中山大学计算机学院
人工智能
本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	202320346	专业（方向）	计算机科学与技术
学号	21312450	姓名	林隽哲

一、 实验题目

支持向量机（SVM）手动实现

支持向量机（SVM）分类任务

手动实现线性支持向量机（SVM）分类器，对Scikit-Learn提供的Iris数据集进行分类任务。本次作业要求不能使用现成的SVM库函数，而是通过手动实现线性SVM算法，完成训练和分类任务。

数据集

使用Scikit-Learn提供的Iris数据集。Iris数据集包含三类鸢尾花：Setosa、Versicolor和Virginica，每个数据样本包含萼片长度、萼片宽度、花瓣长度、花瓣宽度共4个特征。本次实现要求使用SVM进行二分类任务，可自行选择使用哪几个特征。其中Setosa与另外两类线性可分，故选择Setosa、Versicolor或Setosa、Virginica进行分类即可。

参考实验步骤

1. **数据预处理**
   - 从Iris数据集中选择两类鸢尾花的数据，并提取相关特征和标签。
   - 对数据进行标准化处理
   - 按照合理的比例划分训练集和测试集
2. **线性SVM的手动实现**
   - 编写代码实现线性SVM的训练算法，迭代更新直至损失函数收敛或达到最大迭代次数。
3. **模型评估与结果可视化**
   - 在测试集上评估手动实现的SVM分类器的表现（如准确率）。

二、 实验内容

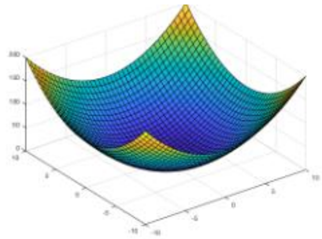
1. 算法原理

原始问题（primal problem）

原问题是一个凸二次规划问题（Quadratic Programming, QP），如下所示：

$$\min \quad \frac{1}{2}\|w\|^2$$
$$\text{subject to} \quad y_i[(wx_i+b)]-1\geq 0 \quad (i=1,2,\ldots,\mathrm{m})$$

凸二次规划特点：有解（全局最优解）



原问题可以通过应用**拉格朗日乘子法**构造**拉格朗日函数**（**Lagrange function**）再通过求解其**对偶问题**（**dual problem**）得到原始问题的最优解。

拉格朗日（Lagrange）函数

定义 Lagrange 对偶函数（简称对偶函数，dual function）如下：

$$g(\lambda,\nu):=\inf_{x\in\mathcal{D}} L(x,\lambda,\nu)$$
$$=\inf_{x\in\mathcal{D}}\left\{f_0(x)+\sum_{i=1}^{m}\lambda_i f_i(x)+\sum_{j=1}^{p}\nu_j h_j(x)\right\}.$$

性质一： $g(\lambda,\nu)$ 是关于 λ 和 ν 的凹函数！

性质二： 对 $\forall\lambda\geq 0$ 和 $\forall\nu$，可以推出 $g(\lambda,\nu)\leq p^*$

不难看出，函数 $g(\lambda,v)$ 实际上给出了**原问题最优值的下界**。我们令：

$$\mathcal{L}(w,\alpha,\beta)=f(w)+\sum_{i=1}^{k}\alpha_i g_i(w)+\sum_{i=1}^{l}\beta_i h_i(w).$$

原问题可以被转换为以下的等价问题：

$$\min_{w}\theta_{\mathcal{P}}(w)=\min_{w}\max_{\alpha,\beta:\alpha_i\geq 0}\mathcal{L}(w,\alpha,\beta)$$

Lagrange 对偶问题

回顾弱对偶定理：对于 Lagrange 对偶函数 $g(\lambda,\nu)$，对任意 $\forall\lambda \geq 0$ 和 $\forall\nu$，$g(\lambda,\nu)$ 给出了原优化问题最优值 p^* 的一个下界。

显然，该下界和乘子 λ 和 ν 的选取相关。而当我们极大化对偶函数 $g(\lambda,\nu)$，试图计算 p^* 的最紧下界时，就引出了原问题的 **Lagrange 对偶问题**：

Lagrange 对偶问题

$$\begin{aligned} \min_x \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \\ & h_j(x) = 0. \end{aligned} \qquad \text{(Primal)} \qquad\qquad \begin{aligned} \sup / \max \quad & g(\lambda,\nu) \\ \text{s.t.} \quad & \lambda \geq 0 \end{aligned} \qquad \text{(Dual)}$$

- 左侧即为原问题(Primal)，右侧即为 Lagrange 对偶问题(Dual)
- 原问题(Primal)的最优值为 $p^* \;\geq\;$ 对偶问题(Dual)的最优值为 d^*
- 对偶问题(Dual)的最优解 (λ^*,ν^*) 称为对偶最优解或最优 Lagrange 乘子
- 无论原问题(Primal)是否为凸，对偶问题(Dual) 总是一个**凹优化问题**

设原问题(Primal)的最优值为 $p^* = f_0(x^*)$，对偶问题(Dual)的最优值为 $d^* = g(\lambda^*,\nu^*)$。[1] 于是，

- 不等式 $d^* \leq p^*$ 总是成立的，称为弱对偶性 (weak duality)
- 等式 $d^* = p^*$ 不必然成立！当等式成立时，称为强对偶性 (strong duality)
- 差值 $p^* - d^*$ 称为对偶间隙 (duality gap)，根据弱对偶性可知对偶间隙总是非负的

对偶性下的最优性条件

显然，强对偶性是很好的性质。如果成立，则可以通过求解对偶问题来求解原问题的最优值。遗憾的是，一般情况下强对偶性并不成立。

但是，对于凸优化问题，在一定（不是特别强）的条件下，强对偶性是成立的，这也说明了凸优化问题的优势。

考虑一般优化问题（可能非凸）

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leqslant 0, \quad i=1,\cdots,m, \\ & h_j(x)=0, \quad j=1,\cdots,p. \end{aligned} \qquad \textbf{(Primal)}$$

假设：（考虑简单情形）

- 问题的定义域为 $\mathbb{R}^n$，即 $\left(\bigcap_{i=0}^{m} \operatorname{dom} f_i\right) \cap \left(\bigcap_{j=1}^{p} \operatorname{dom} h_j\right) = \mathbb{R}^n$；
- 函数 f_i，$i=0,1,\cdots,m$ 和 h_j，$j=1,2,\cdots,p$ 均可微；
- 问题的最优解 x^* 和 最优值 p^* 存在，且**强对偶成立**。

写出其对偶函数 $g(\lambda,\nu)=\inf\limits_x \left\{ f_0(x)+\sum_{i=1}^m \lambda_i f_i(x)+\sum_{j=1}^p \nu_j h_j(x)\right\}$
和对偶问题

$$\max_{\lambda,\nu} \quad g(\lambda,\nu), \quad \text{s.t.} \quad \lambda \geq 0. \qquad \textbf{(Dual)}$$

KKT 条件（最优解的必要条件）

对于可微且**对偶间隙为 0** 的优化问题，原对偶最优解 (x^*,λ^*,ν^*) 必须满足条件：

$$\begin{aligned} f_i(x^*) &\leq 0, \quad i=1,\cdots,m, && \text{(primal feasibility)} \\ h_j(x^*) &= 0, \quad j=1,\cdots,p, && \text{(primal feasibility)} \\ \lambda_i^* &\geq 0, \quad i=1,\cdots,m, && \text{(dual feasibility)} \\ \lambda_i^* f_i(x^*) &= 0, \quad i=1,\cdots,m, && \text{(complementary slacknes} \\ \nabla f_0(x^*)+\sum_{i=1}^m \lambda_i^* \nabla f_i(x^*)+\sum_{j=1}^p \nu_j^* \nabla h_j(x^*) &= 0, && \text{(stationarity)} \end{aligned}$$

以上条件合称为 **Karush-Kuhn-Tucker (KKT)** 条件。

定理 2 (KKT 条件是原对偶最优解的充要条件)

对于目标函数和约束函数均可微，且强对偶性成立的凸优化问题，有：

$$(\tilde{x},\tilde{\lambda},\tilde{\nu}) \text{ 为原对偶最优解} \iff (\tilde{x},\tilde{\lambda},\tilde{\nu}) \text{ 满足 KKT 条件.}$$

总结以上提到的算法原理，我们首先再次明确原问题与对偶问题：

$$\mathcal{L}(w,\alpha,\beta)=f(w)+\sum_{i=1}^k \alpha_i g_i(w)+\sum_{i=1}^l \beta_i h_i(w).$$

$$d^*=\max_{\alpha,\beta:\alpha_i\geq 0}\min_w \mathcal{L}(w,\alpha,\beta) \leq \min_w \max_{\alpha,\beta:\alpha_i\geq 0} \mathcal{L}(w,\alpha,\beta)=p^*.$$

对偶问题 Dual Problem **原问题 Primal Problem**

某些条件下：d*=p*，此时其解满足

Karush-Kuhn-Tucker (KKT) conditions, which are as follows:

原始可行性 $$\frac{\partial}{\partial w_i}\mathcal{L}(w^*,\alpha^*,\beta^*) = 0, \quad i=1,\ldots,n \tag{3}$$

约束条件 $$\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*,\alpha^*,\beta^*) = 0, \quad i=1,\ldots,l \tag{4}$$

互补条件 $$\alpha_i^* g_i(w^*) = 0, \quad i=1,\ldots,k \tag{5}$$

约束条件 $$g_i(w^*) \le 0, \quad i=1,\ldots,k \tag{6}$$

乘子非负 $$\alpha^* \ge 0, \quad i=1,\ldots,k \tag{7}$$

Moreover, if some w^*,α^*,β^* satisfy the KKT conditions, then it is also a solution to the primal and dual problems.

IN OUR CASE

分类任务对应的原始问题：

$$\min \quad \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i[(wx_i+b)]-1\ge 0 \quad (i=1,2,\ldots,\mathrm{m})$$

对应的拉格朗日函数：

When we construct the Lagrangian for our optimization problem we have:

$$\mathcal{L}(w,b,\alpha)=\frac{1}{2}||w||^2-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}(w^Tx^{(i)}+b)-1\right]. \tag{8}$$

带入 KKT 条件对式子进行化简：

$$\nabla_w\mathcal{L}(w,b,\alpha)=w-\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}=0 \qquad \frac{\partial}{\partial b}\mathcal{L}(w,b,\alpha)=\sum_{i=1}^{m}\alpha_i y^{(i)}=0.$$

代入 (8) 得到：

$$\begin{aligned}L(\mathbf{w},b,\boldsymbol{\alpha})&=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)}+b\right)-1\right]\\&=\frac{1}{2}\sum_{i=1}^{m}\alpha_i y^{(i)}(\mathbf{x}^{(i)})^T\sum_{j=1}^{m}\alpha_j y^{(j)}\mathbf{x}^{(j)}-\sum_{i=1}^{m}\alpha_i\alpha_i y^{(i)}(\mathbf{x}^{(i)})^T\sum_{j=1}^{m}\alpha_j y^{(j)}\mathbf{x}^{(j)}-b\sum_{i=1}^{m}\alpha_i y^{(i)}+\sum_{i=1}^{m}\alpha_i\\&=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\alpha_i y^{(i)}(\mathbf{x}^{(i)})^T\sum_{j=1}^{m}\alpha_j y^{(j)}\mathbf{x}^{(j)}=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_j\alpha_j(\mathbf{x}^{(i)})^T\mathbf{x}^{(j)}\end{aligned}$$

从而我们将原问题的求解转换为对偶问题的求解：

$$\begin{aligned}\max_\alpha \quad & W(\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)},x^{(j)}\rangle.\\ \text{s.t.} \quad & \alpha_i\ge 0,\quad i=1,\ldots,m\\ & \sum_{i=1}^{m}\alpha_i y^{(i)}=0,\end{aligned}$$

求解对偶问题：

我们可以通过 **SMO 算法、AML 方法**等等求解出能使 W(α)达到最大的 α 值，然后在求解出取得最有解时对应的 w 和 b：

w取得最优解时:

$$w = \sum_{i=1}^{n} a_i y_i \boldsymbol{x_i} = a_1 y_1 \boldsymbol{x_1} + a_2 y_2 \boldsymbol{x_2} + \ldots + a_m y_m \boldsymbol{x_m}$$

至少存在一个j，使得$y_j[(w\boldsymbol{x_j} + b)] - 1 = 0$，于是可以求得最优$b$

$$b = \frac{1}{y_j} - w\boldsymbol{x_j} = y_j - w\boldsymbol{x_j} = y_j - \sum_{i=1}^{n} a_i y_i \boldsymbol{x_i}\boldsymbol{x_j}$$

至此，所以我们就求得了整个线性可分SVM的解。求得的分离超平面为:

$$\sum_{i=1}^{n} \hat{\alpha}_i y_i X^T X_i + \hat{b} = 0$$

分类的决策函数就是 $f(X) = sign(\sum_{i=1} \hat{\alpha}_i y_i X^T X_i + \hat{b})$

软间隔问题求解

并非所有的样本点都有一个松弛变量与其对应, 只有“离群点”才有，或者也可以这么看，所有没离群的点松弛变量都等于0

松弛变量的值实际上标示出了对应的点到底离群有多远，值越大，点就越远

惩罚因子C决定了你有多重视离群点带来的损失

- 当所有离群点的松弛变量的和一定时，C越大，对目标函数的损失也越大，非常不愿意放弃这些离群点，
- C定为无限大，这样只要稍有一个点离群，目标函数的值马上变成无限大，马上让问题变成无解，这就退化成了硬间隔问题

惩罚因子C不是一个变量，整个优化问题在解的时候，C是一个你必须事先指定的值

尽管加了松弛变量这么一说，但这个优化问题仍然是一个**二次规划**问题，解它的过程比起原始的硬间隔问题来说，没有任何更加特殊的地方

$$\min_{W,b,\xi} \quad \frac{1}{2}||W||^2 + C\sum_{i=1}^{n}\xi_i \tag{3.1.4}$$
$$s.t.\ y_i(X_i^TW+b) \geq 1-\xi_i$$
$$\xi_i \geq 0, i=1,2,\ldots n.$$

上式所述问题即软间隔支持向量机。

式 $(3.1.4)$ 表示的软间隔支持向量机依然是一个凸二次规划问题，和硬间隔支持向量机类似，我们可以通过拉格朗日乘子法将其转换为对偶问题进行求解。 式 $(3.1.4)$ 对应的拉格朗日函数为

$$L(W,b,\xi,\alpha,\beta) = \frac{1}{2}||W||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(X_i^TW+b)-1+\xi_i] - \sum_{i=1}^{n}\beta_i\xi_i \tag{3.2.1}$$

类似2.4节，为了求得对偶问题的解，我们需要先求得 $L(W,b,\xi,\alpha,\beta)$ 对 W、b 和 ξ 的极小再求对 α 和 β 的极大。

对参数部分 W b ζ 求解

(1) 求 $\min\limits_{W,b,\xi} L(W,b,\xi,\alpha,\beta)$: 将 $L(W,b,\xi,\alpha,\beta)$ 分别对 W、b 和 ξ 求偏导并令为0可得

$$W = \sum_{i=1}^{n}\alpha_i y_i X_i \tag{3.2.2}$$
$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{3.2.3}$$
$$C = \alpha_i + \beta_i \tag{3.2.4}$$

将上面三个式子代入式 $(3.2.1)$ 并进行类似式 $(2.4.8)$ 的推导即得

$$\min_{W,b,\xi} L(W,b,\xi,\alpha,\beta) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j X_i^T X_j + \sum_{i=1}^{n}\alpha_i \tag{3.2.5}$$

注意其中的 β 被消去了。

对 α 求解

(2) 求 $\min\limits_{W,b,\xi} L(W,b,\xi,\alpha,\beta)$ 对 α 的极大:

式 $(3.2.5)$ 对 α 求极大，也等价于式 $(3.2.5)$ 取负数后对 α 求极小，即

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j X_i^T X_j - \sum_{i=1}^{n}\alpha_i \tag{3.2.6}$$

同时满足约束条件:

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{3.2.7}$$
$$0 \leq \alpha_i \leq C, i=1,2,\ldots,n.$$

至此，我们得到了原始最优化问题 $(3.1.4)$ 和对偶最优化问题 $(3.2.6)$、$(3.2.7)$。

求解 W 与 b

类似2.4节地，假设我们现在通过通用的二次规划求解方法或者SMO算法求得了(3.2.6)、(3.2.7)的最优解 $\hat{\alpha}$，则根据式(3.2.2)可求得最优 $\hat{W}$：

$$\hat{W}=\sum_{i=1}^{n}\hat{\alpha}_i y_i X_i \tag{3.2.8}$$

再根据KKT条件，即

$$\begin{cases}\text{乘子非负}: \alpha_i\ge 0,\ \beta_i\ge 0 (i=1,2,\ldots n.\text{下同})\\ \text{约束条件}: y_i(X_i^TW+b)-1\ge -\xi_i\\ \text{互补条件}: \alpha_i[y_i(X_i^TW+b)-1+\xi_i]=0,\ \beta_i\xi_i=0\end{cases}$$

可求得整个软间隔SVM的解，即:

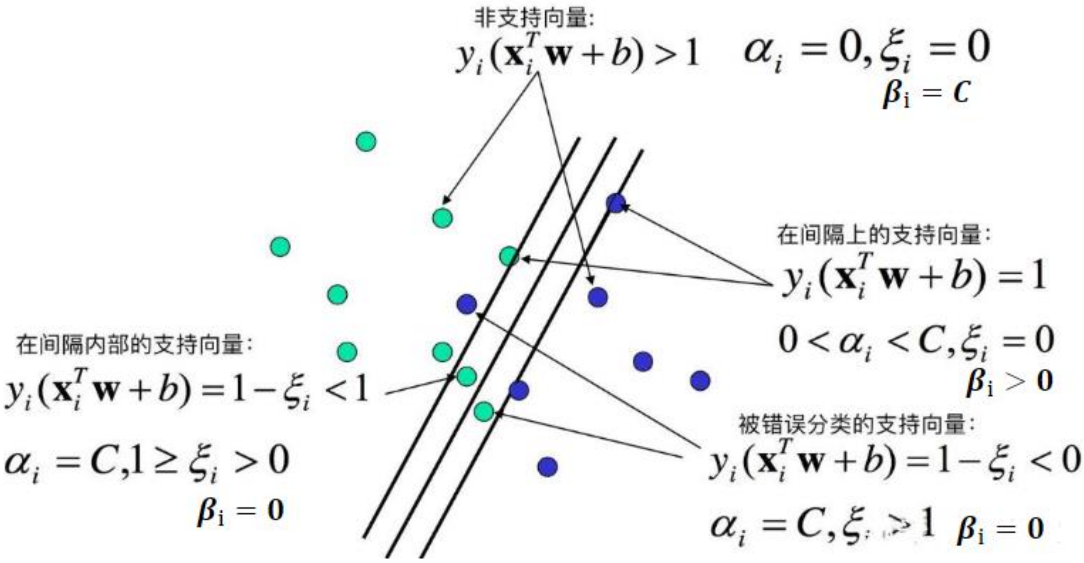
$$\hat{W}=\sum_{i\in SV}\hat{\alpha}_i y_i X_i \tag{3.2.9}$$

$$\hat{b}=y_j-\sum_{i\in SV}\hat{\alpha}_i y_i X_j^T X_i \tag{3.2.10}$$

其中 j 需满足 $0<\hat{\alpha}_j<C$。

对于任意样本 (X_i, y_i)，若 $\alpha_i=0$，此样本点不是支持向量，该样本对模型没有任何的作用；若 $\alpha_i>0$，此样本是一个支持向量。

若满足 $\alpha_i>0$，进一步地，若 $0<\alpha_i<C$，



SMO 算法（Sequential minimal optimization）

3.1 两个变量 α_1，α_2 的优化问题

为推导简单起见，不妨选择两个变量 α_1，α_2，固定其他变量 $\alpha_i(i=3,4,5,\dots N)$，于是最优化问题可以表示为：

$$\min_{\alpha_1,\alpha_2} W(\alpha_1,\alpha_2)=\frac{1}{2}\alpha_1^2K_{11}+\frac{1}{2}\alpha_2^2K_{22}+\alpha_1\alpha_2y_1y_2K_{12}-(\alpha_1+\alpha_2)+\alpha_1y_1\sum_{i=3}^{N}\alpha_iy_iK_{i1}+\alpha_2y_2\sum_{i=3}^{N}\alpha_iy_iK_{i2}$$

$$s.t.\ \alpha_1y_1+\alpha_2y_2=-\sum_{i=3}^{N}\alpha_iy_i=\varsigma$$
$$0\le\alpha_i\le C,\ i=1,2$$

其中 $K_{i,j}$ 是核函数，$K_{i,j}=K(x_i,x_j),i,j=1,2,3,\dots N,\varsigma$ 是常数，上式中省略了不含 α_1,α_2 的常数项；

我们可以看到原始的优化问题变成了关于两个变量 α_1，α_2 的最值问题，并且这两个变量还满足 $\alpha_1y_1+\alpha_2y_2=-\sum_{i=3}^{N}\alpha_iy_i=\varsigma$，所以可以通过变量代换变成一个变量的最值问题。

3.2 α_2^{new} 的取值范围

上面我们说到此问题可以转化为一个变量的最值问题，我们就用 α_1 表示 α_2，上面问题转化为关于 α_2 的单变量极值问题。

我们首先根据约束不等式 $0\le\alpha_2\le C$ 和 $\alpha_1y_1+\alpha_2y_2=\varsigma$ 求解 α_2 的新区间，这里用 $\alpha_1^{old},\alpha_2^{old}$ 表示最初的变量 α_1,α_2 ,用 α_2^{new} 表示同时便满足等式约束和不等式约束的最新的 α_2

当 $y_1\ne y_2$ 时：

$\alpha_1^{old}y_1+\alpha_2^{old}y_2=\varsigma$，两边同时乘 y_1 ,等式变为：$\alpha_1^{old}-\alpha_2^{old}=k$，所以 $\alpha_2^{old}=\alpha_1^{old}-k$；

又 $0\le\alpha_1^{old}\le C$,所以 $-k\le\alpha_2^{old}\le C-k$；

又 $\alpha_1^{old}-\alpha_2^{old}=k$，所以 $-\alpha_1^{old}+\alpha_2^{old}\le\alpha_2^{old}\le C-\alpha_1^{old}+\alpha_2^{old}$；

因为 $0\le\alpha_2^{old}\le C$，$\alpha_2^{old}-\alpha_1^{old}\le\alpha_2^{old}\le C+\alpha_2^{old}-\alpha_1^{old}$，要同时成立，所以最终 α_2^{new} 的范围是以上两不等式的交集：

$$L=max(0,\alpha_2^{old}-\alpha_1^{old})$$
$$H=min(C,C+\alpha_2^{old}-\alpha_1^{old})$$
$$L\le\alpha_2^{new}\le H$$

当 $y_1 = y_2$ 时，步骤同上，可得：

$$L = max(0, \alpha_2^{old} + \alpha_1^{old} - C)$$
$$H = min(C, \alpha_2^{old} + \alpha_1^{old})$$
$$L \le \alpha_2^{new} \le H$$

3.3 未剪辑的解

上面我们得到变量 α_2^{new} 的取值范围，原始目标函数 $W(\alpha_1, \alpha_2)$ 对 求偏导并令偏导等于0得到未剪辑(unclip)的解 $\alpha_2^{new,unc}$：

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

> 此处未剪辑(unclip)的解指的是二次函数的对称轴，未剪辑的解指的是可能为最优值；剪辑就是 $\alpha_2^{new,unc}$ 和 α_2^{new} 的取值范围联立，判断解是否在区间内，进而得到剪辑后的解 α_2^{new} 。

其中： $g(x) = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b$, 该函数描述输入变量 $\boldsymbol{x}$ 的预测值；

$E_i = g(x_i) - y_i$,描述预测值和实际值之差；

$\eta = K_{11} + K_{22} - 2K_{12}$;

3.4

根据未剪辑的解和 α_2^{new} 的范围确定最终最优解的取值（类似于高中学习的开口向上的二次函数定轴动区间问题）：

$$\alpha_2^{new} = \begin{cases} H & \text{if} \quad \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc} & \text{if } H \le \alpha_2^{new,unc} \le L \\ L & \text{if} \quad \alpha_2^{new,unc} < L \end{cases}$$

根据等式约束 $\sum_{i=1}^{N} \alpha_i y_i = 0$ 可以求得相对应的 α_1^{new} ：

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_1^{new})$$

四.第一个变量 α_1 的选择：外循环

上面介绍关于两个变量的理论推到和表达式。SMO算法中要求挑选两个变量进优化，如何进行挑选？对于第一个变量：检验在每个样本点 $(\boldsymbol{x}_i, y_i)$ 是否满足KKT条件的要求，选取违反KKT条件最严重的点：

$$
\begin{aligned}
\alpha_i = 0 \;&\Leftrightarrow\; y_i g(\boldsymbol{x}_i) \geq 1 \qquad (1) \\
0 < \alpha_i < C \;&\Leftrightarrow\; y_i g(\boldsymbol{x}_i) = 1 \qquad (2) \\
\alpha_i = C \;&\Leftrightarrow\; y_i g(\boldsymbol{x}_i) < 1 \qquad (3)
\end{aligned}
$$

- 条件（1）对应决策边界之外的点；
- 条件（2）对应的点即位支持向量；
- 条件（3）对应决策边界之内的点；

具体过程是建立**外层循环，**从**支持向量开始**，看支持向量是否满足KKT条件。如果这些点都满足KKT条件，再遍历整个数据集，从整个数据集中选取违反KKT条件最严重的点。

> 为什么选取违反KKT条件最严重的点?
> 首先是选取的要求是违背KKT条件，我们的**目的是使得所有拉格朗日乘子 α_i 及对应的样本点 $(\boldsymbol{x}_i, y_i)$ 满足KKT条件（最优解的充分必要条件）**，那么就需要对不满足KKT的样本点 $(\boldsymbol{x}_i, y_i)$ 对应的拉格朗日乘子 α_i 进行修正；其次选取的要求是违背条件最严重，违背条件最严重会**使得选择点能对算法收敛贡献最大，目标函数改变的最大。**

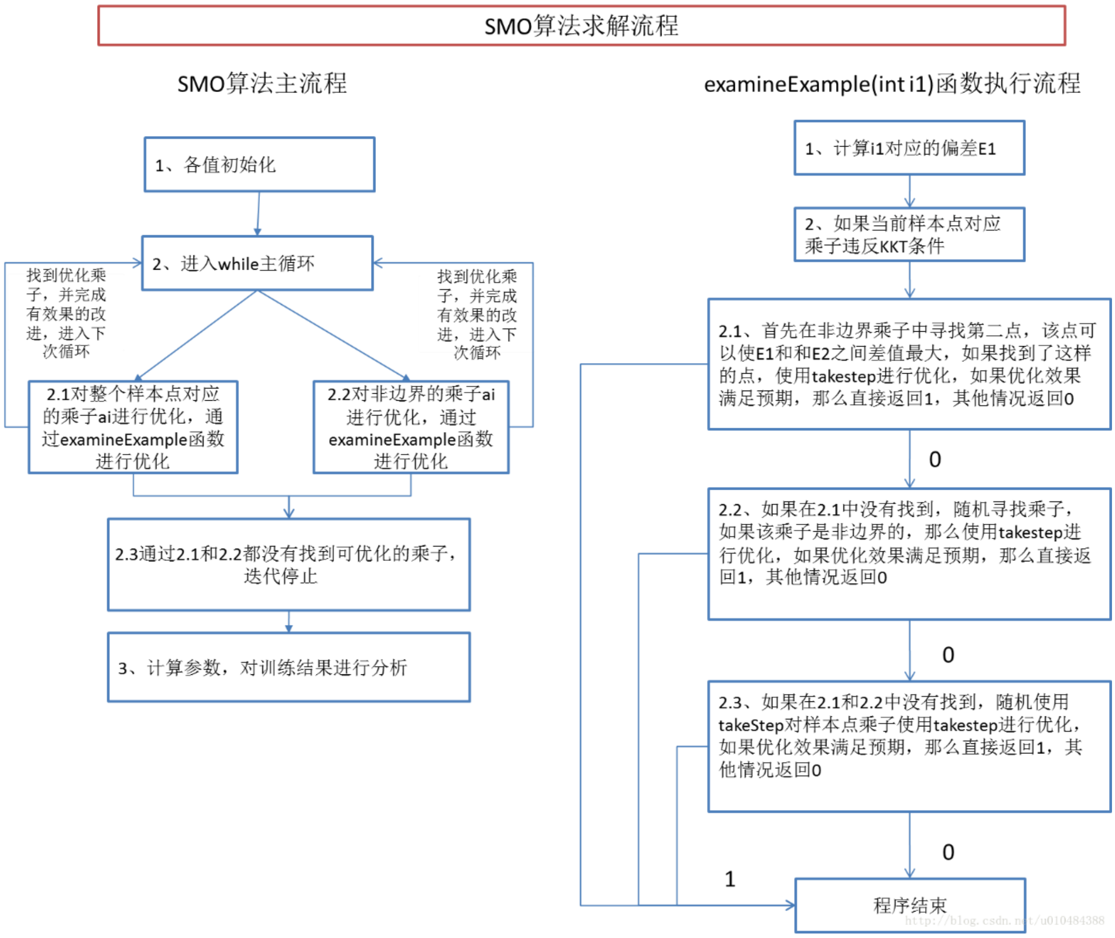
五.第二个变量 α_2 的选择：内循环

第二个变量的选择的具体过程是建立**内层循环。选取的标准是 α_2 使得有足够的变化。**而

$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2}{\eta}(E_2 - E_1)$，所以就要要求 $|E_2 - E_1|$ 最大化，即如果 $E_1 > 0$，选取最小的 E_i 作为 E_2 ;反正选择最大的E_i 作为 E_2。

如果通过$|E_2 - E_1|$ 最大化的方法无法使目标函数有足够的变化，那就遍历间隔边界上的支持向量点，依次将其对应的乘子作为 α_2 ,直到目标函数有足够的下降。如果间隔边界上的支持向量仍未使得目标函数有足够的下降，那么便利整个数据集，如果数据集还未使得目标函数有足够的下降，则更换 α_1，从新找到新的 α_1 进行优化。

2. 伪代码



注：在实际进行代码的编写时，我简化了变量选择的过程，具体可见以下的代码。

3. 关键代码展示

SVM 模型（使用 SMO 算法进行求解）

```python
class SVM:
    def __init__(self, C=1.0, max_iter=1000, tol=1e-3):
        """
        :param C: 惩罚系数
        :param max_iter: 最大迭代次数
        :param tol: 容忍度
        """
        self.C = C
        self.max_iter = max_iter
        self.tol = tol

        """
        self.alphas: 拉格朗日乘子
        self.w: 权重向量
        self.b: 偏置项
```

```
        """
        self.alphas = None
        self.w = None
        self.b = None

    def fit(self, X, y): # SMO
        n_samples, n_features = X.shape
        self.alphas = np.zeros(n_samples)
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.max_iter):
            alpha_prev = np.copy(self.alphas)

            for j in range(n_samples):
                i = np.random.randint(0, n_samples)
                while i == j:
                    i = np.random.randint(0, n_samples)

                xi, xj, yi, yj = X[i], X[j], y[i], y[j]
                # kii, kjj, kij = np.dot(xi, xi), np.dot(xj, xj),
np.dot(xi, xj)
                kii, kjj, kij = self._kernel(xi, xi), self._kernel(xj,
xj), self._kernel(xi, xj)
                eta = 2 * kij - kii - kjj

                Ei = self._decision_function(xi) - yi
                Ej = self._decision_function(xj) - yj

                if eta == 0:
                    continue

                alpha_j_old = self.alphas[j]

                # 计算 L 和 H (L <= alpha_j_new <= H)
                if yi != yj:
                    L = max(0, self.alphas[j] - self.alphas[i])
                    H = min(self.C, self.C + self.alphas[j] -
self.alphas[i])
                else:
                    L = max(0, self.alphas[j] + self.alphas[i] - self.C)
                    H = min(self.C, self.alphas[j] + self.alphas[i])

                if L == H:
```

```
                    continue

                # 计算新的 alpha 值
                alpha_j_new = np.clip(alpha_j_old - yj * (Ei - Ej) /
eta, L, H)
                alpha_i_new = self.alphas[i] + yi * yj * (alpha_j_old -
alpha_j_new)

                self.alphas[j] = alpha_j_new
                self.alphas[i] = alpha_i_new

            # 检查 alpha 是否有足够的变化
            diff = np.linalg.norm(self.alphas - alpha_prev)
            if diff < self.tol:
                break

        # 计算权重向量 w: w = Σ (alpha_i * y_i * x_i)
        self.w = np.sum(self.alphas[:, None] * y[:, None] * X, axis=0)

        # 计算偏置项 b: b = 1/n_samples * Σ (y_i - w^T x_i)
        self.b = np.mean([yi - np.dot(self.w, xi) for xi, yi in zip(X,
y)])

    def _decision_function(self, x):
        # f(x) = w^T x + b
        return np.dot(x, self.w) + self.b

    def _kernel(self, x1, x2):
        return np.dot(x1, x2)

    def predict(self, X):
        return np.sign(self._decision_function(X))
```

当然，你也可以不直接通过 KKT 条件，而改用常规的梯度下降来求解该问题。你同样可以通过支持向量的条件实现最大化分类间隔的迭代，如下：

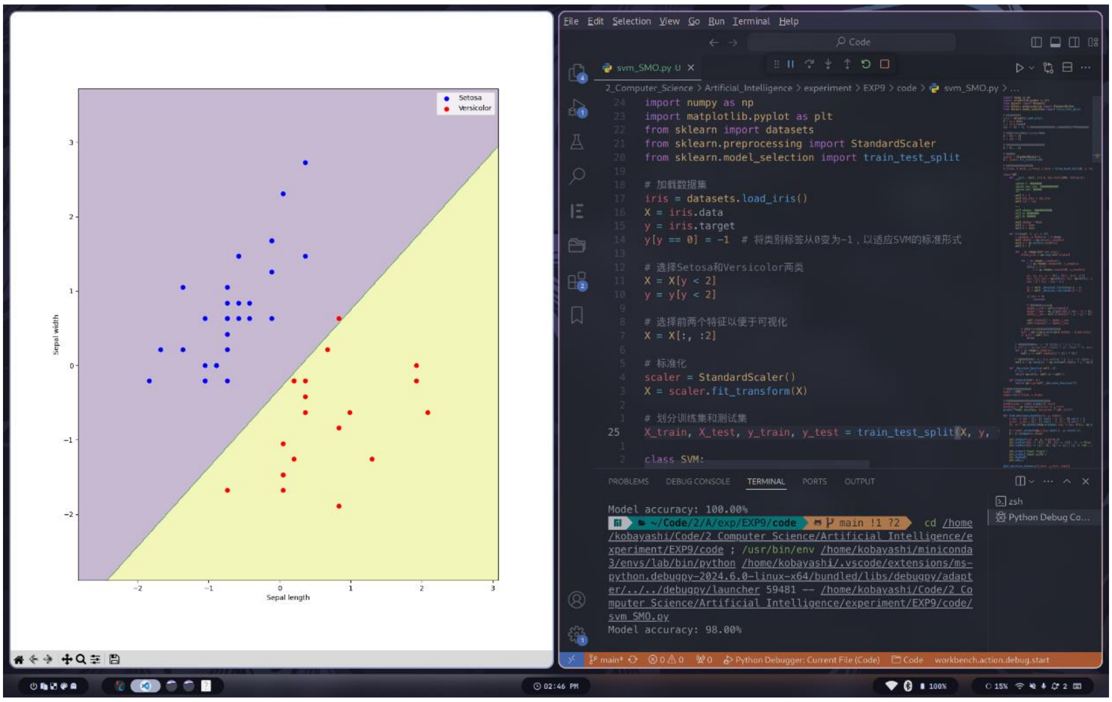
SVM 模型（使用梯度下降）

```
class LinearSVM:
    def __init__(self, learning_rate=0.01, lambda_param=0.01,
n_iters=1000):
        """
        :param learning_rate: 学习率
        :param lambda_param: 正则化参数
        :param n_iters: 迭代次数
        """
```

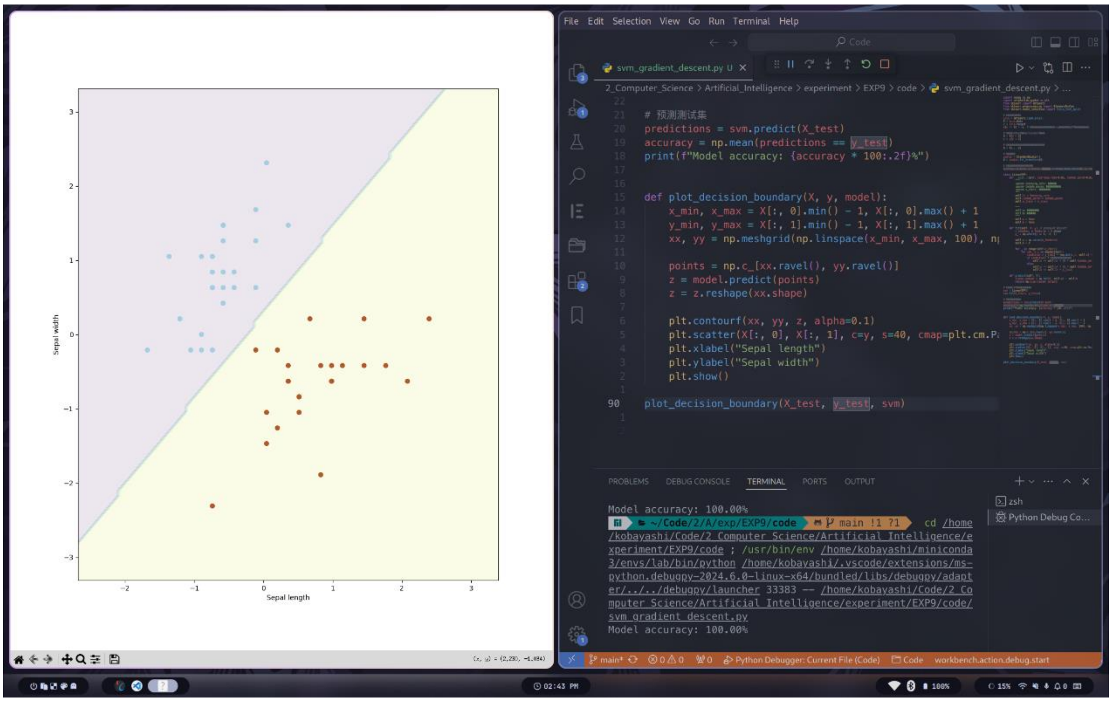
```
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters

        """
        self.w: 权重向量
        self.b: 偏置项
        """
        self.w = None
        self.b = None

    def fit(self, X, y): # gradient descent
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)

        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >=
1
                if condition: # 正确分类的情况
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w
- np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]

    def predict(self, X):
        linear_output = np.dot(X, self.w) - self.b
        return np.sign(linear_output)
```

三、 实验结果及分析

SVM（SMO）



SVM（Gradient Descent）



四、 参考资料

- 理论课课件