# 湖南大學

# HUNAN UNIVERSITY

# 课 程 作 业

| | |
|---|---|
| **论 文 题 目** | 基于 matlab 实现的支持向量机(SVM)算法 |
| 学 生 姓 名 | 何景林 |
| 学 生 学 号 | S230200192 |
| 专 业 班 级 | 机械 2304 班 |
| 课 程 名 称 | 工程优化方法 |
| 学 院 名 称 | 机械与运载工程学院 |
| 课 程 老 师 | 王琥 |
| 学 院 院 长 | 丁荣军 |

2024 年 07 月 16 日

# 基于 matlab 实现的支持向量机(SVM)算法

## 摘　　要

本文基于 matlab 设计一个支持向量机(SVM)用来解决分类问题。设计的支持向量机主要包括二次惩罚、增广拉格朗日和 InteriorPoint Barrier 三种不同的优化方法以及线性、多项式、高斯和 Sigmoid 四种不同的核。同时使用在机器学习领域广泛使用的“一对多”的方法处理多类分类问题。使用的数据集是 matlab 自带的 Fisher Iris 数据，其包含三个类别、四个特征和 150 个样本。取样本中 80% 的数据作为训练集，取样本的 20%数据作为测试集。最后分析了不同优化方法、不同核化函数下、不同 SVM 分类器的收敛速率并对比不同优化方法、不同核化函数下的求解精度；结果表明，在该分类问题中，最佳内核是 RBF，因为其精度一直很高；其中，增广拉格朗日法在所有内核中都具有最好的性能。

**关键词：SVM；matlab；分类**

# 目 录

# 插图索引

# 附表索引

# 第 1 章 SVM 简介

支持向量机（Support Vector Machine，常简称为 SVM）是一种监督式学习的方法，可广泛地应用于统计分类以及回归分析。它是将向量映射到一个更高维的空间里，在这个空间里建立有一个最大间隔超平面。在分开数据的超平面的两边建有两个互相平行的超平面，分隔超平面使两个平行超平面的距离最大化。假定平行超平面间的距离或差距越大，分类器的总误差越小。对于线性可分的任务，找到一个具有最大间隔超平面，如图 1 所示，这个超平面需要满足两个条件：一是能将两类数据点完全分开；二是距离超平面最近的数据点到超平面的距离最大。满足这两个条件的超平面被称为最优超平面，而距离超平面最近的数据点被称为支持向量。

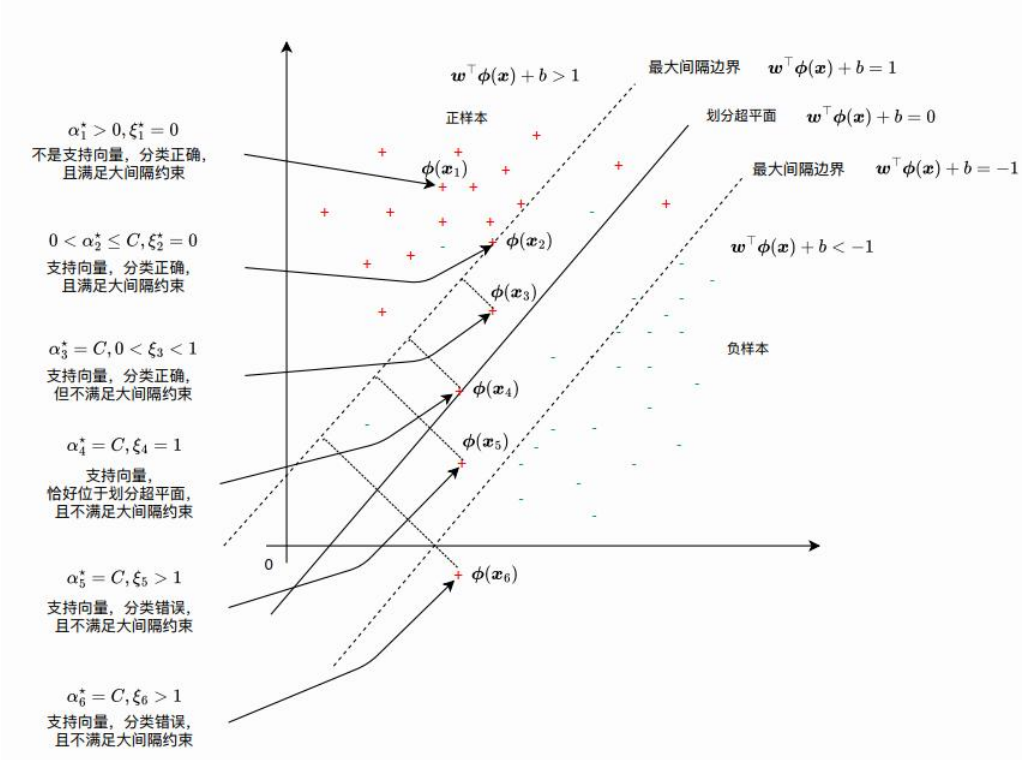



**图 1 最大间隔超平面**

在 SVM 中，我们通常使用核函数来将数据映射到低维空间。常见的核函数包括线性核、多项式核、径向基核等。选择合适的核函数对于 SVM 的性能至关重要。此外，为了处理非线性问题，SVM 还引入了软间隔的概念，即在允许一定误差的情况下寻找最优超平面。

# 第 2 章 基于 SVM 方法的分类问题

## 2.1. 问题描述

本文旨在基于 matlab 设计一个支持向量机(SVM)用来解决分类问题。我使用的数据集是 matlab 自带的 Fisher Iris 数据，其包含三个类别、四个特征和 150 个样本。取样本中 80% 的数据作为训练集，取样本的 20%数据作为测试集。设计的支持向量机主要包括 Quadratic Penalty、Augmented Lagrangian 和 InteriorPoint Barrier 三种不同的优化方法以及 linear、polynomial、RBF 和 Sigmoid 四种不同的核。最后使用在机器学习领域广泛使用的“一对多”的方法处理多类分类问题。

## 2.2. SVM 方法设计

### 2.2.1. 决策规则

通过尽可能优化这些区域（为+1 的区域和-1 的区域）来优化分类器的间隔，定义函数为：

$$f(x) = w * x^{'} + b \tag{1}$$

$$\begin{aligned} f(x) > 1 inregion + 1 \\ f(x) < 1 inregion - 1 \end{aligned} \tag{2}$$

### 2.2.2. 计算间隔宽度

$$x^{+} = x^{-} + r * w \tag{3}$$

式中，$x^{-}$ 是 $f(x^{-}) = -1$ 的最近点，$x^{+}$ 是 $f(x^{+}) = +1$ 最近点，r 是标量倍数，w 是垂直于边界的向量。

位于边缘的点需要满足：

$$\begin{aligned} w \cdot x^{-} + b = -1 \\ w \cdot x^{+} + b = -1 \end{aligned} \tag{4}$$

则，

$$\begin{aligned} & w\cdot(x+rw)+b=+1 \\ \rightarrow & r\|w\|^2+w\cdot x+b=+1 \\ \rightarrow & r\|w\|^2-1=+1 \\ \rightarrow & r=\frac{2}{\|w\|^2} \end{aligned} \tag{5}$$

$$Margin=\|x^+-x^-\|=\|rw\|=\frac{2}{\|w\|^2}\|w\|=\frac{2}{\sqrt{w^Tw}} \tag{6}$$

我们的目标是尽可能地分离出边界。因此，通过使用最大边际分类器，可以得到最优的 w：

$$w^*=\underset{w}{argmax}\frac{2}{\sqrt{w^Tw}} \tag{7}$$

将最大化问题更改为最小化问题，定义为：

$$\begin{aligned} & w^*=argmin_w\sum_j w_j^2 \\ & subjectiveto \\ & y^i=+1\rightarrow w\cdot x^i+b\geq+1 \\ & i^i=-1\rightarrow w\cdot x^i+b\leq+1 \end{aligned} \tag{8}$$

以上是一个具有线性约束的二次代价函数的例，我们需要最小化二次函数的参数，该约束条件可以重写为：

$$y^i(w\cdot x^i+b)\geq+1 \tag{9}$$

其中 $y^i$ 可以取+1 或-1。

### 2.2.3. 拉格朗日优化

通过引入拉格朗日乘子 α（单约束），使难以满足的约束更容易被满足，用拉格朗日乘子来加强不等式约束，拉格朗日方程表示为：

$$\begin{aligned} & L=\frac{1}{2}\sum_j w_j^2+\sum\alpha_i(1-y^i(w.x^i+b)) \\ & w^*=argmin_w argmax_{\alpha\geq0}\frac{1}{2}\sum_j w_j^2+\sum\alpha_i(1-y^i(w\,x^i+b)) \end{aligned} \tag{10}$$

微分拉格朗日方程和关于 w 的平稳条件表示为：

$$\begin{aligned}&\frac{\partial L}{\partial w}=w-\sum\alpha_{ii}x_i=0\rightarrow w^*=\sum_i\alpha_i y_i x_i\\&\frac{\partial L}{\partial b}=-\sum\alpha_i y_i=0\rightarrow\sum\alpha_i y_i=0\end{aligned}\tag{11}$$

转换得，

$$\begin{aligned}&L=\sum_i\alpha_i-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j x_i\cdot x_j\\&w^*=\underset{\alpha\geq0}{argmax}\sum_i\alpha_i-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j x_i\cdot x_j\end{aligned}\tag{12}$$

### 2.2.4. 非完全线性可分的数据处理

为了处理非完全线性可分的数据，我将稍微放宽方约束条件，以允许错误分类的点。使用软间隔为约束并引入松弛变量。

$$\begin{aligned}&w^*=\underset{w,\epsilon}{\operatorname{argmin}}\sum_j w_j^2+R\sum_i\epsilon^i\frac{n!}{r!(n-r)!}\\&subjective\ to\\&y^i(w^Tx^i+b)\geq+1-\ ^i(violate\ margin\ by\ \ )\\&\epsilon\geq0\end{aligned}\tag{13}$$

其中，$\epsilon\geq0$ 是一个松弛变量，$R\sum_i\epsilon^i$ 是惩罚项。

如果 R 取得非常大，我们需要更加注意以确保没有数据违反间隔。如果 R 的选择非常小，我们将最大化大多数数据的间隔，但允许一些数据超过一点。我们总是可以将解 $w$ 、$\epsilon$ 和 B 初始化为某个满足约束的值。

最优 $\epsilon^*$ 作为 w 的函数，可以解析地表示为：

$$J_i=max[0,1-y^i(w^i+b)]\tag{14}$$

代入等式（13）得，

$$w^*=\underset{w}{\operatorname{argmin}}\sum_j w_j^2+R\sum_i J_i(y^i,w\cdot x^i+b)\tag{15}$$

其中，$J_i(y^i,w\cdot x^i+b)$ 是一个铰链损失作为替代损失，如图 2 所示，当 $w+b$ 大于 1 时，铰链损耗等于 0。当 $w+b$ 小于 1 时，铰链损耗呈线性增加。

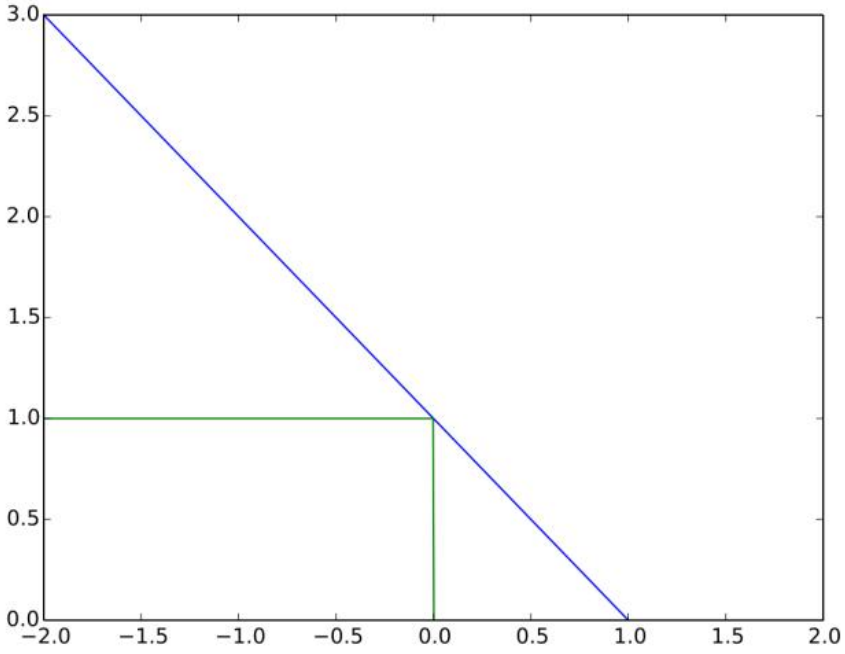



**图 2 铰链损失（蓝色）和零 1 个损失（绿色）**

在这个软间隔 SVM 中，边界错误一侧的数据点有一个惩罚，随着距离它的距离而增加。重新格式化为拉格朗日，如前面所述，我们需要最小化对 $w$，$b$ 和 $\epsilon$，并最大化对 $\alpha$（其中 $\alpha_i \geqslant 0$，$\mu \geqslant 0$），非完全线性可分离的数据的拉格朗日方程可以显示为：

$$L_p \equiv \frac{1}{2}\|w\|^2 + R\sum_{i=1}^{L}{}_i - \sum_{i=1}^{L}\alpha_i[y_i(x_i \cdot w + b) - 1 + {}_i] - \sum_{i=1}^{L}\mu_i{}_i \tag{16}$$

关于 w，b 和 $\epsilon_i$ 和平稳点的区别是：

$$\begin{aligned} \frac{\partial L}{\partial w} &= w - \sum \alpha_{ii} x_i = 0 \rightarrow w^* = \sum_i \alpha_i y_i x_i \\ \frac{\partial L}{\partial b} &= -\sum \alpha_i y_i = 0 \rightarrow \sum \alpha_i y_i = 0 \\ \frac{p}{\partial \epsilon_i} &= 0 \rightarrow R = \alpha_i + \mu_i \end{aligned} \tag{17}$$

现在，将这些导数插入拉格朗日方程，软间隔对偶形式可以显示为：

$$w^* = \underset{0 \leq \alpha \leq R}{argmax} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{18}$$

最大化实际上取决于点积 $x_i \cdot x_j$，因此，我将引入不同类型的内核来处理这个点积。带核的新方程式显示为：

$$w^* = \underset{0 \leq \alpha \leq R}{\text{argmax}} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{19}$$

该方程可以进一步重新格式化为二次规划形式：

$$\begin{aligned} &argmax[\sum_{i=1}^{L}\alpha_i - \frac{1}{2}\alpha^T H\alpha] \\ &s.t. 0 \le \alpha_i \le R \\ &\sum_{i=1}^{L}\alpha_i y_i = 0 \end{aligned} \tag{20}$$

其中，$H_{ij} = y_i y_j x_i \cdot x_j = y_i y_j k(x_i, x_j)$ 。

二次规划（20）的优化可以通过使用 Quadratic Penalty 或 Augmented Lagrangian 法来完成。

支持向量 S 的集合可以通过找到指数来将（$0 \le \alpha_i \le R$）R 设置为 2 来确定。常数 b 可以计算为：

$$\begin{aligned} b &= \frac{1}{N_s}\sum_{s\in S}(y_s - \sum_{m\in S}\alpha_m y_m x_m \cdot x_s) \\ &= \frac{1}{N_s}\sum_{s\in S}(y_s - \sum_{m\in S}\alpha_m y_m k(x_m, x_s)) \end{aligned} \tag{21}$$

式中，$N_s$为支持向量的数量。

### 2.2.5. 优化方法

本文通过四种不同的优化方法优化二次函数（20）。它们是 Quadratic Penalty Method，Augmented Lagrangian Method，InteriorPoint Barrier and quadratic programming package，作为方程（20）的优化方法。使用 quadratic programming package 作为参考，以查看编写的优化方法性能如何。这里的 f (x)是二次规划函数（20）。对 alpha 的最初猜测是 0.2*ones（N，1）。等式约束为$H(x) = Aeq * x$ ，其中，Aeq 是测试集（Y’）的转置，x 是训练集。不等式约束为$0 < \alpha < \text{cost}$，其中，cost 为取 2 的向量。

- 采用牛顿方向的线搜索方法和回溯线搜索方法来解决所有实现方法的每一步的无约束问题。
- 使用$\| x_k - x_{k-1} \| < \epsilon$ 作为所有实现方法的停止标准。

#### 2.2.5.1. 二次惩罚法

$\mu_o = 1$ ，为初始惩罚权重，公差为$10^{-14}$，更新参数 t 为 2。

由于存在两个不等式约束：$\alpha_i - 0 \geq 0$ 和 $R - \alpha \geq 0$ ，其优化函数可以显示为：

$$Q(x;\mu) := f(x) + \frac{\mu}{2}\sum_{i=1}^{p} h_i^2(x) + \frac{\mu}{2}\sum_{i=1}^{m}([f_{1,i}(x)]^+)^2 + \frac{\mu}{2}\sum_{i=1}^{m}([f_{2,i}(x)]^+)^2 \tag{22}$$

其中 $h_i$ 是等式约束和两个 $f_1$ ， $i$ 和 $f_2$ ， $i$ 表示两个不等式约束和 $[f_i]^+ := \max\{f_i, 0\}$ 。这意味着当满足不等式约束时，不等式项将消失为：

*Framework* : for a sequence $\{\mu_k\}: \mu_k \to$ as k $\to$, increasingly pena lising the constraint compute the (approximate, $\| \nabla_x Q(x_k; \mu_k) \| \leq \tau_k, tau \to 0$ sequence $\{x_k\} \to x^*$ of minimiser s $x_k$ *of* $Q$ (x ; $\mu_k$ ).

优化函数的梯度可以表示为：

$$\nabla_x Q(x_k;\mu_k) = \nabla f(x_k) + \sum_{i=1}^{p}\mu_k h_i(x_k)\nabla h_i(x_k) + \sum_{i=1}^{p}\mu_k [f_{1i}(x_k)]^+ [\nabla f_{1i}(x_k)]^+ + \sum_{i=1}^{p}\mu_k [f_{2i}(x_k)]^+ [\nabla f_{2i}(x_k)]^+ \tag{23}$$

优化函数的海塞矩阵表示如下：

$$\begin{aligned}\nabla_{xx}^2 Q(x;\mu_k)p_n = &-\nabla_x^2 f(x) + \sum_{i=1}^{p}\mu_k h_i(x)\nabla^2 h_i(x) + \mu_k \nabla h(x)\nabla h(x)^T \\ &+ \sum_{i=1}^{p}\mu_k [f_{1,i}(x)]^+ \nabla^2 [f_{1,i}(x)]^+ + \mu_k [\nabla f_1(x)]^+ [\nabla f_1(x)^T]^+ \\ &+ \sum_{i=1}^{p}\mu_k [f_{2,i}(x)]^+ \nabla^2 [f_{2,i}(x)]^+ + \mu_k [\nabla f_2(x)]^+ [\nabla f_2(x)^T]^+\end{aligned} \tag{24}$$

2.2.5.2. 增广拉格朗日法

μ= 10 和ν0 是一个大小为（2N+1）的向量，作为一个固定的惩罚权值和初始拉格朗日乘子，公差为$10^{-16}$ 。

优化函数为：

$$\begin{aligned}\mathcal{L}_{\mathcal{A}}(x,\nu,\mu) := &f(x) + \sum_{i=1}^{p}\nu_i h_i(x) + \frac{\mu}{2}\sum_{i=1}^{p} h_i^2(x) \\ &+ \sum_{i=1}^{p}\nu_i [f_{1,i}(x)]^+ + \frac{\mu}{2}\sum_{i=1}^{p}[f_{1,i}^2(x)]^+ \\ &\sum_{i=1}^{p}\nu_i [f_{2,i}(x)]^+ + \frac{\mu}{2}\sum_{i=1}^{p}[f_{2,i}^2(x)]^+\end{aligned} \tag{25}$$

优化函数的梯度为：

$$\begin{aligned}\nabla_x \mathcal{L}_{\mathcal{A}}(x_k,\nu^k;\mu_k) = &\nabla f(x_k) + \sum_{i=1}^{p}[\nu_i^k + \mu_k h_i(x_k)]\nabla h_i(x_k) \\ &+ \sum_{i=1}^{p}[\nu_i^k + \mu_k [f_{1,i}(x)]^+]\nabla [f_{1,i}(x)]^+ \\ &+ \sum_{i=1}^{p}[\nu_i^k + \mu_k [f_{2,i}(x)]^+]\nabla [f_{2,i}(x)]^+\end{aligned} \tag{26}$$

优化函数的海塞矩阵为：

$$\begin{aligned}\nabla^2_{xx}\mathcal{L}_{\mathcal{A}}(x_k,v^k;\mu_k)&=\nabla^2 f(x_k)+\sum_{i=1}^{p}[v^*\nabla^2 h_i(x_k)]+\mu_k\,\nabla h_i(x_k)\nabla h_i(x_k)^T\\&+\sum_{i=1}^{p}[v^*\nabla^2[f_{1,i}(x)]^+]+\mu_k\,\nabla[f_{1,i}(x)]^+\nabla[f_{1,i}(x)^T]^+\\&+\sum_{i=1}^{p}[v^*\nabla^2[f_{2,i}(x)]^+]+\mu_k\,\nabla[f_{2,i}(x)]^+\nabla[f_{2,i}(x)^T]^+\end{aligned}\tag{27}$$

2.2.5.3. InteriorPoint Barrier 法

$\mu_o=1$，为初始惩罚权重，公差为$10^{-10}$，更新参数 t 为 2，最大的迭代次数是 1000，对 alpha 的最初猜测是 0.01*1（N，1）。

优化函数为：

$$\mathcal{I}(x)=tf(x)+\phi(x)\tag{28}$$

其中，

$$\begin{aligned}\phi(x)&=-(\sum_{i=1}^{m}log(-f_{1,i}(x))+\sum_{i=1}^{m}log(-f_{2,i}(x)))\\\mathrm{dom}\phi&=\{x\in\mathcal{R}^n:f_{1,i}(x)<0\mathrm{and}f_{2,i}(x)<0,i=1,\ldots,m\}\end{aligned}\tag{29}$$

优化函数的梯度为：

$$\nabla_x\mathcal{I}(x_k)=t\nabla f(x_k)+\nabla\phi(x)\tag{30}$$

其中，

$$\nabla\phi(x)=\sum_{i=1}^{m}\frac{1}{-f_{1,i}(x)}\nabla f_{1,i}(x)+\sum_{i=1}^{m}\frac{1}{-f_{2,i}(x)}\nabla f_{2,i}(x)\tag{31}$$

优化函数的海塞矩阵为：

$$\nabla^2_{xx}\mathcal{I}(x)=t\nabla^2_x f(x)+\nabla^2\phi(x)\tag{32}$$

其中，

$$\nabla^2\phi(x)=\sum_{i=1}^{m}\frac{1}{f_{1,i}(x)^2}\nabla f_{1,i}(x)\nabla f_{1,i}(x)^T+\sum_{i=1}^{m}\frac{1}{-f_{1,i}(x)}\nabla^2 f_{1,i}(x)+\sum_{i=1}^{m}\frac{1}{f_{2,i}(x)^2}\nabla f_{2,i}(x)\nabla f_{2,i}(x)^T+\sum_{i=1}^{m}\frac{1}{-f_{2,i}(x)}\nabla^2 f_{2,i}(x\tag{33}$$

2.2.6. 标签预测

新的点$x_{new}$通过计算以下方程式进行分类：

$$y_{predication}=sgn(w\cdot x_{new}+b)\tag{34}$$

2.2.7. 核函数

分别引入了如下三种内核来优化方程（19）。

2.2.7.1. 多项式核函数

$$k(a,b)=(1+\sum_j a_j b_j)^d \tag{35}$$

其中，d 为多项式条件，等于 2。

2.2.7.2. 高斯核函数

$$k(a,b)=exp(\frac{-(a-b)^2}{2\sigma^2}) \tag{36}$$

其中，当σ的值很大时，所有的数据点看起来都与内核相似。当σ的值较小时，只有几个点与测试点相似。

2.2.7.3. Sigmoid 核函数

$$k(a,b)=tanh(ca^Tb+h) \tag{37}$$

其中，c 和 h 均等于数据数的倒数。

# 第 3 章 “一对多”的多种类分类

支持向量机（SVM）旨在解决一个二元分类问题。但是，我们使用的数据有三个类，这是一个多类的分类问题。在这里，使用有效地扩展到多类分类的“一对多”方法。这也是 SVM 多类分类中最早使用的。它构造了 k 个 SVM 模型，其中 k 是类的数量（在我们的例子中是 3 个）。第 m 个 SVM 是针对其他两个例子进行训练的。第 m 个 SVM 被考虑与正标签，其他的例子被考虑与负标签。

因此给定 l 个训练数据集 $(x_1,y_1),\ldots,(x_l,y_l)$ ，其中 $x_i \in R^n, i=1,\ldots,l$ 和 $y_i \in \{1,2,3\}$ 是 x 的类。第 m 个 SVM 解决了以下问题：

$$\begin{aligned} &\underset{w^m,b^m,\epsilon^m}{argmin}\frac{1}{2}(w^m)^T w^m + C\sum_{i=1}^{l}\epsilon_i^m \\ &(w^m)^T\phi(x_i)+b^m \geq 1-\epsilon_i^m, if y_i = m, \\ &(w^m)^T\phi(x_i)+b^m -1+\epsilon_i^m, if y_i \neq m, \\ &\epsilon_i^m \geq 0, i=1,\ldots,l, \end{aligned} \tag{38}$$

其中，训练数据 $x_i$ 由函数 $\phi$ 映射到更高维空间，C 为惩罚参数。在这里，我们要想最小化 $\frac{1}{2}(w^m)^T w^m$，这意味着最大化 $\frac{2}{\| w^m \|}$ ，两组数据之间的间隔。因为我们的数据不是线性可分离的，所以我设置了一个极小的项 $C\sum_{i=1}^{l}\epsilon_i^m$ ，用于减少训练错误的数量。SVM 是在正则化 $\frac{1}{2}(w^m)^T w^m$ 和训练错误之间寻找平衡。

求解方程(1)后，由于三个类，有三个决策边界。

$$\begin{aligned} &(w^1)^T\phi(x)+b^1 \\ &(w^2)^T\phi(x)+b^2 \\ &(w^3)^T\phi(x)+b^3 \end{aligned} \tag{39}$$

x 在决策函数的值最大的类中，

$$classofx \equiv \underset{m=1,\cdot,k}{\text{argmax}}((w^m)^T\phi(x)+b^m) \tag{40}$$

多类 SVM 分类器对所有分类器的预测精度可以通过以下方法计算：

- 每个 SVM 分类器将产生一个分数。比较不同 SVM 模型生成的所有分数，预测标签可以由最高分数确定。

$$predlabel_i = find([scores1_i, scores2_i, scores3_i] == max(score1_i, scores2_i, scores3_i)) \quad (41)$$

- 则精度可以计算为：

$$\frac{\sum_i predlabel_i == ytest_i)}{length(ytest)} * 100 \quad (42)$$

# 第 4 章 Matlab 实现

基于前文所述的问题与方法，基于 matlab 实现过程如下：

## 第一步：数据导入与划分

```
%Fisher Iris 数据
load fisheriris
outputs1=[];
outputs2=[];
outputs3=[];
y=grp2idx(species);
N=max(size(species));
for i=1:N
if strcmp(species(i),'setosa')
outputs1=[outputs1;1];
elseif strcmp(species(i),'versicolor')
outputs1=[outputs1;-1];
elseif strcmp(species(i),'virginica')
outputs1=[outputs1;-1];
end
end
for i=1:N
if strcmp(species(i),'versicolor')
outputs2=[outputs2;1];
elseif strcmp(species(i),'setosa')
outputs2=[outputs2;-1];
elseif strcmp(species(i),'virginica')
outputs2=[outputs2;-1];
end
end
for i=1:N
if strcmp(species(i),'virginica')
outputs3=[outputs3;1];
elseif strcmp(species(i),'versicolor')
outputs3=[outputs3;-1];
elseif strcmp(species(i),'setosa')
outputs3=[outputs3;-1];
end
end
X=meas;
y1 = outputs1;
y2 = outputs2;
```

```
y3 = outputs3;
s=rng(3456);
rand_num = randperm(size(X,1));
X_train = X(rand_num(1:round(0.8*length(rand_num))),:);
y_train = y(rand_num(1:round(0.8*length(rand_num))),:);
y1_train = y1(rand_num(1:round(0.8*length(rand_num))),:);
y2_train = y2(rand_num(1:round(0.8*length(rand_num))),:);
y3_train = y3(rand_num(1:round(0.8*length(rand_num))),:);
X_test = X(rand_num(round(0.8*length(rand_num))+1:end),:);
y_test = y(rand_num(round(0.8*length(rand_num))+1:end),:);
y1_test = y1(rand_num(round(0.8*length(rand_num))+1:end),:);
y2_test = y2(rand_num(round(0.8*length(rand_num))+1:end),:);
y3_test = y3(rand_num(round(0.8*length(rand_num))+1:end),:);
```

## 第二步：定义参数

主要包括是多项式核函数的参数、高斯核函数的参数、Sigmoid 核函数的参数精度要求，超参数等。

```
global poly_con gamma kappa1 kappa2 precision Cost
poly_con=2; % 多项式核函数参数
gamma=1/size(X,1);% 高斯核函数参数
kappa1=1/size(X,1);kappa2=kappa1; % Sigmoid 核函数
precision=10^-5;Cost=2;
```

## 第三步：不同核函数和优化方法的 SVM 设计与选择

```
% 选择核函数
kernel=char(Kernel_Cell(1));
% SVM 设计
[alpha1,Ker1,beta01]=SVM(X_train,y1_train,kernel);
[alpha2,Ker2,beta02]=SVM(X_train,y2_train,kernel);
[alpha3,Ker3,beta03]=SVM(X_train,y3_train,kernel);
scores1 = SVM_pred(X_test, X_train, y1_train,kernel,alpha1,beta01);
scores2 = SVM_pred(X_test, X_train, y2_train,kernel,alpha2,beta02);
scores3_fig = SVM_pred(X_test, X_train, y3_train,kernel,alpha3,beta03);
```

### 核函数

线性核

```
function Y=Ker_Linear(X1,X2)
Y=zeros(size(X1,1),size(X2,1));
for i=1:size(X1,1)
for j=1:size(X2,1)
Y(i,j)=dot(X1(i,:),X2(j,:));
end
end
```

```
Return
```

多项式核

```
function Y=Ker_Polynomial(X1,X2)
global poly_con
Y=zeros(size(X1,1),size(X2,1));
for i=1:size(X1,1)
for j=1:size(X2,1)
Y(i,j)=(1+dot(X1(i,:),X2(j,:))).^poly_con;
end
end
return
```

高斯核

```
function Y=Ker_RBF(X1,X2)
global gamma
Y=zeros(size(X1,1),size(X2,1));
for i=1:size(X1,1)
for j=1:size(X2,1)
Y(i,j)=exp(-gamma*norm(X1(i,:)-X2(j,:))^2);
end
end
return
```

Sigmoid 核

```
function Y=Ker_Sigmoid(X1,X2)
global kappa1 kappa2
Y=zeros(size(X1,1),size(X2,1));
for i=1:size(X1,1)
for j=1:size(X2,1)
Y(i,j)=(kappa1*dot(X1(i,:),X2(j,:))+kappa2);
end
end
return
```

## 优化方法

二次惩罚法

```
function [xMin, fMin, t, nIter, infoQP] = Quadratic_Penalty(x0, mu, t, tol, maxIter,N,Aeq,lb,ub,H,f)
% 初始化
nIter = 0;stopCond = false;x_k = x0;infoQP.xs = x_k;infoQP.fs = [];
alpha0 = 1; opts.c1 = 1e-4;opts.c2 = 0.9;opts.rho = 0.5;tolNewton = 1e-12;maxIterNewton = 100;
% 循环
while (~stopCond && nIter < maxIter)
% disp(size(Aeq))
% disp(size(x_k))
```

```
% 为 Q 创建函数处理程序
G.f =@(x) 0.5*x'*H*x + f'*x + (mu/2).*sum((max(lb-x,0)).^2)+ (mu/2).*sum((max(x-ub,0)).^2)+ (mu/2)*sum((Aeq*x).^2);
G.df =@(x) H*x+f + mu*((Aeq*x)*Aeq') + mu*penalty_grad(x,N,lb,ub);
G.d2f =@(x) H + mu*(Aeq'*Aeq);
lsFun = @(x_k, p_k, alpha0) backtracking(G, x_k, p_k, alpha0, opts);
x_k_1 = x_k;
[x_k, f_k, nIterLS, infoIter] = descentLineSearch(G, 'newton', lsFun, alpha0, x_k, tolNewton, maxIterNewton);
% 判断循环终止条件
if norm(x_k - x_k_1) < tol; stopCond = true; end
mu = mu*t;
infoQP.xs = [infoQP.xs x_k];
infoQP.fs = [infoQP.fs f_k];
nIter = nIter + 1;
end
% 赋值
xMin = x_k;
fMin = G.f(x_k);
```

增广拉格朗日

```
function [xMin, fMin, nIter, infoAL] = Augmented_Lagrangian(x0, mu, v0, tol, maxIter,N,Aeq,lb,ub,H,f)
% 初始化
nIter = 0;stopCond = false;x_k = x0;infoAL.xs = x_k;infoAL.fs = [];
v_k=v0;
alpha0 = 1;
opts.c1 = 1e-4;
opts.c2 = 0.9;
opts.rho = 0.5;
tolNewton = 1e-12;
maxIterNewton = 100;

% 循环
while (~stopCond && nIter < maxIter)
disp(strcat('Iteration ', int2str(nIter)));
% 为 Q 创建函数处理程序
% G.f = @(x) 0.5*x'*H*x + f'*x + v_k.*sum(Aeq*x) + (mu/2)*sum((Aeq*x).^2)...
% +(mu/2).*sum((max(lb-x,0)).^2) + v_k.*sum(max(lb-x,0))...
% + (mu/2).*sum((max(x-ub,0)).^2)+ v_k.*sum(max(x-ub,0));
G.f = @(x) 0.5*x'*H*x + f'*x + sum(v_k(1).*(Aeq*x)) + (mu/2)*sum((Aeq*x).^2)...
+(mu/2).*sum((max(lb-x,0)).^2) + sum(v_k(2:(N+1)).*max(lb-x,0))...
+ (mu/2).*sum((max(x-ub,0)).^2)+ sum(v_k((N+2):(2*N+1)).*max(x-ub,0));
G.df = @(x) H*x+f + (v_k(1) + mu*Aeq*x)*Aeq' + augmented_grad(v_k,mu,x,N,lb,ub);
G.d2f = @(x) H + mu*(Aeq'*Aeq);
lsFun = @(x_k, p_k, alpha0) backtracking(G, x_k, p_k, alpha0, opts);
x_k_1 = x_k;
```

```
[x_k, f_k, nIterLS, infoIter] = descentLineSearch(G, 'newton', lsFun, alpha0, x_k, tolNewton, maxIterNewton);
v_k(1) = v_k(1) + mu*Aeq*x_k;
v_k(2:(N+1)) = v_k(2:(N+1)) + mu*max(lb-x_k,0);
v_k((N+2):(2*N+1)) = v_k((N+2):(2*N+1)) + mu*max(x_k-ub,0);
% 判断循环终止条件
if norm(x_k - x_k_1) < tol; stopCond = true; end
infoAL.xs = [infoAL.xs x_k];
infoAL.fs = [infoAL.fs f_k];
nIter = nIter + 1;
end
% 赋值
xMin = x_k;
fMin = G.f(x_k);
```

InteriorPoint Barrier 法

```
function [xMin, fMin, t, nIter, infoBarrier] = interiorPoint_Barrier(F, phi, x0, t, mu, tol, maxIter)
% 初始化
nIter = 0;
stopCond = false;
x_k = x0;
infoBarrier.xs = x_k;
infoBarrier.fs = F.f(x_k);
infoBarrier.inIter = 0;
infoBarrier.dGap = 1/t;

alpha0 = 1;
opts.c1 = 1e-4;
opts.c2 = 0.9;
opts.rho = 0.5;
tolNewton = 1e-12;
maxIterNewton = 100;
% 循环
while (~stopCond && nIter < maxIter)
disp(strcat('Iteration ', int2str(nIter)));
% 为 Q 创建函数处理程序
G.f = @(x) t*F.f(x) + phi.f(x);
G.df = @(x) t*F.df(x) + phi.df(x);
G.d2f = @(x) t*F.d2f(x) + phi.d2f(x);
% 行搜索函数
%lsFun = @(x_k, p_k, alpha0) lineSearch(G, x_k, p_k, alpha0, opts);
lsFun = @(x_k, p_k, alpha0) backtracking(G, x_k, p_k, alpha0, opts);
[x_k, f_k, nIterLS, infoIter] = descentLineSearch(G, 'newton', lsFun, alpha0, x_k, tolNewton, maxIterNewton);
if 1/t < tol; stopCond = true; end
t = mu*t;
```

```
infoBarrier.xs = [infoBarrier.xs x_k];
infoBarrier.fs = [infoBarrier.fs f_k];
infoBarrier.inIter = [infoBarrier.inIter nIterLS];
infoBarrier.dGap = [infoBarrier.dGap 1/t];
nIter = nIter + 1;
end
% 赋值
xMin = x_k;
fMin = F.f(x_k);
```

## 第四步：可视化

```
function SVM_plot(X,Y,alpha,beta0,kernel)
global Cost poly_con gamma kappa1
figure
hold on
P = size(X,2);

if P ~=2
warning('# of input X should be 2 for the 2D visualization!!')
end
plot(X(Y==1,1),X(Y==1,2),'ro',...
'LineWidth', 4,...
'MarkerSize', 4);

plot(X(Y==-1,1),X(Y==-1,2),'bs',...
'LineWidth', 4,...
'MarkerSize', 4);

%
d = 0.02;
[x1Grid,x2Grid] = meshgrid(min(X(:,1)):d:max(X(:,1)),...
min(X(:,2)):d:max(X(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
scores = SVM_pred(xGrid, X, Y,kernel,alpha,beta0);
contour(x1Grid,x2Grid,reshape(scores,size(x1Grid)),[0 0],'k',...
'LineWidth', 4);

xlabel('$X_1$','FontSize', 18,...
'Interpreter','latex');
ylabel('$X_2$', 'FontSize', 18,...
'Interpreter','latex');
switch kernel
case 'linear'
```

```
title({'SVM',strcat('Kernel:',kernel,';C=',num2str(Cost))}, 'FontSize', 18,...
'Interpreter','latex');
case 'ploynomial'
title({'SVM',strcat('Kernel:',kernel,';C=',num2str(Cost),';n=',num2str(poly_con))}, 'FontSize', 18,...
'Interpreter','latex');
case 'RBF'
title({'SVM',strcat('Kernel:',kernel,';C=',num2str(Cost),';$\gamma$=',num2str(gamma))}, 'FontSize', 18,...
'Interpreter','latex');
case 'Sigmoid'
title({'SVM',strcat('Kernel:',kernel,';C=',num2str(Cost),';$\kappa$=',num2str(kappa1))}, 'FontSize', 18,...
'Interpreter','latex');
end
legend({'+1:setosa';'-1:versicolor'},'FontSize',16,'Location', 'Best');
hold off
return
```

# 第 5 章 结果与讨论

## 5.1. 结果

使用数据的 80%为训练集，20%为测试集，三个类别的分类结果如下图所示，左图为训练集数据，右图为训练数据加新数据。

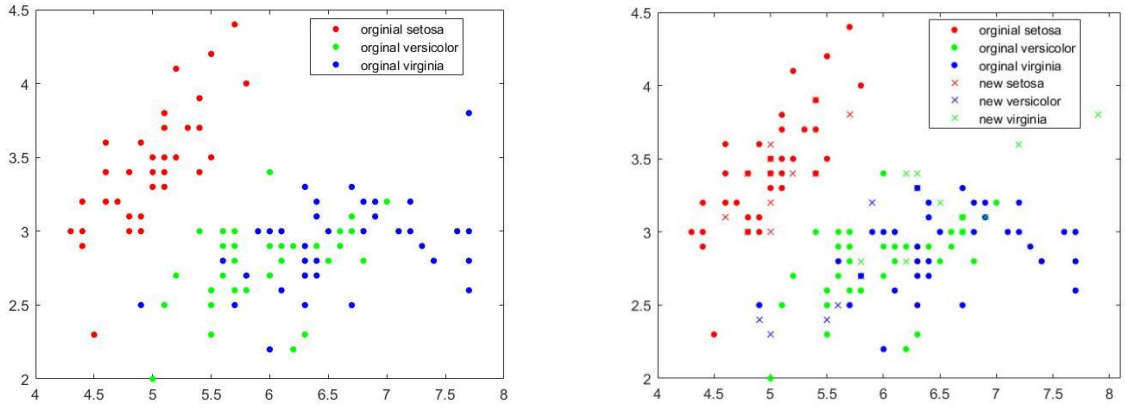


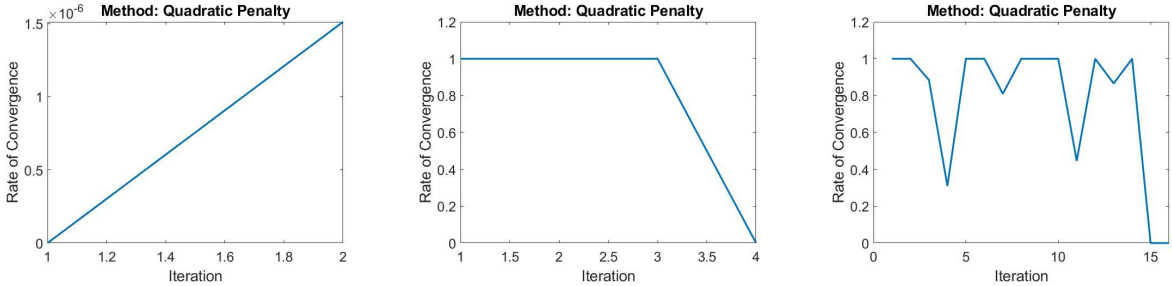

**图 3 分类结果**

## 5.2. 收敛速率

为量化的收敛速率，计算 $\frac{\| x_k - x^* \|_2}{\| x_{k-1} - x^* \|_2}$ 对迭代 k 的比值进行比较。

### 5.2.1. 二次惩罚法

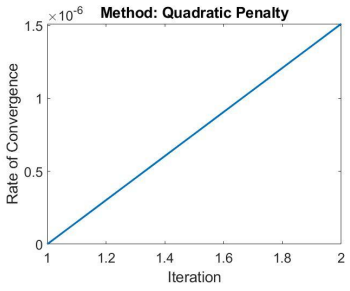
#### 5. 2. 1. 1. 线性核



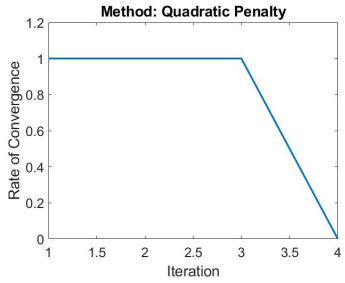

**SVM 分类器 1** **SVM 分类器 2** **SVM 分类器 3**

**图 4 不同 SVM 分类器收敛速率图**

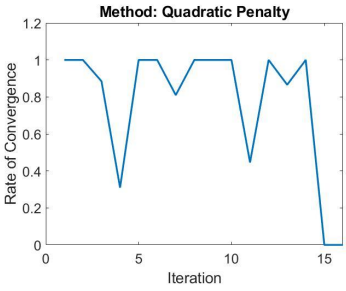
5.2.1.2. 多项式核




SVM 分类器 1



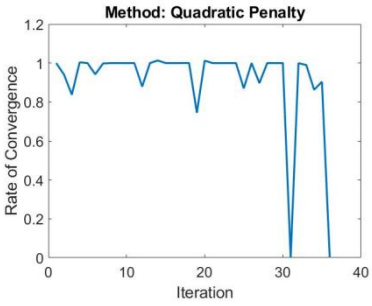

SVM 分类器 2



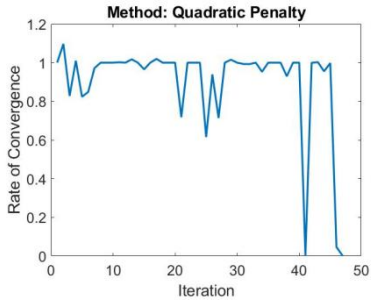

SVM 分类器 3

**图 5 不同 SVM 分类器收敛速率图**

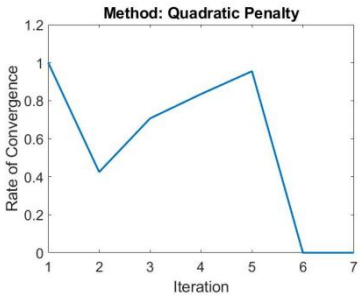
5.2.1.3. 高斯核




SVM 分类器 1



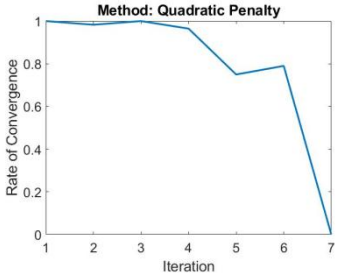

SVM 分类器 2



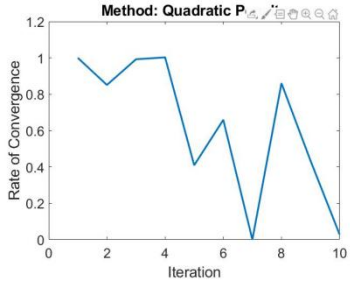

SVM 分类器 3

**图 6 不同 SVM 分类器收敛速率图**

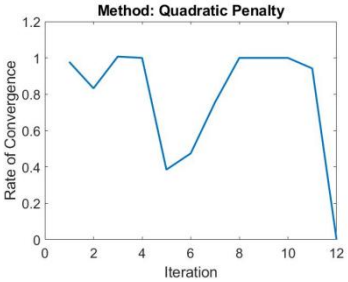
5.2.1.4. Sigmoid 核




SVM 分类器 1




SVM 分类器 2

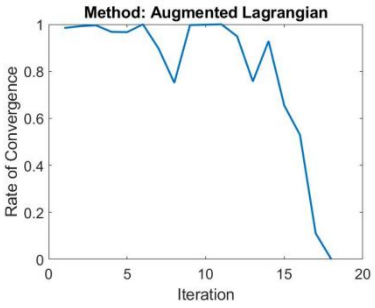



SVM 分类器 3

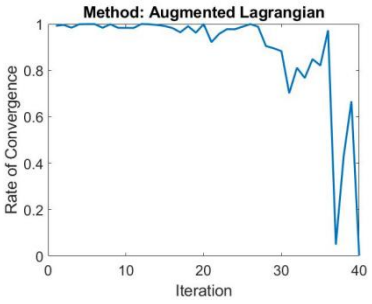
**图 7 不同 SVM 分类器收敛速率图**

### 5.2.2. 增广拉格朗日法

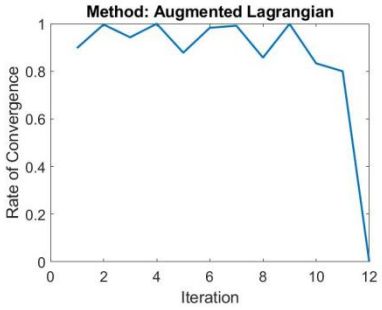
#### 5.2.2.1. 线性核




**SVM 分类器 1**



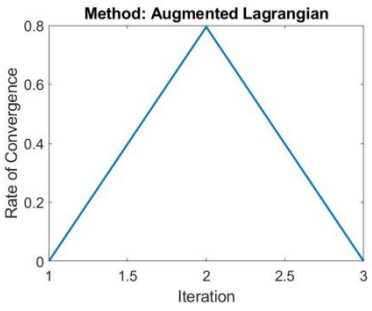

**SVM 分类器 2**



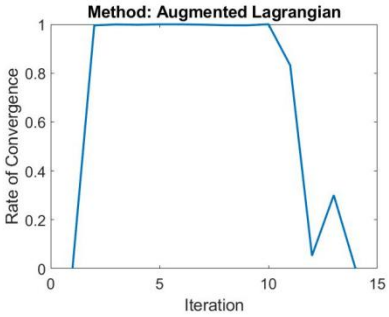

**SVM 分类器 3**

**图 8 不同 SVM 分类器收敛速率图**

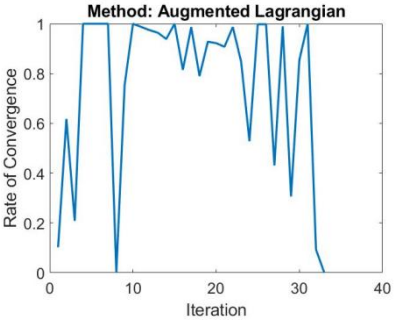
#### 5.2.2.2. 多项式核




**SVM 分类器 1**



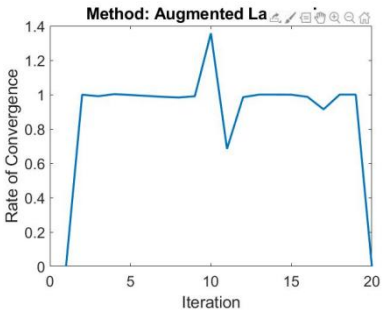

**SVM 分类器 2**



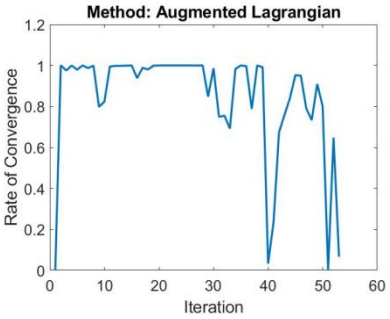

**SVM 分类器 3**

**图 9 不同 SVM 分类器收敛速率图**

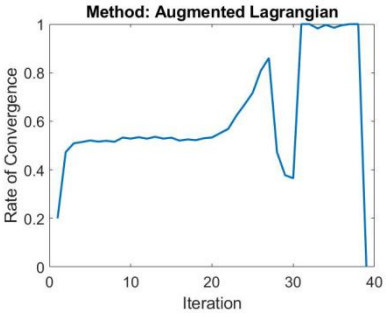
#### 5.2.2.3. 高斯核




**SVM 分类器 1**



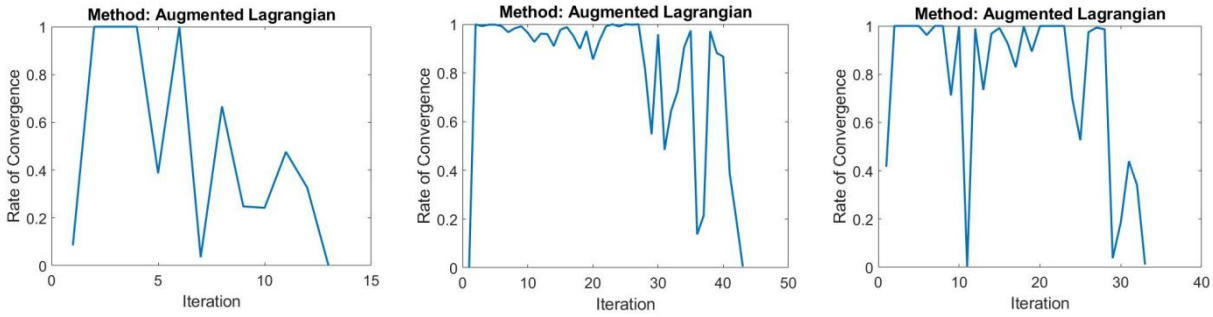

**SVM 分类器 2**




**SVM 分类器 3**

**图 10 不同 SVM 分类器收敛速率图**

#### 5.2.2.4. Sigmoid 核

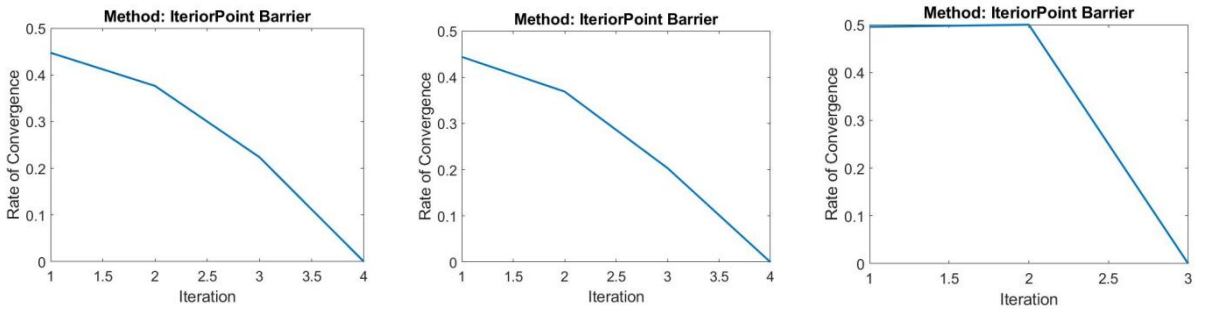



**SVM 分类器 1** **SVM 分类器 2** **SVM 分类器 3**

**图 11 不同 SVM 分类器收敛速率图**

### 5.2.3. InteriorPoint Barrier 法

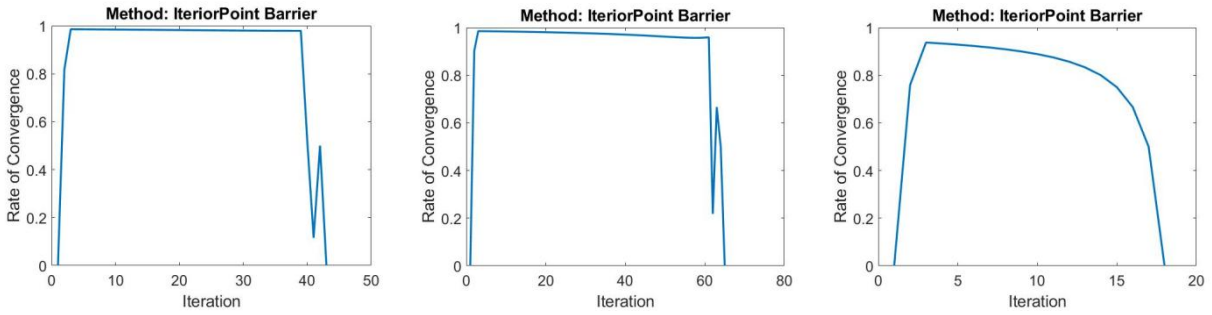
#### 5.2.3.1. 线性核




**SVM 分类器 1** **SVM 分类器 2** **SVM 分类器 3**

**图 12 不同 SVM 分类器收敛速率图**

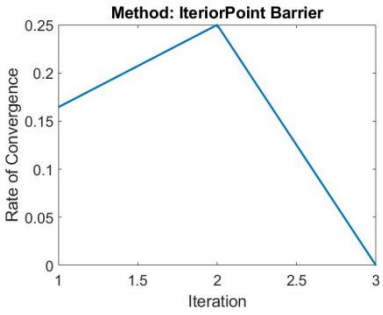
#### 5.2.3.2. 多项式核



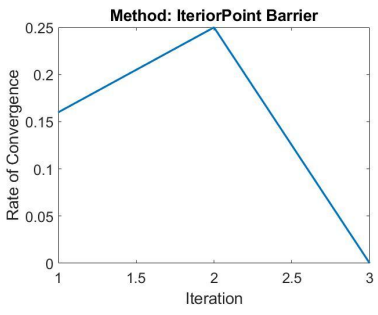

**SVM 分类器 1** **SVM 分类器 2** **SVM 分类器 3**

**图 13 不同 SVM 分类器收敛速率图**

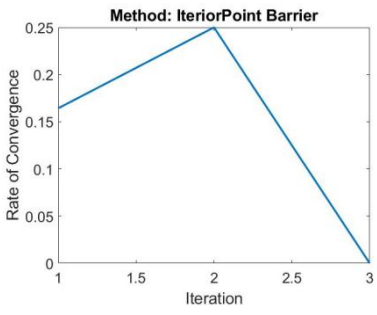
5.2.3.3. 高斯核




SVM 分类器 1



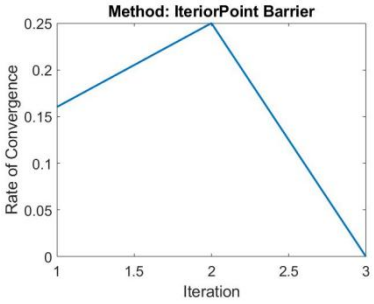

SVM 分类器 2



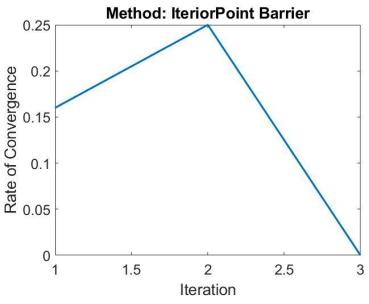

SVM 分类器 3

**图 14 不同 SVM 分类器收敛速率图**

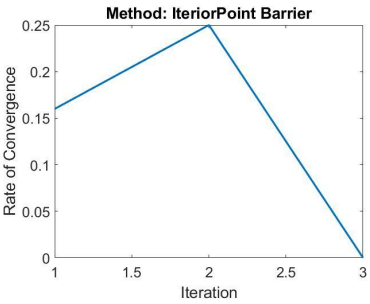
5.2.3.4. Sigmoid 核




SVM 分类器 1




SVM 分类器 2




SVM 分类器 3

**图 15 不同 SVM 分类器收敛速率图**

## 5.3. 精度比较

不同内核的不同优化方法获得的精度如下所示：quadratic programming 是 matlab 自带的，将其结果作为参考。

**表 1 同内核的不同优化方法求解精度**

| Kernel | Quadratic Penalty | Augmented Lagrangian | InteriorPoint Barrier | Quadratic programming |
|---|---|---|---|---|
| Linear | 73.333 | 73.333 | 73.333 | 93.3333 |
| Polynomial | 73.333 | 73.333 | 40 | 26.6667 |
| RBF | 86.6667 | 86.6667 | 76.6667 | 96.6667 |
| Sigmoid | 40 | 76.6667 | 26.6667 | 33.3333 |

根据上表的结果可知，最佳内核是 RBF，因为其精度一直很高。这也解释了为什么 RBF 内核是 SVM 中广泛使用且最常用的内核。其中，增广拉格朗日法在所有内核中都

具有最好的性能。

# 结 论

基于 matlab 设计一个支持向量机(SVM)用来解决分类问题。设计的支持向量机主要包括二次惩罚、增广拉格朗日和 InteriorPoint Barrier 三种不同的优化方法以及线性、多项式、高斯和 Sigmoid 四种不同的核，结果表明：

（1）所设计的 SVM 分类器能够很好的对数据进行分类。

（2）使用的“一对多”的方法能够很好的处理多类分类问题。

（3）通过分析了不同优化方法、不同核化函数下、不同 SVM 分类器的收敛速率并对比不同优化方法、不同核化函数下的求解精度可知，在该分类问题中，最佳内核是高斯核，因为其精度一直很高；其中，增广拉格朗日法在所有内核中都具有最好的性能。

# 参考文献

[1]Andrew Ng, CS229 Lecture notes , Support vector machines, Part V.

[2] Tristan Fletcher, Support Vector Machines Explained

[3]Numerical Optimisation Constraint optimisation: Penalty and augmented Lagrangian methods, Lecture 14 (based on Nocedal, Wright)

[4]Numerical Optimisation Constraint optimisation: Interior point methods, Lecture 14 (based on Boyd, Vander bergher)