西安交通大學
XI'AN JIAOTONG UNIVERSITY

机器学习

实验报告

2023-2024 学年　第二学期

学　　院：　数学与统计学院

班　　级：　数学强基 2102

学　　号：　2213210010

姓　　名：　周冠程

指导教师：　孟德宇老师

实验地点：　数学实验中心

1 实验内容

1.1 问题描述

考虑一个包含多个类别的数据集，你的任务是使用 SVM 模型实现多分类，采用 One-vs-One 或 One-vs-All 策略。

1.2 实验步骤

1. **数据准备：**给定特征矩阵 (X) 和多类别标签向量 (Y)，共包含 (m) 个样本和 (n) 个特征。
2. **实现多类别 SVM 模型：**根据选择的策略（One-vs-One 或 One-vs-All），编写代码实现多类别 SVM 模型，可以使用现有库函数。
3. **数据预处理：**对数据进行标准化处理，并划分训练集和测试集。
4. **模型训练：**使用训练集对 SVM 模型进行训练，优化模型参数。
5. **模型评估：**使用测试集对训练好的 SVM 模型进行评估，计算分类准确率、混淆矩阵等指标

1.3 实验要求

对数据进行标准化处理，划分训练集和测试集；使用正确的核函数和超参数设置来优化模型；尽量不调用 Matlab 中的 fitcsvm 和 fitcecoc 函数实现 SVM 算法；计算并展示模型在测试集上的分类准确率和混淆矩阵。

1.4 提示

可以使用 MNIST 数据集。

2 实验描述

2.1 算法介绍

支持向量机（Support Vector Machine，SVM）是一种用于分类和回归分析的监督学习模型，最初由 Vladimir Vapnik 等人于 1990 年提出。

SVM 的主要思想是寻找一个最优的超平面，将不同类别的数据分开，同时使得与超平面最近的数据点（支持向量）到超平面的距离最大化。

SVM 的基本原理有：

1. **线性可分情况：** 对于线性可分的情况，SVM 试图找到一个超平面，使得正类别和负类别的样本能够被该超平面完美地分开。这个超平面被定义为使得两个类别的支持向量到该超平面的距离最大化的平面。

2. **线性不可分情况：** 对于线性不可分的情况，SVM 引入了核函数（Kernel Function），将数据从原始空间映射到一个更高维度的特征空间，使得在特征空间中的数据线性可分。这样，就可以在新的特征空间中找到一个超平面来进行分类。

SVM 在高维空间中的表现优秀，适用于处理高维数据。通过支持向量，可以降低模型的复杂度，减少过拟合的风险；在较小的数据集上表现良好，尤其适用于样本维度比样本数量大的情况。但是 SVM 对大规模数据和高维数据的处理效率较低，需要选择合适的核函数和调整超参数，同时对噪声和异常值较为敏感，这可能需要大量的实验和调优。

总的来说，SVM 是一个强大的分类器，在许多领域都有广泛的应用，如文本分类、图像识别、生物信息学等。

2.2 理论推导

2.2.1 原始问题推导

SVM 的基本思想是在高维数据空间计算一个超平面，以达到二分类的目的。传统的线性可分支撑向量机分类器 (Linear Support Vector Classifier, Linear SVC) 要求数据是线性可分的，通过优化算法得到一个最优的超平面，使该超平面距离两类数据的距离都相对较远。具体而言，定义数据集：$D = \{(x_i, y_i)\}_{i=1}^{n}, x_i \in D_{data} = \mathbb{R}^m, y_i \in \{-1, 1\}$，分割超平面定义为 $w^T x + b = 0$，基于数据是线性可分的假设，该超平面需要满足可行性约束：

$$y_i(w^T x_i + b) \geq 0, \ \forall i \tag{1}$$

该约束具有尺度不变性，需要确定新的约束以唯一化超平面，在此添加的约束为：

$$\min_i w^T x_i + b = 1 \tag{2}$$

该约束可以合并至原有的可行性约束 (式1) 中，该约束将改进为唯一化可行性约束：

$$y_i(w^T x_i + b) \geq 1, \ \forall i \tag{3}$$

接下来需要约束最优性，为了提高模型的泛化能力，希望超平面距离两类数据都尽可能地远，同时两类数据间隔也尽可能大。根据几何的知识，(x_i, y_i) 到超平面 $w^T x + b = 0$ 的距离为

$$\frac{y_i(w^T x_i + b)}{\| w \|} \tag{4}$$

而最大化间隔距离可以表示为：

$$\max_w \min_{y_i y_j = -1} \frac{|w^T(x_i - x_j)|}{\| w \|} = \max_w \frac{2}{\| w \|} = \max_w \frac{1}{2} \| w \| \tag{5}$$

为了方便计算，我们将上式改写为：

$$\min_w \frac{1}{2} w^T w \tag{6}$$

最终得到传统硬间隔约束下的 Linear SVC 优化问题表达形式：

$$\begin{aligned} &\min_w \frac{1}{2} w^T w \\ &s.t. \ y_i(w^T x_i + b) \geq 1 \end{aligned} \tag{7}$$

由于噪声及分布本身性质的影响，数据空间中采样得到的结果完全线性可分是不切实际的，这导致极少数异常值的出现会极大的影响算法得到的结果。基于此动机，可以开发出软间隔的 SVM，以达到在线性不可分的观测数据下同样可以得到一个折中的结果。为此，我们引入松弛变量对于原有约束进行松弛：

$$\begin{aligned} &y_i(w^T x_i + b) \geq 1 - \epsilon_i \\ &\epsilon_i \geq 0 \end{aligned} \tag{8}$$

理想情况下，$\epsilon_i = 0$，为了达到这个目的，需要将松弛变量也进行优化，最终改进为：

$$\min_{w,\epsilon} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \epsilon_i \tag{9}$$

综上，软间隔的 SVM 优化问题的表形式改进为：

$$\begin{aligned} \min_{w,\epsilon} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \epsilon_i \\ s.t.\ y_i(w^T x_i + b) \geq 1 - \epsilon_i \end{aligned} \tag{10}$$

其中 C 为平衡超参数，需要手动调节。

2.2.2 对偶问题推导

无论是式7还是式10，优化问题的规模（优化目标的维数）均与数据空间维数正相关，同时计算量也较大，对于复杂数据难以优化，我们可以将问题转化为它的对偶问题。在此之前，我们需要引入命题：

[命题] 2.2.1 对于不等式优化问题

$$\begin{aligned} \min_{x} f(x) \\ s.t.\ g(x) \leq 0 \\ h(x) = 0 \end{aligned} \tag{11}$$

定义 *Lagrange* 函数：

$$L(x, \alpha, \beta) = f(x) + \sum \alpha_i h_i(x) + \sum \beta_i g_i(x) \tag{12}$$

满足

$$\min_{x} f(x) = \min_{x} \max_{\alpha;\beta \geq 0} L(x, \alpha, \beta) \tag{13}$$

关于其对偶问题有如下关系：

$$\max_{\alpha;\beta \geq 0} \min_{x} L(x, \alpha, \beta) \geq \min_{x} \max_{\alpha;\beta \geq 0} L(x, \alpha, \beta) \tag{14}$$

等号当且仅当 *KKT* 条件成立。

[定义] 2.2.1 *(KKT 定理)* 对于一个约束问题：

$$\begin{aligned} \min_{x} f(x) \\ s.t.\ c_i(x) = 0,\ i = 1, \cdots, m' \\ c_i(x) \geq 0,\ i = m'+1, \cdots, m \end{aligned} \tag{15}$$

满足：

1. x^* 是局部极小点；

2. $f(x), c_i(x) \in C^1$;

3. 满足约束规范条件；

则满足 KKT 条件 $\exists \lambda_i$：

1. *(Langrange)* $\Delta f(x^*) = \sum_{i=1}^{m} \lambda_i \Delta c_i(x^*)$

2. *(等式约束)* $c_i(x^*) = 0,\ i = 1, \cdots, m'$

3. *(不等式约束)* $c_i(x) \geq 0,\ i = m' + 1, \cdots, m$

4. *(非负乘子条件)* $\lambda_i \geq 0,\ i = m' + 1, \cdots, m$

5. *(互补松弛条件)* $\lambda_i c_i(x^*) = 0,\ i = m' + 1, \cdots, m$

基于上述命题，可以将原始优化问题转化为其对偶问题来解决，首先构造 Lagrange 函数：

$$\min_{w,b} \max_{\lambda \geq 0} L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_i \lambda_i [1 - y_i (w^T x_i + b)] \tag{16}$$

由于原问题是一个二次凸优化，满足 KKT 条件，原问题可以转化为对偶形式：

$$\max_{\lambda \geq 0} \min_{w,b} L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_i \lambda_i [1 - y_i (w^T x_i + b)] \tag{17}$$

对于 $\min\limits_{w,b} L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_i \lambda_i [1 - y_i (w^T x_i + b)]$ 带入 KKT 条件有

$$\begin{aligned} &\frac{\partial L}{\partial w} = w - \sum_i \lambda x_i y_i = 0 \\ &\frac{\partial L}{\partial b} = \sum_i \lambda_i y_i = 0 \\ &\lambda_i [y_i (w^T x_i + b) - 1] = 0,\ i = m' + 1, \cdots, m \end{aligned} \tag{18}$$

则

$$\begin{aligned} w &= \sum_i \lambda x_i y_i \\ \sum_i \lambda_i y_i &= 0 \end{aligned} \tag{19}$$

代入式17转化为

$$\begin{aligned} &\max_{\lambda \geq 0} \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i^T x_j) \\ s.t. \ &\sum_i \lambda_i y_i = 0 \end{aligned} \tag{20}$$

该问题是一个二次规划问题，可以比原问题更快地进行求解。重新考察式16，发现若最优化 $\max_{\lambda \geq 0} L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_i \lambda_i [1 - y_i (w^T x_i + b)]$，则 $\lambda_i \neq 0$ 当且仅当 $1 - y_i (w^T x_i + b)$，此时 (X_i, y_i) 为支撑向量。那么基于式19，有 $w = \sum_i \lambda x_i y_i$，进一步由于 $\sum_i \lambda_i y_i = 0$ 支撑向量中正例和负例式均匀的，可以通过求取支撑向量关于 w 的偏置的均值来得到超平面的偏置，具体表达为 $b = mean\{y_i - w^T x_i : \lambda_i \neq 0\}$，结果为：

$$\begin{aligned} w &= \sum_i \lambda_i y_i x_i \\ b &= mean\{y_i - w^T x_i : \lambda_i \neq 0\} \end{aligned} \tag{21}$$

类似地，可以对软间隔的 SVM 采取相似的操作，原问题转化为强对偶问题

$$\begin{aligned} \max_{\lambda, \mu} \min_{w, b, \epsilon} &\frac{1}{2} w^T w + C \sum_{i=1}^{n} \epsilon_i \\ &+ \sum_i \lambda_i [1 - \epsilon_i - y_i (w^T x_i + b)] - \sum_i \mu_i \epsilon_i \\ s.t. \quad &\lambda_i \geq 0 \\ &\mu_i \geq 0 \end{aligned} \tag{22}$$

带入 KKT 条件有：

$$\begin{aligned} &w = \sum_i \lambda_i y_i x_i \\ &\sum_i \lambda_i y_i = 0 \\ &\lambda_i + \mu_i = C \\ &\lambda_i [1 - \epsilon_i - y_i (w^T x_i + b)] = 0 \\ &\mu_i \epsilon_i = 0 \end{aligned} \tag{23}$$

代回原式有优化问题转变为：

$$\begin{aligned} \max_{\lambda} &= \sum_i \lambda_i - \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i^T x_j) \\ s.t. \ &\sum_i \lambda_i y_i = 0 \\ &\lambda_i \geq 0 \end{aligned} \tag{24}$$

类似地，有 $w = \sum_i \lambda_i y_i x_i$，类似于硬间隔的情况，$\lambda_i \neq 0$ 时 (x_i, y_i) 为准支撑向量，对于超平面参数有影响，需要注意的是，此时必然有向量在间隔上，否则结果可以更优，则需要取 $\max\limits_{i;\lambda_i \neq 0} y_i(w^T x_i + b)$ 的向量，此时必然有 $y_i(w^T x_i + b) = 1$。类似地，取均值有：

$$\begin{aligned} &w = \sum_i \lambda_i y_i x_i \\ &b = mean\{y_i - w^T x_i : \lambda_i \neq 0, y_i(w^T x_i + b) = \max_{i;\lambda_i \neq 0} y_i(w^T x_i + b)\} \end{aligned} \tag{25}$$

或者也可以简化为：

$$\begin{aligned} &w = \sum_i \lambda_i y_i x_i \\ &b = mean\{y_i - w^T x_i : \lambda_i \neq 0\} \end{aligned} \tag{26}$$

2.2.3 非线性情况推导

数据线性可分是一个比较强的条件，在现实世界中具有较大的局限性，需要使用核函数将原始数据空间映射到新的特征空间，特征空间中数据线性可分。而无论是训练还是预测的时候，都只需内积运算进行度量，所以我们可以转而转变原始数据空间中的内积运算来将原始的欧几里得内积空间转变为一个新的内积空间。具体而言，可以定义核函数为新的内积空间内积

$$K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R} \tag{27}$$

且 $K \in C^\infty$ 以保证特征空间的光滑性。

接下来介绍几种常见的非线性核函数，和其边界的可视化:

1. **线性核函数 (Linear):**

$$K(x_1, x_2) = x_1^T x_2 \tag{28}$$

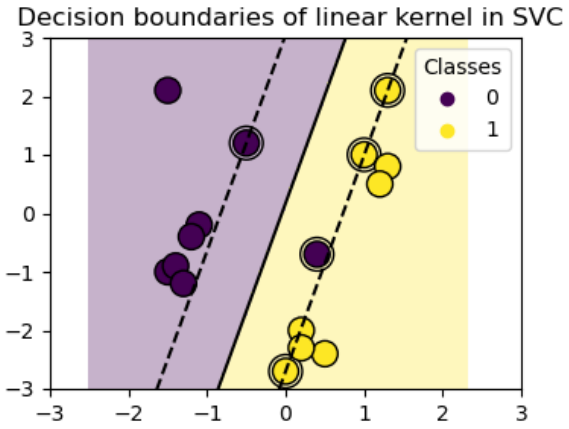


图 1: Linear 核边界 [2]

2. **多项式核 (Poly):**

$$K(x_1, x_2) = (\gamma x_1^T x_2 + r)^d \tag{29}$$

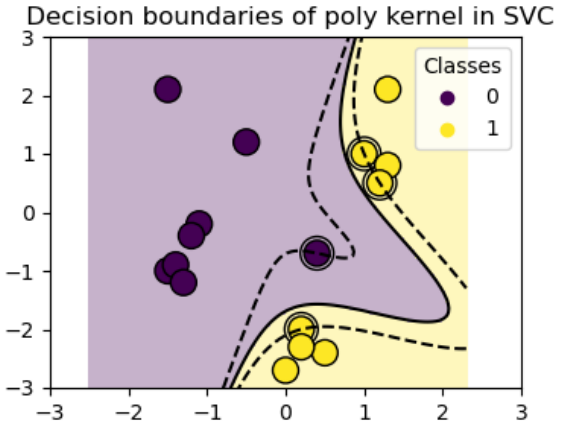


图 2: Poly 核边界 [2]

3. **Sigmoid 核 (Sigmoid):**

$$K(x_1, x_2) = tanh(\gamma x_1^T x_2 + r) \tag{30}$$

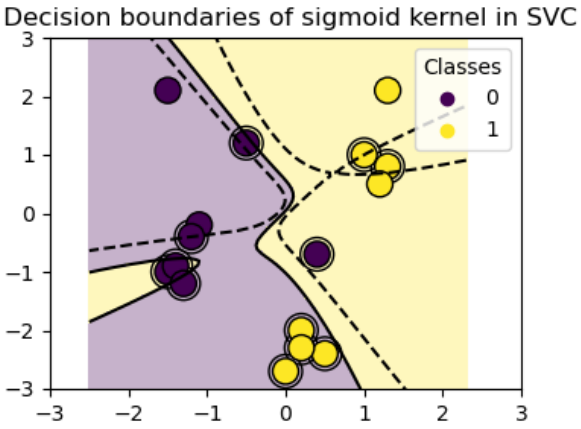


图 3: Sigmoid 核边界 [2]

4. **Radial Basis Function 核 (RBF):**

$$K(x_1, x_2) = exp\{-\gamma \parallel x_1 - x_2 \parallel^2\} \tag{31}$$

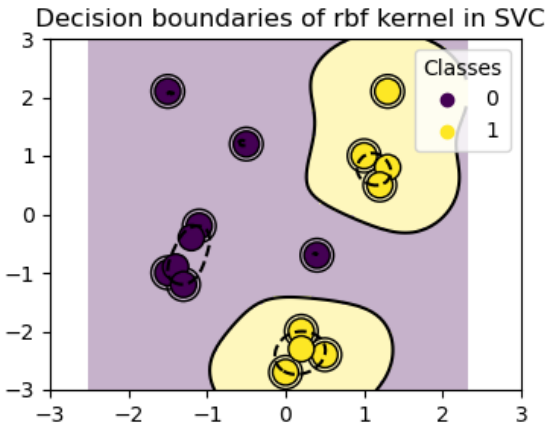


图 4: RBF 核边界 [2]

2.2.4 多分类 SVM 推导

传统 SVC 只能完成二分类任务，对于多分类问题需要采取更进一步地设计。一般而言，将二分器拓展为多分类器有两种经典的策略，分别是“one v.s. one” 策略 (ovo) 和”one v.s. rest” 策略 (ovr)。

ovo 策略的核心思想就是一对一的训练多个二分类器，再整合所有二分类器投票组合为一个多分类器。对于一个 N 类问题，需要训练 $\frac{n(n-1)}{2}$

个二分类器，在预测时，将输入通过二分类器进行预测，最后将所有分类器预测出的结果进行投票统计，投票最多的类别就是结果。

ovr 策略的核心策略就是一对多的训练多个二分类器，每次将某一类作为一类，剩余类作为另一类，形成一个约简任务，训练一个二分类 SVC。对于一个 N 分类问题，需要训练 N 个二分类器，在预测时，将输入在 N 个二分类器上进行预测，选取最好的结果作为预测结果（距离超平面的距离最远）。

2.3 线性可分随机数据生成

实验过程中，需要构造合适的测试数据，测试数据需要具有相当的线性可分性，在此，我才用的是在不同均值的多元正态分布中进行采样，数据可视化结果见下图5，25 个蓝色数据采样自 $N(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix})$，25 个红色数据采样自 $N(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix})$，绿色直线为理想分割超平面 $x+y=2$。

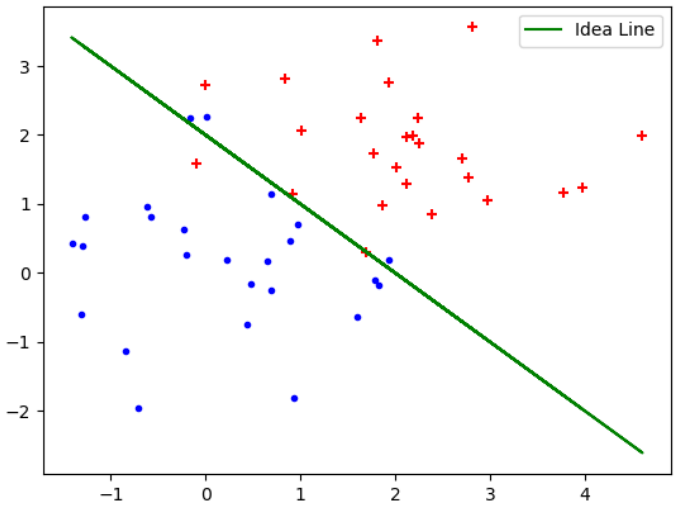


图 5: 随机数据生成结果

3 程序框图

代码使用 Python 实现，由于实验并未要求必须实现全部细节，实验代码使用 sklearn 模块中的 SVM 模块进行实验。事实上，我也初步实现了硬间隔 Linear SVC 的代码，由于时间限制，我仅实现了这种最简单的非对偶情况，同时并未实现二次凸优化部分，而是调用时下性能极优的 Gurobi 库进行凸优化，该库属于商用优化求解器，代码并不开源。接下来我将具体介绍一下我实现的 SVM 代码框架:

Algorithm 1: Train a linear SVC

Data: $X \in \mathbb{R}^{n*N}$;$Y \in \{-1,1\}^n$;C:Balanced parameter

Result: w, b:Model parameters

1 Initialize w, b;

2 Construct object equation $\frac{1}{2}w^Tw + C\sum_{i=1}^{n}\epsilon_i$;

3 Construct constraint $y_i(w^Tx_i + b) \geq 1 - \epsilon_i$;

4 Solve this quadratic convex optimization through *Gurobi*;

5 **return** w, b

预测只需返回 $sign\{w^Tx + b\}$ 即可。

4 实验代码

该项目所有代码已上传github 仓库，在此也将展示所有代码，首先展示随机二维数据生成的代码:

Code Listing 1: 随机二维二分类线性可分数据生成代码

```
import matplotlib.pyplot as plt
import numpy as np

def get_random_data(mu=(0,0), sigma=(1,1), n=10, label=1):
    X = [(np.random.normal(mu[0], sigma[0]), np.random.normal(mu[1], sigma[1])) for
        i in
        range(n)]
    Y = [label for i in  range(n)]
    return np.array(X), np.array(Y)
```

接着展示我自己实现的 Linear SVC 代码:

Code Listing 2: 基于 Gurobi 实现的 Linear SVC 代码

```
from gurobipy import *
import gurobipy as gp

class binary_svm:
    def __init__(self, in_dim, name=''):
        self.in_dim = in_dim
        self.model =Model("binary_svm_{}". format(name))
        self.w = []
        self.w_val = np.array([0 for i in  range(in_dim)])
        for i in  range(in_dim):
            self.w.append(self.model.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype
                =GRB.CONTINUOUS, name='w_{}'.
                format(i)))
        self.b = self.model.addVar(lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.
            CONTINUOUS, name='b')
        self.b_val = 0

    def train(self, x, y, c=1.0):
        """
        Train the binary SVM
        :param x: n * in_dim
        :param y: n
        :return: None
        """
        N, _ = x.shape
        ep = []
        for i in  range(N):
            ep.append(self.model.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS,
                 name='epsilon_{}'.
                format(i)))

        for i in  range(N):
            val = self.b
            for j in  range(self.in_dim):
                val += self.w[j] * x[i][j]
            self.model.addConstr(y[i]*val >= 1-ep[i])

        object_eq = 0
        for i in  range(self.in_dim):
            object_eq += .5*self.w[i] * self.w[i]
        for i in  range(N):
            object_eq += c*ep[i]
        self.model.setObjective(object_eq, GRB.MINIMIZE)
```

```
        self.model.optimize()

        for i in  range(self.in_dim):
            self.w_val[i] = self.w[i].x
        self.b_val = self.b.x

        print(self.w_val)
        print(self.b_val)

    def predict(self, x):
        """
        Predict
        :param x: N * in_dim
        :return:
        """
        N, _ = x.shape
        bs = np.expand_dims(self.b, axis=0).repeat(N) # N
        result = np.einsum('i,ki->k', self.w, self.x) + bs
        result[result<0] = -1
        result[result>=0] = 1
        return result

    def test(self, x, y):
        N, _ = x.shape
        bs = np.expand_dims(self.b, axis=0).repeat(N)  # N
        result = np.einsum('i,ki->k', self.w, self.x) + bs
        result[result < 0] = -1
        result[result >= 0] = 1
        pr = np. sum(result==y)/N
        print('Precision:', pr)
```

最后展示基于 sklearn 实现的 SVM 代码：

```
from sklearn.svm import SVC
from get_mnist import *
from sklearn.preprocessing import StandardScaler
from get_rand_data import *
import matplotlib.pyplot as plt

if __name__=='__main__':
    # Read Data

    # dataset = mnist_loader(.8)
    # train_x, train_y = dataset.get_train_data()
    # test_x, test_y = dataset.get_test_data()
```

```
# train_x = train_x[:1000]
# train_y = train_y[:1000]
# test_x = test_x[:100]
# test_y = test_y[:100]

train_x1, train_y1 = get_random_data(label=1, n=100)
train_x2, train_y2 = get_random_data((2,2), (1,1), label=2, n=100)
train_x = np.append(train_x1, train_x2, axis=0)
train_y = np.append(train_y1, train_y2, axis=0)

print(train_x.shape, train_y.shape)

model = SVC(C=1.0, kernel='linear')
model.fit(train_x, train_y)

# print(model.coef_)
# print(model.intercept_)

# pred_y = model.predict(test_x)
# print(pred_y.shape)
# print(pred_y)
# pr = np.sum(pred_y==test_y)
# print(pr,'%')

p1, p2 = model.coef_[0]
b = model.intercept_[0]

plt.plot(train_x[:,0], -p1/p2*train_x[:,0]-b/p2, c='y', label='Predict␣Line')
plt.plot(train_x[:,0], 2-train_x[:,0], c='g', label='Idea␣Line')

plt.scatter(train_x1[:,0], train_x1[:,1], marker='.', c='b')
plt.scatter(train_x2[:,0], train_x2[:,1], marker='+', c='r')

print(train_x.shape)

plt.legend()

plt.show()
```

5 实验结果与分析

5.1 二维线性可分数据实验

在本实验中采用章节2.3中的算法生成数据，并将其应用于软间隔的SVC 上进行实验。实验中生成两类训练数据，每类数据各 100 个，生成的正态分布分别为 $N(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix})$ 和 $N(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix})$，再从其中各采样 50 个数据作为测试集。基于此，控制软间隔平衡参数 C，结果可视化见下图6，预测准确率见下表1。

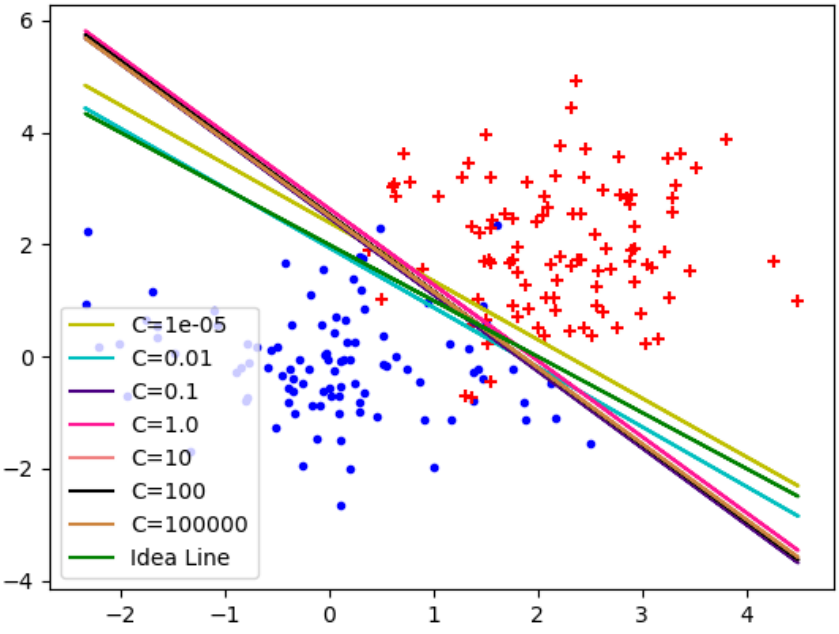


图 6: 二维线性可分二分类 SVC 可视结果

C	训练集正确率 (%)	测试集正确率 (%)
1e-05	94.5	94.0
0.01	94.0	92.0
0.1	94.5	91.0
1.0	**95.0**	**94.0**
10	94.5	92.0
100	94.5	92.0
100000	94.5	92.0

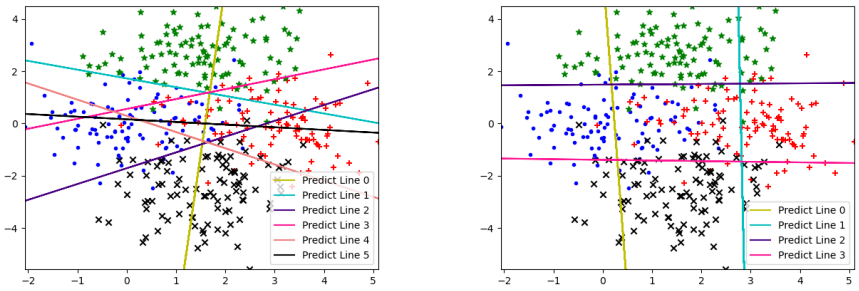
表 1: 二维线性可分二分类 SVC 准确率结果

从正确率的角度进行分析，正确率与 C 之间未呈现显著的线性关系，无法简单的认为 C 越大或越小结果就越优。当 C 过小时，结果会倾向于忽视异常向量的影响，会倾向于有更多异常向量的结果，进而导致当 C 越小时，会越倾向于拟合越带有更多错误的结果，牺牲了一部分精度，但同时又会在一定程度上提高模型的泛化能力。当 C 越大时，会越重视错误向量的影响，进而会逼近于原始的硬间隔 SVM 的结果，其结果可能会受到少量异常向量的影响，模型鲁棒性较差。

从可视化角度进行分析， C 较小时的结果更贴近于理想超平面，但是当 C 过小时，其偏置将有可能与理想结果产生较大的偏差。当 C 较大时，超平面收到少量支撑向量的影响严重，预测超平面的斜率与理想超平面相差甚远，理论上来说泛化性能较差。

综上，C 的选取极度依赖经验和尝试，C 过大时会过度依赖少量支撑向量，导致结果极不稳定，泛化性能下降；C 过小时，算法会忽略异常向量的影响，虽然可以提高模型的泛化能力，但是牺牲了模型的精度。

继承表1的经验，设置 C=1 进行四分类实验，采样数据 $N(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix})$、$N(\begin{bmatrix}3\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix})$、$N(\begin{bmatrix}1.5\\1.5\sqrt{3}\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix})$ 和 $N(\begin{bmatrix}1.5\\-1.5\sqrt{3}\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix})$，从中各自采样 100 个样本作为训练集，各自采样 50 个作为测试集，最终使用 ovo 策略得到的多分类可视化结果见图7，准确率结果见图2。



(a) ovo 策略　　(b) ovr 策略

图 7: 多分类 SVC 可视化结果

策略	训练准确率 (%)	测试准确率 (%)
ovo	**85.25**	85.0
ovr	84.75	**87.0**

表 2: 多分类 SVC 准确率结果

从预测准确率上无法说明哪种策略更优，但是 ovo 策略的子分类器在多分类情况下会远远多于 ovr 策略的子分类器，可以认为 ovo 对于数据信息的利用更为充分，因此结果应该也更为鲁棒，但是这也会导致牺牲拟合精度。ovr 策略在训练时存在样本不均衡问题，通常对于一个各类样本均匀分布的数据分布，某一类样本数量与余类样本数量的差异也就会越大，导致严重的样本不均衡问题，影响模型的效果。

综上所述，ovo 和 ovr 策略并不存在明显的优劣，需要根据情况来选择更为合适的策略。

5.2 MNIST 数据集

基于提示，我将 SVM 技术用于公开手写数据集 MNIST[1] 进行测试。该数据集共有 70000 张 28*28 的灰度图像，并人为标注为了 0-9 十类，要求通过该数据集实现手写数字识别任务。

任务设置训练集和测试集为 8：2，实验结果见下表3。其中控制了软间隔的平衡系数和核函数的类型，结果表明，取 C=1.0，核函数为 Poly 时效果最好，其正确率已经接近现代的 CNN 神经网络方法。

C	核函数	测试准确率 (%)
1.0	Linear	92.828
0.001	Linear	92.085
1000	Linear	92.085
1.0	RBF	98.235
1.0	Poly	**98.678**
1.0	Sigmoid	82.700

表 3: MNIST 数据集结果

6 遇到的问题及其解决措施

6.1 sklearn 库中无经典的 ovr 策略实现

在阅读源码之后，发现 sklearn 库中的 ovr 策略是基于 ovo 策略计算的 $\frac{n(n-1)}{2}$ 个超平面进行计算的，预测结果与 ovo 相同，不符合实验的目的。于是需要自行实现 ovr 策略。具体而言，我将其封装为一个库，成员变量核方法继承自 sklearn 的 SVC 求解器，具体实现代码如下：

Code Listing 3: OvR 策略 SVC 代码

```
class ovr_svc:
    def __init__(self, n_cls):
        """
        Initial the ovr SVC
        :param n_cls: the number of classes
        """
        self.n_cls = n_cls
        self.models = [SVC(C=1.0, kernel='linear', probability=True) for i in
            range(n_cls)]
        self.intercept_ = []
        self.coef_ = []
    def fit(self, X, Y):
        """
        Train the SVC
        :param X: Data
        :param Y: Label
        """
        for i in  range(self.n_cls):
            _Y = Y.copy()
```

```
            _Y[Y==i+1] = 0
            _Y[Y!=i+1] = -1
            self.models[i].fit(X,_Y)
            self.intercept_.append(self.models[i].intercept_[0])
            self.coef_.append(self.models[i].coef_[0])
    def predict(self, X):
        """
        Predict the label of data
        :param X: Data
        :return: Predicted label
        """
        result = []
        for i in  range(self.n_cls):
            # print(self.models[i].predict_proba(X))
            result.append(self.models[i].predict_proba(X)[:,-1])
        result = np.array(result)
        result = np.argmax(result, axis=0) + 1
        return result
```

6.2 大规模数据情况运算较慢

因为我 sklearn 开发时并未接入 cuda，所以无法高效地使用 gpu 资源进行并行计算。虽然目前业界已经有了一系列实现 numpy 等基础库使用 cuda 进行加速的方案，但是这需要大量的时间去配置环境和熟悉操作，该实验提供的时间是不足够我完成这部分的学习的。

参考文献

[1] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.