LIBLINEAR 理论解读

Welt https://welts.xyz

2022 年 1 月 26 日

目录

摘要

支持向量机 (Support Vector Machine, SVM) 是一种流行的二元分类器，并延伸到多分类、回归甚至是异常检测模型。LIBSVM 和 LIBLINEAR 是当前比较成熟的 SVM 软件包，其中 LIBLINEAR 能解决更多的线性问题，且在稀疏数据上表现更好。本文将对 LIBLINEAR 涉及到的 SVM 理论进行解读，为后面阅读 C++ 源码扫清障碍。

关键词:SVM、LIBLINEAR、优化算法

1 SVM，以及 LIBLINEAR

在机器学习中，支持向量机是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM 训练算法创建一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM 模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别[1]。

给定特征-标签数据集 $\{\boldsymbol{x}_j, y_j\}_{j=1}^{l}, \boldsymbol{x}_j \in R^n, y_j \in \{-1, +1\}$，SVM 优化问题标准型如下

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_j + b) \geq 1, j = 1, \cdots, l \end{aligned} \tag{1}$$

设满足上述约束条件的最优解为 $\boldsymbol{w}^*, \boldsymbol{b}^*$，得到分类问题的决策函数

$$f(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^{*T}\boldsymbol{x} + b^*) \tag{2}$$

LIBLINEAR 将 SVM 基本模型扩展到一般线性问题:

$$\min_{\boldsymbol{w}} \quad \frac{1}{p}\|\boldsymbol{w}\|_p^p + C\sum_{j=1}^{l}\xi(\boldsymbol{w}; \boldsymbol{x}_j, y_j) \tag{3}$$

$p \in \{1, 2\}$, 也就是向量的一范数, 二范数的平方; ξ 是损失函数, 可以是铰链损失 (Hinge loss), 铰链损失平方和指数损失。所以 LIBLINEAR 求解的是带有 L1 或 L2 正则化的线性模型，这些线性模型可以采用不同的损失函数。

记号约定 (Notation): 这里对文中涉及到的标记做出规定，数据集的数目为 l，数据的特征数为 n。

2 优化知识基础

这一部分将介绍 LIBLINEAR 中使用到的优化知识，包括拉格朗日对偶性、坐标下降法、牛顿法、共轭梯度法、线搜索方法以及置信域方法。这里略去高等数学和概率统计的知识，默认读者已经掌握。

2.1 拉格朗日对偶性

2.1.1 原始问题

我们将

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \le 0, i = 1, \cdots, m \\ & h_j(x) = 0, j = 1 \cdots, n \end{aligned} \tag{4}$$

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:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{n} \mu_i h_j(x), \quad \lambda_i \ge 0, \mu_j \in \mathbb{R} \tag{5}$$

定义函数 θ_P:

$$\theta_P(x) = \begin{cases} f(x) & x\text{满足约束} \\ +\infty & \text{else} \end{cases} \tag{6}$$

可以证明对任意 x，我们有

$$\begin{aligned} \theta_P(x) &= \max_{\lambda,\mu:\lambda_i \ge 0} \mathcal{L}(x, \lambda, \mu) \\ \min_x \theta_P(x) &= \min_x \max_{\lambda,\mu:\lambda_i \ge 0} \mathcal{L}(x, \lambda, \mu) \end{aligned} \tag{7}$$

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

$$p^* = \min_x \theta_P(x) \tag{8}$$

称为**原始问题的值**。

2.1.2 对偶问题

我们先设函数

$$\theta_D(\lambda, \mu) = \min_x \mathcal{L}(x, \lambda, \mu) \tag{9}$$

然后对其求极大，也就是“极大极小问题”：

$$\max_{\lambda,\mu} \min_x \mathcal{L}(x, \lambda, \mu) = \max_{\lambda,\mu} \theta_D(x) \tag{10}$$

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^*。

2.1.3 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:

$$d^* \le p^* \tag{11}$$

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[2]:

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0 \\ \lambda_i^* g_i(x^*) = 0, i = 1, \cdots, m \\ g_i(x^*) \leqslant 0, i = 1, \cdots, m \\ \lambda_i \geqslant 0, i = 1, \cdots, m \\ h_j(x^*) = 0, i = 1, \cdots, n \end{cases} \tag{12}$$

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 $g_i(x^*) = 0$。

LIBLINEAR 中的一部分问题都提供了原问题求解器和对偶问题求解器，在一些问题下，求解原问题会比求对偶问题更简单，而在某些情况下对偶问题会让求解更容易。

2.2 坐标下降法

坐标下降法属于一种非梯度优化的方法，它在每步迭代中沿一个坐标的方向进行线性搜索（线性搜索是不需要求导数的），通过循环使用不同的坐标方法来达到目标函数的局部极小值。

比如我们想求凸函数 $f(x_1, x_2, x_3)$ 的极小值，坐标下降法先固定 x_2 和 x_3，然后求

$$z_1 = \arg\min_z f(x_1 + z, x_2, x_3) \tag{13}$$

然后将当前坐标由 (x_1, x_2, x_3) 更新为 $(x_1 + z_1, x_2, x_3)$，接着固定 $x_1 + z_1$ 和 x_3，继续求

$$z_2 = \arg\min_z f(x_1 + z_1, x_2 + z, x_3) \tag{14}$$

同理，求得 z_2 之后，固定前两个坐标，求

$$z_3 = \arg\min_z f(x_1 + z_1, x_2 + z_2, x_3 + z) \tag{15}$$

显然坐标下降可以使函数值一直下降，符合优化的目标。可以写出算法伪代码 (算法 1):

Algorithm 1 Coordinate Descent Algorithm

Input: Convex function f, problem size n

Output: Global optimal $\boldsymbol{x}^*$

Start with any initial $\boldsymbol{x}^0$.

for $k = 0, 1, \cdots$ **do**

  for $i = 1, 2, \cdots, n$ **do**

    Solve problem:

$$z \leftarrow \arg\min_z f(x_1^k, \cdots, x_{i-1}^k, x_i^k + z, x_{i+1}^k, \cdots, x_n^k)$$

    $x_i^k \leftarrow x_i^k + z$

  end for

end for

2.3 牛顿法

牛顿法是求解无约束优化问题常用的迭代算法，拥有下降速度快的特点[3]。考虑 $f(\boldsymbol{x})$ 在初始点 $\boldsymbol{x}_0$ 附近的二阶泰勒展开:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^T\nabla^2 f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0) \tag{16}$$

现在考虑 $\nabla f(\boldsymbol{x}) = 0$，也就是最优点的情形:

$$\begin{aligned} \nabla f(\boldsymbol{x}) &= \nabla f(\boldsymbol{x}_0) + \nabla^2 f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0) = 0 \\ \boldsymbol{x} - \boldsymbol{x}_0 &= -\nabla^2 f(\boldsymbol{x}_0)^{-1}\nabla f(\boldsymbol{x}_0) \end{aligned} \tag{17}$$

根据上式进行迭代，直到 $\boldsymbol{x}$ 正是局部最优值，这就是牛顿法:

Algorithm 2 Newton Method

Input: Convex function f

Output: Global optimal $\boldsymbol{x}^*$

Start with any initial $\boldsymbol{x}^0$.

for $k = 0, 1, \cdots$ **do**

$\boldsymbol{s} \leftarrow -\nabla^2 f(\boldsymbol{x}^k)^{-1}\nabla f(\boldsymbol{x}^k)$

$\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \boldsymbol{s}$

end for

2.4 共轭梯度法

共轭梯度法一开始用于求解二次优化问题[4]:

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c \tag{18}$$

其中 A 是对称正定矩阵，很容以得到上述问题的解析解:

$$\boldsymbol{x} = A^{-1}\boldsymbol{b} \tag{19}$$

因此该问题也可以解决线性方程组 $A\boldsymbol{x} = \boldsymbol{b}$,由此扩展到求解牛顿方向,也就是 $\nabla^2 f(\boldsymbol{x}_0)^{-1}\nabla f(\boldsymbol{x}_0)$。一种朴素的迭代方法是最速下降法，也就是从当前点出发，寻找一个方向 $\boldsymbol{p}$，然后在该方向上寻找能够使目标函数下降最大的步长 α，也就是

$$\min_{\alpha} \quad f(\boldsymbol{x} + \alpha\boldsymbol{p}) \tag{20}$$

可以证明，相邻两次迭代的下降方向是正交的，如图 1所示。

在该问题上应用最速下降法有两个缺陷:

- 迭代步数过大，收敛速度慢，而在极端情况下 (条件数很大，也就是 A 的最大特征值和最小特征值的比值很大，此时椭球在某一个维度上会很扁)，收敛速度会更慢;
- 当 $\|\boldsymbol{r}_k\|$ 很小时，计算会出现不稳定。

共轭梯度法 (Conjugate Gradient) 是一种求解大型稀疏对称正定方程组十分有效的方法。我们仍选择一组搜索方向 $p_0, p_1, \cdots$，但它们不再是具有正交性的 $r_0, r_1, \cdots$，而是满足 A-共轭的性质[5]:

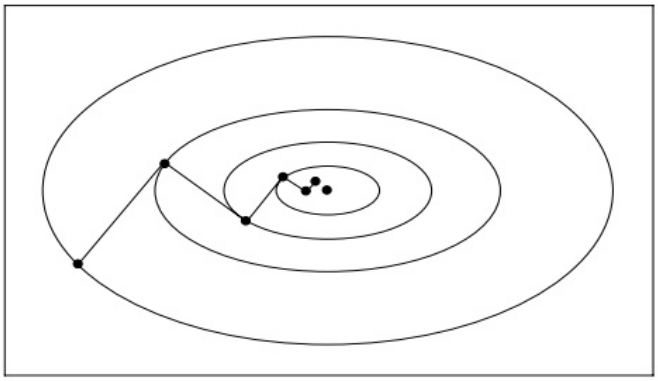

图 1: 最速下降法示意图

Definition 1. 设 A 是对称正定矩阵，若 R^n 中的向量组 $\{p_0, \cdots, p_m\}$ 满足

$$\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0, \forall i, j \in [0, m], i \neq j$$

那么我们称该向量组为一个 A-共轭向量组或 A-正交向量组。

可以证明，沿着共轭向量组中元素作为下降方向，n 阶问题至多下降 n 次就能得到精确解。这是因为算法保证当前点在所有已探索方向上都是最优的，因此之后的下降方向不会与之前重复，反观最速下降法，则无法做到这一点，导致迭代次数增加。共轭梯度法如算法 3所示。

Algorithm 3 Conjugate Descent

Input: SPD matrix A, vector $\boldsymbol{b}$

Output: $\boldsymbol{x}$ satisfies $A\boldsymbol{x} = \boldsymbol{b}$

$\boldsymbol{x}_0 \leftarrow 0$

$\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$

$\boldsymbol{p}_0 \leftarrow \boldsymbol{r}_0$

$k \leftarrow 0$

repeat

$\alpha_k \leftarrow \dfrac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$

$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$

$\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$

$\beta_k \leftarrow \dfrac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$

$\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$

$k \leftarrow k + 1$

until $\boldsymbol{r}_{k+1} \leq \epsilon$

$\boldsymbol{x}_{k+1}$ 就是问题的解。

2.5 线搜索

我们回顾式(20)，在给定搜索方向后，我们希望在该方向上找到让目标函数下降最大的步长 α。像这样沿着射线 $\{\boldsymbol{x} + \alpha \boldsymbol{p} | \alpha \geq 0\}$ 进行搜索的方法称作**精确直线搜索**[6]。当求解式(20)中

的单变量优化问题的成本，同计算搜索方向的成本相比，比较低时，适合进行精确直线搜索。般情况下可以有效的求解该优化问题，特殊情况下可以用解析方法确定其最优解。

实践中主要采用非精确直线搜索方法: 沿着射线 $\{x + t\delta x | t \geq 0\}$ 近似优化 f 确定步长，甚至只要有" 足够的" 减少即可。业已提出了很多非精确直线搜索方法，其中回溯方法既非常简单又相当有效。它取决于满足 $0 < \alpha < 0.5, 0 < \beta < 1$ 的两个常数 α 和 β:

Algorithm 4 Backtracking Line Search

Input: Function f, descent direction $\Delta \boldsymbol{x}$ at $\boldsymbol{x}$,$\alpha \in (0, 0.5), \beta \in (0, 1)$

Output: Descent stepsize t

$t \leftarrow 1$

while $f(\boldsymbol{x} + t\Delta \boldsymbol{x}) > f(\boldsymbol{x}) + \alpha t \nabla f(\boldsymbol{x})^T \Delta \boldsymbol{x}$ **do**

$t \leftarrow \beta t$

end while

2.6 置信域方法

置信域方法 (Trust-region Methods) 又称为信赖域方法，它是一种最优化方法，能够保证最优化方法总体收敛[7]。它解决的是无约束的凸优化问题:

$$\min_{\boldsymbol{x} \in R^n} \quad f(\boldsymbol{x}) \tag{21}$$

其中 f 是定义在 R^n 上的二阶可微函数。对于当前所在的位置 $\boldsymbol{x}_k$，给定一个距离 Δ_k，我们想在以 $\boldsymbol{x}_k$ 为球心，Δ_k 为半径的高维球体中找到一个下降幅度最大的点，也就是

$$\begin{aligned} \min_{pmbs} \quad & f(\boldsymbol{x}_k + \boldsymbol{s}) - f(\boldsymbol{x}_k) \\ \text{s.t.} \quad & \|\boldsymbol{s}\| \leq \Delta_k \end{aligned} \tag{22}$$

这样的问题还是很复杂，因此我们利用泰勒展开，得到一个近似问题:

$$\begin{aligned} \min_{\boldsymbol{s}} \quad & q_k(\boldsymbol{s}) = \nabla f(\boldsymbol{x}_k)^T \boldsymbol{s} + \frac{1}{2} \boldsymbol{s}^T \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{s} \\ \text{s.t.} \quad & \|\boldsymbol{s}\| \leq \Delta_k \end{aligned} \tag{23}$$

这里的黑塞矩阵 $\nabla^2 f(\boldsymbol{x}_k)$ 由于计算复杂的原因，常常会用另一个矩阵进行替代，所以在有的资料上写作 B_k；此外，我们将 Δ_k 称作 "置信域半径"。

优化问题(23)便是置信域方法的一个模型子问题（因该问题是二次模型而得名），我们将目标函数设为 $q_k(\boldsymbol{s})$，也就是第 k 次子问题。通过迭代求解子问题，最终得到目标解。可以知道，随着 $\boldsymbol{s}$ 的增大，由泰勒展开生成的二次模型会越来越偏移原来的目标函数。

设第 k 个子问题解出来的 $\boldsymbol{s}$ 为 $\boldsymbol{s}_k$，我们定义 r_k:

$$r_k = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{q_k(\boldsymbol{s}_k)} \tag{24}$$

分子是真实下降量，分母是二次模型的下降量，也就是预测下降量，其比值用于衡量二次模型与目标函数的近似程度。如果 rk 够大，也就是真实下降量够大，我们会接受这个 $\boldsymbol{s}_k$，令 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$，进行下一轮迭代，否则停留在原处。注意两种情况都是要修改置信域半径 Δ_k 的。r_k 太小，表明二次模型与 $f(\boldsymbol{x})$ 的差距太大，这是因为 Δ_k 太大导致的，我们需要将置信

域大幅度缩小；r_k 太大，说明二次模型的估计过于保守，我们需要扩大置信域；r_k 不大不小，本着准确估计的原则，我们对置信界适当缩小。我们给出置信域算法的具体步骤，如算法 5所示:

Algorithm 5 Trust Region Algorithm

Input: Convex and twice differentiable function f, $\epsilon > 0$, $\bar{\Delta}, \Delta_0 \in (0, \bar{\Delta})$,$0 < \eta_1 \leq \eta_2 < 1$, $0 < \gamma_1 < 1 < \gamma_2$

Output: Global optimal $\boldsymbol{x}^*$

$k \leftarrow 0$

while $\|\nabla f(\boldsymbol{x}_k)\| > \epsilon$ **do**

Solve subproblem (23) to obtain $\boldsymbol{s}_k$.

Calculate r_k through (24)

if $r_k \geq \eta_1$ **then**

$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \boldsymbol{s}_k$

else

$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k$

end if

Adjust trust region using following rules:

$$\begin{cases} \Delta_{k+1} \in [0, \gamma_1 \Delta_k] & \text{if } r_k < \eta_1 \\ \Delta_{k+1} \in [\gamma_1 \Delta_k, \Delta_k] & \text{if } r_k \in [\eta_1, \eta_2) \\ \Delta_{k+1} \in [\Delta_k, \min\{\gamma_2 \Delta_k, \bar{\Delta}\}] & \text{if } r_k \geq \eta_2 \end{cases}$$

$k \leftarrow k + 1$

end while

3 不同模型的优化算法

这一部分会介绍 LIBLINEAR 所支持模型问题以及对应的优化算法。LIBLINEAR 目前支持 12 种模型，**除了用于多分类的 Crammer-Singer 模型和用于异常检测的 One-class SVM 模型外**，剩下的 10 种模型可归纳为 “带某种正则化的某种损失函数的分类/回归模型的原/对偶问题”，具体如下表所示

问题类型 / 模型	L1 正则化的原问题	L2 正则化的原问题	L2 正则化的对偶问题
L1-SVM			√
L2-SVM	√	√	√
LR	√	√	√
L1-SVR			√
L2-SVR		√	√

表 1: LIBLINEAR 可解决的问题

五种模型中，前三种是分类模型，后两种是回归模型，相同任务的模型之间的区别在于采

用了不同的损失函数，后面会详细介绍。

3.1 求解带 L2 正则的 SVM 对偶问题

3.1.1 问题原型与重构

我们先从最基本的 SVM 分类问题开始，原问题:

$$\begin{aligned} \min_{\boldsymbol{w},b\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i \\ & \xi_i > 0, i = 1, \cdots, l \end{aligned} \tag{25}$$

其对偶问题为

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, \cdots, l \end{aligned} \tag{26}$$

其中 $Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$，$\boldsymbol{e}$ 是全一向量。在对偶问题中，约束条件 $\boldsymbol{y}^T\boldsymbol{\alpha} = 0$ 是很讨厌的。该约束是由问题的拉格朗日函数对偏置项 b 求导为零而产生的。LIBLINEAR 通过对数据进行增广，消去了线性约束:

$$\boldsymbol{x}_j^T \leftarrow [\boldsymbol{x}_j^T, 1] \quad \boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b] \tag{27}$$

从而原问题变成最小化带 L2 正则的铰链损失函数:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{j=1}^{l}\max(0, 1 - y_j\boldsymbol{w}^T\boldsymbol{x}_j) \tag{28}$$

除了铰链损失，铰链平方损失也可作为损失函数:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{j=1}^{l}\max(0, 1 - y_j\boldsymbol{w}^T\boldsymbol{x}_j)^2 \tag{29}$$

为了区分，我们称问题(28)为 L1-SVM，问题(29)为 L2-SVM。注意这里的 L1 和 L2 表示损失函数的种类，而不是正则化选项。

由于等式约束被消去，L1-SVM 和 L2-SVM 的对偶问题也被重构:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\bar{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq U, i = 1, \cdots, l \end{aligned} \tag{30}$$

其中 $\bar{Q} = Q + D$，D 是一对角矩阵，对于不同的 SVM，D 和 U 的定义也不相同:

3.1.2 对偶坐标下降法

LIBLINEAR 中使用对偶坐标下降法 (Dual Coordinate Descent Method, DCDM)[8] 来求解 L2 正则的 L1-SVM 和 L2-SVM 的对偶问题。正如我们在前面提到的坐标下降法，给定当前待优化变量 $\boldsymbol{\alpha}$，令

SVM 类型	D_{ii}	U
L1-SVM	0	C
L2-SVM	$1/2C$	∞

表 2: 不同 SVM 下 D 与 U 的设置[9]

$$\boldsymbol{e}_i = [\underbrace{0, \cdots, 0}_{i-1}, 1, 0, \cdots, 0]^T \tag{31}$$

优化 α_i 实际上是求解

$$\min_d \quad f(\boldsymbol{\alpha} + d\boldsymbol{e}_i) = \frac{1}{2}Q_{ii}d^2 + \nabla_i f(\boldsymbol{\alpha})d + \text{constant} \tag{32}$$

在无约束前提下，易求出最优的 d:

$$d^* = -\frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}} \tag{33}$$

考虑 α_i 被限制在 $[0, U]$ 内，因此更新规则应该为

$$\alpha_i \leftarrow \min(\max(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}, 0), U) \tag{34}$$

考虑到 L2-SVM 的 α_i 无上界，因此可简化为

$$\alpha_i \leftarrow \max(\alpha_i - \frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}, 0) \tag{35}$$

算法在每次求 d^* 时都要求梯度，也就是

$$\begin{aligned} \nabla_i f(\boldsymbol{\alpha}) &= (Q\boldsymbol{\alpha})_i - 1 \\ &= \sum_{j=1}^{l} Q_{ij}\alpha_j - 1 \\ &= \sum_{j=1}^{l} y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \alpha_j - 1 \end{aligned} \tag{36}$$

因为该算法考虑的是大规模数据下的 SVM，也就是数据量 l 和特征数 n 很大的情况，因此存储极大的 $Q_{l\times l}$ 是不合理的，这导致 Q_{ij} 无法直接存取，每次只能通过计算得到。观察上式，我们发现求 $\nabla_i f(\boldsymbol{\alpha})$ 的开销是 $O(ln)$。为了减少开销，算法选择保存模型判别函数的参数 $\boldsymbol{w}$:

$$\boldsymbol{w} = \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j \tag{37}$$

这样每次求梯度就可以减少计算量，削减到 $O(n)$:

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 \tag{38}$$

因为 $\boldsymbol{w}$ 是关于 $\boldsymbol{\alpha}$ 的变量，因此需要我们动态维护。我们设 $\bar{\boldsymbol{\alpha}}_i$ 为更新前的 α_i，在对 α_i 进行优化后，对 $\boldsymbol{w}$ 的更新

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i \tag{39}$$

显然这里的开销也是 $O(n)$。

这里给出 DCDM 算法 (算法 6)。算法中的"α_i can be changed" 意思是解出来的 $d^* \neq 0$,

Algorithm 6 Dual Coordinate Descent Method for Linear SVM

Start with any initiaal $\boldsymbol{\alpha}$.
$\boldsymbol{w} \leftarrow \sum_i y_i \alpha_i \boldsymbol{x}_i$
while $\boldsymbol{\alpha}$ is not optimal **do**
  $G \leftarrow y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 + D_{ii}\alpha_i$
  if α_i can be changed **then**
    $\alpha_i \leftarrow \min(\max(\alpha_i - \dfrac{G}{Q_{ii}}, 0), U)$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$
  end if
end while

否则 $\boldsymbol{\alpha}$ 会在原地踏步。事实上存在三种原地踏步的情况:

- $\alpha_i = 0$，函数在 $[0, U]$ 上递增，即 $\nabla_i f(\boldsymbol{\alpha}) \geq 0$;
- α_i 已经是最优的，即 $\nabla_i f(\boldsymbol{\alpha}) = 0$;
- $\alpha_i = U$，函数在 $[0, U]$ 上递减，即 $\nabla_i f(\boldsymbol{\alpha}) \leq 0$.

如图 2所示。这些情况下，解出来的 $d^* = 0$，也就是 α_i 不会更新。

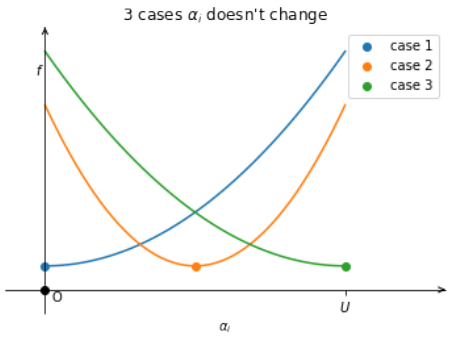

图 2: $d^* = 0$ 的三种情况

3.2 求解带 L2 正则的 LR 原问题

3.2.1 问题原型

先来看带 L2 正则化的对率回归 (Logistic Regression, LR):

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \log(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)) \tag{40}$$

这是一个无约束优化问题。求梯度:

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + C\sum_{i=1}^{l} (\sigma(y_i \boldsymbol{w}^T \boldsymbol{x}_i) - 1) y_i \boldsymbol{x}_i \tag{41}$$

其中

$$\sigma(x) = (1 + \exp(-x))^{-1} \tag{42}$$

其实就是 Sigmoid 函数。再求二阶微分 (Hessian 矩阵):

$$\nabla^2 f(\boldsymbol{w}) = I + CX^T DX \tag{43}$$

D 是一对角矩阵，对角元满足:

$$D_{ii} = \sigma(y_i \boldsymbol{w}_i^T \boldsymbol{x}_i)(1 - \sigma(y_i \boldsymbol{w}_i^T \boldsymbol{x}_i)) \tag{44}$$

这是因为

$$\begin{aligned} f(\boldsymbol{w}) &= \log(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})) \\ \frac{\partial f}{\partial \boldsymbol{w}} &= \frac{1}{1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})} \frac{\partial}{\partial \boldsymbol{w}} (1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})) \\ &= -y_i \boldsymbol{x}_i \frac{\exp(-y_i \boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})} \\ &= (\sigma(y_i \boldsymbol{w}^T \boldsymbol{x}) - 1) y_i \boldsymbol{x}_i \end{aligned} \tag{45}$$

显然 Hessian 矩阵是正定的，因此上述的优化问题是强凸的。

Definition 2. *目标函数在 S 上是强凸的，这是指存在 $m > 0$ 使得*

$$\nabla^2 f(x) \succeq mI$$

强凸性保证了上述优化问题必有唯一的全局最优解，证明可参考原论文《Trust region Newton method for large-scale logistic regression》[11]。

3.2.2 牛顿法的缺陷

根据我们前面提到的牛顿法，我们需要解

$$\nabla^2 f(\boldsymbol{w}^k) \boldsymbol{s}^k = -\nabla f(\boldsymbol{w}^k) \tag{46}$$

因为 Hessian 矩阵始终可逆，保证了牛顿法的可行性，但在该问题上，存在两个问题:

- 序列 $\{\boldsymbol{w}^k\}$ 不一定会收敛到一个最优解，甚至不能保证目标函数是递减的;
- 尽管我们假设数据集 X 是稀疏的，但 $X^T DX$ 还是比较稠密。黑塞矩阵将会变得很大而难以存储，因此求解上面的线性方程组的方法需要认真考虑.

通过调整牛顿步径，我们可以解决第一个问题，通常采用两种方法:线搜索和置信域；而对于第二个问题，解线性方程组 (也称作线性系统, Linear systems) 也有两种方法: 直接法和迭代法。直接法的代表就是高斯消元法，Jacobi 法和共轭梯度法都属于迭代法。迭代法最主要的操作就是计算矩阵（也就是黑塞矩阵）和向量的乘积

$$\begin{aligned} \nabla^2 f(\boldsymbol{w}) &= (I + CX^T DX)\boldsymbol{s} \\ &= \boldsymbol{s} + CX^T (D(X\boldsymbol{s})) \end{aligned} \tag{47}$$

考虑 X 稀疏，那么上式就可以很高效地计算出来而不需要存储整个黑塞矩阵。相比于直接法要存储黑塞矩阵，迭代法无疑是更好的选择。LIBLINEAR 的作者选择了共轭梯度法作为求解牛顿方向的间接法。不幸的是，共轭梯度法在某些情况下不容易收敛。为了节省时间，算法会在收敛之前便从共轭梯度法中获取解，作为近似的牛顿方向，这种方法也被称作截断牛顿法 (Truncated Newton Method)。

3.2.3 置信域牛顿法及其改进

这里给出求解 L2 正则的对率回归问题的置信域牛顿法 (算法 7)。算法设置 $\eta_1 < \eta_2 < 1$，

Algorithm 7 Trust Region Newton Algorithm for Logistic Regression

Start with any initial $\boldsymbol{x}^0$.
for $k = 0, 1, \cdots$ **do**
  if $\nabla f(\boldsymbol{w}^k) = 0$ **then**
    Stop iteration
  end if
  Solve the sub-problem (23) using Conjugate Descent Method to obtain $\boldsymbol{s}^k$.
  Calculate r_k using (24).
  if $r_k > \eta_1$ **then**
    $\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k + \boldsymbol{s}^k$
  else
    $\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k$
  end if
  if $r_k \leq \eta_1$ **then** ▷ Adjust trust region
    $\Delta_{k+1} \in [\sigma_1 \min\{\|\boldsymbol{s}^k\|, \Delta_k\}, \sigma_2 \Delta_k]$
  else if $r_k \in \{\eta_1, \eta_2\}$ **then**
    $\Delta_{k+1} \in [\sigma_1 \Delta_k, \sigma_3 \Delta_k]$
  else
    $\Delta_{k+1} \in [\Delta_k, \sigma_3 \Delta_k]$
  end if
end for

$\sigma_1 < \sigma_2 < 1 < \sigma_3$，可以发现这里置信域的更新规则和上面置信域算法的更新规则有所区别。

考虑算法中的共轭梯度法，由于需要 $\|\boldsymbol{s}^k\| \leq \Delta_k$，所以和普通的共轭梯度法有所不同，算法 8正是修改的共轭梯度算法。

3.2.4 改进的共轭梯度法

Preconditioned Conjugate Gradient 法对共轭梯度法进行了改进，正如名字中的" preconditioned"，算法会进行一个矩阵分解的预处理:

$$\nabla^2 f(\boldsymbol{w}) \approx PP^T \tag{48}$$

然后解一个新的线性系统

$$(P^{-1}\nabla^2 f(\boldsymbol{w}_k)P^{-T})\hat{\boldsymbol{s}} = -P^{-1}\nabla f(\boldsymbol{w}^k), \hat{\boldsymbol{s}} = P^T\boldsymbol{s} \tag{49}$$

Algorithm 8 Conjugate gradient procedure for solving trust region sub-problem

Given$\xi_k < 1, \Delta_k > 0$.
$\bar{\boldsymbol{s}}^0 \leftarrow \boldsymbol{0}, \boldsymbol{r}^0 \leftarrow -\nabla f(\boldsymbol{w}^k), \boldsymbol{d}^0 \leftarrow \boldsymbol{r}^0$.
for $i = 0, 1, \cdots$ **do**
  if $\|\boldsymbol{r}^i\| \leq \xi_k \|\nabla f(\boldsymbol{w}^k)\|$ **then**
    $\boldsymbol{s}^k \leftarrow \bar{\boldsymbol{s}}^i$ and stop iteration.
  end if
  $\alpha_i \leftarrow \dfrac{\|\boldsymbol{r}^i\|^2}{(\boldsymbol{d}^i)^T \nabla^2 f(\boldsymbol{w}^k) \boldsymbol{d}^i}$
  $\bar{\boldsymbol{s}}^{i+1} \leftarrow \bar{\boldsymbol{s}}^i + \alpha) i \boldsymbol{d}^i$
  if $\|\bar{\boldsymbol{s}}^{i+1} \geq \Delta_k\|$ **then**
    Calculate τ that $\|\bar{\boldsymbol{s}}^i + \tau \boldsymbol{d}^i\| = \Delta_k$ ▷ Adjust $\boldsymbol{s}$ into trust region
    $\boldsymbol{s}^k \leftarrow \bar{\boldsymbol{s}}^i + \tau \boldsymbol{d}^i$, output $\boldsymbol{s}^k$ and stop iteration.
  end if
  $\boldsymbol{r}^{i+1} \leftarrow \boldsymbol{r}^i - \alpha_i \nabla^2 f(\boldsymbol{w}^k) \boldsymbol{d}^i$
  $\beta_i \leftarrow \dfrac{\|\boldsymbol{r}^{i+1}\|^2}{\|\boldsymbol{r}^i\|^2}$
  $\boldsymbol{d}^{i+1} \leftarrow \boldsymbol{r}^{i+1} + \beta_i \boldsymbol{d}^i$
end for

如果上面的矩阵分解效果很好，那么 $P^{-1}\nabla^2 f(\boldsymbol{w}^k)P^{-T}$ 就会很接近一个单位矩阵，从而减少迭代数。作者采用了简单的处理方式

$$P = P^T = \sqrt{\text{Diag}(\nabla^2 f(\boldsymbol{w}_k))} \tag{50}$$

3.2.5 L2-SVM 下的置信域牛顿法

L2-SVM 对应无约束优化问题(29)，由于损失函数是可微且强凸，因此同样存在全局最小值，计算目标函数梯度:

$$\nabla f(\boldsymbol{w}) = (I + 2CX_{I,:}^T X_{I,:})\boldsymbol{w} - 2CX_{I,:}^T \boldsymbol{y}_I \tag{51}$$

其中 $I = \{i | 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i > 0\}$。也就是说，$I$ 是一个下标集合，$X_{I,:}$ 就是取下标集对应的行形成的新矩阵。这一点不难理解，因为对于函数 $\max(0, t)^2$，如果自变量小于 0，求出的导数始终为 0。max 运算的存在使得 f 不是二阶可微的，但我们发现 f 是几乎处处二阶可微的。我们可以定义出近似的黑塞矩阵:

$$B(\boldsymbol{w}) = I + 2CX^T DX \tag{52}$$

其中 D 是对角矩阵，对角元是对应的次梯度:

$$D_{ii} = \begin{cases} 1 & \text{if } 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i > 0, \\ \text{any element in } [0, 1] & \text{if } 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i = 0, \\ 0 & \text{if } 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i < 0. \end{cases} \tag{53}$$

由此，对于前面的置信域牛顿法，我们只需要将 $\nabla^2 f(\boldsymbol{w})$ 替换成 $B(\boldsymbol{w})$，就可以将该方法应用到 L2-SVM 的求解上。类似的，共轭梯度法求解的是下面的线性系统

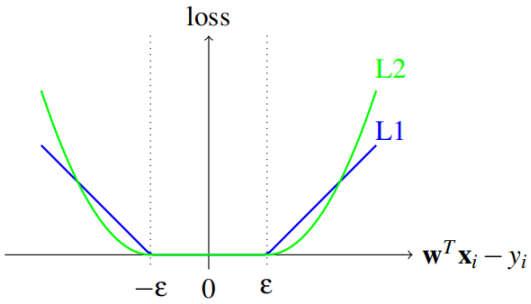

图 3: 两种损失函数

$$B(\boldsymbol{w})\boldsymbol{s} = s + 2CX_{I,:}^T(D_{I,I}(X_{I:,}\boldsymbol{s})) \tag{54}$$

3.3 求解带 L2 正则的 SVR 原问题和对偶问题

3.3.1 回归问题原型

给定数据集 $\{\boldsymbol{x}_i, y_i\}_{i=1}^l, \boldsymbol{x}_i \in R^n, y_i \in R$, 支持向量回归 (Support Vector Regression, SVR) 求解的是下面的优化问题:

$$\min \quad f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{j=1}^{l}\xi_\varepsilon(\boldsymbol{w}; \boldsymbol{x}_j, y_j) \tag{55}$$

其中 C 是正实数，而 ξ_ε 是损失函数，可是是 ε-不敏感损失函数 (L1-loss)，或者是它的平方 (L2-loss):

$$\xi_\varepsilon(\boldsymbol{w}; \boldsymbol{x}_j, y_j) = \begin{cases} \max(|\boldsymbol{w}^T\boldsymbol{x}_j - y_j| - \varepsilon, 0) & \text{or} \\ \max(|\boldsymbol{w}^T\boldsymbol{x}_j - y_j| - \varepsilon, 0)^2 \end{cases} \tag{56}$$

两种损失函数如图 3所示。

我们将采用两种损失函数回归模型为 L1-loss SVR 和 L2-loss SVR。SVR 的对偶问题为

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} \quad & f_A(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-) = \frac{1}{2}(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)^T Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) + \\ & \sum_{i=1}^{l}\left(\varepsilon(\alpha_i^+ + \alpha^-) - y_i(\alpha_i^+ - \alpha_i^-) + \frac{\lambda}{2}((\alpha_i^+)^2 + (\alpha_i^-)^2)\right) \\ \text{s.t.} \quad & 0 \le \alpha_i^+, \alpha_i^- \le U, i = 1, \cdots, l \end{aligned} \tag{57}$$

其中 $Q_{ij} = \boldsymbol{x}_i^T\boldsymbol{x}_j$。类似分类问题这里的 λ 和 U 也会随问题类型而不同，如表 3所示。

SVR 类型	λ	U
L1-loss SVR	0	C
L2-loss SVR	$1/2C$	∞

表 3: 不同 SVR 下 λ 与 U 的设置[14]

因此，上面的双变量问题可以重整为单变量问题:

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix}, f_A(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \begin{bmatrix} \bar{Q} & -Q \\ -Q & \bar{Q} \end{bmatrix} \boldsymbol{\alpha} + \begin{bmatrix} \varepsilon\boldsymbol{e} - \boldsymbol{y} \\ \varepsilon\boldsymbol{e} + \boldsymbol{y} \end{bmatrix}^T \boldsymbol{\alpha} \tag{58}$$

其中 $\bar{Q} = Q + \lambda I$，λ 的定义服从表 2；$\boldsymbol{e}$ 是全 1 向量。对偶问题的解和原问题的解满足

$$\boldsymbol{w}^* = \sum_{i=1}^{l} \left(\alpha_i^{+*} - \alpha_i^{-*}\right)\boldsymbol{x}_i \tag{59}$$

值得注意到是对偶问题的最优性条件

$$(\alpha_i^+)^*(\alpha_i^-)^* = 0, i = 1, \cdots, l \tag{60}$$

该对偶问题的规模是 $2l$，而分类问题 (SVC) 的规模是 l，我们当然可以继续使用求解 SVC 对偶的方法求解 SVR 对偶问题，但这样会造成计算代价的高昂。因此《Large-scale Linear Support Vector Regression》[13] 这篇文章便讨论如何改进原来的算法，使其适用于 SVR 问题。

3.3.2 置信域牛顿法求解 L2-loss SVR 原问题

考虑带置信域的牛顿法求解 L2 -loss SVR:

Algorithm 9 Trust Region Newton Method for L2-loss SVR

Start with any initial $\boldsymbol{x}^0$.
for $k = 0, 1, \cdots$ **do**
  if $\|\nabla f(\boldsymbol{w}^k)\| \leq \varepsilon_s \|\nabla f(\boldsymbol{w}^0)\|$ **then**
    return $\boldsymbol{w}^k$
  end if
  Solve trust region sub-problem
  Update $\boldsymbol{w}^k, \Delta_k$ to $\boldsymbol{w}^{k+1}, \Delta_{k+1}$.
end for

在每一轮迭代中，都需要计算当前的截断牛顿步径，那么就必然需要计算目标函数梯度和黑塞矩阵。L2-loss SVR 的梯度

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2C(X_{I_1,:})^T(X_{I_1,:}\boldsymbol{w} - \boldsymbol{y}_{I_1} - \varepsilon\boldsymbol{e}_{I_1}) - 2C(X_{I_2,:})^T(-X_{I_2}\boldsymbol{w} + \boldsymbol{y}_{I_2} - \varepsilon\boldsymbol{e}_{I_2}) \tag{61}$$

其中 $X = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l]^T, I_1 = \{i|\boldsymbol{w}^T\boldsymbol{x}_i - y_i > \varepsilon\}, I_1 = \{i|\boldsymbol{w}^T\boldsymbol{x}_i - y_i < -\varepsilon\}$，也就是只将对求导结果有影响的数据 (预测与实际值误差大于 ε) 取出来进行求导。L2-loss 目标函数有梯度，这是因为目标函数一阶可微；而函数不是二阶可微的，这代表黑塞矩阵 $\nabla^2 f(\boldsymbol{w})$ 不存在。因此有人 (Mangasarian) 提出该问题下的广义黑塞矩阵。令 $I = I1 \cup I2$，广义黑塞矩阵定义:

$$\nabla^2 f(\boldsymbol{w}) = I + 2C(X_{I,:})^T D_{I,I} X_{I,:} \tag{62}$$

I 是单位矩阵，而 $D_{l\times l}$ 是对角矩阵，且满足

$$D_{ii} = \begin{cases} 1 & \text{if } i \in I \\ 0 & \text{else} \end{cases} \tag{63}$$

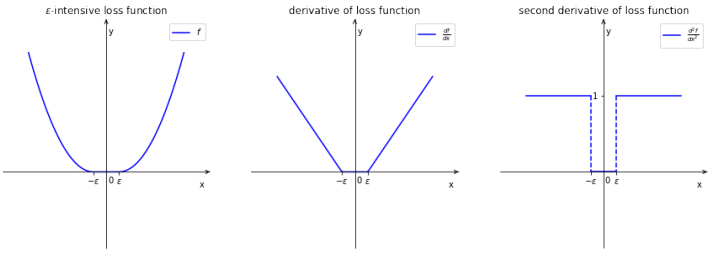

图 4: 对 L2-loss 逐步求导

这样的设置不难理解，对 L2-loss 逐步求导如图 4所示。

在求二阶导时，只有 ε 和 $-\varepsilon$ 这两点处不可微，而考虑到数据点落到这两点的概率极小，所以可以刨去这两点去求导。由于黑塞矩阵很大 $(l \times l)$，因此将这样的矩阵存储是不合理的。和之前一样，LIBLINEAR 采用 Hessian-free 的方法，只计算黑塞矩阵和向量的乘积:

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2C(X_{I,:})^T(D_{I,I}(X_{I,:}\boldsymbol{v})) \tag{64}$$

这样就能达到节省存储空间和提高算法运行速度的目的。

3.3.3 对偶坐标下降法求解

现在考虑用对偶坐标下降法 (DCD) 求解 SVR 问题。对每个分量的更新等价于求解子问题

$$\begin{aligned} \min_{z} \quad & f_A(\boldsymbol{\alpha} + z\boldsymbol{e}_i) - f_A(\alpha) = \nabla_i f_A(\boldsymbol{\alpha})z + \frac{1}{2}\nabla^2_{ii} f_A(\boldsymbol{\alpha})z^2 \\ \text{s.t.} \quad & 0 \le \alpha_i + z \le U \end{aligned} \tag{65}$$

其中 $\boldsymbol{e}_i$ 是长度为 $2l$，第 i 个元素为 1，其余元素为 0 的列向量，z 的更新仍具有闭式解:

$$\alpha_i \leftarrow \min(\max(\alpha_i - \frac{\nabla_i f_A(\boldsymbol{\alpha})}{\nabla^2_{ii} f_A(\boldsymbol{\alpha})}, 0), U) \tag{66}$$

其中

$$\nabla_i f_A(\boldsymbol{\alpha}) = \begin{cases} \left(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)\right)_i + \varepsilon - y_i + \lambda\alpha_i^+, & \text{if } 1 \le i \le l \\ -\left(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)\right)_{i-l} + \varepsilon + y_{i-l} + \lambda\alpha_{i-l}^-. & \text{if } l+1 \le i \le 2l \end{cases} \tag{67}$$

黑塞矩阵也很好求

$$\nabla^2_{ii} f_A(\boldsymbol{\alpha}) = \begin{cases} \bar{Q}_{ii} & \text{if } 1 \le i \le l \\ \bar{Q}_{i-l,i-l} & \text{if } l+1 \le i \le 2l \end{cases} \tag{68}$$

显然黑塞矩阵可以在迭代开始前就求好，后面直接取用即可；此外，和求解 SVC 一样，SVR 也存在更新前后 α_i 原地踏步的情况，因此提前判别并直接进行下一轮更新可以减少计算量；最后，我们可以设置并维护一个向量 $\boldsymbol{u}$:

$$u = \sum_{i=1}^{l}(\alpha_i^+ - \alpha_i^-)\boldsymbol{x}_i \tag{69}$$

这样就可以在求解结束后直接返回最优的 $\boldsymbol{w}$，还可以减少 $(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_i$ 的计算量，因为

$$(Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-))_i = \boldsymbol{u}^T \boldsymbol{x}_i \tag{70}$$

这里给出 SVR 下的对偶坐标下降算法 (算法 10).

Algorithm 10 A DCD method for L1/L2-loss SVR

Given $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$
$\boldsymbol{\alpha} \leftarrow \begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix}$
$\boldsymbol{u} \leftarrow \sum_{i=1}^{l} (\alpha_i - \alpha_{i+l}) \boldsymbol{x}_i$
Calculate $\bar{Q}_{ii}, \forall i = 1, \cdots, l.$
for $k = 0, 1, 2 \cdots$ **do**
  for $i \in \{1, \cdots, 2l\}$ **do** ▷ select an index to update
    if α_i can be updated **then**
      Update α_i by (66).
      Unpdate $\boldsymbol{u}$.
    end if
  end for
end for

3.3.4 改进的对偶坐标下降法

算法 10将 $\boldsymbol{\alpha}$ 中 $2l$ 个变量看作是独立的，但实际上，考虑最优性条件(60)，也就是 α_i^+ 和 α_i^- 至少有一个为 0，如果两者中一个为正，那么另一个十分有可能在最后的循环中一直为 0，所以会有很多检查时间被浪费 (被浪费的没有更新时间，因为这些变量需要先判别后更新，而算法会将它们判别不需要更新，因此只有判别时间被浪费)。因此，DCD 算法在 SVR 问题上可以进一步优化。

考虑最优性条件，可以将对偶问题中的 $(\alpha_i^+)^2 + (\alpha_i^-)^2$ 修改为 $(\alpha_i^+ - \alpha_i^-)^2$:

$$\min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} \quad \frac{1}{2}(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)^T Q(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) + \sum_{i=1}^{l} \left(\varepsilon(\alpha_i^+ + \alpha_i^-) - y_i(\alpha_i^+ - \alpha_i^-) + \frac{\lambda}{2}(\alpha_i^+ - \alpha_i^-)^2 \right) \tag{71}$$

由于我们在 DCD 算法中时刻保持 $\alpha_i^+ \geq 0, \alpha_i^- \geq 0$，因此在最优点处，我们有 $\alpha_i^+ + \alpha_i^- = |\alpha_i^+ - \alpha_i^-|$。定义 $\boldsymbol{\beta} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$，对偶问题可以被改写成

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & f_B(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^T \bar{Q} \boldsymbol{\beta} - \boldsymbol{y}^T \boldsymbol{\beta} + \varepsilon \|\boldsymbol{\beta}\|_1 \\ \text{s.t.} \quad & -U \leq \beta_i \leq U, \forall i. \end{aligned} \tag{72}$$

如果 $\boldsymbol{\beta}^*$ 是上述问题的最优解，那么

$$(\alpha_i^+)^* = \max(\beta_i^*, 0), (\alpha_i^-)^* = \max(-\beta_i^*, 0) \tag{73}$$

现在考虑该优化问题的求解，还是使用坐标下降法，对第 i 个变量进行更新等价于求解子问题

$$\min_s \quad g(s) \text{s.t.} \quad -U \le s \le U \tag{74}$$

其中

$$\begin{aligned} g(s) &= f_B(\boldsymbol{\beta} + (s-\beta_i)\boldsymbol{e}_i) - f_B(\boldsymbol{\beta}) \\ &= \varepsilon|s| + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i(s-\beta_i) + \frac{1}{2}\bar{Q}_{ii}(s-\beta_i)^2 + \text{constant} \end{aligned} \tag{75}$$

分类讨论 $g(s)$ 的导数:

$$\begin{cases} g'_p(s) = \varepsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i + \bar{Q}_{ii}(s-\beta_i), & \text{if } s \ge 0 \\ g'_n(s) = -\varepsilon + (\bar{Q}\boldsymbol{\beta} - \boldsymbol{y})_i + \bar{Q}_{ii}(s-\beta_i), & \text{if } s \le 0 \end{cases} \tag{76}$$

显然两个导函数都是 s 的线性函数，且 $g'_n(s) \le g'_p(s), \forall s \in \mathbb{R}$。当 $g'(s) = 0$ 时，$g(s)$ 取最小值。一共三种情况，如图 5所示。

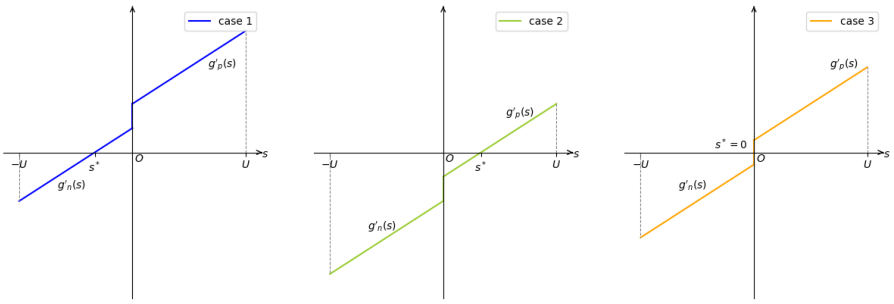

图 5: $g'(s) = 0$ 的三种情况

由此可以得到三种情况下 s^* 的闭式解:

$$s \leftarrow \max(-U, \max(U, \beta_i + d)) \tag{77}$$

其中

$$d = \begin{cases} -\dfrac{g'_p(\beta_i)}{\bar{Q}_{ii}} & \text{if } g'_p(\beta_i) < \bar{Q}_{ii}\beta_i \\ -\dfrac{g'_n(\beta_i)}{\bar{Q}_{ii}} & \text{if } g'_n(\beta_i) > \bar{Q}_{ii}\beta_i \\ -\beta_i & \text{otherwise} \end{cases} \tag{78}$$

到这里，就可以总结出 SVR 问题下的 DCD 改进算法 (算法 11)。

类似的，我们还是维护 $\boldsymbol{u}$ 向量作为解。论文的后面是关于算法收敛性的讨论，算法实现上的细节以及实验。我们这里只提一下算法实现中的 Shrink 过程: 它会衡量各分量违反最优性的程度，对于违反程度不大的分量，算法会将其忽视，只对剩下的分量进行更新，由此达到提高算法效率的目的。具体过程可以参考原论文的算法 4。

Algorithm 11 A new DCD method for L1/L2-loss SVR

Given $\boldsymbol{\beta}$
$\boldsymbol{u} \leftarrow \sum_{i=1}^{l} \beta_i \boldsymbol{x}_i$.
Calculate $\bar{Q}_{ii}, \forall i = 1, \cdots, l$.
for $k = 0, 1, \cdots$ **do**
  for $i \in \{1, \cdots, l\}$ **do** ▷ select an index to update
    find s by (77).
    $\boldsymbol{u} \leftarrow \boldsymbol{u} + (s - \beta_i)\boldsymbol{x}_i$
    $\beta_i \leftarrow s$
  end for
end for

3.4 求解带 L2 正则的 L2-loss SVM 原问题

3.4.1 朴素算法流程

LIBLINEAR 采用坐标下降法求解带 L2 正则的 l2-loss SVM，也就是(29)式。我们采用《Coordinate descent method for large-scale l2-loss linear support vector machines》[15] 一文中的 notation 讲解此算法。我们设 x_{ji} 是数据集的第 j 个数据的第 i 个特征，而数据集的样本数为 l，特征数为 n。坐标下降法必然是从初始点 $\boldsymbol{w}^0$ 开始，形成一个序列 $\{\boldsymbol{w}^k\}_{k=0}^{\infty}$。从 $\boldsymbol{w}^k$ 更新到 $\boldsymbol{w}^{k+1}$，需要经历 n 次单坐标迭代。因此我们设 $\boldsymbol{w}^{k,i} \in \mathbb{R}^n, i = 1, \cdots, n$:

$$\boldsymbol{w}^{k,i} = [w_1^{k+1}, \cdots, w_{i-1}^{k+1}, w_i^k, \cdots, w_n^k]^T \text{ for } i = 1, \cdots, n \tag{79}$$

也就是第 k 轮迭代中，对前 i 个分量进行更新后的结果。将 $\boldsymbol{w}^{k,i}$ 更新到 $\boldsymbol{w}^{k,i+1}$ 的过程中，我们需要解决单变量子问题:

$$\min_z f(w_1^{k+1}, \cdots, w_{i-1}^{k+1}, w_i^k + z, w_{i+1}^k, \cdots .w_n^k) \equiv \min_z f(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i) \tag{80}$$

其中 $\boldsymbol{e}_i$ 是长度为 n，第 i 个元素为 1，其他元素都为 0 的向量。对子问题进行改写:

$$\begin{aligned} D_i(z) &= f(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i) \\ &= \frac{1}{2}(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)^T(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i) + C \sum_{j \in I(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)} (b_j(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i))^2 \end{aligned} \tag{81}$$

其中 $b_j(\boldsymbol{w}) = 1 - y_j \boldsymbol{w}^T \boldsymbol{x}_j$，$I(\boldsymbol{w}) = \{j | b_j(\boldsymbol{w}) > 0\}$。考虑损失函数的性质，只有 $b_j > 0$ 对应的数据 $\boldsymbol{x}_j$ 才会对目标函数值产生影响。对于 $I(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)$ 不变的区间中，$D_i(z)$ 是一个二次函数。因此 $D_i(z), z \in \mathbb{R}$ 是一个分段 (piecewise) 二次函数。由此文章选择使用牛顿法来优化该问题。如果 $D_i(z)$ 是二次可微的，在给定 $\bar{z}$ 处的牛顿步径应该是

$$-\frac{D_i'(\bar{z})}{D_i''(\bar{z})}$$

函数的一阶微分是

$$D_i'(z) = w_i^{k,i} + z - 2C \sum_{j \in I(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)} y_j x_{ji} (b_j(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)) \tag{82}$$

但函数无法进行二阶微分，这是因为 max 函数不可微。面对这种情况，Mangasarian 在 2002 年定义了广义的二阶微分:

$$\begin{aligned} D_i''(z) &= 1 + 2C \sum_{j \in I(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)} y_j^2 x_{ji}^2 \\ &= 1 + 2C \sum_{j \in I(\boldsymbol{w}^{k,i} + z\boldsymbol{e}_i)} x_{ji}^2 \end{aligned} \tag{83}$$

所以设 $z^0 = 0$，使用朴素的牛顿法，也就是按照下式迭代 z 直到 D_i'$(z) = 0$:

$$z^{t+1} = z^t - \frac{D_i'(z^t)}{D_i''(z^t)} \text{ for } t = 0, 1, \cdots \tag{84}$$

整理成伪代码如算法 12所示

Algorithm 12 Coordinate descent method for L2-SVM primial problem

Start with any initial $\boldsymbol{w}^0$.
for $k = 0, 1, \cdots$ **do**
  for $i = 1, \cdots, n$ **do**
    Fix $w_1^{k+1}, \cdots, w_{i-1}^{k+1}, w_{i+1}^k, \cdots, w_n^k$ and approximately solve sub-problem (80)
  end for
end for

3.4.2 算法的问题与改进

Mangasarian 证明了在某种假设下，上面的牛顿法可以在有限步结束并且正确求解子问题。但这种假设在真实情况下不会恒成立，这就导致取完整的牛顿步径进行下降不一定会让目标函数下降；此外，这种求解子问题的方法的求解代价很昂贵。

早期研究坐标下降求解 L2-SVM 的文章不打算精确求解子问题。在 2001 年的文章中，Zhang 和 Oles 提出了 CMLS 算法，它求出来的近似解被限制在一个区域中，通过评估函数在这个区域内的二阶导数上界，然后用这个上界去替代二阶导数，也就是牛顿步径的分母。这种设置保证了函数下降。但仍存在两个问题:

- 函数下降并不等价于下降是收敛的;
- 用二阶导数上界作为分母会使得下降十分保守.

在这篇文章中，作者选取了保证坐标下降法收敛的充分条件:

$$D_i(z) - D_i(0) \leq -\sigma z^2 \tag{85}$$

其中 z 是当前所在的位置，$\sigma \in (0, \frac{1}{2})$ 是任意常数。考虑第一次迭代

$$d = -\frac{D_i'(0)}{D_i''(0)}$$

第一次迭代是否恒满足收敛条件是很重要的，也就是判断

$$D_i(d) - D_i(0) \leq -\sigma d^2 \tag{86}$$

因为 d 很靠近 0，$D_i(z)$ 也是一个二次函数，因此有

$$D_i(d) - D_i(0) = D_i'(0)d + \frac{1}{2}D_i''(0)d^2 \tag{87}$$

因为 $D_i''(z) > 1$，所以

$$\begin{aligned} D_i'(0)d + \frac{1}{2}D_i''(0)d^2 + \sigma d^2 &= -\frac{D_i'(0)^2}{D_i''(0)} + (\frac{1}{2}D_i''(0) + \sigma)\frac{D_i'(0)^2}{D_i''(0)^2} \\ &= -\frac{1}{2}\frac{D_i'(0)^2}{D_i''(0)} + \sigma\frac{D_i'(0)^2}{D_i''(0)} \\ &= (\sigma - \frac{1}{2})\frac{D_i'(0)^2}{D_i''(0)} \leq 0 \\ D_i(d) - D_i(0) &\leq -\sigma d^2 \end{aligned} \tag{88}$$

也就是满足收敛条件。考虑到 $D_i(z)$ 是分段二次函数，所以对于 $z = d$，收敛条件不一定成立，但作者使用了线搜索，同时提出定理 1。

Theorem 1. 给定上述牛顿步径 d，$\forall \lambda \in [0, \bar{\lambda}]$，$z = \lambda d$ 始终满足上述收敛条件。其中

$$\bar{\lambda} = \frac{D_i''(0)}{H_i/2 + \sigma}, H_i = 1 + 2C\sum_{j=1}^{l} x_{ji}^2$$

这样，作者希望通过线搜索找到满足上述条件的 λ 中的最大值，将 λd 作为牛顿步径，也就是下面的线搜索算法 13。

Algorithm 13 Solve the sub-problem using Newton method with line search

Given $\boldsymbol{w}^{k,i}$ and $\beta \in (0, 1)$, usually 0.5.
$d \leftarrow -\frac{D_i'(0)}{D_i''(0)}$.
Compute $\lambda = \max\{1, \beta, \beta^2, \cdots\}$ such that $z = \lambda d$ satisfies (85).

在算法 13 中，计算最耗时的一步是计算 $D_i(\lambda d)$，现在考虑如何减少计算 $D_i(\lambda d)$ 的次数。理论上，我们需要依次计算 $D_i(d), D_i(\beta d), D_i(\beta^2 d) \cdots$，逐一判断是否满足收敛条件。实际上，考虑到我们有定理一，所以在计算更小的 $D_i(\lambda d)$ 之前，先判断 λ 是否满足定理一中的收敛条件，如果满足则直接返回，不需要进行 $D_i(\lambda d)$ 的计算。由于 $D_i''(0)$ 和 H_i 都只需要计算一次，因此相比于计算 $D_i(\lambda d)$，与 $\bar{\lambda}$ 进行比较更节省时间。此外，如果 $\lambda = 1$ 满足定理 1，我们也没必要进行后面的计算了，可以直接退出。由此，算法 2 的计算时间可以大大缩短。

算法中有两个超参数，β 和 σ。因为 $\lambda = 1$ 时常常收敛，因此算法对 β 是不敏感的，设它为 0.5 即可；σ 通常选择 0.01。

3.4.3 收敛性、复杂度与实现技巧

算法 12所生成的 $\{\boldsymbol{w}^k\}$ 序列是线性收敛的，也就是 $\exists \mu \in (0, 1)$，使得

$$f(\boldsymbol{w}^{k+1}) - f(\boldsymbol{w}^*) \leq (1 - \mu)(f(\boldsymbol{w}^k) - f(\boldsymbol{w}^*)), \forall k$$

作者证明了算法在 $O(nC^3P^6(\#nz)^3 \log(1/\epsilon))$ 次循环内能够到达 ϵ 精确解，也就是

$$f(\hat{\boldsymbol{w}}) \leq f(\boldsymbol{w}^*) + \epsilon$$

其中 $\#nz$ 是训练数据中非零值的数目，显然在稀疏数据中，这个值会很小，可以看出该算法适合稀疏数据集分类；P 是数据集元素的绝对值上界:

$$P = \max_{ji} |x_{ji}|$$

由于分类数据预处理中会进行归一化，所以 $P \leq 1$ 常常成立。具体的证明过程参考原论文附录。此外，作者证明了单次迭代的复杂度是 $O(\#nz)$，也就是数据越稀疏，算法计算越快。

论文在这一节列举了 3 种实现上述算法的技巧，可以提高程序速度。第一种方法是从数据表示入手，由于该算法适合稀疏数据，因此数据存储也应该是稀疏矩阵。常用的存储稀疏矩阵的方法有“行存储”和“列存储“，这样存储能够快速读取稀疏数据集元素。由于该算法是以特征为单位求解子问题，因此按列存储是好主意。

求解子问题是按照特征顺序进行，特征的顺序有时会影响收敛速度，因此作者提出每一轮可以按照不同的特征顺序求解子问题，也就是对原数据集特征顺序进行一个排列。作者也证明了这样打乱顺序后，收敛效果和原算法相同。

如果数据的特征数极大，我们常常不需要在每个循环中遍历全部特征。相反，我们可以随机选择一个特征进行下降，也就是从 $\boldsymbol{w}^k$ 到 $\boldsymbol{w}^{k+1}$ 的过程中，只有一个分量被更新，得到所谓的在线算法 (Online algorithm)。

3.5 求解带 L2 正则的 LR 对偶问题

3.5.1 问题原型与变形

LR 原问题是无约束优化问题，如(40)所示，其对应的对偶问题为

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i:\alpha_i>0} \alpha_i \log \alpha_i + \sum_{i:\alpha_i<C} (C-\alpha_i)\log(C-\alpha_i) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \cdots, l \end{aligned} \tag{89}$$

其中 $Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$。考虑到 $0 \log 0 = 0$，则可简化为

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i=1}^{l} \alpha_i \log \alpha_i + (C-\alpha_i)\log(C-\alpha_i) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \cdots, l \end{aligned} \tag{90}$$

《Dual coordinate descent methods for logistic regression and maximum entropy models》[17] 一文考虑用坐标下降法求解对率回归的对偶问题，尽管当时大部分算法是针对原问题提出的。这里设计的坐标下降法不仅能够解决对率回归数值计算上的问题 (因为有对数运算)，同时在性能上超过了当时大部分训练对率回归分类器的算法。

3.5.2 针对 LR 问题的对偶坐标下降

尽管 LR 对偶问题和 SVM 对偶问题相似，但存在很大区别。其中一点就是，SVM 的对偶坐标下降通常会将初始解 $\boldsymbol{\alpha}$ 设置为 0，这是为了保证解的稀疏性。但对于 LR 问题，它在 $\alpha_i = 0$ 或 C 的时候不是良定义 (not well defined) 的，这就导致初始解 $\boldsymbol{\alpha}$ 必须在开区间 $(0, C)^l$。甚至说，我们不能确定最优解是否出现 $\alpha_i = 0$ 或 C。作者证明了下面的定理:

Theorem 2. *LR 对偶问题(89)有一个全局最优解 $\boldsymbol{\alpha}^*$ 且 $\boldsymbol{\alpha} \in (0, C)^l$。*

此外，SVM 对偶问题的坐标下降子问题是有解析解的，但 LR 对偶问题的子问题是求解

$$\begin{aligned} \min_{z} \quad & g(z) = (c_1 + z)\log(c_1 + z) + (c_2 - z)\log(c_2 - z) + \frac{a}{2}z^2 + bz \\ \text{s.t.} \quad & -c_1 \leq z \leq c_2 \end{aligned} \tag{91}$$

其中 $c_1 = \alpha_i, c_2 = C - \alpha_i, a = Q_{ii}, b = (Q\boldsymbol{\alpha})_i$。显然该问题是没有闭式解的，因此采取牛顿法求解:

$$z^{k+1} \leftarrow z^k + d, d = -\frac{g'(z^k)}{g''(z^k)} \tag{92}$$

对于 $(-c_1, c_2)$ 内的 z，可以求得 $g(z)$ 的一阶和二阶导数:

$$\begin{aligned} g'(z) &= az + b + \log\frac{c_1 + z}{c_2 - z} \\ g''(z) &= a + \frac{c_1 + c_2}{(c_1 + z)(c_2 - z)} \end{aligned} \tag{93}$$

为了确保收敛性，我们通常采用线搜索去找一个确保收敛的牛顿步径，即找到第一个满足下面条件的 $\lambda = 1, \beta, \beta^2, \cdots$:

$$g(z^k + \lambda d) - g(z) \leq \gamma\lambda g'(z^k)d \tag{94}$$

其中 $\gamma, \beta \in (0, 1)$。这一做法和上一节求解带 L2 正则的 L2-lossSVM 原问题类似。在迭代中维护解向量已经是常规操作了:

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i\alpha_i\boldsymbol{x}_i \tag{95}$$

通过下式更新解向量

$$\boldsymbol{w}(\boldsymbol{\alpha} + z\boldsymbol{e}_i) = \boldsymbol{w}(\boldsymbol{\alpha}) + zy_i\boldsymbol{x}_i \tag{96}$$

这样的设置允许我们很快计算出 $(Q\boldsymbol{\alpha})_i$:

$$(Q\boldsymbol{\alpha})_i = y_i\boldsymbol{w}^T\boldsymbol{x}_i \tag{97}$$

在每一轮坐标下降更新中，我们需要做的事情:

- 构建子问题，尤其是 $(Q\boldsymbol{\alpha})_i$，从上面可以知道时间是 $O(n)$;
- 寻找牛顿方向 d，涉及到对数运算，时间是 $O(1)$;
- 在线搜索过程中计算 $g(z^k + \lambda d)$，时间是 $O(1)$.

在通常情况下，线搜索一次就可以找到适合的牛顿步径。虽然第二步的公式很简单，但其中的对数运算耗时远超普通加减法；此外，线搜索过程中频繁使用 log 运算也使得线搜索代价远超第 2 步，尽管两者的时间复杂度相同。因此，论文在下面提出改进的坐标下降法。

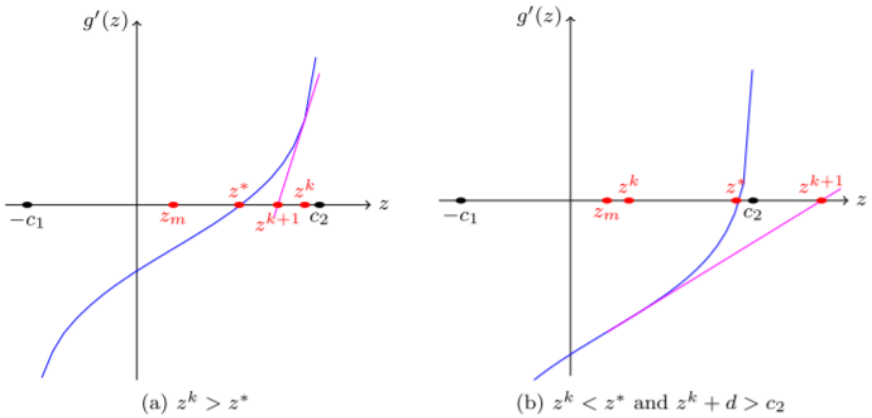

图 6: 牛顿法求 $g'(z)$ 零点的两种情况

3.5.3 改进的子问题求解法

我们期望在新算法不需要线搜索的场景下也可以保持收敛性。作者这里证明了下面的定理:

Theorem 3. *LR 对偶问题的子问题有一个全局最小值点 z^*，且 $z^**\in(-c_1,c_2), g'(z^*)=0$。*

图 6展示了牛顿法求根 (求导函数的根，也就是极值点) 的两种情况。其中 z^k 是初始点，z^{k+1} 是由牛顿步径导出的位置，z^* 是最优值点，而 $z_m=(c_2-c_1)/2$，也就是可行区间的中点。图 (a) 展示了一个好的初始点能让牛顿法奏效；图 (b) 则是一种糟糕的情形: 牛顿迭代让 z^{k+1} 跳至区间外。因此我们需要一种机制，让所有的点都能到达正确的一边，也就是图 (a) 所示。从上图可以发现 (理论上也是)，$g'(z)$ 在 $(-c_1,c_m]$ 上为凹函数，在 $[c_m,c_2)$ 上是凸的。于是作者提出下面的定理判断 z^k 是否在正确的一边:

Theorem 4. *如果 $z^*\geq z_m$，那么由牛顿法生成的 z^k 会收敛到 z^* 只要初始点在 $[z^*,c_2)$ 上；如果 $z^*\leq z_m$，那么 $\{z_k\}$ 会收敛到 z^* 只要初始点在 $(-c_1,z^*)$。*

对于上述两条件之一的 z^k，我们称它在 z^* 正确的一边。如果上述条件皆不满足，则又分以下几种情况:

- z^k+d 没有越界，且处于正确的一边: 满足条件，令 $z^{k+1}=z^k+d$ 继续迭代即可；
- z^k+d 没有越界，但处于错误的一边: 考虑它离最优解更近，令 $z^{k+1}=z^k+d$，继续迭代；
- z^k+d 越界，由于合法区间是开区间，无法做投影。不失一般性，设 $z^{k+1}+d\geq c_2$。我们希望找到点 $z\in[z^k,c_2)$ 更接近于正确的一边:

  $$z^{k+1}=\xi z^k+(1-\xi)c_2, \xi\in(0,1)$$

  因为 z^k 处于错误的一边，作者证明了上述设置最终可以让点到达正确一边，然后这个点就作为满足定理 4 的起始点进行迭代。

Theorem 5. *假定 $z^*\geq z_m$，如果我们以 $z^k<z^*$ 为起始点生成牛顿迭代序列 (也就是从错误一边开始)，同时用下面的更新规则:*

$$z^{k+1}=\begin{cases} z^k+d & if\ z^k+d<c_2 \\ \xi z^k+(1-\xi)c_2 & if\ z^k+1\geq c_2 \end{cases}$$

从而会有一个 $k' > k$ 使得 $z^{k'} \geq z^*$, 也就是说, $z^{k'}$ 是在正确的一边。对于 $z^* \leq z_m$ 且 $z^k \leq z^*$ 的情况是相似的。

基于上述定理，下面的算法 14 可用于求解 LR 对偶问题子问题。

Algorithm 14 A modified Newton method to solve sub-problem(91)

Input: a, b, c_1 and c_2.

Set initial $z^0 \in (-c_1, c_2)$.

for $k = 0, 1, \cdots$ **do**

if $g'(z^k) = 0$ **then**

break.

end if

$$d \leftarrow -\frac{g'(z^k)}{g''(z^k)}$$

$$z^{k+1} \leftarrow \begin{cases} z^k + d & \text{if } z^k + d \in (-c_1, c_2) \\ \xi z^k + (1-\xi)(-c_1) & \text{if } z^k + d \leq -c_1 \\ \xi z^k + (1-\xi)c_2 & \text{if } z^k + d \geq c_2 \end{cases}$$

end for

可以发现该算法摆脱了线搜索，基于定理 4 和定理 5，该算法的收敛性显然成立，有下面的定理:

Theorem 6. 由上述算法生成的 $\{z_k\}$ 可收敛到 z^*，这对任意 $z^0 \in (-c_1, c_2)$ 都成立。

虽然定理 6 声明了初始点的随意性，但我们仍希望有更快的收敛速度，因此初始点的选择还是重要的。在整个坐标下降算法的后期，显然 α_i 变化越来越小，相应的，$z^* \approx 0$，将初始点设成 0 是有道理的。但显然在算法前期这样的设置是不合理的。此外，由于最优点未知，我们无法将初始点指定在正确的一边。定理 4 希望我们这样设置初始点:

$$z^0 \in \begin{cases} (-c_1, z_m), & \text{if } z^* \leq z_m, \\ [z_m, c_2), & \text{if } z^* \geq z_m. \end{cases} \tag{98}$$

因此作者选用了总体为 0，但稍有调整的初始化方案:

$$z^0 = \begin{cases} (1-\xi_0)(-c_1) & \text{if } z^* \leq z_m \leq 0 \\ (1-\xi_o)c(c_2) & \text{if } z^* \geq z_m \geq 0 \\ 0 & \text{else} \end{cases} \tag{99}$$

我们设置 $0 < \xi_0 \leq 0.5$，这样可以让 z^0 满足定理 4 所希望的初始化方式: 若

$$-c_1 < z^* \leq z_m \leq 0 < c_2$$

那么 $(1-\xi_0)(-c_1) \in (-c_1, 0)$ 而且离 $-c_1$ 更近，也就是满足 $(1-\xi_0)(-c_1) \leq z_m$。$z^* \geq z_m \geq 0$ 的情况与之类似。

3.5.4 数值计算上的困难

不幸的是，作者发现上面提出的改进坐标下降法在数值计算上存在阻碍。具体来说，就是当 α_i 接近 0 或 C 时，算法很难去到达最优点 z^*，也就是导数为 0:

$$g'(z^*) = Q_{ii}z^* + (Q\boldsymbol{\alpha})_i + \log(\alpha_i + z^*) - \log(C - \alpha_i - z^*) \approx 0 \tag{100}$$

2005 年的一篇论文指出，当 C 很大 (比如 10^5) 时，$(Q\boldsymbol{\alpha})_i$ 也很大，那么 $\alpha_i + z^*$ 就可能会很小 (比如 10^{-5})，在浮点数表示上有困难。但作者通过实验发现，即使 C 有这么大，$(Q\boldsymbol{\alpha})_i$ 也没有很大，通常也就几百几千的样子。作者发现数值计算问题出现在巨量消失 (catastrophic cancellations，也就是两个相近数值的减法)，即 $\alpha_i + z$ 接近于 0 的时候。此时计算 $\log(\alpha_i + z)$ 会将错误进一步扩大。

一种解决方案是将问题重构，令 $Z_1 = c_1 + z$，$s = c_1 + c_2$，因此得到子问题的等价形式

$$\begin{aligned} \min_{Z_1} \quad & g_1(Z_1) = Z_1 \log Z_1 + (s - Z_1)\log(s - Z_1) + \frac{a}{2}(Z_1 - c_1)^2 + b_1(Z_1 - c_1) \\ \text{s.t.} \quad & 0 \le Z_1 \le s, b_1 = b \end{aligned} \tag{101}$$

也就是将变量在坐标轴上右移 c_1 个单位。当 $z \approx -c_1$，也就是 $\alpha_i + z \approx 0$，

$$s - Z_1 = c_2 - z \approx c_1 + c_2 = s$$

距离 0 很远，也就避免了 catastrophic cancellation。尽管还有一个减运算 $Z_1 - c_1$，但它不参加对数运算，因此相对误差可以接受，不会造成严重的数值计算问题。

类似的，当 $z \approx c_2$，我们要将令 $Z_2 = c_2 - z$，得到子问题的第二种等价形式:

$$\begin{aligned} \min_{Z_2} \quad & g_2(Z_2) = Z_2 \log Z_2 + (s - Z_2)\log(s - Z_2) + \frac{a}{2}(Z_2 - c_2)^2 + b_2(Z_2 - c_2) \\ \text{s.t.} \quad & 0 \le Z_2 \le s, b_2 = -b \end{aligned} \tag{102}$$

为了求解上面两个等价子问题，我们还是用牛顿法，先求一阶和二阶导数

$$\begin{aligned} g_t'(Z_t) &= \log\frac{Z_t}{s - Z_t} + a(Z_t - c_t) + b_t \\ g_t''(Z_t) &= a + \frac{s}{Z_t(s - Z_t)} \end{aligned} \tag{103}$$

现在考虑应该用 $g_1(Z_1)$ 还是 $g_2(Z_2)$，这里选择的依据是 z^* 离哪个界更近:

$$z^* \text{ closer to } \begin{cases} -c_1 \\ c_2 \end{cases} \quad \to \quad \text{choose } \begin{cases} g_1(Z_1), \\ g_2(Z_2). \end{cases} \tag{104}$$

还是那个问题，我们不知道 z^* 在哪儿，于是作者证明了下面的规律:

$$z^* \text{ closer to } \begin{cases} -c_1 \\ c_2 \end{cases} \quad \leftrightarrow \quad z^* \begin{cases} \le z_m \\ \ge z_m \end{cases} \quad \leftrightarrow \quad g'(z_m) \begin{cases} \ge 0 \\ \le 0 \end{cases} \quad \leftrightarrow \quad z_m \begin{cases} \ge -\frac{b}{a} \\ \le -\frac{b}{a} \end{cases} \tag{105}$$

根据这一规律，初始化方案变成了

$$Z_1^0 = \begin{cases} \xi_0 c_1 & \text{if } c_1 \ge \frac{s}{2}, \\ c_1 & \text{otherwise} \end{cases} \quad Z_2^0 = \begin{cases} \xi_0 c_2 & \text{if } c_2 \ge \frac{s}{2}, \\ c_2 & \text{otherwise} \end{cases} \tag{106}$$

而 $z^k + d$ 越界的处理方案也就变成了

$$c_2 - Z_2^{k+1} = \xi(c_2 - Z_2^{k+1}) + (1-\xi)c_2 \tag{107}$$

即

$$Z_2^{k+1} = \xi Z_2^k \tag{108}$$

由此得到改进的算法，即算法 15。

Algorithm 15 A new modified Newton Method for (91)

Input: a, b, c_1 and c_2.

$s \leftarrow c_1 + c_2$.

$$t \leftarrow \begin{cases} 1 & \text{if } z_m \geq -\frac{b}{a}, \\ 2 & \text{if } z_m < -\frac{b}{a}. \end{cases}$$

$Z_t^0 \in (0, s)$.

for $k = 0, 1, \cdots$ **do**

  if $g_t'(Z_t^k) = 0$ **then**

    break.

  end if

  $d \leftarrow -\dfrac{g_t'(Z_t^k)}{g_t''(Z_t^k)}$.

$$Z_t^{k+1} = \begin{cases} \xi Z_t^k & \text{if } Z_t^k + d \leq 0 \\ Z_t^k + d & \text{otherwise} \end{cases}$$

end for

$$\begin{cases} Z_2^k = s - Z_1^k & \text{if } t = 1. \\ Z_1^k = s - Z_2^k & \text{if } t = 2. \end{cases}$$

return (Z_1^k, Z_2^k).

3.5.5 完整的算法流程

前面提到，不同于 SVM，LR 对偶问题的初始点无法设置为 $\mathbf{0}$，一种朴素的初始化方法是将所有元素设为 $\frac{C}{2}$，但实验表明虽然 LR 对偶问题没有稀疏性，但很多 α_i 都是很小的。考虑最优性条件，即最优的 $(\boldsymbol{w}, \boldsymbol{\alpha})$ 满足

$$\alpha_i = \frac{C \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)}{1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)}, \forall i \tag{109}$$

显然随着算法进行，指数项 $\exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)$ 趋近于 0，那么 α_i 也会接近 0，因此我们应当将初始向量设为很小的值，考虑

$$\alpha_i = \min(\epsilon_1 C, \epsilon_2), \forall i. \tag{110}$$

其中 ϵ_1 和 ϵ_2 都是小于 1 的小正数。作者在这里引用了 2005 年的论文，其中的观点是当 $y_i = -1$ 时，将 α_i 初始化为 C/l^-，也是就 C 除以负样本个数；当 $y_i = 1$ 时则令 $\alpha_i = C/l^+$。上式中的 $\epsilon_1 C$ 就对应这个观点，而 ϵ_2 的设置是因为作者觉得初始值不应完全受 C 影响（C 如果太大，初始值就不小了）。

当构建子问题时，如果 $\alpha_i \approx C$，那么前面提到的数值优化问题又会出现在计算 $c_2 = C - \alpha_i$ 的时候。为了解决这一问题，可以考虑 Z_2 的定义

$$Z_2 = c_2 - z = C - \alpha_i^{\text{old}} - z = C - \alpha_i^{\text{new}} \tag{111}$$

因此，如果上一轮对第 i 个坐标分量更新时采用的是最小化 $g_2(Z_2)$，那么这一轮构造子问题时，c_2 可以直接用 Z_2 的值；而如果上一轮是最小化 $g_1(Z_1)$，我们用 $Z_2 = s - Z_1$ 去求 c_2，同样可以避免数值计算问题。这对应了上面的算法 15 为何同时返回 Z_1^k 和 Z_2^k。算法 16 是整个求解 LR 对偶算法的过程。

Algorithm 16 DCD method for LR dual problem

$\alpha_i \leftarrow \min(\epsilon_1 C, \epsilon_2), \forall i.$
$\boldsymbol{w} \leftarrow \sum_i \alpha_i y_i \boldsymbol{x}_i.$
while $\boldsymbol{\alpha}$ is not optimal **do**
  for $i = 1, \cdots, l$ **do**
    Construct sub-problem (91) by

$$c_1 = \alpha_i, c_2 = \alpha_i', a = Q_{ii}, b = y_i \boldsymbol{w}^T \boldsymbol{x}_i$$

    Solve (91) by Algorithm 15.
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + (Z_1 - \alpha_i) y_i \boldsymbol{x}_i.$
    $\alpha_i \leftarrow Z_1, \alpha_i' \leftarrow Z_2$
  end for
end while

最后，作者通过下面的定理证明了上述算法是线性收敛的。

Theorem 7. *算法 16 生成的序列 $\{\boldsymbol{\alpha}^s\}$ 会收敛到唯一的全局最优解 $\boldsymbol{\alpha}^*$，同时存在 $0 < \mu < 1$ 和一个 s^0 使得 (设目标函数为 D^{LR})*

$$D^{LR}(\boldsymbol{\alpha}^{s+1}) - D^{LR}(\boldsymbol{\alpha}^*) \leq \mu(D^{LR}(\boldsymbol{\alpha}^s) - D^{LR}(\boldsymbol{\alpha}^*)), \forall s \geq s_0$$

3.6 求解带 L1 正则的 L2-loss SVM 原问题

3.7 求解带 L1 正则的 LR 原问题

3.8 求解 One-class SVM 对偶问题

3.9 求解 Crammer-Singer 多分类 SVM

参考文献

[1] https://zh.wikipedia.org/wiki/SVM

[2] 李航. 统计学习方法. 清华大学出版社, 2012.

[3] https://welts.xyz/2021/08/18/iteralgo/

[4] https://welts.xyz/2021/12/19/cg/

[5] 李庆扬. 数值分析. 清华大学出版社, 2001.

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

[7] `https://welts.xyz/2021/12/18/tr/`

[8] Hsieh, Cho-Jui, et al. "A dual coordinate descent method for large-scale linear SVM." Proceedings of the 25th international conference on Machine learning. 2008.

[9] `https://www.csie.ntu.edu.tw/~cjlin/talks/msri.pdf`

[10] `https://welts.xyz/2021/12/02/dcdm/`

[11] Lin, Chih-Jen, Ruby C. Weng, and S. Sathiya Keerthi. "Trust region Newton method for large-scale logistic regression." Journal of Machine Learning Research 9.4 (2008).

[12] `https://welts.xyz/2021/12/19/tron/`

[13] Ho, Chia-Hua, and Chih-Jen Lin. "Large-scale linear support vector regression." The Journal of Machine Learning Research 13.1 (2012): 3323-3348.

[14] `https://welts.xyz/2022/01/21/svr/`

[15] Chang, Kai-Wei, Cho-Jui Hsieh, and Chih-Jen Lin. "Coordinate descent method for large-scale l2-loss linear support vector machines." *Journal of Machine Learning Research* 9.7 (2008).

[16] `https://welts.xyz/2022/01/18/cdm/`

[17] Yu, Hsiang-Fu, Fang-Lan Huang, and Chih-Jen Lin. "Dual coordinate descent methods for logistic regression and maximum entropy models." *Machine Learning* 85.1-2 (2011): 41-75.

[18] `https://welts.xyz/2022/01/26/dcd_lr/`