libSVM 源码解读

邢存远, https://welts.xyz

2021 年 7 月 17 日

目录

1 引言

早在学习 SVM 时，笔者便有亲手实现一个 SVM 的想法。后来发现其实现难度与数学技巧远高于单隐层神经网络，这对于只能写出一个二分类感知机的我不亚于小学生做高考题。在老师的建议下，笔者决定去阅读当前最流行的 SVM 代码库：libSVM 的源代码，不仅是学习 SVM 怎么写，也是学习一个合格的代码框架应该如何去设计。在此之前，笔者已经对 SVM 的 SMO 算法和实现技巧进行了一些零散的了解，这里打算将它们串联起来，同时为阅读源码提供一定的数学基础。

2 SVM，以及 libSVM 简介

支持向量机(SVM, Support Vector Machine) 属于一种线性分类器，是建立在统计学习理论的 VC 维理论和结构风险最小原理的基础上，根据有限的训练集，在模型的复杂性和学习性之间寻求最佳的折中，以获得最好的泛化能力的经典分类方法。[1]

libSVM是由国立台湾大学的林智仁教授等开发的一款利用支持向量机用于分类、回归和区间估计等机器学习任务的多语言（C++、Java、Python、MATLAB 等）、跨平台（Windows、Linux、mac OS）的工具包，最新版本为 Version 3.25。

3 数学基础

SVM 本质上是一种统计学习模型，libSVM 的实现中涉及到很多矩阵，导数，概率论，尤其是优化方面的知识。假设读者已经有掌握线性代数和微积分包括概率论的基本知识，我们将 libSVM 涉及到的数学知识先在这里进行整理。

3.1 矩阵导数

此处待补充

3.2 优化问题

我们常用拉格朗日对偶性去求解优化问题，这也在 SVM 中被频繁使用。关于优化问题我们参考的是《Convex optimization》这本书 [2]。先来看优化问题的形式化定义。

3.2.1 原始问题

我们将

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \le 0, i = 1, \cdots, m \\ & h_j(x) = 0, j = 1 \cdots, n \end{aligned} \tag{1}$$

其中 $f(x)$ 和 $g_i(x)$ 都是凸函数，$h_j(x)$ 都是仿射函数，具有这种形式的问题称作凸优化问题。又由于我们常常不会直接求解它，因此也称其为“初始问题”。定义拉格朗日函数：

$$\mathcal{L}(x,\lambda,\mu)=f(x)+\sum_{i=1}^{m}\lambda_i g_i(x)+\sum_{j=1}^{n}\mu_i h_j(x),\quad \lambda_i\geq 0,\mu_j\in\mathbb{R} \tag{2}$$

定义函数 θ_P：

$$\theta_P(x)=\begin{cases}f(x) & x\text{满足约束}\\ +\infty & \text{else}\end{cases} \tag{3}$$

可以证明对任意 x，我们有

$$\begin{aligned}\theta_P(x)&=\max_{\lambda,\mu:\lambda_i\geq 0}\mathcal{L}(x,\lambda,\mu)\\ \min_x\theta_P(x)&=\min_x\max_{\lambda,\mu:\lambda_i\geq 0}\mathcal{L}(x,\lambda,\mu)\end{aligned} \tag{4}$$

可以证明，原问题与拉格朗日函数的“极小极大问题”有相同的解，是等价的。我们将原始问题的最优值

$$p^*=\min_x\theta_P(x) \tag{5}$$

称为**原始问题的值**。

3.2.2 对偶问题

我们先设函数

$$\theta_D(\lambda,\mu)=\min_x\mathcal{L}(x,\lambda,\mu) \tag{6}$$

然后对其求极大，也就是“极大极小问题”：

$$\max_{\lambda,\mu}\min_x\mathcal{L}(x,\lambda,\mu)=\max_{\lambda,\mu}\theta_D(x) \tag{7}$$

这个“极大极小问题”就是原始问题 (1) 的**对偶问题**，类似的，设其最优值为 d^*。

3.2.3 强对偶，弱对偶，以及 KKT 条件

弱对偶性，也就是极大极小问题的最优值必然不大于极小极大问题的最优值，这是普遍存在的：

$$d^*\leq p^* \tag{8}$$

一个形象的理解是，矮子中最高的还是矮子，身高不超过高个子中最矮的。但我们最想要的其实是只有等号成立，方便问题的求解。当两个最优值相等时强对偶性成立。幸运的是强对偶性有一个充分必要条件：KKT 条件，即 x^* 和 λ^*,μ^* 分别是原问题和对偶问题的解当且仅当下面各式成立：

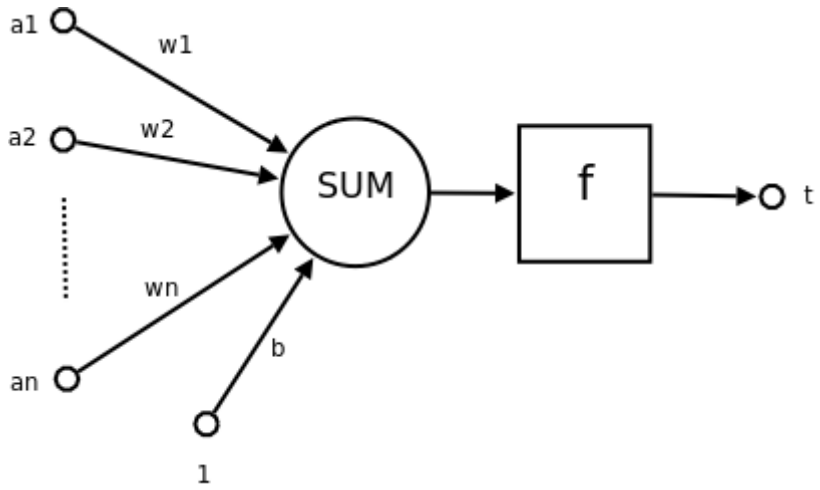
$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) &= 0 \\ \lambda_i^* g_i(x^*) &= 0, i = 1, \cdots, m \\ g_i(x^*) &\leqslant 0, i = 1, \cdots, m \\ \lambda_i &\geqslant 0, i = 1, \cdots, m \\ h_j(x^*) &= 0, i = 1, \cdots, n \end{aligned} \tag{9}$$

其中第二个条件称作 KKT 的**对偶互补条件**，如果 $\lambda_i > 0$ 则必有 $g_i(x^*) = 0$。

4 支持向量机

4.1 感知机

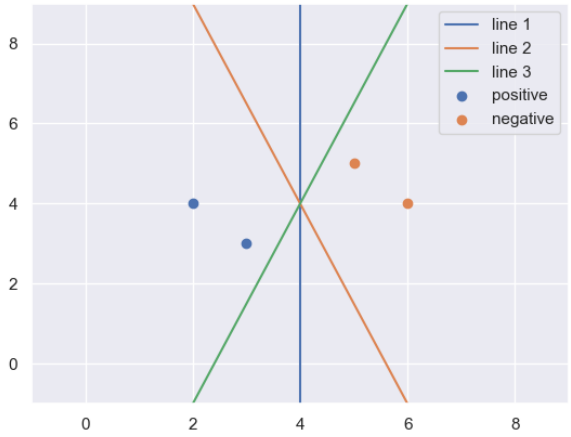
感知机（Perceptron）是最简单的人工神经网络：



也是一种二元线性分类器。给定线性可分的数据集，感知机可以找到一个将样本分开的超平面：

$$\sum_{i=1}^{n} w_i x_i + b = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{10}$$

实际上对于同一个数据集，我们常常可以得到多个超平面：



上面三条直线 l1、l2 和 l3 都可以将正负样本分开，而我们更倾向于选择位于两类样本“中间”的划分超平面 l2，因为它对训练样本的扰动“容忍”性最好。换言之，泛化能力最强。支持向量机便是这样的一种较优的感知机。

4.2 形式化支持向量机

我们不能凭肉眼在感知机中找到支持向量机，我们需要将求解它的过程形式化。我们实际上要找的是这样一个超平面：

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0 \tag{11}$$

$\boldsymbol{w}$ 为超平面法向量，b 为位移项。由简单的解析几何得到空间中点 $\boldsymbol{x}$ 到平面的距离：

$$r = \frac{|\boldsymbol{w}^\top \boldsymbol{x} + b|}{\|\boldsymbol{w}\|} \tag{12}$$

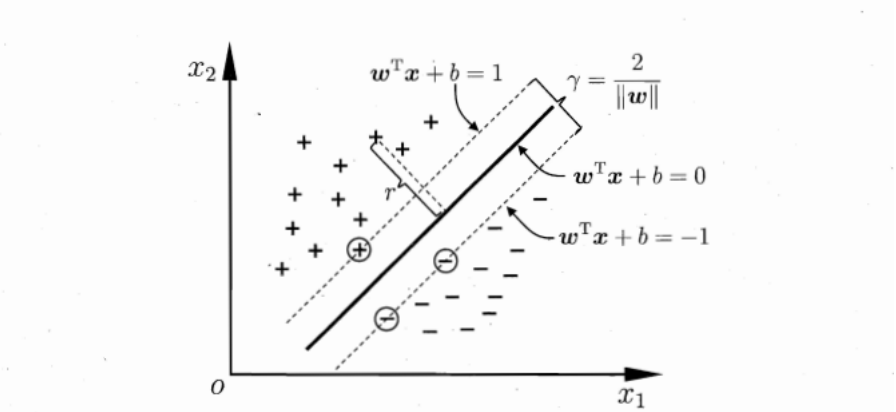
将二分类问题中的数据集标签 y_i 映射到 $\{-1, +1\}$，也就是正类 $y_i = 1$，负类 $y_i = -1$，我们想让该超平面正确分类，则有：

$$\begin{cases} \boldsymbol{w}^\top \boldsymbol{x} + b \geq +1, y_i = +1 \\ \boldsymbol{w}^\top \boldsymbol{x} + b \leq -1, y_i = -1 \end{cases} \tag{13}$$

我们的目标其实是，给定一个分离超平面，距离该超平面最近的正类样本和负类样本（也就是上述约束中的不等号为等号时的样本点，称作**支持向量**）到超平面的距离之和

$$\gamma = \frac{2}{\|\boldsymbol{w}\|} \tag{14}$$

最大，如下图所示：



其中满足 $\boldsymbol{w}^\top \boldsymbol{x}+b=1$ 或者 -1 的点就是支持向量。

我们将这个问题形式化成前面提到的优化原始问题 ：

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & 1-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i+b)\le 0, i=1,2,\cdots,m \end{aligned} \tag{15}$$

有两点值得注意：

- 原来是将 γ 极大化，为了问题的标准和求导的方便，将目标函数写成 $\|\boldsymbol{w}\|^2/2$；
- 这里约束条件里的 y_i 调换了位置，是由 $y_i^2=1$ 导出，后面还会用到。

4.3 用拉格朗日对偶求解

先有原问题的拉格朗日函数：

$$\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\lambda})=\frac{1}{2}\|\boldsymbol{w}\|^2+\sum_{i=1}^{m}\lambda_i(1-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i+b)) \tag{16}$$

其中 $\boldsymbol{\lambda}=(\lambda_1,\lambda_2,...,\lambda_m),\lambda_i\ge 0$. 我们先求 $\min_{\boldsymbol{w},b}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\lambda})$，对变量求偏导：

$$\begin{cases} \dfrac{\partial}{\partial \boldsymbol{w}}\mathcal{L}=\boldsymbol{w}-\sum_{i=1}^{m}\lambda_i y_i \boldsymbol{x}_i \\ \dfrac{\partial}{\partial b}\mathcal{L}=\sum_{i=1}^{m}\lambda_i y_i \end{cases} \tag{17}$$

令上面两式为 0：

$$\begin{cases} \boldsymbol{w}=\sum_{i=1}^{m}\lambda_i y_i \boldsymbol{x}_i \\ 0=\sum_{i=1}^{m}\lambda_i y_i \end{cases} \tag{18}$$

将 (18) 带入 (16)，则可消去变量：

$$\begin{aligned}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\lambda}) &= \frac{1}{2}\|\boldsymbol{w}\|^2+\sum_{i=1}^{m}\lambda_i(1-y_i(\boldsymbol{w}^\top\boldsymbol{x}_i+b))\\ &= \frac{1}{2}(\sum_{i=1}^{m}\lambda_i y_i\boldsymbol{x_i})^2+\sum_{i=1}^{m}\lambda_i-b\sum_{i=1}^{m}\lambda_i y_i-\sum_{i=1}^{m}\lambda_i y_i\boldsymbol{w}^\top\boldsymbol{x}_i\\ &= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j\boldsymbol{x}_i^\top\boldsymbol{x}_j-\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j\boldsymbol{x}_i^\top\boldsymbol{x}_j+\sum_{i=1}^{m}\lambda_i\\ &= \sum_{i=1}^{m}\lambda_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j\boldsymbol{x}_i^\top\boldsymbol{x}_j\end{aligned} \tag{19}$$

从而得到优化问题的对偶问题：

$$\begin{aligned}\max_{\boldsymbol{\lambda}}\quad & \sum_{i=1}^{m}\lambda_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j\boldsymbol{x}_i^\top\boldsymbol{x}_j\\ \text{s.t.}\quad & \sum_{i=1}^{m}\lambda_i y_i=0\\ & \lambda_i\geq 0, i=1,2,...,m.\end{aligned} \tag{20}$$

如果能解出 $\boldsymbol{\lambda}$，我们就得到模型：

$$\begin{aligned}f(\boldsymbol{x}) &= \boldsymbol{w}^\top\boldsymbol{x}+b\\ &= \sum_{i=1}^{m}\lambda_i y_i\boldsymbol{x}_i^\top\boldsymbol{x}+b\end{aligned} \tag{21}$$

从对偶问题解出的 λ_i 式拉格朗日乘子，对应的是样本 $(\boldsymbol{x}_i,y_i)$，由于原问题有不等式约束，所以上述过程需满足 KKT 条件：

$$\begin{cases}\lambda_i\geq 0;\\ y_i f(\boldsymbol{x}_i)-1\geq 0\\ \lambda_i(y_i f(\boldsymbol{x}_i)-1)=0\end{cases}$$

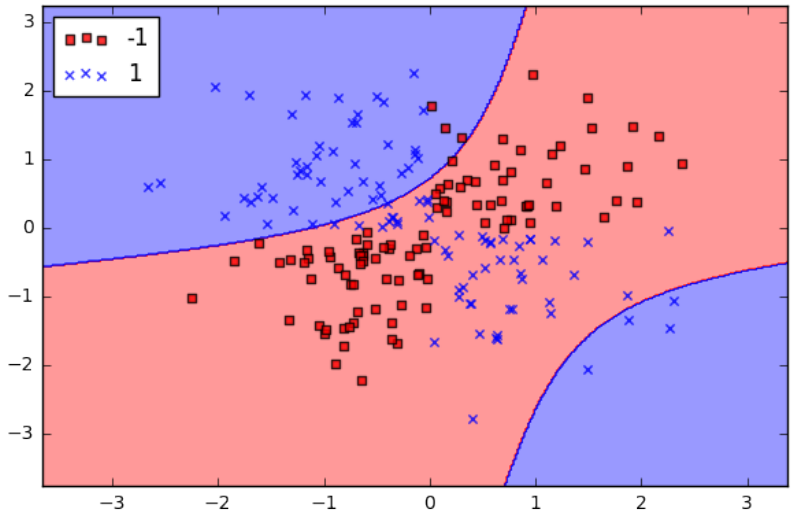
于是，对于任意训练样本 $(\boldsymbol{x_i},y_i)$，总有 $\lambda_i=0$ 或 $y_i f(x_i)=1$. $\lambda_i=0$ 的点不会对 $f(\boldsymbol{x})$ 有任何影响，否则**样本点都在最大间隔边界上**，是一个支持向量. 这为我们的训练带来启示，即大部分样本都不需要，最终模型只与支持向量有关.

4.4 核函数

以下是百度百科上关于核函数的定义：

> 支持向量机通过某非线性变换 $\phi(x)$ ，将输入空间映射到高维特征空间。特征空间的维数可能非常高。如果支持向量机的求解只用到内积运算，而在低维输入空间又存在某个函数 $K(\boldsymbol{x},\boldsymbol{x}')$ ，它恰好等于在高维空间中这个内积，即 $K(\boldsymbol{x},\boldsymbol{x}')=\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x}')\rangle$ 。那么支持向量机就不用计算复杂的非线性变换，而由这个函数 $K(\boldsymbol{x},\boldsymbol{x}')$ 直接得到非线性变换的内积，使大大简化了计算。这样的函数 $K(\boldsymbol{x},\boldsymbol{x}')$ 称为核函数。

而简单地说，通过用 $K(\boldsymbol{x},\boldsymbol{x}')$ 去替换简单的向量内积，使得决策边界不再是分离超平面，而是一个曲面，有效解决了线性不可分的问题，比如下图：



利用 RBF 核，我们的决策边界变成了曲线。读者可能注意到，图中的样本点并没有被完全分开，甚至样本是不可分的。而利用我们在下面提到的 C-SVC，可以解决这个问题。

4.5 更多 SVM

上面的 SVM 是最基础的支持向量机，libSVM 能够求解比这种复杂得多的问题。这一部分我们来介绍 libSVM 中的 5 种 SVM。

4.5.1 C-SVC

C-SVC 是上面的 SVM 的“升级版”，叫做软间隔支持向量机，所谓的“软”，就是在不可分（即使是非线性）的情况下，允许部分样本点不满足约束，如上图所示，但对于这些点必然要有相应“惩罚”。

加入象征不满足约束程度的稀疏变量 $\boldsymbol{\xi}$，C-SVC 其实是解决下面的优化问题：

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b) \geqslant 1-\xi_i \\ & \xi_i \geqslant 0, i = 1,\cdots,l \end{aligned} \tag{22}$$

我们也不难得到其对应的对偶问题（习惯上，前面提到的的拉格朗日乘子 $\boldsymbol{\lambda}$ 用 $\boldsymbol{\alpha}$ 代替）：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} - \boldsymbol{e}^\top\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^\top\boldsymbol{\alpha} = 0, \\ & 0 \leqslant \alpha_i \leqslant C, i = 1,\cdots,l \end{aligned} \tag{23}$$

其中 $\boldsymbol{e}$ 是一个全 1 向量，而 $Q_{ij} = y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) = y_i y_j \phi(\boldsymbol{x}_i)^\top\phi(\boldsymbol{x}_j)$。当我们解决了上面的对偶问题后，我们就可以得到 $\boldsymbol{w}$ 的解：

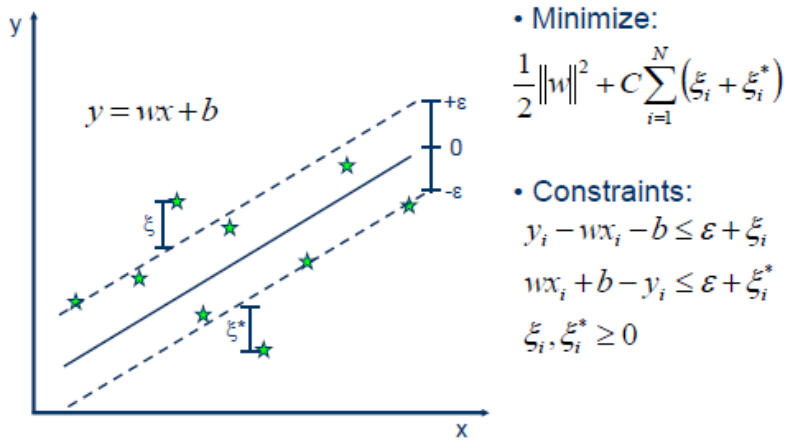
$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i) \tag{24}$$

从而决策函数：

$$\operatorname{sgn}(\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right) \tag{25}$$

4.5.2 ϵ-SVR

ε-SVR，也就是 ε-Support Vector Regression，是利用支持向量机来解决回归问题：



同样，这里的稀疏变量 $\boldsymbol{\xi}$ 用来衡量给定边界 ε 后样本点对决策边界的违背程度。原始优化问题已经在上图写出，而其对偶问题为：

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^\top Q(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) + \varepsilon \sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} z_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \boldsymbol{e}^\top(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) = 0 \\ & 0 \leqslant \alpha_i, \alpha_i^* \leqslant C, i = 1, \cdots, l \end{aligned} \tag{26}$$

这里的 z_i 是对应数据的输出，如此设置是为了不和分类问题中的标签 y_i 混淆。

此处 $Q_{ij} = K(x_i, x_j)$。当我们求出该对偶问题后，也就能得到拟合函数

$$z(\boldsymbol{x}) = \sum_{i=1}^{l}(-\alpha_i + \alpha_i^*) K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{27}$$

4.5.3 ν-SVC

ν-Support Vector Classification 在前面的 C-SVC 基础上引入了一个新参数 ν，用以控制训练误差和支持向量的数量：

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 - \nu\rho + \frac{1}{l}\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b) \geqslant \rho - \xi_i \\ & \xi_i \geqslant 0, i = 1, \cdots, l \\ & \rho \geqslant 0 \end{aligned} \tag{28}$$

对应的对偶问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leqslant \alpha_i \leqslant \frac{1}{l},, i = 1, \cdots, l \\ & \boldsymbol{e}^\top\boldsymbol{\alpha} \geqslant \nu, \boldsymbol{y}^\top\boldsymbol{\alpha} = 0 \end{aligned} \tag{29}$$

可以把它和 C-SVC 的对偶问题进行比较，发现框架大体相同，ν 其实是对 $\boldsymbol{e}^\top\boldsymbol{\alpha}$ 进行了限制。两种 SVC 问题的决策函数是相同的。

4.5.4 ν-SVR

类似的，ν-Support Vector Regression 是将 ν 引入 ε-SVR，它解决的是一个优化问题：

$$\begin{aligned} \min_{\boldsymbol{x},b,\boldsymbol{\xi},\boldsymbol{\xi}^*,\varepsilon} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\big(\nu\varepsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)\big) \\ \text{subject to} \quad & (\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b) - z_i \leqslant \varepsilon + \xi_i \\ & z_i - (\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b) \leqslant \varepsilon + \xi_i^* \\ & \xi_i, \xi^* \geqslant 0, i = 1, \cdots, l \\ & \varepsilon \geqslant 0 \end{aligned}$$

发现约束条件与 ε-SVR 差别不大，其对偶问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^\top Q(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \boldsymbol{z}^\top(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) \\ \text{subject to} \quad & \boldsymbol{e}^\top(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) = 0, \boldsymbol{e}^\top(\boldsymbol{\alpha}+\boldsymbol{\alpha}_i^*) \leqslant C\nu \\ & 0 \leqslant \alpha_i, \alpha_i^* \leqslant C/l, i = 1, \cdots, l \end{aligned}$$

且其解出的近似函数和 ε-SVR 相同。

4.5.5 One-class SVM

现实中存在这样一个问题，在数据集 X 中判断某个数据 x_i 是不是异常数据，也就是说，对于 x_i，我们想判断 x_i 与 $X\backslash x_i$ 有多相似，如果相似度低，就将其剔除。可以发现这与分类任务并不完全相同，因为涉及到的类别只有一种。用于处理这一问题的支持向量机被称作 One-class SVM。这一过程也被称为分布估计（Distribution estimate）。

One-class SVM 的原问题：

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

其对偶问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/(\nu l), i = 1, \cdots l \\ & \boldsymbol{e}^\top \boldsymbol{\alpha} = 1 \end{aligned}$$

其中 $Q_{ij} = K(x_i, x_j)$，决策函数：

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

以上就是 libSVM 支持的五种 SVM 模型，其中最后一个模型，也就是 one-class SVM，在笔者学习过程中很少遇到，因此下一节专门讨论所谓的分布估计问题。

5 SVM 的分布估计

SVM 可以在不提供先验概率的情况下对标签数据（和分类任务中的目标值）进行分布估计，我们这里简单介绍这些问题的思路以及 libSVM 中相应的算法。

5.1 k 分类问题的概率估计

给定共 k 类数据，对于任意数据 $\boldsymbol{x}$，我们的目标是估计出

$$p_i = \Pr(y = i|\boldsymbol{x}), i = 1, \cdots, k \tag{30}$$

我们用“一对一”的方法，先估计成对概率：

$$r_{ij} \approx \Pr(y = i|y = i \text{ or } j, \boldsymbol{x}) \tag{31}$$

我们做出假设，r_{ij} 可以写成如下形式：

$$r_{ij} \approx \frac{1}{1 + \exp(A\hat{f} + B)} \tag{32}$$

其中：

$$\hat{f} = f(\boldsymbol{x}) = \sum_{i=1}^{l}\alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{33}$$

也就是决策函数。然后我们利用基于训练数据的极大似然估计法，估计出参数 A 和 B。考虑到可能会过拟合，因此 libSVM 会先采用 5 折交叉验证去获取 $\hat{f}$。

在得到所有的 r_{ij} 后，我们便着手寻找一组能够最契合 r_{ij} 的概率分布 $[p_1, p_2, \cdots, p_k]$，这相当于求解下面的优化问题：

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \frac{1}{2}\sum_{i=1}^{k}\sum_{j:j\neq i}(r_{ji}p_i - r_{ij}p_j)^2 \\ \text{subject to} \quad & p_i \geq 0, \forall i \\ & \sum_{i=1}^{k} p_i = 1 \end{aligned} \tag{34}$$

该问题基于下面的概率等式（不难证明）：

$$\Pr(y=j|y=i \text{ or } y=j, \boldsymbol{x}) \cdot \Pr(y=i|\boldsymbol{x}) = \Pr(y=i|y=i \text{ or } y=j, \boldsymbol{x}) \cdot \Pr(y=j|\boldsymbol{x}) \tag{35}$$

接着将原问题重构成矩阵形式：

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \frac{1}{2}\boldsymbol{p}^\top Q \boldsymbol{p} \\ & Q_{ij} = \begin{cases} \sum_{s:s\neq i} r_{si}^2 & \text{if } i = j \\ -r_{ji}r_{ij} & \text{else} \end{cases} \end{aligned} \tag{36}$$

可以证明 p_i 非负的约束是冗余的。这样只剩下 $\sum p_i = 1$ 的约束，设其对应的拉格朗日乘子为 b，从而直接写出最优性条件：

$$\begin{bmatrix} Q & \boldsymbol{e} \\ \boldsymbol{e}^\top & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{37}$$

其中 $\boldsymbol{e}$ 是 k 维全 1 向量。除了使用高斯消去法去解这个方程组，我们也可以使用普通的迭代法，以更好地在计算机上解决。由

$$\begin{aligned} Q\boldsymbol{p} + b\boldsymbol{e} &= 0 \\ -\boldsymbol{p}^\top Q p = b\boldsymbol{p}^\top \boldsymbol{e} \\ &= b \end{aligned} \tag{38}$$

从而最优解 $\boldsymbol{p}$ 满足

$$(Q\boldsymbol{p})_t - \boldsymbol{p}^\top Q \boldsymbol{p} = Q_{tt}p_t + \sum_{j:j\neq t} Q_{tj}p_j - \boldsymbol{p}^\top Q \boldsymbol{p} \tag{39}$$

我们根据这个等式提出 $\boldsymbol{p}$ 的迭代算法：

算法迭代 p_i 时是将下面两个步骤融合：

$$\begin{aligned} p_i &\leftarrow \frac{1}{Q_{tt}}\big[-\sum_{j:j\neq t} Q_{tj}p_j + \boldsymbol{p}^\top Q \boldsymbol{p}\big] \\ \boldsymbol{p} &\leftarrow \frac{1}{\sum p_i}\boldsymbol{p} \quad \text{(normalization)} \end{aligned}$$

Algorithm 1

Initialize $\boldsymbol{p}$ randomly with $\sum_{i=1}^{k} p_i = 1, p_i \geq 0, \forall i$
repeat
  $i \leftarrow 1$
  repeat
    $p_i \leftarrow p_i + \frac{1}{Q_{tt}}\left[-(Q\boldsymbol{p})_t + \boldsymbol{p}^\top Q\boldsymbol{p}\right]$
    $i \leftarrow i + 1$
  until $i > k$
until $\max_t |(Q\boldsymbol{p})_t - \boldsymbol{p}^\top Q\boldsymbol{p}| < \frac{0.005}{k}$

此外，考虑到迭代终止条件（即满足线性方程 (37)）过于严苛，我们提出一个收敛阈值：

$$\|Q\boldsymbol{p} - \boldsymbol{p}^\top Q\boldsymbol{p}\boldsymbol{e}\|_\infty = \max_t |(Q\boldsymbol{p})_t - \boldsymbol{p}^\top Q\boldsymbol{p}| < \frac{0.005}{k}$$

利用 k 来控制收敛。

5.2 回归问题的噪声估计

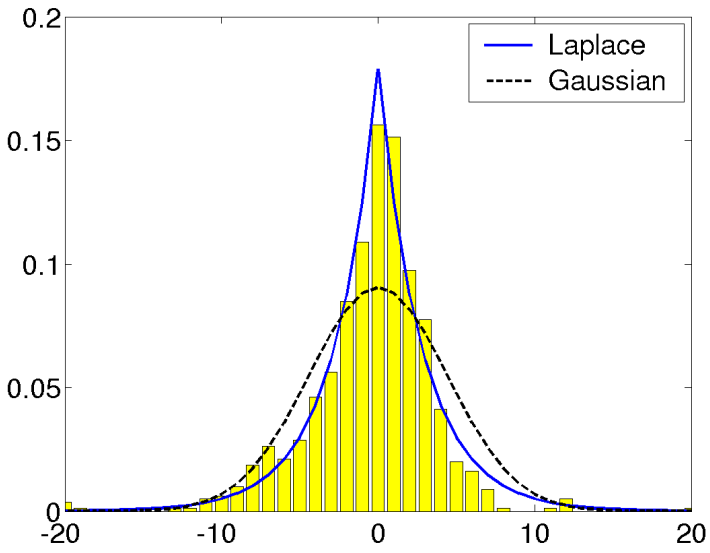
我们假定数据集 $\mathcal{D}$ 是从下面的模型采集得来：

$$y_i = f(\boldsymbol{x}_i) + \delta_i \tag{40}$$

其中 $f(\boldsymbol{x})$ 是潜在的未知函数，δ_i 来自一个独立同分布的随机噪声。给定测试数据 $\boldsymbol{x}$，我们希望估计出 $\Pr(y|\boldsymbol{x}, \mathcal{D})$，从而完成一些概率分布相关任务，比如区间估计：估计出

$$y \in [f(\boldsymbol{x}) - \Delta, f(\boldsymbol{x}) + \Delta]$$

的概率。我们设 $\hat{f}$ 为 SVR 对训练集 $\mathcal{D}$ 学习后得到的拟合函数，然后设 $\zeta = \zeta(\boldsymbol{x}) \equiv y - \hat{f}(\boldsymbol{x})$ 为预测误差。这里需要用交叉验证来减小偏差使得 ζ_i 更准确。根据实验得到下面的直方图：



libSVM 采用零均值的拉普拉斯分布来拟合误差：

$$p(z) = \frac{1}{2\sigma}\exp(-\frac{|z|}{\sigma}) \tag{41}$$

其中的参数 σ 可以利用极大似然法去估计：

$$\sigma = \frac{\sum_{i=1}^{l}|\zeta_i|}{l} \tag{42}$$

于是我们有

$$y = \hat{f}(\boldsymbol{x}) + z \tag{43}$$

其中 z 是满足参数为 σ 的拉普拉斯分布。

6 SMO 算法

SMO（Sequential Minimal Optimization）是求解 SVM 问题的高效算法之一，libSVM 采用的正是该算法。SMO 算法其实是一种启发式算法：先选择两个变量 α_i 和 α_j，然后固定其他参数，从而将问题转化成一个二变量的二次规划问题。求出能使此时目标值最优的一对 α_i 和 α_j 后，将它们固定，再选择两个变量，直到目标值收敛。这一部分以 C-SVC 作为研究对象，讨论 SMO 算法的过程。

6.1 朴素的 SMO 算法

这里的朴素是指变量选择方面，因为我们可以通过二重循环穷举选择，适用于变量少的情况。假设我们选择了变量 α_1 和 α_2，将其代入 (23) 式的目标函数：

$$\begin{aligned}
W(\alpha_1,\alpha_2) &= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) \\
&= \alpha_1+\alpha_2+\sum_{i=3}^{l}\alpha_i - \frac{1}{2}\big[\alpha_1\alpha_1 y_1 y_1 K(\boldsymbol{x}_1,\boldsymbol{x}_1) + 2\alpha_1 y_1\alpha_2 y_2 K(\boldsymbol{x}_1,\boldsymbol{x}_2 \\
&\quad + \alpha_2 y_2\alpha_2 y_2 K(\boldsymbol{x}_2,\boldsymbol{x}_2) + 2\alpha_1 y_1\sum_{i=3}^{l}\alpha_i y_i K(\boldsymbol{x}_1,\boldsymbol{x}_i) \\
&\quad + 2\alpha_2 y_2\sum_{i=3}^{l}\alpha_i y_i K(\boldsymbol{x}_2,\boldsymbol{x}_i) + \sum_{i=3}^{l}\sum_{j=3}^{l}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)\big]
\end{aligned} \tag{44}$$

为了表达方便，我们将 $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ 简写成 K_{ij}。根据 $y_i^2 = 1$ 的性质，我们将函数进一步化简成

$$\begin{aligned}
W(\alpha_1,\alpha_2) = \alpha_1+\alpha_2 - \frac{1}{2}\big[&\alpha_1^2 K_{11} + 2\alpha_1 y_1\alpha_2 y_2 K_{12} + \alpha_2^2 K_{22} \\
&+ 2\alpha_1 y_1\sum_{i=3}^{l}\alpha_i y_i K_{1i} + 2\alpha_2 y_2\sum_{i=3}^{l}\alpha_i y_i K_{2i}\big] + \text{Const}
\end{aligned} \tag{45}$$

其中 Const 是常数，在求极值点过程中可以忽略，即令 $f(\alpha_1,\alpha_2)\leftarrow f(\alpha_1,\alpha_2)-\text{Const}$。考虑后约束条件中后 $l-2$ 个变量被固定，引入常数 C：

$$\begin{aligned}\alpha_1y_1+\alpha_2y_2&=-\sum_{i=3}^{m}\alpha_iy_i=C\\ \alpha_1y_1&=C-\alpha_2y_2\\ \alpha_1&=y_1(C-\alpha_2y_2)\end{aligned}\tag{46}$$

将 α_1 用 α_2 的函数代入，从而将 $W(\alpha_1,\alpha_2)$ 变成 $W(\alpha_2)$：

$$\begin{aligned}W(\alpha_2)=y_1(C-\alpha_2y_2)+\alpha_2-\frac{1}{2}\big[&(C-\alpha_2y_2)^2K_{11}+2(C-\alpha_2y_2)\alpha_2y_2K_{12}+\alpha_2^2K_{22}+\\&2(C-\alpha_2y_2)\sum_{i=3}^{m}\alpha_iy_iK_{1i}+2\alpha_2y_2\sum_{i=3}^{m}\alpha_iy_iK_{2i}\big]\end{aligned}\tag{47}$$

对 $f(\alpha_2)$ 求导并令其为 0：

$$\begin{aligned}W'(\alpha_2)&=-y_1y_2+1-\frac{1}{2}\big[-2(C-\alpha_2y_2)y_2K_{11}+2Cy_2K_{12}-4\alpha_2K_{12}+2\alpha_2K_{22}\\&\quad-2y_2\sum_{i=3}^{m}\alpha_iy_iK_{1i}+2y_2\sum_{i=3}^{m}\alpha_iy_iK_{2i}\big]\\&=1-y_1y_2+Cy_2K_{11}-\alpha_2K_{11}-Cy_2K_{12}+2\alpha_2K_{12}-\alpha_2K_{22}+y_2\sum_{i=3}^{m}\alpha_iy_iK_{1i}\\&\quad-y_2\sum_{i=3}^{m}\alpha_iy_iK_{2i}\\&=0\end{aligned}\tag{48}$$

得到等式：

$$\begin{aligned}(K_{11}-2K_{12}+K_{22})\alpha_2&=1-y_1y_2+Cy_2K_{11}-Cy_2K_{12}+y_2\sum_{i=3}^{m}\alpha_iy_iK_{1i}-y_2\sum_{i=3}^{m}\alpha_iy_iK_{2i}\\&=y_2(y_2-y_1+CK_{11}-CK_{12}+\sum_{i=3}^{m}\alpha_iy_iK_{1i}-\sum_{i=3}^{m}\alpha_iy_iK_{2i})\end{aligned}\tag{49}$$

这里只要将 C 用 $-\sum_{i=1}^{3}\alpha_iy_i$ 替代，就可以得到 α_2 的解析解，但这样的计算代价是巨大的，同时对于计算机来说，迭代算法更加适合它的计算方法，因此我们将 C 设为 $\alpha_1^{\text{old}}y_1+\alpha_2^{\text{old}}y_2$，从而解出新的 α_2，也就是 α_2^{new}：

$$\begin{aligned}(K_{11}-2K_{12}+K_{22})\alpha_2^{\text{new}}=y_2\Big(&y_2-y_1+(\alpha_1^{\text{old}}y_1+\alpha_2^{\text{old}}y_2)K_{11}-\\&(\alpha_1^{\text{old}}y_1+\alpha_2^{\text{old}}y_2)K_{12}+\sum_{i=3}^{m}\alpha_iy_iK_{1i}-\sum_{i=3}^{m}\alpha_iy_iK_{2i}\Big)\end{aligned}\tag{50}$$

考虑支持向量机的表达式：

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{51}$$

所以我们就有

$$\begin{aligned} f(\boldsymbol{x}_1) &= \sum_{i=1}^{m} \alpha_i y_i K_{1i} + b \\ f(\boldsymbol{x}_2) &= \sum_{i=1}^{m} \alpha_i y_i K_{2i} + b \end{aligned} \tag{52}$$

用上式代替迭代式右端的两个求和部分：

$$\begin{aligned} (K_{11} - 2K_{12} + K_{22})\alpha_2^{\text{new}} = y_2 \Bigg(& y_2 - y_1 + (\alpha_1^{\text{old}} y_1 + \alpha_2^{\text{old}} y_2) K_{11} - \\ & (\alpha_1^{\text{old}} y_1 + \alpha_2^{\text{old}} y_2) K_{12} + f(x_1) - \alpha_1^{\text{old}} y_1 K_{11} - \alpha_2^{\text{old}} y_2 K_{12} - b \\ & - f(x_2) + \alpha_1^{\text{old}} y_1 K_{12} + \alpha_2^{\text{old}} y_2 K_{22} + b \Bigg) \\ = y_2 \Bigg(& f(x_1) - y_1 - \big(f(x_2) - y_2\big) + \alpha_2^{\text{old}} y_2 (K_{11} - 2K_{12} + K_{22}) \Bigg) \end{aligned} \tag{53}$$

这里对 $f(x_i) - y_i$ 有多种处理和理解方式，在《统计学习方法》中，它被设为 E_i，表示预测与实际的距离，理解为一种误差（在分类问题中没有误差这一概念）；但我更喜欢 libSVM 论文中的处理方式，也就是写成梯度的形式：

$$\begin{aligned} -y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j &= -y_i (\sum_{k=1}^{m} \alpha_k y_k y_i K_{ki} - 1) + y_j (\sum_{k=1}^{m} \alpha_k y_k y_j K_{kj} - 1) \\ &= \sum_{k=1}^{m} \alpha_k y_k K_{kj} + b - y_j - \sum_{k=1}^{m} \alpha_k y_k K_{ki} - b + y_i \\ &= -(f(\boldsymbol{x}_i) - y_i) + (f(\boldsymbol{x}_j) - y_j) \end{aligned} \tag{54}$$

并将该值设为 b_{ij}。此外设

$$a_{ij} = K_{ii} - 2K_{ij} + K_{jj} \tag{55}$$

从而我们能将 (53) 写成下面的形式：

$$\alpha_2^{\text{new}} = \alpha_2^{\text{old}} + y_2 \frac{b_{21}}{a_{21}}$$

从而参数更新公式：

$$\begin{cases} \alpha_1^{\text{new}} = \alpha_2^{\text{old}} + y_1 \dfrac{b_{12}}{a_{12}} \\ \alpha_2^{\text{new}} = \alpha_2^{\text{old}} - y_2 \dfrac{b_{12}}{a_{12}} \end{cases} \tag{56}$$

以上就是 SMO 迭代基本思路。但有一点值得注意，由于始终有 $0 \le \alpha_i \le C$ 的限制存在，当我们迭代时，是有可能会跑出这个范围，此时应该及时停下来。

6.2 变量选择

一个显然的事实是，选择不同的变量进行迭代会影响目标函数值趋向于最优的速度，那么如何选择变量成为我们接下来要讨论的话题。这一过程也被称作“工作集选择”（Working set selection），$\{i, j\}$ 被称作工作集。

6.2.1 变量选择思路

《统计学习方法》中提到了变量选择的基本思想，为笔者在阅读 libSVM 论文中的大量公式前对这个问题有大致的认识。我们希望选取违反 KKT 条件最严重的变量作为我们的 α_i。由于在找到最优解之前，总存在一个 α_i 不满足下面的 KKT 条件：

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i f(\boldsymbol{x}_i) \geq 1 \\ 0 < \alpha_i < C &\Leftrightarrow y_i f(\boldsymbol{x}_i) = 1 \\ \alpha_i = 0 &\Leftrightarrow y_i f(\boldsymbol{x}_i) \leq 1 \end{aligned} \tag{57}$$

我们希望 SMO 算法通过迭代使这样的 α_i 满足 KKT 条件，因此选择违法条件最严重的变量是好的选择。此外，对于上面三种情况，我们更偏向于选择第二种情况对应的变量，也就是没有到达边界（0 或 C），它会有更大的活动空间。

当已经选择好 α_i 后，我们希望找出更新时有最大变化的 α_j，也就是倾向于寻找绝对值最大的 $\dfrac{b_{ij}}{a_{ij}}$，这样迭代会更快。

6.2.2 基于一阶近似的变量选择

在前面选择第一个变量时，我们只提到了选择“违反 KKT 条件最严重”的样本点，那么我们需要找到一种方法来度量违反 KKT 条件的严重性。由此我们引入基于一阶近似（First order approximation）的变量选择法。

我们写出 C-SVC 的对偶问题的拉格朗日函数：

$$\mathcal{L}(\alpha, \lambda, \mu, \eta) = f(\alpha) - \sum_{i=1}^{m} \lambda_i \alpha_i + \sum_{i=1}^{m} \mu_i (\alpha_i - C) + \eta y^\top \alpha \tag{58}$$

其中 λ、μ、η 均为非负向量。如果 α 是原问题的解，那么它必然是 $\mathcal{L}$ 的有一个驻点，也就是梯度为零，整理后有：

$$\begin{aligned} \nabla f(\alpha) + \eta y &= \lambda - \mu \\ \lambda_i \alpha_i = 0, \mu_i (C - \alpha_i) = 0, &\alpha_i \geqslant 0, \mu_i \geqslant 0, i = 1, \cdots, m \end{aligned} \tag{59}$$

我们也不难求得 $f(\alpha)$ 的梯度为 $Q\alpha - e$。从而我们可以将上面的条件重写成：

$$\begin{aligned} \nabla f(\alpha)_i + \eta y_i &\geqslant 0 \qquad \text{if } \alpha_i < C \\ \nabla f(\alpha)_i + \eta y_i &\leqslant 0 \qquad \text{if } \alpha_i > 0 \end{aligned} \tag{60}$$

我们定义关于 α 的两个集合 I_{up} 和 I_{low}：

$$\begin{aligned} I_{\text{up}}(\alpha) &= \{t|\alpha_t < C, y_t = 1\} \cup \{t|\alpha_t > 0, y_t = -1\} \\ I_{\text{low}}(\alpha) &= \{t|\alpha_t < C, y_t = -1\} \cup \{t|\alpha_t > 0, y_t = 1\} \end{aligned} \tag{61}$$

可以推出这样的性质：$I_{\text{up}}(\alpha)$ 中的所有元素 i 都满足

$$-y_i \nabla f(\alpha)_i \leqslant \eta$$

而 $I_{\text{low}}(\alpha)$ 中的所有元素 i 都满足

$$-y_i \nabla f(\alpha)_i \geqslant \eta$$

α 是原问题的解当且仅当

$$m(\alpha) \leqslant M(\alpha) \tag{62}$$

其中

$$m(\alpha) = \max_{i \in I_{\text{up}}(\alpha)} -y_i \nabla f(\alpha)_i, M(\alpha) = \min_{i \in I_{\text{low}}(\alpha)} -y_i \nabla f(\alpha)_i \tag{63}$$

但我们前面提到，在求得解之前，这样的等式不会被满足，因此必然会存在这样一对 (i, j)，$i \in I_{\text{up}}(\alpha)$，$j \in I_{\text{low}}(\alpha)$ 但 $-y_i \nabla f(\alpha)_i > -y_j \nabla f(\alpha)_j$，那么我们称这对 (i, j) 为一个违反对（violating pair）。如果最大的一个违反对是 i 和 j，那么我们选择变量 α_i 和 α_j。当然我们也可以采用启发式方法，设置一个容忍值（tolerance）ε，如果在第 k 轮选择变量时有

$$m(\alpha^k) - M(\alpha^k) \leqslant \varepsilon \tag{64}$$

就停止算法。

我们重新审视“一阶近似”这个名称，在微积分中，一阶近似指的是用一阶导数（梯度）去近似函数值：

$$f(x + d) \approx f(x) + \nabla f(x)^\top d \tag{65}$$

而之所以称该算法基于一阶近似，是因为最大违反对 (i, j) 是一系列子问题

$$\begin{aligned} \text{Sub}(B) \equiv \min_{d_B} \quad & \nabla f(\alpha^k)_B^\top d_B \\ \text{subject to} \quad & y_B^\top d_B = 0 \\ & d_t \geqslant 0, \text{if } \alpha_t^k = 0, t \in B, \\ & d_t \leqslant 0, \text{if } \alpha_t^k = C, t \in B, \\ & -1 \leqslant d_t \leqslant 1, t \in B \end{aligned} \tag{66}$$

的最优解。这里的下标 B 指的是选定的两个变量对应的数据，比如 $B = \{1, 3\}$，那么 m 维向量 α 就变成 $[\alpha_1, \alpha_3]$。不难发现优化目标函数的由来：

$$\begin{aligned} f(\alpha^k + d) &\approx f(\alpha^k) + \nabla f(\alpha^k)^\top d \\ &= f(\alpha^k) + \nabla f(\alpha^k)_B^\top d_B \end{aligned}$$

正是一阶近似公式中的一阶近似项。

6.2.3 基于二阶近似的变量选择

libSVM 的 working set selection 是根据 second order information 来选择的，它在选择 i 采用的是前面提到的一阶近似方法，而在选择 j 时，不仅要求其与 i 构成违反对，还需要它能够最大程度减小目标函数。

函数的二阶近似：

$$f(x+d) = f(x) + \nabla f(x)^\top d + \frac{1}{2} d^\top \nabla^2 f(x) d \tag{67}$$

类似的，我们想寻找一系列优化问题

$$\begin{aligned} \mathrm{Sub}(B) \equiv \min_{d_B} \quad & \frac{1}{2} d_B^\top \nabla^2 f(\alpha)_{BB} d_B + \nabla f(\alpha^k)_B^\top d_B \\ \text{subject to} \quad & y_B^\top d_B = 0 \\ & d_t \geqslant 0, \text{if } \alpha_t^k = 0, t \in B, \\ & d_t \leqslant 0, \text{if } \alpha_t^k = C, t \in B. \end{aligned} \tag{68}$$

的最优解，考虑到我们已经确定了 i，那么子问题共 $m-1$ 个。形式化 i 和 j 的选取：

$$\begin{aligned} & i \in \arg\max_t \{-y_t \nabla f(\alpha^k)_t | t \in I_{\text{up}}(\alpha^k)\} \\ & j \in \arg\min_t \{\mathrm{Sub}(i,t) | t \in I_{\text{low}}(\alpha^k), -y_t \nabla f(\alpha^k)_t < -y_i \nabla f(\alpha^k)_i\} \end{aligned} \tag{69}$$

我们下面的任务就是尝试求解子问题 $\mathrm{Sub}(i,j)$

$$\begin{aligned} \mathrm{Sub}(B) &= \frac{1}{2} d_B^\top \nabla^2 f(\alpha)_{BB} d_B + \nabla f(\alpha^k)_B^\top d_B \\ &= \frac{1}{2} \begin{bmatrix} d_i & d_j \end{bmatrix} \begin{bmatrix} \frac{\nabla f(\alpha)_i}{\nabla \alpha_i} & \frac{\nabla f(\alpha)_i}{\nabla \alpha_j} \\ \frac{\nabla f(\alpha)_j}{\nabla \alpha_i} & \frac{\nabla f(\alpha)_j}{\nabla \alpha_j} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\alpha)_i & \nabla f(\alpha)_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} d_i & d_j \end{bmatrix} \begin{bmatrix} y_i^2 K_{ii} & y_i y_j K_{ij} \\ y_j y_i K_{ji} & y_j^2 K_{jj} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\alpha)_i & \nabla f(\alpha)_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \end{aligned} \tag{70}$$

这里令 $\hat{d}_i = y_i d_i$，$\hat{d}_j = y_j d_j$，又由 $y_B^\top d_B = 0$，我们得到 $d_i = -d_j$，从而我们进一步化简：

$$\begin{aligned} \mathrm{Sub}(B) &= \frac{1}{2}(K_{ii} - 2K_{ij} + K_{jj})\hat{d}_j^2 + [-y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j]\hat{d}_j \\ &= \frac{1}{2} a_{ij} \hat{d}_j^2 + b_{ij} \hat{d}_j \\ f(\hat{d}_j) &= \frac{1}{2} a_{ij} (\hat{d}_j + \frac{b_{ij}}{a_{ij}})^2 - \frac{b_{ij}^2}{2a_{ij}} \end{aligned} \tag{71}$$

从而对于每个 $\mathrm{Sub}(B)$，对应的最优值为

$$-\frac{b_{ij}^2}{2a_{ij}} = -\frac{[-y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j]^2}{2(K_{ii} - 2K_{ij} + K_{jj})} \tag{72}$$

j 的选取就可以改写成

$$j \in \arg\min_t \{-\frac{b_{it}^2}{a_{it}} | t \in I_{\text{low}}(\alpha^k), -y_t \nabla f(\alpha^k)_t < -y_i \nabla f(\alpha^k)_i\} \tag{73}$$

这也就是 LIBSVM 中的变量选取方法。

6.3 α 的更新与剪辑

6.3.1 更新

我们将更新公式 (56) 中 b_{ij} 用其实际含义替换，发现需要分类讨论：

- 如果 $y_i \neq y_j$：

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k + y_i \frac{b_{ij}}{a_{ij}} \\ &= \alpha_i^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \\ \alpha_j^{k+1} &= \alpha_j^k - y_j \frac{b_{ij}}{a_{ij}} \\ &= \alpha_j^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \end{aligned} \tag{74}$$

定义

$$\delta_{y_i \neq y_j} = \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \tag{75}$$

- 如果 $y_i = y_j$，类似的，我们有

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k - \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \\ \alpha_j^{k+1} &= \alpha_j^k + \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \end{aligned} \tag{76}$$

定义

$$\delta_{y_i = y_j} = \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \tag{77}$$

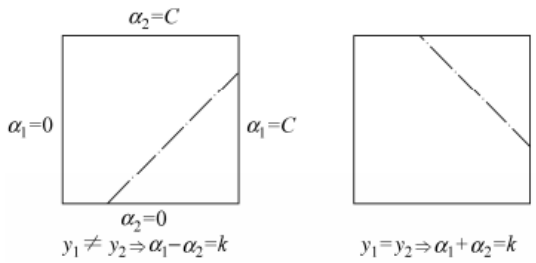
6.3.2 剪辑

我们在前面提到，由于 $\alpha_i \in [0, C]$，因此我们在更新时需要将 α 限制在该区间内，该步骤称为剪辑（Clipping），我们在6.1仅提到了这一步骤存在，这里进行一个深入讲解。

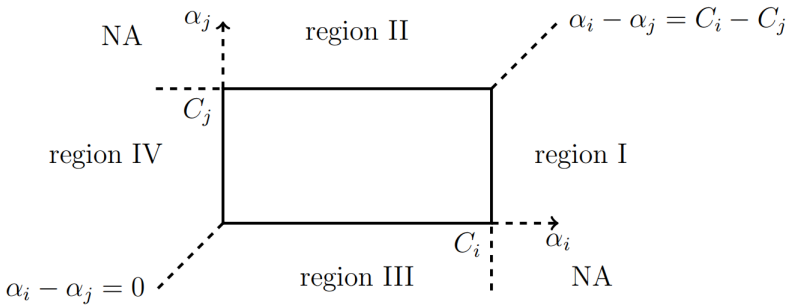
我们只讨论 $y_i \neq y_j$ 的情况，$y_i = y_j$ 的情况下讨论是类似的。由上面的更新公式，我们有：

$$\alpha_i^{k+1} - \alpha_i^k = \alpha_j^{k+1} - \alpha_j^k \tag{78}$$

此时两变量的状态 (α_i, α_j) 对应左图的虚线（图片摘自《统计学习方法》，图中是将 x_1 作为 x_i，x_2 作为 x_j）：



而更一般的情形如下图所示，$\alpha_i \in [0, C_i]$，每个变量的边界不一定相同，常常在类别不平衡的问题中使用：



显然 (α_i, α_j) 必须在图中的矩形中，而在更新的过程中，参数对存在跳出矩形区域的可能，也就是 Region I 到 Region IV 这四个不合理区域，需要将这些区域的点扳回矩形中；NA 则是不可达点，因为 $y_i \neq y_j$ 的情况下参数对只能向左下方或者右上方跳动。

以处于 Region I 的点为例，此时有

$$\begin{cases} \alpha_i > C_i \\ \alpha_i - \alpha_j > C_i - C_j \end{cases} \tag{79}$$

我们将 α_i 挪到矩形边缘中，同时保持 α_i 和 α_j 的关系：

$$\begin{cases} \alpha_i^{k+1} = C_i \\ \alpha_j^{k+1} = C_i - (\alpha_i^k - \alpha_j^k) \end{cases} \tag{80}$$

同理，对于 Region II，我们有

$$\begin{cases} \alpha_i^{k+1} = C_j + \alpha_i^k - \alpha_j^k \\ \alpha_j^{k+1} = C_j \end{cases} \tag{81}$$

Region III：

$$\begin{cases} \alpha_i^{k+1} = \alpha_i^k - \alpha_j^k \\ \alpha_j^{k+1} = 0 \end{cases} \tag{82}$$

Region IV:

$$\begin{cases} \alpha_i^{k+1} = 0 \\ \alpha_j^{k+1} = -(\alpha_i^k - \alpha_j^k) \end{cases} \tag{83}$$

对于 $y_i = y_j$ 的情况有类似的处理方式。

7 大规模数据下的 SVM

Thorsten Joachims，也是支持向量机软件包 SVM-Light 的作者，在《Making large-scale SVM learning practical》中提出在面对大规模数据时提高 SVM 训练效率的方案：

- 更有效和更高效的变量选择法；
- 不断地“收缩”问题规模；
- 计算上的改进：比如缓存机制的引入和梯度的增量式更新。

变量选择我们在前面已经提及；缓存机制涉及到操作系统的知识，我们放到后面讨论；因此这里主要探讨“收缩”和梯度更新技巧。

7.1 Shrink 方法

在解决支持向量机问题时，我们通常去解决其对偶问题，更具体地说，是去解一个长度为样本个数的 α 向量。但在大样本学习过程中，这显然会导致数据存储和运算量过大的问题。幸运的是，SVM 的解存在稀疏性，也就是最终模型仅与支持向量有关。Joachims 提出的 Shrinking 方法能够有效缩短 SVM 的训练时间，并将其应用到他开发的 SVM-light 中，该方法同时也被 libSVM 所采用。

7.1.1 问题引入

从 SVM 的对偶问题 OP1 的目标函数

$$W(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} - \boldsymbol{e}^\top\boldsymbol{\alpha}$$

可以看出解决该问题至少要为矩阵 Q 和 $\boldsymbol{\alpha}$ 提供存储空间，而 $Q_{ij} \equiv y_i y_j K(x_i, x_j)$，假设样本数为 1000，那么我们至少需要为其提供 $1000 \times 1000 \times 4 + 1000 \times 4$ 个字节，也就是约 4 MB 的存储空间，显然是非常不合理的。

Joachims 从算法上基于以下事实提出了 Shrinking 方法来缓解这一问题：

- 支持向量（SVs）的数量要比训练样本少得多；
- 许多支持向量对应的 α_i 的值都是其上界 C。

在硬间隔 SVM 中，所有 $\alpha_i > 0$ 对应的样本点 x_i 都是支持向量，它们都位于分类间隔上，反之也成立；而在软间隔 SVM 中，支持向量不一定全部分布在分类间隔上，在分类间隔中甚至是分类错误的向量也被称作支持向量，这些向量的特征就是其对应的 $\alpha_i = C$。

我们将样本向量分为三类：

1. X 类：支持向量，但 $0 < \alpha_i < C$；

2. Y 类：支持向量，但 $\alpha_i = C$；

3. Z 类：非支持向量，也就是 $\alpha_i = 0$.

因此我们将数据重排：

$$\alpha = \begin{bmatrix} \alpha_X \\ \alpha_Y \\ \alpha_Z \end{bmatrix} = \begin{bmatrix} \alpha_X \\ C\mathbf{1} \\ \mathbf{0} \end{bmatrix}, y = \begin{bmatrix} y_X \\ y_Y \\ y_Z \end{bmatrix}, Q = \begin{bmatrix} Q_{XX} & Q_{XY} & Q_{XZ} \\ Q_{YX} & Q_{YY} & Q_{YZ} \\ Q_{ZX} & Q_{ZY} & Q_{ZZ} \end{bmatrix} \tag{84}$$

我们从而重写 $W(\alpha)$：

$$\begin{aligned} W(\alpha) &= \frac{1}{2}\alpha^\top Q\alpha - C\mathbf{1}^\top\alpha \\ &= \frac{1}{2}\sum_{m\in\{X,Y,Z\}}\sum_{n\in\{X,Y,Z\}}\alpha_m^\top Q_{mn}\alpha_n - C\mathbf{1}^\top\alpha_X - C\mathbf{1}^\top\alpha_Y - C\mathbf{1}^\top\alpha_Z \\ &= \frac{1}{2}\sum_{m\in\{X,Y\}}\sum_{n\in\{X,Y\}}\alpha_m^\top Q_{mn}\alpha_n - C\mathbf{1}^\top\alpha_X - C\mathbf{1}^\top\alpha_Y \\ &= \frac{1}{2}\alpha_X^\top Q_{XX}\alpha_X + C\alpha_X^\top Q_{XY}\mathbf{1} + \frac{1}{2}C^2\mathbf{1}^\top Q_{YY}\mathbf{1} - C\alpha_X^\top\mathbf{1} - |Y|C \\ &= \frac{1}{2}\alpha_X^\top Q_{XX}\alpha_X + C\alpha_X^\top\big(Q_{XY}\mathbf{1} - \mathbf{1}\big) + \frac{1}{2}C^2\mathbf{1}^\top Q_{YY}\mathbf{1} - |Y|C \end{aligned} \tag{85}$$

考虑到后面两项为常数，我们重写对偶问题：

$$\begin{aligned} \min_{\alpha_X} \quad & \frac{1}{2}\alpha_X^\top Q_{XX}\alpha_X + C\alpha_X^\top\big(Q_{XY}\mathbf{1} - \mathbf{1}\big) \\ \text{subject to} \quad & \alpha_X^\top y_X + C\mathbf{1}^\top y_Y = 0 \\ & 0 \le \alpha_X \le C\mathbf{1} \end{aligned} \tag{86}$$

可以发现，问题的规模大幅度减小，矩阵和向量的维数数量级只由支持向量个数决定。这一过程便称为收缩（Shrinking）。

7.1.2 启发式方法

虽然我们可以通过 Shrinking 来缩小问题规模，但它基于我们已知 α_i 是属于 α_X、α_Y 或 α_Z，这对于算法是很难判定的。到目前为止，还不清楚该算法如何识别哪些样本可以消除，也就是对应的 α_i 为 0 或 C 的样本。我们希望在优化过程的早期找到一些条件，这些条件表明某些变量最终会达到一个界限。由于充分条件未知，采用基于拉格朗日乘子估计的启发式方法。

设 A 是当前满足 $\alpha_i \in (0, C)$ 的集合：

$$A = \{\alpha_i | 0 < \alpha < C, i = 1, \cdots, l\} \tag{87}$$

然后设估计值 λ^{eq}：

$$\lambda^{eq} = \frac{1}{|A|}\sum_{i\in A}\left[y_i - \sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right] \tag{88}$$

注意到我们可以用 λ^{eq} 作为 SVM 决策函数中的 bias，也就是 b。以此设置拉格朗日乘子，也就是 α_i 的上下界：

$$\begin{aligned}\lambda_i^{lo} &= y_i\left(\left[\sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)\right]+\lambda^{eq}\right)-1\\ \lambda_i^{up} &= -y_i\left(\left[\sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)\right]+\lambda^{eq}\right)+1\end{aligned} \tag{89}$$

给定一正整数 h，此时考虑 SMO 迭代过程的前 h 个循环，在这 h 个循环中，对于某个 i，如果都有 $\lambda_i^{lo}>0$ 且 $\lambda_i^{up}>0$（也可以用一个极小阈值 ε 代替 0），那么我们就有信心将其删除。也就是说 α_i 已经是最优的，它们可以被固定，从而不需要对其梯度等值进行计算。

由于启发式算法没有定理去证明合理性，必然会存在删错了的情况。因此在 (86) 收敛后，对被删除变量的最优条件进行检查；如有必要，则会在这一次迭代中重新进行一次优化。

7.2 梯度重构策略

Joachims 称之为梯度的增量式更新（Incremental updates of the gradient），而在 libSVM 中被称为梯度重构（Gradient reconstruction）。

7.2.1 一个例子

在线性回归中，我们要解决的是下面的优化问题：

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w})=\frac{1}{2}\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\|^2 \tag{90}$$

该问题固然是有解析解的，但倘若我们用梯度下降法，给定步长 η，在每轮迭代中都有下面的操作：

$$\boldsymbol{w}\leftarrow\boldsymbol{w}-\eta\frac{\partial f}{\partial\boldsymbol{w}} \tag{91}$$

也就是

$$\boldsymbol{w}\leftarrow\boldsymbol{w}-\eta\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}) \tag{92}$$

如果不采用任何优化方法的话，我们每次都要计算一次 $\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})$；而实际上相邻两次更新之间梯度变化量是一个很简单的值：

$$\begin{aligned}\Delta\nabla f(\boldsymbol{w}) &= \nabla f(\boldsymbol{w}+\Delta\boldsymbol{w})-\nabla f(\boldsymbol{w})\\ &= \boldsymbol{X}^\top\big(\boldsymbol{X}(\boldsymbol{w}+\Delta\boldsymbol{w})-\boldsymbol{y}\big)-\boldsymbol{X}^\top\big(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\big)\\ &= \boldsymbol{X}^\top\boldsymbol{X}\Delta\boldsymbol{w}\end{aligned} \tag{93}$$

此外，要是我们能将 $\boldsymbol{X}^\top\boldsymbol{X}$ 存储，那么花费在梯度计算上的时间会更少。这就是梯度的增量式更新带来计算负担减少的一个例子。

7.2.2 libSVM 中的梯度重构

libSVM 继承了 Joachims 在 SVM-Light 中使用的增量更新思想，分别对工作集和非工作集的梯度进行更新。对工作集中变量对应的梯度的更新：

```
double delta_alpha_i = alpha[i] - old_alpha_i;
double delta_alpha_j = alpha[j] - old_alpha_j;
for (int k = 0; k < active_size; k++) {
   G[k] += Q_i[k] * delta_alpha_i + Q_j[k] * delta_alpha_j;
}
```

但我们仍然需要这些 inactive 的参数对应的梯度，也就是全部的 $\nabla f(\boldsymbol{x})$，为了减少梯度重构的开销，libSVM 选择在迭代中维护一个向量 $\bar{G} \in \mathbb{R}^l$：

$$\bar{G}_i = C \sum_{j:\alpha_j=C} Q_{ij}, i = 1, \cdots, l \tag{94}$$

从而对于不属于 active 集合的变量 α_i，我们有

$$\begin{aligned} \nabla f(\boldsymbol{\alpha})_i &= \sum_{j=1}^{l} Q_{ij}\alpha_j - 1 \\ &= \sum_{\alpha_j=0} Q_{ij} \cdot 0 + \sum_{\alpha_j=C} CQ_{ij} + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j - 1 \\ &= C \sum_{\alpha_j=C} Q_{ij} + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j - 1 \\ &= \bar{G}_i + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j - 1 \end{aligned} \tag{95}$$

实现通过提前计算 $\bar{G}$ 加快计算梯度：

```
{
  bool ui = is_upper_bound(i);
  bool uj = is_upper_bound(j);
  update_alpha_status(i);
  update_alpha_status(j);
  int k;
  if (ui != is_upper_bound(i)) {
    Q_i = Q.get_Q(i, l);
    if (ui)
      for (k = 0; k < l; k++)
        G_bar[k] -= C_i * Q_i[k];
    else
      for (k = 0; k < l; k++)
        G_bar[k] += C_i * Q_i[k];
  }

  if (uj != is_upper_bound(j)) {
    Q_j = Q.get_Q(j, l);
```

```
      if (uj)
        for (k = 0; k < l; k++)
          G_bar[k] -= C_j * Q_j[k];
      else
        for (k = 0; k < l; k++)
          G_bar[k] += C_j * Q_j[k];
  }
}
```

这里对更新进行限制：只有状态发生变换的变量，才能参与 $\bar{G}$ 的更新，这是为了减少循环次数。如果 ui 是 false，也就是 α_i 从 active 变成 inactive，对应的就是将对应的增量加入 $\bar{G}$，反之就是从 $\bar{G}$ 中对应 α_i 的增量删除。

参考文献

[1] 崔萌, and 张春雷. ”LIBSVM, LIBLINEAR, SVMmuticlass 比较研究.” 电子技术 6 (2015): 1-5.

[2] Boyd, Stephen, Stephen P. Boyd, and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2004.