支持向量机引论：从线性分类器到核方法

2021 年 1 月 19 日

摘 要

这篇文章从最简单的线性分类器出发，通过先后引入线性可分数据，线性不可分数据，高维空间中的线性可分数据，以及高维空间中的线性不可分数据，逐步推导出了支持向量机模型的大部分公式和参数优化目标，我们以循序渐进的写法，希望以此引发读者对支持向量机的兴趣，并向读者展示支持向量机与核方法理论的简洁与优美．主要成果：1）推导并讲解了支持向量机各种形式的表达式和参数优化目标，介绍了支持向量机的主要原理．2）细致介绍了多项式核的来由和意义，通过将内积运算都等价地替换为核函数的运算，简化了支持向量机的模型表达式，使得模型更容易被理解也更容易被实现．3）在附录中给出了支持向量机在 Mathematica 中的实现，我们在确保代码尽量简短易读的前提下写好了一份基本完整的代码，希望这份代码能够给读者提供一些帮助．4）使用 R 软件包 `e1071` 实现的 `svm` 函数，采用 RBF 核函数 $K(\boldsymbol{\mu},\boldsymbol{\nu}) = \exp\{-\gamma\|\boldsymbol{\mu}-\boldsymbol{\nu}\|^2\}$，在参数 γ 取值 0.001862196 到 0.001901141 的范围内，在 MNIST 数据集上，得到 10 折交叉验证平均正确率：0.993763298．我们想要表达的思想是：支持向量机并不复杂，只要愿意花时间了解，谁都可以玩出花样来，并且事实上，随着越来越多强大的核函数被开发出来，支持向量机的本领也变得越来越高强．

关键词：　支持向量机　径向基函数　MNIST 数据集

目录

1 引言

1.1 分类器

给定带有二值标签的数据集 $\mathcal{D}=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，其中 $\boldsymbol{x}_i\in\mathbb{R}^m, i=1,2,\cdots,N$ 为样本原始特征，$y_i\in\{-1,1\}, i=1,2,\cdots,N$ 为标签，我们希望从中寻找出一个规律，具体来说是一个函数 $f:\mathbb{R}^m\to\{-1,1\}$，使得当我们面临一个新样本 $\boldsymbol{x}^\star$ 时，这个函数（这个规律）能够告诉我们 $\boldsymbol{x}^\star$ 对应的标签 $y^\star$ 是 -1 还是 1．这样的对应规则 f 就叫做 “分类器”(Classifier)．本文主要探讨的支持向量机 (Support Vector Machine, SVM) 也是属于一种分类器．形象地来说，$\boldsymbol{x}_i$ 可以是图片上各个像素值组成的向量，y_i 的 -1 可以对应为猫，1 可以对应为狗，这时 $f:\mathbb{R}^m\to\{-1,1\}, x^\star\mapsto y^\star$ 就是一个能够告诉我们图片中的动物是猫还是狗的分类器．再举一例，令 $\boldsymbol{x}_i$ 表示电子邮件字符向量，再令 y_i 取 -1 对应垃圾邮件，取 1 对应正常邮件，那么这样的分类器 $f:\mathbb{R}^m\to\{-1,1\}, \boldsymbol{x}^\star\mapsto y^\star$ 就是一个垃圾邮件检测器，它能自动判断一封邮件是不是垃圾邮件．由此可见，分类器对于人类的生产和生活都在时刻产生着帮助．

1.2 支持向量机

线性分类器的思想在上世纪 6 0 年代就已经出现，简单来说，就是用一个超平面 $\boldsymbol{\omega}\cdot\boldsymbol{x}+b=0$ 将样本 $\boldsymbol{x}\in\mathcal{D}$ 分成两类，把 $\boldsymbol{x}$ 代入 $\boldsymbol{\omega}\cdot\boldsymbol{x}+b$，如果值是负的，那么 $\boldsymbol{x}$ 属于第一类，如果值是正的，那么 $\boldsymbol{x}$ 属于第二类，用 $f:\mathbb{R}^m\to\{-1,1\}$ 表示这个分类器，那么给定了分类器的参数 $\boldsymbol{\omega}$ 和 b 后，这个分类器的工作方式是这样的

$$f(\boldsymbol{\omega})=\begin{cases}-1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b<0\\ 1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b\geq 0\end{cases}\tag{1.2.1}$$

对于线性可分的数据集 $\mathcal{D}$，这样的分类器是完全奏效的，也就是说，如果 $\mathcal{D}$ 是线性可分的，那么一定可以找到这样的参数 $\boldsymbol{\omega},b$ 使得 f 百分之百地把 $\mathcal{D}$ 中的一切样本分对，可是，当 $\mathcal{D}$ 线性不可分时，就不存在任何可能的超平面把所有数据点都分对，这个时候人们引出了两种解决方法，一是放宽条件，允许一部分样本点被分错，但是要使得这部分分错的样本点尽可能地少，这种方法叫做 “软间隔”(Soft margin)，显然，软间隔方法仍然只是治标不治本，于是人们想出，将现有样本点映射到高维空间，或者使得样本更好分的空间去，可是去高维空间的内积计算成本难以接受，于是最终人们发现了 “核方法” 或者叫核技巧 (Kernal trick)，我们假设 $\phi:\mathbb{R}^m\to\mathbb{R}^n$ 是这样一个把样本点从低维空间映射到高纬空间的函数，那么一个核函数 $K:\mathbb{R}^m\times\mathbb{R}^m\to\mathbb{R}$ 满足这样的性质：对任意 $\boldsymbol{x}_1,\boldsymbol{x}_2\in\mathbb{R}^m$，恒有 $K(\boldsymbol{x}_1,\boldsymbol{x}_2)=\phi(\boldsymbol{x}_1)\cdot\phi(\boldsymbol{x}_2)$，换句话说，原本在高维空间 $\mathbb{R}^n$ 中进行的内积运算，通过这样的核函数，可以等价地转移到当前较低维的空间 $\mathbb{R}^m$ 进行，从而节省了计算量．有了核技巧，人们可以肆无忌惮地将样本映射到要多高就多高维的空间，而不必面对高代价的高维空间内积计算，从而使得支持向量机的理论与应用都有了质的飞跃．

支持向量机的优点是理论丰富且优美，当我们应用支持向量机模型时，我们面对的不是向人工神经网络模型那样深不可测的黑盒子模型，我们面对的是，一个数学上清晰，一个理论上完备，一个既容易理解，又应用广泛功能强大的金鱼缸模型，支持向量机的一切都可以被计算、被预测、被推导，简而言之，支持向量机是一个值得每个人认真学习和研究的模型．

2 模型

2.1 对于线性可分数据集

给定线性可分数据集 $\mathcal{D}=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，我们希望构建一个线性分类器 $f:\mathbb{R}^m\to\{-1,1\}$，定义

$$f(\boldsymbol{x})=\begin{cases}-1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b<0\\ 1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b>0\end{cases}\tag{2.1.1}$$

我们希望对一切 $y_i=-1,i=1,2,\cdots,N$，都有 $f(\boldsymbol{x}_i)=-1$，对一切 $y_i=1,i=1,2,\cdots,N$，都有 $f(\boldsymbol{x}_i)=1$，问题是，这样的 $\boldsymbol{\omega},b$ 怎么求出来呢?

无论如何，f 首先要把 $\mathcal{D}$ 中所有的样本分对，具体地说，有这样的不等式

$$\begin{cases}\boldsymbol{\omega}\cdot\boldsymbol{x}_i+b\geq 1, & y_i=1\\ \boldsymbol{\omega}\cdot\boldsymbol{x}_i+b\leq -1, & y_i=-1\end{cases}\tag{2.1.2}$$

也就是说

$$y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}+b)\geq 1\tag{2.1.3}$$

不等式 (2.1.3) 缩小了参数 $\boldsymbol{\omega},b$ 的搜索范围，事实上，每一组满足不等式 (2.1.3) 的参数 $\boldsymbol{\omega},b$ 都能将已有样本点百分之百地分对.

考虑两组参数 $(\boldsymbol{\omega}_1,b_1)$ 和 $(\boldsymbol{\omega}_2,b_2)$，对应图 (2-1) 和图 (2-2)

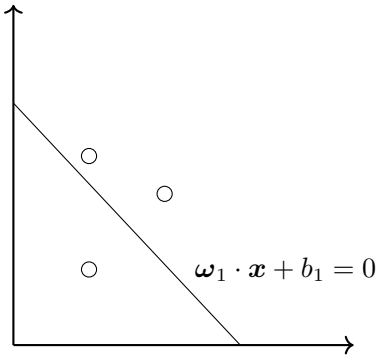

图 2-1: 模型 1

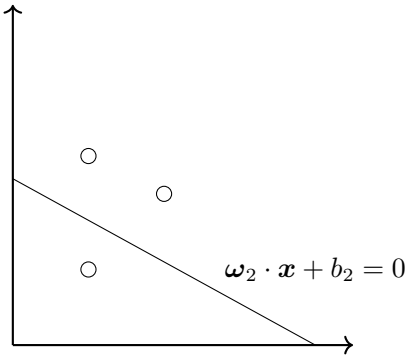

图 2-2: 模型 2

直觉告诉我们，应当选择模型 2 ，也就是应当选择 $(\boldsymbol{\omega}_2,b_2)$ 作为理想的模型参数，这是因为，模型 2 实际上应该会更加稳健，面对未知的样本点，它不容易错，或者说，它的「容错性」更好，比较它学到了更加广泛更加适用的规律，而模型 1 在未知的样本上很可能会犯错，因为它离另一类的样本点太近了，中间没有留多少余地.

令 $g(\boldsymbol{x})=\boldsymbol{\omega}\cdot\boldsymbol{x}+b$，考虑空间中的任意一点 $\boldsymbol{x}$，它到超平面的距离是

$$d(\boldsymbol{x},\boldsymbol{\omega})=\frac{|g(x)|}{\|\boldsymbol{\omega}\|}\tag{2.1.4}$$

如果说我们规定那些离超平面最近的点 $\boldsymbol{x}$ 代入 $g(\boldsymbol{x})$ 得到的函数值为 1，事实上这通过放缩 $\boldsymbol{\omega}$ 和 b 总是可以做到. 那么超平面到最近的点的距离就可以表示为

$$d=\frac{1}{\|\boldsymbol{\omega}\|}\tag{2.1.5}$$

而超平面的两面都会有样本点,那么这两面的样本点围成的这个包括了超平面的区间就叫做“间隔”(Margin),那么这个 Magin 的宽度就是 $\dfrac{2}{\|\boldsymbol{\omega}\|}$.

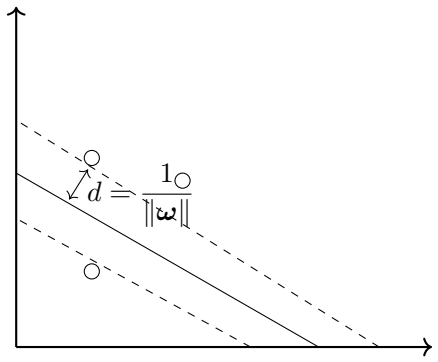

图 2-3: 间隔与支持向量

从几何上可以这样理解：向下和向下平移超平面，碰到最近的样本点就停止平移，这时超平面已经平移到了虚线的位置，虚线上的点就是最近的样本点，这些样本点我们叫它 “支持向量”(Support vectors)．总而言之，我们希望这个间隔越宽越好，因为间隔越宽，模型的泛化性能就越好，也就是说，要在所有满足条件的参数 $(\boldsymbol{\omega}, b)$ 中，寻找使得间隔宽带 $\dfrac{2}{\|\boldsymbol{\omega}\|}$ 最大的那组参数．寻找最优参数对应如下的优化问题：

$$\text{Minimize: } \frac{2}{\|\boldsymbol{\omega}\|} \tag{2.1.6}$$

$$\text{Subjected to: } y_i(\boldsymbol{\omega} \cdot \boldsymbol{x}_i + b) \geq 1, \quad i = 1, 2, \cdots, N \tag{2.1.7}$$

也等价于最小化 $\dfrac{1}{2}\|\boldsymbol{\omega}\|$，我们可以使用拉格朗日乘数法，令

$$\mathcal{L}_P(\boldsymbol{\omega}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{N} \lambda_i y_i(\boldsymbol{\omega} \cdot \boldsymbol{x}_i + b) + \sum_{i=1}^{N} \lambda_i \tag{2.1.8}$$

那么通过置偏导为零也就是令

$$\frac{\partial}{\partial \boldsymbol{\omega}}\mathcal{L}_P(\boldsymbol{\omega}, b, \boldsymbol{\lambda}) = 0, \quad \frac{\partial}{\partial b}\mathcal{L}_P(\boldsymbol{\omega}, b, \boldsymbol{\lambda}) = 0 \tag{2.1.9}$$

得到

$$\boldsymbol{\omega} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{2.1.10}$$

把上式代入式 (2.1.8) 使我们得到原问题 $\mathcal{L}_P$ 的对偶问题 $\mathcal{L}_D$：

$$\mathcal{L}_D(\boldsymbol{\lambda}) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{1 \leq i < j \leq N} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \tag{2.1.11}$$

式中 $\boldsymbol{\lambda}$ 要满足

$$\sum_{i=1}^{N} \lambda_i y_i = 0, \quad \text{并且} \boldsymbol{\lambda} \geq 0 \tag{2.1.12}$$

式 (2.1.11) 所描述的对偶问题是典型的二次规划问题，有许许多多现成的工具可用于求解．

并且式 (2.1.10) 告诉我们，只有那些对应 $\lambda_i \neq 0$ 的样本点才能够「影响」$\boldsymbol{\omega}$，事实上，那些对应的 $\lambda_i \neq 0$ 的样本点正是上图中的「支持向量」，当一个 SVM 模型训练完成后，大部分的样本点都可以丢弃，仅保留那少部分被称为「支持向量」的样本点就足以描述整个模型了．

求解对偶问题 $\mathcal{L}_D$ 使我们得到 $\boldsymbol{\lambda}$，得到了 $\boldsymbol{\lambda}$ 后依照式 (2.1.10) 就可以计算出 $\boldsymbol{\omega}$．任取一个支持向量，记为 $(\boldsymbol{x}_s, y_s)$，按理说，应有

$$y_s(\boldsymbol{x}_s \cdot \boldsymbol{\omega} + b) = 1 \tag{2.1.13}$$

式子两边都乘 y_s，再把式 (2.1.10) 的 $\boldsymbol{\omega}$ 公式代入，得到

$$\sum_{m\in S}\lambda_m y_m \boldsymbol{x}_m \cdot \boldsymbol{x}_s + b = y_s \tag{2.1.14}$$

式中，S 表示所有 $\lambda_i \neq 0$ 的下标 i 组成的集合，也就是那些支持向量的下标组成的集合．变换得

$$b = y_s - \sum_{m\in S}\lambda_m y_m \boldsymbol{x}_m \cdot \boldsymbol{x}_s \tag{2.1.15}$$

这样就求出了超平面参数 $(\boldsymbol{\omega}, b)$．

下面，我们来进行一遍示范．考虑线性可分数据集

$$\mathcal{D} = \{\{\{1,1\},-1\},\{\{2.2,2,2\},1\},\{\{1,2.5\},1\}\} \tag{2.1.16}$$

它作图出来如图 (2-4)

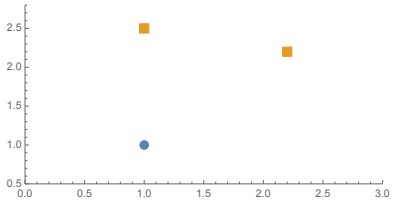

图 2-4: 示例线性可分数据集

容易知道圆点对应 -1 标签，方点对应 $+1$ 标签，我们将要找出一个超平面，把圆点和方点划开．我们求出

$$\lambda_1 = 0.944444, \quad \lambda_2 = 0.666667, \quad \lambda_3 = 0.277778 \tag{2.1.17}$$

这说明这三个点全都是支持向量，先求 $\boldsymbol{\omega}$，我们算得

$$\boldsymbol{\omega} = \begin{bmatrix} 0.333333 & 1.33333 \end{bmatrix} \tag{2.1.18}$$

再求 b，我们算得 $b = -2.66667$．那么这个超平面就是这样的：

$$0.333333x_1 + 1.33333x_2 - 2.666667 = 0 \tag{2.1.19}$$

超平面画出来如图 (2-5)

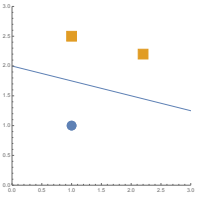

图 2-5: 超平面

我们可以验证这个超平面是对的——把支持向量代入 $\boldsymbol{\omega}\cdot\boldsymbol{x}+b$ 应该得到 ± 1，我们来看是不是：

$$0.333333 \times 1 + 1.333333 \times 1 - 2.666667 = -1 \tag{2.1.20}$$

$$0.333333 \times 2.2 + 1.333333 \times 2.2 - 2.666667 = 0.999998 \tag{2.1.21}$$

$$0.333333 \times 1 + 1.333333 \times 2.5 - 2.666667 = 0.999999 \tag{2.1.22}$$

说明这个超平面就是这样的．

2.2 软间隔方法与线性不可分数据集

考虑线性不可分数据集

$$\mathcal{D} = \{\{1,1\},\{2,-1\},\{3,-1\},\{4,1\}\} \tag{2.2.1}$$

如图 (2-1) 所示

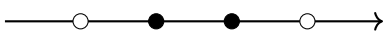

图 2-1: 一份线性不可分的数据集

由图可知，不存在这样的超平面能将所有的黑点和白点分到各自两边．这也就意味着，前面写出的约束条件

$$y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}+b)\geq 1,\quad i=1,2,\cdots,N \tag{2.2.2}$$

再也无法被满足．

结合实际情况，我们考虑适当放宽一下约束条件（式 (2.1.7)），用式 (2.2.3) 来表示放宽后的约束条件：

$$y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}_i+b)+\xi_i\geq 1,\quad i=1,2,\cdots,N \tag{2.2.3}$$

这里的 ξ_i 都是正数，虽然 $y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}_i+b)$ 不通过约束，但我们可以用一个正数 ξ_i 给它加上来帮它一把．当然，在最优化目标函数时我们希望 ξ_i 都尽可能地小．目标函数如下式

$$\Phi(\boldsymbol{\omega})=\frac{1}{2}\|\boldsymbol{\omega}\|^2+C\sum_{i=1}^{N}\xi_i,\quad(\xi_i>0) \tag{2.2.4}$$

式中 C 是一个给定的正数，它代表了我们对 ξ_i 的容忍度，换句话说，C 小，就代表我们漠不关心 ξ_i 究竟会取到多大，反之 C 大，ξ_i 就会很小．拉格朗日函数变为

$$\mathcal{L}_P(\boldsymbol{\omega},b,\boldsymbol{\xi})=\frac{1}{2}\|\boldsymbol{\omega}\|^2+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\lambda_i\left(y_i(\boldsymbol{\omega}\cdot\boldsymbol{x}_i+b)-1+\xi_i\right)-\sum_{i=1}^{N}\mu_i\xi_i \tag{2.2.5}$$

式中 $\boldsymbol{\lambda},\boldsymbol{\mu}$ 是拉格朗日系数．接下来对 $\mathcal{L}_P(\boldsymbol{\omega},b,\boldsymbol{\xi})$ 分别求对 $\boldsymbol{\omega}$，对 b，对 ξ_i，$i=1,2,\cdots,N$ 的偏导，并置零

$$\frac{\partial}{\partial\boldsymbol{\omega}}\mathcal{L}_P(\boldsymbol{\omega},b,\boldsymbol{\xi})=0\implies\boldsymbol{\omega}=\sum_{i=1}^{N}\lambda_i y_i\boldsymbol{x}_i \tag{2.2.6}$$

$$\frac{\partial}{\partial b}\mathcal{L}_P(\boldsymbol{\omega},b,\boldsymbol{\xi})=0\implies\sum_{i=1}^{N}\lambda_i y_i=0 \tag{2.2.7}$$

$$\frac{\partial}{\partial\xi_i}\mathcal{L}_P(\boldsymbol{\omega},b,\boldsymbol{\xi})=0\implies C=\lambda_i+\mu_i,\quad i=1,2,\cdots,N \tag{2.2.8}$$

将偏导置零得到的结果代回式 (2.2.5)，我们得到对偶问题

$$\mathcal{L}_D(\boldsymbol{\lambda})=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\boldsymbol{\lambda}^{\mathsf{T}}H\boldsymbol{\lambda} \tag{2.2.9}$$

其中 $H(i;j)=y_iy_j\boldsymbol{x}_i\cdot\boldsymbol{x}_j$ 并且要满足 $0\leq\boldsymbol{\lambda}\leq C$，以及 $\sum_{i=1}^{N}\lambda_iy_i=0$．

作为演示，考虑线性不可分数据集

$$\mathcal{D}=\{\{\{\{1.5,2\},-1\},\{\{2,1.5\},-1\},\{\{3,3\},-1\},\{\{2.1,3\},1\},\{2.7,2.4\},1\},\{\{2.5,3.5\},1\}\} \tag{2.2.10}$$

如图 (2-2) 所示.

对 $\mathcal{L}_D(\boldsymbol{\lambda})$ 进行最小化，取 $C=1$，得到

$$\boldsymbol{\omega}=\begin{bmatrix}0.423529 & 0.894118\end{bmatrix}^{\mathsf{T}},\quad b=-3.42353 \tag{2.2.11}$$

再尝试不同的 C 值，例如，取 $C=10$，得到

$$\boldsymbol{\omega}=\begin{bmatrix}1.25 & 1.25\end{bmatrix}^{\mathsf{T}},\quad b=-5.375 \tag{2.2.12}$$

图 (2-3)、图 (2-4) 为求得的超平面：

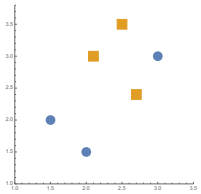

图 2-2: 线性不可分数据集

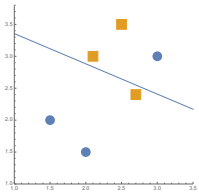

图 2-3: 参数 $C=1$

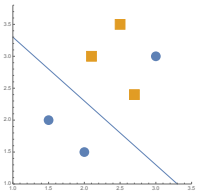

图 2-4: 参数 $C=10$

由此可见，当 C 的值增加时，模型的训练会变得越来越严格，也就是说对 ξ_i 的值越来越敏感，从而超平面到两边的距离也就越远，模型也就越稳健.

2.3 把样本投到别处

考虑线性不可分数据集 $\mathcal{D}=\{\{\boldsymbol{x}_i,y_i\},i=1,2,\cdots,12\}$，定义

$$(\boldsymbol{x}_i,y_i):=\begin{cases}\{\{\cos\theta(i),\sin\theta(i)\},-1\}, & i=1,2,3,4,5,6\\ \{\frac{1}{2}\{\cos\theta(i),\sin\theta(i)\},1\}, & i=7,8,9,10,11,12\end{cases} \tag{2.3.1}$$

令

$$\theta(i)=\frac{\pi}{6}+(i-1)\frac{\pi}{3}. \tag{2.3.2}$$

该数据集如图 (2-1) 所示：

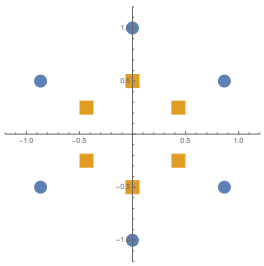

图 2-1: 一份线性不可分的数据集

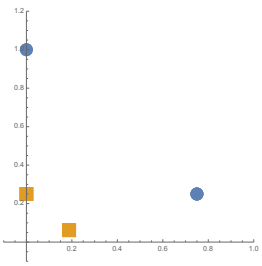

图 2-2: 投影到另外一个空间

确实找不到一条直线满足这条直线的一边全为方块而另一边全为圆点，可是我们发现，能够区分方块和圆点的最明显特征是它们围绕着一个公共圆心的距离，形象地说，我要是在这些点的样本中心点一个点，并以该点为圆心，画一个半径经过精心测量的圆，就足以正确划分全部样本点了.

令

$$\phi:\mathbb{R}^2\longrightarrow\mathbb{R}^2 \tag{2.3.3}$$

$$(x_1,x_2)\longmapsto(x_1^2,x_2^2) \tag{2.3.4}$$

表示我们要对数据集 $\mathcal{D}$ 应用的投影法则，就可以做出图 (2-2) 所示的图形了，圆点的半径大一些，所以它们总是满足 $x_1^2+x_2^2 \geq \frac{1}{2}$，而方块的半径小一些，所以方块总是满足 $x_1^2+x_2^2 < \frac{1}{2}$，你已经发现了，这个 $x_1^2+x_2^2-\frac{1}{2}=0$ 就可以用作是在新的空间 $\phi(\mathbb{R}^2)$ 中对样本点进行正确分类的超平面．把 x_1^2, x_2^2 都各自单独地看做是一个完整的个体，譬如说，令 $t_1=x_1^2, t_2=x_2^2$，那么方程 $t_1+t_2-\frac{1}{2}=0$ 是可以看做是线性的．

做了投影后，参数的估计与原先的并没有太大的差别，无非就是，把公式中出现 $\boldsymbol{x}_i$ 的地方，换成 $\phi(\boldsymbol{x}_i)$，把公式中出现 $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ 的地方，换成 $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$，下面我们就来试着求一下，所有作图和计算的代码都会在附录中给出，正文中只会体现计算结果和图形．

我们求出的参数为：

$$\lambda_i = 1.18519, \quad i = 1, 2, \cdots, 12; \tag{2.3.5}$$

$$\boldsymbol{\omega} = \begin{bmatrix} -2.66667 & -2.66667 \end{bmatrix}^{\mathsf{T}} \tag{2.3.6}$$

$$b = 1 \tag{2.3.7}$$

注意到这个超平面实际上是指投影到的那个空间 $\phi(\mathbb{R}^2)$ 的超平面，而不是原来空间 $\mathbb{R}^2$ 的超平面，所以，超平面的表达式为

$$2.66667x_1^2 + 2.66667x_2^2 = 1 \tag{2.3.8}$$

图 (2-3) 是超平面在原始空间中的形态，而图 (2-4) 是超平面在 $\phi(\mathbb{R}^2)$ 空间中的形态．

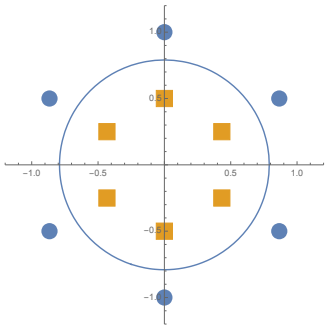

图 2-3: 超平面在原始空间中的形态

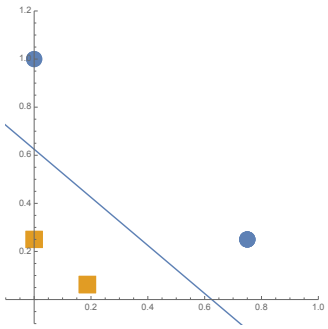

图 2-4: 超平面在投影空间中的形态

我们找到了一个合适的投影 $\phi : \mathbb{R}^m \to \mathbb{R}^n$，把问题漂亮地解决了．而且这次甚至没有用到软间隔，所有的样本都是严格地位于超平面的间隔之外．

更高维的空间有更高维的表现力，细小的变异在高维空间中得到了放大因而能够简单的线性判别器所识别出来，然而，在计算过程中，所有原先是 $\boldsymbol{x}_i$ 的地方都要被 $\phi(\boldsymbol{x}_i)$ 替代，也就是每当要用到 $\boldsymbol{x}_i$ 的值时，都首先要把 $\boldsymbol{x}_i$ 代入 ϕ 去计算 $\phi(\boldsymbol{x}_i)$ 的值，当这个投影法则 ϕ 变得越来越复杂，也就越来越难猜到 ϕ 的正确形式，并且也就会产生越来越不可接受的计算成本．

2.4 两全其美的核方法

考虑投影法则：

$$\phi : \mathbb{R}^m \longrightarrow \mathbb{R}^{\binom{m+2}{2}} \tag{2.4.1}$$

$$\boldsymbol{x}_{m\times 1} \longmapsto \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{bmatrix} \tag{2.4.2}$$

这个 ϕ 能够将 $\mathbb{R}^m$ 中的样本点，投影到一个 $\binom{m+2}{2}$ 维的空间，假如说现在我们已经有了 500 个变量的数据，也就是说 $m = 500$，那么 $\binom{502}{2}$ 将近一万，在一万维的空间中计算向量内积是非常昂贵的操作.

令

$$\boldsymbol{x}_1 = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \end{bmatrix}^{\mathsf{T}} \tag{2.4.3}$$

$$\boldsymbol{x}_2 = \begin{bmatrix} x_{21} & x_{22} & \cdots & x_{2m} \end{bmatrix}^{\mathsf{T}} \tag{2.4.4}$$

我们先计算 $\phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2)$：

$$\phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2) = 1 + 2\sum_{i=1}^{m} x_{1i}x_{2i} + \sum_{i=1}^{m} x_{1i}^2 x_{2i}^2 + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} 2x_{1i}x_{1j}x_{2i}x_{2j} \tag{2.4.5}$$

我们再计算 $(\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2$：

$$(\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2 = \sum_{i=1}^{m} x_{1i}^2 x_{2i}^2 + 2\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} x_{1i}x_{2i}x_{1j}x_{2j} + 2\sum_{i=1}^{m} x_{1i}x_{2i} + 1 \tag{2.4.6}$$

我们惊喜地发现：竟然有 $(\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2 = \phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2)$，这又有什么意义呢？我们都知道 $\boldsymbol{x}_1$ 有 m 个分量，通过 ϕ 投影到了 $\binom{m+2}{2}$ 维空间之后，$\phi(\boldsymbol{x}_1)$ 就有 $\binom{m+2}{2}$ 个分量，对于 $\boldsymbol{x}_2$ 也是如此，在任何情况下，我们都会认为计算两个 m 维向量的内积要比计算两个 $\binom{m+2}{2}$ 维向量的内积要省事得多.

令

$$K : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R} \tag{2.4.7}$$

$$\boldsymbol{x}_1, \boldsymbol{x}_2 \longmapsto (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2 \tag{2.4.8}$$

我们称 K 是一个核函数，这样定义的 K 有一个名字：它叫做多项式核 (Polynomial kernal)．简单来说就是因为它能把内积计算大大简化．从前，我们计算 $\phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2)$ 要去 $\binom{m+2}{2}$ 维空间计算，现在有了

K，我们不再需要真正地去计算 $\phi(\boldsymbol{x}_1)$ 也不需要计算 $\phi(\boldsymbol{x}_2)$ 更不需要去计算 $\phi(\boldsymbol{x}_1)\cdot\phi(\boldsymbol{x}_2)$，就可以知道 $\phi(\boldsymbol{x}_1)\cdot\phi(\boldsymbol{x}_2)$ 的计算结果是多少，因为

$$K(\boldsymbol{x}_1,\boldsymbol{x}_2)=\phi(\boldsymbol{x}_1)\cdot\phi(\boldsymbol{x}_2) \tag{2.4.9}$$

重要的是，对 $K(\boldsymbol{x}_1,\boldsymbol{x}_2)$ 的计算是在 m 维空间进行的．这样一来，我们既获得了高维空间的丰富表现力，又回避了高维空间的复杂计算，所以，我们不想把话说得太绝对，但核方法确实是两全其美的事．

在前面的案例中，我们最终都求出了超平面的几何参数 $\boldsymbol{\omega},b$，但在高维空间中，就算求出了 $\boldsymbol{\omega}$ 和 b，也不太方便把超平面「画」出来，毕竟我们能做出图形的只有三维空间、二维空间和一维空间．事实上，我们从来都不需要把 $\boldsymbol{\omega}$ 和 b 真正「求出来」：回归支持向量机模型：

$$f(\boldsymbol{x})=\begin{cases}-1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b<0,\\ 1, & \boldsymbol{\omega}\cdot\boldsymbol{x}+b\geq 0\end{cases} \tag{2.4.10}$$

令 $g(\boldsymbol{x})=\boldsymbol{\omega}\cdot\boldsymbol{x}+b$，代入上式得

$$f(\boldsymbol{x})=\operatorname{sgn}\left(g(\boldsymbol{x})\right)=\operatorname{sgn}\left(\boldsymbol{\omega}\cdot\boldsymbol{x}+b\right) \tag{2.4.11}$$

再把求导得到的式子

$$\boldsymbol{\omega}=\sum_{i=1}^{N}\lambda_i y_i\boldsymbol{x}_i \tag{2.4.12}$$

代入，得到

$$f(\boldsymbol{x})=\operatorname{sgn}\left(\sum_{i=1}^{N}\lambda_i y_i\boldsymbol{x}_i\cdot\boldsymbol{x}+b\right) \tag{2.4.13}$$

其中 $\operatorname{sgn}\cdot$ 表示的是符号函数，如果输入的值为负它输出 -1，如果输入的值非负它输出 1；并且，非支持向量的 $\lambda_i=0$，因此我们只需考虑支持向量，设全部支持向量的下标构成集合 S，那么有

$$f(\boldsymbol{x})=\operatorname{sgn}\left(\sum_{m\in S}\lambda_m y_m\boldsymbol{x}_m\cdot\boldsymbol{x}+b\right) \tag{2.4.14}$$

看到了式 (2.4.14) 我们可以说，一个支持向量机的全部参数就是它的支持向量！

如果已经将样本通过投影法则 $\phi:\mathbb{R}^m\to\mathbb{R}^n$ 投影到了高维空间，模型表达式式 (2.4.14) 写为：

$$f(\phi(\boldsymbol{x}))=\operatorname{sgn}\left(\sum_{m\in S}\lambda_m y_m\phi(\boldsymbol{x}_m)\cdot\phi(\boldsymbol{x})+b\right) \tag{2.4.15}$$

而有了核函数 $K:\mathbb{R}^m\times\mathbb{R}^m\to\mathbb{R}$，由于 $K(\boldsymbol{x}_1,\boldsymbol{x}_2)=\phi(\boldsymbol{x}_1)\cdot\phi(\boldsymbol{x}_2)$，所以我们根本就不需要专程跑到高维空间去计算投影点在高维空间的内积，我们只需计算 $K(\boldsymbol{x}_1,\boldsymbol{x}_2)$，也就是

$$f(\phi(\boldsymbol{x}))=\operatorname{sgn}\left(\sum_{m\in S}\lambda_m y_m K(\boldsymbol{x}_1,\boldsymbol{x}_2)+b\right) \tag{2.4.16}$$

那么这时候我们又可以说，一个支持向量机的全部参数就是它的支持向量外加上指定的核函数！

至此，我们希望做一些阶段性的小结，关于训练一个支持向量机的步骤：

(1) 使数据 $\mathcal{D}=\{(\boldsymbol{x}_i,y_i),i=1,2,\cdots,N\}$ 就位，选好核函数 $K:\mathbb{R}^m\times\mathbb{R}^m\to\mathbb{R}$．

(2) 计算 H 矩阵：$H=\sum_{i=1}^{N}y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)$

(3) 最小化 $\frac{1}{2}\boldsymbol{\lambda}^{\mathsf{T}}H\boldsymbol{\lambda}-\sum_{i=1}^{N}\lambda_i$，约束条件为：$\sum_{i=1}^{N}\lambda_i y_i=0, 0\leq\lambda_i\leq C$，其中 C 为人为选定的软间隔惩罚系数，并且 $C>0$．解出 $\boldsymbol{\lambda}$．

(4) 寻得支持向量下标集 $S=\{i:\lambda_i>0, 1\leq i\leq N, i\in\mathbb{N}^\star\}$，令 $g(\boldsymbol{x})=\sum_{m\in S}\lambda_m y_m K(\boldsymbol{x}_m,\boldsymbol{x})+b$，则 $f(\boldsymbol{x})=\operatorname{sgn} g(\boldsymbol{x})$ 为支持向量机．

至此，关于支持向量机的一切变得简洁、明朗又易于理解．

我们在附录中给出了关于这四个步骤的 Mathematica 代码实现．

值得注意的是，从开始到现在，我们处理的 $\boldsymbol{x}$ 一直都是数值型数据，也就是说我们迄今为止处理的 $\boldsymbol{x}$ 都是 m 维数值向量，可是，这又是必须的吗？$\boldsymbol{x}$ 必须是数（向量）吗？$\boldsymbol{x}$ 可不可以是一段文字，可不可以是一张图片，可不可以是一段声音，可不可以是一个单词？事实上，$\boldsymbol{x}$ 可以是能够被编码的一切客观实体，必须且只需有了相对应的核函数 $K:X\times X\to\mathbb{R}$，这个 $\boldsymbol{x}$ 所属的空间 X 是什么都行．

3 实践

有了强大的模型，我们已经迫不及待地想去拿些什么东西来赶快试试身手．在这一节中，我们将用前面我们编好的模型在一些公开数据集上做检验，看看效果如何，也算是验证我们已经正确地实现了支持向量机．

3.1 费希尔的鸢尾花实验

安德森鸢尾花卉数据集 (Anderson's Iris data set) 最初是埃德加 · 安德森从加拿大加斯帕半岛上的鸢尾属花朵中提取的形态学变异数据，后由费希尔作为判别分析的一个例子，运用到统计学中．

表 3-1: Iris 数据集简介

变量名	类型	取值范围	单位
`Species`	离散型	`setosa`,`versicolor`,`virginica`	（不适用）
`SepalLength`	数值型	$[4.3, 7.9]$	厘米 (cm)
`SepalWidth`	数值型	$[2.0, 4.4]$	厘米 (cm)
`PetalLength`	数值型	$[1.0, 6.9]$	厘米 (cm)
`PetalWidth`	数值型	$[0.1, 2.5]$	厘米 (cm)

该数据集的简介见表 (3-1)，第一个变量 `Species` 一般作为因变量来分析，后四个 `*Length`, `*Width` 主要是形态学数据，简而言之就是说，人们根据这份数据，可以训练出一个分类模型（统计学中叫判别模型）来仅仅根据鸢尾花的形态学数据就能够说出鸢尾花所属的种类．

我们考虑使用多项式核

$$K : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R} \tag{3.1.1}$$

$$\boldsymbol{x}_1, \boldsymbol{x}_2 \longmapsto (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2 \tag{3.1.2}$$

来训练两个 SVM，一个 SVM 将所有鸢尾花分成 `setosa` 和其他，另一个 SVM 在 `vericolor` 和 `virginica` 之间进行区分．这就是我们使用两个 SVM 来处理三分类问题的思路．

令 $\mathcal{M}_1$ 表示区分 `setosa` 和其它的模型，令 $\mathcal{M}_2$ 表示区分 `versicolor` 和 `virginica` 的模型．我们求出 $\mathcal{M}_1$ 的支持向量为 42 号样本和 99 号样本，$\mathcal{M}_1$ 的正确率为 100%．

表 3-2: $\mathcal{M}_1$ 模型的支持向量

Species	SepalLength	SepalWidth	PetalLength	PetalWidth
`setosa`	4.5	2.3	1.3	0.3
`versicolor`	5.1	2.5	3	1.1

我们求出 $\mathcal{M}_2$ 模型的正确率为 100%．

表 3-3: $\mathcal{M}_2$ 模型的支持向量

Species	SepalLength	SepalWidth	PetalLength	PetalWidth
versicolor	5.9	3.2	4.8	1.8
versicolor	6.3	2.5	4.9	1.5
versicolor	6.1	2.8	4.7	1.2
versicolor	6.7	3.	5.	1.7
versicolor	6.	2.7	5.1	1.6
virginica	4.9	2.5	4.5	1.7
virginica	6.	2.2	5.	1.5
virginica	6.2	2.8	4.8	1.8
virginica	6.3	2.8	5.1	1.5
virginica	6.1	2.6	5.6	1.4
virginica	6.	3.	4.8	1.8
virginica	5.9	3.	5.1	1.8

本小节结论：带多项式核 $K : \boldsymbol{x}_1, \boldsymbol{x}_2 \mapsto (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + 1)^2$ 的 SVM 能够正确区分 Iris 数据集的所有样本.

3.2 在 MNIST 数据集上进行测试

MNIST 数据集包括 60000 张手写数字的图片，图片的长宽都是标准的 28 像素乘以 28 像素，标签是图片上的数字，标签取值 0 到 9 . 在本小节中，我们将取 MNIST 数据集的一个子集——仅包含标签 0 和标签 1 的数据，对 0 与 1 的手写图片做分类，我们还将单独划分测试集与训练集，以此得到模型的更加真实的测试成绩.

3.2.1 预处理

图片在计算机中能够以矩阵的形式存储，彩色的一张图片可能要用到 3 个矩阵（或者 4 个），分别存储 R 值、G 值和 B 值（可能还要加上透明度 alpha 值），黑白的一张图片只需一个矩阵就可以表示了，一张 28 像素乘以 28 像素的图片就可以被表示为一个 28 行 28 列的矩阵，矩阵在计算机中实际上对应的是二维数组（相差无几），矩阵的每个元素，实际上对应的是亮度，值越接近 1 ，图片的这个像素点看起来就越接近白色，值越接近 0 ，图片的这个像素点看起来就越接近黑色. 人用写字跟计算机在屏幕上打印字符不同，人用笔是有力度的，而且不同的人写字的力度不同，有的字迹浅，有的字迹深，打个形象一点的比方，让计算机打印一张图片 0 ，这个图片 0 的矩阵看起来可能是下面这样子的

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.2.1}$$

而如果让一个人来在一张纸上写一个 0，然后再拍照取得这个 0 的图片，再对这个图片矩阵化，得到的矩阵可能是这样子的：

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0.1 & 0.2 & 1 & 1 \\ 1 & 0.11 & 1 & 1 & 0.02 & 1 \\ 1 & 0.115 & 1 & 1 & 0.3 & 1 \\ 1 & 0.121 & 1 & 1 & 0.1 & 1 \\ 1 & 0.001 & 1 & 1 & 0.187 & 1 \\ 1 & 1 & 0.2 & 0.12 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.2.2}$$

可能是纸张材质的问题，也可能是笔的问题，也可能是写字的人写得不够用力，总之，这个 0 的笔划看起来就是有的地方浅，有的地方淡，可是，我们认出 0 这个字符，会考虑到 0 的笔划本身的深浅度吗？就跟字迹的颜色一样，笔划本身的深浅度，写字用力的程度，对于我们识别出一个字符字迹是 0 还是 1，是毫无帮助的．我们看的只是这个符号的总体上的形状，而不是它的字体颜色或者深浅度．就好比汽车有红色的、蓝色的、银色的和黑色的，但是把汽车和行人区分开来的最重要特征，还是汽车和人的的几何轮廓，不管这个汽车漆成什么色，不管这个汽车的挡风玻璃是用什么材料做的，都不影响它成为一部汽车．

令

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N \tag{3.2.3}$$

表示 N 张图片的矩阵经过扁平化（1 维化）之后得到的向量，每个矩阵是 28×28 的，那么每个 $\boldsymbol{x}_i$ 就是 $784 = 28 \times 28$ 维的，只不过把矩阵的元素全写成一行了你可以这样认为．我们这样进行预处理，

$$\boldsymbol{x}_j^{\star}(i) = \begin{cases} 0, & \boldsymbol{x}_j(i) < 0.5 \\ 1, & \boldsymbol{x}_j(i) \geq 0.5 \end{cases}, \quad j = 1, 2, \cdots, N \; i = 1, 2, \cdots, 784 \tag{3.2.4}$$

也就是逐一检查每张图片的每一个像素点，如果值小于 0.5，那么替换为 0，反正替换为 1．

举例如图 (3-1) 与图 (3-2)

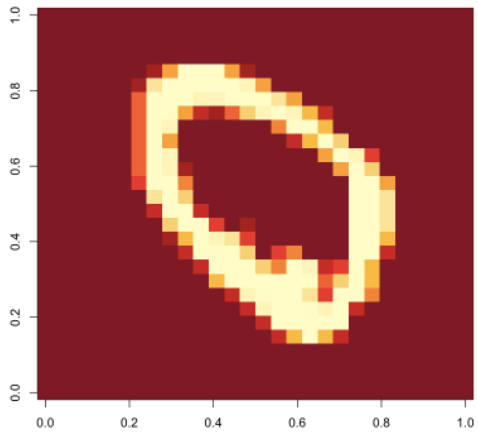

图 3-1: 处理前

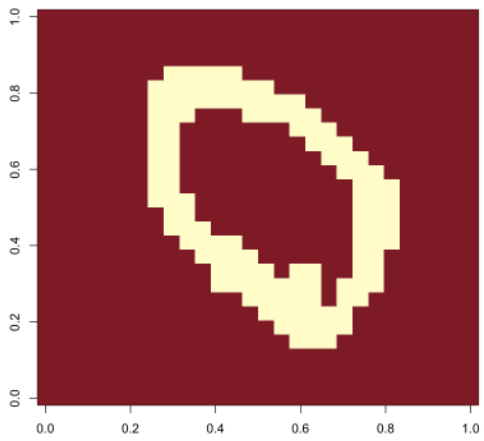

图 3-2: 处理后

正如我们看到图 (3-1) 有深有浅，而图 (3-2) 剩下的只是形状．

另外，我们注意到每张图片都有很多空白的部分，譬如说，有那么一些公共区域，每一张图片在此区域中的像素值都是 1，也就是说图片的公共空白部分，这部分显然对于分类而言是没有什么帮助的，因为信息只能体现在变异中，而这部分常值变量并没有贡献任何变异 (Variance)，所以应当将它们剔除掉．

令

$$P = \{i : 1 = \boldsymbol{x}_1(i) = \boldsymbol{x}_2(i) = \cdots = \boldsymbol{x}_N(i), i = 1, 2, \cdots, 784\} \tag{3.2.5}$$

表示那些公共空白区域的像素序号组成的集合，令

$$\boldsymbol{x}_1^\star, \boldsymbol{x}_2^\star, \cdots \boldsymbol{x}_N^\star \tag{3.2.6}$$

表示剔除了这部分分量后的 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots \boldsymbol{x}_N$.

3.2.2 将全部样本用做训练集

在附录中，我们给出了详细的 R 源代码，使用 R 的原因是 e1071 包的 svm 函数实现了 SMO 算法，使得快速地求解规模巨大的二次规划问题（尤其是 SVM 产生的二次规划问题）成为可能，也就是说，用了 e1071，比我们自己的实现更加节省时间.

样本为所有标记为 0 的图片，以及所有标记为 1 的图片，我们拿所有样本来训练，再拿所有样本来测试. 交叉验证 (Cross-validation) 放到后面去做. 具体的预处理方法如前所述，具体代码请参阅附录有关章节.

软件使用的核函数为 RBF，表达式为

$$K(\boldsymbol{\mu}, \boldsymbol{\nu}) = \exp\{-\gamma \|\boldsymbol{\mu} - \boldsymbol{\nu}\|^2\} \tag{3.2.7}$$

总共有 12665 个样本参加了训练，参数为 $\gamma = 0.00186219739292365$，对应 -1 类的支持向量有 542 个，对应 1 类的支持向量有 253 个. 正确率：0.999842084484801.

由此我们断定，预处理方法正确，参数基本正确，核函数选择正确，模型基本上具备从图片数据集中学习特征的能力.

3.2.3 交叉验证

交叉验证方法有许多种，其中一种较为简单，设 $f : \mathcal{D} \to \{-1, 1\}$ 为二分类器，k 折交叉验证主要步骤是：1）打乱样本序号；2）将样本分为 k 等份；4）将 k 份中的其中一份选做测试集，每选一次，都让余下 $k-1$ 份样本做训练集，让 k 份样本中的每一份都当一遍测试集. 5）计算 f 在 k 份样本上的错误 $E_1, E_2, \cdots, E_k$，求平均.

我们运行了一次 10 折交叉验证，具体代码请参阅附件有关章节，相关参数见表 (3-1)：

表 3-1: 10 折交叉训练参数

第几折	RBF 的 γ 值	准确率
1	0.001873	0.991318
2	0.001883	0.995264
3	0.001876	0.996843
4	0.001862	0.994475
5	0.001866	0.990529
6	0.001901	0.992897
7	0.001862	0.992107
8	0.001876	0.996054
9	0.001869	0.992897
10	0.001883	0.995249

我们求得 10 折交叉验证的平均准确率为 0.993763298944333.

4 总结

这篇文章大概分为两个部分，前半部分是从最基本的线性分类器开始推导支持向量机的理论与公式，最后以核函数作为这一部分的结束．通过阅读这一部分，读者将能够对支持向量机的基本理论增加多一些的了解，并且初步体会到支持向量机的灵活、简洁和强大．

第二部分主要是在一些公开数据集上做训练，一个是 Iris 数据集，我们在 Iris 上训练的模型得到了 100% 的准确率．还有一个是 MNIST 的手写数字训练集，我们在标签为 0 和标签为 1 的图片上训练能够区分手写数字 0 和手写数字 1 的模型，而后我们使用 `e1071` 包的 `svm` 函数和我们自己编写的 `R` 程序进行了 10 折交叉验证，取得了较好的成绩．

参考文献

[1] 李航. 统计学习方法. 清华大学出版社, 2012.

[2] 周志华. 机器学习. 清华大学出版社, 2016.

附录 A　第 2 节计算代码

除非特别说明，计算和作图都是用的 Wolfram Mathematica 12. 代码最左边的数字为代码的行号用作标记方便阅读而不是代码本身. 形如

```
1       (* 注释内容 *)
```

的代码为注释.

我们可算是用 Mathematica 语言手动实现了一遍支持向量机的简单形式，当然也包括了最基本的核方法，只有二次优化的部分，我们没有自己实现，二次优化是通过调用 Mathematica 自带的函数进行的.

A.1　对于线性可分数据集

```
(* 输入数据 *)
classA = {{1, 1}}
classB = {{2.2, 2.2}, {1,2.5}}

(* 对数据作图 *)
scatterPlot = ListPlot[ {classA, classB},
    AspectRatio -> 1, PlotMarkers -> {Automatic, Large}, PlotRange -> {{0, 3}, {0, 3}}
];

(* 计算样本总量 *)
pop = Length[classA] + Length[classB]

(* 计算超平面参数 *)
x[i_ /; 1 <= i <= Length[classA]] := classA[[i]]
x[i_ /; Length[classA] + 1 <= i <= pop] := classB[[i - Length[classA]]]

y[i_ /; 1 <= i <= Length[classA]] := -1
y[i_ /; Length[classA] + 1 <= i <= pop] := 1
h[i_, j_] := y[i]*y[j]*Inner[Times, x[i], x[j], Plus]
matH = Table[h[i, j], {i, 1, pop}, {j, 1, pop}];

lD = ({{Sum[lambda[i], {i, 1, pop}]}} - 1/2*{
    Table[lambda[i], {i, 1, pop}]
}.matH.Transpose[
    {Table[lambda[i], {i, 1, pop}]}
])[[1, 1]];

(* 运行二次优化程序 *)
sol = QuadraticOptimization[
    -lD,
    Sum[lambda[i]*y[i], {i, 1, pop}] == 0 && Table[0 <= lambda[i], {i, 1, pop}],
    Table[lambda[i], {i, 1, pop}]
];

(* 求omega *)
omega = Sum[lambda[i]*y[i]*x[i], {i, 1, pop}] /. sol

(* setS为那些支持向量的下标 *)
setS = Select[Table[i, {i, 1, pop}], Values[sol][[#]] > 10^-3 &]
s = First[setS]

(* 求beta *)
beta = y[s] - Sum[ lambda[m]*y[m]*Inner[Times, x[m], x[s], Plus], {m, setS}] /. sol

(* 画出超平面 *)
Show[{ scatterPlot,
    Plot[
        ((-beta) - (omega[[1]]) x)/(omega[[2]]),
        {x, 0, 3.5},
        AspectRatio -> 1, PlotRange -> {{0, 3.5}, {1, 3.8}}
    ]
}]
```

A.2 软间隔方法与线性不可分数据集

```
(* 输入数据 *)
classA = {{1.5, 2}, {2, 1.5}, {3, 3}}
classB = {{2.1, 3}, {2.7, 2.4}, {2.5, 3.5}}

(* 对数据作图 *)
scatterPlot = ListPlot[
    {classA, classB},
    AspectRatio -> 1,
    PlotMarkers -> {Automatic, Large},
    PlotRange -> {{1, 3.5}, {1, 3.8}}
]

(* 计算样本总量 *)
pop = Length[classA] + Length[classB]

(* 计算超平面参数 *)
x[i_ /; 1 <= i <= Length[classA]] := classA[[i]]
x[i_ /; Length[classA] + 1 <= i <= pop] := classB[[i - Length[classA]]]

y[i_ /; 1 <= i <= Length[classA]] := -1
y[i_ /; Length[classA] + 1 <= i <= pop] := 1
h[i_, j_] := y[i]*y[j]*Inner[Times, x[i], x[j], Plus]
matH = Table[h[i, j], {i, 1, pop}, {j, 1, pop}];

lD = ({{Sum[lambda[i], {i, 1, pop}]}} - 1/2*{
    Table[lambda[i], {i, 1, pop}]
}.matH.Transpose[
    {Table[lambda[i], {i, 1, pop}]}
])[[1, 1]];

(* 参数C取100 *)
c = 100

(* 运行二次优化程序 *)
sol = QuadraticOptimization[
    -lD,
    Sum[lambda[i]*y[i], {i, 1, pop}] == 0 && Table[0 <= lambda[i] <= c, {i, 1, pop}],
    Table[lambda[i], {i, 1, pop}]
];

(* 求omega *)
omega = Sum[lambda[i]*y[i]*x[i], {i, 1, pop}] /. sol

(* setS为那些支持向量的下标 *)
setS = Select[Table[i, {i, 1, pop}], Values[sol][[#]] > 10^-3 &]
s = First[setS]

(* 求beta *)
beta = y[s] - Sum[
    lambda[m]*y[m]*Inner[Times, x[m], x[s], Plus],
{m, setS}] /. sol

(* 画出超平面 *)
Show[{
    scatterPlot,
    Plot[
        ((-beta) - (omega[[1]]) x)/(omega[[2]]),
        {x, 1, 3.5},
        AspectRatio -> 1, PlotRange -> {{1, 3.5}, {1, 3.8}}
    ]
}]
```

A.3 把样本投影到别处

```
(*输入数据*)
classA = Table[{Cos[\[Theta]], Sin[\[Theta]]}, {\[Theta], Pi/6, 2 Pi - Pi/6, Pi/3}]
classB = Table[1/2 {Cos[\[Theta]], Sin[\[Theta]]}, {\[Theta], Pi/6, 2 Pi - Pi/6, Pi/3}]

(*对数据作图*)
scatterPlot = ListPlot[
    {classA, classB}, AspectRatio -> 1,
    PlotMarkers -> {Automatic, Large},
    PlotRange -> {{-1.2, 1.2}, {-1.2, 1.2}}
]

(* 画出数据在投影空间中的散点图 *)
scatterPlot2 = ListPlot[
    {classA*classA, classB*classB}, AspectRatio -> 1,
    PlotRange -> {{-0.1, 1}, {-0.1, 1.2}},
    PlotMarkers -> {Automatic, Large}
]

(* 计算样本总量 *)
pop = Length[classA] + Length[classB]

(* 定义投影函数 *)
\[Phi][input_] := {input[[1]]^2, input[[2]]^2}

(*计算超平面参数*)
x[i_ /; 1 <= i <= Length[classA]] := classA[[i]]
x[i_ /; Length[classA] + 1 <= i <= pop] := classB[[i - Length[classA]]]

y[i_ /; 1 <= i <= Length[classA]] := -1
y[i_ /; Length[classA] + 1 <= i <= pop] := 1

h[i_, j_] := y[i]*y[j]*Inner[Times, \[Phi][x[i]], \[Phi][x[j]], Plus]
matH = Table[h[i, j], {i, 1, pop}, {j, 1, pop}];

lD = ({{Sum[lambda[i], {i, 1, pop}]}} - 1/2*{
    Table[lambda[i], {i, 1, pop}]
}.matH.Transpose[
    {Table[lambda[i], {i, 1, pop}]}
])[[1, 1]];

(*运行二次优化程序*)
sol = QuadraticOptimization[
    -lD,
    Sum[lambda[i]*y[i], {i, 1, pop}] == 0 && Table[0 <= lambda[i], {i, 1, pop}],
    Table[lambda[i], {i, 1, pop}]
];

(*求omega*)
omega =
Sum[lambda[i]*y[i]*\[Phi][x[i]], {i, 1, pop}] /. sol

(*setS为那些支持向量的下标*)
setS =
Select[Table[i, {i, 1, pop}], Values[sol][[#]] > 10^-3 &]
s = First[setS]

(*求beta*)
beta =
y[s] - Sum[
    lambda[m]*y[m]*Inner[Times, \[Phi][x[m]], \[Phi][x[s]], Plus],
    {m, setS}
] /. sol

(* 画出投影空间的超平面 *)
Show[{
    scatterPlot2,
    Plot[((-beta) - (omega[[1]]) x)/(omega[[2]]), {x, -1.2, 1.2}, AspectRatio -> 1]
}]
```

```
69
70  (* 画出原始空间的超平面 *)
71  Show[{
72      scatterPlot,
73      ContourPlot[
74          omega[[1]] x1^2 + omega[[2]] x2^2 + beta == 0, {x1, -1, 1}, {x2, -1, 1}
75      ]
76  }]
```

A.4 两全其美的核方法

A.4.1 测试核函数

```
(* 定义多项式映射 *)
phi[x_List] := Module[
    {
        m = Length[x],
        constantTerm = {1},
        linearTerm = Sqrt[2]*x,
        pureQuadraticTerm = x*x,
        quadraticCrossTerm
    },
    (
        quadraticCrossTerm = Table[
            Sqrt[2]*x[[i]]*x[[j]], {i, 1, m - 1}, {j, i + 1, m}
        ] // Flatten;

        Join[
            constantTerm,
            linearTerm,
            pureQuadraticTerm,
            quadraticCrossTerm
        ]
    )
]

(* 定义多项式核函数 *)
polynomialKernal[x_List, y_List] := (Inner[Times, x, y, Plus] + 1)^2

(* 测试向量 *)
x1 = {x11, x12, x13, x14}
y1 = {y11, y12, y13, y14}

(* 应该输出True *)
FullSimplify[
    polynomialKernal[x1, y1] == Inner[Times, phi[x1], phi[y1], Plus]
]
```

A.4.2 测试数据集

```
(* 定义多项式核 *)
polynomialKernal[x_List, y_List] := (Inner[Times, x, y, Plus] + 1)^2

(* 产生数据 *)
classA = RandomVariate[NormalDistribution[1, 1], {100, 3}];
classB = RandomVariate[NormalDistribution[4, 1], {100, 3}];

(* 计算各类样本量和总样本量 *)
numA = Length[classA]
numB = Length[classB]
pop = numA + numB

(* 构建自变量矩阵和因变量向量 *)
pointsX = Join[classA, classB];
pointsY = Join[Table[-1, numA], Table[1, numB]];

(* 计算H矩阵 *)
matH = Outer[Times, pointsY, pointsY]* Outer[polynomialKernal, pointsX, pointsX, 1];

(* 设定惩罚系数 *)
c = 10000

(* 求解模型参数 *)
matA = Join[IdentityMatrix[pop], -IdentityMatrix[pop]];
vecB = Join[Table[0, pop], Table[c, pop]];
vecC = Table[1, pop];

(* 设自变量 t *)
(* 最小化 1/2 t^T . H . t - sum_{i=1}^N t_i *)
(* 约束条件: sum_{i=1}^N t_i y_i == 0 *)
(* 约束条件: 0 <= t_i <= C *)
sol = QuadraticOptimization[
    {matH, -vecC}, {matA, vecB},
    PerformanceGoal -> "Quality"
];

(* 求支持向量下标 *)
lambda[i_] := sol[[i]]
setS = Select[Range[1, pop], lambda[#] > 10^-2 &]

(* 定义判别辅助函数 *)
yi[m_] := pointsY[[m]]
xi[m_] := pointsX[[m]]
g[x_] := Sum[lambda[m]*yi[m]*polynomialKernal[xi[m], x], {m, setS}]

(* 运行回测 *)
predictVal = Sign[g[xi[#]]] & /@ Range[1, pop];

(* 查看回测结果 *)
Transpose[{predictVal, pointsY}] // Counts

(* 查看预测值为-1，实际值也为-1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, -1}]

(* 查看预测值为1，实际值也为1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, 1}]

(* 查看预测值为-1，实际值却为1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, 1}]

(* 查看预测值为1，实际值却为-1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, -1}]
```

附录 B　第 3 节计算代码

B.1　费希尔的鸢尾花实验

```
(* 定义多项式核 *)
polynomialKernal[x_List, y_List] := (Inner[Times, x, y, Plus] + 1)^2

(* 载入数据 *)
iris = ResourceData["Sample Data: Fisher's Irises"];

(* 整理数据成数值形式 *)
pointsX = {
    iris[All, 2] // Normal // QuantityMagnitude,
    iris[All, 3] // Normal // QuantityMagnitude,
    iris[All, 4] // Normal // QuantityMagnitude,
    iris[All, 5] // Normal // QuantityMagnitude
} // Transpose;

(* 给标签赋值 *)
pointsY = ( iris[All, 1] // Normal) /. {
    "setosa" -> 1,
    "versicolor" -> -1,
    "virginica" -> -1
};

(* 计算样本总量 *)
pop = Length[pointsX]

(* 计算矩阵 H *)
matH = Outer[Times, pointsY, pointsY]* Outer[polynomialKernal, pointsX, pointsX, 1];

(* 设定惩罚系数 *)
c = 10000;

(* 求解模型参数 *)
matA = Join[IdentityMatrix[pop], -IdentityMatrix[pop]];
vecB = Join[Table[0, pop], Table[c, pop]];
vecC = Table[1, pop];

(* 设自变量 t *)
(* 最小化 1/2 t^T . H . t - sum_{i=1}^N t_i *)
(* 约束条件: sum_{i=1}^N t_i y_i == 0 *)
(* 约束条件: 0 <= t_i <= C *)
sol = QuadraticOptimization[
    {matH, -vecC}, {matA, vecB},
    PerformanceGoal -> "Quality"
];

(* 求支持向量下标 *)
lambda[i_] := sol[[i]]
setS = Select[Range[1, pop], lambda[#] > 10^-2 &]

(* 定义判别辅助函数 *)
yi[m_] := pointsY[[m]]
xi[m_] := pointsX[[m]]
g[x_] := Sum[lambda[m]*yi[m]*polynomialKernal[xi[m], x], {m, setS}]

(* 运行回测 *)
predictVal = Sign[g[xi[#]]] & /@ Range[1, pop];

(* 查看回测结果 *)
Transpose[{predictVal, pointsY}] // Counts

(* 查看预测值为 -1，实际值也为 -1 的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, -1}]

(* 查看预测值为 1，实际值也为 1 的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, 1}]

```

```
(* 查看预测值为 -1，实际值却为1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, 1}]

(* 查看预测值为1，实际值却为 -1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, -1}]

(* 查看支持向量 *)
iris[[setS]]

(* 挑选出只含有versicolor和virginica的子集 *)
irisSubset = iris[Select[#Species == "versicolor" || #Species == "virginica" &]]

(* 整理数据成数值形式 *)
pointsX = {
    irisSubset[All, 2] // Normal // QuantityMagnitude,
    irisSubset[All, 3] // Normal // QuantityMagnitude,
    irisSubset[All, 4] // Normal // QuantityMagnitude,
    irisSubset[All, 5] // Normal // QuantityMagnitude
} // Transpose;

(* 给标签赋值 *)
pointsY = (irisSubset[All, 1] // Normal) /. {"versicolor" -> -1, "virginica" -> 1};

(* 计算样本总量 *)
pop = Length[pointsX]

(* 计算矩阵 H *)
matH = Outer[Times, pointsY, pointsY]* Outer[polynomialKernal, pointsX, pointsX, 1];

(* 设定惩罚系数 *)
c = 10000;

(* 求解模型参数 *)
matA = Join[IdentityMatrix[pop], -IdentityMatrix[pop]];
vecB = Join[Table[0, pop], Table[c, pop]];
vecC = Table[1, pop];

(* 设自变量 t *)
(* 最小化 1/2 t^T . H . t - sum_{i=1}^N t_i *)
(* 约束条件: sum_{i=1}^N t_i y_i == 0 *)
(* 约束条件: 0 <= t_i <= C *)
sol = QuadraticOptimization[
    {matH, -vecC}, {matA, vecB},
    PerformanceGoal -> "Quality"
];

(* 求支持向量下标 *)
lambda[i_] := sol[[i]]
setS = Select[Range[1, pop], lambda[#] > 10^-2 &]

(* 定义判别辅助函数 *)
yi[m_] := pointsY[[m]]
xi[m_] := pointsX[[m]]
g[x_] := Sum[lambda[m]*yi[m]*polynomialKernal[xi[m], x], {m, setS}]

(* 运行回测 *)
predictVal = Sign[g[xi[#]]] & /@ Range[1, pop];

(* 查看回测结果 *)
Transpose[{predictVal, pointsY}] // Counts

(* 查看预测值为 -1，实际值也为 -1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, -1}]

(* 查看预测值为1，实际值也为1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, 1}]

(* 查看预测值为 -1，实际值却为1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{-1, 1}]

```

```
(* 查看预测值为1，实际值却为-1的频数 *)
(Transpose[{predictVal, pointsY}] // Counts)[{1, -1}]

(* 查看支持向量 *)
irisSubset[[setS]]
```

B.2 在 MNIST 上跑

以下代码是用 R 写的，需要在 R 软件上运行．运行它需要提前安装 e1071 包．

```
library("e1071");

# 读入图片数据，每一行对应一张图片的像素值向量，每一行是784个值
# 相当于原来28 * 28像素矩阵一行一行地写成一行
x=read.delim("imgData1.csv", header=FALSE, sep=",")

# 转换成矩阵形式
mx = matrix(as.numeric(as.matrix(x)), nrow=nrow(x), ncol=ncol(x));

# 试打印第一幅图片
image(matrix(mx[1,], nrow=28, ncol=28))

# 读入标签值向量，每一行对应一张图片，每一行只有一个值，取值只能是0或1
y = read.delim("imgLabels.csv", header=FALSE, sep=",")

# 转换为矩阵形式
my=matrix(as.numeric(as.matrix(y)), nrow=nrow(y), ncol=ncol(y))

# 将0替换为-1，给SVM提供方便
my[my[,1] == 0, 1] = -1;

critical = 0.5
# 将图片模糊的地方绝对化
for (rowNum in c(1:nrow(mx))) {
    mx[rowNum,mx[rowNum,] < critical]=0;
    mx[rowNum,mx[rowNum,] >= critical]=1;
}

# 试打印第一幅图片，看看效果如何
image(matrix(mx[1,], nrow=28, ncol=28))

# sampleSize是训练所需的样本量
# 可以自己设置，取值范围：1到nrow(mx)，这里我们拿全部样本做训练集
sampleSize = nrow(mx)
selectedRows=sample(c(1:nrow(mx)), sampleSize)

# 去掉图片公共空白区域
mxSubset = mx[selectedRows, ]
mySubset = my[selectedRows, ]
usefulCols= c(1:ncol(mxSubset))[
    sapply(
        1:ncol(mxSubset),
        function(j) {
            min(mxSubset[,j])<1
        }
    )
]

# 开始训练SVM模型，核函数选用RBF，参数由软件自己确定
model1 = svm(
    mxSubset[, usefulCols],
    mySubset,
    type="C-classification"
)

# 求出混淆矩阵
result = table(model1$fitted, mySubset)

# 计算正确率
sum(diag(result))/sum(sum(result))

# 开始进行k折交叉验证，设定k值
kNum = 10

# 求出每一份的样本容量
sizePerPiece = ceiling(nrow(mx)/kNum)

```

```
# 打乱序号
indexes = sample(c(1:nrow(mx)), nrow(mx))

# 求出每一份测试集的样本序号集
testSetIndexes = list();
for (i in c(1:(kNum-1))) {
    testSetIndexes[[length(testSetIndexes)+1]] = indexes[c(((i-1)*sizePerPiece+1):(i*sizePerPiece))];
}
testSetIndexes[[length(testSetIndexes)+1]] = indexes[c(((kNum-1)*sizePerPiece):nrow(mx))];

# 用来记录模型
models = list();

# 用来记录混淆矩阵
results = list();

# 运行k折交叉验证
for (i in 1:kNum) {
    testSetX = mx[testSetIndexes[[i]], ];
    testSetY = my[testSetIndexes[[i]], ];

    trainingSetX = mx[-testSetIndexes[[i]], ];
    trainingSetY = my[-testSetIndexes[[i]], ];

    usefulCols = c(1:ncol(mx))[
        sapply(
            1:ncol(mx),
            function(j) {
                min(trainingSetX[,j])<1
            }
        )
    ];

    model = svm(trainingSetX[, usefulCols], trainingSetY, type = "C-classification");
    models[[length(models)+1]] = model;

    predictedVal = predict(model, testSetX[, usefulCols]);
    results[[length(results)+1]] = table(predictedVal, testSetY);
}

# 计算每一折的错误率
errors = sapply(
    results,
    function (result) {
        sum(diag(result))/sum(sum(result))
    }
);

# 查看错误率
errors

# 计算平均错误率
mean(errors)

# 记录每一折模型用到的参数
gammas = lapply(
    models,
    function (model) {
        model$gamma
    }
);

# 查看参数
gammas

# 查看每一个模型
for (model in models) {
    print(summary(model));
}
```