机器学习实验报告 5

姓名：陈海弘

学号：23354049

陈海弘

2024.11.21

Contents

1 摘要

本次实验引入的库有：numpy，pandas，cvxopt，matplotlib。实验内容包括线性可分支持向量机与硬间隔最大化、线性支持向量机与软间隔最大化、非线性支持向量机与核函数。实验目的是通过求解对偶问题，得到最优解 $\boldsymbol{\alpha}^*$，并计算 $\boldsymbol{\omega}^*$ 和 b^*，最后绘制散点图与决策边界。

2 线性可分支持向量机与硬间隔最大化

2.1 数据读入

读入数据集'dataset1.csv', 把数据类型都转换成 np.double 类型，并画出数据集的散点图，给正样本（y 为 +1）和负样本（y 为-1）分别标上不的颜色。

```
data = pd.read_csv('dataset1.csv')
data = data.astype(np.double)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
plt.figure(figsize=(8, 6))
positive_samples = X[y == 1]
negative_samples = X[y == -1]
plt.scatter(positive_samples[:, 0], positive_samples[:, 1], color='
    blue', label='Positive␣(+1)', marker='o')
plt.scatter(negative_samples[:, 0], negative_samples[:, 1], color='
    red', label='Negative␣(-1)', marker='x')
plt.xlabel('Feature␣1')
plt.ylabel('Feature␣2')
plt.title('Scatter␣Plot␣of␣Dataset')
plt.legend()
plt.grid()
plt.show()
```

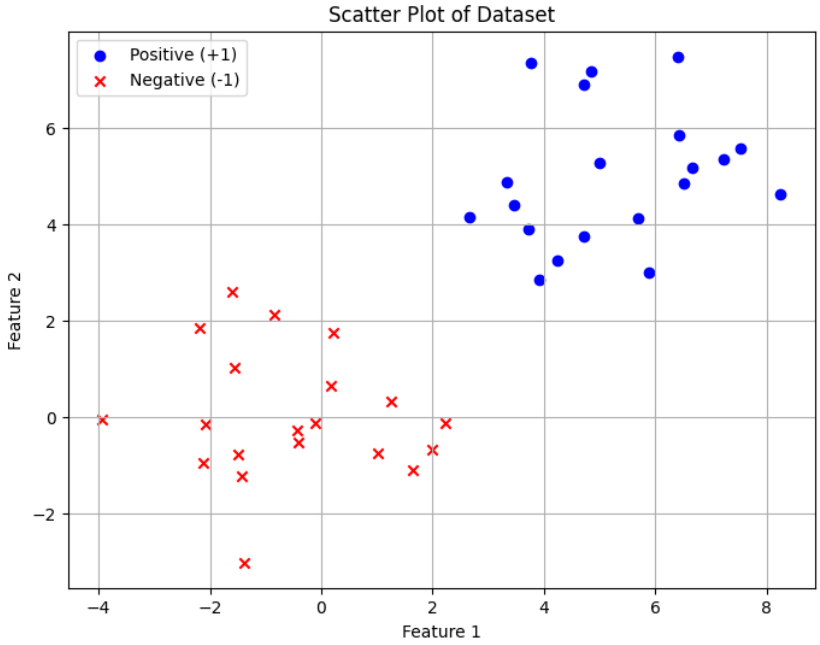

Figure 1: 数据集散点图

2.2 对偶问题求解

求解如下对偶问题：

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{m}\alpha_i, \\ \text{s.t.} \quad & \sum_{i=1}^{m}\alpha_i y_i = 0, \\ & \boldsymbol{\alpha} \geq \mathbf{0}. \end{aligned}$$

这个优化问题是一个二次规划问题，其中参数定义如下：

- P 是一个 $m \times m$ 的矩阵，其中 $P_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$，
- q 是一个 $m \times 1$ 的列向量，所有值均为 -1，即

$$q := \begin{bmatrix} -1 & -1 & \cdots & -1 \end{bmatrix}^T,$$

- $G := \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}_{m\times m} = -\boldsymbol{I}$，$\boldsymbol{I}$ 为单位矩阵，
- h 是一个 $m \times 1$ 的零向量，即

$$h := \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^T,$$

- $A := \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T$，$b := \begin{bmatrix} 0 \end{bmatrix}$(一个标量)。

将上述参数送入求解器 `solvers.qp()` 中即可得到最优解 α^*。

附：P 矩阵的计算方法

设

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{m1} & x_{m2} \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix},$$

计算

$$X' = \begin{bmatrix} x_{11}y_1 & x_{12}y_1 \\ x_{21}y_2 & x_{22}y_2 \\ \vdots & \vdots \\ x_{m1}y_m & x_{m2}y_m \end{bmatrix} = X * Y \text{ （注意这里是星乘）},$$

则 $P = X'X'^T$。

本实验的目标是通过求解对偶问题， 到最优解 $\boldsymbol{\alpha}^*$。我们利用 `cvxopt.solvers.qp()` 求解器完成优化，以下为具体实现思路:

1. 数据与参数初始化

- 输入数据 X 的维度为 $m \times n$，其中 m 表示样本数，n 表示特征维数。
- 标签向量 y 被重塑为列向量形式，即 $\boldsymbol{y} \in \mathbb{R}^m$。

2. 构造矩阵 P

- 根据公式计算 X':

$$X' = X * \boldsymbol{y},$$

其中 $*$ 表示逐元素相乘（广播操作）。

- 计算矩阵 P 为:

$$P = X'X'^{T}.$$

矩阵 P 是目标函数中二次项系数矩阵。

3. 定义其他参数

- q 是一个长度为 m 的列向量，所有元素均为 -1:

$$q = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}.$$

- G 是一个 $m \times m$ 的对角矩阵，等于负单位矩阵:

$$G = -\boldsymbol{I}.$$

- h 是一个长度为 m 的零向量:

$$h = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

- A 是标签向量 $\boldsymbol{y}$ 的转置:

$$A = \boldsymbol{y}^{T}.$$

- b 是一 标量，值为 0：

$$b = \left[0\right].$$

4. 转换为 cvxopt 格式

- 矩阵 P, q, G, h, A, b 被转换为 `cvxopt` 的专用矩阵格式 `matrix`。

5. 调用优化求解器

- 调用 `solvers.qp()`，输入定义的参数：

```
solvers.qp(P, q, G, h, A, b)
```

- 如果求解报错，可以在调用时添加参数 `kktsolver='ldl'` 来避免奇异矩阵问题。

6. 解析优化结果

- 求解器返回的结果存储在 `solution['x']` 中。
- 将结果展平成一维向量，即为 $\boldsymbol{\alpha}$ 的最优解。

7. 输出最优解

最终，程序输出的 $\boldsymbol{\alpha}^*$ 即为对偶问题的最优解，满足约束条件：

$$\boldsymbol{\alpha}^* \geq 0, \quad \sum_{i=1}^{m} \alpha_i^* y_i = 0.$$

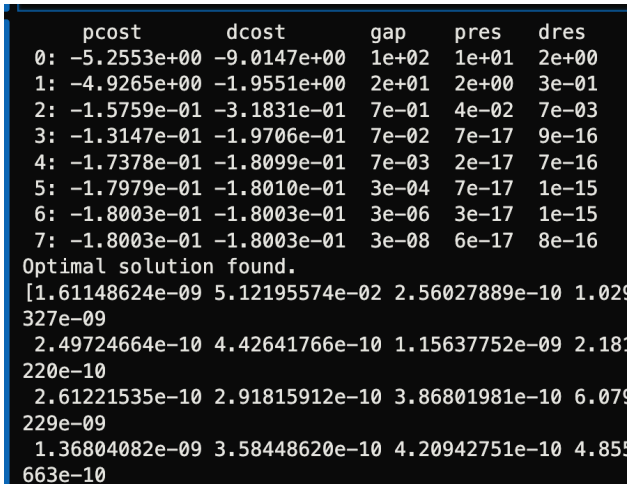

Figure 2: 对偶问题求解结果

2.3 计算 $\boldsymbol{\omega}^*$ 和 b^*

通过求解对偶问题得到最优解 $\boldsymbol{\alpha}^*$ 后，可以计算 $\boldsymbol{\omega}^*$ 和 b^*，具体步骤如下：

1. 计算 $\boldsymbol{\omega}^*$

$$\boldsymbol{\omega}^* = \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{x}_i,$$

其中：

- $\boldsymbol{\alpha}^* \in \mathbb{R}^m$ 是对偶问题的最优解；
- $\boldsymbol{y} \in \mathbb{R}^m$ 是样本标签向量；
- $\boldsymbol{x}_i$ 是样本输入向量。

2. 计算 b^*

$$b^* = y_j - \boldsymbol{\omega}^{*T} \boldsymbol{x}_j,$$

其中 j 是 $\boldsymbol{\alpha}^*$ 中一个正分量 $\alpha_j^* > 0$ 的下标。

3. 筛选 α^* 的正分量

由于求解器 `solvers.qp()` 求得的解是近似解，$\boldsymbol{\alpha}^*$ 中很多理论上为 0 的分量可能略大于 0。因此，为筛选出有效的正分量，可以采取以下两种方法：

- **方法 1：设置阈值**将 $\boldsymbol{\alpha}^*$ 中小于某一阈值 ϵ 的分量筛去（例如 $\epsilon = 10^{-6}$），剩下的分量中选取一个正分量对应的下标 j 来计算 b^*：

$$\boldsymbol{\alpha}_{\text{筛选}} = \{\alpha_i^* \mid \alpha_i^* > \epsilon\}.$$

- **方法 2：取最大分量**直接取 $\boldsymbol{\alpha}^*$ 中最大的分量 $\max(\boldsymbol{\alpha}^*)$ 对应的下标 j 计算 b^*。

4. 注意事项

无论选择哪种方法，都需要确保：

- 所选 $\alpha_j^* > 0$；
- 使用的下标 j 是有效的（即 $j \in \{1, 2, \ldots, m\}$）。

```
threshold = 1e-5
alpha[alpha < threshold] = 0
alpha = alpha.reshape(-1, 1)
y = y.reshape(-1, 1)
omega = np.sum(alpha * y * X, axis=0)
print("Optimal ω*:", omega)
j = np.argmax(alpha)
b = y[j] - np.dot(omega, X[j])
print("Optimal b*:", b)
```

```
Optimal ω*: [0.40821111 0.43979826]
Optimal b*: [-1.85568125]
```

Figure 3: 计算 $\boldsymbol{\omega}^*$ 和 b^*

2.4 绘制散点图与决策边界

画出数据集的散点图，给正样本（y 为 +1）和负样本（y 为-1）分别标上不同的颜色，再为支持向量（训练数据中 $\alpha_j^* > 0$ 的对应的样本）标上不同的颜色，并画出决策边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 0$ 和间隔边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 1$ 与 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = -1$。

1. 决策边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 0$ 的计算公式为：

$$x_2 = -\frac{\omega_1 \cdot x_1 + b}{\omega_2}.$$

2. 上间隔边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 1$ 的计算公式为：

$$x_2 = -\frac{\omega_1 \cdot x_1 + b - 1}{\omega_2}.$$

3. 下间隔边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = -1$ 的计算公式为：

$$x_2 = -\frac{\omega_1 \cdot x_1 + b + 1}{\omega_2}.$$

绘制方法的实现步骤如下：

1. 计算特征 x_1 的取值范围：

$$x_{\min} = \min(X[:, 0]) - 1, \quad x_{\max} = \max(X[:, 0]) + 1.$$

2. 在该范围内均匀采样若干点，用于计算决策边界和间隔边界：

$$x_{\text{values}} = \text{linspace}(x_{\min}, x_{\max}, 500).$$

3. 使用上述公式计算 x_2 的值。

Listing 1: 计算并绘制决策边界和间隔边界

```
# 计算 x_1 的取值范围
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
x_values = np.linspace(x_min, x_max, 500) # 均匀采样

# 计算决策边界和间隔边界
decision_boundary = -(omega[0] * x_values + b) / omega[1]
margin_positive = -(omega[0] * x_values + b - 1) / omega[1]
margin_negative = -(omega[0] * x_values + b + 1) / omega[1]

# 绘图
plt.plot(x_values, decision_boundary, 'k-', label='Decision Boundary')
plt.plot(x_values, margin_positive, 'k--', label='Margin (+1)')
plt.plot(x_values, margin_negative, 'k--', label='Margin (-1)')
plt.legend()
```

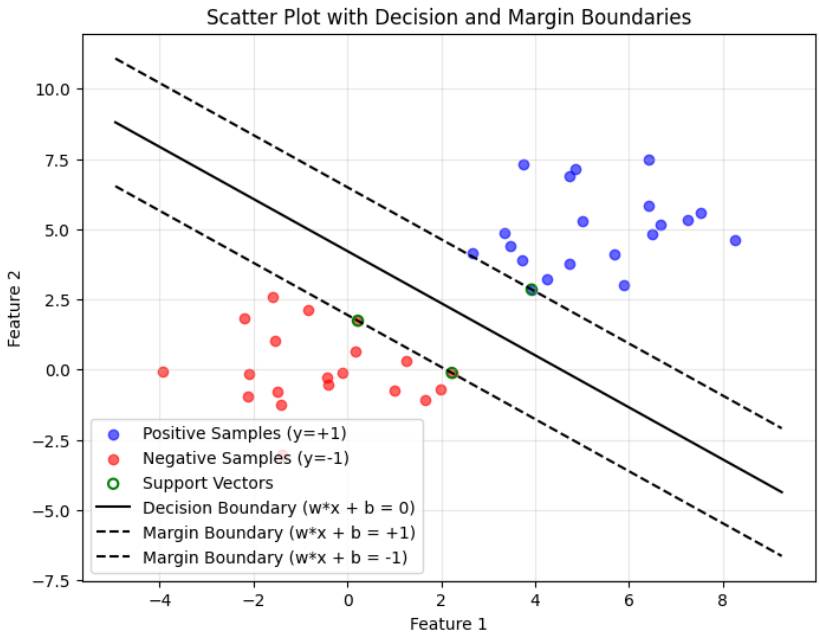

Figure 4: 绘制散点图与决策边界

可以看到，红色和蓝色的点被正确分类，且决策边界和间隔边界被正确绘制。

3 线性支持向量机与软间隔最大化

3.1 数据读入

```
data = pd.read_csv('dataset2.csv')
data = data.astype(np.double)
print(data.head())
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
plt.figure(figsize=(8, 6))
positive_samples = X[y == 1]
negative_samples = X[y == -1]
plt.scatter(positive_samples[:, 0], positive_samples[:, 1], color='
    blue', label='Positive␣(+1)', marker='o')
plt.scatter(negative_samples[:, 0], negative_samples[:, 1], color='
    red', label='Negative␣(-1)', marker='x')
plt.xlabel('Feature␣1')
plt.ylabel('Feature␣2')
plt.title('Scatter␣Plot␣of␣Dataset')
plt.legend()
plt.grid()
plt.show()
```

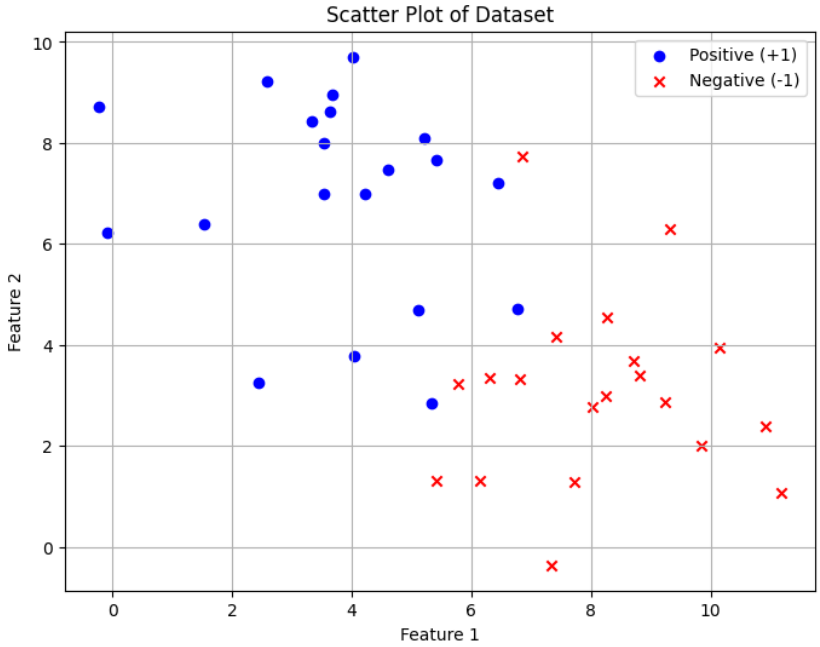

Figure 5: 数据集散点图

3.2 对偶问题求解

硬间隔和软间隔的求解方法相同，只是软间隔在求解时多了一个惩罚系数 C，这个惩罚系数是用来控制间隔的松弛程度，C 越大，间隔越小，C 越小，间隔越大。

支持向量机的目标是通过二次规划（Quadratic Programming, QP）求解优化问题。以下描述了构造二次规划问题的具体步骤：

首先，给定数据集的规模：样本数 m 和特征维度 n，分别为：

$$m = \text{样本数}, \quad n = \text{特征维数}.$$

定义超参数 C，用于控制松弛变量的权重：

$$C = 7.4.$$

计算核矩阵 K（线性核）：

$$K = XX^T.$$

构造矩阵 P 和向量 q：

$$P = \text{diag}(y) \cdot K \cdot \text{diag}(y),$$

$$q = -\mathbf{1}.$$

构造矩阵 G 和向量 h：

$$G = \begin{bmatrix} -\boldsymbol{I} \\ \boldsymbol{I} \end{bmatrix}, \quad h = \begin{bmatrix} \mathbf{0} \\ C \cdot \mathbf{1} \end{bmatrix}.$$

构造矩阵 A 和标量 b：

$$A = \boldsymbol{y}^T, \quad b = 0.$$

接下来，将上述构造的参数矩阵传入二次规划求解器 `solvers.qp()`，求解问题：

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T P \boldsymbol{\alpha} + \boldsymbol{q}^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & G\boldsymbol{\alpha} \preceq h, \\ & A\boldsymbol{\alpha} = b. \end{aligned}$$

代码实现如下：

Listing 2: 求解二次规划问题

```
# 使用二次规划求解器
sol = solvers.qp(P, q, G, h, A, b)
# 提取结果
alpha = np.array(sol['x']).flatten()
print("Optimal alphas:", alpha)
```

通过求解器的输出，可以获得拉格朗日乘子 $\boldsymbol{\alpha}^*$：

$$\boldsymbol{\alpha}^* = \text{sol['x']}$$

这些值将用于进一步计算模型参数 $\boldsymbol{\omega}^*$ 和 b^*。

```
     pcost       dcost       gap    pres   dres
 0: -4.8039e+01 -1.6021e+03  4e+03  8e-01  2e-13
 1: -3.1598e+01 -5.0113e+02  7e+02  1e-01  1e-13
 2: -2.5083e+01 -1.0900e+02  1e+02  1e-02  1e-13
 3: -3.3721e+01 -5.5723e+01  3e+01  2e-03  1e-13
 4: -3.8079e+01 -4.1395e+01  3e+00  1e-04  2e-13
 5: -3.9638e+01 -4.0438e+01  8e-01  5e-06  2e-13
 6: -3.9848e+01 -3.9859e+01  1e-02  8e-08  2e-13
 7: -3.9850e+01 -3.9851e+01  1e-04  8e-10  2e-13
 8: -3.9850e+01 -3.9850e+01  1e-06  8e-12  1e-13
Optimal solution found.
Optimal alphas: [1.16398057e-09 4.06543857e-09 9.20
108254e-10 7.39999999e+00
 1.71371466e-09 1.98999079e+00 8.05507129e-08 1.793
41543e-09
 1.96568605e-09 2.07028434e-09 4.28044148e-09 6.108
07383e-09
 1.87313280e-09 1.82557917e-09 1.78677450e-09 5.594
99677e+00
 1.87107139e-09 1.46524646e-09 1.77978674e-09 1.798
57026e-09
 2.27431103e-09 2.55517606e-09 3.60500608e+00 7.400
00000e+00
 1.36569297e-09 1.88516132e-09 9.83750544e-10 1.758
68242e-09
 1.53430783e-09 6.82366753e-10 1.43217487e-09 1.762
15063e-09
 1.80685887e-09 7.39999997e+00 1.86289608e-09 7.399
99998e+00
 2.03699571e-09 1.73991101e-09 2.13121653e-09 1.723
60686e-09]
```

Figure 6: 对偶问题求解结果

3.3 计算 $\boldsymbol{\omega}^*$ 和 b^*

在得到优化后的 $\boldsymbol{\alpha}^*$ 后，可以计算支持向量机模型的参数 $\boldsymbol{\omega}^*$ 和偏置项 b^*。以下是具体步骤：

1. **筛选非零 $\boldsymbol{\alpha}^*$** 因为求解器给出的 $\boldsymbol{\alpha}^*$ 是近似解，有些数值接近 0。我们设置一个阈值 threshold，将所有小于该阈值的 $\boldsymbol{\alpha}$ 置为 0：

$$\text{threshold} = 10^{-5}.$$

过滤后：

$$\boldsymbol{\alpha}^* = \begin{cases} 0, & \text{若} \alpha_i^* < \text{threshold}, \\ \alpha_i^*, & \text{其他}. \end{cases}$$

2. **计算 $\boldsymbol{\omega}^*$** 模型的权重向量 $\boldsymbol{\omega}^*$ 可通过以下公式计算：

$$\boldsymbol{\omega}^* = \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{x}_i.$$

3. **选择支持向量** 从 $\boldsymbol{\alpha}^*$ 中选择一个非零分量 $\alpha_j^* > 0$ 对应的样本索引 j。可通过取 $\boldsymbol{\alpha}^*$ 中最大值的下标实现：

$$j = \text{argmax}_i \, \alpha_i^*.$$

4. **计算偏置项 b^*** 偏置项 b^* 的计算公式为：

$$b^* = y_j - \boldsymbol{\omega}^{*T} \boldsymbol{x}_j,$$

其中，j 为选取的支持向量的下标。

5. **最终结果输出** 权重向量 $\boldsymbol{\omega}^*$ 和偏置项 b^* 的值分别为：

$$\boldsymbol{\omega}^* = \text{计算得到的权重向量},$$

$$b^* = \text{计算得到的偏置项}.$$

以下是代码实现：

Listing 3: 计算 ω^* 和 b^*

```
threshold = 1e-5
alpha[alpha < threshold] = 0
alpha = alpha.reshape(-1, 1)
y = y.reshape(-1, 1)
omega = np.sum(alpha * y * X, axis=0)
print("Optimal␣*:", omega)
j = np.argmax(alpha)
b = y[j] - np.dot(omega, X[j])
```

```
print("Optimal b*:", b)
```

```
Optimal ω*: [-1.25061236  0.56121514]
Optimal b*: [6.83758471]
```

Figure 7: 计算 ω^* 和 b^*

3.4 绘制散点图与决策边界

画出数据集的散点图，给正样本（$y = +1$）和负样本（$y = -1$）分别标上不同的颜色，再为支持向量（训练数据中 $\alpha_j^* > 0$ 的对应样本）标上不同的颜色，并画出决策边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 0$ 和间隔边界 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = 1$ 与 $\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = -1$。代码实现：

Listing 4: 绘制散点图与决策边界

```
plt.figure(figsize=(8, 6))

support_vectors = X[alpha.flatten() > 1e-5]

plt.scatter(positive_samples[:, 0], positive_samples[:, 1], color="blue", label="Positive Samples (y=+1)", alpha=0.6)
plt.scatter(negative_samples[:, 0], negative_samples[:, 1], color="red", label="Negative Samples (y=-1)", alpha=0.6)

plt.scatter(support_vectors[:, 0], support_vectors[:, 1], edgecolors="green", facecolors="none", linewidths=1.5, label="Support Vectors")

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x_values = np.linspace(x_min, x_max, 500) # 取点

decision_boundary = -(omega[0] * x_values + b) / omega[1]
margin_positive = -(omega[0] * x_values + b - 1) / omega[1]
margin_negative = -(omega[0] * x_values + b + 1) / omega[1]
```

```
17
18    plt.plot(x_values, decision_boundary, "k-", label="Decision␣
          Boundary␣(w*x␣+␣b␣=␣0)")
19    plt.plot(x_values, margin_positive, "k--", label="Margin␣Boundary␣
          (w*x␣+␣b␣=␣+1)")
20    plt.plot(x_values, margin_negative, "k--", label="Margin␣Boundary␣
          (w*x␣+␣b␣=␣-1)")
21
22    plt.legend()
23    plt.title("Scatter␣Plot␣with␣Decision␣and␣Margin␣Boundaries")
24    plt.xlabel("Feature␣1")
25    plt.ylabel("Feature␣2")
26    plt.grid(alpha=0.3)
27    plt.show()
```

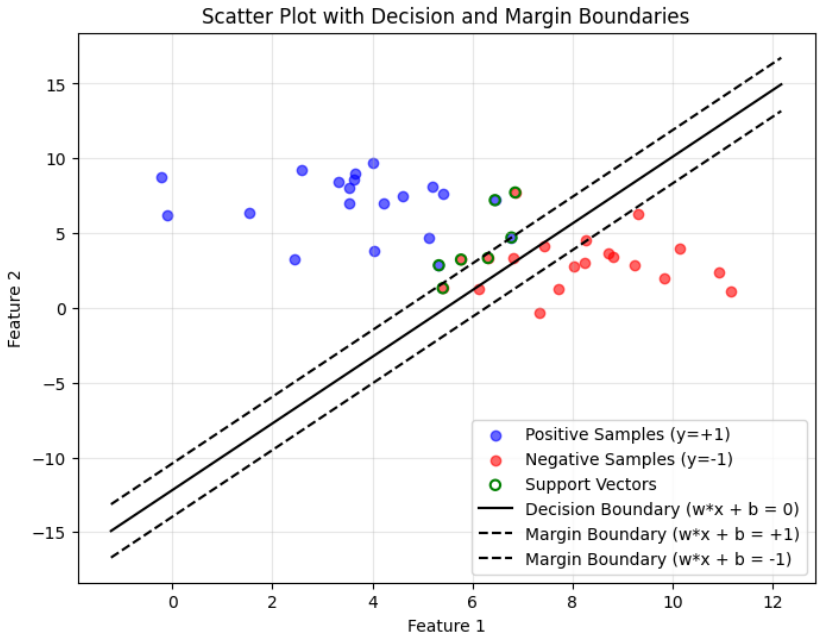

Figure 8: 绘制散点图与决策边界

可以看到，蓝点和红点被分类，有些点在决策边界上，有些点在间隔边界上，这些点的颜色不同，支持向量用绿色标出。支持向量可以用来确定

决策边界和间隔边界。

4 非线性支持向量机与核函数

4.1 数据读入

Raisin Dataset 是一个葡萄干的数据集，总共有 900 个样本，每个样本包含 7 个 (都是连续的) 特征以及 1 个标签，每个标签只有两种可能取值。本次实验已经按照 8: 2 的比例划分成了训练数据集'Raisin_train.csv' 以及测试数据集'Raisin_test.csv'，且每个数据集都已经做了特征归一化处理以及把标签的值替换成了 +1 和-1。

```
train_data = pd.read_csv('Raisin_train.csv')
train_data = train_data.astype(np.double)
print(train_data.dtypes)
```

4.2 对偶问题求解

选择一个核函数 $K(\boldsymbol{x}, \boldsymbol{z})$ 以及参数 C，求解如下对偶问题 (参考课件):

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{m}\alpha_i \\ s.t. \quad & \sum_{i=1}^{m}\alpha_i y_i = 0 \\ & \mathbf{0} \le \boldsymbol{\alpha} \le C \end{aligned}$$

相较于软间隔最大化的优化问题，该优化问题仅需要对矩阵 P 做改动。从以下常用的核函数中选择一个作为该优化问题中的 K (参数自己进行调整): - 线性核: $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T\boldsymbol{z}$ - 多项式核: $K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T\boldsymbol{z} + 1)^p$ - 高斯核: $K(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}\right)$ - 拉普拉斯核: $K(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|}{\sigma}\right)$ - Sigmoid 核: $K(\boldsymbol{x}, \boldsymbol{z}) = \tanh(\beta\boldsymbol{x}^T\boldsymbol{z} + \theta)$

则 P 是一个 $m \times m$ 的矩阵，其中 $P_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$。

4.3 选择核函数求解对偶问题

本节展示了使用核方法求解支持向量机（SVM）对偶问题的完整实现流程。代码的主要逻辑分为以下几部分：

Step 1: 构造核矩阵 K 根据输入数据 $\mathbf{X}$ 和标签 $\mathbf{y}$ 计算核矩阵 K。支持的核函数包括线性核、多项式核、高斯核（RBF）、拉普拉斯核和 Sigmoid 核：

- **线性核:** $K = \mathbf{X}\mathbf{X}^\top$
- **多项式核:** $K = (\mathbf{X}\mathbf{X}^\top + 1)^p$
- **高斯核 (RBF):** $K_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$
- **拉普拉斯核:** $K_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_1}{\sigma}\right)$
- **Sigmoid 核:** $K = \tanh(\beta(\mathbf{X}\mathbf{X}^\top) + \theta)$

核矩阵通过函数 `compute_P` 生成，并根据 $\mathbf{y}$ 构造矩阵 P，公式为：

$$P = \mathbf{y}\mathbf{y}^\top \odot K$$

Step 2: 对偶问题的优化目标 SVM 对偶问题的优化目标为：

$$\max_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^\top P \boldsymbol{\alpha} - \mathbf{1}^\top \boldsymbol{\alpha}$$

约束条件为：

$$0 \le \alpha_i \le C, \quad \forall i$$
$$\mathbf{y}^\top \boldsymbol{\alpha} = 0$$

Step 3: 转换为 `cvxopt` 格式并求解 使用 `cvxopt` 库求解该二次优化问题，具体转换格式如下：

- 优化变量 $\boldsymbol{\alpha}$ 的系数矩阵为 P
- 常数项向量为 $\mathbf{q} = -\mathbf{1}$

- 不等式约束矩阵为：

$$\mathbf{G} = \begin{bmatrix} -\mathbf{I} \\ \mathbf{I} \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} \mathbf{0} \\ C \cdot \mathbf{1} \end{bmatrix}$$

- 等式约束为：

$$\mathbf{A} = \mathbf{y}^\top, \quad b = 0$$

Step 4: 求解对偶问题 调用 `cvxopt.solvers.qp` 求解优化问题，得到优化变量 $\boldsymbol{\alpha}$ 的值。

Step 5: 实现与运行 最后，根据输入数据 $\mathbf{X}_{\text{train}}$ 和标签 $\mathbf{y}_{\text{train}}$，设置核函数类型与参数（如 $C = 1.0$ 和 $\sigma = 1.0$），并调用上述步骤完成对 SVM 对偶问题的求解。

我选择的是高斯核函数，和其他核函数一样，高斯核函数的参数 σ 也需要调整，这里我选择了 $\sigma = 1.0$，并且设置了惩罚系数 $C = 1.0$。代码如下：

```
def compute_P(X, y, kernel='rbf', kwargs):
m = X.shape[0]
y = y.reshape(-1, 1) # 确保 y 是列向量
    kernel == 'rbf':
    sigma = kwargs.get('sigma', 1.0)
    pairwise_dists = cdist(X, X, 'sqeuclidean') # 欧氏距离的平方
    K = np.exp(-pairwise_dists / (2 * sigma 2)) # 高斯核
P = y * y.T * K# 构造 P 矩阵
return P
def solve_dual_problem(P, y, C):
    m = P.shape[0]
    # 转换为 cvxopt 格式
    P = matrix(P)
    q = matrix(-np.ones((m, 1)))
    G = matrix(np.vstack((-np.eye(m), np.eye(m))))
    h = matrix(np.hstack((np.zeros(m), C * np.ones(m))))
    A = matrix(y.reshape(1, -1))
    b = matrix(0.0)
```

```
19
20          sol = solvers.qp(P, q, G, h, A, b)
21          alpha = np.array(sol['x']).flatten()
22          return alpha
23  X_train = train_data.iloc[:, :-1].to_numpy() # 特征
24  y_train = train_data.iloc[:, -1].to_numpy() # 标签
25
26  # 转换标签为列向量
27  y_train = y_train.astype(np.double)
28
29  # 参数设置
30  C = 1.0
31  sigma = 1.0
32
33  # 构造 P
34  P = compute_P(X_train, y_train, kernel='rbf', sigma=sigma)
35
36  # 求解对偶问题
37  alpha = solve_dual_problem(P, y_train, C)
38
39  # 输出结果
40  print("Optimal alpha values:", alpha)
```

```
     pcost       dcost       gap    pres   dres
 0: -3.3326e+02 -2.0029e+03  1e+04  3e+00  7e-15
 1: -2.1611e+02 -1.3588e+03  2e+03  2e-01  5e-15
 2: -2.1993e+02 -3.7171e+02  2e+02  1e-02  5e-15
 3: -2.6262e+02 -3.1175e+02  5e+01  3e-03  4e-15
 4: -2.7392e+02 -2.9895e+02  3e+01  1e-03  4e-15
 5: -2.7885e+02 -2.9310e+02  1e+01  8e-04  4e-15
 6: -2.8149e+02 -2.9005e+02  9e+00  4e-04  4e-15
 7: -2.8318e+02 -2.8801e+02  5e+00  2e-04  4e-15
 8: -2.8448e+02 -2.8647e+02  2e+00  6e-05  5e-15
 9: -2.8509e+02 -2.8579e+02  7e-01  2e-05  5e-15
10: -2.8529e+02 -2.8556e+02  3e-01  4e-06  5e-15
11: -2.8540e+02 -2.8544e+02  4e-02  5e-07  5e-15
12: -2.8542e+02 -2.8542e+02  8e-04  4e-09  6e-15
13: -2.8542e+02 -2.8542e+02  8e-06  4e-11  6e-15
Optimal solution found.
Optimal alpha values: [6.21883282e-09 1.48830179e-08 9.99999992e-
01 1.13826654e-07
```

Figure 9: 对偶问题求解结果

4.4 b 的计算

1. **寻找支持向量：**支持向量满足条件 $0 < \alpha_i < C$，其中 α_i 是拉格朗日乘子，C 是松弛变量的上限。

$$\text{support\_indices} = \{i \mid \epsilon < \alpha_i < C\}$$

其中 $\epsilon = 10^{-5}$ 用于数值精度控制。

如果未找到支持向量，则报错：

Error: No support vectors found. Check α and C values.

2. **选择支持向量的一个下标：**选择第一个支持向量 j 的下标：

$$j = \text{support\_indices}[0]$$

对应的支持向量为：

$$\boldsymbol{x}_j,\, y_j$$

3. **核函数计算** $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$**：**根据核函数类型计算核矩阵 K：

- **线性核（Linear Kernel）**

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j^T$$

- **多项式核（Polynomial Kernel）**

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j^T + 1)^p$$

其中 p 为多项式阶数。

- **高斯核（RBF Kernel）**

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$$

其中 σ 为核函数的参数。

- **拉普拉斯核（Laplacian Kernel）**

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1}{\sigma}\right)$$

- Sigmoid Kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \cdot (\boldsymbol{x}_i \cdot \boldsymbol{x}_j^T) + \theta)$$

其中 β 和 θ 是核函数的超参数

- 如果核类型未知，则报错：

Error: Unknown kernel type.

4. **计算** b^*：使用公式：

$$b^* = y_j - \sum_{i=1}^{m} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

其中 m 是样本数量，$K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ 是核函数计算结果。

在这里，我选择了高斯核进行计算，惩罚参数和核函数参数与上面一致。代码如下：

```
def compute_b_star(X, y, alpha, kernel='rbf', kwargs):

# 找到支持向量：满足 0 < alpha_i < C
C = kwargs.get('C', 1.0)
support_indices = np.where((alpha > 1e-5) & (alpha < C))[0] # 支持向量下标

# 如果没有支持向量，返回错误提示
if len(support_indices) == 0:
    raise ValueError("No support vectors found. Check alpha and C values.")

# 使用第一个支持向量的下标
```

```
    j = support_indices[0]
    x_j, y_j = X[j], y[j]
    m = X.shape[0]
    kernel == 'rbf':
    sigma = kwargs.get('sigma', 1.0)
    pairwise_dists = np.linalg.norm(X - x_j, axis=1) 2
    K = np.exp(-pairwise_dists / (2 * sigma 2))
    # 计算 b*
    b_star = y_j - np.sum(alpha * y * K)
    return b_star

# 参数设置
C = 1.0
sigma = 1.0

# 计算 b*
b_star = compute_b_star(X_train, y_train, alpha, kernel='rbf', sigma=
    sigma, C=C)

print(f"Optimal b*: {b_star}")
```

4.5 预测准确率计算

给定测试数据集 Raisin_test.csv，使用分类决策函数对其进行预测，并计算预测准确率。具体步骤如下：

1. **读入测试数据集：** 从文件 Raisin_test.csv 中读取测试数据集，记为 $\boldsymbol{x}$。

2. **分类决策函数：** 对于每个测试样本，使用以下分类决策函数 $f(\boldsymbol{x})$ 进行预测：

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right)$$

其中：

- m：训练集样本数量；

- α_i^*：训练集中的拉格朗日乘子；
- y_i：训练集样本的标签；
- $\boldsymbol{x}_i$：训练集样本的特征；
- $\boldsymbol{x}$：测试集样本的特征；
- $K(\boldsymbol{x}_i, \boldsymbol{x})$：训练样本和测试样本之间的核函数；
- b^*：通过支持向量计算得到的偏置。

3. **计算预测准确率：**将模型预测结果与测试集的真实标签 y_{test} 进行比较，计算预测准确率：

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of test samples}} \times 100\%$$

4. **输出预测准确率：**将最终的预测准确率以百分比形式输出。

重要代码如下：

```
def predict_and_evaluate(X_train, y_train, alpha, b_star, X_test,
    y_test, kernel='rbf', kwargs):

m_train = X_train.shape[0]
m_test = X_test.shape[0]

elif kernel == 'rbf':
    sigma = kwargs.get('sigma', 1.0)
    pairwise_dists = cdist(X_train, X_test, 'sqeuclidean') # 欧氏距
        离的平方
    K = np.exp(-pairwise_dists / (2 * sigma 2))

# 计算分类决策函数 f(x) = sign(Σ _i * y_i * K(x_i, x) + b*)
f_x = np.dot(alpha * y_train, K) + b_star
y_pred = np.sign(f_x)

# 计算准确率
```

```
    accuracy = np.mean(y_pred == y_test)
    return accuracy

accuracy = predict_and_evaluate(
    X_train=X_train,
    y_train=y_train,
    alpha=alpha,
    b_star=b_star,
    X_test=X_test,
    y_test=y_test,
    kernel='rbf', # 使用高斯核
    sigma=1.0     # 高斯核参数
)

print(f"Prediction accuracy: {accuracy * 100:.2f}%")
```

Figure 10: 预测准确率

最后的正确率高达 87.22%，说明高斯核函数对于这个数据集是有效的。

5 实验总结

这次试验主要是对支持向量机的理论知识进行了实践，通过对线性支持向量机和非线性支持向量机的实现，我对支持向量机的工作原理有了更深入的理解。通过实验，我学会了如何使用 Python 的 cvxopt 库来求解二次规划问题，以及如何使用核函数来处理非线性分类问题。在实验中，我还学会了如何绘制支持向量机的决策边界和间隔边界，以及如何计算支持向量机的预测准确率。总的来说，这次实验让我对支持向量机的原理和实现有了更深入的认识，也提高了我的编程能力和数据分析能力。

硬间隔最大化就是通过最大化间隔来找到最优的决策边界，而软间隔最大化则是通过引入松弛变量来允许一些样本点出现在间隔边界和决策边界之间。通过引入惩罚系数 C，可以控制间隔的松弛程度，C 越大，间隔越小，C 越小，间隔越大。在实验中，我通过调整 C 的值，观察了支持向量机的决策边界和间隔边界的变化。

这些是线性支持向量机，而非线形支持向量机则是通过核函数来将数据映　到高维空间，从而使数据在高维空间中线性可分。也就是映射到一个平面，然后在平面上找到最优的决策边界。实验中我选择了高斯核函数，通过调整核函数的参数 σ，观察了支持向量机的预测准确率的变化。通过实验，我对支持向量机的工作原理和实现有了更深入的理解。