桂林電子科技大學

GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

机器学习支持向量机算法综述

课　　程：机器学习

课　　号：2222231

学　　号：2000500927

姓　　名：吴河山

学　　院：计算机与信息安全

专　　业：计算机科学与技术

指导老师：雷春晓

2023 年 5 月

摘 要

支持向量机（SVM）是一种基于统计学习理论的二分类模型，它的目标是寻找一个最大化几何间隔的线性超平面，将不同类别的样本分开。SVM 在计算机视觉、自然语言处理、生物信息学等领域有广泛的应用。本文首先阐述了 SVM 的基本理论，然后概述了 SVM 的各种算法，如块算法、分解算法、序列最小优化算法、最小二乘支持向量机、模糊支持向量机和粒度支持向量机等。接着介绍了 SVM 在各个领域的应用情况，最后对 SVM 的研究问题和发展趋势进行了展望。

关 键 词：神经网络与机器学习，优化算法，梯度下降，态度

目 录

摘 要 ..... **I**
1 引言 **1**
2 研究背景 **2**
2.1 **逻辑回归问题** 2
2.2 **无约束优化** 2
2.2.1 最优化问题 2
2.2.2 无约束优化问题的基本算法 3
2.3 **梯度类算法** 3
2.3.1 线搜索类算法 3
2.3.2 梯度类算法 4
2.3.3 最大似然估计 4
3 研究现状 **7**
3.1 **线搜索** 7
3.2 Barzilai-Borwein **梯度法** 7
3.3 **自适应步长的梯度下降方法** 8
3.4 **一种新型非单调线搜索技术** 9
3.5 **最终改进形式** 10
4 数值实验 **13**
5 小结 **17**
参考文献 **18**
附录 A 公式定理证明 **19**
附录 B 算法 **21**
B.1 代码 21

1 引言

机器学习是一门利用数据中的规律来预测未知或难以观察的数据的学科，它的一个重要理论基础是统计学。统计学习理论 [1] 针对有限样本情况下的机器学习问题，提出了一种新的通用学习方法，叫做支持向量机 (support vector machines,SVM)。它采用结构风险最小化原则，而不是传统统计学的经验风险最小化原则 [2, 3]，在解决小样本、非线性和高维模式识别问题中表现出许多特有的优势，并在很大程度上克服了“维数灾难”和“过学习”等问题，在当时表现出许多优于已有方法的性能，迅速引起各领域的注意和研究兴趣，取得了大量的应用研究成果，推动了各领域的发展。

SVM 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。SVM 是由 AT&T 贝尔实验室的 Vladimir Vapnik 和他的同事 [Cortes and Vapnik,1995[2]] 开发的，是基于统计学习框架或 Vapnik(1982，1995) 和 Chervonenkis(1974) 提出的 VC 理论 [4] 的最稳健的预测方法之一。

SVM 的目标是寻找一个最大化几何间隔的线性超平面，将不同类别的样本分开。SVM 利用核函数将原始特征空间映射到更高维的特征空间，在此特征空间中构造线性决策面。决策面的特殊性质保证了学习机的高泛化能力从而实现非线性分类 [2]。SVM 的优化问题可以通过拉格朗日乘子法和二次规划求解，也可以采用一些高效的算法，如序列最小优化 (SMO) 算法。SVM 在计算机视觉、自然语言处理、生物信息学等领域 [5] 有广泛的应用 [6]。本文首先对 SVM 的理论进行系统的介绍，通过实验例子展现 SVM 算法的效果，同时阐述 SVM 在各个领域的应用情况，并对未来的研究方向进行展望。

2 研究背景

2.1 逻辑回归问题

将数据进行分类是机器学习中的一项常见任务。假设某些给定的数据点各自属于两个类之一，而目标是确定新数据点将在哪个类中。对于支持向量机来说，数据点被视为 p 维向量，而我们想知道是否可以用 $(p-1)$ 维超平面来分开这些点。这就是所谓的线性分类器。可能有许多超平面可以把数据分类。最佳超平面的一个合理选择是以最大间隔把两个类分开的超平面。因此，我们要选择能够让到每边最近的数据点的距离最大化的超平面。如果存在这样的超平面，则称为最大间隔超平面，而其定义的线性分类器被称为最大间隔分类器，或者叫做最佳稳定性感知器。

实际上，此逻辑回归问题可以简化为最优化问题当中的无约束优化问题。

2.2 无约束优化

2.2.1 最优化问题

2.2.1.1 无约束优化问题

作为优化领域的重要分支和约束优化领域的基础部分，无约束优化问题的核心问题在于寻求高效的数值求解方法。其对自变量 x 的取值范围不施加任何约束条件，因此无需考虑 x 的可行性，且很多无约束优化问题的思想可以较好的推广到其他的优化问题上去。

随着大型计算机和工作站的出现，诸多高效的最优化算法也随之出现与发展，并在未知变量规模较大的问题上取得了较好的效果。

无约束优化问题主要研究：

$$\min f(x), x \in \mathbb{R}^n \tag{2-1}$$

的最优性条件，它包括一阶条件和二阶条件。其中 $f : \mathbb{R}^n \to \mathbb{R}$ 称为目标函数，x 是变量。设 x_k 为当前迭代点，$g_k = \nabla f(x_k), G_k = \nabla^2 f(x_k)$ 分别表示函数 f 在 x_k 点处的梯度与 Hessian 矩阵。

无约束优化问题的极小点类型主要有局部较小点和全局极小点两种。全局极小点一定是局部极小点，反之不然，一般来说，求解全局极小点相当困难，因此我们通常把求解局部极小点作为实际应用当中的需求。

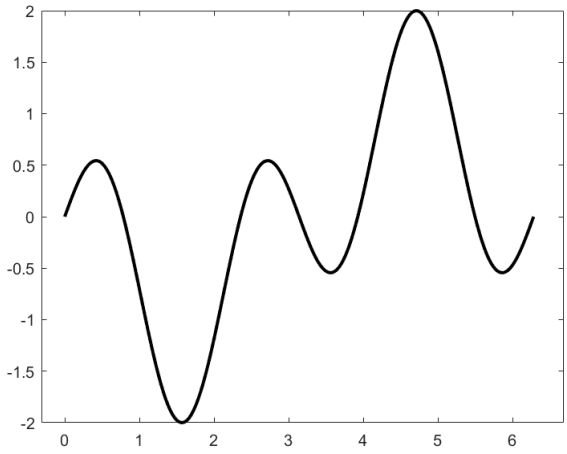


图 2-1 求解全局优化的一个例子

2.2.2 无约束优化问题的基本算法

在数值优化中，一般采取迭代法求解无约束优化问题的极小点。迭代法的基本思想是：给定一个初始点 x_0，按照某一迭代规则产生一个迭代序列 $\{x_k\}$。使得若该序列是有限的，则最后一个点就是问题2-1的极小点；否则，若 $\{x_k\}$ 是无穷点列时，它有极限点且这个极限点即为问题2-1的极小点。

无约束优化算法问题的优化算法主要分为两大类：线搜索类型的优化算法和信赖域类型的的优化算法。他们都是对函数 $f(x)$ 在局部进行近似，但处理近似问题的方式不同。线搜索类算法根据搜索方向的不同可以分为梯度类算法、次梯度算法、牛顿算法、拟牛顿算法等. 一旦确定了搜索的方向，下一步即沿着该方向寻找下一个迭代点. 而信赖域算法主要针对 $f(x)$ 二阶可微的情形，它是在一个给定的区域内使用二阶模型近似原问题，通过不断直接求解该二阶模型从而找到最优值点 [?]

在本次报告中，我们则主要关心线搜索类算法中的梯度类算法。

2.3 梯度类算法

2.3.1 线搜索类算法

我们首先需关心线搜索类算法，其数学表述为：给定当前的迭代点 x_k，首先通过某种算法选取向量 d_k，之后确定正数 α_k，则下一步的迭代点可写作

$$x_{k+1} = x_k + \alpha_k d_k \tag{2-2}$$

我们称 d_k 为迭代点 x_k 处的搜索方向，α_k 为相应的步长。这里要求 d_k 是一个下降方向，即 $(d_k)^T \nabla f(x_k) < 0$。这个下降性质保证了沿着此方向搜索函数 f 的值会减小。线搜索类算法的关键是如何选取一个好的方向 $d_k \in \mathbb{R}^n$ 以及合适的步长 α_k。

下面，我们给出求解无约束优化问题2-1的线搜索类算法一般框架如下:

1: 给定初始点 $x_0 \in \mathbb{R}^n$, 令 $k := 0$
2: 计算 $g_k = \nabla f(x_k)$, 若 $\|g_k\| \leq \epsilon$, 停算, 输出近似极小点 $x^* \approx x_k$
3: 依照一定规则，确定下降方向 d_k.
4: 计算步长因子 α_k，使其满足某个下降规则
5: 置 $x_{k+1} = x_k + \alpha_k d_k, k := k + 1$, 转步骤 2.

算法 2-1 (一般下降算法)

2.3.2 梯度类算法

梯度类算法 (Gradient Method) 以负梯度作为搜索方向, 成为所有需计算导数的优化算法中最简单的方法，也是最优化中最基本的算法。梯度类算法所需内存少，对于求解大规模问题较为适用。

梯度法中步长的选取至关重要, 最古老也是最经典的最速下降法 (Steepest Descent Method), 源自 1847 年的 Cauchy[？], 迭代过程中每次都需经过线搜索求得步长因子, 但是在实际问题当中，最速下降方向仅是算法的局部性质，并非“最速下降”，而是下降非常缓慢。数值实验证明, 当目标函数的等值线是一个扁长的圆球时，最速下降法开始几步下降较快，后来就出现锯齿现象，下降十分缓慢 [？？]。

一些最新的研究 [？？] 还证明了其具有的一些新的性质。

Barzilai 和 Borwein[？] 于 1988 年对梯度类算法提出的改进两点步长梯度法 (Barzilai-Borwein Gradient Method)，因其对于梯度类算法计算效率的极大改善，再度引发了学界关于梯度法的研究热潮，对其进行了大量效果优异的改进，这也是本次实验报告所关注的内容。

2.3.3 最大似然估计

最大似然估计是一种统计方法，它用来求一个样本集的相关概率密度函数的参数。这个方法最早是遗传学家以及统计学家罗纳德·费雪爵士在 1912 年至 1922 年间开始使用的。“似然”是对 likelihood 的一种较为贴近文言文的翻译，“似然”用现代的中文来说即“可能性”。故而，若称之为“最大可能性估计”则更加通俗易懂。

最大似然法明确地使用概率模型，其目标是寻找能够以较高概率产生观察数据的系统发生树。最大似然法是一类完全基于统计的系统发生树重建方法的代表。该方法在每组序列比对中考虑了每个核苷酸替换的概率。例如，转换出现的概率大约是颠换的三倍。在一个三条序列的比对中，如果发现其中有一列为一个 C，一个 T 和一个 G，我们有理由认为，C 和 T 所在的序列之间的关系很有可能更接近。由于被研究序列的共同祖先序列是未知的，概率的计算变得复杂；又由于可能在一个位点或多个位点发生多次替换, 并且不是所有的位点都是相互独立，概率计算的复杂度进一步加大。尽管如此，还是能用客观标准来计算每个位点的概率，计算表示序列关系的每棵可能的树

表 2-1 表题也是五号字

Interference	DOA (deg)	Bandwidth (MHz)	INR (dB)
1	−30	20	60
2	20	10	50
3	40	5	40

的概率。然后，根据定义，概率总和最大的那棵树最有可能是反映真实情况的系统发生树。

例 2.1： 随机变量 $Z_i=(X_i,Y_i), i=1,2,\cdots,n$ 属于独立同分布. X_i,Y_i 有两个可能的取值 0 或者 1. $P(X_i=1)=\alpha, P(Y_i=1|X_i)=\beta X_i$ （一般地，X_i 与 Y_i 不是相互独立）. n 是一个正整数，$0<\alpha<1, 0<\beta<1$ 是未知参数，请解答下面的问题:

(1) 确定 $P(X_i,Y_i)$ 在 (x,y) 取所有可能的值的分布律，可用 α 和 β 表示;
(2) $Z_i, i=1,2,\cdots,n$, 利用所有 Z_i 去估计 α 和 β 的最优估计 $\hat{\alpha}_n$ 和 $\hat{\beta}_n$;
(3) 假设 $\alpha+\beta=1$，求在该约束条件下，利用 Z_i 计算 α 的最优估计量 $\hat{\alpha}_n$;
(4) 在第 (3) 问中，当 $n\to\infty$，$\hat{\alpha}_n$ 收敛于一个值，求该值. ♦

解：(1) $Z_i=(X_i,Y_i)$ 的分布律为

(X_i,Y_i)	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
p_k	$1-\alpha$	0	$\alpha(1-\beta)$	$\alpha\beta$

(2) 把以上分布律写成一个通式

$$P\{X_i=x\}=\alpha^x(1-\alpha)^{1-x}, \tag{2-3a}$$
$$P\{Y_i=y|X_i=x\}=(\beta x)^y(1-\beta x)^{1-y}, \tag{2-3b}$$
$$P\{X_i=x,Y_i=y\}=\alpha^x(1-\alpha)^{1-x}(\beta x)^y(1-\beta x)^{1-y}. \tag{2-3c}$$

如果将式子(2-3b)、(2-3c)中 0^0 定义为 1, 那么通式(2-3c)和联布律表格的意思相同. 但是 0^0 在教材上是没有意义的，可恶的 0^0!

不要慌，不要慌，太阳落下有月光;
不要慌，不要慌，抄手吃完还有汤.

当 $x=0,y=0$ 时, 底数 βx 增加一个量 $1-x-y$, 这样底数就不是 0, 也就不会产生 0^0. 于是(2-3b)、(2-3c)可改写为

$$P\{Y_i=y|X_i=x\}=[(\beta-1)x+1-y]^y(1-\beta x)^{1-y}, \tag{2-3b'}$$
$$P\{X_i=x,Y_i=y\}=\alpha^x(1-\alpha)^{1-x}[(\beta-1)x+1-y]^y(1-\beta x)^{1-y}. \tag{2-3c'}$$

至此, 通式(2-3c')和分布律表格的意思完全相同.

于是，似然函数为

$$\begin{aligned} L(\alpha,\beta) &= \prod_{i=1}^{n} P\{X_i = x_i, Y_i = y_i\} \\ &= \prod_{i=1}^{n} \alpha^{x_i}(1-\alpha)^{1-x_i}[(\beta-1)x_i + 1 - y_i]^{y_i}(1-\beta x_i)^{1-y_i} \\ &= \alpha^{\sum\limits_{i=1}^{n} x_i}(1-\alpha)^{n-\sum\limits_{i=1}^{n} x_i}\beta^{\sum\limits_{i=1}^{n} x_i y_i}(1-\beta)^{\sum\limits_{i=1}^{n} x_i(1-y_i)} \end{aligned} \tag{2-5}$$

注意上式中 $\sum\limits_{i=1}^{n} x_i y_i$ 表示 n 个样本取值中 $(1,1)$ 的个数；$\sum\limits_{i=1}^{n} x_i(1-y_i)$ 表示 n 个样本取值中 $(1,0)$ 的个数.

为了求 $L(\alpha,\beta)$ 的最大值点, 先对(2-5)两边取对数, 得

$$\ln L(\alpha,\beta) = \sum_{i=1}^{n} x_i \ln\alpha + (n - \sum_{i=1}^{n} x_i)\ln(1-\alpha) + \sum_{i=1}^{n} x_i y_i \ln\beta + \sum_{i=1}^{n} x_i(1-y_i)\ln(1-\beta).$$

可求得最大值点, 即最大似然估计值为

$$\hat{\alpha}_n = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \hat{\beta}_n = \frac{\sum\limits_{i=1}^{n} x_i y_i}{\sum\limits_{i=1}^{n} x_i}.$$

(3) 当 $\beta = 1-\alpha$ 时, 似然函数 $L(\alpha,\beta)$ 可改写为

$$\ln L(\alpha, 1-\alpha) = (2\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i y_i)\ln\alpha + (n - \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} x_i y_i)\ln(1-\alpha).$$

此时, α 的最大似然估计值为

$$\hat{\alpha}_n = \frac{2\sum\limits_{i=1}^{n} x_i - \sum\limits_{i=1}^{n} x_i y_i}{n + \sum\limits_{i=1}^{n} x_i}.$$

(4) 注意到，当 $n\to\infty$ 时, $\frac{1}{n}\sum\limits_{i=1}^{n} x_i \to P\{X=1\} = \alpha$, $\frac{1}{n}\sum\limits_{i=1}^{n} x_i y_i \to P\{X=1, Y=1\} = \alpha(1-\alpha)$ （频率趋于概率）. 因此

$$\lim_{n\to\infty}\hat{\alpha}_n = \frac{2\alpha - \alpha(1-\alpha)}{1+\alpha} = \alpha.$$

3 研究现状

3.1 线搜索

线搜索通常分为精确线搜索和不精确线搜索。如果计算 α_k，使得

$$f(x_k+\alpha_k d_k)=\min_{\alpha>0} f(x_k+\alpha d_k) \tag{3-1}$$

寻找步长条件成立的方式称作精确线搜索技术。但是在实际计算中，对于一维问题的求解较为困难，所需迭代次数过多，在实际使用中，不精确线搜索使用更为广泛。在不精确线搜索中，只要函数值 $f(x)$ 在新的迭代点 $x_k+\alpha d_k$ 有一定下降即可。

常用的不精确下降准则有以下几种。

- Armijo 条件

$$f(x_k+\alpha_k d_k)\geq f(x_k)+\rho\alpha_k g_k^T d_k$$

- Wolfe 条件

$$\begin{aligned} f(x_k+\alpha_k d_k)&\geq f(x_k)+\sigma_1\alpha g_k^T d_k\\ g(x_k+\alpha_k d_k)^T d_k&\leq \sigma_2 g_k^T d_k; \end{aligned}$$

- 强 Wolfe 条件

$$\begin{aligned} f(x_k+\alpha_k d_k)&\geq f(x_k)+\sigma_1\alpha g_k^T d_k\\ |g(x_k+\alpha_k d_k)^T d_k|&\leq \sigma_2 g_k^T d_k; \end{aligned}$$

其中 $\rho\in(0,1), 0<\sigma_1<\sigma_2<1$.

针对约束优化问题，还有一种重要的思想，那就是非单调技术。非单调算法是一种不要求每次迭代结果严格下降却能获得更好收敛性的算法。Chamberlain 等人 [?] 于 1982 年针对约束优化问题提出了 watchdog 这一非单调技术的重要雏形。Grippo 等人 [?] 在 1986 年提出了一个最初的非单调技术。自此，非单调技术开始飞速发展。

3.2 Barzilai-Borwein 梯度法

Barzilai-Borwein(BB) 梯度法 [?] 是由加拿大皇家科学院院士、数学会前会长 Borwein 与其合作者 Barzilai 在 1988 年提出的一种新型梯度算法，当问题较为病态时，梯度下降法的收敛性质往往会收到很大影响，而 BB 方法的计算效果比最速下降法好很多，现已成为解决大规模问题的极具竞争力的一种方法。

BB 步长目前在求解无约束优化问题的其它方法中也有广泛的应用, 例如信赖域方法、共轭梯度法、变尺度 BFGS 方法和尺度化的共轭梯度法。此外, Barzilai-Borwein

方法也用于求解其他类型的优化问题, 如约束优化问题、多目标规划、非光滑优化问题、非线性方程组的求解与张量特征值问题。

BB 方法仅适用当前迭代点与上一步迭代点的信息来确定步长, 把迭代公式 $x_{k+1} = x_k - \alpha_k g_k$ 看成是

$$x_{k+1} = x_k - D_k g_k$$

其中 $D_k = \alpha_k I$. 为了使矩阵 D_k 具有拟牛顿性质，求解问题

$$\min_{\alpha_k} \|s_{k-1} - D_k y_{k-1}\|_2$$

或者

$$\min_{\alpha_k} \|D_k^{-1} s_{k-1} - y_{k-1}\|_2$$

容易验证问题的解分别为

$$\alpha_k^{\mathrm{BB1}} \overset{\mathrm{def}}{:=} \frac{\left(s^{k-1}\right)^T y^{k-1}}{\left(y^{k-1}\right)^T y^{k-1}} \quad \text{和} \alpha_k^{\mathrm{BB2}} \overset{\mathrm{def}}{:=} \frac{\left(s^{k-1}\right)^T s^{k-1}}{\left(s^{k-1}\right)^T y^{k-1}}, \tag{3-2}$$

因此可以得到 BB 方法的两种迭代格式:

$$\begin{aligned} x_{k+1} &= x_k - \alpha_k^{BB1} \nabla f(x_k) \\ x_{k+1} &= x_k - \alpha_k^{BB2} \nabla f(x_k) \end{aligned}$$

我们从式3-2注意到，计算两种 BB 步长的任何一种仅仅需要函数相邻两步的梯度信息和迭代点信息，不需要任何线搜索算法即可选取算法步长。因为这个特点，BB 方法的使用范围特别广泛。对于一般的问题，通过式3-2计算出的步长可能过大或过小，因此我们还需将步长做上界和下界的截断，即选取 $0 < \alpha_m < \alpha_M$ 使得

$$\alpha_m \leq \alpha_k \leq \alpha_M$$

还需注意的是，BB 方法本身是非单调方法，有时也配合非单调收敛准则使用以获得更好的实际效果，我们可以通过图3-1了解到 BB 方法相对于 BFGS 法的优异效果。

下面我们给出本次实验中非单调线搜索的 BB 方法的伪代码:

算法 3-1 (非单调线搜索的 BB 方法)

(其中，$\max_{0\leqslant j\leqslant \min(k,M)} f\left(x^{k-j}\right)$ 使用回溯法寻找局部最大值)

3.3 自适应步长的梯度下降方法

BB 方法在数值实验中取得了较梯度下降方法优异得多的表现。对于严格凸二次极小化问题

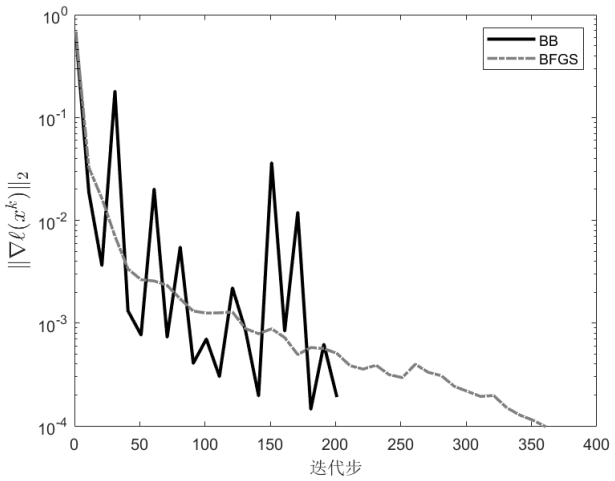


图 3-1 BB 方法与 BFGS 法相对比

$$\min_{x\in\mathbb{R}} \frac{1}{2}x^T Ax - b^T x$$

其中 $A \in \mathbb{R}^{n\times n}$ 是对称正定矩阵，$b \in \mathbb{R}$。Barzilai 和 Borwein[?] 证明了 $n = 2$ 时方法的 R-超线性收敛性。对于 $n \leq 3$ 的情形，Raydan[?] 得到了全局收敛性的结果，Dai 和 Liao[?] 进一步证明了方法是 R-线性收敛的。Dai 等人在 [? ?] 讨论了 BB 型方法的渐进收敛性。借助于非单调线搜索 [?],Raydan[?] 把 BB 方法应用到了求解一般非二次函数的极小化问题中，大量实验表明此时的 BB 方法可以和 PRP+[?] 方法相媲美。

在以上这些方法中，我们更关注于自适应 BB 方法 (Adaptive Barzilai–Borwein,ABB)[?]。

$$\alpha_k^{\mathrm{ABB}} = \begin{cases} \alpha_k^{\mathrm{BB2}}, & \text{if} \quad \dfrac{\alpha_k^{\mathrm{BB2}}}{\alpha_k^{\mathrm{BB1}}} < \tau \\ \alpha_k^{\mathrm{BB1}}, & \text{otherwise} \end{cases} \tag{3-3}$$

ABB 方法使用一种类似信赖域的策略，从原始 BB 方法的两种替代公式中选择其步长，且其作为非单调算法，对于一般函数无需行行搜索，在计算无约束优化问题时可以节省大量的计算工作。在系数矩阵条件较为恶劣，并对于高精度有要求的时候，ABB 方法是一个很好的备选。

为了保证全局的收敛性，我们将在下部分将 ABB 方法与 Zhang 和 Hager 提出的一种新型非单调线搜索技术 [?] 相结合。

3.4 一种新型非单调线搜索技术

Zhang 和 Hager 提出了一种新型非单调线搜索技术 [?]，在这项技术中要求连续函数值的平均值减小，而在 Grippo 等人的工作 [?] 中则要求最近点的函数最大值减少，即从要求

$$f(x_k+\alpha_k d_k) \leq \max_{0\leqslant j\leqslant \min(k,M)} f(x_{k-j}) + \delta\alpha_k \nabla f(x_k) d_k$$

转变为要求

$$f(x_k+\alpha_k d_k) \leq C_k + \delta\alpha_k \nabla f(x_k) d_k$$

其中，C_k 满足一定的迭代条件。这种非单调线搜索技术具有非凸、光滑函数的全局收敛性，以及强凸函数的 R-线性收敛性。相比 L-BFGS 方法，该技术使用更少的函数值和梯度值计算, 因此性能更加优越。

下面我们给出这种新型非单调线搜索技术 (Nonmonotone Line Search Algorithm,NLSA)：

- 初始化：选定初始点 x_0，和参数 $0 \leq \alpha_{min} \leq \alpha_{max} \leq 1, 0 < \delta < \sigma < 1 < \rho, \mu > 0$，设定 $C_0 = f(x_0), Q_0 = 1, k = 0$.
- 收敛性测试：若 $\|\nabla f(x_k)\|$ 充分小，则停止迭代.
- 升级版线搜索：令 $x_{k+1} = x_k + \alpha_k d_k$, 其中 α_k 需满足 Wolfe 条件

  $$\begin{aligned} f(x_k+\alpha_k d_k) \quad &\leq C_k + \delta\alpha_k \nabla f(x_k) d_k, \\ \nabla f(x_k+\alpha_k d_k) d_k \qquad &\geq \sigma \nabla f(x_k) d_k \end{aligned}$$

  或满足 Armijo 条件：$\alpha_k = \bar{\alpha_k}\rho^{h_k}$, 其中 $\bar{\alpha_k} > 0$ 是试验步，h_k 是满足 $f(x_k + \alpha_k d_k) \leq C_k + \delta\alpha_k \nabla f(x_k) d_k$ 的最大整数，并且 $\alpha_k \leq \mu$
- 升级版成本：在 $\alpha_k \in [\alpha_{\min}, \alpha_{\max}]$，并设定

  $$Q_{k+1} = \alpha_k Q_k + 1, \qquad C_{k+1} = (\alpha_k Q_k C_k + f(x_{k+1}))/Q_{k+1}, \qquad k := k+1.$$

  并且重新返回进行收敛性测试。

3.5 最终改进形式

我们给出基于非单调线搜索的 BB 步长改进的最终版本伪代码：

```
Data: 计算节点 client
Result: 计算文件 test.py
begin
    data = client.recv(1024)
    if data == b'exist' then
        if os.path.exists("test.py") then
            client.send("Y".encode('utf-8'))
        else
            client.send("N".encode('utf-8'))
            while True do
                with open("test.py", "ab") as f
                data = client.recv(1024)
                if data == b'quit' then
                    client.send("received".encode('utf-8'))
                    break
                else
                end
                f.write(data)
            end
            print(" 节点%d：计算文件已经接收！存储为 test.py" % (cnt)) break
        end
    else
    end
end
```

算法 3-2 FileSplitSend()

并在测试函数 Rosenbrock 函数 $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ 上验证了数值效果, 如图3-2所展示。

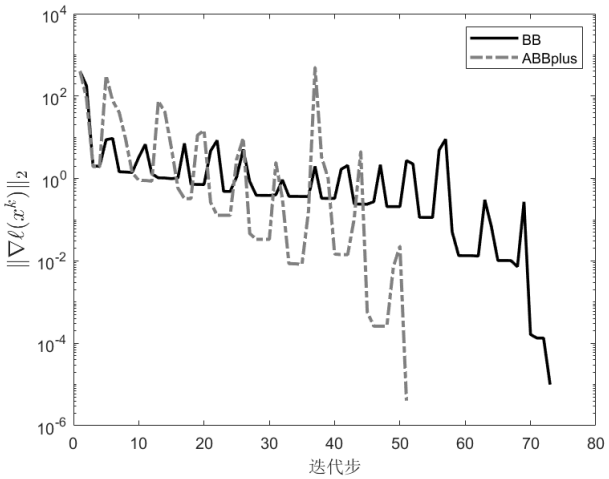


图 3-2 由二次函数生成的 $\{\|g_k\|\}$ 序列

4 数值实验

本节, 我们通过数值试验来阐述以上几项改进形式在实际问题当中的数值表现。所有的代码用 MATLAB R2021 编写, 在具有 2.50GHzCPU 处理器, 8.00G 内存, Windows 10 操作系统的个人电脑上运行。

终止准则为

$$\|\nabla f(x_k)\| \leq \epsilon$$

所有算法均通过以下三个数据集‘a9a.test’, ‘CINA.test’, ‘ijcnn1.test’进行数值实验, 并据此进行算法效果对比, 并通过表4-2进行维度展示, 其中我们选取 a9a 数据集作为实验的主数据集。

图4-1展示了由 BB 方法 ($\mu = 1e-2/m$) 产生的梯度序列 $\{\|g_k\|\}$(取 $\log_{10}$ 对数之后) 随迭代步数变化的情况。图形阐述了 BB 型方法的非单调性, 也在某种意义上说明了方法的 R-线性收敛性结果.

下面我们将进一步对 BB 方法以及其几个改进方法进行对比, 并通过图4-2展示了各项改进相对于改进前的效果对比。

	迭代次数 (次)	CPU 运行时间 (s)	最终 $\|g_k\|$
BFGS	360	8.9688	9.8513e-05
BB	207	7.6094	9.8505e-05
BBplus	219	4.7656	9.7806e-05
ABB	340	9.8594	9.6564e-05
ABBplus	250	5.1094	9.8973e-05

表 4-1 算法对比

- 各项改进方法均能较好的求解出实际问题且效果良好。
- 相对于 **BB** 方法, 各项改进虽均在迭代步数上有所增加, 但是却在步长选择、计算开销、全局收敛性上有了较大的提升, 我们在图3-2上也可了解到其改进效果。
- 各方法在三个数据集上均表现良好, 如图4-3所示。
- 我们还对自适应步长的 BB 方法 (ABB) 公式3-3中的参数 $\tau \in (0, 1)$ 产生了兴

	a9a	CINA	ijcnn1
A	16281 × 122	3206×132	91701×22
b	16281×1	3206×1	91701×1

表 4-2 数据集维度展示

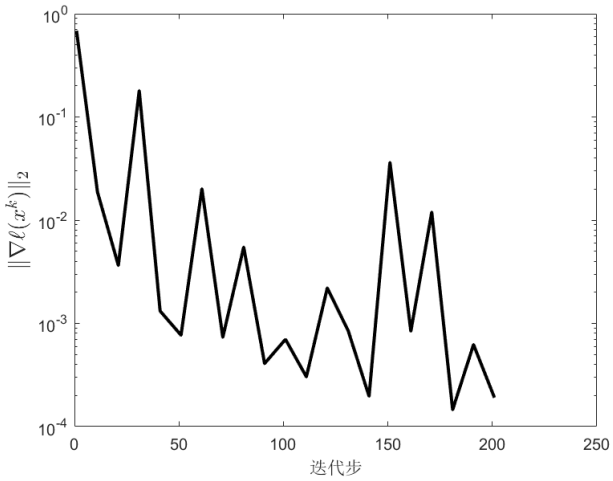
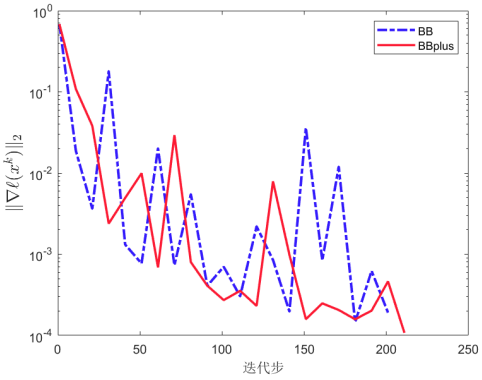
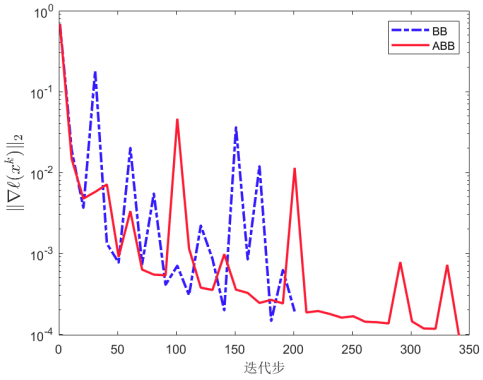


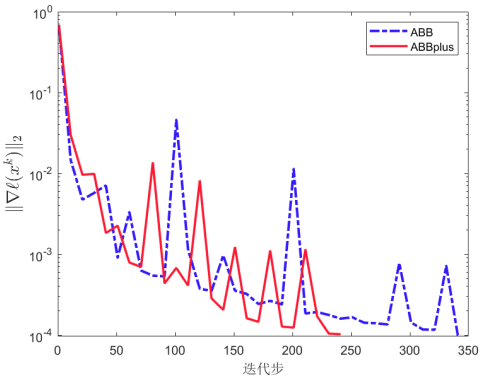
图 4-1 BB 方法 ($\mu = 1e-2/m$) 产生的梯度序列 $\{\|g_k\|\}$



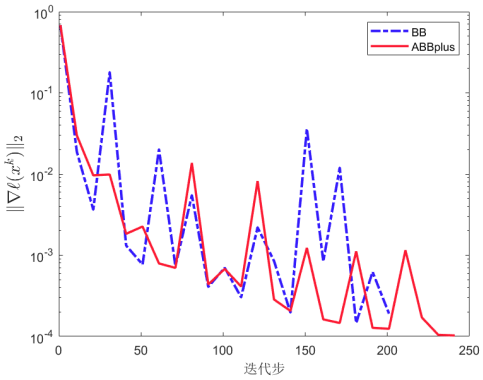
a) BB 与 BBplus



b) BB 与 ABB

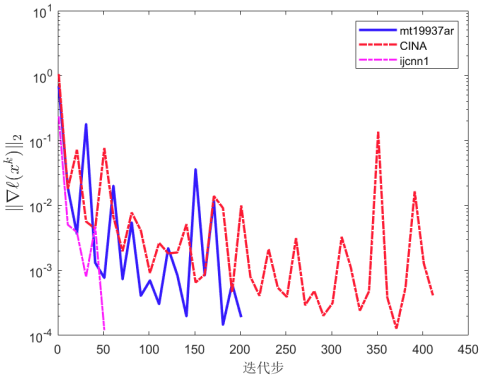


c) ABB 与 ABBplus

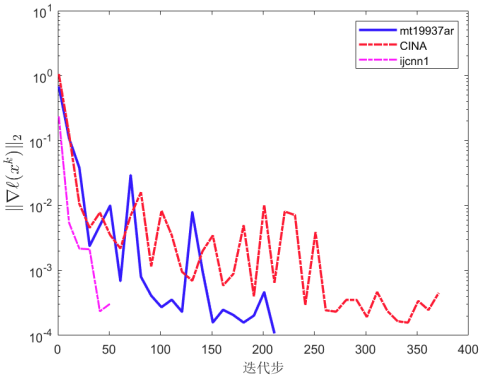


d) BB 与 ABBplus

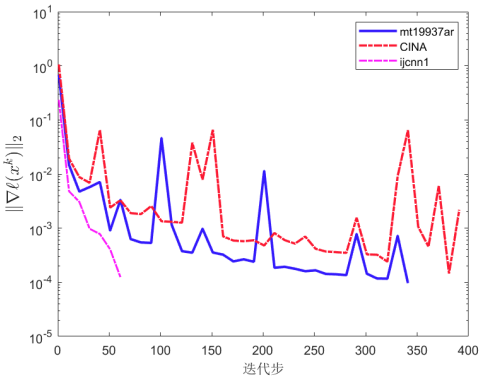
图 4-2 改进的 BB 方法与原方法 $\{\|g_k\|\}$ 序列对比图



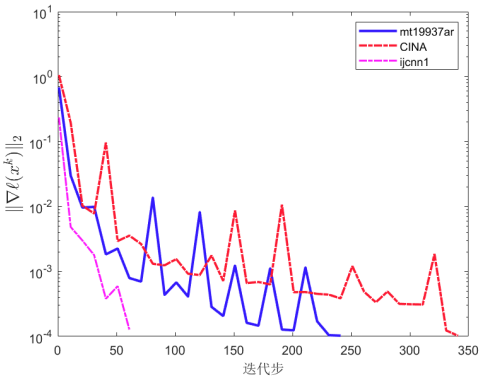
a) BB



b) BBplus



c) ABB



d) ABBplus

图 4-3 三个数据集验证 $\{\|g_k\|\}$ 序列对比图

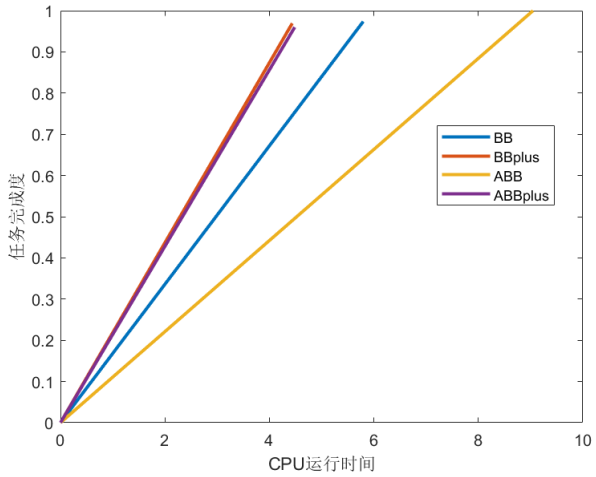


图 4-4 BB 方法及其改进基于 CPU 时间的对比

趣，并对其进行了调整测试，具体结果可由表4-3可知。如无特殊说明，本次实验中 ABB 算法的 $\tau = 0.5$。

	迭代次数 (次)	CPU 运行时间 (s)	最终 $\|g_k\|$
0.1	328	14.0871	9.9887e-05
0.3	287	12.3906	9.9184e-05
0.5	340	9.8594	96564e-05
0.7	296	13.4844	9.7743e-05
0.9	224	9.2813	9.9791e-05

表 4-3 ABB 中参数 τ 调整效果对比

- 我们从表4-1可知，最终改进结果的算法 ABBplus 既保证了在不大规模增加迭代次数的情况下提升了求解精度，又极大的节省了计算开销，计算时间相对于原始 BB 方法而言减少了 48.9% ，相对于 ABB 方法而言更是减少了 93.0%。
- 从图4-4可以观察到，就任务完成度而言，在 CPU 运行时间到达第 5 秒时，算法 BBplus 和 ABBplus 已经分别完成了计算任务的 100% 和 97.9%，而 BB 方法和 BFGS 方法则仅仅完成了整体计算任务的 65.7% 和 55.7%。
- 同时为了算法调优，我们对正则化系数 m 进行了调整，见下表??。我们可以观察到，随着正则化系数的增加，迭代次数大幅度下降，但是精度也随之下降。迭代次数下降的原因我们猜测由于测试函数本身属性的影响，具体原因还有待具体研究。

  从上述调整的参数来看，正则化系数在被选择为 $1/m$ 附近时，迭代次数较少、CPU 运行时间较低，同时维持了一个较好的精度。

5 小结

本次报告主要研究 BB 型步长及其改进形式在优化算法中的应用。首先，我们实现了基于 Grippo 等人提出的非单调技术 [?] 的 BB 型步长 [?] 和自适应步长 BB 方法 (ABB)[?]。其次，我们将由 Zhang 和 Hager 提出的一种新型非单调线搜索技术 [?] 分别应用到 BB 方法和 ABB 方法上，在极大减少 ABB 方法迭代次数的同时，吸收了 BB 方法的非单调及高精度等优点，在多项数值实验上取得了优异的效果。最后，我们还关注了不同数量级正则化系数对于逻辑回归函数的影响，并针对数值结果选取了表现相对优异的系数。

参考文献

[1] 张学工. 关于统计学习理论与支持向量机 [J]. 自动化学报. 2000, 26(1):32–42.
[2] Cortes C, Vapnik V. Support-vector networks[J]. Machine learning. 1995, 20:273–297.
[3] 祁亨年. 支持向量机及其应用研究综述 [J]. 计算机工程. 2004, 30(10):4.
[4] Blumer A, Ehrenfeucht A, Haussler D, et al. Learnability and the vapnik-chervonenkis dimension[J]. Journal of the ACM (JACM). 1989, 36(4):929–965.
[5] 顾亚祥, 丁世飞. 支持向量机研究进展 [J]. 计算机科学. 2011, 38(002):14–17.
[6] 张松兰. 支持向量机的算法及应用综述 [J]. 江苏理工学院学报. 2016, (2):14–17.

附录 A 公式定理证明

附录编号依次编为附录 A，附录 B。附录中的图、表、公式另行编排序号，编号前加“附录 A-”字样。

排版数学定理等环境时最好给环境添加结束符，以明确定理等内容的起止标志，方便阅读。例如定义的结束符采用 ◊，例子的结束符采用 ⧫，定理的结束符采用 □，证明的结束符采用 ■。

定义 A.1 (**向量空间**)**：** 设 X 是一个非空集合，$\mathbb{F}$ 是一个数域 (实数域 $\mathbb{R}$ 或者复数域 $\mathbb{C}$)。如果在 X 上定义了加法和数乘两种运算，并且满足以下 8 条性质：

1. 加法交换律，$\forall\, x, y \in X$，$x + y = y + x \in X$；
2. 加法结合律，$\forall\, x, y, z \in X$，$(x + y) + z = x + (y + z)$；
3. 加法的零元，$\exists\, 0 \in X$，使得 $\forall\, x \in X$，$0 + x = x$；
4. 加法的负元，$\forall\, x \in X$，$\exists\ -x \in X$，使得 $x + (-x) = x - x = 0$。
5. 数乘结合律，$\forall\, \alpha, \beta \in \mathbb{F}$，$\forall\, x \in X$，$(\alpha\beta)x = \alpha(\beta x) \in X$；
6. 数乘分配律，$\forall\, \alpha \in \mathbb{F}$，$\forall\, x, y \in X$，$\alpha(x + y) = \alpha x + \alpha y$；
7. 数乘分配律，$\forall\, \alpha, \beta \in \mathbb{F}$，$\forall\, x \in X$，$(\alpha + \beta)x = \alpha x + \beta x$；
8. 数乘的幺元，$\exists\, 1 \in \mathbb{F}$，使得 $\forall\, x \in X$，$1x = x$，

那么称 X 是数域 $\mathbb{F}$ 上的一个向量空间 (linear space)。 ◊

例 A.1 (**矩阵空间**)**：** 所有 $m \times n$ 的矩阵在普通矩阵加法和矩阵数乘运算下构成一个向量空间 $\mathbb{C}^{m\times n}$。如果定义内积如下：

$$\langle A, B\rangle = \mathrm{tr}(B^H Q A) = \sum_{i=1}^{n} b_i^H Q a_i \tag{A-1}$$

其中 a_i 和 b_i 分别是 A 和 B 的第 i 列，而 Q 是 HPD 矩阵，那么 $\mathbb{C}^{m\times n}$ 构成一个 Hilbert 空间。当 $Q = I$ 时

$$\langle A, B\rangle = \mathrm{tr}(B^H A) \tag{A-2}$$

称为 Frobenius 内积，对应的范数称为 Frobenius 范数，即矩阵所有元素模平方之和再开方：

$$\|A\|_F = \sqrt{\mathrm{tr}(A^H A)} = \sqrt{\sum_{j=1}^{n}\sum_{i=1}^{m} |a_{ij}|^2} \tag{A-3}$$

如果 $m = n$，那么所有 $m \times m$ 的 Hermite 矩阵构成 $\mathbb{C}^{m\times m}$ 的子空间。但是所有 $m \times m$ 的 HPD 矩阵并不构成子空间，因为 HPD 矩阵对线性运算不封闭。 ⧫

定理 A.1 (Riesz **表示定理**)**：** 设 H 是 Hilbert 空间，H^* 是 H 的对偶空间，那么对 $\forall\, f \in H^*$，存在唯一的 $x_f \in H$，使得

$$f(x) = \langle x, x_f \rangle, \qquad \forall\ x \in H \tag{A-4}$$

并且满足 $\|f\| = \|x_f\|$。 □

证明： 先证存在性，再证唯一性，最后正 $\|f\| = \|x_f\|$。 ■

附录 B 算法

B.1 代码

源代码使用 listings 宏包，LMS 算法的 Verilog 模块端口声明如代码 B-1 所示。

代码 B-1 空时 LMS 算法 Verilog 模块端口声明

language

```
int main()
{

    std::string root_path = "E:\\Code\\heils_mobileface\\heils_mobileface\\heils_mobileface\\images\\lack";
    std::string suffix = ".jpg";

    std::vector<std::string> fns;
    glob(fns, root_path, suffix);
    //cout << pattern << endl;
    std::vector<cv::Mat> images;

    size_t face_num = 0;
    MtcnnDface facedetector = MtcnnDface();

    facedetector.initModel();
    for (int i = 0; i < fns.size(); i++) {
        cv::Mat cv_mat = cv::imread(fns[i]);
        ncnn::Mat ncnn_mat = ncnn::Mat::from_pixels(cv_mat.data, ncnn::Mat::PIXEL_BGR2RGB, cv_mat.cols, cv_mat.rows);

        std::vector<Bbox> finboxes;
        ptimer pt;
        //double start = static_cast<double>(cv::getTickCount());
        facedetector.detect(ncnn_mat, finboxes);
        //Sleep(1000);
        double inference = cv::getTickCount();
        std::cout << "boost timer: " << pt.elapsed() << "秒" << std::endl;
        //std::cout << "opencv timer: " << (inference - start) / cv::getTickFrequency() << "秒" << std::endl;

```

```
30          if (finboxes.size() != 0)
31              face_num++;
32          for (auto &box : finboxes)
33          {
34              cv::rectangle(cv_mat, cv::Rect(box.x1, box.y1, box.x2 - box.x1 + 1, box.y2 - box.y1 + 1),
                    cv::Scalar(0, 255, 255), 2);
35              for (int j = 0; j < 5; ++j)
36              {
37                  cv::circle(cv_mat, cvPoint(box.ppoint[j], box.ppoint[j + 5]), 2, CV_RGB(0, 255, 0),
                        CV_FILLED);
38              }
39          }
40          cv::imshow("show", cv_mat);
41          cv::waitKey(0);
42      }
43      std::cout<<"image count" << fns.size() <<"face num:"<< face_num << std::endl;
44
45      facedetector.releaseModel();
46
47      return 0;
48  }
```