# LIBSVM：一种支持向量机的程序库

张治中和林智仁*

原文最后更新于：2009 年 2 月 27 号

**摘要**

LIBSVM 是一种支持向量机（SVM）的程序库。它的目标是帮助用户以便容易地使用 SVM 来充当一种工具。在这篇文章中，我们提供了所有它的实施细节。而对于 LIBSVM 的使用，软件包所内含的 README 文件和在线的 LIBSVM FAQ 则提供了相应的信息。

## 1 介绍

LIBSVM 是一种支持向量机（SVM）的程序库。它的目标是使用户能够容易地使用 SVM 来充当一种工具。在这篇文章中，我们提供了所有它的实现细节。而对于 LIBSVM 的使用，软件包所内含的 README 文件则提供了相应的信息。

在第二部分，我们将展示在 LIBSVM 中所使用到的公式：$C$-支持向量分类（$C$-SVC），$\nu$-支持向量分类（$\nu$-SVC），分布估计（one-class SVM），$\epsilon$-支持向量回归（$\epsilon$-SVR）和 $\nu$-支持向量回归（$\nu$-SVR）。在第三部分，我们将讨论对于解决二次问题的实现细节。在第四部分，我们将阐述两种实现技术：收缩和缓存。而对于不平衡的数据，我们也支持不同的惩罚系数，详细细节将在第五部分中阐述。然后在第六部分，我们将讨论一下对于多分类的实现。参数选择对于获得好的 SVM 模型是至关重要的，对于这一点，LIBSVM 相应地提供了一些简单而且实用的工具，这些将在第七部分中讨论。在第八部分中，我们将展示概率输出的实现。

## 2 公式

### 2.1 $C$-支持向量分类

假设给定训练向量组 $x_i \in R^n, i = 1, \cdots, l$ 且分属于两类，和一个向量 $y \in R^l$ 使得满足 $y_i \in \{1, -1\}$，那么 $C$-SVC 就是来解决如下原始问题的：

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \textit{subject to} \quad & y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi, \\ & \xi_i \geq 0, i = 1, \cdots, l. \end{aligned} \tag{2.1}$$

它的对偶问题就是：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \textit{subject to} \quad & \mathbf{y}^T\boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_i \leq C, i = 1, \cdots, l. \end{aligned} \tag{2.2}$$

其中，$\mathbf{e}$ 是完全由 1 组成的向量，$C \geq 0$ 是上界，$Q$ 是 $l \times l$ 的正半定矩阵，$Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$，且 $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ 是核。在这里，训练向量 $\mathbf{x}_i$ 通过函数 $\phi$ 被映射到一个更高的（可能是无穷的）维空间中。

这里的决策函数是：

$$sgn\left(\sum_{i=1}^{l} y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right).$$

*林智仁,国立台湾大学资讯系教授（http://www.csie.ntu.edu.tw/~cjlin） E-mail:cjlin@csie.ntu.edu.tw

## 2.2 $\nu$-支持向量分类

这种 $\nu$-支持向量分类将使用一个新的参数 $\nu$，这个参数是用来控制支持向量和训练误差的数目的。这个参数 $\nu \in (0,1]$ 既是训练误差的分数的上界，同时又是支持向量的分数的下界。

假设给定训练向量组 $\mathbf{x}_i \in R^n, i = 1, \cdots, l$ 且分属于两类，和一个向量 $y \in R^l$ 使得满足 $y_i \in \{1, -1\}$，那么我们所考虑的问题的原始形式就是：

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \nu\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ subject\ to \quad & y_i(\mathbf{w}^T\phi(x_i) + b) \geq \rho - \xi, \\ & \xi_i \geq 0, i = 1, \cdots, l.\rho \geq 0 \end{aligned}$$

它的对偶问题就是：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ subject\ to \quad & 0 \leq \alpha_i \leq 1/l, i = 1, \cdots, l, \\ & \mathbf{e}^T\boldsymbol{\alpha} \geq \nu, \\ & \mathbf{y}^T\boldsymbol{\alpha} = 0. \end{aligned} \tag{2.3}$$

其中 $Q_{ij} \equiv y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$。

这里的决策函数是:

$$sgn\left(\sum_{i=1}^{l} y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right).$$

在以下（[5];[2]）两篇文章中，已经向我们展示了如下性质：$\mathbf{e}^T\boldsymbol{\alpha} \geq \nu$ 可以 $\mathbf{e}^T\boldsymbol{\alpha} = \nu$ 被所替换。所以在 LIBSVM 中，根据这个性质，我们解决了（2.3）缩放后的一个问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ subject\ to \quad & 0 \leq \alpha_i \leq 1, i = 1, \cdots, l, \\ & \mathbf{e}^T\boldsymbol{\alpha} = \nu l, \\ & \mathbf{y}^T\boldsymbol{\alpha} = 0. \end{aligned}$$

我们将输出 $\boldsymbol{\alpha}/\rho$，所以被计算出的决策函数将是

$$sgn\left(\sum_{i=1}^{l} y_i(\alpha_i/\rho)(K(\mathbf{x}_i, \mathbf{x}) + b)\right)$$

而且它的两个边界是

$$y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) = \pm 1$$

这两个边界和 $C$-SVC 的是完全一样的。

## 2.3 分布估计（one-class SVM）

One-class SVM 是在[19]一文中所提出的，用来评估一个高维分布的支持性。假设给定训练向量组 $\mathbf{x}_i \in R^n, i = 1, \cdots, l$ 且没有任何类别信息，那么在[19]一文中的问题的原始形式就是：

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} - \rho + \frac{1}{\nu l}\sum_{i=1}^{l}\xi_i \\ subject\ to \quad & \mathbf{w}^T\phi(\boldsymbol{x}_i) \geq \rho - \xi_i, \\ & \xi_i \geq 0, i = 1, \cdots, l. \end{aligned}$$

它的对偶问题就是：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ subject\ to \quad & 0 \le \alpha_i \le 1/(\nu l), i = 1, \cdots, l, \\ & \mathbf{e}^T\boldsymbol{\alpha} = 1. \end{aligned} \tag{2.4}$$

其中 $Q_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$。

在 LIBSVM 中，我们解决了（2.4）缩放后的一个问题：

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ subject\ to \quad & 0 \le \alpha_i \le 1, i = 1, \cdots, l, \\ & \mathbf{e}^T\boldsymbol{\alpha} = \nu l. \end{aligned} \tag{2.5}$$

这里的决策函数是：

$$sgn\left(\sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho\right).$$

### 2.4 $\epsilon$-支持向量回归（$\epsilon$-SVR）

假设给定一个数据点集，$\{(\mathbf{x}_1, z_1), \cdots, (\mathbf{x}_l, z_l)\}$，使之满足 $x_i \in R^n$ 是一个输入且 $z_i \in R_1$ 是对应的一个目标输出，那么支持向量回归问题的标准形式就是：

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i + C\sum_{i=1}^{l}\xi_i^* \\ subject\ to \quad & \mathbf{w}^T\phi(\boldsymbol{x}_i) + b - z_i \le \epsilon + \xi_i, \\ & z_i - \mathbf{w}^T\phi(\boldsymbol{x}_i) - b \le \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \ge 0, i = 1, \cdots, l. \end{aligned}$$

它的对偶问题就是：

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} z_i(\alpha_i - \alpha_i^*) \\ subject\ to \quad & \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0, 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \cdots, l. \end{aligned} \tag{2.6}$$

其中 $Q_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$。

它的近似函数是：

$$\sum_{i=1}^{l}(-\alpha_i + \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b.$$

在 LIBSVM 实现中，我们将 $\boldsymbol{\alpha}$ 和 $\boldsymbol{\alpha}^*$ 一起储存在一个数组中。注意:$\boldsymbol{\alpha}^*$ 是排在前面的，所以代码中的联合数组事实上是 $[\boldsymbol{\alpha}^*\ \boldsymbol{\alpha}]^T$。

### 2.5 $\nu$-支持向量回归（$\nu$-SVR）

对于回归而言，其实和 $\nu$-SVC 是很相似的，[18]一文中使用了一个参数 $\nu$ 来控制支持向量的数量。但是，不像 $\nu$-SVC 中，$\nu$ 可以用 $C$ 代换，这里的 $\nu$ 则是可以用 $\epsilon$-SVR 中的参数 $\epsilon$ 来代

换的。问题的原始形式是：

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*,\epsilon} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\nu\epsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)) \\ subject\ to \quad & (\mathbf{w}^T\phi(\mathbf{x_i}) + b) - z_i \le \epsilon + \xi_i^*, \\ & z_i - (\mathbf{w}^T\phi(\mathbf{x}_i) + b) \le \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \ge 0, i = 1, \cdots, l, \epsilon \ge 0. \end{aligned} \tag{2.7}$$

它的对偶问题是：

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \mathbf{z}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \\ subject\ to \quad & \mathbf{e}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0, \mathbf{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) \le C\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C/l, i = 1, \cdots, l. \end{aligned} \tag{2.8}$$

相似的，不等式 $\mathbf{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) \le C\nu$ 能够被一个等式所替换。在 LIBSVM 中，我们认为 $C \leftarrow C/l$，那么所解决的对偶问题就是：

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \mathbf{z}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \\ subject\ to \quad & \mathbf{e}^T(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0, \mathbf{e}^T(\boldsymbol{\alpha} + \boldsymbol{\alpha}^*) = Cl\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \cdots, l. \end{aligned} \tag{2.9}$$

这里的决策函数就是：

$$\sum_{i=1}^{l}(-\alpha_i + \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b$$

这一决策函数和 $\epsilon$-SVR 中的是完全一样的。

## 2.6 效果测量

在通过解决以上优化问题来训练出模型后，用户可以应用 LIBSVM 来预测测试数据的标值（目标值）。让 $x_1, \cdots, x_{\bar{l}}$ 成为测试数据且让 $f(x_1), \cdots, f(x_{\bar{l}})$ 成为 LIBSVM 的被预测的决策值（对回归而言就是目标值）。如果测试数据的正确的标值（目标值）实现已知道且被记为 $y_i, \cdots, y_{\bar{l}}$，那么我们可以通过如下方法来评估预测性：

### 2.6.1 分类

$$\begin{aligned} & Accuracy \\ = \quad & \frac{\#\,correctly\ predicted\ data}{\#\,total\ data} \times 100\% \\ = \quad & \frac{|\{i \,|\, y_i f(x_i) > 0\}|}{\bar{l}} \times 100\%. \end{aligned}$$

### 2.6.2 回归

LIBSVM 输出 MSE（均方差）和 $r^2$（相关系数的平方）：

$$MSE = \frac{1}{\bar{l}}\sum_{i=1}^{\bar{l}}(f(x_i) - y_i)^2,$$

$$r^2 = \frac{\left(\bar{l}\sum_{i=1}^{\bar{l}} f(x_i)y_i - \sum_{i=1}^{\bar{l}} f(x_i)\sum_{i=1}^{\bar{l}} y_i\right)^2}{\left(\bar{l}\sum_{i=1}^{\bar{l}} f(x_i)^2 - (\sum_{i=1}^{\bar{l}} f(x_i))^2\right)\left(\bar{l}\sum_{i=1}^{l} y_i^2 - (\sum_{i=1}^{\bar{l}} y_i)^2\right)}$$

# 3 解决二次问题

## 3.1 对于 $C$-SVC，$\epsilon$-SVR 和 one-class SVM 的分解法

我们考虑 $C$-SVC，$\epsilon$-SVR 和 one-class SVM 的如下通式：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{p}^T\boldsymbol{\alpha} \\ \textit{subject to} \quad & \mathbf{y}^T\boldsymbol{\alpha} = \Delta, \\ & 0 \le \alpha_t \le C, t = 1, \cdots, l. \end{aligned} \tag{3.1}$$

其中 $y_t = \pm 1, t = 1, \cdots, l$。我们可以清晰地看出，$C$-SVC 和 one-class SVM 就已经是（3.1）式的形式了。而对于 $\epsilon$-SVR，我们考虑（2.6）式的如下重组形式：

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}\left[\boldsymbol{\alpha}^T, (\boldsymbol{\alpha}^*)^T\right]\begin{bmatrix} Q & -Q \\ -Q & Q \end{bmatrix}\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} + \left[\epsilon\mathbf{e}^T + \mathbf{z}^T, \epsilon\mathbf{e}^T - \mathbf{z}^T\right]\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} \\ \textit{subject to} \quad & \mathbf{y}^T\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} = 0, 0 \le \alpha_t, \alpha_t^* \le C, t = 1, \cdots, l. \end{aligned} \tag{3.2}$$

其中 $y$ 是一个 $2l \times l$ 向量，满足 $y_t = 1, t = 1, \cdots, l$ 和 $y_t = -1, t = l+1, \cdots, 2l$。

解决（3.1）式的难点在于矩阵 $Q$ 的密度，因为一般 $Q_{ij}$ 非零。在 LIBSVM 中，我们考虑分解法来克服这个困难。例如，在[14];[9];[17]中可以看到关于这一方法的一些工作。这一方法在每一次迭代中仅仅修改 $\boldsymbol{\alpha}$ 的一个子集。我们把这一子集标记为工作集 $B$。而这一子集在每次迭代中将导致一个小的子问题被最小化。一个极端的例子就是序列最小化优化（SMO）[17]，这一算法限制工作集 $B$ 只含有两个元素。然后在每次迭代中各自解决一个简单的两变量的问题且无需优化软件。这里我们考虑在[6]中被提出的一个 SMO 型的分解法。

**算法 1** *(一种 SMO 型的分解法，[6])*

1. 找到 $\boldsymbol{\alpha}^1$ 作为初始可行解。置 k = 1。
2. 如果 $\boldsymbol{\alpha}^k$ 是（2.2）式的一个固定的点，那么算法终止。否则，通过 WSS 1（将在 3.2 部分中讲到）找到一个两元素的工作集 $B = \{i, j\}$。定义 $N \equiv \{1, \cdots, l\} \backslash B$，且定义 $\boldsymbol{\alpha}_B^k$ 和 $\boldsymbol{\alpha}_N^k$ 来作为 $\boldsymbol{\alpha}^k$ 的子向量，分别对应于 $B$ 和 $N$。
3. 如果 $a_{ij} \equiv K_{ii} + K_{jj} - 2K_{ij} > 0$，那么用变量 $\boldsymbol{\alpha}_B$ 来处理如下的子问题：

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix}\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + \left(\mathbf{p}_B + Q_{BN}\boldsymbol{\alpha}_N^k\right)^T\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} \\ \textit{subject to} \quad & 0 \le \alpha_i, \alpha_j \le C, \\ & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^k. \end{aligned} \tag{3.3}$$

否则，则处理：

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix}\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} + \left(\mathbf{p}_B + Q_{BN}\boldsymbol{\alpha}_N^k\right)^T\begin{bmatrix} \alpha_i \\ \alpha_j \end{bmatrix} \\ & + \frac{\tau - a_{ij}}{4}((\alpha_i - \alpha_i^k)^2 + (\alpha_j - \alpha_j^k)^2) \\ \textit{subject to} \quad & 0 \le \alpha_i, \alpha_j \le C, \\ & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^k. \end{aligned} \tag{3.4}$$

4. $\boldsymbol{\alpha}_B^{k+1}$ 置 $\boldsymbol{\alpha}_B^{k+1} \equiv \boldsymbol{\alpha}_B^k$ 为（3.3）式的最优解，且置 $k \leftarrow k+1$。置且跳回到步骤 2。

注意：$B$ 在每次迭代中都会被更新。为了简化符号，我们简单地使用 $B$ 而不是 $B^k$。如果 $a_{ij} \le 0$，那么（3.3）式就是一个凹问题。因此我们在（3.4）式中使用一个凸修改。

### 3.2 对于 $C$-SVC，$\epsilon$-SVR 和 one-class SVM 的终止标准和工作集的选择

（3.1）式的 Karush-Kuhn-Tucker（KKT）最优化条件表明，向量 $\alpha$ 是（3.1）式的一个固定点，当且仅当存在一个数 $b$ 和两个非负的向量 $\lambda$ 和 $\mu$ 满足

$$\nabla f(\boldsymbol{\alpha}) + b\mathbf{y} = \boldsymbol{\lambda} - \boldsymbol{\mu},$$
$$\lambda_i \alpha_i = 0, \mu_i(C - \alpha_i) = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1, \cdots, l.$$

其中 $\nabla f(\boldsymbol{\alpha}) \equiv Q\boldsymbol{\alpha} + \mathbf{p}$ 是 $f(\boldsymbol{\alpha})$ 的梯度。这一条件可以被重写为

$$\nabla f(\boldsymbol{\alpha})_i + by_i \geq 0 \quad \text{if } \alpha_i < C, \tag{3.5}$$
$$\nabla f(\boldsymbol{\alpha})_i + by_i \leq 0 \quad \text{if } \alpha_i > 0. \tag{3.6}$$

因为根据如下定义，我们可以得到 $y_i = \pm 1$

$$I_{up}(\boldsymbol{\alpha}) \equiv \{t \,|\, \alpha_t < C, y_t = 1 \, or \, \alpha_t > 0, y_t = -1\}, and$$
$$I_{low}(\boldsymbol{\alpha}) \equiv \{t \,|\, \alpha_t < C, y_t = -1 \, or \, \alpha_t > 0, y_t = 1\} \tag{3.7}$$

一个可行的 $\boldsymbol{\alpha}$ 是（3.1）式的一个固定的点，当且仅当

$$m(\boldsymbol{\alpha}) \leq M(\boldsymbol{\alpha}) \tag{3.8}$$

其中

$$m(\boldsymbol{\alpha}) \equiv \max_{i \in I_{up}(\boldsymbol{\alpha})} -y_i \nabla f(\boldsymbol{\alpha})_i, and \; M(\boldsymbol{\alpha}) \equiv \min_{i \in I_{low}(\boldsymbol{\alpha})} -y_i \nabla f(\boldsymbol{\alpha})_i.$$

根据这个，我们可以得到如下的终止条件：

$$m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) \leq \epsilon. \tag{3.9}$$

而关于工作集 B 的选择，我们考虑如下的程序流程：

**WSS 1**

1. 对于所有的 $t$，$s$，定义

$$a_{ts} \equiv K_{tt} + K_{ss} - 2K_{ts}, b_{ts} \equiv -y_t \nabla f(\boldsymbol{\alpha}^k)_t + y_s \nabla f(\boldsymbol{\alpha}^k)_s > 0 \tag{3.10}$$

和

$$\bar{a}_{ts} \equiv \begin{cases} a_{ts}, & \text{if } a_{ts} > 0, \\ \tau, & \text{otherwise.} \end{cases} \tag{3.11}$$

选择

$$i \in arg \max_t \{-y_t \nabla f(\boldsymbol{\alpha}^k)_t \,|\, t \in I_{up}(\boldsymbol{\alpha}^k)\},$$
$$j \in arg \min_t \left\{ -\frac{b_{it}^2}{\bar{a}_{it}} \,|\, t \in I_{low}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_i \nabla f(\boldsymbol{\alpha}^k)_i \right\}. \tag{3.12}$$

2. 返回 $B = \{i, j\}$。

关于我们如何决定这个工作集的细节主要在([6],Section II）中。

### 3.3 分解法的收敛性

对于算法 1 的收敛性的详细讨论请参看（[6],Section III）或者（[3]）。

### 3.4 对于 $\nu$-SVC 和 $\nu$-SVR 的分解法

$\nu$-SVC 和 $\nu$-SVR 两者都可以被考虑为如下的通式：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{p}^T\boldsymbol{\alpha} \\ subject\ to \quad & \mathbf{y}^T\boldsymbol{\alpha} = \Delta_1, \\ & \mathbf{e}^T\boldsymbol{\alpha} = \Delta_2, \\ & 0 \le \alpha_t \le C, t = 1, \cdots, l. \end{aligned} \tag{3.13}$$

（3.13）的 KKT 条件表明

$$\begin{aligned} \nabla f(\boldsymbol{\alpha})_i - \rho + by_i = 0 \quad & \text{if } 0 < \alpha_i < C, \\ \ge 0 \quad & \text{if } \alpha_i = 0, \\ \le 0 \quad & \text{if } \alpha_i = C. \end{aligned}$$

定义

$$r_1 \equiv \rho - b, r_2 \equiv \rho + b.$$

如果 $y_i = 1$，那么 KKT 条件变成

$$\begin{aligned} \nabla f(\boldsymbol{\alpha})_i - r_1 \ge 0 \quad & \text{if } \alpha_i < C, \\ \le 0 \quad & \text{if } \alpha_i > 0. \end{aligned} \tag{3.14}$$

另一方面，如果 $y_i = -1$，那么条件变成

$$\begin{aligned} \nabla f(\boldsymbol{\alpha})_i - r_2 \ge 0 \quad & \text{if } \alpha_i < C, \\ \le 0 \quad & \text{if } \alpha_i > 0. \end{aligned} \tag{3.15}$$

因此假设给定一个容差 $\epsilon$> 0，那么终止条件将是：

$$max(m_p(\boldsymbol{\alpha}) - M_p(\boldsymbol{\alpha}), m_n(\boldsymbol{\alpha}) - M_n(\boldsymbol{\alpha})) < \epsilon \tag{3.16}$$

其中

$$\begin{aligned} m_p(\boldsymbol{\alpha}) &\equiv \max\nolimits_{i \in I_{up}(\boldsymbol{\alpha}), y_i = 1} -y_i \nabla f(\boldsymbol{\alpha})_i, \quad M_p(\boldsymbol{\alpha}) \equiv \min_{i \in I_{low}(\boldsymbol{\alpha}), y_i = 1} -y_i \nabla f(\boldsymbol{\alpha})_i, and \\ m_n(\boldsymbol{\alpha}) &\equiv \max\nolimits_{i \in I_{up}(\boldsymbol{\alpha}), y_i = -1} -y_i \nabla f(\boldsymbol{\alpha})_i, \quad M_n(\boldsymbol{\alpha}) \equiv \min_{i \in I_{low}(\boldsymbol{\alpha}), y_i = 1} -y_i \nabla f(\boldsymbol{\alpha})_i. \end{aligned}$$

对于工作集的选择，我们可以通过扩展 WSS 1 到如下程序流程来实现

**WSS 2**（扩展 *WSS 1* 到 *$\nu$-SVM*）

1. 找到

$$\begin{aligned} & i_p \in arg\, m_p(\boldsymbol{\alpha}^k), \\ & j_p \in arg \min_t \left\{ -\frac{b_{i_p t}^2}{\bar{a}_{i_p t}} \,|\, y_t = 1, \boldsymbol{\alpha}_t \in I_{low}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_{i_p} \nabla f(\boldsymbol{\alpha}^k)_{i_p} \right\} \end{aligned}$$

2. 找到

$$\begin{aligned} & i_n \in arg\, m_n(\boldsymbol{\alpha}^k), \\ & j_n \in arg \min_t \left\{ -\frac{b_{i_n t}^2}{\bar{a}_{i_n t}} \,|\, y_t = 1, \boldsymbol{\alpha}_t \in I_{low}(\boldsymbol{\alpha}^k), -y_t \nabla f(\boldsymbol{\alpha}^k)_t < -y_{i_n} \nabla f(\boldsymbol{\alpha}^k)_{i_n} \right\} \end{aligned}$$

3. 3. 根据哪一个给出更小的 $-b_{ij}^2/\bar{a}_{ij}$ 来相应返回 $\{i_p, j_p\}$ 或 $\{i_n, j_n\}$。

### 3.5 解析解

具体细节将在第 5 部分中被阐述，而在第 5 部分中，我们将讨论一种更加普通的子问题的解。

### 3.6 $b$ 或 $\rho$ 的计算

在对偶最优化问题的解 $\boldsymbol{\alpha}$ 获得以后，变量 $b$ 或 $\rho$ 必须被计算出来，因为在决策函数中将被用到。

我们首先分析一下 $C$-SVC，$\epsilon$-SVR 和 one-class SVM 的情况。对于这一情况，$b$ 在 one-class SVM 中的作用和 $-\rho$ 一样，所以我们可以定义 $rho = -b$ 并且讨论如何去找到它。如果存在 $\alpha_i$ 满足 $0 < \alpha_i < C$，那么从（3.8）式的 KKT 条件，我们可以得知 $\rho = y_i \nabla f(\boldsymbol{\alpha})_i$。实际上为了避免数值误差，我们取它们的平均值：

$$\rho = \frac{\sum_{0<\alpha_i<C} y_i \nabla f(\boldsymbol{\alpha})_i}{\sum_{0<\alpha_i<C} 1}.$$

另一方面，如果不存在那样的 $\alpha_i$，那么 KKT 条件将变为

$$\begin{aligned} & -M(\boldsymbol{\alpha}) = max\{y_i \nabla f(\boldsymbol{\alpha})_i \,|\, \alpha_i = 0, y_i = 1 \, or \, \alpha_i = C, y_i = 1\} \\ \leq \quad & \rho \\ \leq \quad & -m(\alpha) = min y_i \nabla f(\boldsymbol{\alpha})_i \,|\, \alpha_i = 0, y_i = 1 \, or \, \alpha_i = C, y_i = -1. \end{aligned}$$

那么我们把这一范围的中点取为 $\rho$ 值。

对于 $\nu$-SVC 和 $\nu$-SVR 的情况，$b$ 和 $\rho$ 同时出现。（3.13）式的 KKT 条件已经在（3.14）式和（3.15）式中被展示了。那么现在我们考虑 $y_i = 1$ 的情况。如果存在 $\alpha_i$ 满足 $0 < \alpha_i < C$，那么 $r_1 = \nabla f(\alpha)_i$。实际上为了避免数值误差，我们取它们的平均值：

$$r_1 = \frac{\sum_{0<\alpha_i<C, y_i=1} \nabla f(\boldsymbol{\alpha})_i}{\sum_{0<\alpha_i<C, y_i=1} 1}.$$

另一方面，如果不存在那样的 $\alpha_i$，且因为 $r_1$ 必须满足

$$\max_{\alpha_i=C, y_i=1} \nabla f(\boldsymbol{\alpha})_i \leq r_1 \leq \min_{\alpha_i=0, y_i=1} \nabla f(\boldsymbol{\alpha})_i$$

那么我们把这一范围的中点取为 $r_1$ 的值。

而对于 $y_i = -1$ 的情况，我们用相似的方法计算出 $r_2$。

在 $r_1$ 和 $r_2$ 都获得后，

$$\rho = \frac{r_1 + r_2}{2} \; and \; -b = \frac{r_1 - r_2}{2}.$$

### 3.7 初始值

在算法 1 的开始，我们必须给出一个可行的 $\boldsymbol{\alpha}$。对于 $C$-SVC 和 $\epsilon$-SVR，初始的 $\boldsymbol{\alpha}$ 就是简单的 0。而对于 $\nu$-SVC，在我们对（2.5）式所进行的缩放后，

$$\alpha_i \leq 1 \; and \; \sum_{i:y_i=1} \alpha_i = \frac{\nu l}{2}.$$

因此我们把 $y_i = 1$ 时 $\alpha_i$ 的第一个 $\frac{\nu l}{2}$ 元素的值取为 1。对于 $y_i = -1$ 的情况也是相似处理。

对于 one-class SVM 和 $\nu$-SVR，我们采用相同的设置方法。

# 4 收缩和缓存

## 4.1 收缩

因为对于很多问题，自由的支持向量（例如，$0 < \alpha_i < C$）的数量是少的，所以收缩技术可以来缩小工作问题的规模而无需考虑一些有界变量。在临近迭代过程的最后，分解法将辨识出一个可能的集合 $A$，而且可能所有最后的自由的 $\alpha_i$ 都存在于这个集合中。事实上我们可以得出以下的定理，并且这个定理向我们展示了在第 3.2 部分中所提出的分解法的最后一步迭代中，仅仅只有对应于小部分集合的变量仍然是被允许移动的：

**定理 4.1**（*[6]*中的定理 *IV*）假定 $Q$ 是正的半正定矩阵。

1. 以下集合独立于任何最优解 $\bar{\boldsymbol{\alpha}}$：

$$I \equiv \{i \mid -y_i \nabla f(\bar{\boldsymbol{\alpha}})_i > M(\bar{\boldsymbol{\alpha}}) \, or \, -y_i \nabla f(\bar{\boldsymbol{\alpha}})_i < m(\bar{\boldsymbol{\alpha}})\}. \tag{4.1}$$

   问题（2.2）将在 $\alpha_i, i \in I$ 处得到唯一且有界的最优解。

2. 假定算法 1 将产生一个无穷序列 $\{\boldsymbol{\alpha}^k\}$。且存在 $\bar{k}$ 满足 $k \geq \bar{k}$ 时，每一个 $\alpha_i^k, i \in I$ 都能取得唯一且有界的最优解。这一点在所有随后的迭代和 $\forall k \geq \bar{k}$ 时仍然都成立：

$$i \notin \{t \mid M(\boldsymbol{\alpha}^k) \leq -y_t \nabla f(\boldsymbol{\alpha}^k)_t \leq m(\boldsymbol{\alpha}^k)\}. \tag{4.2}$$

因此，我们不需要使用分解法来处理整个（2.2）问题，而是来处理一个更小的问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T Q_{AA} \boldsymbol{\alpha}_A - (\mathbf{p}_A - Q_{AN}\boldsymbol{\alpha}_N^k)^T \boldsymbol{\alpha}_A \\ subject\ to \quad & 0 \leq (\boldsymbol{\alpha}_A)_t \leq C, t = 1, \cdots, q, \\ & \mathbf{y}_A^T \boldsymbol{\alpha}_A = \Delta - \mathbf{y}_N^T \boldsymbol{\alpha}_N^k. \end{aligned} \tag{4.3}$$

其中，$N = \{1, \cdots, l\} \backslash A$ 是被收缩后的变量的集合。

当然，如果一些元素被错误地收缩，那么这种启发式方法可能会失败。当这种情况发生时，这整个问题（2.2）将从起点 $\boldsymbol{\alpha}$ 被重新优化，其中 $\boldsymbol{\alpha}_A$ 是（4.3）问题的最优解，而 $\boldsymbol{\alpha}_N$ 则对应于收缩后的有界变量。注意，当处理收缩后的问题（4.3）时，我们仅仅知道（4.3）式的梯度 $Q_{AA}\boldsymbol{\alpha}_A + Q_{AN}\boldsymbol{\alpha}_N + \mathbf{p}_A$。因此，当问题（2.2）被重新优化时，我们必须重新构造在第 4.3 部分中被讨论的这整个梯度 $\nabla f(\boldsymbol{\alpha})$。

一些 SVM 实现程序是要在临近迭代过程的最后时才开始收缩过程，但在 LIBSVM 中，我们是从一开始就开始了的。这一算法流程是按照如下进行的：

1. 在每 $min\{l, 1000\}$ 次迭代后，我们将尝试收缩一些变量。注意，在迭代过程期间

$$m(\boldsymbol{\alpha}^k) > M(\boldsymbol{\alpha}^k) \tag{4.4}$$

   虽然（3.9）式并不满足这一式子。根据定理 4.1，我们推测以下集合中的变量能够被收缩：

$$\begin{aligned} & \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m(\boldsymbol{\alpha}^k), t \in I_{low}(\boldsymbol{\alpha}^k), \alpha_t^k \, is \, bounded\} \cup \\ & \quad \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M(\boldsymbol{\alpha}^k), t \in I_{up}(\boldsymbol{\alpha}^k), \alpha_t^k \, is \, bounded\} \\ = & \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m(\boldsymbol{\alpha}^k), \alpha_t^k = C, y_t = 1 \, or \, \alpha_t^k = 0, y_t = -1\} \cup \\ & \quad \{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M(\boldsymbol{\alpha}^k), \alpha_t^k = 0, y_t = 1 \, or \, \alpha_t^k = C, y_t = -1\} \end{aligned} \tag{4.5}$$

   因此，被激活变量的集合 A 每 $min(l, 1000)$ 次迭代后将被动态地缩小。问题（4.3）也因此而不断地被改变。注意，以上的 $m(\boldsymbol{\alpha}^k)$ 和 $M(\boldsymbol{\alpha}^k)$ 将在问题（4.3）中被计算。

2. 当然，以上的收缩策略可能会太过激。因为分解法将会显得慢收敛并且会因为要达到所要求精确率的最终数字而花费大量的迭代过程，所以我们并不希望，仅仅因为对（4.3）问题的一次错误的收缩，而使这些迭代过程被浪费。因此，当分解法第一次满足以下条件时

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + 10\epsilon \tag{4.6}$$

其中，$\epsilon$ 是被指定的终止容差，我们重新构造整个梯度。
详细内容将在第 4.3 部分中被阐述。

而对于 $\nu$-SVC 和 $\nu$-SVR，因为终止条件（3.16）式不同于（3.9）式，所以集合（4.5）必须被调整。对于 $y_t = 1$ 的情况，我们将在以下集合中收缩元素

$$\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m_p(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = 1\} \cup$$
$$\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M_p(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = 1\}.$$

对于 $y_t = -1$ 的情况，我们将考虑以下集合：

$$\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t > m_n(\boldsymbol{\alpha}^k), \alpha_t = 0, y_t = -1\} \cup$$
$$\{t \mid -y_t \nabla f(\boldsymbol{\alpha}^k)_t < M_n(\boldsymbol{\alpha}^k), \alpha_t = C, y_t = -1\}.$$

## 4.2 缓存

一种减少计算时间的有效技术就是缓存。尽管矩阵 $Q$ 是高密度的并且可能无法存贮于电脑内存中，但是我们仍然需要元素 $Q_{ij}$ 来被用来计算。我们可以使用一种被叫做缓存的特殊存贮器来储存最近被使用过的 $Q_{ij}$。然后一些核心元素就不需要被重复计算了。

因为在最后的迭代中仅仅只需要矩阵 $Q$ 的一些列，所以定理 4.1 也支持缓存的使用。如果缓存能够保存这些列，那么我们就能够在最后迭代中避免大多数的核心计算。

在 LIBSVM 中，对于缓存，我们将采用一种简单的最近最少使用策略。我们将使用一种由多个结构构成的循环表。每一个结构对应一个核心列，并且缓存那个列的一些元素。由于（4.3）将被动态地减小，所以如果第 i 列被需要但却不在缓存中，那么我们计算的同时也要将 $Q_{1,i}, \cdots, Q_{|A|,i}$ 储存到缓存中。因此，在电脑内存中缓存的列可能是不同长度的。

## 4.3 重新构造梯度

为了减少重新构造梯度 $\nabla f(\boldsymbol{\alpha})$ 所产生的花费，在迭代期间，我们将维持以下式子不变

$$\bar{G}_i = C \sum_{j:\alpha_j = C} Q_{ij}, i = 1, \cdots, l.$$

然后，对于 $\nabla f(\boldsymbol{\alpha})_i, i \notin A$，我们有

$$\nabla f(\boldsymbol{\alpha})_i = \sum_{j=1}^{l} Q_{ij}\alpha_j + p_i = \bar{G}_i + p_i + \sum_{\substack{j:j\in A \\ 0<\alpha_j<C}} Q_{ij}\alpha_j. \tag{4.7}$$

事实上，如果 $j \notin A$，那么 $\alpha_j = 0$ 或 $C$。

为了计算（4.7）式，我们需要一个由 $i$ 和 $j$ 所构成的两层循环。首先使用 $i$ 或者首先使用 $j$ 可能为导致一个非常不同的 $Q_{ij}$ 评估值。以下我们将讨论它们的不同点。注意，在我们的实现流程中，我们将一直交换 $\boldsymbol{\alpha}$ 的元素来保持 $A = \{1, \cdots, |A|\}$。

1. 1. 对于 $|A| + 1 \leq i \leq l$，计算 $Q_{i,1:|A|}$。虽然在（4.7）式中，由于我们的缓存的实现，所以仅仅只需要 $\{Q_{ij} \mid 0 < \alpha_j < C, j \in A\}$，但我们仍然必须获得 $Q_{i,1:|A|}(i.e., \{Q_{ij} \mid j \in A\})$ 的所有的元素。这种方法至多需要

$$(l - |A|) \cdot |A| \tag{4.8}$$

个核心估计。

2. 使 $F = \{j \,|\, 1 \le j \le |A| \ and \ 0 < \alpha_j < C\}$。对于每个 $j \in F$，我们获得 $Q_{1:l,j}$。虽然由于我们的缓存的实现，在计算 $\nabla f(\boldsymbol{\alpha})_i, i = |A|+1, \cdots, l$ 中仅仅只需要 $Q_{|A|+1:l,j}$，但我们仍然必须得到整个列。这一方法至多需要

$$l \cdot |F| \tag{4.9}$$

个核心估计。

通过比较（4.8）式和（4.9）式，我们可能会选择一种方法。但是，这一决策取决于 $Q$ 矩阵的元素是否已经被缓存了。如果缓存足够大，那么 $Q$ 矩阵的前 $|A|$ 个列往往会在缓存中，因为它们最近已经被使用了。因此，在方法 1 可能仍然需要 $(l - |A|) \cdot |A|$ 个核心估计的时候，方法 2 所需要的核心估计可能会少于 $l \cdot |F|$。

因此，因为方法 2 利用了缓存的实现，所以我们考虑如下的规则：

$$\begin{aligned} If \quad & l \cdot |F| > 2 \cdot (l - |A|) \cdot |A| \\ & \text{use method 1} \\ Else \quad & \\ & \text{use method 2} \end{aligned}$$

因为我们没有把缓存容量考虑进去，所以这一规则可能无法给出最优选择。但是，在最坏的情况下，被以上规则所选定的方法相比于另一种方法只是稍微慢了一点而已。在给出以下的一些详细解释之前，我们先做几个假设：

- 一个 LIBSVM 的训练过程涉及两次梯度的重新构造：第一次进行是当达到 $10\epsilon$ 容差时；参看公式（4.6）。第二次则是在训练过程的最后。
- 我们的规则将会安排同样的方法来实现两次梯度的重新构造。与此同时，这两次重新构造将花费一段相近的时间。

在这里，我们用“方法 1 的全部训练时间”来指代整个 LIBSVM 的训练时间（在这期间，方法 1 被用来重新构造梯度），同时用“方法 1 的重构时间”来指代用方法 1 单次重构梯度所花费的时间。然后我们考虑如下两种情况：

1. 选择了方法 1，但事实上方法 2 更好我们可以得到

$$\begin{aligned} & Total \ time \ of \ method \ 1 \\ \le \ & (Total \ time \ of \ method \ 2) + 2 \times (Reconstruction \ time \ of \ method \ 1) \\ \le \ & 2 \times (Time \ time \ of \ method \ 2). \end{aligned} \tag{4.10}$$

   我们详细地解释一下第二个不等式。如果方法 2 被用来重构梯度，那么在训练过程中，$Q_{:j}, j \in F$ 必定已经被计算过了（要么在正常的迭代过程中，要么在重构梯度的过程中）。既然 $l \cdot |F| > 2 \cdot (l - |A|) \cdot |A|$，那么必然有

$$(Reconstruction \ time \ of \ method \ 1) \le \frac{1}{2} \cdot (Total \ time \ of \ method \ 2)$$

   因此（4.10）式成立。

2. 选择了方法 2，但事实上方法 1 更好我们考虑最坏情况，即 $Q$ 矩阵的前 $|A|$ 个列的元素不在缓存中。因为 $|A|+1, \cdots, l$ 是收缩变量的指数，所以很可能 $Q$ 矩阵剩下的 $l - |A|$ 列元素也不在缓存中。既然 $l \cdot |F| \le 2 \cdot (l - |A|) \cdot |A|$，那么必然有

$$(Reconstruction \ time \ of \ method \ 2) \le 2 \cdot (Reconstruction \ time \ of \ method \ 1).$$

   因此，

$$\begin{aligned} & Total \ time \ of \ method \ 2 \\ \le \ & (Total \ time \ of \ method \ 1) + 2 \times (Reconstruction \ time \ of \ method \ 1). \end{aligned}$$

表 4.1 比较了在重构梯度的过程中核心评估的数目。其中我们考虑的是数据集 a7a 和数据集 ijcnn1。很明显，以上被提出的规则对于这两个数据集都选出了更好的方法。我们在 LIBSVM2.88 版本后应用了这一技术。

### 4.4 是否收缩都会带来更好的结果

我们发现，如果迭代的次数很大，那么收缩将缩短训练的时间。但是，如果我们早点结束训练过程（例如，用一个大的 $\epsilon$ 来作为终止容差），那么没有收缩的实现或许会更快。在这种情况下，迭代的次数通常是很小的。而且在所有迭代过程中所花费的时间相比于单个的梯度重构所花费的时间甚至都要更少。

| Cache = 1000MB | | Cache = 10MB | | | | |
|---|---|---|---|---|---|---|
| Method 1 | Method 2 | Method 1 | Method 2 | $\|F\|$ | $\|A\|$ | $l$ |
| 0 | 21,470,526 | 45,213,024 | 170,574,272 | 10,597 | 12,476 | 16,100 |
| 0 | 0 | 45,213,024 | 171,118,048 | 10,630 | 12,476 | 16,100 |
| 102s | 108s | 341s | 422s | | | |
| No shrinking: 111s | | No shrinking: 381s | | | | |

(a) a7a:$C$=1,$\gamma$=4,$\epsilon$=0.001.

| Cache = 1000MB | | Cache = 10MB | | | | |
|---|---|---|---|---|---|---|
| Method 1 | Method 2 | Method 1 | Method 2 | $\|F\|$ | $\|A\|$ | $l$ |
| 274,297,840 | 5,403,072 | 275,695,536 | 88,332,330 | 1,767 | 43,678 | 49,990 |
| 263,843,538 | 28,274,195 | 264,813,241 | 115,346,805 | 2,308 | 6,023 | 49,990 |
| 189s | 46s | 203s | 116s | | | |
| No shrinking: 42s | | No shrinking: 87s | | | | |

(b) ijcnn1:$C$=16,$\gamma$=4,$\epsilon$=0.5.

表 1: 分解法重构了梯度两次。我们在每一行展示了每次重构的核心估计的次数。我们选择了两种缓存容量来反映有或者没有充足缓存容量时的情况。最后一行用秒为单位给出了全部的训练时间（其中梯度构造是唯一组成部分）。我们认为 RBF 核为 $K(x_i, x_j) = exp(-\gamma\|x_i - x_j\|^2)$。

在表 4.1 中，我们展示了不用收缩法时的全部训练时间。对于 a7a，我们使用了默认的 $\epsilon$=0.001。在参数 $C$ = 1 和 $\gamma$=4 的情况下，迭代次数将超过 30,000 次。然后得出收缩是有用的。但是，对于 ijcnn1，我们使用一个宽松的容差 $\epsilon$=0.5，所以迭代的次数仅仅在 4,000 次左右。因为我们的收缩策略是相当的过激的，所以在第一次梯度重构之前，仅仅 $Q_{F,F}$ 存于缓存中，然后为了去计算梯度，我们又不得不去取 $Q_{:,F}$。

如果有足够的迭代，A 中大多数的元素都应该是自由的。相反，如果迭代的次数是小的（例如，表 4.1 中的 ijcnn1），我们会得到 $|F| \ll |A|$。因此，我们可以通过检查 $|F|$ 和 $|A|$ 的关系来推测是否收缩是有用的。在 LIBSVM 中，如果收缩是被启用的且在重构梯度时 $2 \cdot |F| < |A|$，那么我们将发出一个警告信息来表明：没有收缩的话，程序可能更快。

### 4.5 计算复杂度

在第 3.3 部分中，我们讨论了分解法的渐进收敛性。这里，我们将讨论计算复杂度。

主要操作是关于寻找（3.3）式的 $Q_{BN}\boldsymbol{\alpha}_N^k + \mathbf{p}_B$ 并且将 $\nabla f(\boldsymbol{\alpha}^k)$ 更新为 $\nabla f(\boldsymbol{\alpha}^{k+1})$。注意，在工作集的选择和终止条件中将用到 $\nabla f(\boldsymbol{\alpha})$。它们可以被如下式一起考虑进去

$$Q_{BN}\boldsymbol{\alpha}_N^k + \mathbf{p}_B = \nabla f(\boldsymbol{\alpha}^k) - Q_{BB}\boldsymbol{\alpha}_B^k \tag{4.11}$$

和

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k) \tag{4.12}$$

其中，$Q_{:,B}$ 是用集合 B 中的元素作为列的 $Q$ 矩阵的子矩阵。那就是，在第 $k$ 次迭代中，因为我们已经有了 $\nabla f(\boldsymbol{\alpha}^k)$，所以（4.11）的右式被用来构造子问题。在子问题被解决之后，（4.12）式被用来取得下一个 $\nabla f(\boldsymbol{\alpha}^{k+1})$。因为 B 仅仅只有两个元素并且解决子问题是容易的，所以主要的花费是（4.12）中的 $Q_{:,B}(\boldsymbol{\alpha}_B^{k+1} - \boldsymbol{\alpha}_B^k)$。这个操作本身将花费 $O(2l)$，但是如果 $Q_{:,B}$ 并不在缓存中且每一次核心估计要花费 $O(n)$，那么 $Q_{:,B}$ 的一列索引就已经需要 $O(ln)$。因此，复杂度为：

1. $\#Iterations \times O(l)$,如果在迭代过程中，$Q$ 矩阵的大多数列都已被缓存。

2. $\#Iterations \times O(nl)$,如果在迭代过程中，$Q$ 矩阵的大多数列都已被缓存且每一次核心估计的复杂度为 $O(n)$。

注意，如果收缩没有被合并进去的话，在迭代期间 l 将逐渐减小。

## 5 不平衡的数据

对于一些分类问题，处于不同类的数据的数量是不平衡的。因此，一些研究者（例如，([15]）已经提出了在 SVM 方程中使用不同惩罚系数：例如，$C$-SVM 变成

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C_+ \sum_{y_i=1} \xi_i + C_- \sum_{y_i=-1} \xi_i \\ subject\ to \quad & y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \cdots, l. \end{aligned}$$

它的对偶问题是

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ subject\ to \quad & 0 \leq \alpha_i \leq C_+, \text{if } y_i = 1, \qquad (5.1) \\ & 0 \leq \alpha_i \leq C_-, \text{if } y_i = -1, \qquad (5.2) \\ & \mathbf{y}^T\boldsymbol{\alpha} = 0. \end{aligned}$$

注意，用不同的 $C_i, i = 1, \cdots, l$ 来替换 $C$，这在早先的大多数分析中仍然是正确的。现在使用 $C_+$ 和 $C_-$ 仅仅是其中的一种特例。因此，实现流程几乎是相同的。一个主要的不同点是关于（3.3）的子问题的解。现在这个问题将变成：

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i \\ \alpha_j\end{bmatrix} + (Q_{i,N}\boldsymbol{\alpha}_N - 1)\alpha_i + (Q_{j,N}\boldsymbol{\alpha}_N - 1)\alpha_j \qquad (5.3) \\ subject\ to \quad & y_i\alpha_i + y_j\alpha_j = \Delta - \mathbf{y}_N^T\boldsymbol{\alpha}_N^k, \\ & 0 \leq \alpha_i \leq C_i, 0 \leq \alpha_j \leq C_j. \end{aligned}$$

其中，根据 $y_i$ 和 $y_j$，$C_i$ 和 $C_j$ 可以是 $C_+$ 或 $C_-$。

置 $\alpha_i = \alpha_i^k + d_i, \alpha_j = \alpha_j^k + d_j$ 并且 $\hat{d}_i \equiv y_i d_i, \hat{d}_j \equiv y_j d_j$。那么（5.3）式可以被写为

$$\begin{aligned} \min_{d_i,d_j} \quad & \frac{1}{2}\begin{bmatrix}d_i & d_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj}\end{bmatrix}\begin{bmatrix}d_i \\ d_j\end{bmatrix} + \begin{bmatrix}\nabla f(\boldsymbol{\alpha}^k)_i & \nabla f(\boldsymbol{\alpha}^k)_j\end{bmatrix}\begin{bmatrix}d_i \\ d_j\end{bmatrix} \qquad (5.4) \\ subject\ to \quad & y_i d_i + y_j d_j = 0, \\ & -\alpha_i^k \leq d_i \leq C - \alpha_i^k, -\alpha_j^k \leq d_j \leq C - \alpha_j^k. \end{aligned}$$

就像在（3.10）式中那样定义 $a_{ij}$ 和 $b_{ij}$。注意，如果 $a_{ij} \leq 0$，那么做相似于（3.4）式的修改。用 $\hat{d}_i = -\hat{d}_j$，那么目标函数可以被写为

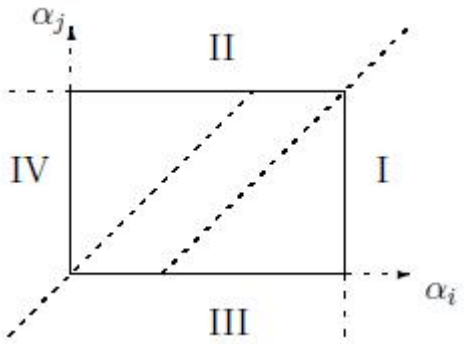
$$\frac{1}{2}\bar{a}_{ij}\hat{d}_j^2 + b_{ij}\hat{d}_j.$$

因此，

$$\begin{aligned} \alpha_i^{new} &= \alpha_i^k + y_i b_{ij}/\bar{a}_{ij}, \\ \alpha_j^{new} &= \alpha_i^k - y_j b_{ij}/\bar{a}_{ij}. \end{aligned} \qquad (5.5)$$

为了将它们改回到可行域，我们首先考虑 $y_i \neq y_j$ 的情况，并且将（5.5）式写为

$$\begin{aligned} \alpha_i^{new} &= \alpha_i^k + (-\nabla f(\boldsymbol{\alpha}^k)_i - \nabla f(\boldsymbol{\alpha}^k)_j)/\bar{a}_{ij}, \\ \alpha_j^{new} &= \alpha_i^k - (-\nabla f(\boldsymbol{\alpha}^k)_i - \nabla f(\boldsymbol{\alpha}^k)_j)/\bar{a}_{ij}. \end{aligned}$$




如果 $\boldsymbol{\alpha}^{new}$ 不是可行的，那么 $(\alpha_i^{new}, \alpha_j^{new})$ 将处于以下四个区域中的一个：

如果它不在区域 I 中，那么 $\alpha_i^{k+1}$ 将首先被设成 $C_i$，然后

$$\alpha_j^{k+1} = C_i - (\alpha_i^k - \alpha_j^k).$$

当然，首先我们必须检查它是否在区域 I 中。如果是，我们可以得到

$$\alpha_i^k - \alpha_j^k > C_i - C_j \; and \; \alpha_i^{new} \geq C_i.$$

对于其它情况也是相似处理的。因此，我们可以得出以下程序流程来辨识在不同区域中的 $(\alpha_i^{new}, \alpha_j^{new})$，并且将其变回到可行集。

```
if(y[i]!=y[j])
{
        double quad_coef = Q_i[i]+Q_j[j]+2*Q_i[j];
        if(quad_coef <= 0)
                quad_coef = TAU;
        double delta = (-G[i]-G[j])/quad_coef;
        double diff = alpha[i]-alpha[j];
        alpha[i] += delta;
        alpha[j] += delta;

        if(diff > 0)
        {
                if(alpha[j] < 0) // in region III
                {
                        alpha[j] = 0;
                        alpha[i] = diff;
                }
                else
                {
                        if(alpha[i] < 0) // in region IV
                        {
                                alpha[i] = 0;
                                alpha[j] = -diff;
                        }
                }
        }
        if(diff > C_i - C_j)
        {
                if(alpha[i] > C_i) // in region I
                {
                        alpha[i] = C_i;
                        alpha[j] = C_i - diff;
```

```
                }
        }
        else
        {
                if(alpha[j] > C_j) // in region II
                {
                        alpha[j] = C_j;
                        alpha[i] = C_j + diff;
                }
        }
}
```

## 6 多类别分类

对于多类别分类，我们采用“一对一”的方法（[10]），该方法将构造 $k(k-1)/2$ 个分类器，并且每一个分类器从两个不同类中训练数据。这种策略在 SVM 中的第一次尝试是在（[7]; [11]）中。为了训练来自第 $i$ 和第 $j$ 类的数据，我们解决了以下的二分类问题：

$$\begin{aligned}
\min_{\mathbf{w}^{ij},b^{ij},\boldsymbol{\xi}^{ij}} \quad & \frac{1}{2}(\mathbf{w}^{ij})^T\mathbf{w}^{ij} + C(\sum_t(\xi^{ij})_t) \\
subject\ to \quad & (\mathbf{w}^{ij})^T\phi(\mathbf{x}_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{if } \mathbf{x}_t \ in\ the\ \mathbf{i}th\ class, \\
& (\mathbf{w}^{ij})^T\phi(\mathbf{x}_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{if } \mathbf{x}_t \ in\ the\ \mathbf{j}th\ class, \\
& \xi_t^{ij} \geq 0.
\end{aligned}$$

在分类中我们使用一种投票策略：每一次二分类被当作为一次投票，并且可以向所有的数据点 x 投票，在最后，该点将被归为拥有最多票数的那类。

如果两类有相同的票数，那么我们将简单地选择拥有最小索引号的那一类，虽然这种方法可能并不是一种好的策略。

当然，对于多分类，还存在其他的方法。之所以我们选择“一对一”的方法的原因以及方法之间详细的比较，请参看[8]。

## 7 参数选择

LIBSVM 提供了一种使用 RBF 核的参数选择工具：借助于并行的格点搜索的交叉验证。虽然对于格点搜索而言，交叉验证是同时适用于 SVC 和 SVR 的，但当前我们只支持借助于参数 $C$ 和 $\gamma$ 的 $C$-SVC。对于其他核类型如线性核和多项式核，或者对于 SVR 而言，上述两个参数都可以很容易地进行相应地更改。

对于中等规模的问题，交叉验证对于参数选择而言可能是最为可靠的方式。首先，训练数据将被分成几折。随后，其中一折被当作为验证集，而其余的则被当作为训练集。预测验证集而得到的准确率的平均值就是交叉验证率。

我们的实现过程是按照如下方式进行的。用户提供一个可能的 $C$（或 $\gamma$）的区间来作为格点空间。然后，尝试所有的（$C$，$\gamma$）格点来看哪一个格点给出最高的交叉验证率。然后用户用所得到的最好的参数来训练整个训练集并产生最后的模型。

对于简单的实现，我们考虑借助于参数（$C$，$\gamma$）把每个 SVM 当作为一个独立的问题。因为它们是不同的工作，所以我们可以很容易地并行化地来处理它们。当前，LIBSVM 提供了一个非常简单的工具以便于将工作分派到一个共享同一文件系统的计算机集群中去。

注意，当前在同一个（$C$，$\gamma$）下，“一对一”方法是被用于训练不同类数据的。因此，在最后的模型中，所有 $k(k-1)/2$ 个决策函数将共享同一个（$C$，$\gamma$）。

LIBSVM 也将输出交叉验证率的等高线图。图 1 就是一个例子。

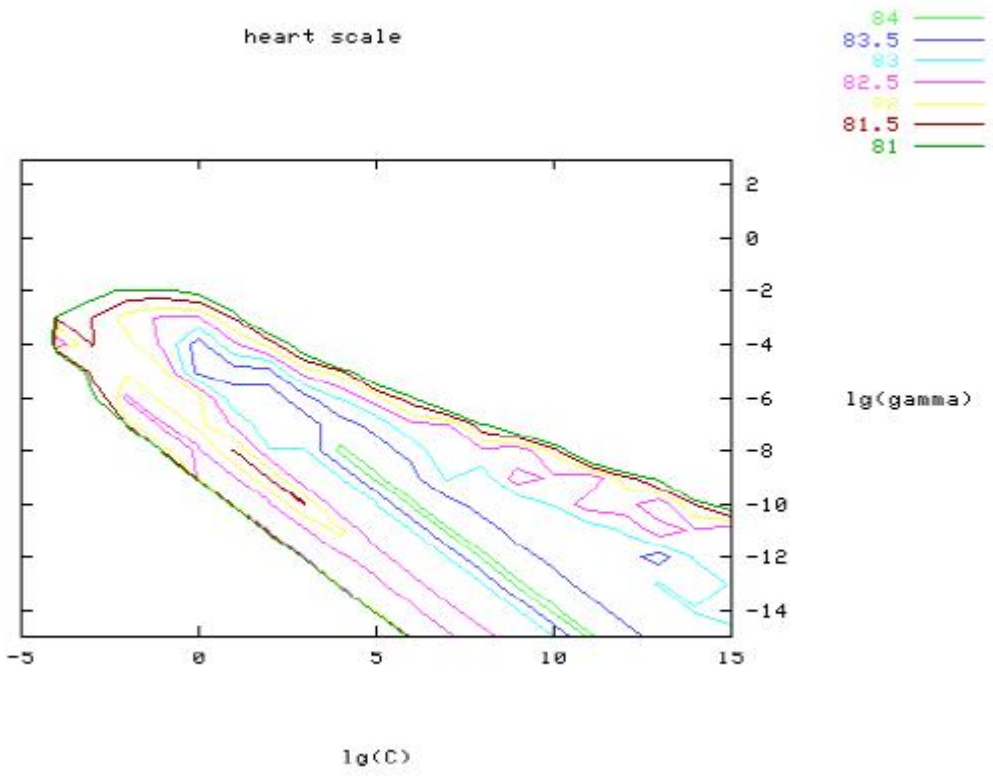



图 1: LIBSVM 软件包自带的 heart_scale 数据集的等高线图

## 8 概率估计

支持向量分类（或回归）仅仅预测类别标签（或近似目标值）而不是概率信息。以下我们将简短地阐述一下我们是如何将 SVM 扩展到概率估计的。对于分类的更详细的信息请参看[21]，对于回归的更详细信息请参看[12]。

假设给定 $k$ 类数据，对于任意 x，我们的目标是去估计

$$p_i = p(y = i \,|\, x), i = 1, \cdots, k.$$

对于多分类，在我们采用“一对一”的方法（例如，两两配对）的设定下，我们首先将估计配对类的概率

$$r_{ij} \approx p(y = i \,|\, y = i \, or \, j, x)$$

对于这一概率，我们将通过对（[16]）进行改进后的方法（[13]）来实现

$$r_{ij} \approx \frac{1}{1 + e^{A\hat{f}+B}} \tag{8.1}$$

其中，我们将使用已知的训练集数据和它们的决策值 $\hat{f}$ 来最小化负的对数似然函数，从而来估计 $A$ 和 $B$。由于我们需要独立的标签值和决策值，所以这里我们将进行五折交叉验证来获得决策值。

然后，我们将采用[21]中的第二种方法来从所有这些 $r_{ij}$ 中获得 $p_i$。它所解决的就是以下的最优化问题：

$$\begin{aligned} \min_{\mathbf{p}} \quad & \frac{1}{2} \sum_{i=1}^{k} \sum_{j:j \neq i} (r_{ji} p_i - r_{ij} p_j)^2 \\ subject \ to \quad & \sum_{i=1}^{k} p_i = 1, p_i \geq 0, \forall i. \end{aligned} \tag{8.2}$$

并且可以被整理成

$$\min_{\mathbf{p}} \frac{1}{2}\mathbf{p}^T\mathbf{p} \tag{8.3}$$

其中，

$$Q_{ij} = \begin{cases} \sum_{s:s\neq i} r_{si}^2, & \text{if } i = j, \\ -r_{ji}r_{ij}, & \text{if } i \neq j. \end{cases} \tag{8.4}$$

因为这是一个凸问题，所以它的最优化条件是存在一个常量 $b$ 必须满足

$$\begin{bmatrix} Q & \mathbf{e} \\ \mathbf{e}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}. \tag{8.5}$$

其中，$\mathbf{e}$ 是 $k \times 1$ 的全 1 向量，0 是 $k \times 1$ 的全 0 向量，并且 $b$ 是等式约束式 $\sum_{i=1}^k p_i = 1$ 的拉格朗日乘子。以下我们将派生一个简单的迭代方法，而不是直接来处理线性系统（8.5）。

因为

$$-\mathbf{p}^T Q\mathbf{p} = -\mathbf{p}^T Q(-bQ^{-1}\mathbf{e}) = b\mathbf{p}^T\mathbf{e} = b$$

所以解 $\mathbf{p}$ 必然满足

$$Q_{tt}p_t + \sum_{j:j\neq t} Q_{tj}p_j - \mathbf{p}^T Q\mathbf{p} = 0, \; for\, any\, t. \tag{8.6}$$

通过使用（8.6）式，我们可以来考虑以下算法：

**算法 2**

1. 以一些初始值 $p_i \geq 0, \forall i$ 来作为开始，并且 $\sum_{i=1}^k p_i = 1$。
2. 重复 $(t = 1, \cdots, k, 1, \cdots)$

$$p_t \leftarrow \frac{1}{Q_{tt}}[-\sum_{j:j\neq t} Q_{tj}p_j + \mathbf{p}^T Q\mathbf{p}] \tag{8.7}$$

$$normalize\ \mathbf{p} \tag{8.8}$$

直到（8.5）式被满足。

这一算法流程确保了得到（8.2）式的一个全局最优解。通过使用一些手段，我们就并不需要在每次迭代中都重复计算一遍 $\mathbf{p}^T Q\mathbf{p}$。详细的实现记录，请参看[**?**]的附录 $C$。对于算法 2，我们考虑一个相对终止条件：

$$\|Q\mathbf{p} - \mathbf{p}^T Q\mathbf{p}\mathbf{e}\|_1 = \max_t |(Q\mathbf{p})_t - \mathbf{p}^T Q\mathbf{p}| < 0.005/k.$$

当 $k$ 是很大的时候，$\mathbf{p}$ 将趋近于 0，所以我们可以通过 $k$ 的一个因子来减小容差。

接下来，我们来讨论一下 SVR 的概率推论。对于一个给定的训练集 $\mathcal{D} = \{(x_i, y_i) \,|\, x_i \in R^n, y_i \in R, i = 1, \cdots, l\}$，我们假定数据是从以下模型中收集而得：

$$y_i = f(x_i) + \delta_i \tag{8.9}$$

其中，$f(x)$ 是底层函数且 $\delta_i$ 是独立且同分布的随机噪声。假设给定一个测试数据 $x$，那么在给定 $x$ 和 $\mathcal{D}$ 下的 $y$ 的分布将是 $P(y \,|\, x, \mathcal{D})$，且这一分布将帮助我们得到关于 $y$ 的概率推论；例如，我们可以构建一个可预测性的区间 $\mathcal{I} = \mathcal{I}(x)$ 且满足在预先制定概率下 $y \in \mathcal{I}$。基于使用 SVR 的训练数据集 $\mathcal{D}$，我们把 $\hat{f}$ 标志为估计函数，那么 $\zeta = \zeta(x) \equiv y - \hat{f}(x)$ 就是样本外剩余（或者是预测偏差），并且 $y \in \mathcal{I}$ 等价于 $\zeta \in \mathcal{I} - \hat{f}(x)$。在基于使用训练集 $\mathcal{D}$ 而得到的样本外剩余集合 $\{\zeta_i\}_{i=1}^l$ 下，我们提议构建 $\zeta$ 的分布。通过首先执行一次 $k$ 折交叉验证来产生

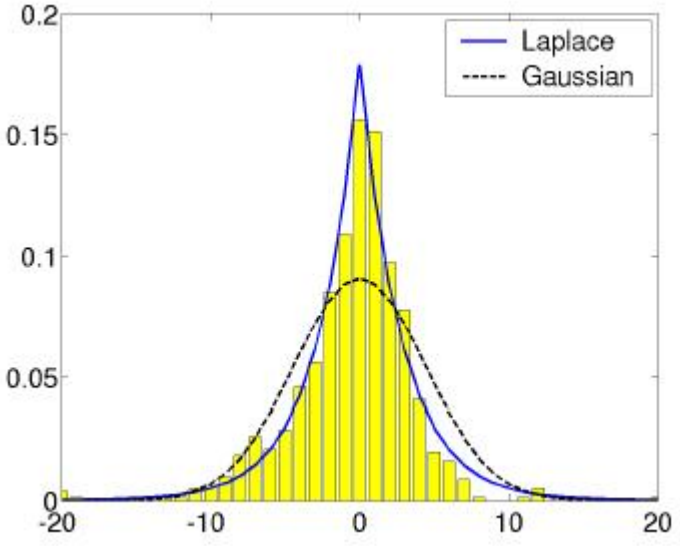



图 2: 借助于拉普拉斯和高斯分布，对一个数据集建模而得到的 $\zeta_i s$ 的直方图。$x$ 坐标是使用 5 折交叉验证而得到的 $\zeta_i$，而 $y$ 坐标则是用每个宽度为 1 的格子而构成的数据的规格化数。

$\zeta_i s$，从而来得到 $\hat{f}_j, j = 1, \cdots, k$，然后在第 $j$ 折中对于 $(x_i, y_i)$ 来设定 $\zeta_i \equiv y_i - \hat{f}_j(x_i)$。显然，$\zeta_i s$ 的分布可能类似于预测偏差 $\zeta$ 的分布。

图 2 向我们展示了从一个真实数据中得到的 $\zeta_i s$。基本上，像直方图那样的一个离散分布可以被使用来构建数据的模型；但是，这样是复杂的，因为所有的 $\zeta_i s$ 必须被保留。相反，像高斯和拉普拉斯那样的分布（通常被用来作为噪声模型）只需要定位点和缩放参数。在图 2 中我们绘制了用这两种分布的合适的曲线以及 $\zeta_i s$ 的直方图。图表明，$\zeta_i s$ 的分布似乎关于 0 对称，并且高斯和拉普拉斯分布都合理地刻画出了 $\zeta_i s$ 的形状。因此，我们提议用均值为 0 的高斯和拉普拉斯分布来构建 $\zeta_i s$ 的模型，或者等价的，在以高斯和拉普拉斯分布的均值来作为 $\hat{f}(x)$ 的情况下，来构建 $y$ 的条件分布。

（[12]）中讨论了一种方法来判断是否一种拉普拉斯和高斯分布应该被使用。此外，他们通过实验表明，在所有他们已经尝试过的情况中，拉普拉斯分布是最好的。因此，这里我们考虑均值为 0 的拉普拉斯分布，其密度函数为：

$$p(z) = \frac{1}{2\sigma} e^{-\frac{|z|}{\sigma}} \tag{8.10}$$

假定 $\zeta_i$ 是独立的，那么我们可以通过最大似然估计来估计缩放参数。对于拉普拉斯分布，其最大似然估计是

$$\sigma = \frac{\sum_{i=1}^{l} |\zeta_i|}{l} \tag{8.11}$$

（[12]）中指出，一些“非常极端的”$\zeta_i$ 可能会引起对 $\sigma$ 的不准确的估计。因此，他们提议通过丢弃那些超过 $\pm 5\times$（$\zeta_i$ 的标准偏差）的 $\zeta_i s$ 来估计缩放参数。因此，对于任何新数据 x，我们认为

$$y = \hat{f}(x) + z$$

其中，$z$ 是在带有参数 $\sigma$ 的拉普拉斯分布之后的一个随机变量。

理论上，$\zeta$ 的分布可能要根据输入的 $x$ 而定，但是这里我们假设它与 $x$ 无关。这与用于分类的模型（8.1）之间是相似的。这样的假设在实践中运行良好，而且还因此产生出一个简单的模型。

## 致谢

这项工作得到了台湾国家科学委员会授予的 NSC 89-2213-E-002-013 和 NSC 89-2213-E-002-106 的部分支持。作者在此感谢许智玮和李仁豪，感谢他们在此期间给予我们的许多有益的讨论

和评论。同时我们也要感谢 Ryszard Czerminski 和 Lily Tian 所提供的一些有用的评论。

## 参考文献

[1] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144-152. ACM Press, 1992.

[2] C.-C. Chang and C.-J. Lin. Training $\nu$-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119-2147, 2001.

[3] P.-H. Chen, R.-E. Fan, and C.-J. Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 17:893-908, July 2006. URL http://www.csie.ntu.edu.tw/~cjlin/papers/generalSMO.pdf.

[4] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273-297,1995.

[5] D. J. Crisp and C. J. C. Burges. A geometric interpretation of $\nu$-SVM classifiers. In S. Solla, T. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems*, volume 12, Cambridge, MA, 2000. MIT Press.

[6] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889-1918, 2005. URL http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf.

[7] J. Friedman. Another approach to polychotomous classication. Technical report, Department of Statistics, Stanford University, 1996. Available at http://www-stat.stanford.edu/reports/friedman/poly.ps.Z.

[8] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415-425, 2002.

[9] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[10] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1990.

[11] U. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 255-268, Cambridge, MA, 1999. MIT Press.

[12] C.-J. Lin and R. C. Weng. Simple probabilistic predictions for support vector regression. Technical report, Department of Computer Science, National Taiwan University, 2004. URL http://www.csie.ntu.edu.tw/~cjlin/papers/svrprob.pdf.

[13] H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on Platt's probabilistic outputs for support vector machines. Technical report, Department of Computer Science, National Taiwan University, 2003. URL http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.ps.

[14] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, pages 130-136, New York, NY, 1997a. IEEE.

[15] E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. AI Memo 1602, Massachusetts Institute of Technology, 1997b.

[16] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, Cambridge, MA, 2000. MIT Press. URL citeseer.nj.nec.com/platt99probabilistic.html.

[17] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[18] B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207-1245, 2000.

[19] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443-1471, 2001.

[20] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY, 1998.

[21] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975-1005, 2004. URL http://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf.