

—武大本本科生课程



第7讲 支持向量机(II)

(Lecture 7 Support Vector Machines: Part 2)

武汉大学计算机学院机器学习课程组

2021.06

第5章 支持向量机

内容目录 (以下红色字体为本讲3学时讲授内容)

5.1 支持向量机概述

5.2 支持向量机分类的思想

5.3 支持向量机理论基础

5.3.1 线性可分支持向量机

5.3.2 线性支持向量机

5.3.3 非线性支持向量机

5.3.4 二分类SVM推广到多分类SVM(*: 选学)

5.4 支持向量机应用及Python编程

小结

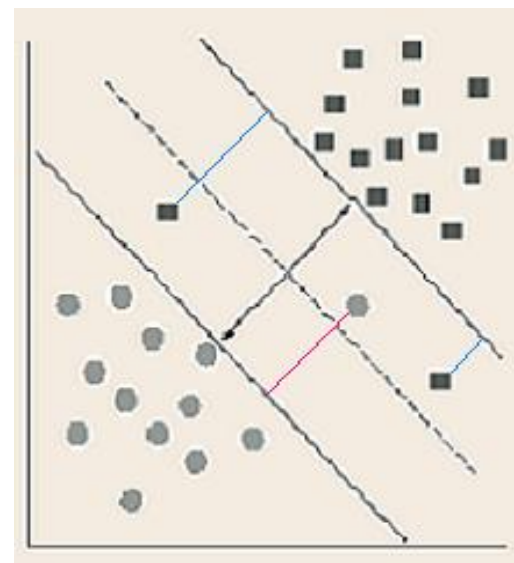
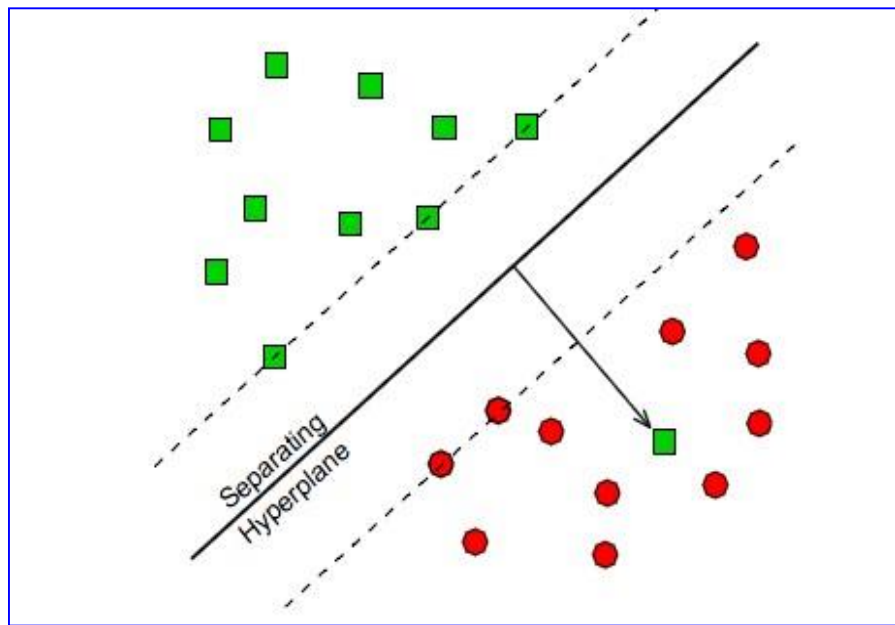
5.3.2 线性支持向量机

- 前面讨论的是线性可分条件下的最优线性分界面的计算方法，对于线性不可分的情况下，如果仍要使用线性分界面，则必然有部分训练样本向量被错分。在线性分界面条件下，被错分的样本从本类别训练样本占主导的决策域，进入了对方的决策域。

1. 约束条件

在线性不可分条件下，严格意义上的间隔已不存在，前面公式也对一些样本不适用。但是仍然可以保留求最大间隔的框架，允许有些样本能进入间隔区，甚至到对方的决策域中。只是对这部分样本的数量要严加控制。为了实行控制，可以采用的一种方法是放宽约束对(5)式作一些改动，增加一种起缓冲作用的量 ξ_i (slack variable: 松弛变量), $\xi_i > 0$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, N \quad (5)$$



此时约束条件变为:

Subject to $y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N$

$$\xi = (\xi_1, \xi_2, \dots, \xi_N)^T \quad (16)$$

即:
$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq +1 - \xi_i, & y_i = +1 (\mathbf{x}_i \text{ 为正例}) \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi_i, & y_i = -1 (\mathbf{x}_i \text{ 为负例}) \end{cases}$$

- Purpose: to allow for a small number of misclassified points for better generalization or computational efficiency.

经验风险最小, 即 $\min \sum \xi_i (i=1, 2, \dots, N)$, 间隔(margin)最大。

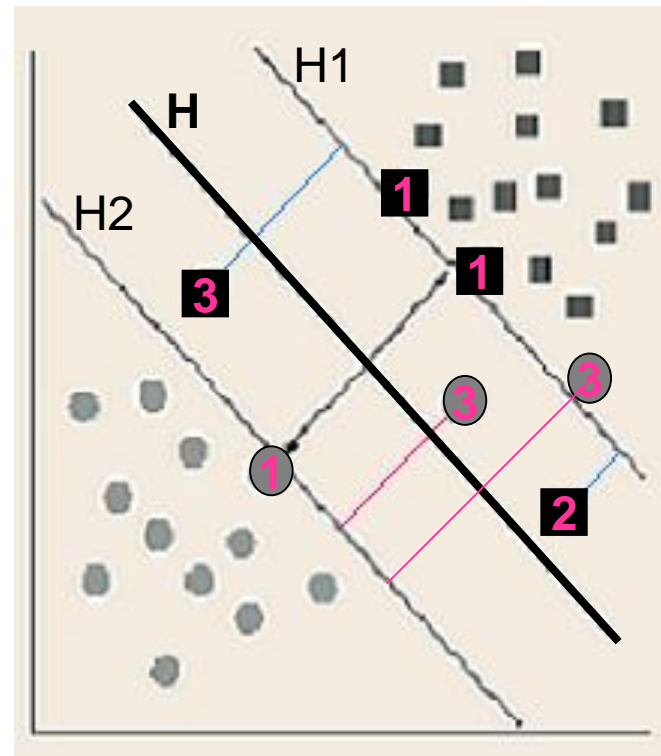
- 下图可帮助我们理解 ξ_i 的含义，具体可分三种情况考虑：

(1)当 $\xi_i=0$ 时，约束条件退化为线性可分时的情况
 $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ ，它对应于分类间隔以外且被正确分类的那些样本，如图中左侧H2线以左(包括H2线上)的所有“圆形”样本点以及右侧H1线(包括H1线上)的所有“方形”样本点。

(2) 当 $0 < \xi_i < 1$ 时, $y_i(\mathbf{w}^T \mathbf{x}_i + b)$ 是一个 0~1 之间的数, 小于 1 意味着我们对约束条件放宽到允许样本落在分类间隔之内, 大于 0 说明样本仍可被分离超平面正确分类, 对应于图中标号为 2 的样本。

(3)最终当 $\xi_i > 1$ 、 $y_i(\mathbf{w}^T \mathbf{x}_i + b) < 0$ 时，此时约束条件已放宽到可以允许有分类错误的样本，如图中标号为3的样本。具体来说，标号为3的位于H1和H2之间(即分类间隔中)的错分样本对应于 $1 < \xi_i < 2$ ，标号为3的位于H2左侧或H1右侧的错分样本对应于 $\xi_i > 2$ 。

图中标号为“1、2、3”的样本均为在线性不可分情况下的支持向量。由于在这种情况下允许样本落入分类间隔之内，常称该分类间隔为**软间隔**(soft margin)。



标号1: 分类间隔支持向量,
对应 $\xi_i=0$

标号2: 非分类间隔支持向量
(分类间隔中), 对应 $0 < \xi_i < 1$

标号3: 非分类间隔支持向量

图 非线性可分情况下的最优分类超平面

2. 目标函数

利用一个附加错误损失系数 C ，称为惩罚因子。目标函数变为：

$$g(\mathbf{w}, b, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (17)$$

$$\text{Minimize } \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{i=1}^N \xi_i \quad (18)$$

Subject to

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N \quad (16)$$

- Role of C :
- As a **regularization**(正则化) parameter (cf. Radial Basis Function, fitting).
 - large $C \implies$ Minimize the number of misclassified points.
 - small $C \implies$ maximize the minimum distance $\frac{1}{\|\mathbf{w}\|}$

$$g(\mathbf{w}, b, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (17)$$

在式(17)中， $\|\mathbf{w}\|^2$ 是目标函数（其实系数可有可无，这里加1/2是因为要求导数的缘故），希望它越小越好，因而损失就必然是一个能使之变大的量(能使它变小就不叫损失了，我们本来就希望目标函数值越小越好)。

衡量损失有两种常用的形式：

- 1) $\sum \xi_i$ (i=1,2,...,N)
- 2) $\sum \xi_i^2$ (i=1,2,...,N)

上面两种形式没有大的区别。若选择第1)种L1正则化形式，得到的分类器称为一阶软间隔分类器，若选择第2)种L2正则化形式，得到的分类器称为二阶软间隔分类器。把损失加入目标函数时，就需要一个惩罚因子C (Cost)。C为是一个可调参数，它表示对错误的惩罚程度，C越大惩罚越重。

对(18)的几点说明：

(1)并非所有的样本点都有一个松弛变量与其对应。实际上只有“离群点”(Outlier)才有，或者也可以这么看，所有没离群的点，松弛变量都等于0 (对负类来说，离群点就是在前面有标号的图中，跑到H2右侧的那些负样本点，对正类来说，就是跑到H1左侧的那些正样本点)。

(2)松弛变量的值实际上标识出了对应的点到底离群有多远，值越大，点就越远。

(3)惩罚因子C决定了重视离群点带来损失的程 度，显然当所有离群点的松弛变量的和一定时，若C取得越大，则对目标函数的损失也越大，此时就暗示着你非常不愿意放弃这些离群点，最极端的情况是你把C取为无限大，这样只要稍有一个点离群，目标函数的值很快变成无限大，马上让问题变成无解，这就退化成了硬间隔问题。

(4) 惩罚因子 C 不是一个变量，整个优化问题在求解的过程中， C 是一个必须事先指定的值，指定该值后，求解得到一个分类器，然后用测试样本查看结果如何。如果结果不够好，就换一个 C 的值，再解一次优化问题，得到另一个分类器，再查看效果如何，如此就是一个参数寻优的过程，但这和优化问题本身绝不是一回事，优化问题在求解的过程中， C 一直是定值，这一点要务必注意。

(5) 尽管加了松弛变量，但该优化问题仍然是一个优化问题。对它求解的过程比起线性可分的硬间隔(hard margin)分类问题来说，没有任何更为特殊的地方。

正因实际处理非线性可分问题中引入了惩罚因子 C ，这种支持向量机常被称为 **C -SVM**。

样本不平衡问题中惩罚因子C的选取方法：

样本的偏斜/不平衡问题，也叫数据集偏斜(**unbalanced**)，它是指参与分类的两个类别(也可以是多个类别)样本数量差异很大。比如，正类有 $N_1=10000$ 个样本，负类只有 $N_2=100$ 个样本，由此引起的问题叫**样本不平衡**问题。

对付数据集偏斜问题的方法之一就是在惩罚因子C上作文章，如给样本数量少的负类更大的惩罚因子 C_- ，表示我们重视这部分样本。因此这里目标函数中因松弛变量而损失的部分就变为：

$$C_+ \sum_{i=1}^{N_1} \xi_i + C_- \sum_{i=1}^{N_2} \xi_i$$

如何确定 C_+ 和 C_- ？一般来说， C_+ 和 C_- 取值是在实验过程试出来的（**参数调优**）。但是它们的比例可以有些方法来确定，如先假设 $C_+=5$ ，那确定 C_- 的一个很直观的方法就是使用两类样本数的比来计算，对应上面样本不平衡例子， C_- 就可取500这一数值(因为 $10000:100=100:1$)。

3. 优化求解

线性支持向量机的对偶学习算法，首先是求解对偶问题得到最优解 α^* ，然后求原始问题最优解 \mathbf{w}^* 和 b^* ，得出分离(类)超平面和分类决策函数。

类似于线性可分情况下的推导过程，最终可以得到相应的对偶问题(Dual problem)。

- **Dual Problem**

- *Maximize* $L(\alpha) = -\frac{1}{2} \alpha^T D \alpha + \sum_{i=1}^N \alpha_i$

- *subject to*

$$\sum_{i=1}^N y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N$$

{ **KT/KKT**(**K**aruch-**K**uhn-**T**ucher) **conditions** }

同样在采用二次规划方法求解出最优的 α 的值

$\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ 后，则可以计算出权向量 \mathbf{w}^* 的值：

$$\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i \quad (13)$$

若存在 α^* 的一个正分量 $0 < \alpha_j^* < C$ ，则可计算出偏置 b^* 的值：

$$\begin{aligned} b^* &= y_j - \mathbf{w}^* \cdot \mathbf{x}_j \\ &= y_j - \sum_{i=1}^N y_i \alpha_i^* (\mathbf{x}_i \cdot \mathbf{x}_j) \end{aligned} \quad (13b)$$

对偶问题的解中满足 $\alpha_i^* > 0$ 的实例点 \mathbf{x}_i 称为支持向量。支持向量可在间隔边界上，也可在间隔边界和分离超平面之间，或者在分离超平面误分一侧。最优分离超平面由支持向量完全决定。

所得到的最优分类函数与式(15)相同:

$$f(\mathbf{x}) = \text{sgn}\{\mathbf{w}^* \cdot \mathbf{x} + b^*\} = \text{sgn}\left\{\sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right\} \quad (15)$$

设支持向量集用符号 SV 表示, 同样由于对非支持向量 \mathbf{x}_k 而言其 $\alpha_k^*=0$, 因此 (15)式还可以表示为:

$$f(\mathbf{x}) = \text{sgn}\left\{\sum_{\mathbf{x}_i \in SV} \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right\} \quad (15a)$$

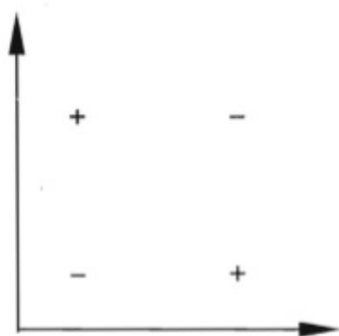
上式只包含待分类样本与训练样本中的支持向量的内积运算, 可见, 要解决一个特征空间中的最优线性分类问题, 我们只需要知道这个空间中的内积运算即可。

对非线性问题, 可以通过非线性变换转化为某个高维空间中的线性问题, 在变换空间求最优分类面。这种变换可能比较复杂, 因此这种思路在一般情况下不易实现。

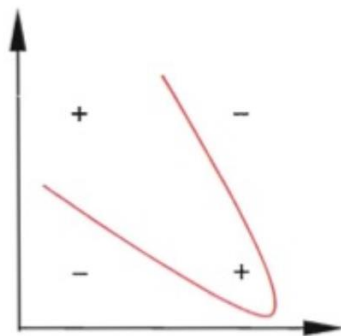
5.3.3 非线性支持向量机

1. 非线性可分二分类模型

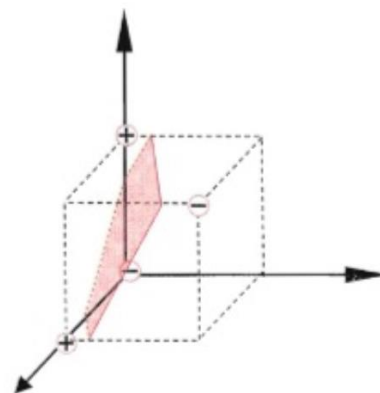
- 线性可分是比较理想的情况，很多任务是**非线性可分**的
- 异或（**XOR**）问题是一个典型的**非线性可分问题**
- 解决异或问题的两种方案
 - 保持**数据集不变**，使用**非线性的划分边界**
 - 对**数据集的特征向量**进行**特征映射**，使其映射到**高维空间**后变成**线性可分**



(a) 数据集



(b) 非线性划分边界

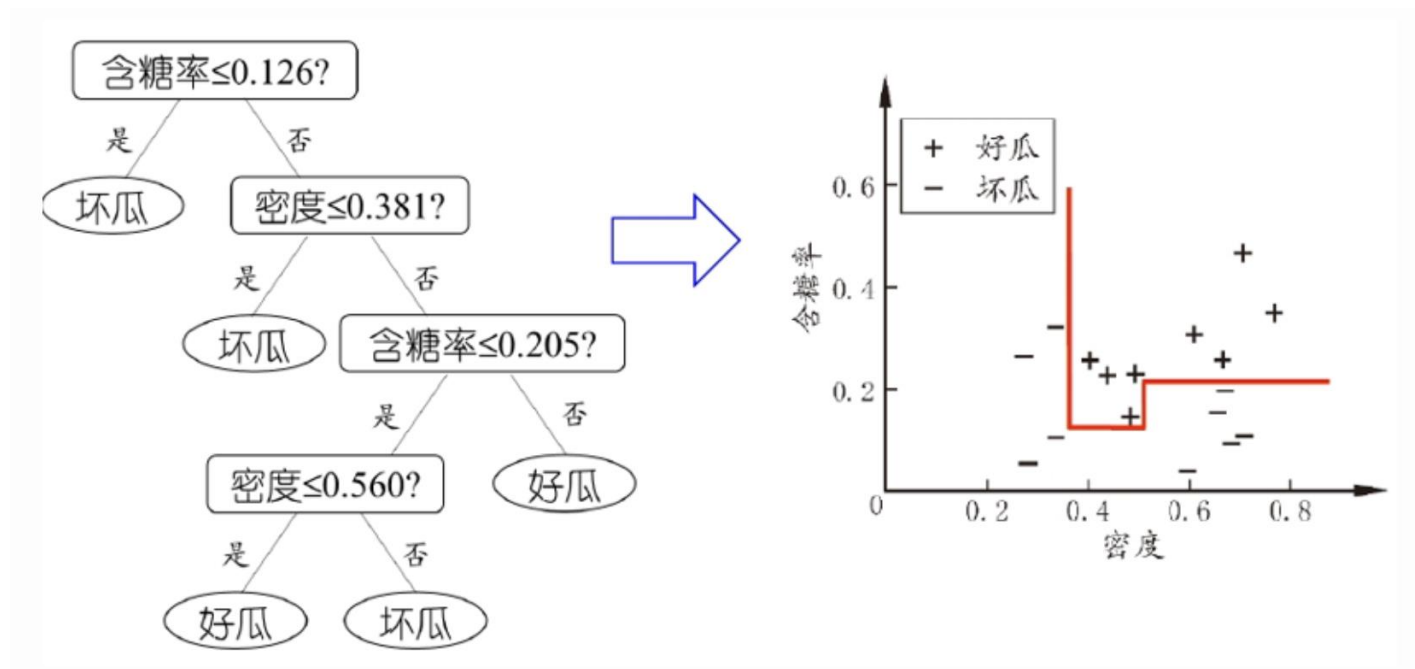


(c) 数据特征映射

5.3.3 非线性支持向量机

1. 非线性可分二分类模型

- 非线性分类模型典型代表是**决策树模型**
- 决策树模型采用**分段函数**组成划分边界



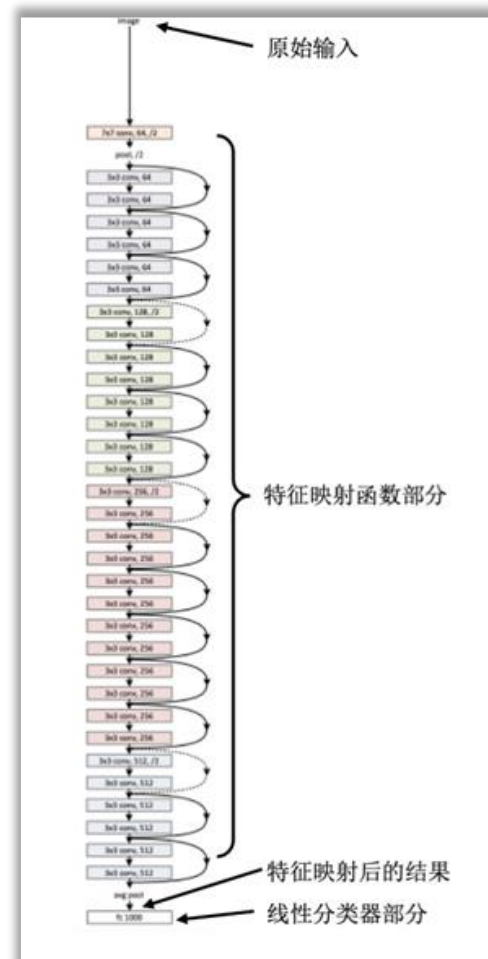
决策树模型本质是用分段线性近似整体的非线性划分边界！

5.3.3 非线性支持向量机

1. 非线性可分二分类模型

➤ 深度学习模型是两种方案的集成

- ❑ 深度学习模型由多层神经网络组成，每层存在**非线性激活函数**，网络深度越深，网络的**非线性**表示能力越强，对应的划分边界的**非线性程度越高**
- ❑ 模型整体包括两个部分：最后一层全连接层看作是**线性分类器**部分，最后一层全连接层之前的所有层看作**特征映射函数**部分
- ❑ 模型同时学习对**原始输入**的映射函数和**线性分类器**的参数，最后只需一个**线性分类器**即可完成分类任务



5.3.3 非线性支持向量机

1. 非线性分类面

线性分类器的分类性能毕竟有限，而对于非线性总是一味放宽约束条件只能导致大量样本的错分，这时可以通过非线性变换将其转化为某个高维空间的线性问题，在变换空间内求得最优分类超平面。

- 非线性可分的数据样本在高维空间有可能转化为线性可分。
- 在训练问题中，涉及到训练样本的数据计算只有两个样本向量点积的形式。
- 使用函数 $\Phi: d \rightarrow n$ ，将所有样本点映射到高维空间 ($d < n$)，则新的样本集为 $(\Phi(\mathbf{x}_1), y_1), \dots, (\Phi(\mathbf{x}_N), y_N)$ 。
- 设函数 $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2)$ 。

2. SVM的基本方程

- 已知： N 个观测样本， $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

- 基本型： $\min_{w, b} \frac{1}{2} w^T w$

$$s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, 2, \dots, N$$

- 对偶型： $\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^N \alpha_i$

$$s.t. \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, N$$

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

3. SVM的特征映射方案

- SVM通过特征映射的方案解决非线性可分的问题
- 使用函数 $\Phi: d \rightarrow n$, 将所有样本点映射到高维空间 ($d < n$), 则新的样本集为 $(\Phi(\mathbf{x}_1), y_1), \dots, (\Phi(\mathbf{x}_N), y_N)$
- 基本型变为:
$$\min_{w, b} \frac{1}{2} w^T w$$
$$s. t. \quad y_i (w^T \phi(x_i) + b) \geq 1, i = 1, 2, \dots, N$$
- 对偶型变为:

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

4. 特征映射后的带来的问题

- 基本型：计算 $\min_{w,b} \frac{1}{2} w^T w$ 和 $y_i(w^T \phi(x_i) + b)$ 需要在 n 维空间中计算 w^T 和 w 的内积，以及 w^T 和 $\phi(x_i)$ 的内积，计算的复杂度为 $O(n)$
- 对偶型：计算 $\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) - \sum_{i=1}^N \alpha_i$ 需要在 n 维空间中计算 $\phi(x_i)^T$ 和 $\phi(x_j)$ 的内积，计算的复杂度为 $O(n)$
- 当特征被映射到非常高维的空间（ n 特别大），计算负担会很大

5. 核技巧

- 核技巧（kernel trick）:对对偶型的计算进行简化
- 原因：在对偶型中，被映射到高维的特征向量总是以内积的形式存在，即 $\phi(x_i)^T \phi(x_j)$
- 如果先计算特征在 n 维空间中的映射 $\phi(x_i)$ 和 $\phi(x_j)$ ，再计算内积 $\phi(x_i)^T \phi(x_j)$ ，计算的复杂度为 $O(n)$
- 核技巧的本质是将特征映射和内积这两步运算合成一步，使得计算复杂度由 $O(n)$ 降为 $O(d)$

6. 核函数

- 核技巧（kernel trick）可以对对偶型的计算进行简化
- 核技巧希望构造一个函数： $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$,
并且 $K(x_i, x_j)$ 的计算复杂度是 $O(d)$
- 对偶型就可以写作如下形式

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \\ & \sum_{i=1}^N \alpha_i y_i = 0 \end{aligned}$$

上式也被称之为硬间隔核化SVM的对偶型

6. 核函数

(1) 核函数的引入

核函数在机器学习算法中进行非线性改进的主要思路如下：

①对于低维空间非线性可分的样例，可将样例特征映射到高维空间中，即“升维”，以此达到高维空间线性分类的目的。其中关键的部分在于找到低维到高维的非线性映射函数(基函数) $\Phi(\mathbf{x})$ ，确定参数和高维特征空间维数等问题，实现低维数据到高维数据的映射。

②非线性可分的样例映射到高维空间后，需要计算高维空间中数据之间的内积，计算量大且计算困难，甚至引起“维数灾难”，这些问题可以应用核函数解决。

③核函数是对低维数据的计算，其计算结果与高维数据的内积运算结果相同，从而不需要再选取映射函数。用核函数代替高维数据的内积，避免了直接在高维空间中进行复杂计算。

(6)核函数定义

定义：核函数 (Kernel function) 是一个对称函数 K ，对所有 $\mathbf{x}, \mathbf{y} \in X$ ，满足

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$$

这里 Φ 是从 d 维输入空间 X 到 n 维特征空间 H 的映射；
 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$ 为向量 \mathbf{x} 、 \mathbf{y} 映射到特征空间上的向量之间的内积。

$$\mathbf{x} = (x_1, \dots, x_d) \text{ a } \Phi(\mathbf{x}) = (\Phi_1(\mathbf{x}), \dots, \Phi_n(\mathbf{x}))$$

将输入空间 X 映射到一个新的空间 $H = \{\Phi(\mathbf{x}) \mid \mathbf{x} \in X\}$

$$\text{例如: } (x_1, x_2) \text{ a } \Phi(x_1, x_2) = (x_1^2, x_2^2, x_1x_2)$$

Mercer定理/Mercer条件：当且仅当对于任意满足 $\int g(\mathbf{x})^2 d\mathbf{x}$ 为有限值的函数 $g(\mathbf{x})$ ，则核函数 $K(\mathbf{x}, \mathbf{y})$ 满足以下条件

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

满足以上条件的核函数 $K(\mathbf{x}, \mathbf{y})$ 称为正定 (positive definite) 核函数。

核函数有严格的数学要求，凡满足Mercer定理的函数都可以作为核函数。Mercer定理确保高维空间任意两个向量的内积一定可以被低维空间中两个向量的某种计算表示(大多数情况下是内积的某种变换)。

(3)核函数特点

核函数方法的广泛应用与其特点是分不开的。核函数主要特点有：

- ①核函数是在原空间进行计算，既避免了“维数灾难”，又减小了计算量，因此，核函数方法可以有效处理高维输入。
- ②无须知道变换函数 $\Phi(\mathbf{x})$ 的形式和参数。
- ③核函数的形式和参数的变化会改变从输入空间到特征空间的映射，进而对特征空间的性质产生影响，最终改变核函数方法的性能。
- ④核函数方法可以和不同的算法相结合，形成多种不同的基于核函数计算的方法，且这两部分的设计可以单独进行，并可为不同的应用选择不同的核函数和算法。

6. 核函数示例

- 核函数: $K(x_i, x_j) = (x_i^T x_j)^2$
- 对应的特征映射
- 此时将特征向量 x 由 d 维映射到 d^2 维
- 直接计算 $K(x_i, x_j)$ 的计算复杂度为 $O(d)$
- 先计算特征在 d^2 维空间中的映射
 $\phi(x_i)$ 和 $\phi(x_j)$, 再计算内积 $\phi(x_i)^T \phi(x_j)$,
计算复杂度将提高到 $O(d^2)$

$$\phi: x \in \mathbb{R}^d \mapsto \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ \vdots \\ x_1 x_d \\ x_2 x_1 \\ x_2 x_2 \\ \vdots \\ x_2 x_d \\ \vdots \\ x_d x_1 \\ x_d x_2 \\ \vdots \\ x_d x_d \end{bmatrix} \in \mathbb{R}^{d^2},$$

6. 核函数示例

- 证明如下：

$$\begin{aligned}\kappa(\mathbf{x}_i, \mathbf{x}_j) &:= (\mathbf{x}_i^\top \mathbf{x}_j)^2 \\ &= (\mathbf{x}_i^\top \mathbf{x}_j)(\mathbf{x}_i^\top \mathbf{x}_j) \\ &= \left(\sum_{u=1}^d x_{iu} x_{ju} \right) \left(\sum_{v=1}^d x_{iv} x_{jv} \right) \\ &= \sum_{u=1}^d x_{iu} x_{ju} \sum_{v=1}^d x_{iv} x_{jv} \\ &= \sum_{u=1}^d \sum_{v=1}^d x_{iu} x_{ju} x_{iv} x_{jv} \\ &= \sum_{u=1}^d \sum_{v=1}^d (x_{iu} x_{iv})(x_{ju} x_{jv}) \\ &= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j).\end{aligned}$$

即 $K(x_i, x_j)$ 与 $\phi(x_i)^\top \phi(x_j)$ 等价

6. 核函数示例

例：设 $f(\mathbf{z})$ 定义为将三维的数据 $[z_1, z_2, z_3]^T$ 映射为九维的数据， $f(\mathbf{z})=[z_1z_1, z_1z_2, z_1z_3, z_2z_1, z_2z_2, z_2z_3, z_3z_1, z_3z_2, z_3z_3]^T$ ，现有两个样本 $\mathbf{x}=[1,2,3]^T$ ， $\mathbf{y}=[4,5,6]^T$ ，计算 $f(\mathbf{x})$ 与 $f(\mathbf{y})$ 的内积。

解：由 $f(\mathbf{z})$ 的定义知：

$$f(\mathbf{x})=[1,2,3,2,4,6,3,6,9]^T$$

$$f(\mathbf{y})=[16,20,24,20,25,30,24,30,36]^T$$

$$f(\mathbf{x}) \cdot f(\mathbf{y})=16+40+72+40+100+180+72+180+324=1024$$

$$K(\mathbf{x}, \mathbf{y})=(\mathbf{x} \cdot \mathbf{y})^2=(4+10+18)^2=1024$$

显然 $f(\mathbf{x}) \cdot f(\mathbf{y})$ 和 $K(\mathbf{x}, \mathbf{y})$ 的计算结果相同，区别在于 $f(\mathbf{x}) \cdot f(\mathbf{y})$ 是先将 \mathbf{x} 和 \mathbf{y} 映射到高维空间中，然后再根据内积的公式进行计算，而 $K(\mathbf{x}, \mathbf{y})$ 直接在原来的低维空间中进行计算，不需要显式求出 $f(\mathbf{x})$ 映射后的结果。两者相比，核函数的计算量要比映射函数小，维数越大，效率区分越大。因此，在算法中遇到求映射后的 $f(\mathbf{x})$ 与 $f(\mathbf{y})$ 的内积 $f(\mathbf{x}) \cdot f(\mathbf{y})$ 时可直接使用 $K(\mathbf{x}, \mathbf{y})$ 代替，不需要显式计算每个 $f(\mathbf{x})$ ，甚至不需要知道 $f(\mathbf{x})$ 是如何定义的。

7. 基于核函数的SVM

根据泛函理论，只要一种核函数 $K(\mathbf{x}_i, \mathbf{x}_j)$ 满足Mercer条件，它就对应某一变换空间中的内积。因此，在最优分类面中采用适当的内积函数 $K(\mathbf{x}_i, \mathbf{x}_j)$ 就可以实现某一非线性变换后的线性分类，而计算复杂度却没有增加。

$$K(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$$

相应的分类函数也变为

$$f(\mathbf{x}) = \text{sign}\left\{\sum_{i=1}^s \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^*\right\}$$

这就是支持向量机。

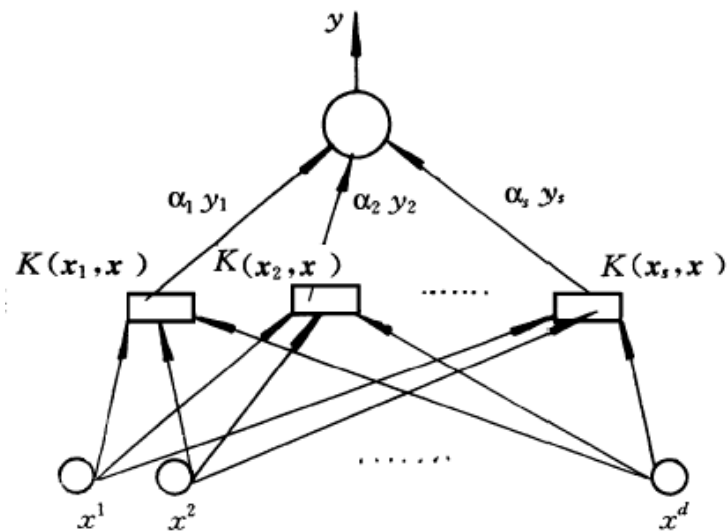
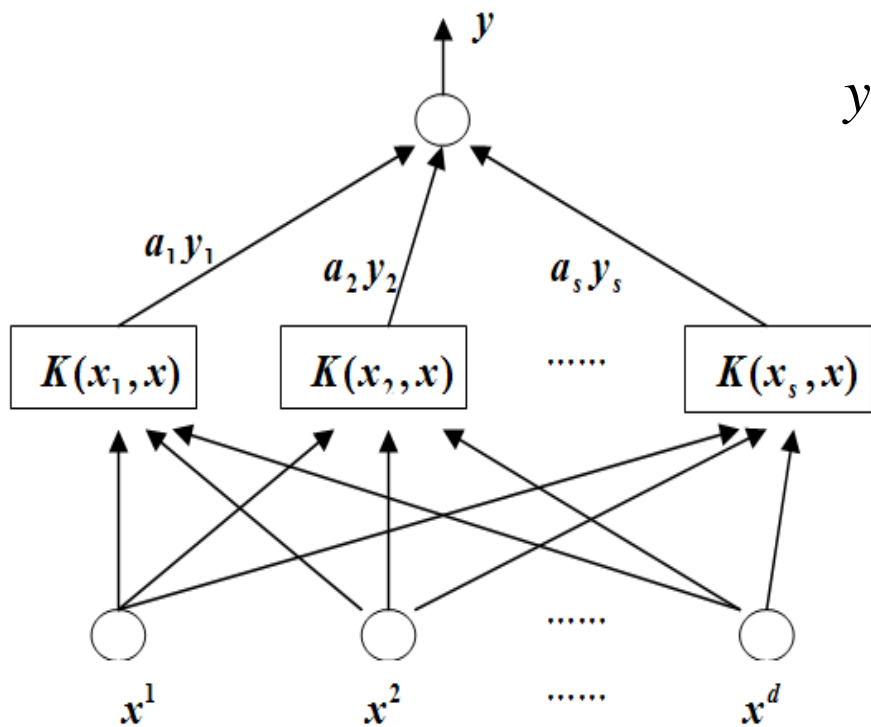


图 支持向量机示意图

概括地说，支持向量机就是首先通过用内积函数定义的非线性变换将输入空间变换到一个高维空间，在这个空间中求最优分类面。

SVM分类函数形式上类似于一个神经网络，输出是中间节点的线性组合，每个中间节点对应一个输入样本与一个支持向量的内积，因此也称为支持向量网络。



支持向量机示意图

$y = \text{sign}(\sum_{i=1}^s w_i K(\mathbf{x}_i, \mathbf{x}) + b)$ 为输出函数.

$w_i = \alpha_i y_i$ 为权值.

$K(\mathbf{x}_1, \mathbf{x}), K(\mathbf{x}_2, \mathbf{x}), \dots, K(\mathbf{x}_s, \mathbf{x})$ 是输入向量 \mathbf{x} 与 s 个支持向量的内积.

$\mathbf{x} = (x^1, x^2, \dots, x^d)$ 为输入向量.

8. 常用核函数

- 线性核: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- 多项式核: $K(\mathbf{x}_i, \mathbf{x}_j) = [\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r]^d$
 $d = 1, 2, \dots$
- 径向基核RBF: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right\}$
- S型核: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r)$
(\tanh 为双曲正切, $\gamma > 0$, $r < 0$)

下面以多项式核(21)式为例进行说明:

$$K(\mathbf{x}_i, \mathbf{x}_j) = [\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r]^d \quad (21)$$

$$d = 1, 2, \dots$$

公式(21)中, 有 γ 、 d 和 r 共3个参数, γ 表示对内积 $\mathbf{x} \cdot \mathbf{y}$ 进行缩放, $\gamma > 0$ 且一般等于 $1/\text{类别数}$ 。 r 代表常数项, 取值范围为 $r \geq 0$; 当 $r > 0$ 时, 称为非齐次多项式; 当 $r = 0$ 时, 称为齐次多项式。 d 为整数, 代表多项式的阶次, 一般设 $d = 2$ 。升维的维度随 d 的增大而指数倍增长, 计算量也随之增大。 d 取值过大, 学习的复杂性也会过高, 易出现过拟合现象。参数 $d = 1$ 、 $\gamma = 1$ 、 $r = 0$ 时, $K(\mathbf{x}_i, \mathbf{x}_j)$ 是线性核函数。

常使用的多项式核函数公式:

$$K(\mathbf{x}_i, \mathbf{x}_j) = [\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + 1]^2 \quad (21a)$$

$$K(\mathbf{x}, \mathbf{y}) = [(\mathbf{x} \cdot \mathbf{y}) + 1]^2 \quad (21b)$$

这里以公式(21b)为例，说明相应的映射函数从低维数据到高维数据的转换。设 \mathbf{x} 、 \mathbf{y} 为 n 维向量， x_i 、 y_i 分别对应向量 \mathbf{x} 和 \mathbf{y} 的第 i 个分量。

$$\begin{aligned} Q \ K(\mathbf{x}, \mathbf{y}) &= [(\mathbf{x} \cdot \mathbf{y}) + 1]^2 = \left(\sum_{i=1}^n x_i y_i + 1 \right)^2 \\ &= \sum_{i=1}^n x_i^2 y_i^2 + \sum_{i=2}^n \sum_{j=1}^{i-1} \left(\sqrt{2} x_i x_j \right) \left(\sqrt{2} y_i y_j \right) + \sum_{i=1}^n \left(\sqrt{2} x_i \right) \left(\sqrt{2} y_i \right) + 1 \end{aligned}$$

$$\therefore \text{设 } \phi(\mathbf{z}) = \left[z_n^2, z_{n-1}^2, \text{L}, z_1^2, \sqrt{2} z_n z_{n-1}, \text{L}, \sqrt{2} z_2 z_1, \sqrt{2} z_n, \sqrt{2} z_{n-1}, \text{L}, \sqrt{2} z_1, 1 \right]^T,$$

可以验证 $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$ 。

5.3.4 二分类SVM推广到多分类SVM(*: 选学)

前面的问题描述中的SVM都是二分类器，只能用于两类样本的分类。下面研究如何将二分类SVM进行推广，使其能处理多类问题。

以4类问题为例，介绍可用于推广SVM解决多类问题的三种常用策略。

1. 一对多的最大响应策略(one against all)

设有A、B、C、D四类样本需要划分。在抽取训练集时，分别按照如下四种方式划分：

- (1) A所对应的样本特征向量作为正集(类标签为+1)，B、C、D所对应的样本特征向量作为负集(类标签为-1)。
- (2) B所对应的样本特征向量作为正集，A、C、D所对应的样本特征向量作为负集。
- (3) C所对应的样本特征向量作为正集，A、B、D所对应的样本特征向量作为负集。
- (4) D所对应的样本特征向量作为正集，A、B、C所对应的样本特征向量作为负集。

对上述4个训练集分别进行训练，得到4个二分类的SVM分类器。在测试时，将未知类别的测试样本 \mathbf{x} 分别送到这4个分类器进行判别，每个分类器都有1个响应，分别为 $f_1(\mathbf{x})$ 、 $f_2(\mathbf{x})$ 、 $f_3(\mathbf{x})$ 、 $f_4(\mathbf{x})$ ，最终的决策结果为 $\text{Max}\{f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}), f_4(\mathbf{x})\}$ ，即这4个响应中的最大者。

注意这时所说的决策函数 $h(\mathbf{x})=\text{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})+b)$ 在符号化之前的输出 $f(\mathbf{x})=\mathbf{w} \cdot \Phi(\mathbf{x})+b$ ， $h(\mathbf{x})$ 表示 \mathbf{x} 位于分割超平面的哪一侧，只反映了 \mathbf{x} 的类别，而 $f(\mathbf{x})$ 还能体现出 \mathbf{x} 与分割平面的远近(绝对值越大越远)，因此它能反映出样本 \mathbf{x} 属于某个类别的置信度。如，同样位于分割超平面正侧的两个样本，显然更加远离超平面的样本是正例的可信度较大，而紧贴着超平面的样本则很有可能是跨过分割平面的一个反例。

2. 一对一的投票策略(one against one with voting)

将A、B、C、D四类样本两类两类地组成训练集，即(A,B)、(A,C)、(A,D)、(B,C)、(B,D)、(C,D)，得到6个(对于n类问题，为 $n(n-2)/2$ 个)二分类器。在测试时，将测试样本x依次送到这6个二分类器，采用投票形式，最后得到一组结果。投票按如下方式进行：

- (1) 初始化： $\text{vote(A)}=\text{vote(B)}=\text{vote(C)}=\text{vote(D)}=0$ 。
- (2) 投票过程：若使用训练集(A,B)得到的分类器将x判定为A类，则 $\text{vote(A)}=\text{vote(A)}+1$ ，否则 $\text{vote(B)}=\text{vote(B)}+1$ ；若使用训练集(A,C)得到的分类器将x判定为A类，则 $\text{vote(A)}=\text{vote(A)}+1$ ，否则 $\text{vote(C)}=\text{vote(C)}+1$ ；.....；若使用训练集(C,D)得到的分类器将x判定为C类，则 $\text{vote(C)}=\text{vote(C)}+1$ ，否则 $\text{vote(D)}=\text{vote(D)}+1$ 。
- (3) 最终判决： $\text{Max}(\text{vote(A)}, \text{vote(B)}, \text{vote(C)}, \text{vote(D)})$ 。若有两个以上的最大值，则一般可简单地取第1个最大值所对应的类别。

3. 一对一的淘汰策略(one against one with eliminating)

这是一种专门针对SVM提出的一种多类推广策略，实际上它也适用于所有可以提供分类器置信度信息的二分类器。该方法同样基于一对一判别策略解决多类问题，即：将A、B、C、D四类样本两类两类地组成训练集，需要6个分类器(A,B)、(A,C)、(A,D)、(B,C)、(B,D)、(C,D)。

显然，对于这4类中的任意一类，如第A类中的某一样本，就可以由(A,B)、(A,C)、(A,D)这3个二分类器中的任意一个来识别，即判别函数间存在冗余。于是将这些二分类器根据其置信度从大到小排序，置信度越大表示此二分类器分类的结果越可靠，反之则越有可能出现误判。

对这6个分类器按其置信度由大到小排序并分别编号，假设为：1# (A,C)，2# (A,B)，3# (A,D)，4# (B,C)，5# (C,D)，6# (B,D)。此时，判别过程如下：

- (1) 设被识别对象为 \mathbf{x} ，首先由1#判别函数进行识别。若判别函数 $h(\mathbf{x})=+1$ ，则结果类型为A，所有关于类型C的判别函数均被淘汰；若判别函数 $h(\mathbf{x})=-1$ ，则结果为类型C，所有关于类型A的判别函数均被淘汰；若判别函数 $h(\mathbf{x})=0$ ，则为“拒绝决策”的情形，则直接选2#判别函数进行识别。假设结果为类型C，则所剩判别函数为4# (B,C)，5# (C,D)，6# (B,D)。
- (2) 被识别对象 \mathbf{x} 再由4#判别函数进行识别。若结果为类型+1，则淘汰所有关于C类的判别函数，则所剩判别函数为6# (B,D)。
- (3) 被识别对象 \mathbf{x} 再由6#判别函数进行识别。若结果为类型+1，则可判别最终的分类结果为B。

那么如何表示置信度呢？对于SVM而言，分割超平面的分类间隔越大，就说明两类样本越容易分开，这表明问题本身有较好的可分性。因此，可用各个SVM二分类器的分类间隔大小作为其置信度。

在上述一对一淘汰策略中，每经一个判别函数就有某一类别被排除，与该类别有关的判别函数也被淘汰。因此，一般经过 $(c-1)$ 次判别就能得到结果。然而，由于判别函数在决策时有可能会遇到“拒绝决策”的情形 $h(\mathbf{x})=0$ ，此时若直接令 $h(\mathbf{x})=+1$ 或 -1 ，而把它归于某一类，则可能导致误差。所以不妨利用判别函数之间所存在的冗余进行再决策，以减少这种因“拒绝决策”而导致误判，一般再经过一到两次决策即可得到最终结果。

上面3种多类问题的推广策略在实际应用中一般都能取得满意的效果，相比之下第2种和第3种推广策略在很多情况下能取得更好的效果。

5.4 支持向量机应用及Python编程

5.4.1 XOR异或问题的SVM分类方法举例及Python编程

例：XOR异或问题的SVM分类方法分析及Python编程。

XOR异或问题

输入向量 \mathbf{x}_i	输出 y_i
$(1,1)^T$	-1
$(1,-1)^T$	+1
$(-1,-1)^T$	-1
$(-1,1)^T$	+1

1. XOR异或问题的SVM分类方法分析

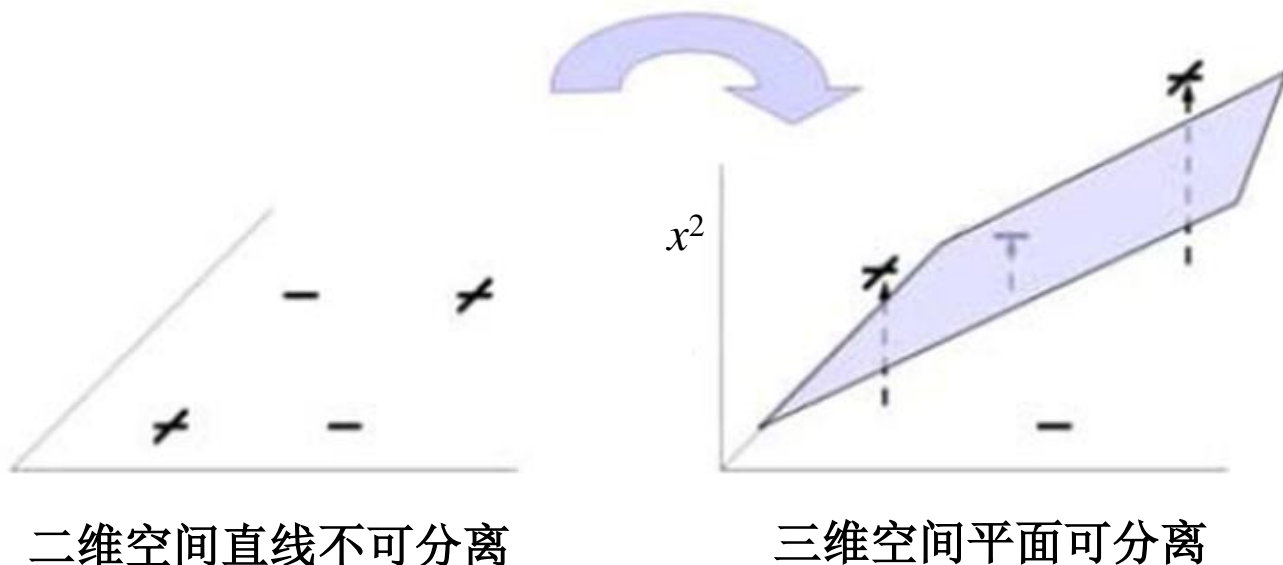


图 XOR的线性不可分

该问题在二维输入空间不是线性可分的，为此可利用核映射方法将其映射到高维空间，然后采用SVM分类法解决。

解题步骤:

- 核函数
- 基函数
- 目标函数
- 优化权值向量 \mathbf{w}
- 最优超平面

(1) 核函数

$$K(\mathbf{x}_i, \mathbf{x}_j) = [1 + (\mathbf{x}_i \cdot \mathbf{x}_j)]^2 \text{ -----采用二次多项式核}$$

$$\mathbf{x}_i = [x_{i1}, x_{i2}]^T \quad \mathbf{x}_j = [x_{j1}, x_{j2}]^T$$

将内积核 $K(\mathbf{x}_i, \mathbf{x}_j)$ 表示为多项式的不同阶数展开式:

$$K(\mathbf{x}_i, \mathbf{x}_j) = 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1} x_{i2} x_{j1} x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1} x_{j1} + 2x_{i2} x_{j2}$$

(2) 基函数

根据核函数的展开式，我们得到基函数，也就是输入向量在高维空间中的映射，在本例中，输入 \mathbf{x}_i 为二维空间，通过基函数映射到六维空间：

$$\varphi(\mathbf{x}_i) = (1, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2})^T$$

根据以上特征函数，分别计算出每个样本映射到六维空间中的向量：

$$\varphi(\mathbf{x}_1) = (1, 1, \sqrt{2}, 1, \sqrt{2}, \sqrt{2})^T$$

$$\varphi(\mathbf{x}_2) = (1, 1, -\sqrt{2}, 1, \sqrt{2}, -\sqrt{2})^T$$

$$\varphi(\mathbf{x}_3) = (1, 1, \sqrt{2}, 1, -\sqrt{2}, -\sqrt{2})^T$$

$$\varphi(\mathbf{x}_4) = (1, 1, -\sqrt{2}, 1, -\sqrt{2}, \sqrt{2})^T$$

(3) 目标函数

$$\begin{aligned} L_D(\boldsymbol{\alpha}) &= -\frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^4 \alpha_i \\ &= -\frac{1}{2} (9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 + 9\alpha_2^2 \\ &\quad + 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 + 9\alpha_3^2 + 2\alpha_3\alpha_4 + 9\alpha_4^2) \\ &\quad + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 \end{aligned}$$

关于拉格朗日乘子 $\{\alpha_i\}$ 最大化目标函数 $L_D(\boldsymbol{\alpha})$ ，得到联立方程组

$$\begin{aligned} 9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 &= 1 \\ -\alpha_1 + 9\alpha_2 + \alpha_3 - \alpha_4 &= 1 \\ -\alpha_1 + \alpha_2 + 9\alpha_3 - \alpha_4 &= 1 \\ \alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4 &= 1 \end{aligned} \quad k = \begin{vmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{vmatrix}$$

(4) 权值向量 \mathbf{w}

求解得到拉格朗日系数 α 的最优值为:

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{8}$$

说明此例中的四个样本都是支持向量

$$\mathbf{w} = \sum_{i=1}^4 \alpha_i y_i \varphi(\mathbf{x}_i)$$

根据以上 \mathbf{w} 的计算公式, 得到最优权值向量:

$$\mathbf{w} = (0, 0, -\frac{\sqrt{2}}{2}, 0, 0, 0)^T$$

(5) 最优超平面

根据最优超平面的定义： $\mathbf{w} \cdot \varphi(\mathbf{x}) = 0$

$$\mathbf{w} = [0, 0, -1/\sqrt{2}, 0, 0, 0]^T$$

$$\varphi(\mathbf{x}) = [1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]^T \quad \mathbf{x} = [x_1, x_2]^T$$

计算得到分类面方程为： $-x_1x_2 = 0$

该分类面在特征空间中是一个超平面。

图形解释

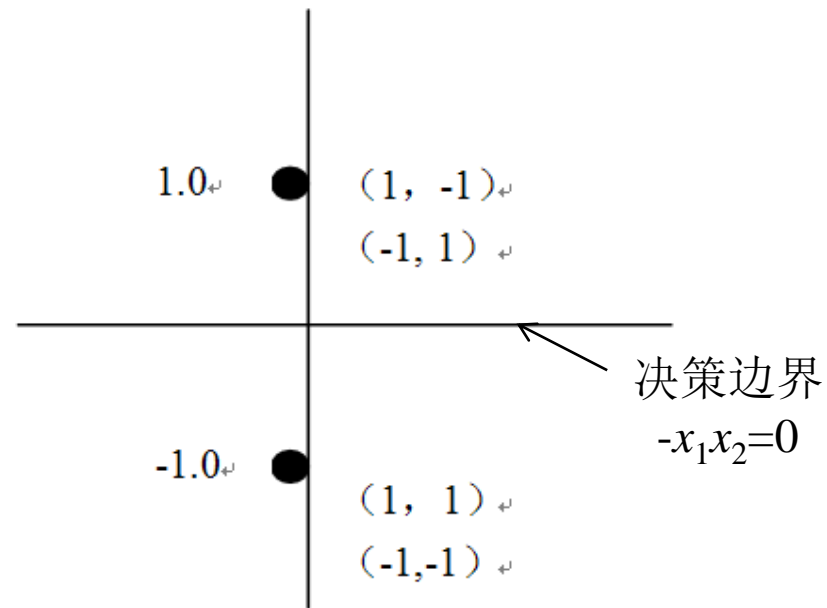


图 XOR异或问题的4个样本推导出的多维空间中的映射

2. XOR问题的SVM分类Python编程

程序清单:

```
#Filename: svm_xor_1.ipynb
```

```
#采用支持向量机实现XOR异或逻辑(Python Sklearn库)
```

```
from numpy import *
```

```
from sklearn.svm import *
```

```
X=array([[ -1, -1], [-1, 1], [1, -1], [-1, -1]]) # -1:假,1:真
```

```
y=array([-1, 1, 1, -1])
```

```
svc=SVC(C=1.0,kernel='poly',degree=2,gamma=1.0)
```

```
#采用2次多项式核函数建立C-SVM(软间隔SVM)支持向量机分类器
```

```
svc.fit(X,y) #训练支持向量机分类器
```

```
predicted1=svc.predict([[1,-1]])
```

```
print(predicted1)
```

```
predicted2=svc.predict([[-1,-1], [-1,1]])
```

```
print(predicted2)
```

运行结果:

```
[1]
```

```
[-1 1]
```

5.4.2 SVM的核函数与参数选择

1. Scikit-learn中的SVM常用核函数

- **linear核**: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- **poly核**: $K(\mathbf{x}_i, \mathbf{x}_j) = [\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r]^d, \gamma > 0, d = 1, 2, \dots$
($d : \text{degree}, r : \text{coef0}$)
- **rbf核**: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$ ($\gamma : \text{gamma}$)
- **sigmoid核**: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + r), \gamma > 0$
($\gamma : \text{gamma}, r : \text{coef0}$)

C-SVC函数调用格式及常见参数说明:

格式: `class sklearn.svm.SVC(kernel='rbf', C=1.0, degree=3, gamma='auto', coef0=0.0, ...)`

常见参数:

- 1) **kernel**: 核函数, string类型, 可选, 默认为rbf, 表示径向基函数; 核函数类型必须是'linear', 'poly', 'rbf', 'sigmoid'等。
- 2) **C**: 目标函数的惩罚因子/正则化参数, 用来平衡分类间隔margin和错分样本, 默认为C = 1.0; 采用固定的L1正则化。
- 3) **degree**: 多项式核函数的最高次幂 d , int类型, 可选, 默认是3, 和其他核函数无关。
- 4) **gamma**: γ , float类型, 可选, 默认是'auto', 它是 'poly'、'rbf'、'sigmoid'的核参数 γ ; 如果是'auto', 则 $\gamma = 1/\text{特征数}$, 即核函数的带宽。

Scikit-learn的SVM中有一个专门的线性支持向量机类LinearSVC，它是用于解决线性分类的线性支持向量机。

使用LinearSVC线性支持向量机进行分类的调用格式及常见参数说明如下：

格式： `class sklearn.svm.LinearSVC(penalty='l2', C=1.0, ...)`

常见参数：

1) **penalty:** L1正则化/L2正则化可选，默认为l2；

2) **C:** 目标函数的惩罚因子/正则化参数，用来平衡分类间隔margin和错分样本，默认为C = 1.0；

(C决定关于Scikit-learn中的SVM目标函数的任务更偏向于优化哪一部分。数值越小，容错空间越宽松；数值越大，越接近于硬间隔SVM。)

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$L_1 = \|\xi\|_1 = \sum_{i=1}^N \xi_i$$

L1正则化(一阶软间隔SVM)

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i^2$$

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$L_2 = \|\xi\|_2^2 = \sum_{i=1}^N \xi_i^2$$

L2正则化(二阶软间隔SVM)

Scikit-learn的SVM中还有一个专门用于回归的支持向量机类SVR，它是用于解决回归问题的支持向量机。

使用**SVR支持向量机**进行回归的调用格式及常见参数说明如下：

格式：`class sklearn.svm.SVR(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, ...,)`

常见参数：

1) **C**: 目标函数的惩罚因子/正则化参数，用来平衡分类间隔margin和错分样本，默认为C = 1.0；采用固定的L1正则化。

2) **kernel**: 核函数，string类型，可选，默认为rbf，表示径向基函数；核函数类型必须是'linear'，'poly'，'rbf'，'sigmoid'等。

3) **degree**: 多项式核函数的最高次幂d，int类型，可选，默认是3，和其他核函数无关。

4) **gamma**: γ ，float类型，可选，默认是'scale'，它是 'poly'、'rbf'、'sigmoid'的核参数 γ ；如果是'scale'，则gamma默认为'scale'，此时，使用 $\text{gamma} = 1 / (\text{n_features} * \text{X.std}())$ ；如果是'auto'，则 $\text{gamma} = 1 / \text{n_features}$ ，即核函数的带宽。

...

2. 不同核函数的SVM对比

案例1：SVM的核函数与参数选择。

这里以Scikit-learn内置的红酒数据集进行对比实验。

(1) 了解红酒数据集

首先，在Jupyter Notebook中输入程序段加载红酒数据集：

```
In [1]: ▶ from sklearn.datasets import load_wine
#从sklearn的datasets模块加载红酒数据集
wine_dataset=load_wine() #load_wine()函数加载的红酒数据集是一种Bunch对象，包括键(keys)和数值(values)
```

接着，打印红酒数据集中的键：

```
In [2]: ▶ print("红酒数据集中的键：\n0".format(wine_dataset.keys()))

红酒数据集中的键：
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

从以上结果可以看出，红酒数据集中包括数据“data”、目标分类“target”、数据描述“DESCR”、目标分类名称“target_names”、特征变量名“feature_names”。

使用.shape打印数据形状：

```
In [3]: ▶ print("数据形状：0".format(wine_dataset['data'].shape))

数据形状：(178, 13)
```

从以上结果可以看出，红酒数据集包括178个样本记录，每个样本有13个特征字段。

打印酒数据集细节描述信息:

```
In [4]: ▶ print(wine_dataset['DESCR'])
```

```
.. _wine_dataset:
```

```
Wine recognition dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 178 (50 in each of three classes)
```

```
:Number of Attributes: 13 numeric, predictive attributes and the class
```

```
:Attribute Information:
```

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

```
- class:
```

- class_0
- class_1
- class_2

```
:Summary Statistics:
```

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

```
:Missing Attribute Values: None
```

```
:Class Distribution: class_0 (59), class_1 (71), class_2 (48)
```

从以上打印结果可以看出，红酒数据集中的178个样本被归到class_0~class_2三个类别中；其中class_0包含59个样本，class_1包含71个样本，class_1包含48个样本；有13个特征(属性)信息，包括酒精含量、苹果酸、镁含量、青花素含量、色彩饱和度等。

(2) 红酒数据集前两维特征数据使用不同核函数的SVM结果对比

为了能直观体验不同核函数的SVM算法在分类中的不同表现，我们借助数据可视化技术，对红酒数据集的前两维特征数据使用不同核函数的SVM结果进行对比，在Jupyter Notebook中输入以下程序代码：

```
#导入numpy
```

```
import numpy as np
```

```
#导入绘图工具
```

```
import matplotlib.pyplot as plt
```

```
#导入支持向量机
```

```
from sklearn import svm
```

```
def make_meshgrid(x, y, h=.02):
```

```
    x_min, x_max = x.min() - 1, x.max() + 1
```

```
    y_min, y_max = y.min() - 1, y.max() + 1
```

```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),  
                          np.arange(y_min, y_max, h))
```

```
    return xx, yy
```

```
def plot_contours(ax, clf, xx, yy, **params):
```

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
    Z = Z.reshape(xx.shape)
```

```
    out = ax.contourf(xx, yy, Z, **params)
```

```
    return out
```

```
#选取红酒数据集的前两个特征
```

```
X = wine_dataset.data[:, :2]
```

```
y = wine_dataset.target
```

C = 1.0 **#SVM正则化参数**

```
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, C=C))
models = (clf.fit(X, y) for clf in models)
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel')
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)
for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.plasma, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.plasma, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Feature 0')
    ax.set_ylabel('Feature 1')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
plt.show()
```

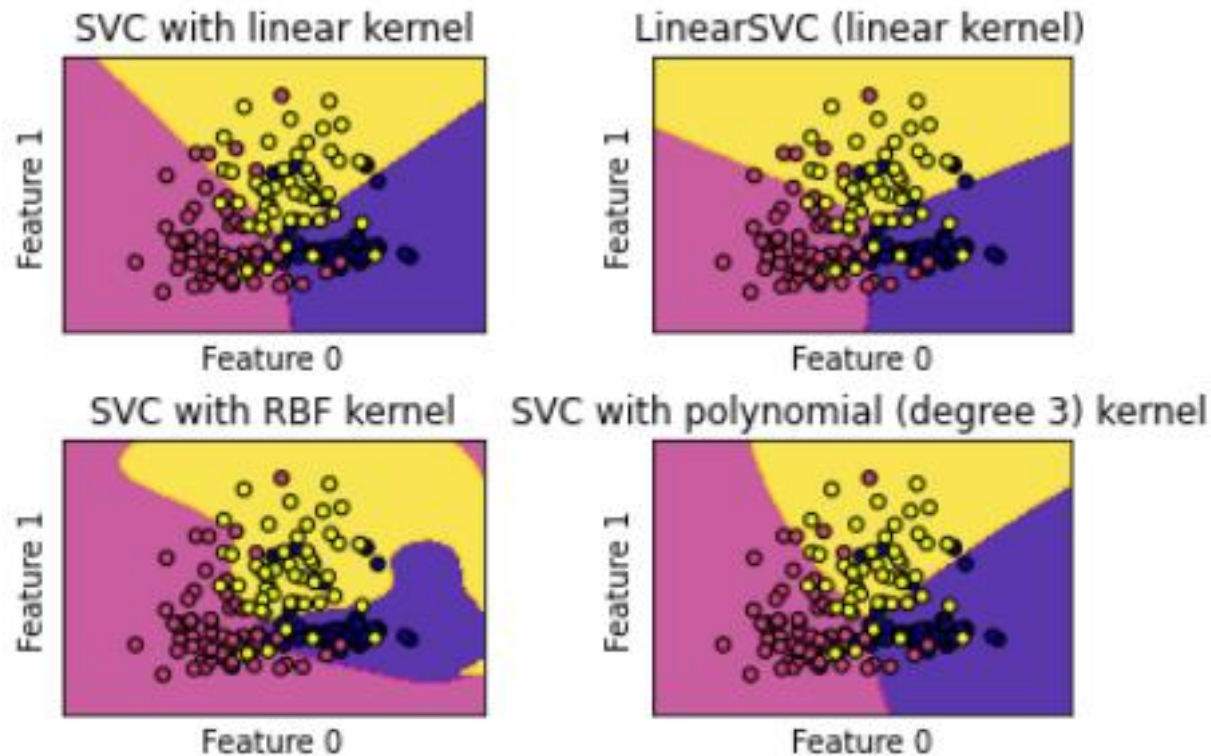


图 采用不同的SVM分类器对红酒数据集进行的分类可视化

从上图可以看出，线性核函数SVC与linearSVC得到的结果非常近似，但仍然有点差别。其中一个原因是linearSVC使用L2正则化进行优化，而线性核函数SVC使用L1正则化进行优化。不管怎样，linearSVC和线性核函数SVC生成的决策边界都是线性的，在更高维数据集中表现为相交的超平面。而RBF核函数和polynomial核函数的SVC分类器则是非线性的，它们更加有弹性；而决定它们边界形状的就是它们的参数。在polynomial核SVC中，起决定性作用的参数是degree和正则化参数C，在本例中degree=3，也就是对原始数据集的特征乘3次方操作。而在RBF核SVC中，起决定性作用的参数是正则化参数C和参数gamma，接下来重点介绍RBF核SVC的gamma参数调整。

(3) SVM的gamma参数调整

首先观察不同的gamma值对于RBF核SVC分类器的影响，在Jupyter Notebook中输入以下程序代码：

```
C = 1.0 # SVM正则化参数
models = (svm.SVC(kernel='rbf', gamma=0.1, C=C),
          svm.SVC(kernel='rbf', gamma=1, C=C),
          svm.SVC(kernel='rbf', gamma=10, C=C))
models = (clf.fit(X, y) for clf in models)
titles = ('gamma = 0.1',
          'gamma = 1',
          'gamma = 10',
          )
fig, sub = plt.subplots(1, 3, figsize = (10,3))
#plt.subplots_adjust(wspace=0.8, hspace=0.2)
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)
for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.plasma, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.plasma, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Feature 0')
    ax.set_ylabel('Feature 1')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
plt.show()
```

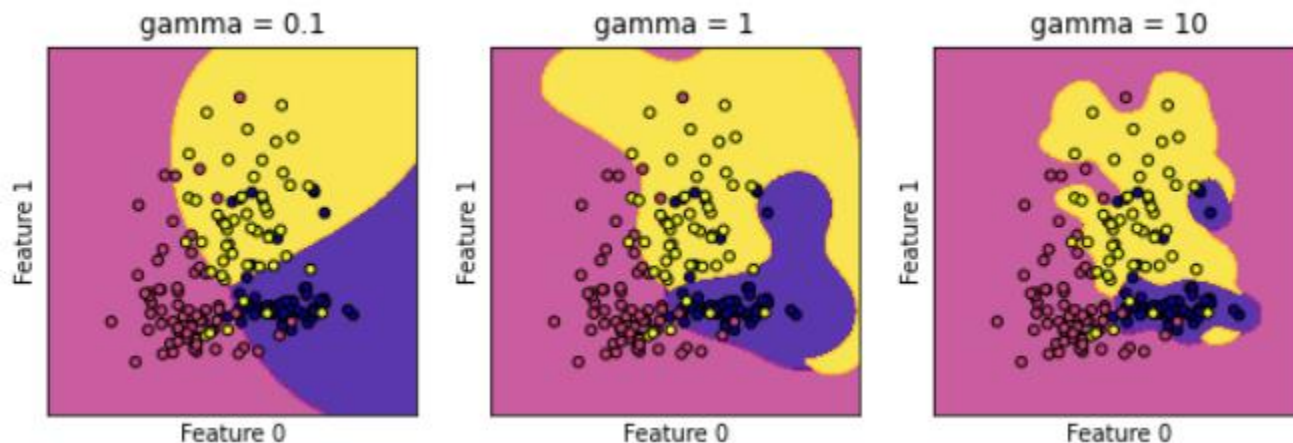


图 选取不同的gamma值对应的RBF核SVC分类器

从上图可以看出，从左到右gamma参数值分别取0.1、1和10。gamma参数值越小，则RBF核直径越大，这样就会有更多的点被模型圈进决策边界中，所以决策边界就越平滑，这时的模型就越简单；随着gamma参数值增大，模型则更倾向于把每个点都放到相应的决策边界中，这时的模型复杂度就相应得到提高。因此，gamma参数值越小，模型越倾向于欠拟合，反之，gamma参数值越大，模型越倾向于过拟合。

至于正则化参数C，C参数值越小，模型就越受限，就是说单个数据点对模型的影响越小，模型就越简单；C参数值越大，每个数据点对模型的影响就越大，模型就会越复杂。

在实际应用中，常常要先决定是使用L1正则化的模型还是L2正则化的模型。大体原则是，如果数据集有很多特征，而这些特征中并不是每一个都对结果有重要影响，那么就优先考虑使用L1正则化的模型；如果数据集特征本来就不多，并且每一个特征都有重要作用，则优先考虑使用L2正则化的模型。

(4) SVM算法的优势与不足

SVM算法可以在数据特征很少的情况下生成非常复杂的决策边界，当然在特征数量较多的情况下也有不错的表现。换句话说，SVM算法应对高维数据集和低维数据集都有不错的表现。还有，数据集中样本和特征的测度都比较接近时，SVM也能游刃有余。

SVM发挥良好性能表现的前提是数据集规模不能太大。如果数据集的样本数量在1万以内，SVM能够驾驭；但如果样本数量超过10万，使用SVM就会非常耗费时间和内存。SVM还有一个短板就是对数据预处理和调参要求极高，因此现在很多应用场景，人们更乐意采用随机森林算法或梯度上升决策树(GBDT, Gradient Boosting Decision Tree)算法。因为它们不需要对数据进行预处理。也不用费尽心机去调参。

提醒大家留意的是，SVM算法有3个参数比较重要：第一个是核函数选择；第二个是核函数的参数，如RBF核的 γ 值；第三个是正则化参数C。RBF核的 γ 值用来调节核宽度， γ 和C值一起控制模型的复杂度，数值越大模型越复杂；反之，数据越小模型越简单。实际应用中， γ 和C值往往要一起调节才能达到最好的效果。

5.4.3 SVM回归分析应用实例

案例2: SVM回归分析应用-波士顿房价回归分析

一、理解波士顿房价数据集

1. 加载波士顿房价数据集

首先在Jupyter Notebook中新建一个笔记本文件，输入以下程序代码加载Boston房价数据集：

```
In [1]: ▶ #Filename: ch6_boston_svr.ipynb
import matplotlib.pyplot as plt
#导入绘图工具
from sklearn.datasets import load_boston
#从sklearn的datasets模块导入波士顿房价数据集
boston = load_boston()
#load_boston()函数加载的波士顿房价数据集是一种Bunch对象，包括键(keys)和数值(values)
```

2. 打印波士顿房价数据集中的键

```
In [2]: ▶ print(boston.keys())

dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

从以上结果可以看出，波士顿房价数据集中包含数据"data"、目标"target"、特征名称"feature_names"、数据短描述"DESCR",但无目标分类名称"target_names"。

3. 打印波士顿房价数据集细节描述信息

```
In [3]: ▶ print(boston['DESCR'])

**Data Set Characteristics:**

: Number of Instances: 506

: Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

: Attribute Information (in order):
- CRIM      per capita crime rate by town
- ZN        proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS     proportion of non-retail business acres per town
- CHAS      Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX       nitric oxides concentration (parts per 10 million)
- RM        average number of rooms per dwelling
- AGE       proportion of owner-occupied units built prior to 1940
- DIS       weighted distances to five Boston employment centres
- RAD       index of accessibility to radial highways
- TAX       full-value property-tax rate per $10,000
- PTRATIO   pupil-teacher ratio by town
- B         1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT     % lower status of the population
- MEDV      Median value of owner-occupied homes in $1000's
```

从以上描述可以看出，数据集共有506个样本，每个样本有13个特征变量。还有一个叫作中位数的第14个变量，该变量是该数据集中的target，它是业主自住房屋屋价格的中位数，以千美元为单位。接下来，我们通过SVR算法建立房价预测模型。

二、使用SVR建模

1. 准备训练集和测试集

```
In [4]: ▶ #导入数据集拆分工具
from sklearn.model_selection import train_test_split
#建立训练集和测试集
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=8)
print(X_train.shape)
print(X_test.shape)

(379, 13)
(127, 13)
```

2. 使用linear和rbf两种不同的核函数进行SVR建模

```
In [5]: ▶ #导入支持向量机回归模型
from sklearn.svm import SVR
#分别测试linear核函数和rbf核函数
for kernel in ['linear', 'rbf']:
    svr = SVR(kernel=kernel)
    svr.fit(X_train, y_train)
    print(kernel, '核函数的模型训练集得分: {:.3f}'.format(
        svr.score(X_train, y_train)))
    print(kernel, '核函数的模型测试集得分: {:.3f}'.format(
        svr.score(X_test, y_test)))

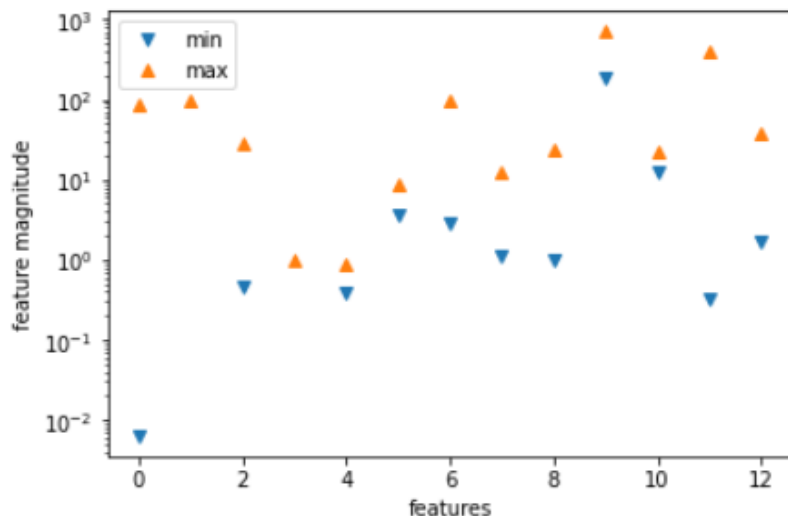
linear 核函数的模型训练集得分: 0.709
linear 核函数的模型测试集得分: 0.696
rbf 核函数的模型训练集得分: 0.192
rbf 核函数的模型测试集得分: 0.222
```

从以上结果可以看出，使用linear核函数的模型训练得分和测试得分都不能令人满意，而使用rbf核函数的模型更糟。

我们可以思考一下原因，会不会是数据集的各个特征之间的量级差得较远？在前面5.4.2中曾提到过SVM的一个短板是对数据预处理和调参要求极高，如果数据特征量级差异较大，就需要对数据进行预处理。下面我们利用数据可视化技术观察一下数据集中各个特征数量级的情况。

3. 特征数值中的最小值和最大值数据可视化

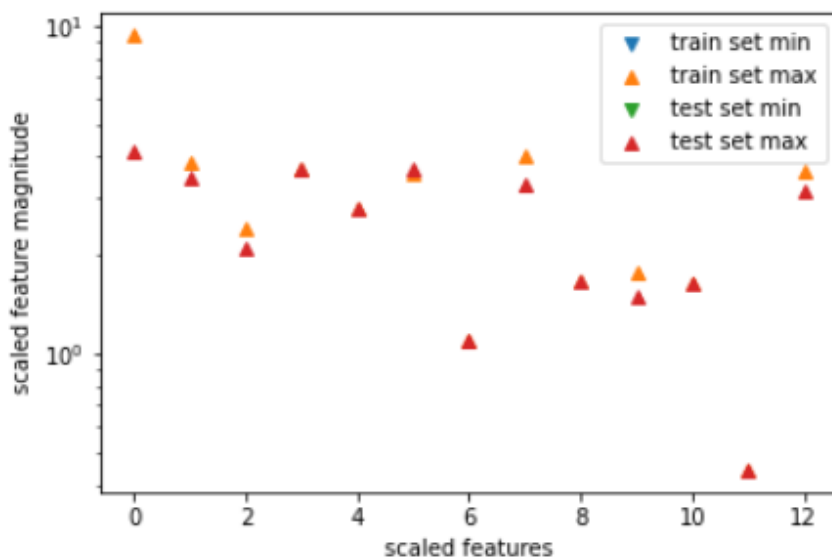
```
In [6]: #将特征数值中的最小值和最大值用散点画出来
plt.plot(X.min(axis=0), 'v', label='min')
plt.plot(X.max(axis=0), '^', label='max')
#设定纵坐标为对数形式
plt.yscale('log')
#设定图注位置为最佳
plt.legend(loc='best')
#设定横纵轴标题
plt.xlabel('features')
plt.ylabel('feature magnitude')
#显示图形
plt.show()
```



观察左图可以看出，数据集中各特征的量级新差异还是较大的。如：第1个特征“城市犯罪率”最小值在 10^{-2} 、最大值达到了 10^2 (有可能是一个错误数据点)，而第10特征“税收”的最小值和最大值都在 $10 \sim 10^{12}$ 之间。接下来，我们对数据进行预处理。

3. 对训练集和测试集进行数据预处理

```
In [7]: # 导入数据预处理工具
from sklearn.preprocessing import StandardScaler
# 对训练集和测试集进行数据预处理
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
plt.plot(X_train_scaled.min(axis=0), 'v', label='train set min')
plt.plot(X_train_scaled.max(axis=0), '^', label='train set max')
plt.plot(X_test_scaled.min(axis=0), 'v', label='test set min')
plt.plot(X_test_scaled.max(axis=0), '^', label='test set max')
plt.yscale('log')
# 设置图注位置
plt.legend(loc='best')
# 设置横纵轴标题
plt.xlabel('scaled features')
plt.ylabel('scaled feature magnitude')
# 显示图形
plt.show()
```



从左图可以看出，对训练集和测试集进行预处理后，基本上所有特征的最大值都不会超过10，且最小值也都趋近于0，以至于图中未能显示(这与使用的预处理工具原理有关，后续案例会详细介绍数据预处理方法)。接下来，我们用预处理后的数据重新训练模型。

4. 用预处理后的数据重新训练模型


```
In [8]: ▶ #使用预处理后的数据重新训练模型
for kernel in ['linear', 'rbf']:
    svr = SVR(kernel=kernel)
    svr.fit(X_train_scaled, y_train)
    print('数据预处理后', kernel, '核函数的模型训练集得分: {:.3f}'.format(
        svr.score(X_train_scaled, y_train)))
    print('数据预处理后', kernel, '核函数的模型测试集得分: {:.3f}'.format(
        svr.score(X_test_scaled, y_test)))
```

```
数据预处理后 linear 核函数的模型训练集得分: 0.706
数据预处理后 linear 核函数的模型测试集得分: 0.698
数据预处理后 rbf 核函数的模型训练集得分: 0.665
数据预处理后 rbf 核函数的模型测试集得分: 0.695
```

从以上结果可以看到，经过预处理后，linear核的SVR得分变化不大，而rbf核的SVR得分有了巨大提升。那么，如果我们进一步调整rbf核的SVR模型参数，性能会不会还能继续提升呢？下面我们进行实验。

和SVC一样，SVR模型也有gamma和C两个参数，接下来我们对这两个参数进行调整。

5. 调整模型的C参数和gamma参数

```
In [9]:  #设置模型的C参数和gamma参数
svr = SVR(C=100, gamma=0.1)
svr.fit(X_train_scaled, y_train)
print('调整参数后的模型在训练集得分: {:.3f}'.format(
    svr.score(X_train_scaled, y_train)))
print('调整参数后的模型在测试集得分: {:.3f}'.format(
    svr.score(X_test_scaled, y_test)))
```

调整参数后的模型在训练集得分: 0.966

调整参数后的模型在测试集得分: 0.894

在以上程序代码中，设置了SVR模型的参数C=100、gamma=0.1。可以看出，通过调参，rbf核的SVR模型在训练集上得分已高达0.996，而在测试集上得分达到了0.894。可以说，经过预处理和调参后，现在的模型表现是能够接受的。

本章小结:

1. 线性可分支持向量机

线性可分支持向量机或硬间隔支持向量机。构建它的条件是训练数据线性可分，其学习策略是最大间隔法。可表示为凸二次规划问题。线性可分支持向量机的最优解 \mathbf{w}^* 、 b^* 存在且唯一，位于间隔边缘上的实例点为支持向量。最优分类超平面由支持向量完全决定。

2. 线性支持向量机

现实中训练数据是线可分的情形较少，训练数据往往是近似线性可分的，这时使用线性支持向量机或软间隔支持向量机。线性支持向量机是基本的支持向量机。线性支持向量的 \mathbf{w}^* 唯一但 b^* 不唯一。

3. 非线性支持向量机

在线性支持向量机学习的对偶问题中，用核函数替代内积，求解得到的就是非线性支持向量机：

$$f(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^* \right)$$

4. SVM应用中的核函数与参数选择

5. SVM分类和SVM回归举例(SVM分类、SVM回归各1例)

向量的范数：

$$\forall \mathbf{x} = (x_1, x_2, \dots, x_n)^T$$

$$L_0\text{-}0 \text{ norm} : \|\mathbf{x}\|_0 = \sum_{i=1}^n |x_i|^0 = \sum_{i=1}^n |\text{sgn}(x_i)|$$

$$L_1\text{-}1 \text{ norm} : \|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$$

$$L_2\text{-}2 \text{ norm} : \|\mathbf{x}\|_2 = \left(\sum_{i=1}^n |x_i|^2 \right)^{1/2}$$

$$L_p\text{-}p \text{ norm} : \|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

$$L_\infty\text{-}\infty \text{ norm} : \|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

本章主要参考文献:

- [01] 李航. 统计学习方法(第2版): 111-141, 清华版, 2019
- [02] 周志华. 机器学习-支持向量机: 121-139, 清华版, 2016
- [03] Nello Cristianini et al. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000
- [04] Nello Cristianini等. 支持向量机导论, 电子工业出版社, 2004
- [05] 孙即祥. 现代模式识别第2版, 高等教育出版社, 2008

课外作业5-6(SVM):

1. 简要说明支持向量机技术背后的基本思想，并说明软间隔的具体含义。
2. 简要说明核函数在非线性支持向量机中的作用。
3. 试给出采用Lagrange乘子法求解以下带等式约束的凸函数优化问题：

$$\min f = 2x_1^2 + 3x_2^2 + 3x_3^2$$

$$s.t. \quad 2x_1 + x_2 = 1$$

$$2x_2 - 3x_3 = 2$$

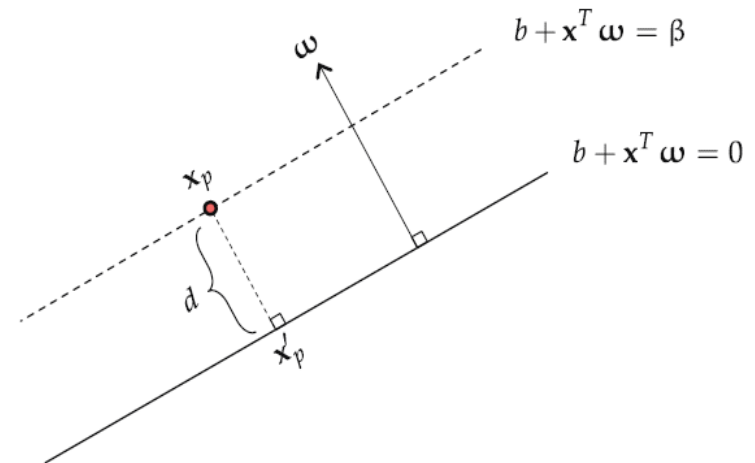
4. 试给出采用KKT条件(Kuhn-Tucker conditions)及广义Lagrange乘子法求解以下带不等式约束的凸函数优化问题：

$$\min f = x_1^2 - 2x_1 + 1 + x_2^2 + 4x_2 + 4$$

$$s.t. \quad x_1 + 10x_2 \geq 10$$

$$10x_1 - x_2 \leq 10$$

5. 试证：见右图，位于超平面 $b + x^T \omega = \beta$ 中的点 x_p 到超平面 $b + x^T \omega = 0$ 的代数距离为 $d = \frac{\beta}{\|\omega\|}$ 。



(第5题题图)

单元上机实验：

以本课件中的SVM分类问题和SVM回归问题 (或结合本人思考的与机器学习相关的实际应用问题) 案例程序为基础，实验调参并撰写实验报告。

End of this lecture.
Thanks!