专业课程实验报告

课程名称： 模式识别

开课学期： 2022 至 2023 学年 第 1 学期

专　　业： 智能科学与技术

年级班级： 20 级 3 班

学生姓名： 严中圣

学生学号： 222020335220177

实验教师： 杨颂华

人工智能学院

《模式识别》实验课报告书

实验编号：	1	**实验名称:**	基于 SVM 的数字识别
姓　　名：	严中圣	**学　　号：**	222020335220177
日　　期：	2022 年 11 月 13 日	**教师打分：**	

1 实验目标

支持向量机（support vector machines, SVM ）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。通过本实验目的掌握 SVM 的原理和实现算法，并将其应用到手写数字识别。

2 实验环境

- PyCharm 2022.1.3 (Professional Edition)
- python 3.7.13, numpy 1.21.5, torchvision 0.11.3
- OS: Windows 11 22H2
- CPU:12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz

3 实验原理

支持向量机学习方法包含构建由简至繁的模型：线性可分支持向量机、线性支持向量机以及非线性支持向量机。简单模型是复杂模型的基础，也是复杂模型的特殊情况。当训练数据线性可分时，通过硬间隔最大化，学习一个线性的分类器，即线性可分支持向量机，又称为硬间隔支持向量机；当训练数据近似线性可分时，通过软间隔最大化,也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向量机；当训练数据线性不可分时，通过使用核技巧及软间隔最大化，学习非线性支持向量机。

首先给定一个特征空间上的训练数据集 $T = (x_l, y_l), (x_2, y_2), ..., (x_N, y_N)$，其中，$x_i \in \mathcal{R}^n, y_i \in \{+1, -1\}, i = 1, 2, ..., N$ 。x_i 为第 i 个特征向量,y_i 为 x_i 的类标记。当 $y_i = +1$ 时，称 x_i 为正例；当 $y_i = -1$ 时，称 x_i 为负例。(x_i, y_i) 称为样本点。

3.1 线性可分支持向量机

假设训练数据集是线性可分的，当训练数据集线性可分时，存在无穷个分离超平面可将两类数据正确分开。线性可分支持向量机利用间隔最大化求最优分离超平面，此时的解是唯一的。线性可分支持向量机定义如下：

给定线性可分训练数据集，通过间隔最大化或等价地求解相应的凸二次规划问题学习得到的分离超平面为

$$\omega^* x + b^* = 0 \tag{1}$$

以及相应的分类决策函数

$$f(x) = sign(\omega^* x + b^*) \tag{2}$$

称为线性可分支持向量机。

对于给定的训练数据集 T 和超平面 (ω, b)，定义超平面 (ω, b) 关于样本点 (x_i, yi) 的几何间隔为

$$\gamma_i = y_i(\frac{w}{\|\omega\|}x_i + \frac{b}{\|\omega\|}) \tag{3}$$

超平面关于所有样本点的几何间隔的最小值为

$$\gamma = \min_{i=1,2,...,N} \gamma_i \tag{4}$$

这个距离就是我们所谓的支持向量到超平面的距离。根据以上定义，SVM 模型的求解最大分割超平面问题可以表示为以下约束最优化问题：

$$\begin{aligned} &\max_{\omega,b} \gamma \\ &s.t. \quad y_i(\frac{w}{\|\omega\|}x_i + \frac{b}{\|\omega\|}) \geq \gamma, i = 1, 2, ..., N \end{aligned} \tag{5}$$

即我们希望最大化超平面 $\omega, b)$ 关于训练数据集的几何间隔 γ，约束条件表示的是超平面 $\omega, b)$ 关于每个训练样本点的几何间隔至少是 γ。将约束条件两边同时除以 γ，得到：

$$y_i(\frac{w}{\|\omega\|\gamma}x_i + \frac{b}{\|\omega\|\gamma}) \geq 1 \tag{6}$$

而式 (6) 中 $\|\omega\|, \gamma$ 均为标量，故令

$$\omega = \frac{w}{\|\omega\|\gamma}, \quad b = \frac{b}{\|\omega\|\gamma} \tag{7}$$

得到：

$$y_i(\omega x + b) \geq 1, i = 1, 2, ..., N \tag{8}$$

而式 (5) 中最大化 γ，即最小化 $\|\omega\|^2$，因此 SVM 模型的求解最大分割超平面问题又可转化为以下约束最优化问题。

$$\begin{aligned} &\min_{\omega,b} \|\omega\|^2 \\ &s.t. \quad y_i(\omega x + b) \geq 1, i = 1, 2, ..., N \end{aligned} \tag{9}$$

这显然是一个凸二次规划问题。为了式 (9)，将它作为原始最优化，应用拉格朗日对偶性，通过求解对偶问题得到原始问题的最优解，这就是线性可分支持向量机的对偶算法。这样做的优点，一是对偶问题往往更容易求解; 二是自然引入核函数，进而推广到非线性分类问题。

首先构建拉格朗日函数。为此，对每一个不等式约束引进拉格朗日乘子 $\alpha_i \geq 0, i = 1, 2, ..., N$，定义拉格朗日函数：

$$\mathcal{L}(\omega, b, \alpha) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{N} \alpha_i y_i(\omega x_i + b) + \sum_{i=1}^{N} \alpha_i \tag{10}$$

其中，极值点是目标函数和约束条件相切的点，此时梯度是同方向的。$\nabla\|\omega\| = \alpha\nabla(\sum_{i=1}^{N} y_i(\omega x_i + b))$，此时 α 必须满足 $\alpha \geq 0$，从而保持梯度同方向。根据拉格朗日对偶性，原始问题的对偶问题是极大极小问题:

$$\max_{\alpha} \min_{\omega,b} \mathcal{L}(\omega, b, \alpha) \tag{11}$$

所以，为了得到对偶问题的解，需要先求 $\mathcal{L}(\omega, b, \alpha)$ 对 ω, b 的极小，再求对 α 的极大。

将式 (10) 中分别对 ω, b 求偏导并令其等于 0：

$$\begin{aligned} \nabla_w \mathcal{L}(w, b, \alpha) &= w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \\ \nabla_b \mathcal{L}(w, b, \alpha) &= -\sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{12}$$

可得：

$$\begin{aligned} w &= \sum_{i=1}^{N} \alpha_i y_i x_i \\ \sum_{i=1}^{N} \alpha_i y_i &= 0 \end{aligned} \tag{13}$$

将式 (13) 代入式 (10)，得:

$$\begin{aligned} \mathcal{L}(w, b, \alpha) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) - \sum_{i=1}^{N} \alpha_i y_i \left(\left(\sum_{j=1}^{N} \alpha_j y_j x_j \right) \cdot x_i + b \right) + \sum_{i=1}^{N} \alpha_i \\ &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) + \sum_{i=1}^{N} \alpha_i \end{aligned} \tag{14}$$

即

$$\min_{w,b} L(w, b, \alpha) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) + \sum_{i=1}^{N} \alpha_i \tag{15}$$

此时问题转化为：

$$\begin{aligned} &\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) + \sum_{i=1}^{N} \alpha_i \\ &\text{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \cdots, N \end{aligned} \tag{16}$$

将上式中的目标函数由求解极大值转化为求解极小值，就得到了下面与之等价的对偶最优化问题：

$$\begin{aligned} &\max_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right) - \sum_{i=1}^{N} \alpha_i \\ &\text{s.t.} \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \cdots, N \end{aligned} \tag{17}$$

若满足 KKT 条件，则问题即可转化为式 (17) 的对偶问题求解。KKT 条件要求满足：

$$\begin{cases} \alpha_i \geq 0 \\ y_i \left(\boldsymbol{w}_i \cdot \boldsymbol{x}_i + b\right) - 1 \geq 0 \\ \alpha_i \left(y_i \left(\boldsymbol{w_i} \cdot \boldsymbol{x_i} + b\right) - 1\right) = 0 \end{cases} \tag{18}$$

根据先前的推导，满足 KKT 条件，故原始问题可以转换为求解对偶问题式 (17)。

利用序列最小优化 (SMO)(见 4.2) 我们可以求出最优解 α^*，再根据 α^* 我们就可以求解出 ω 和 b，进而求得我们最初的目的：找到超平面，即决策平面。

根据式 (13)(18) 可知，在 α^* 中，至少存在一个 $\alpha_j^* > 0$（利用反证法易证），对此 j 有

$$y_j(\omega^* x + b^*) - 1 = 0 \tag{19}$$

由此可得

$$\begin{aligned} \boldsymbol{w}^* &= \sum_{i=1}^{N} \alpha_i^* y_i \boldsymbol{x}_i \\ b^* &= y_j - \sum_{i=1}^{N} \alpha_i^* y_i \left(\boldsymbol{x_i} \cdot \boldsymbol{x_j}\right) \end{aligned} \tag{20}$$

故最终的分类决策函数为：

$$f(x) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i^* y_i \left(x \cdot x_i\right) + b^*\right) \tag{21}$$

3.2 线性支持向量机

对于线性可分问题，上述线性可分支持向量机的学习 (硬间隔最大化) 算法是完美的。但是，训练数据集线性可分是理想的情形。在现实问题中，训练数据集往往是线性不可分的，即在样本中出现噪声或特异点。此时，有更一般的学习算法。

线性不可分意味着某些样本点 (x_i, y_i) 不能满足函数间隔大于等于 1 的约束条件式 (9)。为了解决这个问题，可以对每个样本点 (x_i, y_i) 引进一个松弛变量 $\xi_i \geq 0$，使函数间隔加上松弛变量大于等于 1。这样，约束条件变为:

$$y_j(\omega x + b) \geq 1 - \xi_i \tag{22}$$

此时最优化问题变为：

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i \left(w \cdot x_i + b\right) \geq 1 - \xi_i, \quad i = 1, 2, \cdots, N \\ & \xi_i \geq 0, \quad i = 1, 2, \cdots, N \end{aligned} \tag{23}$$

其中 $\xi_i = \max\left(0, 1 - y_i \left(\boldsymbol{w} \cdot \boldsymbol{x_i} + b\right)\right)$，$C > 0$ 称为惩罚参数，C 值越大，对分类的惩罚越大。与先前线性可分思路一致，利用拉格朗日乘子法得到拉格朗日函数，再求其对偶问题。对偶问题具体形式如下：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x_i} \cdot \boldsymbol{x_j}\right) - \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, N \end{aligned} \tag{24}$$

求解最终决策函数的步骤与先前一致。

3.3 非线性支持向量机

非线性分类问题是指通过利用非线性模型才能很好地进行分类的问题。对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维特征空间中的线性分类问题，在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里，目标函数和分类决策函数都只涉及实例和实例之间的内积，所以不需要显式地指定非线性变换，而是用核函数替换当中的内积。核函数表示，通过一个非线性转换后的两个实例间的内积。具体地，$K(x, z)$ 是一个函数或正定核，意味着存在一个从输入空间到特征空间的映射 $\phi(x)$，对任意输入空间中的 x, z，有

$$K(x, z) = \phi(x)\phi(z) \tag{25}$$

在线性支持向量机学习的对偶问题中，用核函数 $K(x,z)$ 替代内积，求解得到的就是非线性支持向量机

$$f(x)=\operatorname{sign}\left(\sum_{i=1}^{N}\alpha_i^* y_i K\left(x,x_i\right)+b^*\right) \tag{26}$$

常用的核函数有多项式和函数，高斯核函数，字符串核函数等，这里我们采用高斯径向基函数

$$K(x,z)=\exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) \tag{27}$$

此时分类决策函数成为

$$f(x)=\operatorname{sign}\left(\sum_{i=1}^{N_s}a_i^* y_i \exp\left(-\frac{\|x-x_i\|^2}{2\sigma^2}\right)+b^*\right) \tag{28}$$

3.4 序列最小最优化算法

SMO 算法是一种启发式算法，其基本思路是: 如果所有变量的解都满足此最优化问题的 KKT 条件，那么这个最优化问题的解就得到了。因为 KKT 条件是该最优化问题的充分必要条件。否则，选择两个变量，固定其他变量，针对这两个变量构建一个二次规划问题。这个二次规划问题关于这两个变量的解应该更接近原始二次规划问题的解，因为这会使得原始二次规划问题的目标函数值变得更小。重要的是，这时子问题可以通过解析方法求解，这样就可以大大提高整个算法的计算速度。子问题有两个变量，一个是违反 KKT 条件最严重的那一个，另一个由约束条件自动确定。如此，SMO 算法将原问题不断分解为子问题并对子问题求解，进而达到求解原问题的目的。

算法流程如下

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i\in\mathcal{X}=\mathbf{R}^n$，$y_i\in\mathcal{Y}=\{-1,+1\}$，$i=1,2,\cdots,N$，精度 ε；

输出：近似解 $\hat{\alpha}$。

(1) 取初值 $\alpha^{(0)}=0$，令 $k=0$；

(2) 选取优化变量 $\alpha_1^{(k)},\alpha_2^{(k)}$，解析求解两个变量的最优化问题 (7.101)~(7.103)，求得最优解 $\alpha_1^{(k+1)},\alpha_2^{(k+1)}$，更新 α 为 $\alpha^{(k+1)}$；

(3) 若在精度 ε 范围内满足停机条件

$$\sum_{i=1}^{N}\alpha_i y_i=0,\quad 0\leqslant\alpha_i\leqslant C,\quad i=1,2,\cdots,N$$

$$y_i\cdot g(x_i)\begin{cases}\geqslant 1, & \{x_i|\alpha_i=0\}\\ =1, & \{x_i|0<\alpha_i<C\}\\ \leqslant 1, & \{x_i|\alpha_i=C\}\end{cases}$$

其中，

$$g(x_i)=\sum_{j=1}^{N}\alpha_j y_j K(x_j,x_i)+b$$

则转 (4)；否则令 $k=k+1$，转 (2)；

(4) 取 $\hat{\alpha}=\alpha^{(k+1)}$。

4 实验步骤

4.1 数据集来源

本次实验采用来自 MNIST 数据集 (Mixed National Institute of Standards and Technology database)，美国国家标准与技术研究院收集整理的大型手写数字数据库，包含 60,000 个示例的训练集以及 10,000 个

示例的测试集

4.2 数据加载

利用 torchvision 中所集成的 MNIST 数据集加载数据，此时返回的数据为 PIL Image 格式，在后续处理中，直接利用 $numpy.array()$ 将其转化为数组处理即可。

```
import torchvision.datasets as dataset  # 公开数据集的下载和管理

def mnist_dataset(download=False):
    """
    dataset preparation
    :param download: whether to download to lacally or not
    :return: MNIST dataset (PIL Image)
    """
    train_dataset = dataset.MNIST(root="mnist", train=True, download=
    download)
    test_dataset = dataset.MNIST(root="mnist", train=False, download=
    download)
    return train_dataset, test_dataset
```

4.3 SVM 算法训练

在训练过程中，将数据分 batch 依次加载到模型中，如此可提高算法的训练速度和训练精度。SVM.py 实现了一个 SVM 多分类器，具体实现原理是：对于样本中的每两个类别之间都训练一个 SVM 二分类器。对于 k 个类别，共可训练出 $k(k-1)/2$ 个 SVM 二分类器。在预测时，将测试样例分别输入到 $k(k-1)/2$ 分类器中。假设 (i,j) 表示划分类别 i 和类别 j 的 SVM 分类器，对于每个分类器 (i,j)：

- 若分类结果为 $+1$，则 $count[i]+=1$
- 若分类结果为 -1，则 $count[j]+=1$

最后分类结果取相应类别计数最大的那个类别作为最终分类结果。SV.py 还实现了将训练的模型保存成文件，方便预测时直接从文件读取，省去了再次训练的时间。训练求解过程中采用 SMO 进行优化，在选择优化变量时，选择误差步长最大的两个变量进行优化，如此可以大幅提高优化速度。同时采用了 RBF 核函数，具体实现参见 $kernelTrans(self,x,z)$。详细代码见附录或`https://github.com/ZS-Yan/SVM-digit-recognition`.

最终在 MNIST 数据集上得到了 92.48% 的准确率。

5 实验分析

SVM 作为经典的分类器模型，在手写数字识别任务依然取得了不错的准确率。且相较于现代神经网络方法，能够保持参数量低的优势同时取得较高准确率。理解 SVM 无论是对数学基础或是编程能力均是一次极大的考验，在此向提出这些经典且精妙的大师们致敬！

6 参考文献

[1] 周志华. 机器学习 [M]. 清华大学出版社, 2016.

[2] 李航. 统计学习方法 [M]. 清华大学出版社, 2019

[3] https://zhuanlan.zhihu.com/p/31886934

[4] Christopher M. Bishop: Pattern Recognition and Machine Learning, Chapter 4.3.4

7 附录 – SVM 实验完整代码

SVM.py

```
import sys
import numpy as np
from torch.utils.data import DataLoader
from tqdm import tqdm
from SMO import SMO
import pickle

from dataloader import mnist_dataset


class SVM:
    def __init__(self, train_dataset, test_dataset, C=0, toler=0, maxIter
   =0, batch_size=16, **kernelargs):
        self.classNum = len(train_dataset.classes)
        self.labels = [i for i in range(self.classNum)]
        self.classifierNum = (self.classNum * (self.classNum - 1)) / 2
        self.classifiers = []
        self.dataSet = [[[] for _ in range(self.classNum)] for k in range
   (len(train_dataset) // batch_size)]
        self.C = C
        self.toler = toler
        self.maxIter = maxIter
        self.kernelargs = kernelargs
        self.train_dataset = train_dataset
        self.test_dataset = test_dataset

        for batch in range(len(self.dataSet)):
            count = 0
            for i in range(count, count + batch_size):
                image, label = self.train_dataset[i]
                image = np.array(image).flatten()
                self.dataSet[batch][label].append(image)
            count += batch_size
        print(len(self.dataSet))

    def train(self):
        # train n * (n-1) classifiers
        for batch in range(len(self.dataSet)):
            print("batch {}:".format(batch))
```

```
            for i in range(self.classNum):
                for j in range(i + 1, self.classNum):
                    # print("train classifier {} and {}.".format(i, j))
                    data = []
                    label = [1.0] * np.shape(self.dataSet[batch][self.
labels[i]])[0]
                    label.extend([-1.0] * np.shape(self.dataSet[batch][
self.labels[j]])[0])
                    data.extend(self.dataSet[batch][self.labels[i]])
                    data.extend(self.dataSet[batch][self.labels[j]])
                    svm = SMO(np.array(data), np.array(label), self.C,
self.toler, self.maxIter, **self.kernelargs)
                    svm.smoP()
                    self.classifiers.append(svm)

    def test(self, test_dataset):
        """
        test phase
        :param test_dataset:
        :return:
        """
        predict_labels = []
        count = 0.0
        for n in tqdm(range(len(test_dataset))):
            image, label = test_dataset[0]
            result = [0] * self.classNum
            index = -1
            for i in range(self.classNum):
                for j in range(i + 1, self.classNum):
                    index += 1
                    classifier = self.classifiers[index]
                    # t = classifier.predict([data[n]])[0]
                    t = classifier.predict([np.array(image)])[0]
                    if t > 0.0:
                        result[i] += 1
                    else:
                        result[j] += 1
            predict_labels.append(result.index(max(result)))
            if predict_labels[-1] != label:
                count += 1
        return count / len(test_dataset)

    def predict(self, predict_dataset):
        """
        predict phase
        :param images: a set of images without labels
        :return: a list of results
```

```
        """
        predict_labels = []
        for n in range(len(predict_dataset)):
            result = [0] * self.classNum
            index = -1
            for i in range(self.classNum):
                for j in range(i + 1, self.classNum):
                    index += 1
                    classifier = self.classifiers[index]
                    t = classifier.predict(np.array([predict_dataset[n]])
)[0]
                    if t > 0.0:
                        result[i] += 1
                    else:
                        result[j] += 1
            predict_labels.append(result.index(max(result)))
        return predict_labels

    def save(self, filename):
        fw = open(filename, 'wb')
        pickle.dump(self, fw, 2)
        fw.close()

    @staticmethod
    def load(filename):
        fr = open(filename, 'rb')
        svm = pickle.load(fr)
        fr.close()
        return svm


def main():
    # '''
    # data, label = loadImage('trainingDigits')
    train_data, test_data = mnist_dataset(download=True)
    # print("dataset is loaded!")
    # svm = SVM(train_data, 200, 0.0001, 10000, name='rbf', theta=20)
    # svm.train()
    # svm.save("svm.txt")
    # '''
    svm = SVM.load("svm.txt")
    # test,testlabel = loadImage('testDigits')
    result = svm.test(test_data)
    print(result)


if __name__ == "__main__":
```

```
    sys.exit(main())
```

SMO.py

```
# from SVM import *
from os import listdir
import numpy as np


class SMO:
    def __init__(self, data, labels, C, toler, maxIter, **kernelargs):
        self.x = data
        self.label = np.array(labels).transpose()
        self.C = C
        self.toler = toler
        self.maxIter = maxIter
        self.m = np.shape(data)[0]
        # self.n = np.shape(data)[1]
        self.alpha = np.array(np.zeros(self.m), dtype='float64')
        self.b = 0.0
        self.eCache = np.array(np.zeros((self.m, 2)))
        self.K = np.zeros((self.m, self.m), dtype='float64')
        self.kwargs = kernelargs
        self.SV = ()
        self.SVIndex = None
        for i in range(self.m):
            for j in range(self.m):
                self.K[i, j] = self.kernelTrans(self.x[i, :], self.x[j,
    :])

    def calcEK(self, k):
        fxk = np.dot(self.alpha * self.label, self.K[:, k]) + self.b
        Ek = fxk - float(self.label[k])
        return Ek

    def updateEK(self, k):
        Ek = self.calcEK(k)
        self.eCache[k] = [1, Ek]

    def selectJ(self, i, Ei):
        maxE = 0.0
        selectJ = 0
        Ej = 0.0
        validECacheList = np.nonzero(self.eCache[:, 0])[0]
        if len(validECacheList) > 1:
            for k in validECacheList:
                if k == i: continue
                Ek = self.calcEK(k)
                deltaE = abs(Ei - Ek)
                if deltaE > maxE:
```

```
                    selectJ = k
                    maxE = deltaE
                    Ej = Ek
            return selectJ, Ej
        else:
            selectJ = selectJrand(i, self.m)
            Ej = self.calcEK(selectJ)
            return selectJ, Ej

    def innerL(self, i):
        Ei = self.calcEK(i)
        if (self.label[i] * Ei < -self.toler and self.alpha[i] < self.C)
    or \
                (self.label[i] * Ei > self.toler and self.alpha[i] > 0):
            self.updateEK(i)
            j, Ej = self.selectJ(i, Ei)
            alphaIOld = self.alpha[i].copy()
            alphaJOld = self.alpha[j].copy()
            if self.label[i] != self.label[j]:
                L = max(0, self.alpha[j] - self.alpha[i])
                H = min(self.C, self.C + self.alpha[j] - self.alpha[i])
            else:
                L = max(0, self.alpha[j] + self.alpha[i] - self.C)
                H = min(self.C, self.alpha[i] + self.alpha[j])
            if L == H:
                return 0
            eta = 2 * self.K[i, j] - self.K[i, i] - self.K[j, j]
            if eta >= 0:
                return 0
            self.alpha[j] -= self.label[j] * (Ei - Ej) / eta
            self.alpha[j] = clipAlpha(self.alpha[j], H, L)
            self.updateEK(j)
            if abs(alphaJOld - self.alpha[j]) < 0.00001:
                return 0
            self.alpha[i] += self.label[i] * self.label[j] * (alphaJOld -
     self.alpha[j])
            self.updateEK(i)
            b1 = self.b - Ei - self.label[i] * self.K[i, i] * (self.alpha
    [i] - alphaIOld) - \
                 self.label[j] * self.K[i, j] * (self.alpha[j] -
    alphaJOld)
            b2 = self.b - Ej - self.label[i] * self.K[i, j] * (self.alpha
    [i] - alphaIOld) - \
                 self.label[j] * self.K[j, j] * (self.alpha[j] -
    alphaJOld)
            if 0 < self.alpha[i] and self.alpha[i] < self.C:
                self.b = b1
```

```
            elif 0 < self.alpha[j] and self.alpha[j] < self.C:
                self.b = b2
            else:
                self.b = (b1 + b2) / 2.0
            return 1
        else:
            return 0

    def smoP(self):
        iter = 0
        entrySet = True
        alphaPairChanged = 0
        while iter < self.maxIter and ((alphaPairChanged > 0) or (
entrySet)):
            alphaPairChanged = 0
            if entrySet:
                for i in range(self.m):
                    alphaPairChanged += self.innerL(i)
                iter += 1
            else:
                nonBounds = np.nonzero((self.alpha > 0) * (self.alpha <
self.C))[0]
                for i in nonBounds:
                    alphaPairChanged += self.innerL(i)
                iter += 1
            if entrySet:
                entrySet = False
            elif alphaPairChanged == 0:
                entrySet = True
        self.SVIndex = np.nonzero(self.alpha)[0]
        self.SV = self.x[self.SVIndex]
        self.SVAlpha = self.alpha[self.SVIndex]
        self.SVLabel = self.label[self.SVIndex]
        self.x = None
        self.K = None
        self.label = None
        self.alpha = None
        self.eCache = None

    def kernelTrans(self, x, z):
        if np.array(x).ndim != 1 or np.array(z).ndim != 1:
            raise Exception("input vector is not 1 dim")
        if self.kwargs['name'] == 'linear':
            return sum(x * z)
        elif self.kwargs['name'] == 'rbf':
            theta = self.kwargs['theta']
            return np.exp(sum((x - z) * (x - z)) / (-1 * theta ** 2))
```

```
    def calcw(self):
        for i in range(self.m):
            self.w += np.dot(self.alpha[i] * self.label[i], self.x[i, :])

    def predict(self, image):
        """
        :param predict_dataset: an image array
        :return: predicted label
        """
        # test = np.array(testData)
        # return (test * self.w + self.b).getA()
        result = []
        m = np.shape(image)[0]
        for i in range(m):
            tmp = self.b
            for j in range(len(self.SVIndex)):
                tmp += self.SVAlpha[j] * self.SVLabel[j] * self.
   kernelTrans(self.SV[j], image[i, :])
            while tmp == 0:
                tmp = np.random.uniform(-1, 1)
            if tmp > 0:
                tmp = 1
            else:
                tmp = -1
            result.append(tmp)
        return result

def selectJrand(i, m):
    j = i
    while j == i:
        j = int(np.random.uniform(0, m))
    return j


def clipAlpha(a_j, H, L):
    if a_j > H:
        a_j = H
    if L > a_j:
        a_j = L
    return a_j
```