第 9 章 支持向量机

目录

第 9 章 支持向量机 1
9.1 SVM 思想 1
9.2 SVM 原理 2
9.2.1 超平面的表达 2
9.2.2 函数间隔 2
9.2.3 几何间隔 3
9.2.4 最大间隔分类器 5
9.2.5 函数间隔的性质 6
9.2.6 小结 6
9.3 SVM 示例代码与线性不可分 7
9.3.1 线性 SVM 示例代码 7
9.3.2 从线性不可分谈起 7
9.3.3 将低维特征映射到高维空间 8
9.3.4 SVM 中的核技巧 9
9.3.5 从高维到无穷维 10
9.3.6 常见核函数 10
9.3.7 小结 11
9.4 SVM 中的软间隔 11
9.4.1 软间隔定义 11
9.4.2 最大化软间隔 12
9.4.3 SVM 软间隔示例代码 12
9.4.4 小结 14
9.5 拉格朗日乘数法 14
9.5.1 条件极值 15
9.5.2 求解条件极值 15
9.5.3 小结 16
9.6 对偶性与 KKT 条件 16
9.6.1 广义拉格朗日乘数法 16
9.6.2 原始优化问题 17
9.6.3 对偶优化问题 17

9.6.4 KKT 条件 18
9.6.5 计算示例 19
9.6.6 小结 21
9.7 SVM 优化问题 21
9.7.1 构造硬间隔广义拉格朗日函数 21
9.7.2 硬间隔求解计算示例 24
9.7.3 构造软间隔广义拉格朗日函数 25
9.7.4 软间隔中的支持向量 28
9.7.5 小结 29
9.8 SMO 算法 29
9.8.1 坐标上升算法 29
9.8.2 SMO 算法思想 30
9.8.3 SMO 算法原理 31
9.8.4 偏置b求解 34
9.8.5 SVM 算法求解示例 35
9.8.6 小结 36
9.9 从零实现支持向量机 36
9.9.1 常见核函数实现 36
9.9.2 SMO 求解过程实现 36
9.9.3 SVM 二分类代码实现 39
9.9.4 SVM 多分类代码实现 39
9.9.5 小结 41

第 9 章 支持向量机

在前面几章中，笔者已经陆续介绍了多种分类算法模型，相信各位读者朋友对于机器学习也算是有了一定的了解。在接下来的这一章中，笔者将开始逐步介绍本书中的最后一个分类模型——支持向量机。支持向量机（Support Vector Machine, SVM）可以算得上是机器学习算法中最为经典的模型。之所以称之为经典是因为支持向量机的背后有着完美的数学推导与证明。当然，也正是因为这个原因使得学习 SVM 有着较高的门槛。因此，在接下来的内容中，笔记将会尽可能以最通俗的表达来介绍 SVM 中的相关原理。

9.1 SVM 思想

什么是支持向量机呢？初学者刚接触到这个算法时基本上都会被这个名字所困扰，到底什么叫“向量机”，听起来总觉得怪怪的。因此首先需要明白的是，支持向量机其实和“机”一点关系也没有，算法的关键在于“支持向量”。如图 9-1 所示为 4 种不同模型对同一个数据集分类后的决策边界图。可以看到尽管每个模型都能准确地将数据集分成两类，但是从各自的决策边界到两边样本点的距离来看却有着很大的区别。

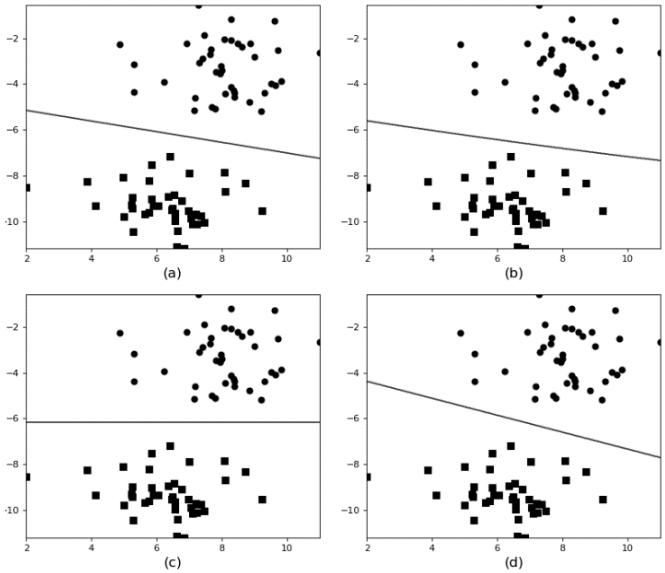

图 9-1 不同模型决策边界

为了能更加清楚的进行观察，下面将 4 个决策边界放到一张图中，如图 9-2 所示。

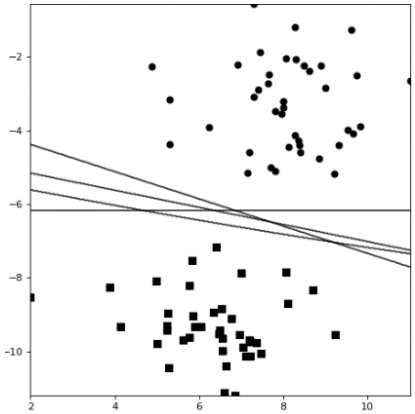

图 9-2 决策边界图

如图 9-2 所示，图中左边从上到下分别为模型(d)(a)(b)(c)在数据集上的决策边界。可以发现模型(c)的泛化能力应该会是最差的，因为从数据的分布位置来看真实的决策面应该是一条左高右低倾斜的直线。其次是模型(b)的泛化能力，因为从图 9-2 可以看出模型(b)的决策面太过于偏向方块形的样本点。因为在评估分类决策面优劣的一个原则就是，当没有明确的先验知识告诉我们决策面应该偏向于哪边时，最好的做法应该是居于中间位置，也就是类似于模型(a)和模型(d)的决策面。那么模型(a)和模型(d)谁又更胜一筹呢？进一步，可以将(a)和(d)这两个模型各自到两侧样本点距离可视化出来，如图 9-3 所示。

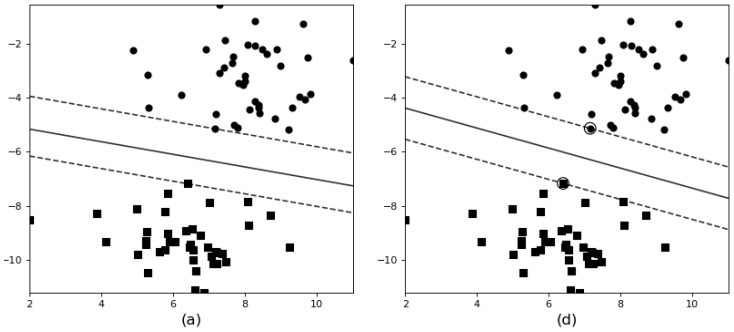

图 9-3 决策边界宽度图

从图 9-3 中一眼便可以看出，模型(d)的决策面要更居于“中间”（事实上就是在中间），而模型(a)的决策面也是略微偏向于方块形的样本点。因此在这 4 个模型中，模型(d)的泛化能力通常情况下都会是最强的。此时有读者可能就会问，假如把模型(a)中的决策面向上平移一点，使得其也居于两条虚线之间，那么此时应该选择谁呢？答案当然依旧是模型(d)，原因在于模型(d)的决策面还满足另外一个条件，到两条虚线的距离最大。换句话说也就是，模型(d)中两条虚线之间的距离要大于模型(a)中两条虚线之间的距离。

说到这里，相信各位读者已经猜到，模型(d)对应的就是支持向量机模型，同时虚线上的两个样本点就被称为支持向量。可以发现，最终对决策面其决定性作用的也只有这两个样本点，说得通俗点就是仅根据这两个点就能训练得到模型(d)。因此，这里可以得出的结论就是，通过支持向量机我们便能够得到一个最优超平面，该超平面满足到左右两侧最近样本点的间隔相同，且离左右最近样本点的间隔最大。不过那又该如何来找到这个超平面呢？

9.2 SVM 原理

9.2.1 超平面的表达

在正式定义距离之前，这里先回顾一下超平面的表达式

$$w^T x + b = 0 \tag{9-1}$$

其中 w 表示权重参数（系数），b 表示截距，x 表示样本点。同时需要说明的是，在 SVM 中，用 $y=+1, y=-1$ 分别来表示正样本和负样本。

从上述表达式可知，当通过某种方法找到参数 w, b 后，也就代表确立了超平面。不过对于 SVM 建模来说应该从哪个地方入手呢？从 SVM 的核心思想最大化间隔（Gap） 入手。

9.2.2 函数间隔

上面说到 SVM 的核心思想就是最大化间隔，既然是最大化间隔那总得有个度量间隔的方法才行。根据中学知识可知，当超平面 $w^T x + b = 0$ 确定后，可以通过 $|w^T x + b|$ 来表示每个样本点到超平面的相对距离，也就是说虽然实际距离不是 $|w^T x + b|$ 这么多，但是它依旧遵循绝对值大的离超平面更远的原则，如图 9-4 所示。

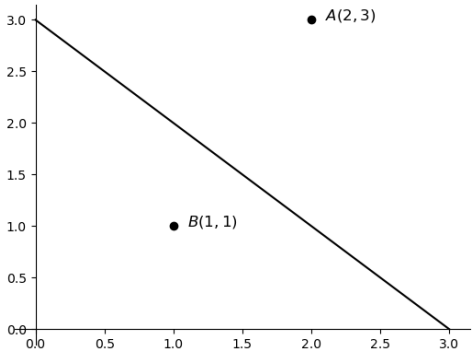

图 9-4 函数间隔图

如图 9-4 所示，其中直线方程为 $x_1 + x_2 - 3 = 0$，且 A, B 分别为正负两个样本点，即 $y^A = +1, y^B = -1$，则此时有点 A 到直线的相对距离为 $|w^T x + b| = |2 + 3 - 3| = 2$，点 B 到直线的相对距离为 $|w^T x + b| = |1 + 1 - 3| = 1$。

同时还可以注意到，只要分类正确，$y^{(i)}(w^T x + b) > 0$ 就成立；或者说如果 $y^{(i)}(w^T x + b) > 0$ 成立，则这就意味着分类正确。并且其值越大说明其分类正确的可信度就越高，而这也是 y 为什么取 ± 1 的原因。所以此时可以将训练集中所有样本点到超平面的函数间隔（Functional Margin）定义为[①]

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) \tag{9-2}$$

且定义训练集中样本点到超平面的函数间隔中的最小值为

$$\hat{\gamma} = \min_{i=1,2,\ldots,m} \hat{\gamma}^{(i)} \tag{9-3}$$

但是此时可以发现，如果在式(9-1)的两边同时乘以 $k(k \neq 0)$，虽然此时超平面并没有发生改变，但是相对距离却变成了之前的 k 倍。所以仅有函数间隔显然不能够唯一确定这一距离，还需要引入另外一种度量方式——几何间隔。

9.2.3 几何间隔

所谓几何间隔（Geometric Margin），就是样本点到直线实实在在的距离。只要直线不发生改变，那么间隔就不会发生任何改变，这样就避免了在函数间隔中所存在的问题。那么应该如何来表示几何间隔呢？如果 9-5 所示，线段 AB 就表示样本点 A 到直线的真实距离。

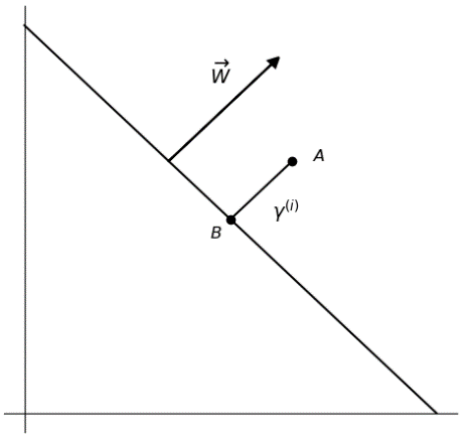

图 9-5 几何间隔图

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

如图 9-5 所示，直线方程为 $w^T x+b=0$，A 为数据集中任意一个点 $x^{(i)}$，$\gamma^{(i)}$ 为 A 到直线的距离，可以看成是向量 $\overrightarrow{BA}$ 的模；W 为垂直于 $w^T x+b=0$ 的法向量。此时便可以得到点 B 的坐标为

$$x^{(i)}-\gamma^{(i)}\cdot\frac{W}{\|W\|} \tag{9-4}$$

又因为 B 点在直线上，所以满足

$$w^T(x^{(i)}-\gamma^{(i)}\cdot\frac{W}{\|W\|})+b=0 \tag{9-5}$$

因此可以通过化简等式(9-5)来得到几何距离的计算公式。不过此时的问题在于 W 该怎么得到？

现在假设有一直线 $w^T x+b=0, w=(w_1,w_2)^T$，即 $w_1x_1+w_2x_2+b=0$，那么该直线的斜率便为 $k_1=-w_1/w_2$。又因为 W 垂直于该直线，所以 W 的斜率为 $k_2=w_2/w_1$。因此 W 的一个方向向量为 $(1,k_2)$。进一步再同时乘以 w_1 即可得到 $W=(w_1,w_2)=w$，即 W 其实就是 w。也就是说，如果直线 $w^T x+b=0$，那么 w 就是该直线的其中一条法向量。

所以根据式(9-5)有

$$w^T(x^{(i)}-\gamma^{(i)}\cdot\frac{w}{\|w\|})+b=0 \tag{9-6}$$

因此几何距离计算公式为

$$\gamma^{(i)}=\frac{w^T x^{(i)}+b}{\|w\|}=(\frac{w}{\|w\|})^T x^{(i)}+\frac{b}{\|w\|} \tag{9-7}$$

当然，这只是当样本 A 为正样本，即 $y_A=+1$ 的情况，更一般地几何距离的计算公式为

$$\gamma^{(i)}=y^{(i)}((\frac{w}{\|w\|})^T x^{(i)}+\frac{b}{\|w\|}) \tag{9-8}$$

故，根据图 9-4 可知，样本点 A,B 到直线 $x_1+x_2-3=0$ 的距离分别为

$$\begin{aligned}\gamma^A&=+1\cdot\left((\frac{w}{\|w\|})^T x^{(A)}+\frac{b}{\|w\|}\right)=(\frac{(1,1)}{\sqrt{1+1}})^T(2,3)+\frac{-3}{\sqrt{1+1}}=\sqrt{2}\\ \gamma^B&=-1\cdot\left((\frac{w}{\|w\|})^T x^{(A)}+\frac{b}{\|w\|}\right)=-(\frac{(1,1)}{\sqrt{1+1}})^T(1,1)+\frac{3}{\sqrt{1+1}}=\frac{1}{\sqrt{2}}\end{aligned} \tag{9-9}$$

此时可以发现，同函数间隔类似只要在分类正确的情况下几何间隔也都满足条件 $y^{(i)}\cdot\gamma^{(i)}>0$。进一步，定义训练集中样本点到超平面的几何间隔中最小值为

$$\gamma=\min_{i=1,2,\ldots,m}\gamma^{(i)} \tag{9-10}$$

同时，函数间隔与几何间隔存在以下关系

$$\gamma=\frac{\hat{\gamma}}{\|w\|} \tag{9-11}$$

可以发现，几何间隔其实就是在函数间隔的基础上施加了一个约束限制。此时我们已经有了对于间隔度量的方式，所以下一步自然就是最大化这个间隔来求得分类超平面。

9.2.4 最大间隔分类器

什么是最大间隔分类器（Maximum Margin Classifiers） 呢？上面说到，有了间隔的度量方式后，接着就是最大化这一间隔，然后求得超平面 $w^T x+b=0$ 。最后通过函数 $g(w^T x+b)$ 将所有样本点输出只含 $\{-1,+1\}$ 的值，以此来完成对数据集样本的分类任务。由于 $g(w^T x+b)$ 就是一个分类器，又因为它是通过最大化几何间隔得来的，故将其称之为最大间隔分类器。

因为在式(9-10)中已经得到了几何间隔的表达式，所以再对其最大化即可

$$\begin{aligned}&\max_{w,b}\ \gamma\\&s.t.\ \ y^{(i)}((\frac{w}{\|w\|})^T x^{(i)}+\frac{b}{\|w\|})\geq\gamma, i=1,2,...m\end{aligned}\tag{9-12}$$

其中 $s.t.$ 表示服从于约束条件。同时，式(9-12)的含义就是找到参数 w,b ，使得满足以下条件：

① γ 尽可能大，因为目的就是最大化 γ；

② 同时要使得样本中所有的几何距离都大于 γ ，因为由式(9-10)可知 γ 是所有间隔中的最小值；

所以，进一步由式(9-11)中函数间隔与几何间隔的关系，可以将式(9-12)中的优化问题转化为

$$\begin{aligned}&\max_{w,b}\ \frac{\hat{\gamma}}{\|w\|}\\&s.t.\ \ y^{(i)}(w^T x^{(i)}+b)\geq\hat{\gamma}, i=1,2,...m\end{aligned}\tag{9-13}$$

此时可以发现，约束条件由几何间隔变成了函数间隔，准确说应该既是函数间隔同样也是几何间隔。因此，既然可以看作函数间隔，那么令 $\hat{\gamma}=1$ 自然也不会影响最终的优化结果。

所以，式(9-13)中的优化问题便可以再次转化为如下形式

$$\begin{aligned}&\max_{w,b}\ \frac{1}{\|w\|}\\&s.t.\ \ y^{(i)}(w^T x^{(i)}+b)\geq 1, i=1,2,...m\end{aligned}\tag{9-14}$$

但是对于(9-14)这样一个优化问题还是无法直接解决。不过，对于 $f(x)>0$ 来说 $\max 1/\ f(x)$ 就等价于 $\min\ f(x)$，进一步也就等价于 $\min(f(x))^2$，这三者求解出的 x 都相同。所以进一步可以将式(9-14)化简为

$$\begin{aligned}&\min_{w,b}\ \frac{1}{2}\|w\|^2\\&s.t.\ \ y^{(i)}(w^T x^{(i)}+b)\geq 1, i=1,2,...m\end{aligned}\tag{9-15}$$

之所以要进行这样的处理，是因为这样可以将其转换为一个典型的凸优化问题，并用现有的方法进行求解；而在前面乘以 $1/2$ 是为了后面求导时方便，同时这也不会影响优化结果。到这一步，我们就算是搞清楚了 SVM 的基本思想，以及它需要求解的优化问题。

9.2.5 函数间隔的性质

在 9.2.1 节优化问题的化简过程中笔者直接将函数间隔设置为了 1，不过相信对于不少读者来说在这一点上仍旧比较疑惑。当然，这也是一个在学习 SVM 中最典型的问题，因此接下来就这点进行一个简要的说明。

假设现在有如下函数间隔

$$\hat{\gamma} = y^{(i)}(w^T x^{(i)} + b) \tag{9-16}$$

那么对等式(9-16)两边同时除以 $\hat{\gamma}$ 便有

$$y^{(i)}((\frac{w}{\hat{\gamma}})^T x^{(i)} + \frac{b}{\hat{\gamma}}) = 1 \tag{9-17}$$

此时令 $W = \frac{w}{\hat{\gamma}}, B = \frac{b}{\hat{\gamma}}$，便可以将(9-17)转化为

$$y^{(i)}(W^T x^{(i)} + B) = 1 \tag{9-18}$$

接着再把式(9-18)中的 W, B 换成 w, b 即可得到

$$y^{(i)}(w^T x^{(i)} + b) = 1 \tag{9-19}$$

不过需要明白的是，式(9-16)和式(9-19)中的 w, b 并不是同一个。

例如现有如下平面方程

$$2x_1 + 4x_2 - 8 = 0 \tag{9-20}$$

某正样本 $y^{(k)} = +1$ 的函数间隔为 $\hat{\gamma}^{(k)} = 2$，所以有

$$+1(2x_1^{(k)} + 4x_2^{(k)} - 8) = 2 \tag{9-21}$$

进一步在等式(9-21)两边同时除以 2 有

$$x_1^{(k)} + 2x_2^{(k)} - 4 = 1 \tag{9-22}$$

虽然此时的 w, b 同时都缩小了两倍，函数间隔变成了 1，但是 $2x_1 + 4x_2 - 8 = 0$ 与 $x_1 + 2x_2 - 4 = 0$ 所表示的依旧是同一个平面。所以此时可知，$w^T x^{(i)} + b = 0$ 与 $W^T x^{(i)} + B = 0$ 代表的是同一个平面，故可以直接由式(9-16)得到式(9-19)，也就是说同一个平面与用什么字母表示无关。因此可以将函数间隔直接设为 1（实质是同时除以了函数间隔）。

9.2.6 小结

在本节中，笔者首先通过一个引例介绍了支持向量机的核心思想；接着介绍了支持向量机中衡量间隔的两种度量方式，即函数间隔和几何间隔；然后介绍了如何通过结合函数间隔与几何间隔来建模支持向量机的优化问题；最后还介绍了 SVM 中的一个经典问题，函数间隔为什么可以设为 1。

9.3 SVM 示例代码与线性不可分

在前面两节内容中，笔者介绍了支持向量机的基本思想以及对应的数学原理。不过说一千道一万，还是不如自己亲手来做一做。在接下来的内容中，笔者将首先介绍如何通过 sklearn 来搭建相应的 SVM 分类模型，然后将接着介绍如何处理 SVM 中的线性不可分问题。

9.3.1 线性 SVM 示例代码

在 sklearn 中可以通过 from sklearn.svm import SVC 这句代码就能够导入 SVM 分类模型了。有读者可能会觉得奇怪，为什么导入的是一个叫 SVC 的东西？这是因为其实 SVM 不仅可以用来分类，它同样也能用于回归问题，因此 SVC 其实就是支持向量分类的意思。

点击进入 SVC 定义的地方便可以发现里面有很多超参数可以进行设置

```
def __init__(self,
    C=1.0, kernel='rbf',
    degree=3, gamma='scale'
    coef0=0.0,decision_function_shape='ovr'):
```

在上述代码中只列举了 SVM 中常见的一些超参数。不过这里暂时只对 kernel 这个参数进行介绍，其它的参数等介绍完相关原理后再进行解释。根据前面两节内容可知 SVM 是一个线性分类器，因此这里只需要将参数 kernel 设置为 kernel='linear'便能达到这一目的。

在完成 SVC 的导入工作后，根据如下代码便可以使用线性 SVM 进行分类建模，完整实例代码参见 Book/Chapter09/01_linear_svm.py 文件。

```
def train(x_train, x_test, y_train, y_test):
    model = SVC(kernel='linear')
    model.fit(x_train, y_train)
    y_pre = model.predict(x_test)
    print(f"准确率为: {model.score(x_test, y_test)}")
#准确率为: 0.975925925925926
```

上述代码就是通过 sklearn 实现线性 SVM 的全部代码。可以看出，在 sklearn 中使用一个模型的步骤依旧是笔者在 5.3.1 节中总结的 3 步走：建模、训练和预测。同时，由于这里的超参数 kernel 暂时只有一个取值，因此也不需要进行模型选择。从最后在测试集上的结果来看，线性 SVM 分类器的表现在准确率上有着不错的结果。

9.3.2 从线性不可分谈起

根据 9.2 节内容中 SVM 的思想来看，到目前为止谈到的情况都是线性可分的，也就是说总能找到一个超平面将数据集分开。可事实上却是，在大多数场景中各个类别之间都是线性不可分的，即类似如图 9-6 所示的情况。

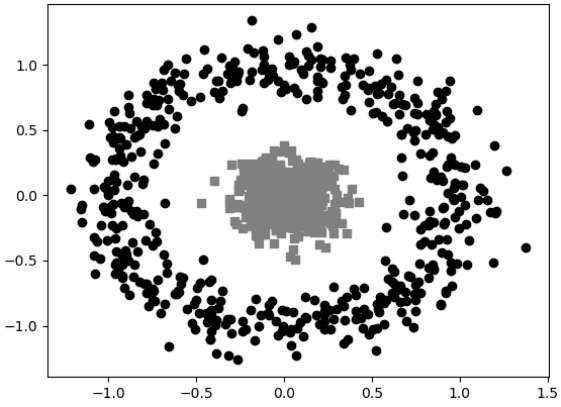

图 9-6 SVM 线性不可分图

对于图 9-6 中这种情况应该怎么才能将其分开呢？在 4.2.4 节中笔者介绍过，这类问题可以使用特征映射的方法将原来的输入特征映射到更高维度的空间，然后再找寻一个超平面将数据集中不同类别的样本进行分类，如图 9-7 所示。

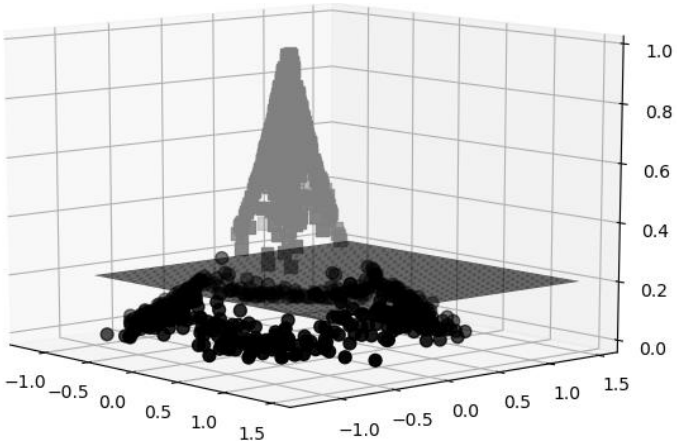

图 9-7 SVM 特征映射图

如图 9-7 所示，现在我们已经用一个超平面完美的将不同类别的样本进行了分开。不过此时有读者可能会疑惑到这还是刚刚的数据集么？之前明明在二维平面，现在却跑到三维空间。虽然数据集确确实实已经不是同一个数据集了，但是每个数据样本所对应的类别却依旧和原来的一样，只不过现在给它穿上了一件“马甲”。也就是说，假如 $x^{(i)}$ 是正样本，那么它穿上马甲变成 $\hat{x}^{(i)}$ 后仍然属于正样本，只要能把 $\hat{x}^{(i)}$ 进行正确分类，那么自然也就能够对 $x^{(i)}$ 进行分类。

在介绍完特征映射的基本思想后，下面笔者就来介绍如何在 SVM 中将低维特征映射到高维空间中。

9.3.3 将低维特征映射到高维空间

所谓将低维特征映射到高维空间指的是用一定的映射关系，将原始特征映射到更高维度的空间。比如通过一个函数 $\phi(x)$ 将一维特征 x 映射到三维特征 x, x^2, x^3。在这里，笔者先直接给出 SVM 中权重 w 的计算解析式，其具体由来可参见 9.7.1 节内容。

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{9-23}$$

根据式(9-23)可知，假如此时已求得 α_i 和 b，那么对一个新输入的样本点其预测结果为

$$\begin{aligned} w^T x + b &= \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} x + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \end{aligned} \tag{9-24}$$

其中 $x^{(i)}$ 表示训练集中的样本点（其实就是支持向量），x 为新输入的样本点；$\langle a, b \rangle$ 表示 a, b 之间的内积（Inner Products）。当且仅当式(9-24)大于 0 时，新输入样本 x 的类别为 $y = 1$。

按照上面提到的通过函数 $\phi(x)$ 将低维映射到高维的思想，只需要在预测时将之前的 x，全部替换成 $\phi(x)$，则此时有

$$\begin{aligned} y &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle \phi(x^{(i)}), \phi(x) \rangle + b \end{aligned} \tag{9-25}$$

虽然这样一来算是一定程度上解决了 SVM 中线性不可分的难题，但是又出现了一个新的问题——“维度爆炸”。

假设现有数据集 X，其样本点 $x^{(i)}$ 有 3 个维度，分别为 $x_1^{(i)}, x_2^{(i)}, x_3^{(i)}$（下面简写为 x_1, x_2, x_3）。现通过函数 $\phi(x)$ 将其映射到某个 9 维空间中，且假设映射后的 9 个维度分别为 $x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3$。如果此时要对新样本 z 进行预测，则首先需要对 $\langle \phi(x), \phi(z) \rangle$ 进行计算

$$\begin{aligned} \phi(x) &= [x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3]^T \\ \phi(z) &= [z_1z_1, z_1z_2, z_1z_3, z_2z_1, z_2z_2, z_2z_3, z_3z_1, z_3z_2, z_3z_3]^T \\ \langle \phi(x), \phi(z) \rangle &= [x_1x_1z_1z_1 + x_1x_2z_1z_2 + \cdots + x_3x_3z_3z_3] \end{aligned} \tag{9-26}$$

此时各位读者应该会发现这个过程的计算量太大了，整体复杂度为 $o(n^2)$（分别为 $o(n^2), o(n^2), o(n)$），其中 n 为特征的维数。因此，若是在高维数据中进行更为复杂的映射，那么整个过程的时间复杂度将不可想象，而这就是“维度爆炸”。但是此时我们仔细想一想，“映射”和“预测”之间到底是什么关系？“映射”是作为一种思想将低维映射到高维，从而解决线性不可分到可分的问题；而“预测”时所计算的则是 $\langle \phi(x), \phi(z) \rangle$，但说穿了它就是一个值。不管最后采用的是什么样的映射规则，在预测时都只需要计算这么一个值。因此，假如能通过某种“黑箱”直接计算出这一个值岂不最好？那有没有这样的黑箱呢？当然有，这一“黑箱”操作就称为核函技巧（Kernel Trick）。

9.3.4 SVM 中的核技巧

设 $\mathcal{X}$ 是输入空间（欧式空间 R^n 的子集或离散集合），又设 $\mathcal{H}$ 为特征空间（希尔伯特空间），如果存在一个从 $\mathcal{X}$ 到 $\mathcal{H}$ 的映射 $\phi(x): \mathcal{X} \to \mathcal{H}$ 使得对所有 $x, z \in \mathcal{X}$，函数 $K(x, z)$ 满足条件 $K(x, z) = \phi(x) \cdot \phi(z)$，则称 $K(x, z)$ 为核函数，$\phi(x)$ 称为映射函数[①]。

说得简单点就是，存在某个映射能够找到一个与之对应的核函数 $K(x, z)$ 用来代替计算 $\langle \phi(x), \phi(z) \rangle$，从而避免了上面出现“维度爆炸”的问题。因此，核函数可以看作是实现“黑箱”操作（核技巧）的工具。

现假设式(9-26)中的两个样本点分别为 $x = (1, 2, 3), z = (2, 3, 4)$，则此时有

$$\begin{aligned} \phi(x) &= (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)^T \\ &= (1\times1, 1\times2, 1\times3, 2\times1, 2\times2, 2\times3, 3\times1, 3\times2, 3\times3)^T \\ \phi(z) &= (z_1z_1, z_1z_2, z_1z_3, z_2z_1, z_2z_2, z_2z_3, z_3z_1, z_3z_2, z_3z_3)^T \\ &= (2\times2, 2\times3, 2\times4, 3\times2, 3\times3, 3\times4, 4\times2, 4\times3, 4\times4)^T \\ \langle \phi(x), \phi(z) \rangle &= (x_1x_1z_1z_1 + x_1x_2z_1z_2 + \cdots + x_3x_3z_3z_3)^T \\ &= 4 + 12 + 24 + 12 + 36 + 72 + 24 + 72 + 144 = 400 \end{aligned} \tag{9-27}$$

同时，还可以通过另外一种方式来计算得到这个结果

[①] 李航，统计机器学习，清华大学出版社

$$K(x,z)=(x^Tz)^2=(2+6+12)^2=400 \tag{9-28}$$

此时可以发现，虽然式(9-27)与式(9-28)计算得到的结果相同，但是两者在计算过程上却是大相径庭。前者需要$O(n^2)$的时间复杂度，但后者只需要$O(n)$的时间复杂度。接着可能有读者会疑惑，笔者是怎么知道$(x^Tz)^2$等于$\phi(x)\cdot\phi(z)$的？那下面就是推导

$$\begin{aligned}(x^Tz)^2&=\left(\sum_{i=1}^{n}x_iz_i\right)\left(\sum_{j=1}^{n}x_jz_j\right)=\sum_{i=1}^{n}\sum_{j=1}^{n}x_ix_jz_iz_j\\&=x_1x_1z_1z_1+x_1x_2z_1z_2+x_1x_3z_1z_3+\cdots+x_nx_nz_nz_n\\&=\phi(x)\cdot\phi(z)\end{aligned} \tag{9-29}$$

其实也就是说，笔者先进行了推导知道$(x^Tz)^2$等于$\phi(x)\cdot\phi(z)$，然后在举例过程中才列出了$\phi(x)$这个映射规则。但是话又说回来，我们关心映射规则干什么？我们需要的是映射规则吗？我们需要的不就是这个内积吗？假如笔者换成$K(x,z)=(x^Tz)^5$，那么也只需要计算$(2+6+12)^5$的值即可，而根本不用关系原始特征被映射到了一个什么样的高维空间，并且从一定程度上来说映射到的空间越高越有利于找到分类决策面。所以，我们需要担心的应该是核$K(x,z)$背后所表示的空间是否存在，即核函数的有效性。

9.3.5 从高维到无穷维

上面笔者介绍到，从一定程度上来说映射到越高维度的空间就越有利于找到分类决策面。因此，一种自然而然的想法就是如果能将原始特征映射到n维空间岂不是更好？说得倒是没错，但这该怎么实现呢？是令$K(x,z)=(x^Tz)^n$吗？要实现从低维到无穷维的映射的方法之一就是借助高斯核函数（Gaussian Kernel）或者称为径向基函数[①]（Radial Basis Function, RBF）。

$$\begin{aligned}K(x,z)&=\exp\left(\frac{-\|x-z\|^2}{2\sigma^2}\right)\\&=\exp\left(\frac{-\|x\|^2}{2\sigma^2}\right)\exp\left(\frac{-\|z\|^2}{2\sigma^2}\right)\exp\left(\frac{\langle x,z\rangle}{\sigma^2}\right)\end{aligned} \tag{9-30}$$

不过为什么借助式(9-30)就能实现到无穷维的映射呢？回忆一下泰勒展开就会得出，式(9-30)中第 2 行第 3 项的泰勒展式为

$$\exp\left(\frac{\langle x,z\rangle}{\sigma^2}\right)=1+\frac{\langle x,z\rangle}{\sigma^2}+\frac{\langle x,z\rangle^2}{2\sigma^4}+\frac{\langle x,z\rangle^3}{3!\sigma^6}+\cdots=\sum_{i=0}^{n}\frac{\langle x,z\rangle^n}{n!(\sigma^2)^n} \tag{9-31}$$

也就是说，由于泰勒展开的存在，RBF 自然也就隐含的实现了从低维到无穷维的映射。

9.3.6 常见核函数

在实际解决问题的时候，甚至都不用关心核函数它到底是如何映射的，只需要正确选用核函数，实现分类的目的即可。下面是一些常见的核函数，其中使用最为广泛的就算高斯核函数[②]。

1) **线性核**（Linear Kernel）

$$K(x,z)=\langle x,z\rangle \tag{9-32}$$

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.
[②] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

2) 多项式核（Polynomial Kernel）

$$K(x,z)=(\gamma\langle x,z\rangle+r)^d \tag{9-33}$$

其中 $\gamma>0$ 为核函数系数，r 为常数。

3) 高斯核（Gaussian Kernel）

$$K(x,z)=\exp\left(-\gamma\|x-z\|^2\right) \tag{9-34}$$

其中 $\gamma>0$ 为核函数系数。这里需注意的是，式(9-34)与(9-30)虽然在形式上有所差异，但是本质上都是一样。为了便于后续介绍 sklearn 中的相关参数，所以这里笔者采用了式(9-34)中的形式。

4) Sigmoid 核

$$K(x,z)=tanh(\gamma\langle x,z\rangle+r) \tag{9-35}$$

其中 $\gamma>0$ 为核函数系数，$thanh$ 为双曲正切函数。

通过 9.3.3 节内容的讨论可知，我们总是希望能够找到一个使得样本点线性可分的特征映射空间，因此对于核函数的选择就显得至关重要。同时需要注意的是，由于核函数也仅仅只是隐式的定义了这个特征空间，所以核函数的选择也成为了支持向量机最大的变数。最后，对于 sklearn 中核函数的使用只需要在定义模型时通过参数 kernel 进行指定即可。

9.3.7 小结

在本节中，笔者首先介绍了如何在 sklearn 中搭建一个 SVM 分类模型；接着进一步解释了 SVM 中的线性不可分问题；然后详细介绍了如何通过以特征映射的方式来解决线性不可分的问题；最后也进一步的说明了为什么使用核函数能够将低维特征映射到无穷维的原理。

9.4 SVM 中的软间隔

在前面 3 节内容中，笔者分别介绍了什么是支持向量机以及如何通过 sklearn 来完成整个 SVM 的建模过程；然后还介绍了什么是线性不可分与核函数。在接下来的这节内容中，笔者将继续介绍 SVM 中的软间隔与 sklearn 相关 SVM 模型的实现。

9.4.1 软间隔定义

在 9.2 节和 9.3 节中，笔者分别介绍了以下两种情况的分类任务：①原始样本接线性可分；②原始样本线性不可分，但通过 $\phi(x)$ 映射到高维空间之后“线性可分”。为什么后面这个“线性可分”要加上引号呢？这是因为在上一节中其实有一件事没有和各位读者交代，即虽然通过将原始样本映射到高维空间的方法能够很大程度上使得原先线性不可分的样本点线性可分，但是这也并不能够完全保证每个样本点都是线性可分[①]。或者是保守点说，即使完全线性可分了，但也极大可能会出现过拟合的现象。这可能是因为超平面对于异常点过于敏感，或者数据本身的属性所造成，如图 9-8 所示。

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

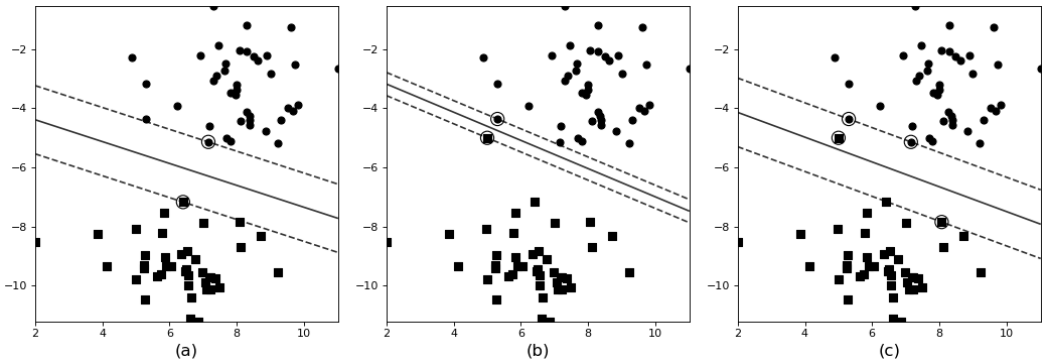

图 9-8 SVM 软间隔与硬间隔图

在图 9-8 中 ，实线为相应的决策面，黑色方块和黑色圆点分别为两个类别的样本。在图(a)中，通过 SVM 建模得到的决策面已经完美的将两种类别的样本点进行了区分。但是，如果此时训练样本中加入一个异常点，并且继续用 SVM 建模求解，那么将会得到图(b)中所示的分类决策面。可以发现，虽然此时决策面也成功的区分开了每一个样本点，但是相较于(a)中的决策面却发生了剧烈的摆动，决策面到支持向量的距离也变得十分狭窄。

在 SVM 中，将图(a)和(b)中决策面到支持向量的间隔称之为硬间隔（Hard Margin），即不允许任何样本出现错分的情况，即使可能导致过拟合。当然，理想情况下期望的应该是图(c)中的这种情况，容许少量样本被错分从而得到一个次优解，而这个容忍的程度则通过目标函数来调节。或者再极端一点就是根本找不到一个超平面能够将样本无误的分开，必须得错分一些样本点。此时图(c)中决策面到支持向量的间隔就被称之为软间隔（Soft Margin）。

9.4.2 最大化软间隔

从上面的介绍可知，如数据集中出现了异常点那么必将导致该异常点的函数间隔小于 1。所以，可以为每一个样本引入一个松弛变量（$\xi_i \geq 0$）来使得函数间隔加上松弛变量大于等于 1。

$$y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \tag{9-36}$$

那么此时的目标函数可以重新改写为如下形式

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{m} \xi_i \\ &s.t.\ \ y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ \ \xi_i \geq 0, i = 1, 2, ...m \end{aligned} \tag{9-37}$$

其中$C > 0$称为惩罚系数，C越大时对误分类样本的惩罚就越大，其作用等同于正则化中的参数λ 。可以发现，只要错分一个样本点，目标函数都将付出$C\xi_i$的代价，并且为了使得目标函数尽可能小，那么就需要整个惩罚项相对要小。因此，如果使用较大的惩罚系数，那么将会得到较窄的分类间隔，即惩罚力度大允许错分的样本数就会减少；如果使用较小的惩罚系数，则会得到相应较宽的分类间隔，即惩罚力度小允许多的错分样本。

9.4.3 SVM 软间隔示例代码

在 9.3.1 节内容中，笔者大致列出了 SVM 分类器中常见的几个重要参数，如下所示：

```
def init(self,
    C=1.0,
    kernel='rbf',
    degree=3,
```

```
    gamma='scale',
    coef0=0.0):
```

在上述代码中，其中C就表示式(9-37)中的惩罚系数，它的作用是用来控制容忍决策面错分样本的程度，越大则模型越偏向于过拟合。如图 9-9 所示为C在不同取值下决策面（分类间隔较大时$C=1$，分类间隔较小时$C=1000$）。参数 kernel 表示选择哪种核函数，当 kernel='poly'时可以用参数 degree 来选择多项式的次数。但是通常情况下都会选择效果更好的高斯核（kernel='rbf'）来作为核函数，因此该参数用得比较少。参数 gamma 为核函数系数，使用默认值即可；coef0 为多项式核和 sigmoid 核中的常数r。

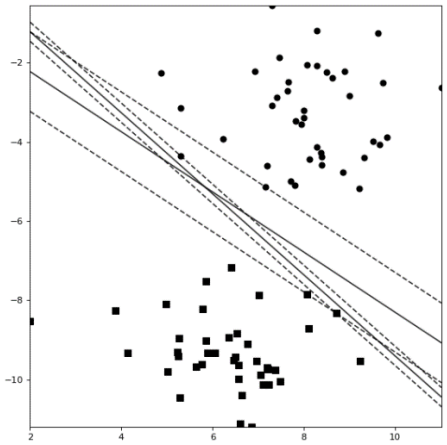

图 9-9 SVM 不同惩罚系数下的决策平面

下面笔者将采用网格搜索来选择一个最佳的 SVM 分类器对数据集 iris 进行分类。从上面对 sklearn 中 SVM 的 API 的介绍可知，SVC 中需要用到的超参数有 5 个，其取值这里分别设为：'C': np.arange(1, 10, 5)， 'kernel': ['rbf', 'linear', 'poly']， 'degree': np.arange(1, 10, 2)，'gamma':['scale', 'auto'], 'coef0': np.arange(-10, 10, 5)。由此便有$2\times3\times5\times2\times4=240$个备选模型。同时，这里以 3 折交叉验证进行训练，则一共需要拟合 720 次模型。完整示例代码参见 Book/Chapter09/02_soft_margin_svm.py 文件。

1) 模型选择

首先，需要根据列举出的超参数在数据集上根据交叉验证搜索得到最优超参数组合，代码如下：

```
def model_selection(x_train, y_train):
    model = SVC()
    paras = {'C': np.arange(1, 10, 5),
           'kernel': ['rbf', 'linear', 'poly'],
           'degree': np.arange(1, 10, 2),
           'gamma': ['scale', 'auto'],
           'coef0': np.arange(-10, 10, 5)}
    gs = GridSearchCV(model, paras, cv=3, verbose=2, n_jobs=3)
    gs.fit(x_train, y_train)
    print('best score:', gs.best_score_,'best
                    parameters:', gs.best_params_)
```

在完成超参数搜索后，便能够得到一组最优的参数组合，如下所示：

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
[CV]  END  ..C=1,  coef0=-10,degree=1,gamma=scale,kernel=rbf;total  time=
0.0s
[CV]  END  ..C=1,  coef0=-10,degree=1,gamma=scale,kernel=rbf;total  time=
0.0s
…
```

```
[CV] END ...C=6, coef0=5, degree=9,gamma=auto,kernel=rbf; total time= 0.5s
best score: 0.986
best parameters: {'C': 6, 'coef0': -10, 'degree': 1, 'gamma': 'scale',
'kernel': 'rbf'}
```

从上面的结果可以看出，当惩罚系数 C=6，以及选取高斯核函数时对应的模型效果最好，准确率为 0.986。并且由于最后选取的是高斯核，所以此时 coef0 和 degree 这两个参数无效。

2) 训练与预测

在通过网格搜索找到对应的最优模型后，可以再次以完整的训练集对该模型进行训练，代码如下：

```
def train(x_train, x_test, y_train, y_test):
    model = SVC(C=6, kernel='rbf',gamma='scale')
    model.fit(x_train, y_train)
    score = model.score(x_test, y_test)
    y_pred = model.predict(x_test)
    print("测试集上的准确率: ", score)# 0.985
```

可以看出，此时模型在测试集上的准确率为 0.985 左右。当然，如果有需要的话还可以将训练好模型进行保存以便于后期复用，具体方法将在第 7 章中进行介绍。

9.4.4 小结

在本节中，笔者首先介绍了什么是软间隔及其原理；然后以 iris 分类数据集为例，再次介绍了在 sklearn 中如何用网格搜索来寻找最佳的模型参数。到此，笔者就介绍完了 SVM 算法第一阶段的主要内容，可以发现内容不少并且也有相应的难度。在接下来的几节内容中，笔者将首先和读者朋友们一起回顾一下好久不见的拉格朗日乘数法；然后再介绍求解模型参数需要用到的对偶问题；最后便是 SVM 的整个优化求解过程。

9.5 拉格朗日乘数法

在正式介绍 SVM 算法的求解过程之前，笔者先来带着各位读者回顾一下拉格朗日乘数法，因为这同时也是第 10 章中用来对聚类算法求解的工具。可能对于一部分读者来说，使用拉格朗日乘数法已经是很多年前的事情了，其中的细节也自然是慢慢模糊了起来。但是对于拉格朗日乘数法的作用几乎是大家都不会忘记的，那就是用来求解条件极值。既然大多数读者的记忆都停留在这个地方，那么笔者下面就从条件极值开始来简单的介绍一下拉格朗日乘数法。

这里首先以一个例题来重温条件极值的求解过程。求解目标函数 $z = xy$ 在约束条件下 $x + y = 1$的条件极值。

首先作拉格朗日函数

$$F(x, y, \lambda) = xy + \lambda(x + y - 1) \tag{9-38}$$

由式(9-38)可得函数 F 的驻点为

$$\begin{aligned} F_x &= y + \lambda = 0 \\ F_y &= x + \lambda = 0 \\ F_\lambda &= x + y - 1 = 0 \end{aligned} \tag{9-39}$$

求解方程组(9-39)便可求得 x, y, λ 分别为

$$x = \frac{1}{2}; y = \frac{1}{2}; \lambda = -\frac{1}{2} \tag{9-40}$$

由此便可以知道，目标函数 $z=xy$ 在约束条件下 $x+y=1$ 的条件极值为 $z=0.5\times0.5=0.25$。不过为什么可以通过这样的方法来求得条件极值呢？

9.5.1 条件极值

在数学优化问题中，拉格朗日乘数法（Lagrange multipliers ）是一种用于求解等式约束条件下局部最小（最大）值的策略。它的基本思想是通过将含约束条件的优化问题转化为无约束条件下的优化问题，以便于得到各个未知变量的梯度，进而求得极值点[①]。因此，一句话总结就是拉格朗日乘数法是一种用来求解条件极值的工具。那么什么又是条件极值呢？所谓条件极值是指，在一定约束条件下（通常为方程）目标函数的极值就称为条件极值。

如图 9-10 所示，目标函数 $z=f(x,y)$ 在其定义域上的极大值（也是最大值）为 $z=f(x_1,y_1)$，但如果此时对其施加一个约束条件 $\varphi(x,y)=0$，那这就等价的告诉函数 $z=f(x,y)$ 取得极值点同时还要满足约束条件[②]。因此，$z=f(x,y)$ 在约束条件 $\varphi(x,y)=0$ 下的极值点只能在 (x_0,y_0) 处获得（因为此时的 $\varphi(x_0,y_0)=0$，而 $\varphi(x_1,y_1)\neq0$ 即不满足约束条件）。

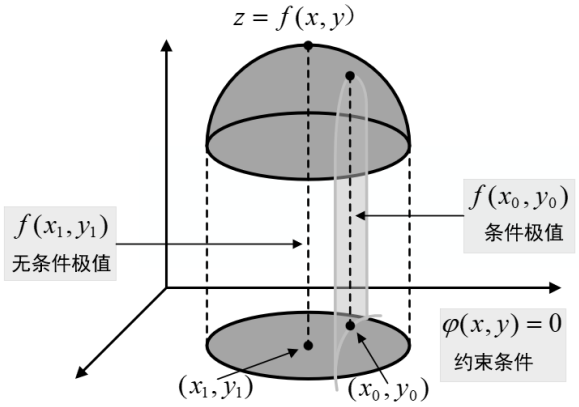

图 9-10 条件极值图

现在，相信各位读者朋友对条件极值已经有了一个直观上的理解，那么接下来要探究的就是怎么才能求得这个极值。

9.5.2 求解条件极值

通常来说，对于包含有等式约束条件目标函数的条件极值可以通过拉格朗日乘数法（Lagrange Multipliers）来进行求解。因此，对于多元函数 $Z=f(x,y,z,...)$ 在多个约束条件 $\varphi(x,y,...)=0,\phi(x,y,...)=0,...$ 下的条件极值，利用拉格朗日乘数法求解的步骤可以总结为[③]：

1) 作拉格朗日函数

$$F(x,y,z,...,\lambda,\mu,...)=f(x,y,z,...)+\lambda\varphi(x,y,...)+\mu\phi(x,y,...)+... \tag{9-41}$$

其中 λ,μ 称为拉格朗日乘子。

2) 求多元函数 $F(x,y,z,...,\lambda,\mu,...)$ 的驻点

解如下方程组求得驻点 $(x_0,y_0,z_0,...,\lambda_0,\mu_0,...)$。

① https://en.wikipedia.org/wiki/Lagrange_multiplier
② 徐小湛,高等数学学习手册,科学出版社
③ 徐小湛,高等数学学习手册,科学出版社

$$\begin{cases} F_x = 0 \\ F_y = 0 \\ \cdots \\ F_\lambda = 0 \\ \cdots \end{cases} \tag{9-42}$$

那么此时 $f(x_0, y_0, z_0, ...,)$ 便是可能的条件极值。

9.5.3 小结

在本节中，笔者首先介绍通过一个引例介绍了如何通过拉格朗日乘数法来求解条件极值；然后总结了如何用拉格朗日乘数法来求解多元函数的条件极值。对于拉格朗日乘数法，我们在后续介绍聚类算法的求解过程中同样也会用到，因此有必要知道其具体求解步骤。

9.6 对偶性与 KKT 条件

在上一节内容中，笔者介绍了什么是拉格朗日乘数法以及它的作用。同时笔者还特意说到，拉格朗日乘数法只能用来求解等式约束条件下的极值。但是当约束条件为不等式的时候又该如何进行求解呢？

9.6.1 广义拉格朗日乘数法

由拉格朗日乘数法可知，对于如下等式条件的约束问题

$$\begin{aligned} &\min_w \ f(w) \\ &s.t. \ \ h_i(w) = 0, i = 1, \cdots, l. \end{aligned} \tag{9-43}$$

其中 w 是一个 n 维向量。

从式(9-43)可以很明显看出这是一个含有等式约束条件下的条件极值问题，因此用拉格朗日乘数法就能解决。进一步可构造如下拉格朗日函数

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w) \tag{9-44}$$

其中 β_i 是拉格朗日乘子。最后，通过对式子中所有的参数求偏导，令其为 0 便可求解得到所有未知变量。

此时，我们接着看如下优化问题

$$\begin{aligned} &\min_w \ f(w) \\ &s.t. \ \ g_i(w) \le 0, i = 1, \cdots, k. \\ &\quad\ \ h_i(w) = 0, i = 1, \cdots, l. \end{aligned} \tag{9-45}$$

从式(9-45)可以看出，与式(9-43)明显不同的就是在式(9-45)中多了不等式约束条件。因此，为了解决这类问题需要定义如下所示的广义拉格朗日乘数法（Generalized Lagrangian）[①]：

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w) \tag{9-46}$$

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

其中 α_i 和 β_i 都是拉格朗日乘子，但接下来的求解过程与之前就大相径庭了。

9.6.2 原始优化问题

根据式(9-45)(9-46)考虑如下定义：

$$\theta_{\mathcal{P}}(w)=\max_{\alpha,\beta:\alpha_i\geq 0}\mathcal{L}(w,\alpha,\beta) \tag{9-47}$$

式(9-47)表示的含义是求得最大化 $\mathcal{L}(w,\alpha,\beta)$ 时 α,β 的取值，即 α,β 作为自变量与 w 无关，最终求得的结果 $\theta_{\mathcal{P}}$ 是关于 w 的函数。

因此，如果原约束条件 $g_i(w)\leq 0$ 和 $h_i(w)=0$ 均成立，那么式(9-47)即等价为

$$\max_{\alpha,\beta:\alpha_i\geq 0}\left[\sum_{i=1}^{k}\alpha_i g_i(w)+\sum_{i=1}^{l}\beta_i h_i(w)\right] \tag{9-48}$$

则此时有 $\theta_{\mathcal{P}}(w)=f(w)+0$。

同时，我们现在来做这样一个假设，如果存在 g_i 或 h_i 使得原约束条件不成立，即 $g_i(w)>0$ 或者 $h_i(w)\neq 0$，那在这样的条件下 $\theta_{\mathcal{P}}$ 会发生什么变化呢？如果 $g_i(w)>0$，为了最大化 $\mathcal{L}$，只需要取 α_i 为无穷大，则此时 $\mathcal{L}$ 为无穷大，但这样没有意义；同样，如果 $h_i(w)\neq 0$，取 β 为无穷大（h_i 与 β 同号），则最后结果同样会无穷大。于是在这种情况下便能得到 $\theta_{\mathcal{P}}(w)=\infty$。

进一步，结合上述两种情况就能得到下面这个式子：

$$\theta_{\mathcal{P}}(w)=\begin{cases}f(w), & \text{if } w \text{ satisfies primal constraints}\\ \infty, & \text{otherwise}\end{cases} \tag{9-49}$$

再进一步，在满足约束条件的情况下最小化 $\theta_{\mathcal{P}}(w)$ 就等同于式(9-45)中所要求解的问题。于是便能得到如下定义

$$p^*=\min_w\theta_{\mathcal{P}}(w)=\min_w\max_{\alpha,\beta:\alpha_i\geq 0}\mathcal{L}(w,\alpha,\beta) \tag{9-50}$$

同时，将式(9-50)其称为原始优化问题（Primal Optimization Problem）。

9.6.3 对偶优化问题

接下来继续定义

$$\theta_{\mathcal{D}}(\alpha,\beta)=\min_w\mathcal{L}(w,\alpha,\beta) \tag{9-51}$$

式(9-51)表示的含义是求得最小化 $\mathcal{L}(w,\alpha,\beta)$ 时 w 的取值，即 w 作为自变量（与 α,β 无关），最终求得的结果 $\theta_{\mathcal{D}}$ 是关于 α,β 的函数。

此时便能定义出原问题的对偶问题

$$d^*=\max_{\alpha,\beta:\alpha_i\geq 0}\theta_{\mathcal{D}}(\alpha,\beta)=\max_{\alpha,\beta:\alpha_i\geq 0}\min_w\mathcal{L}(w,\alpha,\beta) \tag{9-52}$$

并将(9-52)称为对偶优化问题（Dual Optimization Problem）。

可以发现，式(9-50)和式(9-52)的唯一区别就是求解顺序发生了变化，前者是先最大化再最小化；而后者是先最小化然后再最大化。

那么原始问题和对偶问题有什么关系呢？ 为什么又要用对偶问题？ 通常情况下两者满足以下关系

$$d^* = \max_{\alpha,\beta:\alpha_i\geq 0} \min_{w} \mathcal{L}(w,\alpha,\beta) \leq \min_{w} \max_{\alpha,\beta:\alpha_i\geq 0} \mathcal{L}(w,\alpha,\beta) = p^* \tag{9-53}$$

证明

由式(9-47)(9-51)可知，对于任意的 w,α,β 有

$$\theta_{\mathcal{D}}(\alpha,\beta) = \min_{w} \mathcal{L}(w,\alpha,\beta) \leq \mathcal{L}(w,\alpha,\beta) \leq \max_{\alpha,\beta:\alpha_i\geq 0} \mathcal{L}(w,\alpha,\beta) = \theta_{\mathcal{P}}(w) \tag{9-54}$$

所以有

$$\theta_{\mathcal{D}}(\alpha,\beta) \leq \theta_{\mathcal{P}}(w) \tag{9-55}$$

进一步根据式(9-55)有

$$\max_{\alpha,\beta:\alpha_i\geq 0} \theta_{\mathcal{D}}(\alpha,\beta) \leq \min_{w} \theta_{\mathcal{P}}(w) \tag{9-56}$$

由此便得到了式(9-53)中的不等式。

在上述过程中，之所以要用对偶问题是因为直接对原始问题进行求解异常困难，所以一般会通过将其转转换为对偶问题进行求解。但就目前来看，两者并不完全等同，其解也就必然不会相同。所以下面就需要进一步介绍 KKT 条件。

9.6.4 KKT 条件

在 9.6.3 节中笔者介绍到，要想用对偶问题的解来代替原始问题的解，就必须使得两者等价。对于原始问题和对偶问题，假设函数 $f(w)$ 和 $g_i(w)$ 是凸函数，$h_i(w)$ 是仿射函数，且存在一个 w，使得不等式 $g_i(w)$ 严格可行（即对于所有的 i 都有 $g_i(w)<0$），则 w^* 和 α^*,β^* 同时是原始问题和对偶问题解的充分必要条件是 w^*,α^*,β^* 满足 （Karush-Kuhn-Tucker, KKT）条件[①]：

$$\frac{\partial}{\partial w_i}\mathcal{L}(w^*,\alpha^*,\beta^*) = 0, i = 1,\cdots,n \tag{9-57}$$

$$\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*,\alpha^*,\beta^*) = 0, i = 1,\cdots,l \tag{9-58}$$

$$\alpha_i^* g_i(w^*) = 0, i = 1,\cdots,k \tag{9-59}$$

$$g_i(w^*) \leq 0, i = 1,\cdots,k \tag{9-60}$$

$$\alpha_i^* \geq 0, i = 1,\cdots,k \tag{9-61}$$

其中式(9-59)称为 KKT 的对偶互补条件（Dual Complementarity Condition）。由此可以得到，如果 $\alpha_i^*>0$，则必有 $g_i(w)=0$，而这一点也将用来说明 SVM 仅仅只有少数的“支持向量”。同时，需要注意的是 KKT 条件中计算的是目标函数 $\mathcal{L}(w,\alpha,\beta)$ 中的未知数以及所有等式约束条件拉格朗日乘子的偏导数。

因此，若存在 w^*,α^*,β^* 满足上述 KKT 条件，那么 w^*,α^*,β^* 既是对偶问题的解同样也是原始问题的解。

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

9.6.5 计算示例

在介绍完对偶问题的相关求解原理后，下面再通过一个示例来进行说明。试求解以下优化问题：

$$\begin{aligned} \min_{x} \quad & f(x)=x_1^2+x_2^2 \\ \text{s.t.} \quad & h(x)=x_1-x_2-3=0 \\ & g(x)=(x_1-3)^2+x_2^2-2\le 0 \end{aligned} \tag{9-62}$$

由于式(9-62)中的优化问题相对简单，所以可以先通过作图来直观的理解一下，如图 9-11 所示。

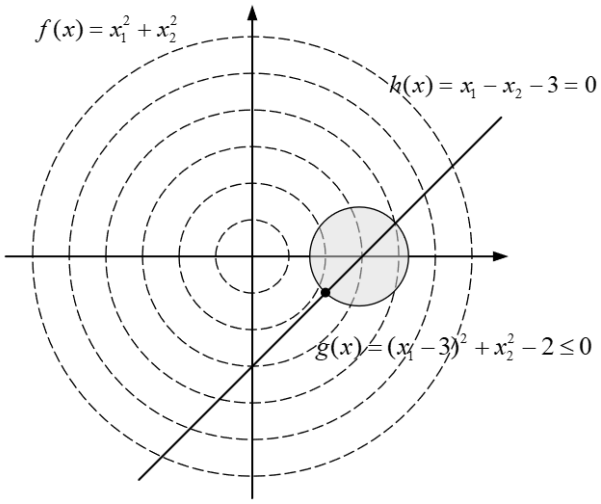

图 9-11 对偶问题计算示例图

如图 9-11 所示，黑色虚线圆环为目标函数 $f(x)$ 在水平面上的等高线；黑色直线为等式约束条件 $h(x)$ 的函数图像；整个灰色圆为不等式约束条件 $g(x)$ 的函数图像。从图示中可以看出，由于目标函数 $f(x)$ 需要约束条件 $h(x)$，这就意味着最终求得的最优解必须位于直线 $h(x)$ 上；同时，由于 $f(x)$ 还要满足约束条件 $g(x)$，所以解必定会在 $h(x)$ 与 $g(x)$ 相交的线段上。最后，由于是求解 $f(x)$ 在约束条件下的最小值，所以最优解就是图 9-11 中的黑色圆点。

进一步，针对这一问题可以得到如下广义拉格朗日函数

$$L(x,\alpha,\beta)=(x_1^2+x_2^2)+\alpha\left[(x_1-3)^2+x_2^2-2\right]+\beta(x_1-x_2-3) \tag{9-63}$$

根据式(9-50)可知，式(9-63)对应的原始问题为

$$p^*=\min_{x}\max_{\alpha,\beta:\alpha\ge 0} L(x,\alpha,\beta) \tag{9-64}$$

进一步，式(9-64)对应的对偶问题为

$$d^*=\max_{\alpha,\beta:\alpha\ge 0}\min_{x} L(x,\alpha,\beta) \tag{9-65}$$

因此，接下来便可以通过式(9-65)中的顺序来进行求解。

1) 最小化 $\mathcal{L}(x,\alpha,\beta)$

此时将 α,β 视为常数，那么这时 $L(x,\alpha,\beta)$ 就只是 x 的函数。因此可以通过令偏导数为零的方式来求得 $L(x,\alpha,\beta)$ 的最小值。故，此时有

$$\begin{cases}\dfrac{\partial L}{\partial x_1}=2x_1+\alpha(2x_1-6)+\beta=0\\ \dfrac{\partial L}{\partial x_2}=2x_2+2\alpha x_2-\beta=0\end{cases} \tag{9-66}$$

根据式(9-66)可以解得

$$\begin{cases}x_1=\dfrac{6\alpha-\beta}{2\alpha+2}\\ x_2=\dfrac{\beta}{2\alpha+2}\end{cases} \tag{9-67}$$

进一步，将式(9-67)代入目标函数(9-63)可得到

$$\theta_{\mathcal{D}}(\alpha,\beta)=-\frac{\beta^2+6\beta+4\alpha^2-14\alpha}{2(\alpha+1)} \tag{9-68}$$

2) 最大化 $\theta_{\mathcal{D}}(\alpha,\beta)$;

此时可以将 $\theta_{\mathcal{D}}(\alpha,\beta)$ 看成是一个二元函数求极值的问题。设 $D=\theta_{\mathcal{D}}(\alpha,\beta)$，则 D 分别对 α,β 求偏导并令其为 0 有

$$\begin{aligned}&\frac{\partial D}{\partial\alpha}=4\alpha^2+8\alpha-\beta^2-6\beta-14=0;\\ &\frac{\partial D}{\partial\beta}=\beta+3=0\end{aligned} \tag{9-69}$$

由式(9-69)得

$$\alpha_1=\frac{1}{2},\alpha_2=-\frac{5}{2},\beta=-3 \tag{9-70}$$

3) 求解原参数

根据 9.6.4 节中内容可知，欲使原问题式(9-64)与对偶问题式(9-65)同解，那么必须满足如下 KKT 条件

$$\frac{\partial}{\partial x_1}L(x^*,\alpha^*,\beta^*)=x_1^*(2+2\alpha)-6\alpha+\beta^*=0 \tag{9-71}$$

$$\frac{\partial}{\partial x_2}L(x^*,\alpha^*,\beta^*)=x_2^*(2+2\alpha)-\beta^*=0 \tag{9-72}$$

$$\frac{\partial}{\partial\beta}L(x^*,\alpha^*,\beta^*)=x_1^*-x_2^*-3=0 \tag{9-73}$$

$$\alpha^* g(x^*)=0 \tag{9-74}$$

$$g(x^*)\le 0 \tag{9-75}$$

$$\alpha^*\ge 0 \tag{9-76}$$

由此可得

$$\begin{cases} x_1^*=\dfrac{6\alpha^*-\beta^*}{2\alpha^*+2} \\ x_2^*=\dfrac{\beta^*}{2\alpha^*+2} \\ \alpha^*=\dfrac{1}{2} \\ \beta^*=-3 \end{cases} \tag{9-77}$$

最后可求得原始优化问题的解为

$$x_1^*=2,\ x_2^*=-1 \tag{9-78}$$

9.6.6 小结

在本节中，笔者首先介绍了广义的拉格朗日乘数法；然后进一步介绍了如何通过拉格朗日对偶方法来对原始问题进行求解；最后还通过一个实际的例子来对整个求解过程进行了示例。

9.7 SVM 优化问题

经过前面几节内容的介绍，我们已经知道了支持向量机背后的原理。同时，为了求解 SVM 中的目标函数，笔者还在前面两节内容中陆续介绍了拉格朗日乘数法和对偶性问题。接下来，在这篇文章中将开始正式介绍 SVM 的求解过程。同时，为了便于各位读者朋友循序渐进的了解整个求解过程，下面笔者会依次先后介绍硬间隔和软间隔中目标函数的求解步骤。

9.7.1 构造硬间隔广义拉格朗日函数

由 9.2 节的内容可知，SVM 硬间隔最终的优化目标为[①]

$$\begin{aligned} &\min_{w,b}\frac{1}{2}\|w\|^2 \\ &s.t.\ \ y^{(i)}(w^Tx^{(i)}+b)\ge 1, i=1,2,...m \end{aligned} \tag{9-79}$$

由此可以得到广义的拉格朗日函数为

$$\mathcal{L}(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}(w^Tx^{(i)}+b)-1\right] \tag{9-80}$$

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

其中 $\alpha_i \geq 0$ 为拉格朗日乘子，且同时记

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0 \tag{9-81}$$

进一步可得原始问题的对偶优化问题为

$$d^* = \max_{\alpha, \alpha_i \geq 0} \min_{w,b} \mathcal{L}(w, b, \alpha) \tag{9-82}$$

所以，为了求得对偶问题的解，需要按照式(9-82)中的顺序进行。

1) 关于参数 w,b 求 $\mathcal{L}$ 的极小值 $W(\alpha)$

为了求解式(9-82)中的对偶优化问题，首先需要最小化式(9-80)，即分别对 w,b 求偏导数并令其为 0 有

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \\ \frac{\partial \mathcal{L}}{\partial b} &= -\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{9-83}$$

进一步有

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{9-84}$$

到此便得到了权重 w 的解析表达式，而它对于理解 SVM 中核函数的利用有着重要作用。接着将式(9-83)(9-84)代入式(9-80)可得

$$W(\alpha) = \min_{w,b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} \tag{9-85}$$

化简得到式(9-85)的具体步骤为

$$\begin{aligned} \mathcal{L}(w, b, \alpha) &= \frac{1}{2} w^T w - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right] \\ &= \frac{1}{2} w^T w - \sum_{i=1}^{m} \alpha_i y^{(i)} w^T x^{(i)} - \sum_{i=1}^{m} \alpha_i y^{(i)} b + \sum_{i=1}^{m} \alpha_i \\ &= \frac{1}{2} w^T w - w^T \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} - b \sum_{i=1}^{m} \alpha_i y^{(i)} + \sum_{i=1}^{m} \alpha_i \end{aligned} \tag{9-86}$$

将式(9-83)和式(9-84)代入式(9-86)可得

$$\begin{aligned} W(\alpha) &= \frac{1}{2} w^T w - w^T w + \sum_{i=1}^{m} \alpha_i \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} w^T w \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)} \end{aligned} \tag{9-87}$$

到此，便得到了参数 w 的解析式。

2) 关于参数α求$W(\alpha)$的极大值

由式(9-85)可以得出如下优化问题

$$\begin{aligned}&\max_{\alpha} W(\alpha)=\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}\\&s.t.\ \ \alpha_i\geq 0, i=1,...,m\\&\sum_{i=1}^{m}\alpha_i y^{(i)}=0\end{aligned}\tag{9-88}$$

之所以式(9-83)中最后一个等式也会成为约束条件，是因为$W(\alpha)$是通过式(9-83)求解得到是$W(\alpha)$存在的前提。

进一步，假设$\alpha^*=(\alpha_1^*,\alpha_2^*,...,\alpha_m^*)^T$为对偶优化问题的解，那么为了使得对偶优化问题与原始优化问题同解需要满足如下 KKT 条件

$$\frac{\partial}{\partial w}\mathcal{L}(w^*,b^*,\alpha^*)=w^*-\sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)}=0\tag{9-89}$$

$$\frac{\partial}{\partial b}\mathcal{L}(w^*,b^*,\alpha^*)=-\sum_{i=1}^{m}\alpha_i^* y^{(i)}=0\tag{9-90}$$

$$\alpha_i^* g_i(w^*)=\alpha_i^*\left[y^{(i)}(w^*\cdot x^{(i)}+b^*)-1\right]=0,\quad i=1,2,...,,m\tag{9-91}$$

$$g_i(w^*)=-y^{(i)}(w^*\cdot x^{(i)}+b^*)+1\leq 0,\ i=1,2,...,m\tag{9-92}$$

$$\alpha_i^*\geq 0,\ i=1,2,...,m\tag{9-93}$$

根据式(9-89)可得

$$w^*=\sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)}\tag{9-94}$$

从式(9-94)可以看出，至少存在一个$\alpha_j^*>0$。因为如果所有的α_i均为 0，那么由式(9-94)可知此时的w^*也为 0，而这显然不是原始优化问题的解[①]。

同时，由式(9-91)可知，若存在$\alpha_j^*>0$，那么必有

$$y^{(j)}(w^*\cdot x^{(j)}+b^*)=1\tag{9-95}$$

根据式(9-95)可知，此时的样本点$x^{(j)}$是一个支持向量，而这也显示出的一个重要性质是超平面仅仅与支持向量有关。

进一步，将式(9-94)代入式(9-95)并注意$(y^{(j)})^2=1$可得

$$b^*=y^{(j)}-\sum_{i=1}^{m}\alpha_i^* y^{(i)}(x^{(i)}\cdot x^{(j)})\tag{9-96}$$

[①] 李航，统计机器学习，清华大学出版社

综上所述，对于任意给定线性可分数据集，首先可以根据式(9-88)中的优化问题求解得到α^*；然后再利用式(9-94)和(9-96)分别求解得到w^*,b^*；最后即可得到分离超平面。

9.7.2 硬间隔求解计算示例

为了使各位读者更加清楚 SVM 中硬间隔决策面的求解步骤，下面笔者将以一个实际的数据集来进行计算示例。现有数据集一共包含 7 个样本点，其中$x^{(1)}=(1,3)^T, x^{(2)}=(3,0)^T$，为负样本，$x^{(3)}=(3,5)^T, x^{(4)}=(2,7)^T$为正样本。黑色实现为决策面，黑色虚线为间隔边界，带圈的样本点为支持向量，如图 9-12 所示。

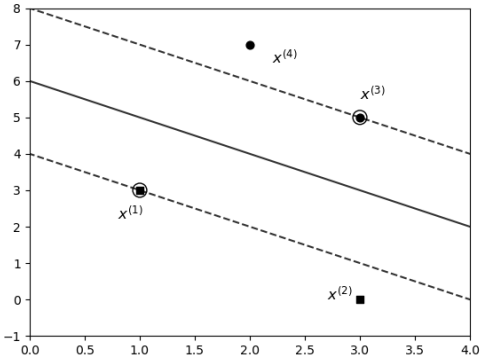

图 9-12 SVM 硬间隔示例图

由给定数据集以及式(9-88)可知，对偶问题为

$$\begin{aligned}&\max_{\alpha}\sum_{i=1}^{4}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{4}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}\\&=(\alpha_1+\alpha_2+\alpha_3+\alpha_4)-\frac{1}{2}(10\alpha_1^2+9\alpha_2^2+34\alpha_3^2+53\alpha_4^2+6\alpha_1\alpha_2\\&-36\alpha_1\alpha_3-46\alpha_1\alpha_4-18\alpha_2\alpha_3-12\alpha_2\alpha_4+82\alpha_3\alpha_4)\\&s.t.\ \ \alpha_i\geq 0, i=1,2,3,4\\&\quad -\alpha_1-\alpha_2+\alpha_3+\alpha_4=0\end{aligned}\tag{9-97}$$

注意：$\langle x+y,x+y\rangle=\langle x,x\rangle+\langle x,y\rangle+\langle y,x\rangle+\langle y,y\rangle$

此时由约束条件可知$\alpha_1=\alpha_3+\alpha_4-\alpha_2$，将其代入目标函数并记为

$$\begin{aligned}\varphi(\alpha_2,\alpha_3,\alpha_4)=2\alpha_3+2\alpha_4-\frac{13}{2}\alpha_2^2-4\alpha_3^2-\frac{17}{2}\alpha_4^2\\-2\alpha_2\alpha_3-10\alpha_2\alpha_4-10\alpha_3\alpha_4\end{aligned}\tag{9-98}$$

对$\alpha_2,\alpha_3,\alpha_4$分别求偏导并令其为 0 可得

$$\begin{aligned}\frac{\partial\varphi}{\partial\alpha_2}&=-13\alpha_2-2\alpha_3-10\alpha_4=0\\\frac{\partial\varphi}{\partial\alpha_3}&=2-2\alpha_2-8\alpha_3-10\alpha_4=0\\\frac{\partial\varphi}{\partial\alpha_4}&=2-10\alpha_2-10\alpha_3-17\alpha_4=0\end{aligned}\tag{9-99}$$

不过根据式(9-99)中的 3 个等式联立求解后发现，同时满足这 3 个式子的解并不存在，也就是说最终的解只可能在约束条件的边界出产生，即不考虑某些约束条件。因此，在式(9-99)中需要考虑如下边界情况：

①当$\alpha_2=0$时，根据式(9-99)中后两个等式可求得$\alpha_4=-1/9<0$不满足式(9-97)中的约束条件；

②当$\alpha_3=0$时，根据式(9-99)中第 1 和第 3 个等式可求得$\alpha_2<0$，也不满足约束条件；

③当$\alpha_4=0$时，根据式(9-99)中前两个等式求解后发现同样不满足约束条件；

④最后，只有当$\alpha_2=0,\alpha_4=0$时，才能求得满足约束条件的解，即此时$\alpha_1^*=0.25,\alpha_2^*=0,\alpha_3^*=0.25,\alpha_4^*=0$。对于其它情况，读者朋友可以自行验算。

进一步，根据上述求得的结果通过式(9-94)便可求得决策面中w为

$$w^*=-\frac{1}{4}\cdot(1,3)+\frac{1}{4}\cdot(3,5)=(\frac{1}{2},\frac{1}{2}) \tag{9-100}$$

同时，根据式(9-96)并任取α_1,α_3其中一个作为α_j即可求得b为

$$b^*=-1-(-\frac{1}{4}\cdot 10+\frac{1}{4}\cdot 18)=-3 \tag{9-101}$$

最后，可以得到决策面方程为

$$\frac{1}{2}x_1+\frac{1}{2}x_2-3=0 \tag{9-102}$$

9.7.3 构造软间隔广义拉格朗日函数

由 9.4 节的内容可知，SVM 软间隔最终的优化目标为

$$\begin{aligned}&\min_{w,b,\xi}\frac{1}{2}\|w\|^2+C\sum_{i=1}^{m}\xi_i\\&s.t.\ \ y^{(i)}(w^Tx^{(i)}+b)\geq 1-\xi_i,\ \ \xi_i\geq 0,i=1,2,...m\end{aligned} \tag{9-103}$$

由此可以得到广义的拉格朗日函数为[①]

$$\mathcal{L}(w,b,\xi,\alpha,r)=\frac{1}{2}\|w\|^2+C\sum_{i=1}^{m}\xi_i-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}(w^Tx^{(i)}+b)-1+\xi_i\right]-\sum_{i=1}^{m}r_i\xi_i \tag{9-104}$$

其中$\alpha_i\geq 0,r_i\geq 0$为拉格朗日乘子,且同时记

$$\begin{aligned}&g_i(w)=-y^{(i)}(w^Tx^{(i)}+b)+1-\xi_i\leq 0\\&h_i(\xi)=-\xi_i\leq 0;\ \ i=1,2,...,m\end{aligned} \tag{9-105}$$

进一步可以得到原始问题的对偶优化问题为

$$d^*=\max_{\alpha,r}\min_{w,b,\xi}\mathcal{L}(w,b,\xi,\alpha,r) \tag{9-106}$$

所以，为了求得对偶问题的解，需要按照式(9-106)中的顺序进行。

1) 关于参数w,b,ξ求$\mathcal{L}$的极小值$W(\alpha,r)$

首先需要最小化式(9-104)，即分别对w,b,ξ求偏导数并令其为 0 有

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w}&=w-\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}=0\\ \frac{\partial \mathcal{L}}{\partial b}&=-\sum_{i=1}^{m}\alpha_i y^{(i)}=0\\ \frac{\partial \mathcal{L}}{\partial \xi_i}&=C-\alpha_i-r_i=0\end{aligned}\tag{9-107}$$

进一步有

$$w=\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\tag{9-108}$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)}=0\tag{9-109}$$

$$C-\alpha_i-r_i=0\tag{9-110}$$

接着，将式(9-108)到(9-110)代入式(9-104)有

$$\begin{aligned}W(\alpha,r)&=\frac{1}{2}w^Tw+C\sum_{i=1}^{m}\xi_i-w^Tw+\sum_{i=1}^{m}\alpha_i-\sum_{i=1}^{m}\alpha_i\xi_i-\sum_{i=1}^{m}r_i\xi_i\\ &=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}w^Tw+\sum_{i=1}^{m}\xi_i\left(C-\alpha_i-r_i\right)\\ &=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}w^Tw\end{aligned}\tag{9-111}$$

此时可以发现，式(9-111)化简后r已经消去了。

2) **关于参数α求$W(\alpha)$的极大值**

由式(9-111)求α的极大可以得出如下优化问题

$$\max_{\alpha}W(\alpha)=\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}w^Tw\tag{9-112}$$

$$s.t.\sum_{i=1}^{m}\alpha_i y^{(i)}=0\tag{9-113}$$

$$C-\alpha_i-r_i=0,\ i=1,2,...,m\tag{9-114}$$

$$\alpha_i\geq 0,\ i=1,2,...,m\tag{9-115}$$

$$r_i\geq 0,\ i=1,2,...,m\tag{9-116}$$

接着，将式(9-116)代入式(9-114)便可得到化简后的约束条件

$$0\leq\alpha_i\leq C\tag{9-117}$$

最后便可得到最终的对偶优化问题

$$\begin{aligned}&\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}\\&s.t.\ 0\le\alpha_i\le C, i=1,2,...,m\\&\sum_{i=1}^{m}\alpha_i y^{(i)}=0\end{aligned}\tag{9-118}$$

从式(9-118)可以发现，SVM 软间隔的对偶优化问题同硬硬间隔的对偶优化问题仅仅只是在约束问题上发生了变化。

现在假设$\alpha^*=(\alpha_1^*,\alpha_2^*,...,\alpha_m^*)^T$为对偶优化问题(9-118)的解，那么为了使得对偶优化问题与原始优化问题同解需要满足如下 KKT 条件

$$\frac{\partial}{\partial w}\mathcal{L}(w^*,b^*,\xi^*,\alpha^*,r^*)=w^*-\sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)}=0\tag{9-119}$$

$$\frac{\partial}{\partial b}\mathcal{L}(w^*,b^*,\xi^*,\alpha^*,r^*)=-\sum_{i=1}^{m}\alpha_i^* y^{(i)}=0\tag{9-120}$$

$$\frac{\partial}{\partial \xi}\mathcal{L}(w^*,b^*,\xi^*,\alpha^*,r^*)=C-\alpha^*-r^*=0\tag{9-121}$$

$$\alpha_i^* g_i(w^*)=\alpha_i^*\left[y^{(i)}(w^*\cdot x^{(i)}+b^*)-1+\xi_i^*\right]=0,\quad i=1,2,...,,m\tag{9-122}$$

$$r_i^* h_i(\xi^*)=r_i^*\xi_i^*=0;\quad i=1,2,...,m\tag{9-123}$$

$$g_i(w^*)=-y^{(i)}(w^*\cdot x^{(i)}+b^*)+1-\xi_i^*\le 0,\quad i=1,2,...,m\tag{9-124}$$

$$h_i(\xi^*)=-\xi_i\le 0;\quad i=1,2,...,m\tag{9-125}$$

$$\alpha_i^*\ge 0,\ r_i^*\ge 0;\quad i=1,2,...,m\tag{9-126}$$

因此，根据式(9-119)可得

$$w^*=\sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)}\tag{9-127}$$

从式(9-127)可以看出，至少存在一个$0<\alpha_j^*\le C$。因为如果所有的α_i均为 0，那么由式(9-127)可知此时的w^*同样为 0，而这显然不是原始优化问题的解。进一步，若存在$0<\alpha_j^*<C$，那么由式(9-121)知，此时对应的$r_j^*>0$；再由式(9-123)可知，此时必有$\xi_j^*=0$；所以最后由式(9-122)可知，此时必有

$$y^{(j)}(w^*\cdot x^{(j)}+b^*)=1\tag{9-128}$$

进一步，将式(9-127)代入式(9-128)可得

$$b^*=y^{(j)}-\sum_{i=1}^{m}\alpha_i^* y^{(i)}(x^{(i)}\cdot x^{(j)})\tag{9-129}$$

综上所述，对于任意给定数据集，首先可以根据式(9-118)中的优化问题求解得到α^*；然后再利用式(9-127)和(9-129)分别求解得到w^*, b^*；最后即可得到分离超平面。

9.7.4 软间隔中的支持向量

在本章伊始，以及 9.7.1 和 9.7.2 节内容中，笔者都提到了“支持向量”这个词，并且还介绍到位于间隔边界上的样本点就是支持向量。不过那到底支持向量是什么呢？直白点说，能够影响决策面形成样本点就是支持向量。例如从图 9-12 可以看出，对于影响最后决策面形成的就只有 $x^{(1)}, x^{(3)}$ 这两个样本点。同时，从硬间隔的定义可以看出，在求解得到决策面后所有样本点的分布只存在两种情况。第一种是位于间隔边界上，而另外一种就是在间隔边界以外。因此，在硬间隔中影响决策面形成的就只有位于间隔边界上的样本点，即支持向量。

但是从软间隔的定义可以看出，由于软间隔允许样本点被错误分类，并且其程度可用通过惩罚系数C来进行控制。因此可以得出在软间隔中，C可以影响决策面的位置，并且支持向量也不仅仅只会位于间隔边界上，如图 9-13 所示。

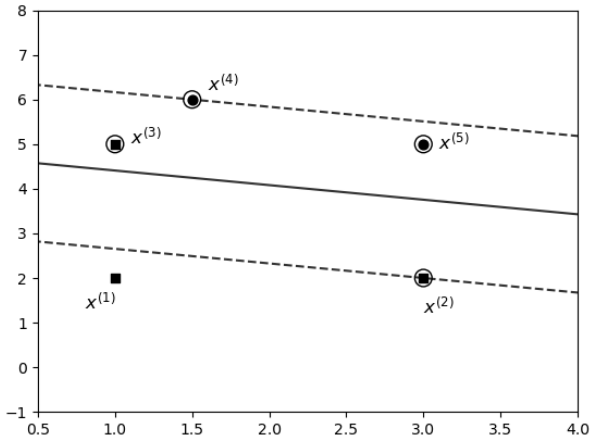

图 9-13 软间隔示例图

从图 9-13 可以看出，为了增强模型最终的泛化能力，软间隔目标函数在求解过程将 $x^{(3)}, x^{(5)}$ 这两个样本点划分到了间隔边界以内，并且样本点 $x^{(5)}$ 还被错误的划分到了另外一个类别中（当然 $x^{(3)}, x^{(5)}$ 本身也可能是异常样本）。因此可以得出，在 SVM 软间隔的建模过程中，影响决策面位置的除了位于间隔边界上的样本还有位于间隔边界内的样本点，所以这些都被称之为支持向量。因为如果去掉 $x^{(3)}$ 这个被错分类的样本点，那么最终得到的决策面会更加趋于水平。

同时，由式(9-121)到(9-124)可知，当$0 < \alpha_i < C$时，则$\xi_i = 0$，此时支持向量 $x^{(i)}$ 将位于间隔边界上；当$\alpha_i = C, 0 < \xi_i < 1$时，则分类正确，此时支持向量 $x^{(i)}$ 将位于间隔边界与决策面之间；当$\alpha_i = C, \xi_i = 1$时，此时支持向量 $x^{(i)}$ 将位于决策面上；当$\alpha_i = C, \xi_i > 1$时，则分类错误，此时支持向量 $x^{(i)}$ 将位于决策面的另一侧[①]。进一步，可以总结得到

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1 \end{aligned} \tag{9-130}$$

[①] 李航，统计机器学习，清华大学出版社

9.7.5 小结

在本节中，笔者首先介绍了在求解 SVM 模型中如何构造硬间隔的广义的拉格朗日函数以及其对应的对偶问题；接着通过求解对偶问题中 $\mathcal{L}(w,b,\alpha)$ 的极小值得到了 w 的计算解析式，需要提醒的是 w 的表达式也是理解 SVM 核函数的关键；然后介绍了以一个实际的示例介绍了 SVM 硬间隔的求解过程；最后，笔者还介绍了如何构造软间隔中的拉格朗日函数以及软间隔中的支持向量，而 SVM 软间隔的实际求解过程将在下一节中进行介绍。

9.8 SMO 算法

在 9.7 节中，笔者分别就 SVM 中软间隔与硬间隔目标函数的求解过程进行了介绍。但是在实际应用过程中，从效率的角度来讲那样的做法显然是不可取的，尤其是在大规模数据样本和稀疏数据中[①]。在接下来的这节内容中，笔者将会介绍一种新的求解算法，序列最小化优化算法来解决这一问题。

序列最小优化算法（Sequential Minimal Optimization, SMO）于 1998 年由 John Platt 所提出，并且 SMO 算法初次提出的目的就是为了解决 SVM 的优化问题[②]。SMO 算法是一种启发式的算法，它在求解过程中通过以分析的方式来定位最优解可能存在的位置，从而避免了传统方法在求解中所遭遇的大量数值计算问题，并且最终以迭代的方式来求得最优解。在正式介绍 SMO 算法之前，笔者将先来介绍 SMO 算法的基本原理坐标上升算法（Coordinate Ascent）。

9.8.1 坐标上升算法

在第 2.5 节内容中，笔者详细的介绍了什么是梯度下降算法以及梯度下降算法的作用。对于一个待优化的目标函数来说，在初始化一个起始位置后，便可以以该点为基础每次沿着该点梯度的反方向向前移动一小步，以此来迭代求解得到目标函数的全局（局部）最优解。而所谓的坐标上升（下降）算法可以看作是初始位置只沿着其中的一个（或几个）方向移动来求解得到目标函数的全局（局部）最优解[③]，如图 9-14 所示。

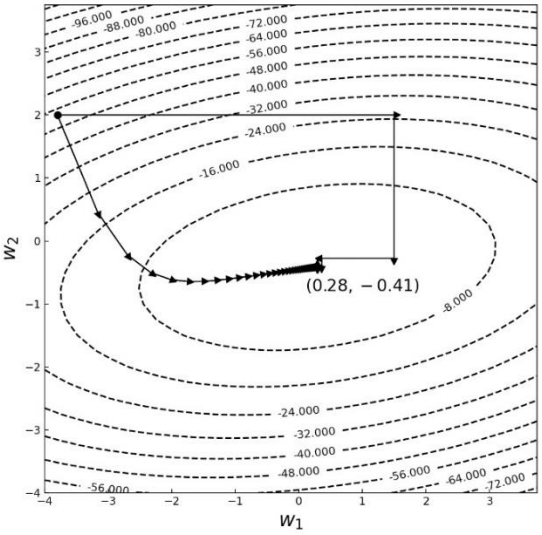

图 9-14 梯度上升与坐标上升图

① John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Microsoft Research Technical Report MSR-TR-98-14.

② https://en.wikipedia.org/wiki/Sequential_minimal_optimization

③ https://en.wikipedia.org/wiki/Coordinate_descent

在图 9-14 中，虚线为目标函数 $J(w_1,w_2)=-0.5(w_1-1)^2-(2w_2+1)^2-0.5(w_1-w_2)^2$ 的等高线，黑色箭头曲线为梯度上升算法最大化目标函数 $J(w_1,w_2)$ 的求解过程，而黑色箭头折线为坐标上升算法最大化目标函数的求解过程。

具体的，对于待求解目标函数 $J(w_1,w_2,...,w_n)$ 来说可以通过如下步骤来进行求解

(1) 随机初始化向量 $w=(w_1,w_2,...,w_n)$ 为初始参数值；

(2) 在 $w_1,w_2,...,w_n$ 中依次选择 $w_i,i=1,2,...,n$ 为变量固定其它参数为常量，然后求目标函数关于 w_i 的导数并令其为 0 求得 w_i；

(3) 重复执行步骤(2)直到目标函数收敛或者是误差小于某一阈值结束。

例如在上面这个示例中 w_1,w_2 的求解表达式分别为

$$\begin{aligned} w_1^{new}&=\frac{1}{2}(w_2^{old}+1)\\ w_2^{new}&=\frac{1}{9}(w_1^{new}-4)\end{aligned} \tag{9-131}$$

那么在初始化一组 w_1^{old},w_2^{old} 后，便可以通过式(9-131)来迭代求解得到 w_1,w_2 的解。

同时，上述步骤(2)对于 w_i 顺序的选择这里采用了最为简单的按顺序依次进行，一种更优的做法便是每次选择余下常量中能够使得目标函数产生最大增量参数作为优化对象。

9.8.2 SMO 算法思想

根据 9.7.3 节内容可知，SVM 软间隔最终需要求解的目标函数为

$$\begin{aligned} &\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}\\ &s.t.\ 0\le\alpha_i\le C,i=1,2,...,m\\ &\sum_{i=1}^{m}\alpha_i y^{(i)}=0\end{aligned} \tag{9-132}$$

假设随机初始化后的 α_i 均满足式(9-132)中的约束条件，现在通过坐标上升算法来求解 α。如果此时固定 $\alpha_2,...,\alpha_m$ 为常量，α_1 为变量来求解 α_1，那么这样能够求解得到 α_1 吗？答案是不能[①]。因为根据式(9-132)中第 2 个约束条件有

$$\alpha_1 y^{(1)}=-\sum_{i=2}^{m}\alpha_i y^{(i)} \tag{9-133}$$

进一步，在式(9-133)两边同时乘上 $y^{(1)}$ 有

$$\alpha_1=-y^{(1)}\sum_{i=2}^{m}\alpha_i y^{(i)} \tag{9-134}$$

根据式(9-134)可知，α_1 完全取决于 $\alpha_2,...,\alpha_m$，如果 $\alpha_2,...,\alpha_m$ 固定那么也就意味着 α_1 也是固定的。因此，在这样的情况下每次至少需要同时选择两个参数为变量，同时再固定其他参数为常量才能够最终求得所有参数。

此时，假设固定 $\alpha_3,...,\alpha_m$ 不变来求解参数 α_1,α_2。根据式(9-132)中的约束条件有

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)} \tag{9-135}$$

由于此时式(9-135)右边是固定的，因此可以用一个常数 ζ 来表示

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta \tag{9-136}$$

进一步，可以将 α_1 表示为

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)} \tag{9-137}$$

此时，式(9-132)中的目标函数便可以改写为

$$W(\alpha_1, \alpha_2, ..., \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, ..., \alpha_m) \tag{9-138}$$

在式(9-138)中，由于此时固定 $\alpha_3, ..., \alpha_m$ 为常数，因此其可以简单表示为一个关于 α_2 的一元二次多项式（这一点也可以从式(9-98)中看出）

$$a\alpha_2^2 + b\alpha_2 + c \tag{9-139}$$

接着，对于式(9-139)来说通过令 α_2 的导数为 0 便可以轻易求得 α_2 的取值；然后再将 α_2 代入式(9-136)即可得到 α_1 的值。进一步，按照相同的过程便可求得 $\alpha_3, ..., \alpha_m$ 的值，最后再重复执行上述整个过程便可以迭代求解得到 $\alpha_1, \alpha_2, ..., \alpha_m$ 。

以上就是 SMO 算法求解的主要思想。虽然看起来不太复杂，但是里面仍旧有很多值得深究的内容，下面开始正式介绍 SMO 算法的原理。

9.8.3 SMO 算法原理

为了更加广义的表示软间隔中的优化问题，可以通过如下形式来表示待求解的优化问题

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K(x^{(i)}, x^{(j)}) \\ &s.t.\ 0 \le \alpha_i \le C, i = 1, 2, ..., m \\ &\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{9-140}$$

其中 $K(\cdot)$ 为任意核函数。

进一步，不失一般性，这里假设首先选择参数 α_1, α_2 为变量，固定其它参数为常量。于是式(9-140)便可以改写成如下形式[①]

$$\begin{aligned} \max_{\alpha_1, \alpha_2} \alpha_1 + \alpha_2 &- \frac{1}{2} \alpha_1^2 K_{11} - \alpha_1 \alpha_2 y^{(1)} y^{(2)} K_{12} - \alpha_1 y^{(1)} \sum_{i=3}^{m} \alpha_i y^{(i)} K_{1i} \\ &- \frac{1}{2} \alpha_2^2 K_{22} - \alpha_2 y^{(2)} \sum_{i=3}^{m} \alpha_i y^{(i)} K_{2i} + \Psi_{constant} \\ s.t.\ 0 \le \alpha_i \le C, i &= 1, 2 \\ \alpha_1 y^{(1)} + \alpha_2 y^{(2)} &= -\sum_{i=3}^{m} \alpha_i y^{(i)} = \zeta \end{aligned} \tag{9-141}$$

[①] John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Microsoft Research Technical Report MSR-TR-98-14.

其中 $K_{ij}=K(x^{(i)},x^{(j)})$， $\Psi_{constant}$ 表示与 α_1,α_2 无关的常量。
同时，记

$$\begin{aligned} g(x)&=\sum_{i=1}^{m}\alpha_i y^{(i)}K(x,x^{(i)})+b \\ v_i&=\sum_{j=3}^{m}\alpha_j y^{(j)}K(x^{(i)},x^{(j)})=g(x^{(i)})-\sum_{j=1}^{2}\alpha_j y^{(j)}K(x^{(i)},x^{(j)})-b,\ i=1,2 \end{aligned} \tag{9-142}$$

则目标函数(9-141)可以改写为

$$\begin{aligned} W(\alpha_1,\alpha_2)=\alpha_1+\alpha_2-\frac{1}{2}\alpha_1^2K_{11}-\alpha_1\alpha_2y^{(1)}y^{(2)}K_{12}-\alpha_1y^{(1)}v_1 \\ -\frac{1}{2}\alpha_2^2K_{22}-\alpha_2y^{(2)}v_2+\Psi_{constant} \end{aligned} \tag{9-143}$$

将 $\alpha_1=(\zeta-\alpha_2y^{(2)})y^{(1)}$ 代入式(9-143)便可以得到一个只含有变量 α_2 目标函数

$$\begin{aligned} W(\alpha_2)=(\zeta-\alpha_2y^{(2)})y^{(1)}+\alpha_2-\frac{1}{2}(\zeta-\alpha_2y^{(2)})^2K_{11} \\ -(\zeta-\alpha_2y^{(2)})\alpha_2y^{(2)}K_{12}-(\zeta-\alpha_2y^{(2)})v_1-\frac{1}{2}\alpha_2^2K_{22}-\alpha_2y^{(2)}v_2 \end{aligned} \tag{9-144}$$

进一步式(9-144)关于 α_2 的导数为

$$\begin{aligned} \frac{\partial W}{\partial\alpha_2}=-y^{(1)}y^{(2)}+1+\zeta y^{(2)}K_{11}-\alpha_2K_{11}+2\alpha_2K_{12} \\ -\zeta y^{(2)}K_{12}+v_1y^{(2)}-\alpha_2K_{22}-y^{(2)}v_2 \end{aligned} \tag{9-145}$$

令式(9-145)为 0 可以得到

$$\alpha_2=\frac{y^{(2)}\left(y^{(2)}-y^{(1)}+\zeta K_{11}-\zeta K_{12}+v_1-v_2\right)}{K_{11}-2K_{12}+K_{22}} \tag{9-146}$$

此时，记

$$\begin{aligned} \eta&=K_{11}-2K_{12}+K_{22} \\ E_i&=g(x^{(i)})-y^{(i)}=\left(\sum_{j=1}^{m}\alpha_j y^{(j)}K(x^{(i)},x^{(j)})+b\right)-y^{(i)},\ i=1,2 \end{aligned} \tag{9-147}$$

那么在初始化一组 $\alpha_1^{old},\alpha_2^{old}$ 后，将式(9-147)和 $\zeta=\alpha_1^{old}y^{(1)}+\alpha_2^{old}y^{(2)}$ 代入式(9-146)有

$$\begin{aligned} \alpha_2^{new}=\frac{y^{(2)}}{\eta}\Big[y^{(2)}-y^{(1)}+\left(\alpha_1^{old}y^{(1)}+\alpha_2^{old}y^{(2)}\right)K_{11} \\ -\left(\alpha_1^{old}y^{(1)}+\alpha_2^{old}y^{(2)}\right)K_{12}+g(x_1)-\sum_{j=1}^{2}\alpha_j^{old}y^{(j)}K_{1j} \\ -b-g(x_2)+\sum_{j=1}^{2}\alpha_j^{old}y^{(j)}K_{2j}+b\Big] \end{aligned} \tag{9-148}$$

将式(9-148)进一步化简后可得

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y^{(2)}(E_1 - E_2)}{\eta} \tag{9-149}$$

到此，便初步求得了α_2的求解表达式。为什么是初步呢？因为此时的求得的α_2还没有经过约束条件裁剪。

由式(9-141)中的第 1 个约束条件可知，α_1, α_2的解只能位于$[0, C]\times[0, C]$这个正方形盒子中；进一步，由式(9-141)中的第 2 个约束条件可知，α_1, α_2还必须位于平行于盒子对角线的线段上，如图 9-15 所示。

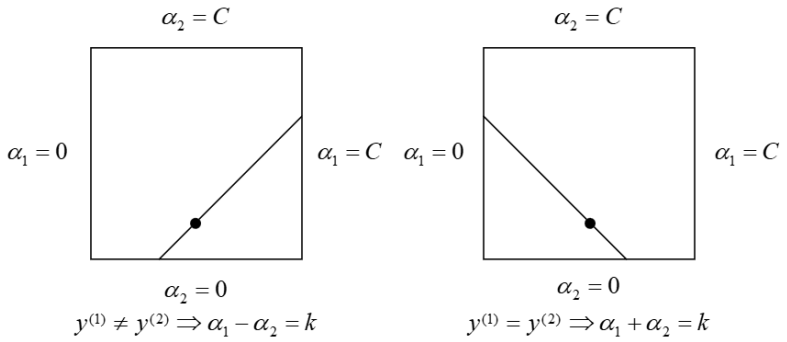

图 9-15 约束条件下的α_1, α_2修正图

如图 9-15 所示，α_1, α_2的解只能位于盒子内部的线段上。由式(9-144)可知根据式(9-149)求得到α_2^{new}的值必定位于线段所在的直线上，因此还需要使得α_2^{new}满足

$$L \le \alpha_2^{new} \le H \tag{9-150}$$

其中，L与H是图 9-15 中线段的两个端点。

从图 9-15 可以看出，如果$y^{(1)} \neq y^{(2)}$则

$$L = \max(0, \alpha_2^{old} - \alpha_1^{old}), \quad H = \min(C, C + \alpha_2^{old} - \alpha_1^{old}) \tag{9-151}$$

如果$y^{(1)} = y^{(2)}$则

$$L = \max(0, \alpha_2^{old} + \alpha_1^{old} - C), \quad H = \min(C, \alpha_2^{old} + \alpha_1^{old}) \tag{9-152}$$

因此，α_2^{new}裁剪后的值应该为

$$\alpha_2^{new,clipped} = \begin{cases} H, & \alpha_2^{new} \ge H \\ \alpha_2^{new}, & L < \alpha_2^{new} < H \\ L, & \alpha_2^{new} \le L \end{cases} \tag{9-153}$$

进一步，由$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$和$\zeta = \alpha_1^{old} y^{(1)} + \alpha_2^{old} y^{(2)}$可得

$$\begin{aligned} \alpha_1^{new} &= (\alpha_1^{old} y^{(1)} + \alpha_2^{old} y^{(2)} - \alpha_2^{new,clipped} y^{(2)}) y^{(1)} \\ &= \alpha_1^{old} + y^{(1)} y^{(2)} (\alpha_2^{old} - \alpha_2^{new,clipped}) \end{aligned} \tag{9-154}$$

到此就介绍完了α_1, α_2的求解步骤。进一步，可以按照类似的方法求解得到$\alpha_3, ..., \alpha_m$；最后再迭代整个过程便可以求解得到$\alpha_1, \alpha_2, ..., \alpha_m$的最优解。

9.8.4 偏置b求解

在每次计算得到α_i,α_j两个变量后，都需要同步的来更新偏置b的值。例如当求得$0<\alpha_1^{new}<C$时，根据式(9-130)中的第 3 个 KKT 条件可知

$$y^{(1)}(w^T x^{(1)}+b)=y^{(1)}g(x^{(1)})=y^{(1)}\left(\sum_{i=1}^{m}\alpha_i y^{(i)}K_{1i}+b\right)=1 \tag{9-155}$$

进一步可得

$$\sum_{i=1}^{m}\alpha_i y^{(i)}K_{1i}+b=y^{(1)} \tag{9-156}$$

于是根据式(9-156)有

$$b_1^{new}=y^{(1)}-\sum_{i=3}^{m}\alpha_i y^{(i)}K_{1i}-\alpha_1^{new}y^{(1)}K_{11}-\alpha_2^{new,clipped}y^{(2)}K_{12} \tag{9-157}$$

由式(9-147)中E_i的定义可知

$$E_1=\sum_{i=3}^{m}\alpha_i y^{(i)}K_{1i}+\alpha_1^{old}y^{(1)}K_{11}+\alpha_2^{old}y^{(2)}K_{12}+b^{old}-y^{(1)} \tag{9-158}$$

根据式(9-158)可知，式(9-157)的前两项可以改写为

$$y^{(1)}-\sum_{i=3}^{m}\alpha_i y^{(i)}K_{1i}=b^{old}-E_i+\alpha_1^{old}y^{(1)}K_{11}+\alpha_2^{old}y^{(2)}K_{12} \tag{9-159}$$

最后，将式(9-159)代入式(9-157)可得

$$b_1^{new}=b^{old}-E_1-y^{(1)}K_{11}(\alpha_1^{new}-\alpha_1^{old})-y^{(2)}K_{12}(\alpha_2^{new,clipped}-\alpha_2^{old}) \tag{9-160}$$

同理可得，当$0<\alpha_2^{new,clipped}<C$时有

$$b_2^{new}=b^{old}-E_2-y^{(1)}K_{12}(\alpha_1^{new}-\alpha_1^{old})-y^{(2)}K_{22}(\alpha_2^{new,clipped}-\alpha_2^{old}) \tag{9-161}$$

同时，当$\alpha_1^{new},\alpha_2^{new,clipped}$同时满足条件时，那么$b_1^{new},b_2^{new}$均是有效的，且两者相等。如果$\alpha_1^{new},\alpha_2^{new,clipped}$为$0$或者$C$，那么所有在$b_1^{new},b_2^{new}$之间的值均满足 KKT 条件的值，此时选择两者的均值作为b^{new}。因此，偏置b的计算公式为

$$b^{new}=\begin{cases}b_1^{new}, & 0<\alpha_1^{new}<C\\ b_2^{new}, & 0<\alpha_2^{new,clipped}<C\\ (b_1^{new}+b_2^{new})/2, & otherwise\end{cases} \tag{9-162}$$

最后，需要注意的是在上述求解过程中，最终得到w的解是唯一的，而偏置b的解却可能不唯一（它存在于一个区间中），详细证明过程可以参见《数据挖掘中的新方法-支持向量机》一书[①]第 5.3 节中的介绍。当然，除了理论上的证明还可从另外一个更直观的角度来理解。如果w唯一而b不唯一也就意味着决策面并不唯一，而这在 SVM 软间隔中显然是成立的。因为此时允许个别样本被分类错误，但是这些被错分的样本是不确定的。也就是说，在这样的情况下可以存在不同的分类决策面，而它们的分类间隔却相同。

[①] 邓乃扬,田英杰. 数据挖掘中的新方法-支持向量机. 北京：科学出版社, 2004.

9.8.5 SVM 算法求解示例

经过 9.8.3 和 9.8.4 两节内容的介绍，相信各位读者朋友对于如何通过 SMO 算法来求解 SVM 中的参数已经有了一定的了解。同时，对于整个求解过程还可以通过如下一段伪代码来进行表示[①]

输入：

C：惩罚项系数；

tol：误差容忍度；

max_passes：当α_i不再发生变化时继续迭代更新的最大次数；

$((x^{(1)}, y^{(1)}),...,(x^{(m)}, y^{(m)}))$：训练集；

输出：

$\alpha \in \mathbb{R}^m$：求解得到的拉格朗日乘子；

$b \in \mathbb{R}$：求解得到的偏置。

```
初始化所有 alpha_i = 0, b = 0, passes = 0
while (passes < max_passes)
    num_changed_alphas = 0
    for i = 1,...,m
        计算 E_i
        if ((y_i*E_i < -tol and a_i < C)||(y_i*E_i > tol and alpha_i > 0))
            随机选择 j，且 j 不等于 i
            计算 E_j
            保存: alpha_i_old = alpha_i,alpha_j_old = alpha_j
            计算 L 和 H
            if (L == H):
                continue
            计算 eta
            if (eta >- 0):
                continue
            计算 alpha_j 并裁剪
            if (|alpha_j - alpha_j_old| < 10e-5):
                continue
            分别计算 alpha_i, b_1, b_2
            计算 b
            num_changed_alphas += 1
    if (num_changed_alphas == 0):
        passes += 1
    else:
        passes = 0
```

当然，根据上述伪代码的描述，还可以通过代码将其完整的实现，具体可以参见 Book/Chapter09/03_svm_smo.py 文件。同时，根据实现的代码，如果以 9.7.4 节中图 9-13 里的数据样本为输入，且惩罚系设为$C = 0.2$，那么最终的求解结果为

```
data_x = np.array([[5, 1], [0, 2], [1, 5], [3., 2], [1, 2], [3, 5], [1.5,
6], [4.5, 6], [0, 7]])
data y = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1])
alphas,  b  =  smo(C=.2,  tol=0.001,  max passes=200,  data x=data x,
data y=data y)
print(alphas) #[0.  0.  0.2  0.142  0.  0.2  0.142  0.  0.]
print(b)# 2.66
w = compute_w(data_x,data_y,alphas)
```

[①] Machine Learning, Stanford University, CS229, Autumn 2009.

```
print(w)# [-0.186,  -0.569]
```

根据上述输出结果可知，$\alpha_3=\alpha_6=0.2$，$\alpha_4=\alpha_7=0.142$，即对应的支持向量为$(1,5),(3,5),(3,2),(1.5,6)$，且同时$w=(-0.186,-0.569),b=2.66$。需要注意的是，为了作图方便，图 9-13 中左右两边各自还有两个样本点没有画出，所以上述代码中有 9 个样本。

9.8.6 小结

在本节中，笔者首先以坐标上升算法为铺垫，介绍了 SMO 算法的基本思想，从整体层面上阐述了 SMO 算法的求解过程；然后详细介绍 SMO 算法的具体求解过程及原理，包括参数α和偏置b的求解方法；最后以伪代码的形式展示了 SMO 算法的求解过程，并通过一个实例进行了展示。

9.9 从零实现支持向量机

经过前面几节内容的介绍我们现在已经清楚了 SVM 的基本原理，并且根据第 9.8.5 节内容的讲解我们对于 SVM 的求解也有了一定的认识，对于 SVM 整个内容的介绍就只差最后一步编码实现了。下面，笔者将根据前面介绍的各个求解公式来一步一步向各位读者展示如何实现一个简单 SVM 分类器。

9.9.1 常见核函数实现

根据第 9.3.6 节内容介绍可知，常见的核函数有线性核函数、多项式核函数、高斯核函数和 Sigmoid 核函数等，这里笔者以典型的线性核与高斯核函数为例进行实现，剩余的几种各位读者可以自行实现并验证。

1) 线性核实现

线性核的实现比较简单，根据式(9-24)大和(9-32)可知实现如下：

```
def kernel_linear(X, x):
    return np.dot(X, x)
```

在上述代码中，第 1 行中 X 为支持向量形状为[m, n]或者[n,]，x 为待预测样本形状为[n,]；第 2 行是返回线性组合后的结果形状为[n,]。

2) 高斯核实现

高斯核是将低维特征空间映射到无穷维的特征空间，根据式(9-24)和(9-30)可知实现如下：

```
def kernel_rbf(X, x):
    if X.ndim > 1:
        sigma = 1 / X.shape[1]
    else:
        sigma = 1 / X.shape[0]
    k = -0.5 * np.sum((X - x) ** 2, axis=-1) / sigma ** 2
    return np.exp(k)
```

在上述代码中，第 2-5 行用来计算高斯核函数里的参数σ（参考的是 sklearn 中的做法）；第 6-7 行是返回经高斯核函数变换后的结果。

9.9.2 SMO 求解过程实现

在介绍完核函数的实现部分后再来 SMO 算法的求解实现过程。首先，需要根据第 9.8.3 和第 9.8.4 小节中的内容来实现相关辅助函数。

根据式(9-24)可知，预测函数$f(x)$的编码实现为：

```
def f_x(X, y, alphas, x, b, kernel):
    k = kernel(X, x)
    r = alphas * y * k
    return np.sum(r) + b
```

在上述代码中，第 1 行 X 和 y 为训练集，alphas 为求解得到的参数，x 为预测样本，b 为偏置，kernel 为核函数。

进一步，根据式(9-147)和(9-149)可知，η 和 E_i 以及 α_2^{new} 的编码实现为：

```
def compute_eta(x_1, x_2, kernel):
    return kernel(x_1, x_1) - 2 * kernel(x_1, x_2) + kernel(x_2, x_2)

def compute_E_i(f_x_i, y_i):
    return f_x_i - y_i

def compute_alpha_2(alpha_2, E_1, E_2, y_2, eta):
    return alpha_2 + (y_2 * (E_1 - E_2) / eta)
```

同时，根据式(9-151)和(9-152)可知，计算 L 和 H 的编码实现如下：

```
def compute L H(C, alpha 1, alpha 2, y 1, y 2):
    L = np.max((0., alpha 2 - alpha 1))
    H = np.min((C, C + alpha 2 - alpha 1))
    if y 1 == y 2:
        L = np.max((0., alpha 1 + alpha 2 - C))
        H = np.min((C, alpha_1 + alpha_2))
    return L, H
```

此时根据式(9-153)和(9-154)便可以编码实现 α 的计算，如下：

```
def clip alpha 2(alpha 2, H, L):
    if alpha 2 > H:
        return H
    if alpha_2 < L:
        return L
    return alpha_2

def compute_alpha_1(alpha_1, y_1, y_2, alpha_2, alpha_old_2):
    return alpha_1 + y_1 * y_2 * (alpha_old_2 - alpha_2)
```

最后，根据式(9-160)到(9-162)可以编码实现 b 的计算，如下：

```
def compute_b1(b, E_1, y_1, alpha_1, alpha_old_1,
               x_1, y_2, alpha_2, alpha_2_old, x_2, kernel):
    p1 = b - E_1 - y_1 * (alpha_1 - alpha_old_1) * kernel(x_1, x_1)
    p2 = y_2 * (alpha_2 - alpha_2_old) * kernel(x_1, x_2)
    return p1 - p2

def compute_b2(b, E_2, y_1, alpha_1, alpha_old_1,
               x_1, x_2, y_2, alpha_2, alpha_2_old, kernel):
    p1 = b - E_2 - y_1 * (alpha_1 - alpha_old_1) * kernel(x_1, x_2)
    p2 = y_2 * (alpha_2 - alpha_2_old) * kernel(x_2, x_2)
    return p1 - p2

def clip_b(alpha_1, alpha_2, b1, b2, C):
    if alpha_1 > 0 and alpha_1 < C:
        return b1
    if alpha_2 > 0 and alpha_2 < C:
        return b2
    return (b1 + b2) / 2
```

在实现上述相关计算函数后，便可以第 9.8.5 节中的伪代码来实现 SVM 参数的求解过程。由于这部分代码较长，所以这里笔者分块进行介绍。

```
def smo(C, tol, max_passes, data_x, data_y, kernel):
    m, n = data_x.shape
    b, passes = 0., 0
    alphas = np.zeros(shape=(m))
    alphas_old = np.zeros(shape=(m))
    while passes < max_passes:
        num_changed_alphas = 0
        for i in range(m):
            x_i, y_i, alpha_i = data_x[i], data_y[i], alphas[i]
            f_x_i = f_x(data_x, data_y, alphas, x_i, b, kernel)
            E_i = compute_E_i(f_x_i, y_i)
            if ((y_i*E_i<-tol and alpha_i<C)or(y_i*E_i>tol and alpha_i>0)):
                j = select_j(i, m)
                x_j, y_j, alpha_j = data_x[j], data_y[j], alphas[j]
                f_x_j = f_x(data_x, data_y, alphas, x_j, b, kernel)
                E_j = compute_E_i(f_x_j, y_j)
                alphas_old[i], alphas_old[j] = alpha_i, alpha_j
                L, H = compute_L_H(C, alpha_i, alpha_j, y_i, y_j)
```

在上述代码中，第 2-5 行用来初始化需要进行求解的相关参数；第 8 行开始计算训练集中每个样本点对应的参数α_i；第 9-11 行是依次去每个样本点，然后计算其预测值及预测值与真实值之间的误差；第 13-16 行是取另一个待优化的参数α_j以及其对应的样本点，并计算真实值与预测值之间的误差；第 18 行是计算得到当前α_i,α_i对应的约束范围。

```
                if L == H:
                    continue
                eta = compute_eta(x_i, x_j, kernel)
                if eta <= 0:
                    continue
                alpha j = compute alpha 2(alpha j, E i, E j, y j, eta)
                alpha j = clip alpha 2(alpha j, H, L)
                alphas[j] = alpha j
                if np.abs(alpha j - alphas old[j]) < 10e-5:
                    continue
                alpha_i  =  compute_alpha_1(alpha_i,  y_i,  y_j,  alpha_j,
                                            alphas_old[j])
                alphas[i] = alpha_i
                b1 = compute_b1(b, E_i, y_i, alpha_i, alphas_old[i],
                             x_i, y_i, alpha_j, alphas_old[j], x_j, kernel)
                b2 = compute_b2(b, E_j, y_i, alpha_i, alphas_old[i],
                             x_i, x_j, y_j, alpha_j, alphas_old[j], kernel)
                b = clip_b(alpha_i, alpha_j, b1, b2, C)
                num_changed_alphas += 1
        if num_changed_alphas == 0:
            passes += 1
        else:
            passes = 0
    return alphas, b
```

在上述代码中，第 3 行是根据当前这一组样本点来计算得到η；第 6-12 行是计算得到当前这一组样本点对应的参数α_i和α_j，并更新到参数列表中；第 13-17 是根据α_i,α_j计算得到偏置b；第 19-22 行是用来统计α_i不再发生变化时的次数；第 23 行是返回最终计算得到的参数和偏置。

到此，对于利用 SMO 算法来求解 SVM 中参数α 的代码就介绍完了。

9.9.3 SVM 二分类代码实现

在完成 SMO 算法的求解编码过程后，便可以来编码实现一个基础的 SVM 二分类器。首先，定义一个类并完成对应的初始化方法，代码如下：

```
class SVM(object):
    def __init__(self, C, tol, kernel='rbf', max_passes=20):
        self.C = C
        self.tol = tol
        self.max passes = max passes
        self.alphas = []
        self.bias = []
        if kernel == 'rbf':
            self.kernel = kernel_rbf
        elif kernel == 'linear':
            self.kernel = kernel_linear
        else:
            raise ValueError(f"核函数{kernel}未实现")
```

在上述代码中，第 3-5 行是上面 smo 函数中对应的参数，这里就不再赘述；第 6-7 行用来保存每个二分类器计算得到的α 参数，因为在多分类问题中采用的是 ovr 策略，同时 bias 用来保存每个分类器对应的偏置；第 8-13 行则是取对应的核函数。

进一步，实现二分类器的拟合过程，代码如下：

```
    def  fit binary(self, X, y):
        alphas, bias = smo(C=self.C, tol=self.tol,
                           max passes=self.max passes,
                           data x=X, data y=y, kernel=self.kernel)
        self.alphas.append(alphas)
        self.bias.append(bias)
```

在上述代码中，第 1 行 X 是原始的训练特征，y 是每个样本对应的标签值（只含有 -1 和 +1）；第 2-4 行是根据 SMO 算法来求解得到当前二分类器对应的参数；第 5-6 行是分别将当前二分类器对应的参数放入到列表中。

在拟合得到分类器对应的参数后，便可以用其对新样本进行预测。这里需要先实现一个方法来完成对单个样本的预测，代码如下：

```
    def predict_one_sample(self, x, y, alphas, bias):
        y_score = np.sum(y * self.kernel(self._X, x) * alphas) + bias
        return y_score
```

进一步，实现对多个样本点的预测，代码如下：

```
    def _predict_binary(self, X, y, alphas, bias):
        y_scores = []
        for x in X:
            y_scores.append(self.predict_one_sample(x, y, alphas, bias))
        return y_scores
```

9.9.4 SVM 多分类代码实现

在完成 SVM 二分类的代码实现后，下一步便可以通过 ovr 的策略（这部分内容可以参见第 3.2 节）来实现多分类支持向量机。在二分类的基础上，定义一个类方法来完成多个二分类器的拟合过程，代码如下：

```
    def fit(self, X, y):
        self._X = X
        labelbin = LabelBinarizer(neg_label=-1)
        Y = labelbin.fit_transform(y)
        self.classes_ = labelbin.classes_
        if Y.shape[1] == 1:  #
            Y = np.concatenate((-1 * Y, Y), axis=1)
        self.n_classes = Y.shape[1]
        self.Y = Y
        for c in range(self.n_classes):
            self. fit_binary(self. X, Y[:, c])
        self.alphas = np.vstack((self.alphas))
        self.bias = np.array(self.bias)
```

在上述代码中，第 3-4 行用于将原始的类别标签转化为 one-hot 形式的标签值，且正样本用 1 表示负样本用-1 表示，最终将会得到一个形状为[m,c]的标签矩阵（m 表示样本数量，c 表示类别数量）；第 5 行代码表示得到原始的分类标签。

例如对于标签 y=[0,0,1,1,2]来说，其转换后的结果为：

```
y = [0, 0, 1, 1, 2]
labelbin = LabelBinarizer(neg_label=-1)
Y = labelbin.fit_transform(y)
print(Y)
[[ 1 -1 -1]
 [ 1 -1 -1]
 [-1  1 -1]
 [-1  1 -1]
 [-1 -1  1]]
print(labelbin.classes_)
[0 1 2]
```

同时，第 6-7 行判断当数据集为二分类时 fit_transform 处理后的结果并不是 one-hot 形式，因此需要得到上述类似的标签矩阵；第 10-11 是分别为每个类别拟合得到一个 SVM 二分类器；第 12-13 行是保存得到每个分类器对应的参数。

在完成多分类 SVM 的拟合过程后，便可以借助上面实现的二分类预测方法来完成多分类样本的预测过程，实现代码如下：

```
    def predict(self, X, return_prob=False):
        all_y_scores = []
        for c in range(self.n_classes):
            y_scores = self._predict_binary(X, self.Y[:, c],
                                            self.alphas[c], self.bias[c])
            all_y_scores.append(y_scores)
        all_y_scores = np.vstack((all_y_scores)).transpose()
        prob = np.exp(all_y_scores) /
                            np.sum(np.exp(all_y_scores), 1, keepdims=True)
        y_pred = np.argmax(prob, axis=-1)
        if return_prob:
            return y_pred, prob
        return y_pred
```

在上述代码中，第 3-5 行是使用每个二分类器来对新输入的所有样本进行预测；第 6-8 行则是根据返回得到的预测值来计算得到每个样本最终对应的类别标签。这里需要注意的是，由于 ovr 策略是为每一个类别都构建一个二分类器，因此在根据式(9-24)来判断样本类别时不再是通过计算结果是否大于 0 来决定，而是通过每个分类器对应计算得到的最大值来决定。

在完成整个 SVM 分类器的实现后，便可以通过如下方式进行使用：

```
def load_data():
    x, y = load_iris(return_X_y=True)
    x_train, x_test, y_train, y_test = train_test_split(x, y,
                          test_size=0.3, random_state=2022)
    ss = StandardScaler()
    x_train = ss.fit_transform(x_train)
    x_test = ss.transform(x_test)
    return x_train, x_test, y_train, y_test

def test_iris_classification():
    x_train, x_test, y_train, y_test = load_data()
    model = SVM(C=1., tol=0.001, max_passes=20, kernel='rbf')
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    print("手动实现准确率: ", accuracy_score(y_pred, y_test))

    model = SVC(C=1, kernel='rbf')
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    print("sklean 上准确率: ", accuracy_score(y_pred, y_test))

if __name__ == '__main__':
    test_iris_classification()
```

在上述代码运行结束后，便会得到如下所示的输出结果：

```
手动实现准确率:  0.978
sklean 上准确率:  0.978
```

到此，对于整个支持向量机从零实现部分的内容就介绍完了。

9.9.5 小结

在本节中，笔者首先介绍了两种常用核函数的实现方法；然后介绍了 SMO 算法中各个求解公式的实现过程以及其自身的编码实现过程；接着介绍了如何从零实现原始的二分类 SVM 分类器以及对新样本的预测过程；最后，基于 ovr 的思想实现了多分类的 SVM 分类器，并对整个模型的使用方法进行了示例。

总结一下，在本章中笔者首先介绍了 SVM 的基本思想，即最大化分类间隔；接着详细介绍了 SVM 的原理以及推导过程；进一步又介绍了 SVM 中的线性不可分问题以及核函数的应用；然后介绍了 SVM 中软间隔的由来，拉格朗日乘数法和对偶优化问题；最后，分别介绍了 SVM 中硬间隔和软间隔的优化求解问题，同时还介绍了一种高效的凸二次规划求解方法 SMO 算法，并且借助于 SMO 算法还从零实现了整个 SVM 模型的求解过程。

本次内容就到此结束，感谢您的阅读！如果你觉得上述内容对你有所帮助，欢迎分享至一位你的朋友！若有任何疑问与建议，请添加笔者微信'nulls8'或加群进行交流。**青山不改，绿水长流，我们月来客栈见！**

代码仓库： https://github.com/moon-hotel/MachineLearningWithMe

2021年

掌柜谈如何入门机器学习（必读）！

第一章：机器学习环境安装

Python版本为3.6，各个Python包版本见`requirements.txt`，使用如下命令即可安装：

```
pip install -r requirements.txt
```

第二章：从零认识线性回归

第三章：从零认识逻辑回归

第四章：模型的改善与泛化

第五章：K近邻算法与原理

第六章：朴素贝叶斯算法

第七章：文本特征提取与模型复用

第八章：决策树与集成模型

第九章：支持向量机

第十章：聚类算法

知识点索引