alpha_2 - \alpha_3 \end{aligned} \tag{55}$$

将 $\alpha_3 = \alpha_1 + \alpha_2$ 代入，得到：

$$\begin{aligned} f(\alpha_1,\alpha_2) &= 9\alpha_1^2 + \frac{25}{2}\alpha_2^2 + \alpha_3^2 + 21\alpha_1\alpha_2 - 6\alpha_1\alpha_3 - 7\alpha_2\alpha_3 - \alpha_1 - \alpha_2 - \alpha_3 \\ &= 4\alpha_1^2 + \frac{13}{2}\alpha_2^2 + 10\alpha_1\alpha_2 - 2\alpha_1 - 2\alpha_2 \end{aligned} \tag{56}$$

对 $f(\alpha_1, \alpha_2)$ 分别关于 α_1 和 α_2 求偏导数，并令它们为 0：

$$\begin{cases} \dfrac{\partial f(\alpha_1,\alpha_2)}{\partial \alpha_1} = 8\alpha_1 + 10\alpha_2 - 2 = 0 \quad \Rightarrow \quad 4\alpha_1 + 5\alpha_2 = 1 \\ \dfrac{\partial f(\alpha_1,\alpha_2)}{\partial \alpha_2} = 13\alpha_2 + 10\alpha_1 - 2 = 0 \quad \Rightarrow \quad 5\alpha_1 + \dfrac{13}{2}\alpha_2 = 1 \end{cases} \tag{57}$$

解得 $\alpha_1^* = \frac{3}{2}$ 和 $\alpha_2^* = -1$。但是，α_2^* 不满足 KKT 条件 $\alpha_2 \geqslant 0$，于是令 $\alpha_2^* = 0$，重新解得 $\alpha_1^* = \alpha_3^* = \frac{1}{4}$，则：

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{3} \alpha_i^* y_i \boldsymbol{x}_i = \frac{1}{4}(3,3)^T - \frac{1}{4}(1,1)^T = (\frac{1}{2}, \frac{1}{2})^T \\ b^* &= y_1 - \sum_{i=1}^{3} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x}_1 \quad \alpha_1^* > 0, (\boldsymbol{x}_1, y_1) \\ &= -2 \end{aligned} \tag{58}$$

于是，得到最优分离超平面：

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 - 2 = 0 \tag{59}$$

□

2 一般线性支持向量机

2.1 学习策略：软间隔最大化

前面介绍的线性可分支持向量机，适用于数据集完全线性可分的情形——这表现为，在正负间隔超平面之间，不会存在任何实例样本点，而处于正负间隔超平面正中间位置的分离超平面，能够完全正确地将正负实例点分隔开。

在另一些情况下，数据集本身不是线性可分的——只有去掉“少数”特异点 (Outlier) 之后，其余的数据集才是线性可分的。这意味着，对此类数据集，找不到一个能够完全正确地将正负实例点分隔开的分离超平面。然而，对于此类情况，可以考虑允许“少数”实例样本点分类错误。当然，同时也需要考虑以最小的代价来进行这种分类。

为了能够继续应用线性可分支持向量机中使用到的最优化方法，可以考虑将 (硬) 间隔进行“软化”——为每个样本点 $(\boldsymbol{x}_i, y)$ 引入 1 个松弛变量 $\xi_i \geqslant 0$，从而使得每个样本点的函数间隔加上自己的松弛变量 ξ_i 之后大于等于 1，因而约束条件就变为：

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) + \xi_i \geqslant 1 \quad \Rightarrow \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geqslant 1 - \xi_i \tag{60}$$

这就是所谓的软间隔最大化方法。因此，一般线性支持向量机就是将学习策略从硬间隔最大化扩展为软间隔最大化的方法。硬间隔最大化是软间隔最大化的特殊情形。

在引入松弛变量 ξ_i 之后，从损失函数的角度看，这相当于，为每个样本点额外地配置了一个分类成本或代价 $\xi_i \geqslant 0$。因此，为获得最优的分离超平面，一般线性支持向量机的学习策略一定要考虑这种分类代价。于是，原先适用于线性可分支持向量机的目标函数 $\frac{1}{2}\|\boldsymbol{w}\|^2$ 不再适用，可以将其修改为：

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \tag{61}$$

其中，$C > 0$ 表示权重参数，用于调节两类损失的比重。上式的含义是，既要使得间隔最大化 (第 1 项)，又要使得误分类点的松弛代价之和尽量小 (第 2 项)，C 是调节两者权重的系数。

于是，一般线性支持向量机的原始最优化问题为：

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ &\qquad \xi_i \geqslant 0 \quad i = 1, 2, \ldots, N \end{aligned} \tag{62}$$

与线性可分支持向量机一样，原始最优化问题也是一个凸二次规划问题，因而关于 $(\boldsymbol{w}, b, \boldsymbol{\xi})$ 的解是存在的。可以证明，关于 $\boldsymbol{w}$ 的解是唯一的；但是，b 的解可能不是唯一的，而是存在于一个区间[16]。

下面给出一般线性支持向量机的定义。

定义 **2.1**（一般线性支持向量机）对于给定的线性不可分训练数据集，使用最大软间隔法得到的分离超平面：

$$\boldsymbol{x}^T\boldsymbol{w}^* + b^* = 0 \tag{63}$$

及其相应的分类决策函数：

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{x}^T\boldsymbol{w}^* + b^*) \tag{64}$$

它们被称为一般线性支持向量机。

对上述优化问题，直接进行求解，可以得到最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^*。该方法被称为一般线性支持向量机的原始学习算法。下面，给出算法描述。

算法 **2.1**（一般线性支持向量机的原始学习算法）

[16]详见《数据挖掘中的新方法——支持向量机》，邓乃扬，田英杰。科学出版社，2004。

Input:

Linear Inseparable Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Best Model Parameters: $\boldsymbol{w}^*$, b^*

Algorithm:

Get $\boldsymbol{w}^*$ and b^* by:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ &\qquad \xi_i \geqslant 0 \quad i = 1, 2, \ldots, N \end{aligned} \tag{65}$$

2.2 学习的对偶算法

为方便，将原始最优化问题 (公式 (62)) 复述如下：

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ &\qquad \xi_i \geqslant 0 \quad i = 1, 2, \ldots, N \end{aligned} \tag{66}$$

为了得到对偶最优化问题，首先需要写出拉格朗日优化函数：

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i\left(1 - \xi_i - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)\right) - \sum_{i=1}^{N}\mu_i\xi_i \\ &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i\left(1 - \xi_i - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)\right) - \sum_{i=1}^{N}\mu_i\xi_i \end{aligned} \tag{67}$$

其中，α_i 和 μ_i 为相应不等式约束的拉格朗日乘数。其主要的 KKT 条件为：

$$\begin{cases} \xi_i \geqslant 0 \\ \alpha_i \geqslant 0 \\ \mu_i \geqslant 0 \qquad\qquad i = 1, 2, \ldots, N \\ \alpha_i\left(1 - \xi_i - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)\right) = 0 \\ \mu_i\xi_i = 0 \end{cases} \tag{68}$$

然后，求解 $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})$ 关于 $\boldsymbol{w}$、b 和 $\boldsymbol{\xi}$ 的偏导数，并令它们为 0：

$$\begin{cases} \dfrac{\partial L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum\limits_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum\limits_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \\ \dfrac{\partial L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial b} = -\sum\limits_{i=1}^{N} \alpha_i y_i = 0 \quad \Rightarrow \quad \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \\ \dfrac{\partial L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad \Rightarrow \quad C = \alpha_i + \mu_i \end{cases} \tag{69}$$

将上述求解结果代入拉格朗日优化函数 (公式 (67)) 中，得到：

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = & \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + (\alpha_i + \mu_i) \sum_{i=1}^{N} \xi_i + \\ & \sum_{i=1}^{N} \alpha_i \left(1 - \xi_i - y_i (\sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j^T \boldsymbol{x}_i + b) \right) - \sum_{i=1}^{N} \mu_i \xi_i \\ = & -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i \end{aligned} \tag{70}$$

注意到，上式中已经不包含拉格朗日乘数 $\boldsymbol{\mu}$ 了。于是，得到对偶最优化问题：

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^{N} \alpha_i \tag{71}$$

按照惯例，将最大化问题转换为等价的最小化问题：

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{N} \alpha_i \tag{72}$$

相应的主要 KKT 条件为：

$$\begin{cases} \alpha_i \geqslant 0 \\ \mu_i \geqslant 0 \\ \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \\ C = \alpha_i + \mu_i \end{cases} \quad i = 1, 2, \ldots, N \tag{73}$$

利用 α_i 和 μ_i 之间的冗余性，消去 μ_i：

$$\begin{cases} \alpha_i \geqslant 0 \\ \mu_i \geqslant 0 \\ \alpha_i = C - \mu_i \end{cases} \quad \Rightarrow \quad 0 \leqslant \alpha_i \leqslant C \tag{74}$$

因此，得到简化的对偶最优化问题：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\qquad 0 \leqslant \alpha_i \leqslant C \quad i = 1, 2, \ldots, N \end{aligned} \tag{75}$$

值得注意的是，一般线性支持向量机与线性可分支持向量机的对偶最优化问题 (公式 (44)) 非常相似，唯一的区别在于 α_i 的取值范围：在一般线性支持向量机中，$0 \leqslant \alpha_i \leqslant C$；而在线性可分支持向量机中，$0 \leqslant \alpha_i$。

现在，回顾一下，在线性可分支持向量机中，根据 α_i 的取值，可以将实例点分为 2 类：支持向量和非支持向量，前者 $\alpha_i > 0$，而后者 $\alpha_i = 0$，且支持向量一定在硬间隔超平面上。

那么，在一般线性支持向量机中，情况又如何呢？总体上而言，由于松弛变量 ξ_i 的存在，硬间隔变为软间隔，情况要变得相对复杂些：支持向量不一定在 (软) 间隔超平面上。同样，根据 α_i 的取值，可以对实例点进行如下分类：

1. $\alpha_i = 0$：非支持向量

   与线性可分支持向量机中的情形一样，这类实例点，对分离超平面的形成，没有作用，它们被正确地分类且“远离”正负 (软) 间隔超平面。

2. $0 < \alpha_i$：(软间隔) 支持向量

   (a) $\alpha_i < C$：软间隔超平面上的支持向量

   利用 (2 阶段优化中的) 主要 KKT 条件 (公式 (68) 和公式 (73))，得到：

$$\begin{aligned} &\begin{cases} C = \alpha_i + \mu_i \\ \alpha_i < C \end{cases} \quad \Rightarrow \quad \mu_i > 0 \\ &\begin{cases} \mu_i > 0 \\ \mu_i \xi_i = 0 \end{cases} \quad \Rightarrow \quad \xi_i = 0 \end{aligned} \tag{76}$$

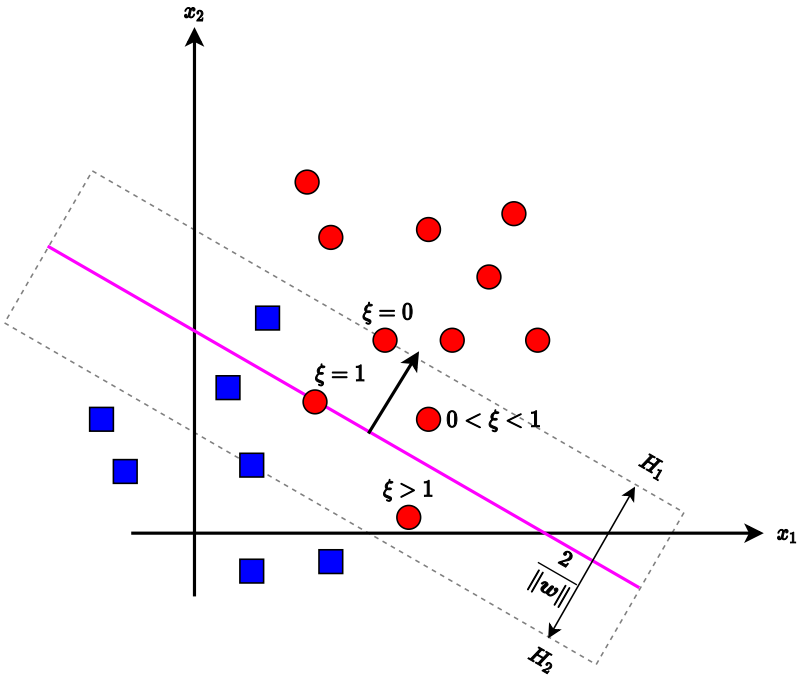


图 2-3: 一般线性支持向量机的 (软间隔) 支持向量

这表明，当 $\alpha_i < C$ 时，该实例点的松弛变量 $\xi_i = 0$，那么：

$$\begin{cases} \alpha_i > 0 \\ \alpha_i \left(1 - \xi_i - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)\right) = 0 \\ \xi_i = 0 \end{cases} \quad \Rightarrow \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1 \tag{77}$$

即实例点 $\boldsymbol{x}_i$ 刚好在 (软) 间隔超平面上。

(b) $\alpha_i = C$：一般支持向量

$$\begin{cases} \alpha_i > 0 \\ \alpha_i \left(1 - \xi_i - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)\right) = 0 \end{cases} \quad \Rightarrow \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1 - \xi_i \tag{78}$$

i. $0 < \xi_i < 1$：实例点被正确分类，但函数间隔小于正常间隔 1，即实例点位于间隔超平面与分离超平面之间；

ii. $\xi_i = 1$：实例点在分离超平面上；

iii. $\xi_i > 1$：实例点被错误分类；

图2-3，展示了一般线性支持向量机的非支持向量与支持向量。在该图中，紫

色实线表示分离超平面，与其平行的 2 条虚线表示软间隔超平面；红色实心圆表示正实例点，蓝色方块表示负实例点；图中分别标出了 ξ 取不同值时所对应的支持向量。需要注意的是，所标注的 ξ 值以函数间隔值为度量基准，而非几何间隔值；软间隔超平面 H_1 和 H_2 之间标注的间隔值为几何间隔值。

定理 2.1 设 $\boldsymbol{\alpha}^* = (\alpha_1^*, \alpha_2^*, \ldots, \alpha_N^*)^T$ 是对偶最优化问题(公式(75)) 的一个解，则必定存在某个实例点 j，其相应的拉格朗日乘数 $0 < \alpha_j^* < C$ 成立，且根据下式可以计算出原始最优化问题(公式(62)) 的解 $\boldsymbol{w}^*$ 和 b^*：

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\ b^* &= y_j - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \end{aligned} \tag{79}$$

证明：第 1 个公式直接来源于公式 (69)，第 2 个公式来源于公式 (77)，证明过程与定理1.2类似，在此不再赘述。 □

下面给出对偶学习算法的描述。

算法 **2.2**（一般线性支持向量机的对偶学习算法）

Input:

Linear Inseparable Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Best Model Parameters: $\boldsymbol{w}^*$, b^*

Algorithm:

Get $\boldsymbol{\alpha}^*$ by:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{N} \alpha_i \\ &s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \\ &\qquad 0 \leqslant \alpha_i \leqslant C \quad i = 1, 2, \ldots, N \end{aligned} \tag{80}$$

Get $\boldsymbol{w}^*$ and b^* by:

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\ b^* &= y_j - \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \quad \exists 0 < \alpha_j^* < C, (\boldsymbol{x}_j, y_j) \end{aligned} \tag{81}$$

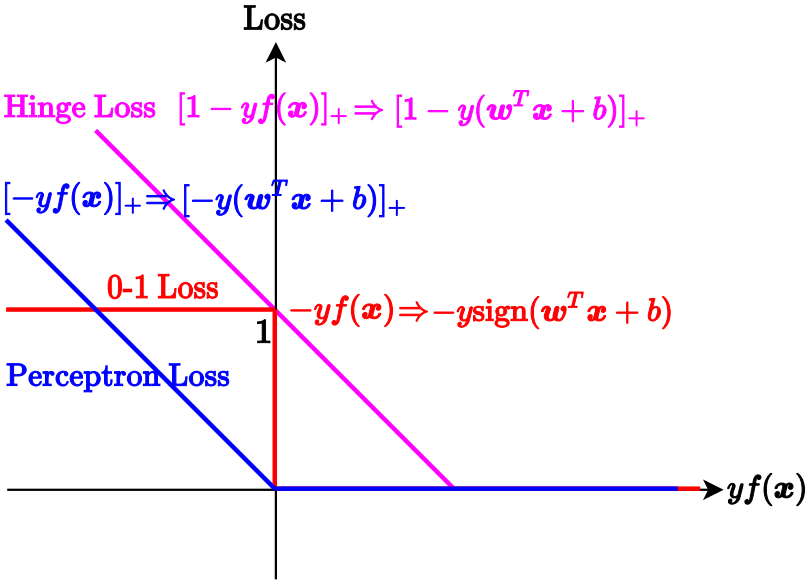


图 2-4: 0-1 损失、感知机损失与合页损失

2.3 合页损失函数的视角

一般线性支持向量机的学习策略为软间隔最大化，从损失函数的角度看，它等价于如下带正则项的合页损失函数的最小化：

$$\sum_{i=1}^{N}\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+ + \lambda\|\boldsymbol{w}\|^2 \tag{82}$$

其中，上式的第 1 项表示经验损失，被称为合页损失函数 (Hinge Loss Function)，因其函数曲线形似门合页而得名。其核心函数为：

$$[z]_+ = \begin{cases} z & z>0 \\ 0 & z\leqslant 0 \end{cases} \tag{83}$$

其中，下标 $+$ 表示当 $z>0$ 时，函数值取 z；否则，函数值为 0。对于单实例样本 $(\boldsymbol{x}_i, y_i)$ 而言，当其函数间隔 $y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geqslant 1$ 时，表明该实例点以较高的确信度被正确分类，那么 $\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+=0$，即损失为 0；否则，损失为 $1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)$。带正则项的合页损失函数的第 2 项为 $\boldsymbol{w}$ 的 L_2 范数，λ 为正则项系数。

图2-4展示了三种损失函数，它们分别为 0-1 损失 (红色实线)、感知机损失 (蓝色实线) 与合页损失 (紫色实线)。

需要注意的是，横轴表示 $yf(\boldsymbol{x})$，纵轴表示损失。对于二分类问题，0-1 损失函数才是真正的损失函数。然而，由于该损失函数不能提供有效的优化信息，因此，在实际应用中，往往需要使用其它的代理损失函数 (Surrogate Loss Function) 作为实际的优化函数。例如，一般可以使用 0-1 损失的上界函数作为实际的优化函数。对于线性支持向量机而言，合页损失函数是 0-1 损失的上界函数。

感知机的损失函数为 $\left[-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+$，而一般线性支持向量机的损失函数为 $\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+$。它们的主要区别在于，后者的损失函数中增加了最小函数间隔 1 这一项——线性支持向量机不仅要求分类正确，而且还要求以较高的确信度进行分类；而感知机模型仅要求分类正确即可。

下面的定理，给出了软间隔最大化与正则项合页损失函数的最小化之间的等价性证明。

定理 2.2 一般线性支持向量机的原始最优化问题：

$$\begin{aligned}&\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\xi_i\\&s.t.\quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1-\xi_i\\&\qquad\xi_i\geqslant 0\quad i=1,2,\ldots,N\end{aligned}\tag{84}$$

等价于如下带正则项的合页损失函数的最小化：

$$\min_{\boldsymbol{w},b}\sum_{i=1}^{N}\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+\lambda\|\boldsymbol{w}\|^2\tag{85}$$

证明：

观察到式 $1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)$ 与松弛变量 ξ_i 之间的关系，证明过程相当直接。

首先证明，带正则项的合页损失函数的最小化问题 (公式 (85)) 可以转换为一般线性支持向量机的原始最优化问题 (公式 (84))。令：

$$\xi_i=\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+\tag{86}$$

1. 根据函数 $[z]_+$ 的定义可知，$\xi_i\geqslant 0$ 成立，这表明公式 (84) 中关于 ξ_i 的不等式约束 $\xi_i\geqslant 0$ 成立；

2. 根据函数 $[z]_+$ 的定义可知，$\xi_i\geqslant 1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)$ 即 $y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geqslant 1-\xi_i$ 始终成立，这表明公式 (84) 中另一个不等式约束 $y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geqslant 1-\xi_i$ 也成立；

3. 对带正则项的合页损失函数的最小化问题进行变形：

$$\begin{aligned}
&\arg\min_{\boldsymbol{w},b}\sum_{i=1}^{N}\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+ + \lambda\|\boldsymbol{w}\|^2 = \\
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\lambda\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N}\xi_i = \\
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}} 2\lambda\left(\frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2\lambda}\sum_{i=1}^{N}\xi_i\right) = \\
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i
\end{aligned} \tag{87}$$

其中，$C=\frac{1}{2\lambda}$。因此，可以将带正则项的合页损失函数的最小化问题 (公式 (85)) 转换为一般线性支持向量机的原始最优化问题 (公式 (84))。

然后证明，一般线性支持向量机的原始最优化问题 (公式 (84)) 可以转换为带正则项的合页损失函数的最小化问题 (公式 (85))。首先，将公式 (84) 中的 2 个不等式约束条件转换为 $[z]_+$ 函数：

$$\begin{cases} y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1-\xi_i \\ \xi_i\geqslant 0 \end{cases} \quad\Rightarrow\quad \xi_i=\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+ \tag{88}$$

最后，对公式 (84) 中的目标函数进行变形：

$$\begin{aligned}
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i = \\
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}} C\sum_{i=1}^{N}\xi_i + \frac{1}{2}\|\boldsymbol{w}\|^2 = \\
&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}} C\left(\sum_{i=1}^{N}\xi_i + \frac{1}{2C}\|\boldsymbol{w}\|^2\right) = \\
&\arg\min_{\boldsymbol{w},b}\sum_{i=1}^{N}\left[1-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right]_+ + \lambda\|\boldsymbol{w}\|^2
\end{aligned} \tag{89}$$

其中，$\lambda=\frac{1}{2C}$。因此，也可以将一般线性支持向量机的原始最优化问题 (公式 (84)) 转换为带正则项的合页损失函数的最小化问题 (公式 (85))。

综上所述，它们之间具有等价性。 □

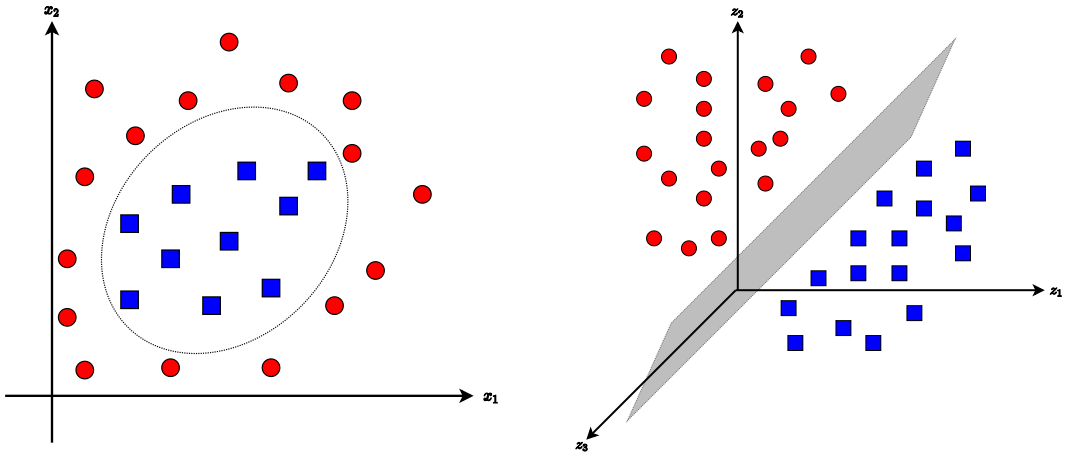


图 3-5: 非线性分类问题

3 非线性支持向量机

前面介绍的方法，适用于线性分类问题——存在分离超平面能够将数据集完全地或几乎完全地正确分类，即数据集是线性可分或基本线性可分的。

然而，在实际应用中，还有一类问题，并不存在能够将样本进行正确分类的分离超平面，或者说，分类决策函数形成的决策边界根本就不是决策超平面，而是决策超曲面。我们把这种问题称为非线性分类问题，它需要使用非线性模型才能很好地进行分类。

如图3-5所示，左子图展示了一个非线性分类问题，可以看出，它需要一条曲线而不是直线，才能将样本完全正确地进行分类；右子图展示了对输入空间进行非线性变换之后再进行线性分类的示意图——将样本从原始特征空间映射到更高维的特征空间，使得样本在高维特征空间中变得线性可分，即通过非线性变换，将非线性分类问题转换为线性分类问题。幸运的是，如果原始空间为有限维，那么一定存在高维的特征空间，使得样本线性可分。

因此，针对非线性分类问题，令 $\phi(\boldsymbol{x})$ 表示对原始特征向量 $\boldsymbol{x}$ 进行某种非线性变换，于是，在变换后的特征空间中，相应的分离超平面与分类决策函数分别为：

$$\begin{aligned} \boldsymbol{w}^T\phi(\boldsymbol{x}) + b = 0 \\ f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\phi(\boldsymbol{x}) + b) \end{aligned} \tag{90}$$

类似于一般线性支持向量机的原始最优化问题 (公式 (62))，得到非线性支持向量

机的原始最优化问题[17]：

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b) \geqslant 1-\xi_i \\ &\qquad \xi_i \geqslant 0 \quad i=1,2,\ldots,N \end{aligned} \tag{91}$$

类似于一般线性支持向量机的对偶最优化问题 (公式 (75))，得到非线性支持向量机的对偶最优化问题为：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\qquad 0 \leqslant \alpha_i \leqslant C \quad i=1,2,\ldots,N \end{aligned} \tag{92}$$

于是，理论上，求解非线性支持向量机的对偶最优化问题 (公式 (92))，便可以得到非线性支持向量机的最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^*。但是，求解该式，涉及到内积 $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ 的计算，它是样本 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 映射到高维特征空间后的内积。由于映射后特征空间的维度可能很高，甚至可能为无穷维。因此，直接计算 $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ 通常是很困难的。

为了解决这个问题，可以利用所谓的核函数和核技巧。构造的核函数满足条件：

$$\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) =< \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) >= K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{93}$$

上式表示，$\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$ 在变换后的特征空间上的内积 $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$，等于它们在原始特征空间中通过核函数 $K(\cdot,\cdot)$ 直接计算的结果 $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$。因此，如果存在这样的核函数，那么直接使用核函数的计算结果来代替对特征先执行非线性变换而后再执行内积计算的结果，这就是所谓的核技巧——不必直接计算高维或无穷维特征

[17] 当 $\xi_i = 0 (i=1,2,\ldots,N)$ 时，一般线性支持向量机便转变为线性可分支持向量机。因此，在此，对其不做讨论。

空间上的内积。于是，非线性支持向量机的对偶最优化问题 (公式 (92)) 变换为：

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\qquad 0 \leqslant \alpha_i \leqslant C \quad i = 1,2,\ldots,N \end{aligned} \tag{94}$$

求解上式，即可得到分离超平面的最优权值参数 $\boldsymbol{w}^*$ 和偏置参数 b^*：

$$\begin{cases} \boldsymbol{w}^* = \sum_{i=1}^{N}\alpha_i^* y_i \phi(\boldsymbol{x}_i) \\ b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i K(\boldsymbol{x}_i,\boldsymbol{x}_j) \quad \exists 0 < \alpha_j^* < C, (\boldsymbol{x}_j, y_j) \end{cases} \tag{95}$$

相应的分离超平面为：

$$\sum_{i=1}^{N}\alpha_i^* y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b^* = 0 \quad \Rightarrow \quad \sum_{i=1}^{N}\alpha_i^* y_i K(\boldsymbol{x},\boldsymbol{x}_i) + b^* = 0 \tag{96}$$

上述结果的意义十分明确，非线性支持向量机的分离超平面 (权值与偏置) 表现为实例点的核函数的线性组合，或表现为实例点的核函数展开，这种展开式被称为支持向量展式 (Support Vector Expansion)。

显然，如果非线性映射函数 $\phi(\boldsymbol{x})$ 是已知的，那么我们就可以写出核函数 $K(\cdot,\cdot)$ 的具体形式。但是，在实际问题中，通常并不能够知道函数 $\phi(\boldsymbol{x})$ 的具体形式，或者没有办法知道问题适合哪种 $\phi(\boldsymbol{x})$ 函数。核技巧在于，我们并不需要显式地定义函数 $\phi(\boldsymbol{x})$，且相关的核定理能够告诉我们，什么样的函数能够做核函数。

定理 3.1（正定核函数的充要条件）设 $K: \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ 为对称函数，则 $K(\cdot,\cdot)$ 为正定核函数的充要条件是，对任意 $\boldsymbol{x}_i \in \mathcal{X}$，$i = 1,2,\ldots,N$，$K(\cdot,\cdot)$ 对应的 Gram 矩阵：

$$K = [K(\boldsymbol{x}_i,\boldsymbol{x}_j)]_{N\times N} \tag{97}$$

是半正定矩阵。正定核函数 $K(\cdot,\cdot)$ 简称为正定核，核 Gram 矩阵简称为核矩阵。

该定理表明，只要一个对称函数所对应的核矩阵半正定，那么它就能够作为核函数使用。实际上，对于一个半正定核矩阵，总能够找到一个与之对应的映射

函数 ϕ。因此，任何一个核函数都隐式地定义了一个被称为“再生核希尔伯特空间”(Reproducing Kernel Hilbert Space, RKHS) 的特征空间。

对于一个具体的对称函数，检验它是否为正定核并不容易，原因在于，需要对任意有限的输入集，验证相应的 Gram 矩阵是否为半正定矩阵。因此，在实际应用中，通常使用已有的核函数。但是，需要注意的是，在不知道问题的特征映射函数 ϕ 的具体形式时，我们也就无法知道什么样的核函数是合适的。然而，可以依据问题的性质来确定核函数的具体形式。例如，对于文本数据，可以选用线性核；当情况不明时，可以先尝试高斯核。

常用的核函数如下：

- 线性核

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{98}$$

- 多项式核

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^d \tag{99}$$

  其中，$d \geqslant 1$ 表示多项式次数。

- 高斯核或 RBF 核

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) \quad \sigma > 0 \tag{100}$$

- 拉普拉斯核

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma}\right) \quad \sigma > 0 \tag{101}$$

- Sigmoid 核

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh\left(\beta \boldsymbol{x}_i^T \boldsymbol{x}_j + \theta\right) \quad \beta > 0, \theta < 0 \tag{102}$$

下面给出非线性支持向量机的对偶学习算法。

算法 **3.1**（非线性支持向量机的对偶学习算法）[18]

[18] 当核函数 $K(\cdot,\cdot)$ 为线性核函数时，非线性支持向量机的对偶算法就退化为一般线性支持向量机的对偶算法。

Input:

Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Output:

Decision Surface:$f(\boldsymbol{x}) = 0$

Algorithm:

Get $\boldsymbol{\alpha}^*$ by:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{N}\alpha_i \\ &s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\qquad 0 \leqslant \alpha_i \leqslant C \quad i = 1, 2, \ldots, N \end{aligned} \tag{103}$$

Get $\boldsymbol{w}^*$, b^* and $f(\boldsymbol{x}) = 0$ by:

$$\begin{cases} \boldsymbol{w}^* = \sum_{i=1}^{N}\alpha_i^* y_i \phi(\boldsymbol{x}_i) \\ b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \exists 0 < \alpha_j^* < C, (\boldsymbol{x}_j, y_j) \\ \sum_{i=1}^{N}\alpha_i^* y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b^* = 0 \end{cases} \tag{104}$$

4 序列最小最优化算法 (SMO)

本节讨论对偶最优化算法的具体实现问题。前文已经给出了相关的计算公式与基本算法。由于对偶最优化问题可形式化为凸二次规划问题，因而其解具有全局最优解，有多种最优化算法可以求解该问题。

但是，由于对偶最优化算法中的拉格朗日乘数个数与训练数据集中样本实例点的个数一样多，当训练样本容量很大时，常规算法将变得非常低效。下面介绍的序列最小最优化 (Sequential Minimal Optimization, SMO) 算法，可以有效地求解对偶最优化问题，由 Platt 于 1998 年提出。

SMO 算法是一种启发式算法，其基本思路是：

1. 如果所有的拉格朗日乘数 $\alpha_i (i = 1, 2, \ldots, N)$ 都满足 KKT 条件，那么整个

算法结束，即已经获得了 $\boldsymbol{\alpha}^*$，然后再依据公式 (104) 就可以计算出分离超平面[19]；

2. 否则，依据启发式信息选取 2 个拉格朗日乘数，并固定住其它乘数，然后针对这 2 个乘数构建一个二次规划 (子) 问题。此时，该二次规划子问题可以通过解析法求解，这极大地加速了求解过程。这 2 个乘数的选取方法是：选取违反 KKT 条件最严重的乘数作为第 1 个乘数；在确定了第 1 个乘数之后，第 2 个乘数可由约束条件自动地确定；

3. 上述过程迭代进行，直至步骤 (1) 的条件满足为止；

需要注意的是，每次之所以选择 2 个乘数变量一起进行优化，主要原因在于，对偶最优化问题中，存在约束条件：

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad i = 1, 2, \ldots, N \tag{105}$$

即 α_i 之间并不是完全独立的——例如，当优化 α_j 时，需要至少再相应地调整另一个乘数变量 $\alpha_k (k \neq j)$，才能继续满足上述约束条件。至于每次固定住除 α_j 和 α_k 之外的其它乘数变量，主要是为了降低求解难度，且便于使用解析法求解二次规划子问题，这样就可以大大地加快求解速度[20]。

SMO 算法包括 2 个主要部分：

1. 使用解析法，求解 2 个变量的二次规划子问题；

2. 使用启发式方法，选取 2 个变量；

4.1 解析法：求解 2 变量二次规划子问题

为讨论方便，约定 2 个待优化变量分别为 α_1 和 α_2，则其余变量 $\alpha_i (i = 3, 4, \ldots, N)$ 固定不变。于是，针对对偶最优化问题 (公式 (94)) 的 2 变量子问题，

[19] 原因是，KKT 条件的满足性是对偶最优化问题获得最优解 $\boldsymbol{\alpha}^*$ 的充分必要条件。

[20] 实际上，SMO 算法与坐标上升法有些类似。坐标上升法每次也只优化一个变量，执行一次一维最优化；然后再选取另一个分量，重复上述过程，直至目标函数达到局部最优点。SMO 算法每次选取一个主变量，次变量依据约束条件自动确定。虽然 SMO 算法需要同时更新 2 个变量，但是两个变量中只有一个自由变量。

可以将目标函数改写为：

$$
\begin{aligned}
&\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)-\sum_{i=1}^{N}\alpha_i \\
&=\arg\min_{\alpha_1,\alpha_2} \frac{1}{2}\alpha_1^2 y_1^2 K(\boldsymbol{x}_1,\boldsymbol{x}_1)+\frac{1}{2}\alpha_1\alpha_2 y_1 y_2 K(\boldsymbol{x}_1,\boldsymbol{x}_2)+\frac{1}{2}\alpha_1 y_1\sum_{j=3}^{N}\alpha_j y_j K(\boldsymbol{x}_1,\boldsymbol{x}_j)+ \\
&\qquad \frac{1}{2}\alpha_2\alpha_1 y_2 y_1 K(\boldsymbol{x}_2,\boldsymbol{x}_1)+\frac{1}{2}\alpha_2^2 y_2^2 K(\boldsymbol{x}_2,\boldsymbol{x}_2)+\frac{1}{2}\alpha_2 y_2\sum_{j=3}^{N}\alpha_j y_j K(\boldsymbol{x}_2,\boldsymbol{x}_j)+ \\
&\qquad \vdots \\
&\qquad \frac{1}{2}\alpha_N\alpha_1 y_N y_1 K(\boldsymbol{x}_N,\boldsymbol{x}_1)+\frac{1}{2}\alpha_N\alpha_2 y_N y_2 K(\boldsymbol{x}_N,\boldsymbol{x}_2)+\frac{1}{2}\alpha_N y_N\sum_{j=3}^{N}\alpha_j y_j K(\boldsymbol{x}_N,\boldsymbol{x}_j) \\
&\qquad -\alpha_1-\alpha_2-\sum_{i=3}^{N}\alpha_i \\
&=\arg\min_{\alpha_1,\alpha_2} \frac{1}{2}\alpha_1^2 K(\boldsymbol{x}_1,\boldsymbol{x}_1)+\frac{1}{2}\alpha_2^2 K(\boldsymbol{x}_2,\boldsymbol{x}_2)+\alpha_1\alpha_2 y_1 y_2 K(\boldsymbol{x}_1,\boldsymbol{x}_2)+ \\
&\qquad \alpha_1 y_1\sum_{j=3}^{N}\alpha_j y_j K(\boldsymbol{x}_1,\boldsymbol{x}_j)+\alpha_2 y_2\sum_{j=3}^{N}\alpha_j y_j K(\boldsymbol{x}_2,\boldsymbol{x}_j)-\alpha_1-\alpha_2
\end{aligned}
\tag{106}
$$

上式中，舍弃了不含待优化变量 α_1 和 α_2 的常数项，这不会影响到最优变量的位置。同样，将约束条件改写为：

$$
\sum_{i=1}^{N}\alpha_i y_i=0 \quad\Rightarrow\quad \alpha_1 y_1+\alpha_2 y_2=-\sum_{i=3}^{N}\alpha_i y_i \quad\Rightarrow\quad \alpha_1 y_1+\alpha_2 y_2=\zeta \tag{107}
$$

其中，$\zeta=-\sum_{i=3}^{N}\alpha_i y_i$ 为常数。为表达简便，令 $K_{11}=K(\boldsymbol{x}_1,\boldsymbol{x}_1)$、$K_{22}=K(\boldsymbol{x}_2,\boldsymbol{x}_2)$、$K_{12}=K(\boldsymbol{x}_1,\boldsymbol{x}_2)=K(\boldsymbol{x}_2,\boldsymbol{x}_1)$、$K_{1j}=K(\boldsymbol{x}_1,\boldsymbol{x}_j)=K(\boldsymbol{x}_j,\boldsymbol{x}_1)$ 及 $K_{2j}=K(\boldsymbol{x}_2,\boldsymbol{x}_j)=K(\boldsymbol{x}_j,\boldsymbol{x}_2)$，于是得到如下的 2 变量二次规划子问题：

$$
\begin{aligned}
&\min_{\alpha_1,\alpha_2} \frac{1}{2}\alpha_1^2 K_{11}+\frac{1}{2}\alpha_2^2 K_{22}+\alpha_1\alpha_2 y_1 y_2 K_{12}+\alpha_1 y_1\sum_{j=3}^{N}\alpha_j y_j K_{1j}+\alpha_2 y_2\sum_{j=3}^{N}\alpha_j y_j K_{2j} \\
&\qquad -\alpha_1-\alpha_2 \\
&s.t. \quad \alpha_1 y_1+\alpha_2 y_2=\zeta \\
&\qquad 0\leqslant\alpha_i\leqslant C \quad i=1,2
\end{aligned}
\tag{108}
$$

仔细观察上述最优化问题，可以发现，与前文分析一致，虽然待优化的变量有 2 个，但是自由变量只有 1 个，因为它们之间存在等式约束 $\alpha_1 y_1+\alpha_2 y_2=\zeta$。

因此，可以将该最优化问题转换为等价的单变量最优化问题。此处，不妨保留 α_2，消去 α_1。利用 $y_1^2=1$，对等式约束进行变形：

$$\begin{aligned}&\alpha_1 y_1+\alpha_2 y_2=\zeta \quad \Rightarrow\\ &\alpha_1=(\zeta-\alpha_2 y_2)y_1\end{aligned} \tag{109}$$

为描述方便，令目标函数为 $L(\alpha_2)$。将上式代入该目标函数：

$$\begin{aligned}L(\alpha_2)=&\frac{1}{2}\alpha_1^2K_{11}+\frac{1}{2}\alpha_2^2K_{22}+\alpha_1\alpha_2y_1y_2K_{12}+\alpha_1y_1\sum_{j=3}^{N}\alpha_jy_jK_{1j}+\alpha_2y_2\sum_{j=3}^{N}\alpha_jy_jK_{2j}\\&-\alpha_1-\alpha_2\\=&\frac{1}{2}(\zeta-\alpha_2y_2)^2K_{11}+\frac{1}{2}\alpha_2^2K_{22}+(\zeta-\alpha_2y_2)\alpha_2y_2K_{12}+(\zeta-\alpha_2y_2)\sum_{j=3}^{N}\alpha_jy_jK_{1j}\\&+\alpha_2y_2\sum_{j=3}^{N}\alpha_jy_jK_{2j}-(\zeta-\alpha_2y_2)y_1-\alpha_2\\=&\frac{1}{2}(\zeta-\alpha_2y_2)^2K_{11}+\frac{1}{2}\alpha_2^2K_{22}+\zeta\alpha_2y_2K_{12}-\alpha_2^2K_{12}+(\zeta-\alpha_2y_2)\sum_{j=3}^{N}\alpha_jy_jK_{1j}\\&+\alpha_2y_2\sum_{j=3}^{N}\alpha_jy_jK_{2j}-(\zeta-\alpha_2y_2)y_1-\alpha_2\end{aligned} \tag{110}$$

然后，分析 2 个约束条件：

$$\begin{aligned}&\alpha_1y_1+\alpha_2y_2=\zeta\\&0\leqslant\alpha_i\leqslant C\quad i=1,2\end{aligned} \tag{111}$$

由于 $y_i=\{+1,-1\}$，因而 y_1 与 y_2 只有 2 种组合：$y_1=-y_2$ 和 $y_1=y_2$。

当 $y_1=-y_2$ 时，α_1 与 α_2 之间的关系如图4-6所示。在该图中，横轴表示 α_1，纵轴表示 α_2，它们均服从不等式约束 $0\leqslant\alpha_i\leqslant C$，图中使用方框进行限定。此时，另一个等式约束可以变换为：

$$\begin{aligned}&\alpha_1y_1+\alpha_2y_2=\zeta \quad \Rightarrow\\&\alpha_2=\alpha_1+y_2\zeta\end{aligned} \tag{112}$$

它表示一条斜率为 $+1$ 的直线。在不等式约束的限制下，直线将被截断，在图中表现为 2 条红色线段之一。

为了降低目标函数 $L(\alpha_2)$(公式 (110)) 的求解难度，我们不使用拉格朗日优化函数将带约束的最优化问题转换为不带约束的最优化问题，而是直接求解目标函

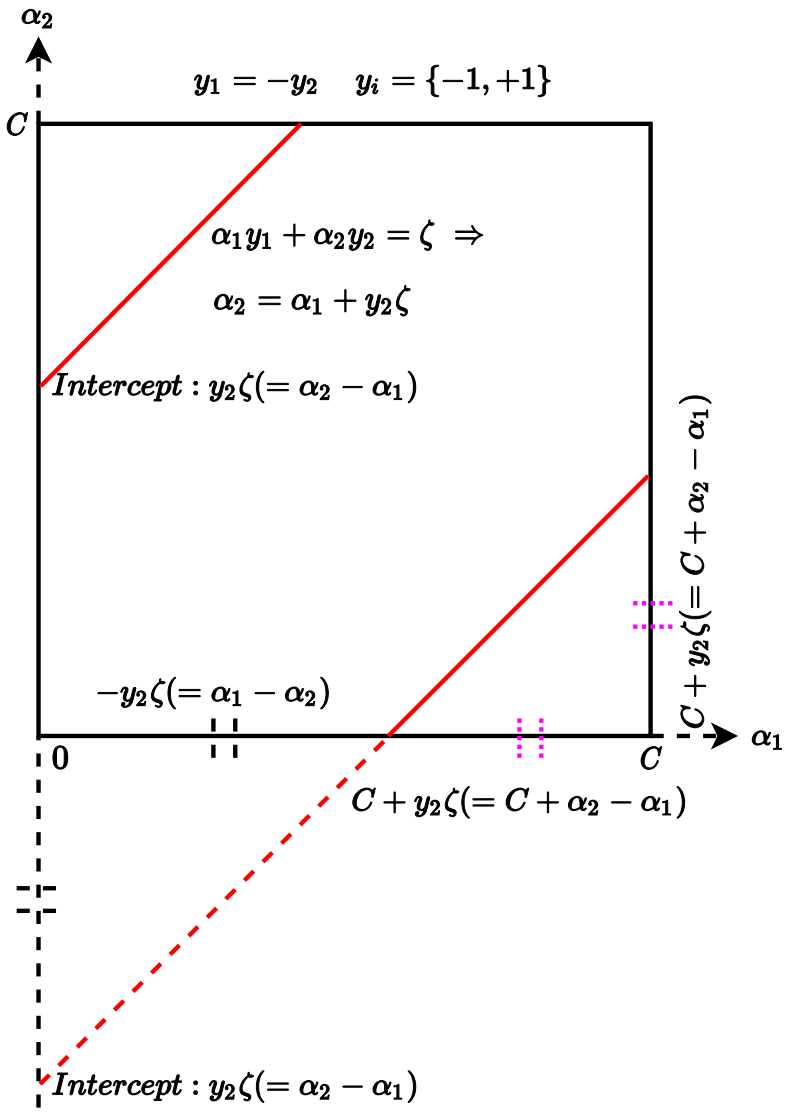


图 4-6: 当 $y_1 = -y_2$ 时，α_1 与 α_2 之间的关系

数 $L(\alpha_2)$；求解完毕之后，再将 α_2 进行截断处理，以满足约束条件；最后，再利用 α_1 与 α_2 之间的等式约束关系，求解出 α_1。下面，确定 α_2 的取值 (截断) 范围。

首先考虑 α_2 允许的最小值 L。在图4-6中，已经标示出了各主要线段的计算公式。对于上面的红色线段，α_2 允许的最小值为 $L = y_2\zeta = \alpha_2 - \alpha_1$；对于下面的红色线段，$\alpha_2$ 允许的最小值为 $L = 0$。将两者合并在一起，得到[21]：

$$L = \max(0, \alpha_2 - \alpha_1) \tag{113}$$

然后考虑 α_2 允许的最大值 H。对于上面的红色线段，α_2 允许的最大值为 $H = C$；对于下面的红色线段，α_2 允许的最大值为 $H = C + y_2\zeta = C + \alpha_2 - \alpha_1$。将两者合并在一起，得到[22]：

$$H = \min(C, C + \alpha_2 - \alpha_1) \tag{114}$$

当 $y_1 = y_2$ 时，α_1 与 α_2 之间的关系，如图4-7所示。它们表现为一条斜率为 -1 的直线。同样，在不等式约束的限制下，直线将被截断，在图中表现为 2 条红色线段之一。

类似地，可以确定 α_2 的取值 (截断) 范围，在图4-7中，也已经标示出了各主要线段的计算公式，下面直接给出相关结论[23]。α_2 允许的最小值 L 为：

$$L = \max(0, \alpha_2 + \alpha_1 - C) \tag{115}$$

α_2 允许的最大值 H 为：

$$H = \min(C, \alpha_2 + \alpha_1) \tag{116}$$

综上所述，α_2 需要满足的约束条件为 $L \leqslant \alpha_2 \leqslant H$。此外，为了体现迭代时间步的概念，需要为 α_2 与 α_1 添加时刻 t 标识。于是，将上述约束公式总结如下：

$$y_1 = -y_2, \begin{cases} L = \max(0, \alpha_2^{(t)} - \alpha_1^{(t)}) \\ H = \min(C, C + \alpha_2^{(t)} - \alpha_1^{(t)}) \end{cases} \tag{117}$$

$$y_1 = y_2, \begin{cases} L = \max(0, \alpha_2^{(t)} + \alpha_1^{(t)} - C) \\ H = \min(C, \alpha_2^{(t)} + \alpha_1^{(t)}) \end{cases} \tag{118}$$

[21]取 max 的原因在于，对于下面的红色线段，其截距为 $y_2\zeta = \alpha_2 - \alpha_1 \leqslant 0$，在方框的下端范围，需要截断。

[22]取 min 的原因与取 max 的原因类似，只不过，此时，对于上面的红色线段，$C + \alpha_2 - \alpha_1 \geqslant C$，在方框的上端范围，需要截断。

[23]α_2 截断公式的含义，可以按照前述方法，进行类似的分析。

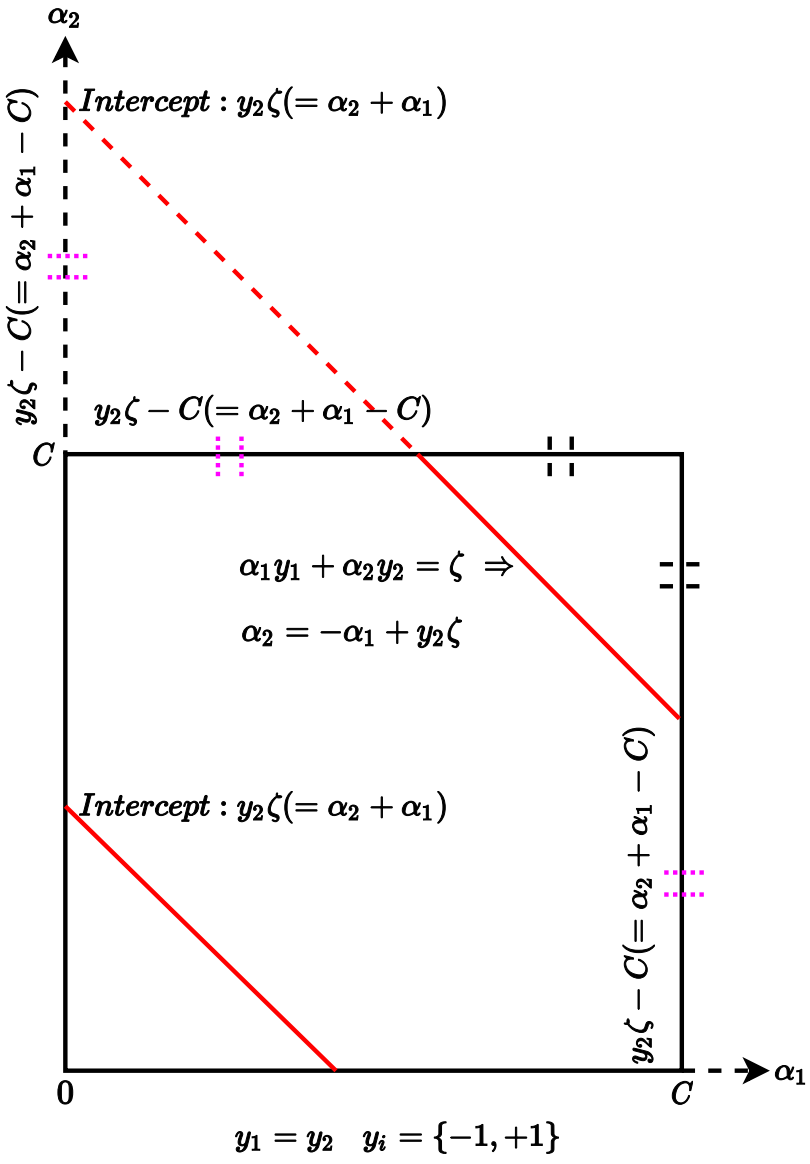


图 4-7: 当 $y_1 = y_2$ 时，α_1 与 α_2 之间的关系

$$\alpha_2^{(t+1)}=\begin{cases}H & \alpha_2^{(t+1)}>H\\ \alpha_2^{(t+1)} & L\leqslant\alpha_2^{(t+1)}\leqslant H\\ L & \alpha_2^{(t+1)}<L\end{cases}\tag{119}$$

下面，求解目标函数 $L(\alpha_2)$(公式 (110))。同样，将时间步 t 引入，得到：

$$\begin{aligned}L(\alpha_2^{(t+1)})=&\frac{1}{2}(\zeta-\alpha_2^{(t+1)}y_2)^2K_{11}+\frac{1}{2}(\alpha_2^{(t+1)})^2K_{22}+\zeta\alpha_2^{(t+1)}y_2K_{12}-(\alpha_2^{(t+1)})^2K_{12}+\\&(\zeta-\alpha_2^{(t+1)}y_2)\sum_{j=3}^{N}\alpha_j^{(t)}y_jK_{1j}+\alpha_2^{(t+1)}y_2\sum_{j=3}^{N}\alpha_j^{(t)}y_jK_{2j}-(\zeta-\alpha_2^{(t+1)}y_2)y_1-\alpha_2^{(t+1)}\end{aligned}\tag{120}$$

在求解的过程中，需要用到一些计算公式与中间量。其中，分离超平面为：

$$f(\boldsymbol{x})=\sum_{j=1}^{N}\alpha_j^{(t)}y_jK(\boldsymbol{x}_j,\boldsymbol{x})+b\tag{121}$$

$\alpha_1(\alpha_2)$ 对应的实例点 $\boldsymbol{x}_1(\boldsymbol{x}_2)$，其预测值与真实输出值 $y_1(y_2)$ 之间的差 $E_1(E_2)$ 分别为：

$$\begin{aligned}E_1=f(\boldsymbol{x}_1)-y_1=\left(\sum_{j=1}^{N}\alpha_j^{(t)}y_jK(\boldsymbol{x}_j,\boldsymbol{x}_1)+b\right)-y_1\\E_2=f(\boldsymbol{x}_2)-y_2=\left(\sum_{j=1}^{N}\alpha_j^{(t)}y_jK(\boldsymbol{x}_j,\boldsymbol{x}_2)+b\right)-y_2\end{aligned}\tag{122}$$

利用公式 (109) 中的 $\alpha_1^{(t)}=(\zeta-\alpha_2^{(t)}y_2)y_1$ 对 $\sum\limits_{j=3}^{N}\alpha_j^{(t)}y_jK_{1j}$ 进行变形：

$$\begin{aligned}\sum_{j=3}^{N}\alpha_j^{(t)}y_jK_{1j}&=f(\boldsymbol{x}_1)-\alpha_1^{(t)}y_1K_{11}-\alpha_2^{(t)}y_2K_{12}-b\\&=f(\boldsymbol{x}_1)-(\zeta-\alpha_2^{(t)}y_2)K_{11}-\alpha_2^{(t)}y_2K_{12}-b\end{aligned}\tag{123}$$

利用公式 (109) 中的 $\alpha_1^{(t)}=(\zeta-\alpha_2^{(t)}y_2)y_1$ 对 $\sum\limits_{j=3}^{N}\alpha_j^{(t)}y_jK_{2j}$ 进行变形：

$$\begin{aligned}\sum_{j=3}^{N}\alpha_j^{(t)}y_jK_{2j}&=f(\boldsymbol{x}_2)-\alpha_1^{(t)}y_1K_{12}-\alpha_2^{(t)}y_2K_{22}-b\\&=f(\boldsymbol{x}_2)-(\zeta-\alpha_2^{(t)}y_2)K_{12}-\alpha_2^{(t)}y_2K_{22}-b\end{aligned}\tag{124}$$

下面，正式求解 $\alpha_2^{(t+1)}$ 的解析解。首先，求解 $\frac{\partial L(\alpha_2^{(t+1)})}{\partial \alpha_2^{(t+1)}}$：

$$\begin{aligned}\frac{\partial L(\alpha_2^{(t+1)})}{\partial \alpha_2^{(t+1)}} &= -(\zeta - \alpha_2^{(t+1)} y_2) y_2 K_{11} + \alpha_2^{(t+1)} K_{22} + \zeta y_2 K_{12} - 2\alpha_2^{(t+1)} K_{12} \\ &\quad - y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{1j} + y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{2j} + y_1 y_2 - 1 \\ &= \alpha_2^{(t+1)} K_{11} - \zeta y_2 K_{11} + \alpha_2^{(t+1)} K_{22} + \zeta y_2 K_{12} - 2\alpha_2^{(t+1)} K_{12} \\ &\quad - y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{1j} + y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{2j} + y_1 y_2 - 1 \\ &= \alpha_2^{(t+1)} (K_{11} + K_{22} - 2K_{12}) - \zeta y_2 K_{11} + \zeta y_2 K_{12} \\ &\quad - y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{1j} + y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{2j} + y_1 y_2 - 1\end{aligned} \tag{125}$$

然后，令上述偏导数为 0，得到：

$$\begin{aligned}&\alpha_2^{(t+1)} (K_{11} + K_{22} - 2K_{12}) \\ &= \zeta y_2 K_{11} - \zeta y_2 K_{12} + y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{1j} - y_2 \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{2j} - y_1 y_2 + y_2^2 \\ &= y_2 \left(\zeta K_{11} - \zeta K_{12} + \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{1j} - \sum_{j=3}^{N} \alpha_j^{(t)} y_j K_{2j} - y_1 + y_2 \right)\end{aligned} \tag{126}$$

结合公式 (122)-公式 (124)，对上式继续进行变形：

$$\begin{aligned}&\alpha_2^{(t+1)} (K_{11} + K_{22} - 2K_{12}) \\ &= y_2 \Big(\zeta K_{11} - \zeta K_{12} + f(\boldsymbol{x}_1) - (\zeta - \alpha_2^{(t)} y_2) K_{11} - \alpha_2^{(t)} y_2 K_{12} - b \\ &\quad - f(\boldsymbol{x}_2) + (\zeta - \alpha_2^{(t)} y_2) K_{12} + \alpha_2^{(t)} y_2 K_{22} + b - y_1 + y_2 \Big) \\ &= y_2 \Big(f(\boldsymbol{x}_1) + \alpha_2^{(t)} y_2 K_{11} - \alpha_2^{(t)} y_2 K_{12} - f(\boldsymbol{x}_2) - \alpha_2^{(t)} y_2 K_{12} + \alpha_2^{(t)} y_2 K_{22} - y_1 + y_2 \Big) \\ &= y_2 \Big((f(\boldsymbol{x}_1) - y_1) - (f(\boldsymbol{x}_2) - y_2) + \alpha_2^{(t)} y_2 \left(K_{11} + K_{22} - 2K_{12} \right) \Big) \\ &= y_2 \left(E_1 - E_2 \right) + \alpha_2^{(t)} \left(K_{11} + K_{22} - 2K_{12} \right)\end{aligned} \tag{127}$$

得到：

$$\alpha_2^{(t+1)} = \alpha_2^{(t)} + \frac{y_2 \left(E_1 - E_2 \right)}{K_{11} + K_{22} - 2K_{12}} = \alpha_2^{(t)} + \frac{y_2 \left(E_1 - E_2 \right)}{\eta} \tag{128}$$

在得到 $\alpha_2^{(t+1)}$ 之后，再应用约束条件 (公式 (119))，即可得到本次迭代的 2 变量二次规划子问题 (公式 (108)) 的最优解 $\alpha_2^{(t+1)}$。

最后，使用 $\alpha_2^{(t+1)}$，计算出 $\alpha_1^{(t+1)}$：

$$\begin{cases} \alpha_1^{(t)} y_1 + \alpha_2^{(t)} y_2 = \zeta \\ \alpha_1^{(t+1)} y_1 + \alpha_2^{(t+1)} y_2 = \zeta \end{cases} \Rightarrow \begin{array}{l} \alpha_1^{(t)} y_1 + \alpha_2^{(t)} y_2 = \alpha_1^{(t+1)} y_1 + \alpha_2^{(t+1)} y_2 \quad \Rightarrow \\ \alpha_1^{(t)} + \alpha_2^{(t)} y_1 y_2 = \alpha_1^{(t+1)} + \alpha_2^{(t+1)} y_1 y_2 \quad \Rightarrow \\ \alpha_1^{(t+1)} = \alpha_1^{(t)} + y_1 y_2 \left(\alpha_2^{(t)} - \alpha_2^{(t+1)} \right) \end{array} \tag{129}$$

4.2 启发式：2 变量的选择方法

SMO 算法每次迭代时，选取 2 个变量进行优化，其中至少有 1 个变量违反了 KKT 条件。与前面一致，设其为 α_2。

具体而言，在时刻 t，检验训练样本点 $(\boldsymbol{x}_i, y_i)$ 是否满足如下的 KKT 条件：

$$\begin{aligned} &\alpha_i^{(t)} = 0 \Leftrightarrow y_i f(\boldsymbol{x}_i) \geqslant 1 \\ &0 < \alpha_i^{(t)} < C \Leftrightarrow y_i f(\boldsymbol{x}_i) = 1 \\ &\alpha_i^{(t)} = C \Leftrightarrow y_i f(\boldsymbol{x}_i) \leqslant 1 \end{aligned} \tag{130}$$

其中，$f(\boldsymbol{x}_i) = \sum_{j=1}^{N} \alpha_j^{(t)} y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + b$。在正常情况下，它们分别对应如下的实例点：

- 分类正确的内部实例点，对分离超平面的形成不起作用；
- 正负间隔超平面上的实例点 (松弛变量 $\xi_i = 0$)，(硬) 支持向量；
- 松弛变量 $\xi_i > 0$ 的实例点，一般 (软) 支持向量 (有 3 种情形，公式 (78))；

在算法实现时，设置一定的检验容忍度 (Tolerance)，即允许检验操作在一定的范围 ϵ 内成立即可。根据优先级，算法首先检验 $0 < \alpha_i^{(t)} < C$ 的实例样本点，即正负间隔超平面上的支持向量；如果都满足，那么再检验其它类型的实例样本点。

在变量 α_2 确定的情况下，再选取变量 α_1。选取变量 α_1 的原则是，确保 α_2 能够得到最大的更新——依据公式 (128)，就是找到使得 $|E_1 - E_2|$ 最大化的 α_1。在算法实现时，为了提高效率，可以将实例点的 E_i 值保存起来。

在特殊情况下，如果上述选取 α_1 的方法不起作用，那么还可以采取随机方法继续选取。具体的执行步骤，请阅读下面的 SMO 算法描述。

4.3 E 与 b 值的更新

在每次迭代结束之后，2 个变量已经得到了更新。此时，需要更新 b 和 E 值。

首先，更新 b 值，结合时间步 t，将时刻 t 时的 b 标记为 $b^{(t)}$。那么在时刻 $t+1$，$b^{(t+1)}$ 如何更新呢？

$b^{(t+1)}$ 的更新依赖于 $\alpha_1^{(t+1)}$ 和 $\alpha_2^{(t+1)}$ 及其对应的 $b_1^{(t+1)}$ 和 $b_2^{(t+1)}$，其更新规则如下：

- 如果 $0<\alpha_1^{(t+1)}<C$ 或 $0<\alpha_2^{(t+1)}<C$，表明它们对应的 2 个实例点中至少有一个是间隔超平面上的支持向量，那么 $b^{(t+1)}=b_1^{(t+1)}$ 或 $b^{(t+1)}=b_2^{(t+1)}$。如果 $0<\alpha_1^{(t+1)}<C$ 和 $0<\alpha_2^{(t+1)}<C$ 同时成立，表明它们对应的 2 个实例点都是间隔超平面上的支持向量，那么 $b_1^{(t+1)}=b_2^{(t+1)}$，$b^{(t+1)}$ 简单地取其中一个值即可，所列规则依然适用；
- 否则，令 $b^{(t+1)}=\frac{b_1^{(t+1)}+b_2^{(t+1)}}{2}$；

不论上述哪种情形，$b_1^{(t+1)}$ 和 $b_2^{(t+1)}$ 都按如下相同的方式计算。下面，以 $b_1^{(t+1)}$ 的计算为例，$b_2^{(t+1)}$ 则直接给出计算公式。依据间隔超平面上的支持向量公式，有：

$$\begin{aligned}
&\sum_{i=1}^{N}\alpha_i y_i K_{1i}+b_1^{(t+1)}=y_1 \quad \Rightarrow \\
&b_1^{(t+1)}=y_1-\sum_{i=1}^{N}\alpha_i y_i K_{1i} \\
&\quad =y_1-\sum_{i=3}^{N}\alpha_i^{(t)} y_i K_{1i}-\alpha_1^{(t+1)} y_1 K_{11}-\alpha_2^{(t+1)} y_2 K_{12}
\end{aligned} \tag{131}$$

然后，再根据 $E_1^{(t)}$ 的定义有：

$$\begin{aligned}
E_1^{(t)}&=f(\boldsymbol{x}_1)-y_1=\sum_{i=1}^{N}\alpha_i^{(t)} y_i K_{1i}+b^{(t)}-y_1 \\
&=-y_1+\sum_{i=3}^{N}\alpha_i^{(t)} y_i K_{1i}+b^{(t)}+\alpha_1^{(t)} y_1 K_{11}+\alpha_2^{(t)} y_2 K_{12}
\end{aligned} \tag{132}$$

得到：

$$y_1-\sum_{i=3}^{N}\alpha_i^{(t)} y_i K_{1i}=-E_1^{(t)}+b^{(t)}+\alpha_1^{(t)} y_1 K_{11}+\alpha_2^{(t)} y_2 K_{12} \tag{133}$$

将上式代入公式 (131) 中，得到：

$$\begin{aligned}
b_1^{(t+1)}&=-E_1^{(t)}+b^{(t)}+\alpha_1^{(t)} y_1 K_{11}+\alpha_2^{(t)} y_2 K_{12}-\alpha_1^{(t+1)} y_1 K_{11}-\alpha_2^{(t+1)} y_2 K_{12} \\
&=-E_1^{(t)}+y_1 K_{11}(\alpha_1^{(t)}-\alpha_1^{(t+1)})+y_2 K_{12}(\alpha_2^{(t)}-\alpha_2^{(t+1)})+b^{(t)}
\end{aligned} \tag{134}$$

类似地，可以得到：

$$b_2^{(t+1)} = -E_2^{(t)} + y_1 K_{12}(\alpha_1^{(t)} - \alpha_1^{(t+1)}) + y_2 K_{22}(\alpha_2^{(t)} - \alpha_2^{(t+1)}) + b^{(t)} \tag{135}$$

在得到 $b^{(t+1)}$ 之后，需要更新所有样本点的 E 值：

$$E_i^{(t+1)} = f(\boldsymbol{x}_i) - y_i = \sum_{j=1}^{N} \alpha_j^{(t+1)} y_j K_{ij} + b^{(t+1)} - y_i \tag{136}$$

下面，给出 SMO 算法。

算法 **4.1**（*SMO* 算法）

Input:

Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$ or ($\boldsymbol{X}$ and $\boldsymbol{Y}$)

Regularization Parameter: C

Numerical Tolerance: TOLERANCE

Output:

Lagrange Multipliers for Solution: $\boldsymbol{\alpha}$

Threshold for Solution: b

Algorithm:

```
Initialize α_i = 0, i = 1, 2, ..., N
Initialize b = 0
numChanged = 0
examineAll = True
while numChanged > 0 or examineAll:
    numChanged = 0
    if examineAll:
        for (x_i, y_i) in X, Y:
            numChanged += examineSample(i, X, Y)
    else:
        for (x_i, y_i) in X, Y with α_i ≠ 0 and α_i ≠ C:
            numChanged += examineSample(i, X, Y)
    if examineAll:
        examineAll = False
    else if numChanged == 0:
        examineAll = True
```

def takeStep(i_1, i_2, $\boldsymbol{X}$, $\boldsymbol{Y}$):

if $i_1 == i_2$:

return False

$\boldsymbol{x}_1 = \boldsymbol{X}[i_1]$, $\boldsymbol{x}_2 = \boldsymbol{X}[i_2]$

$y_1 = \boldsymbol{Y}[i_1]$, $y_2 = \boldsymbol{Y}[i_2]$

$E_1 = f(\boldsymbol{x}_1) - y_1$, $E_2 = f(\boldsymbol{x}_2) - y_2$

α_1, α_2 =Lagrange Multiplier for $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

Calculate L and H

if $L == H$:

return False

Calculate K_{11},K_{12},K_{22}

$\eta = K_{11} + K_{22} - 2K_{12}$

if $\eta \leqslant 0$:

return False

$\alpha_2^{(t+1)} = \alpha_2^{(t)} + \frac{y_2(E_1 - E_2)}{\eta}$

Get $\alpha_2^{(t+1)}$ by $L \leqslant \alpha_2^{(t+1)} \leqslant H$

$\alpha_1^{(t+1)} = \alpha_1^{(t)} + y_1 y_2 \left(\alpha_2^{(t)} - \alpha_2^{(t+1)}\right)$

$b_1^{(t+1)} = -E_1^{(t)} + y_1 K_{11}(\alpha_1^{(t)} - \alpha_1^{(t+1)}) + y_2 K_{12}(\alpha_2^{(t)} - \alpha_2^{(t+1)}) + b^{(t)}$

$b_2^{(t+1)} = -E_2^{(t)} + y_1 K_{12}(\alpha_1^{(t)} - \alpha_1^{(t+1)}) + y_2 K_{22}(\alpha_2^{(t)} - \alpha_2^{(t+1)}) + b^{(t)}$

if $0 < \alpha_1^{(t+1)} < C$:

$b^{(t+1)} = b_1^{(t+1)}$

elif $0 < \alpha_2^{(t+1)} < C$:

$b^{(t+1)} = b_2^{(t+1)}$

else:

$b^{(t+1)} = \frac{b_1^{(t+1)} + b_2^{(t+1)}}{2}$

$E_i^{(t+1)} = f(\boldsymbol{x}_i) - y_i = \sum_{j=1}^{N} \alpha_j^{(t+1)} y_j K_{ij} + b^{(t+1)} - y_i, i = 1, 2, \ldots, N$

return True

需要注意的是，由于：

$$\begin{aligned}\|\phi(\boldsymbol{x}_1)-\phi(\boldsymbol{x}_2)\|^2 &= (\phi(\boldsymbol{x}_1)-\phi(\boldsymbol{x}_2))^T(\phi(\boldsymbol{x}_1)-\phi(\boldsymbol{x}_2)) \\ &= \phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_1)-2\phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_2)+\phi(\boldsymbol{x}_2)^T\phi(\boldsymbol{x}_2) \\ &= K(\boldsymbol{x}_1,\boldsymbol{x}_1)+K(\boldsymbol{x}_2,\boldsymbol{x}_2)-2K(\boldsymbol{x}_1,\boldsymbol{x}_2) \\ &= K_{11}+K_{22}-2K_{12}=\eta\end{aligned} \tag{137}$$

上式表明，$\eta \geqslant 0$。此外，由于 η 为 α_2 更新公式 (128) 中的分母项，故 $\eta > 0$。因此，函数 takeStep 对 η 进行了相应的处理：如果 $\eta \leqslant 0$，则直接返回，不进行任何更新。

```
def examineSample(i2, X, Y):
    x2 = X[i2]
    y2 = Y[i2]
    α2 =Lagrange Multiplier for x2
    E2 =Error for x2
    tol=E2y2
    if (tol < -TOLERANCE and α2 < C) or (tol > TOLERANCE and α2 > 0):
        Get best x1 with 0 < α1 < C and the largest |E1 − E2|, i1 index for x1
        if takeStep(i1,i2,X,Y):
            return 1
        Loop over the rest samples:
            Get x1, i1 index for x1
            if takeStep(i1,i2,X,Y):
                return 1
    return 0
```

在函数 examineSample 中，tol=E_2y_2 表示：

$$E_2y_2 = (f(\boldsymbol{x}_2)-y_2)\,y_2 = y_2f(\boldsymbol{x}_2)-1 \tag{138}$$

在正常情况下，如果 $0 < \alpha_2 < C$，那么 $y_2f(\boldsymbol{x}_2) = 1$，则 $E_2y_2 = 0$ 成立。在算法实现时，允许在一定的容忍度内，该条件成立，因此设置 TOLERANCE 为 1 个较小的数 (例如 0.001)。在阅读该部分伪代码时，试着与公式 (130) 中的 3 个 KKT 条件结合在一起进行分析，便于理解。

5 支持向量机实验

SMO 算法的实现

```
In [1]: import numpy as np
        import random
```

```
In [2]: class SVM:
            def __init__(self, C, tolerance, epsilon, max_step, kernel, sigma = 0.5):
                self.C = C # 参数 C
                self.tolerance = tolerance # 容错率
                self.epsilon = epsilon # 拉格朗日乘数更新的最小比率
                self.max_step = max_step # 训练次数
                self.kernel = kernel # 核函数类型: 'linear'、'rbf'
                self.sigma = sigma

            # 定义核函数
            #sample: 单样本, 行向量
            def Kernel(self, X, sample):
                if self.kernel == 'linear':
                    result = np.dot(X, sample.T).reshape(-1, 1)
                elif self.kernel == 'rbf':
                    result = np.exp(-np.linalg.norm(X - sample, ord = 2, axis = 1,
                     keepdims = True)**2 / (2*self.sigma**2))
                else:
                    result = None
                    print('Non-legal kernel')
                return result

            # 计算 Gram 矩阵
            def Gram(self, X):
                self.K = np.zeros((self.N, self.N))
                for i, x in enumerate(X):
                    self.K[i, :] = self.Kernel(X, x).reshape(1, -1)

            # 更新 alpha1、alpha2、b、E
            # 返回: 成功更新, True; 否则, False
            def update(self, index1, index2, X, Y):
                if (index1 == index2): # 两个相同的拉格朗日乘数, 无更新必要
                    return False

                # 取出乘数 alpha
                alpha1 = self.alpha[index1][0]
                alpha2 = self.alpha[index2][0]
                # 取出输入向量
                x1 = X[index1].reshape(-1, 1)
                x2 = X[index2].reshape(-1, 1)
                # 取出标签
                y1 = Y[index1][0]
                y2 = Y[index2][0]
```

```
        # 取出 E
        E1 = self.E[index1][0]
        E2 = self.E[index2][0]
        # 计算 alpha 的上界 H 与下界 L
        if (y1 == y2):
            L = max(0, alpha2 + alpha1 - self.C)
            H = min(self.C, alpha2 + alpha1)
        else:
            L = max(0, alpha2 - alpha1)
            H = min(self.C, self.C + alpha2 - alpha1)
        if (L == H): # 相同的下界与上界，无更新必要
            return False
        # 计算 eta
        k11 = self.K[index1, index1]
        k22 = self.K[index2, index2]
        k12 = self.K[index1, index2]
        eta = k11 + k22 - 2 * k12
        if eta <= 0: # 不更新
            return False
        alpha2_new = alpha2 + y2 * (E1 - E2) / eta # 计算 alpha2 的新值
        alpha2_new = min(max(alpha2_new, L), H) # 对 alpha2 进行 [L, H] 裁剪
        if abs(alpha2_new - alpha2) < self.epsilon * (alpha2_new + alpha2 +
         self.epsilon): # 无更新必要
            return False
        # 计算 alpha1 的新值
        alpha1_new = alpha1 + y1 * y2 * (alpha2 - alpha2_new)
        # 计算 b1 和 b2, 并且更新 b
        b1 = -E1 - y1 * k11 * (alpha1_new - alpha1) - y2 * k12 *
         (alpha2_new - alpha2) + self.b
        b2 = -E2 - y1 * k12 * (alpha1_new - alpha1) - y2 * k22 *
         (alpha2_new - alpha2) + self.b
        if alpha1_new > 0 and alpha1_new < self.C:
            self.b = b1
        elif alpha2_new > 0 and alpha2_new < self.C:
            self.b = b2
        else:
            self.b = (b1 + b2) / 2
        # 保存 alpha1_new 和 alpha2_new, 并确保约束 >=0
        self.alpha[index1][0] = max(0, alpha1_new)
        self.alpha[index2][0] = max(0, alpha2_new)
        # 更新 E
        self.E = np.dot((self.alpha * Y).T, self.K).reshape(-1, 1) +
         self.b - Y
        return True

    # 找出最佳的 2 个样本点, 并启动更新
    #index: 给定的第 2 个样本点索引
    def MatchTwoSamples(self, index, X, Y):
```

```
            y2 = Y[index][0]
            alpha2 = self.alpha[index][0]
            E2 = self.E[index][0]
            # 容忍度: E*y=(g(x)-y)y=g(x)y-y^2=g(x)y-1, g(x) 是 y 的实际输出值,
            # 两者越相似越好, 此时 tol 接近 0
            tol = E2 * y2
            if ((tol < -self.tolerance and alpha2 < self.C) or
             (tol > self.tolerance and alpha2 > 0)): # 不符合 KKT 条件, 需调整
                # 采取启发式方法, 确定另一个乘数 alpha1, 找出满足 0<alpha1<C 的点
                # 找出所有非 0 的 alpha (索引)
                list1 = np.nonzero(self.alpha)[0].tolist()
                # 找出所有非 C 的 alpha (索引)
                list2 = np.nonzero(self.alpha - self.C)[0].tolist()
                result = list(set(list1) & set(list2))
                if len(result) > 1: # 至少有 2 个及以上的候选点, 选择一个最好的
                    max_diff = -1
                    max_index = -1
                    for k in result:
                        diff = abs(self.E[k][0] - E2)
                        if diff > max_diff:
                            max_diff = diff
                            max_index = k
                    if self.update(max_index, index, X, Y): # 更新这 2 个乘数
                        return True
                    # 如果没有成功更新, 则随机产生一个位置, 逐个尝试每个乘数
                    random_index = random.randint(0, len(result)-1)
                    for k in range(random_index, len(result)):
                        if self.update(k, index, X, Y): # 更新这 2 个乘数
                            return True
                    for k in range(0, random_index):
                        if self.update(k, index, X, Y): # 更新这 2 个乘数
                            return True
                # 从所有可能的乘数中, 随机选择一个
                random_index = random.randint(0, self.N-1)
                if self.update(random_index, index, X, Y): # 更新这 2 个乘数
                    return True
            return False

        def fit(self, X, Y):
            self.X = X # 在使用核函数的情况下, 需要保存数据集
            self.Y = Y
            self.N = len(X)
            self.b = 0
            self.alpha = np.zeros((self.N, 1))
            self.Gram(X) # 计算 Gram 矩阵
            self.E = np.dot((self.alpha * Y).T, self.K).reshape(-1, 1) +
             self.b - Y # 初始化 E
```

```
        # 开始训练
        self.paras = [] # 存放中间权值的结果，用于生成动画
        for step in range(self.max_step):
            numChanged = 0
            everyone = True # 确保第一次要全局遍历
            while numChanged > 0 or everyone:
                numChanged = 0
                if everyone:
                    for k in range(self.N):
                        numChanged += self.MatchTwoSamples(k, X, Y)
                        if self.kernel == 'linear':
                            # 保存权值
                            self.paras.append((self.b, *np.dot(X.T,
                             (self.alpha * Y)).reshape(1, -1).tolist()))
                        else: # 其它的核函数
                            # 在使用其它核函数的情况下，只能保存每个样本对应的
                            # 乘数 alpha
                            self.paras.append((self.b, self.alpha))
                else:
                    for k in range(self.N):
                        if self.alpha[k] == 0 or self.alpha[k] == self.C:
                            continue
                        numChanged += self.MatchTwoSamples(k, X, Y)
                        if self.kernel == 'linear':
                            # 保存权值
                            self.paras.append((self.b, *np.dot(X.T,
                             (self.alpha * Y)).reshape(1, -1).tolist()))
                        else: # 其它的核函数
                            # 在使用其它核函数的情况下，只能保存每个样本对应的
                            # 乘数 alpha
                            self.paras.append((self.b, self.alpha))
                if everyone:
                    everyone = False
                elif numChanged == 0:
                    everyone = True
        if self.kernel == 'linear': # 计算 W
            self.W = np.dot(X.T, (self.alpha * Y))
            print(self.b, self.W)
        print('SVM training completed!')

    def predict(self, X):
        result = []
        for x in X:
            z = np.dot((self.alpha * self.Y).T, self.Kernel(self.X, x)) +
             self.b
            result.append(1 if z >= 0 else -1)
        return result
```

载入预存的鸢尾花数据集

```
In [3]: from sklearn.model_selection import train_test_split
        import matplotlib.pyplot as plt
        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'

In [4]: iris_npz = np.load('iris.npz')
        data = iris_npz['data']
        X = iris_npz['X']
        Y = iris_npz['Y']
        # 转换成适合 SVM 处理的标签
        Y[:50] = 1
        Y[50:] = -1

In [5]: XTRAIN, XTEST, YTRAIN, YTEST = train_test_split(X, Y, test_size = 0.25)
```

使用训练数据训练“线性核 SVM”

```
In [6]: svm = SVM(1.0, 0.001, 0.01, 2, 'linear')
        svm.fit(XTRAIN, YTRAIN.reshape(-1, 1))

5.17987952631 [[-2.26813742]
 [ 2.24382937]]
SVM training completed!
```

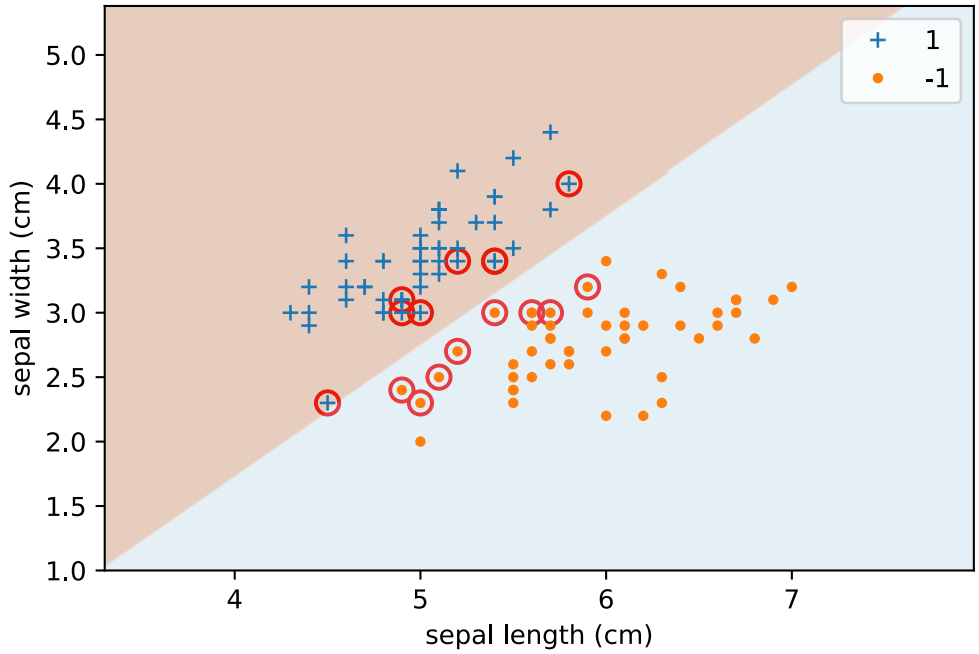
简单的预测测试

```
In [7]: XSAMPLES = np.array([(5.5, 2.8), (5.5, 4.0), (4.5, 3.5), (6.5, 2.5)])
        results = svm.predict(XSAMPLES)
        print(results)

[-1, 1, 1, -1]
```

绘制 SVM 分类结果

```
In [8]: plt.plot(data[:50, 0], data[:50, 1], '+', label='1')
        plt.plot(data[50:100, 0], data[50:100, 1], '.', label='-1')

        # 绘制支持向量
        for k, alpha in enumerate(svm.alpha):
            if alpha > 0.1:
                plt.scatter(XTRAIN[k][0], XTRAIN[k][1], s = 80, c = 'none',
                 linewidth=1.5, edgecolor = 'red')
        # 生成绘图用的网格
        x0_min, x0_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        x1_min, x1_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
xx0, xx1 = np.meshgrid(np.arange(x0_min, x0_max, 0.02),
                       np.arange(x1_min, x1_max, 0.02))

z = np.array(svm.predict(np.c_[xx0.ravel(), xx1.ravel()]))
z = z.reshape(xx0.shape)
plt.contourf(xx0, xx1, z, cmap = plt.cm.Paired, alpha = 0.3)

plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.legend()
plt.savefig('SVM_OUTPUT1.pdf', bbox_inches='tight')
```



绘制训练动画

```
In [9]: import matplotlib.animation as animation
        from IPython.display import HTML
        # 动画播放，如果是%matplotlib inline，则会嵌入图片
        %matplotlib notebook

        fig = plt.figure()
        ax = plt.axes(xlim=(3, 7), ylim=(2, 5))

        ax.plot(data[:50, 0], data[:50, 1], '+', label='1')
        ax.plot(data[50:100, 0], data[50:100, 1], '.', label='-1')
        ax.set_xlabel('sepal length (cm)')
        ax.set_ylabel('sepal width (cm)')
```

```
        ax.set_title('Support Vector Machine')
        ax.legend(loc = 'upper left')

        line, = ax.plot([], [], lw = 2)
        def update(para):
            b = para[0]
            W0 = para[1][0]
            W1 = para[1][1]
            if W1 == 0:
                return

            X_points = np.linspace(4, 7,10)
            Y_points = -(W0 * X_points + b)/W1
            line.set_data(X_points, Y_points)
        ani = animation.FuncAnimation(fig, update, svm.paras)
        # 视频播放
        # 安装 FFMPEG 库
        # 在 anaconda 环境下，需要执行“conda install -c conda-forge ffmpeg”
        # 需要 FFMPEG.exe，直接下载 BUILD，解压到目录，例如 E:\ffmpeg\bin
        # 然后在环境变量 PATH 中添加“E:\ffmpeg\bin”，路径名称需要根据实际情况做相应的调整
        HTML(ani.to_html5_video())
```

```
In [10]: ani.save('Fitting-duxiaoqin.mp4')
```

使用鸢尾花测试数据集进行测试

```
In [11]: from collections import Counter

         results = svm.predict(XTEST)
         scores = (results == YTEST)
         scores = [score for score in scores]
         print('Accuracy = {:3.2f}%'.format(Counter(scores)[True]/len(YTEST) * 100))
```

```
Accuracy = 100.00%
```

使用训练数据训练“RBF 核 SVM”

```
In [12]: svm = SVM(1.0, 0.001, 0.01, 2, 'rbf', sigma = 1.0)
         svm.fit(XTRAIN, YTRAIN.reshape(-1, 1))
```

```
SVM training completed!
```

简单的预测测试

```
In [13]: XSAMPLES = np.array([(5.5, 2.8), (5.5, 4.0), (4.5, 3.5), (6.5, 2.5)])
         results = svm.predict(XSAMPLES)
         print(results)
```

```
[-1, 1, 1, -1]
```

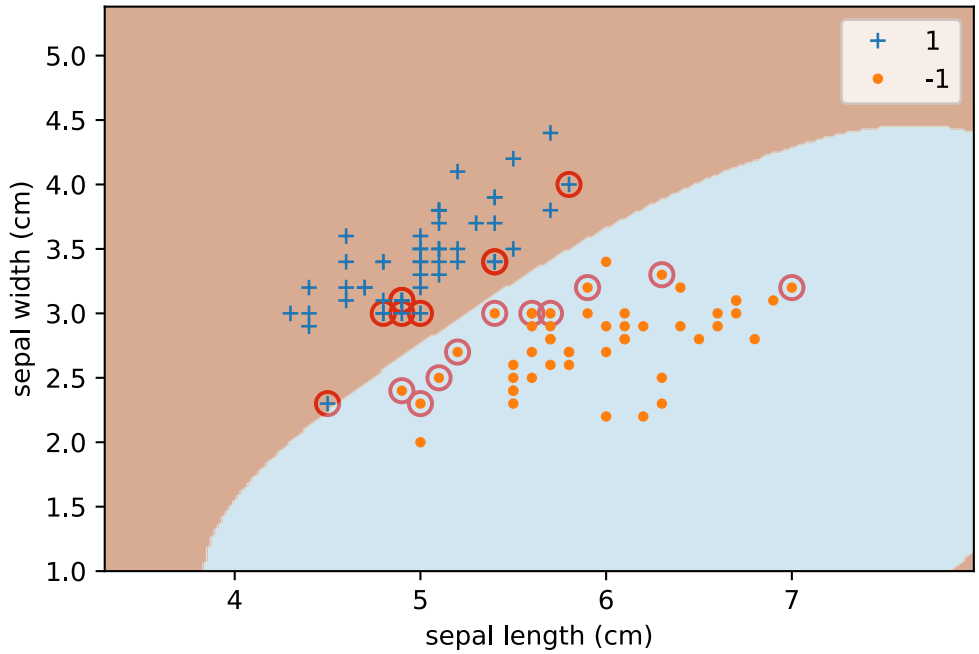
使用鸢尾花测试数据集进行测试

```
In [14]: results = svm.predict(XTEST)
         scores = (results == YTEST)
         scores = [score for score in scores]
         print('Accuracy = {:3.2f}%'.format(Counter(scores)[True]/len(YTEST) * 100))
```

```
Accuracy = 100.00%
```

绘制 SVM 分类结果

```
In [15]: %matplotlib inline
         %config InlineBackend.figure_format = 'svg'
         plt.plot(data[:50, 0], data[:50, 1], '+', label='1')
         plt.plot(data[50:100, 0], data[50:100, 1], '.', label='-1')

         # 绘制支持向量
         for k, alpha in enumerate(svm.alpha):
             if alpha > 0.1:
                 plt.scatter(XTRAIN[k][0], XTRAIN[k][1], s = 80, c = 'none',
                  linewidth=1.5, edgecolor = 'red')

         # 生成绘图用的网格
         x0_min, x0_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         x1_min, x1_max = X[:, 1].min() - 1, X[:, 1].max() + 1

         xx0, xx1 = np.meshgrid(np.arange(x0_min, x0_max, 0.02),
                                np.arange(x1_min, x1_max, 0.02))

         z = np.array(svm.predict(np.c_[xx0.ravel(), xx1.ravel()]))
         z = z.reshape(xx0.shape)
         plt.contourf(xx0, xx1, z, cmap = plt.cm.Paired, alpha = 0.5)

         plt.xlabel('sepal length (cm)')
         plt.ylabel('sepal width (cm)')
         plt.xlim(xx0.min(), xx0.max())
         plt.ylim(xx1.min(), xx1.max())
         plt.legend()
         plt.savefig('SVM_OUTPUT2.pdf', bbox_inches='tight')
```



使用 sklearn.svm.SVC 进行分类
使用说明
参数：

- C：C-SVC 的惩罚参数 C? 默认值是 1.0

C 越大，相当于惩罚松弛变量，使得松弛变量向 0 靠拢，即对误分类的惩罚增大，准确率会提高，但泛化能力将减弱；C 值小，对误分类的惩罚减小，允许一定的容错，但是泛化能力将增强；

- kernel：核函数，默认是 rbf，可以是 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'
  - 线性：u'v
  - 多项式：(gammau'v + coef0)^degree
  - RBF 函数：exp(-gamma|u-v|^2)
  - sigmoid：tanh(gammau'v + coef0)
- degree ：多项式 poly 函数的维度，默认是 3，选择其他核函数时会被忽略。
- gamma：'rbf','poly' 和 'sigmoid' 的核函数参数。默认是'auto'，则会选择 1/n_features
- coef0：核函数的常数项。对于 'poly' 和 'sigmoid' 有用。
- tol：停止训练的误差值大小，默认为 1e-3

主要的调节参数：C、kernel、degree、gamma、coef0。
使用数据集训练 ''线性核 SVM''

```
In [16]: from sklearn.svm import SVC
         clf = SVC(kernel = 'linear')
         clf.fit(XTRAIN, YTRAIN)
```

```
Out[16]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

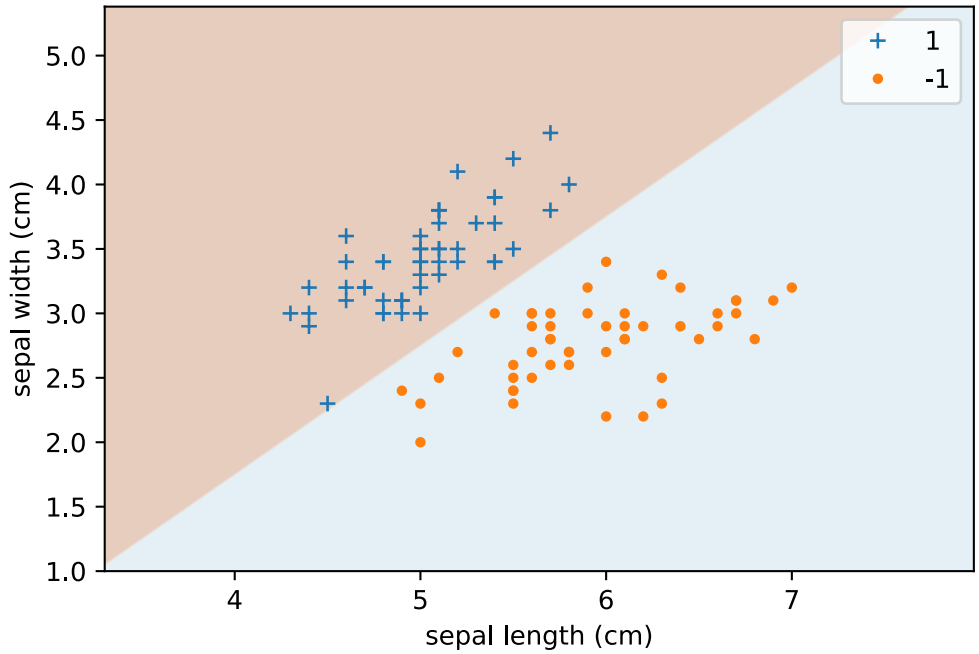
训练结果：分类决策面

```
In [17]: print(clf.coef_, clf.intercept_)
```

```
[[-2.22261578  2.22261101]] [ 5.00093675]
```

绘制 SVM 分类结果

```
In [18]: %matplotlib inline
         %config InlineBackend.figure_format = 'svg'

         plt.plot(data[:50, 0], data[:50, 1], '+', label='1')
         plt.plot(data[50:100, 0], data[50:100, 1], '.', label='-1')

         # 生成绘图用的网格
         x0_min, x0_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         x1_min, x1_max = X[:, 1].min() - 1, X[:, 1].max() + 1

         xx0, xx1 = np.meshgrid(np.arange(x0_min, x0_max, 0.02),
                                np.arange(x1_min, x1_max, 0.02))

         z = clf.predict(np.c_[xx0.ravel(), xx1.ravel()])
         z = z.reshape(xx0.shape)
         plt.contourf(xx0, xx1, z, cmap = plt.cm.Paired, alpha = 0.3)

         plt.xlabel('sepal length (cm)')
         plt.ylabel('sepal width (cm)')
         plt.legend()
         plt.savefig('SVM_OUTPUT3.pdf', bbox_inches='tight')
```



使用测试数据进行测试

```
In [19]: clf.score(XTEST, YTEST)
```

```
Out[19]: 1.0
```

使用数据集训练“RBF 核 SVM”

```
In [20]: clf = SVC(kernel = 'rbf')
         clf.fit(XTRAIN, YTRAIN)
```

```
Out[20]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

使用测试数据进行测试

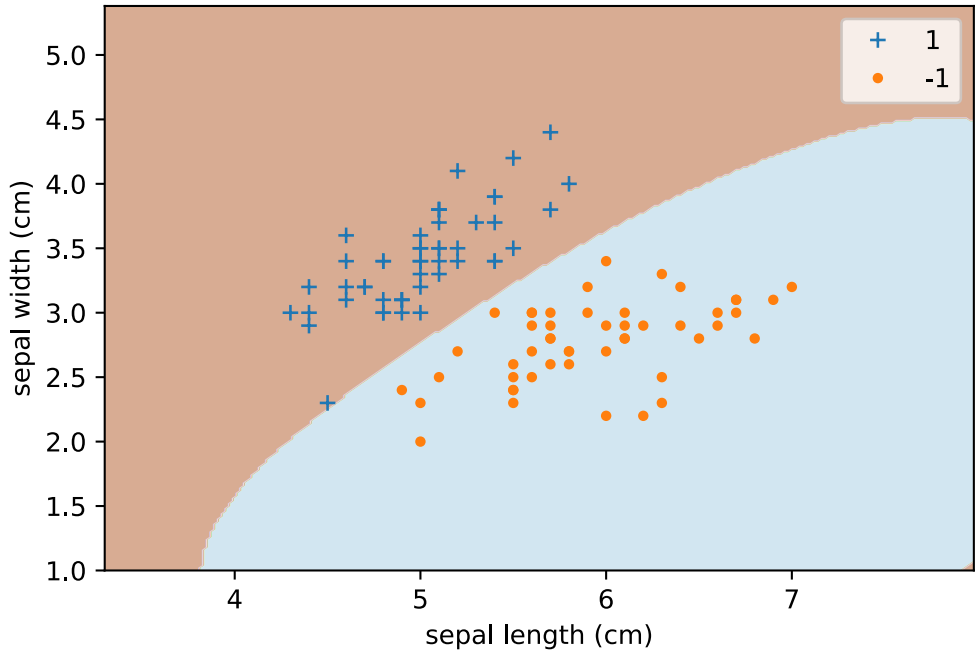
```
In [21]: clf.score(XTEST, YTEST)
```

```
Out[21]: 1.0
```

绘制 SVM 分类结果

```
In [22]: %matplotlib inline
         %config InlineBackend.figure_format = 'svg'
```

```
plt.plot(data[:50, 0], data[:50, 1], '+', label='1')
plt.plot(data[50:100, 0], data[50:100, 1], '.', label='-1')

# 生成绘图用的网格
x0_min, x0_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x1_min, x1_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx0, xx1 = np.meshgrid(np.arange(x0_min, x0_max, 0.02),
                       np.arange(x1_min, x1_max, 0.02))

z = clf.predict(np.c_[xx0.ravel(), xx1.ravel()])
z = z.reshape(xx0.shape)
plt.contourf(xx0, xx1, z, cmap = plt.cm.Paired, alpha = 0.5)

plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.legend()
plt.savefig('SVM_OUTPUT4.pdf', bbox_inches='tight')
```



6 参考文献

1. Platt, John. Fast Training of Support Vector Machines using Sequential Minimal Optimization, in Advances in Kernel Methods -Support Vector Learning, B. Scholkopf, C. Burges, A. Smola, eds., MIT Press (1998).
2. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。
3. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。
4. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
5. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
6. https://github.com/wzyonggege/statistical-learning-method.
7. https://blog.csdn.net/wolfcsharp/article/details/90181429.
8. https://www.zybuluo.com/Duanxx/note/433281.
9. http://blog.csdn.net/xianlingmao/article/details/7919597.
10. https://my.oschina.net/dfsj66011/blog/517766.
11. http://blog.csdn.net/wds2006sdo/article/details/53156589.