机器学习实验报告

题　　目:	支持向量机
学生姓名:	张芮熙
学　　号:	22354188
指导教师:	马倩
专业班级:	22 智科 3 班

2021 年5 月

本科生院制

目录

1. 实验目的、思路：

针对给出的数据集dataset，将正类和负类的点可视化表示，并通过支持向量机将正类和负类的点分隔开来，找出最合适（鲁棒性最好）的分割界面。

核心代码实现与测试结果

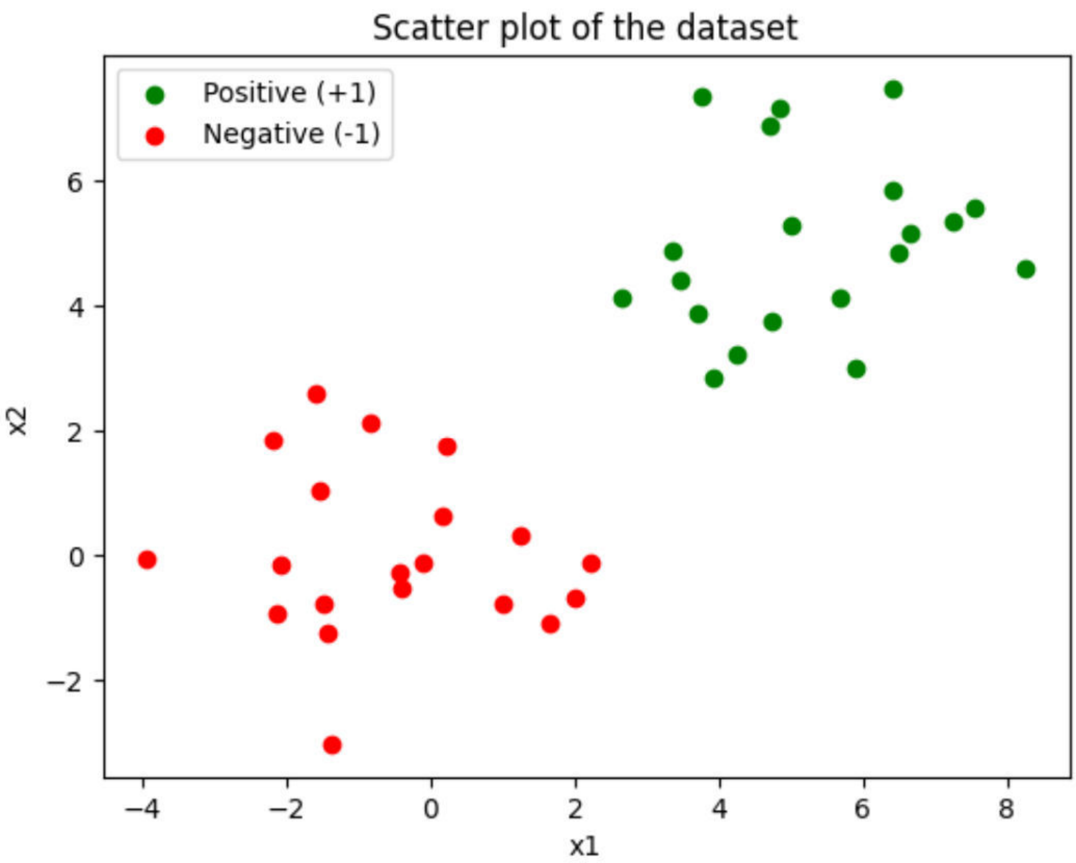
（1） 准备工作，导入数据集dataset，将dataset上的点转换成double形，并将正类和负类用不同的颜色打印 来。

1) 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）分别标上不同的颜色。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cvxopt
from cvxopt import matrix
from cvxopt import solvers
# 1) 读入数据集
data = pd.read_csv('dataset1.csv')
X = data[['x1', 'x2']].values.astype(np.double)
y = data['y'].values.astype(np.double)
# 画出数据集的散点图
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', label='Positive (+1)')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='r', label='Negative (-1)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatter plot of the dataset')
plt.legend()
plt.show()
```

运行结果：

可见，正类点用绿色表示，负类点用红色表示，二者之间大致可以由一条直线分割开来。



（2）求解对偶问题

这个优化问题是一个二次规划问题。

- 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$m \times 1$的所有值都为-1的列向量，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

把上述参数送入求解器solvers.qp()中即可得到最优解α^*。

附：P矩阵的一个计算方法：设$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{m1} & x_{m2} \end{bmatrix}$, $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$,

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

则$P = X'X'^T$。

按照提示，分别表示出所求的矩阵，并把这些矩阵带入求解器中，即可求得最优解。

```
# 2) 求解对偶问题
m = len(y)
Xy = X * y[:, np.newaxis]
P = matrix(Xy @ Xy.T)
q = matrix(-np.ones((m, 1)))
G = matrix(-np.eye(m))
h = matrix(np.zeros((m, 1)))
A = matrix(y.reshape(1, -1))
b = matrix(0.0)

# 调用求解器
sol = solvers.qp(P, q, G, h, A, b)
alpha = np.array(sol['x'])
```

运行结果：

```
     pcost       dcost       gap    pres   dres
 0: -5.2553e+00 -9.0147e+00  1e+02  1e+01  2e+00
 1: -4.9265e+00 -1.9551e+00  2e+01  2e+00  3e-01
 2: -1.5759e-01 -3.1831e-01  7e-01  4e-02  7e-03
 3: -1.3147e-01 -1.9706e-01  7e-02  2e-17  8e-16
 4: -1.7378e-01 -1.8099e-01  7e-03  1e-16  9e-16
 5: -1.7979e-01 -1.8010e-01  3e-04  5e-17  1e-15
 6: -1.8003e-01 -1.8003e-01  3e-06  4e-17  7e-16
 7: -1.8003e-01 -1.8003e-01  3e-08  4e-17  8e-16
Optimal solution found.
```

注意，P矩阵的计算是X矩阵*Y矩阵求得，可以直接调用*乘函数。

3. 计算w和b,得出分割直线的斜率和截距。

3) 求出$\boldsymbol{\omega}^* = \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{x}_i$和$b^* = y_j - \boldsymbol{\omega}^{*T} \boldsymbol{x}_j$, 其中$j$为$\alpha^*$中的一个正分量$\alpha_j^* > 0$的下标。注意：由于求解器求出来的是一个近似解，所以α^*中很多实际上为0的分量会略大于0，这时候可以设置一个阈值把非常靠近0的那些分量筛去，再从剩下的分量中选取一个正分量来计算b^*,或者也可以直接取α^*中最大的分量来计算b^*。

2]:

```
# 3) 计算 ω 和 b
w = np.sum(alpha * y[:, None] * X, axis=0)
b = y[np.argmax(alpha)] - np.dot(X[np.argmax(alpha)], w)
print(w)
print(b)
```

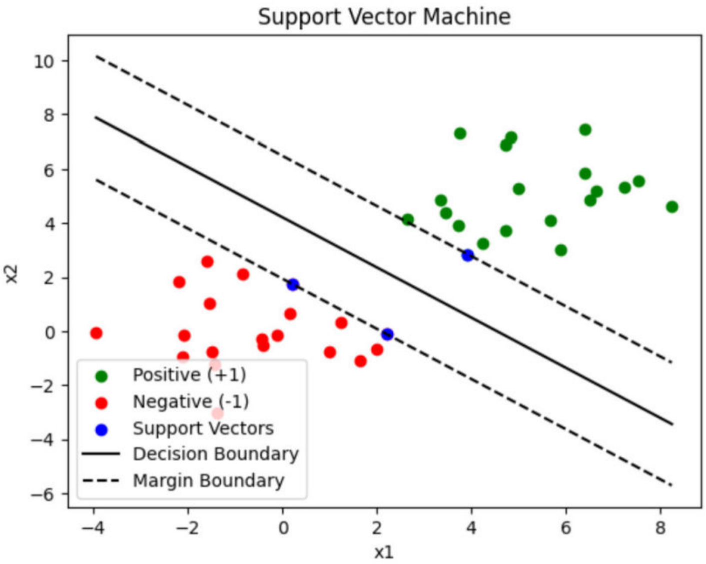
```
[0.40821121 0.43979838]
-1.8556819838183283
```

4. 数据可视化处理，将所有样本点表示在图像上，再将分割直线展示在图像上，观察分割线是否正确。

4) 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$\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = -1$。

```
# 4) 画出数据集散点图和决策边界
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', label='Positive (+1)')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='r', label='Negative (-1)')
plt.scatter(X[alpha.flatten() > 1e-4, 0], X[alpha.flatten() > 1e-4, 1], c='b', label='Support Vectors')
x1 = np.linspace(np.min(X[:,0]), np.max(X[:,0]), 100)
x2 = (-w[0]*x1 - b) / w[1]  # 决策边界
x2_margin1 = (-w[0]*x1 - b + 1) / w[1]  # 间隔边界
x2_margin2 = (-w[0]*x1 - b - 1) / w[1]
plt.plot(x1, x2, 'k', label='Decision Boundary')
plt.plot(x1, x2_margin1, 'k--', label='Margin Boundary')
plt.plot(x1, x2_margin2, 'k--')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Support Vector Machine')
plt.legend()
plt.show()
```

运行结果：



可见，绝大多数的正负类样本点都被直线分割开来，存在几个点距离分割曲线较近，用蓝色表示（支持向量）

2. 线性支持向量机与软间隔最大化

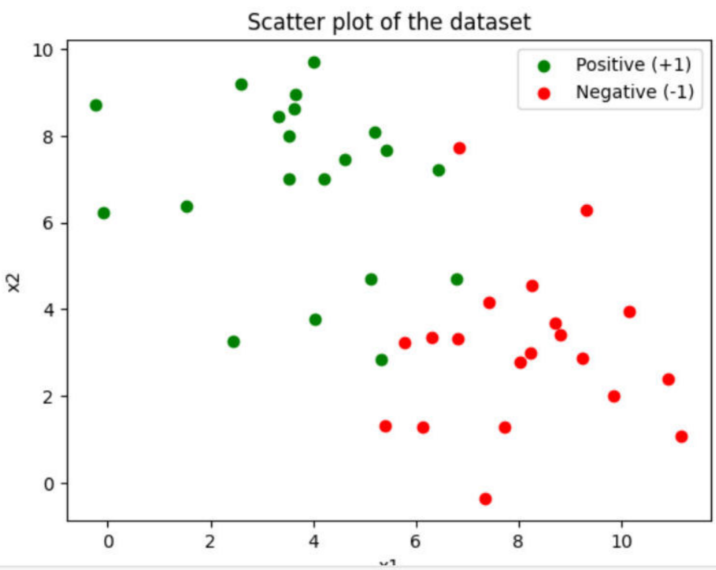
和线性可分支持向量机与硬间隔最大化类似，区别在于软间隔删去了一些不符合的条件（比如样本值过于接近0或者过于接近C）以及函数计算时的矩阵要求略有不同。

（1）数据的预处理和可视化

代码及结果：

1) 读入数据集'dataset2.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）分别标上不同的颜色。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cvxopt
from cvxopt import matrix
from cvxopt import solvers
# 1) 读入数据集
data = pd.read_csv('dataset2.csv')
X = data[['x1', 'x2']].values.astype(np.double)
y = data['y'].values.astype(np.double)
# 画出数据集的散点图
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', label='Positive (+1)')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='r', label='Negative (-1)')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatter plot of the dataset')
plt.legend()
plt.show()
```



将dataset2中的数据可视化，可见相对于dataset1的数据来说，数据的可分度略差。

（3） 求解对偶问题

2) 求解如下对偶问题（参考课件）：

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

这个优化问题是一个二次规划问题。

- $\boldsymbol{P}$是一个$m\times m$的矩阵，其中$\boldsymbol{P}_{ij} = y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j$,
- $\boldsymbol{q}$是一个$m\times 1$的所有值都为-1的列向量，即$\boldsymbol{q} := \begin{bmatrix}-1 & -1 & \cdots & -1\end{bmatrix}^T$,
- $\boldsymbol{G} := \begin{bmatrix}-1 & 0 & \cdots & 0\\ 0 & -1 & \cdots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & 0 & -1\end{bmatrix}_{m\times m} = -\boldsymbol{I}$, $\boldsymbol{I}$为单位矩阵,
- $\boldsymbol{h}$是一个$m\times 1$的零向量, 即$\boldsymbol{h} := \begin{bmatrix}0 & 0 & \cdots & 0\end{bmatrix}^T$,
- $\boldsymbol{A} := \begin{bmatrix}y_1 & y_2 & \cdots & y_m\end{bmatrix}^T$,
- $\boldsymbol{b} := \begin{bmatrix}0\end{bmatrix}$, 一个标量

把上述参数送入求解器solvers.qp()中即可得到最优解α^*。

附：$\boldsymbol{P}$矩阵的一个计算方法：设$X = \begin{bmatrix}x_{11} & x_{12}\\ x_{21} & x_{22}\\ \vdots & \vdots\\ x_{m1} & x_{m2}\end{bmatrix}, Y = \begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_m\end{bmatrix}$,

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

则$\boldsymbol{P} = X'X'^T$。

和硬间隔最大化很相似，就是矩阵略有不同。

代码及结果：

```
# 2) 选择参数C并求解对偶问题
C = 1.0
m = len(y)
Xy = X * y[:, np.newaxis]
P = matrix(Xy @ Xy.T)
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)

# 调用求解器
sol = solvers.qp(P, q, G, h, A, b)
alpha = np.array(sol['x'])
```

```
     pcost       dcost       gap    pres   dres
 0: -1.1816e+01 -8.2073e+01  4e+02  2e+00  3e-14
 1: -7.7195e+00 -4.7758e+01  7e+01  3e-01  3e-14
 2: -5.7411e+00 -1.4569e+01  1e+01  5e-02  2e-14
 3: -5.6219e+00 -6.8029e+00  2e+00  4e-03  3e-14
 4: -5.8632e+00 -6.2495e+00  5e-01  9e-04  2e-14
 5: -5.9709e+00 -6.0062e+00  4e-02  4e-05  2e-14
 6: -5.9845e+00 -5.9849e+00  4e-04  4e-07  2e-14
 7: -5.9847e+00 -5.9847e+00  4e-06  4e-09  2e-14
Optimal solution found.
```

（4） 求解w,b

3) 求出$\boldsymbol{\omega}^* = \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{x}_i$和$b^* = y_j - \boldsymbol{\omega}^{*T} \boldsymbol{x}_j$，其中$j$为$\alpha^*$中的一个正分量$0 < \alpha_j^* < C$的下标。与硬间隔优化问题同理，应该筛掉非常接近0和非常接近C的分量。

```
# 3) 计算 ω 和 b
w = np.sum(alpha * y[:, None] * X, axis=0)

# 找到支持向量的索引并计算 b
sv_idx = (alpha > 1e-4).flatten()
b = np.mean(y[sv_idx] - np.dot(X[sv_idx], w))
```

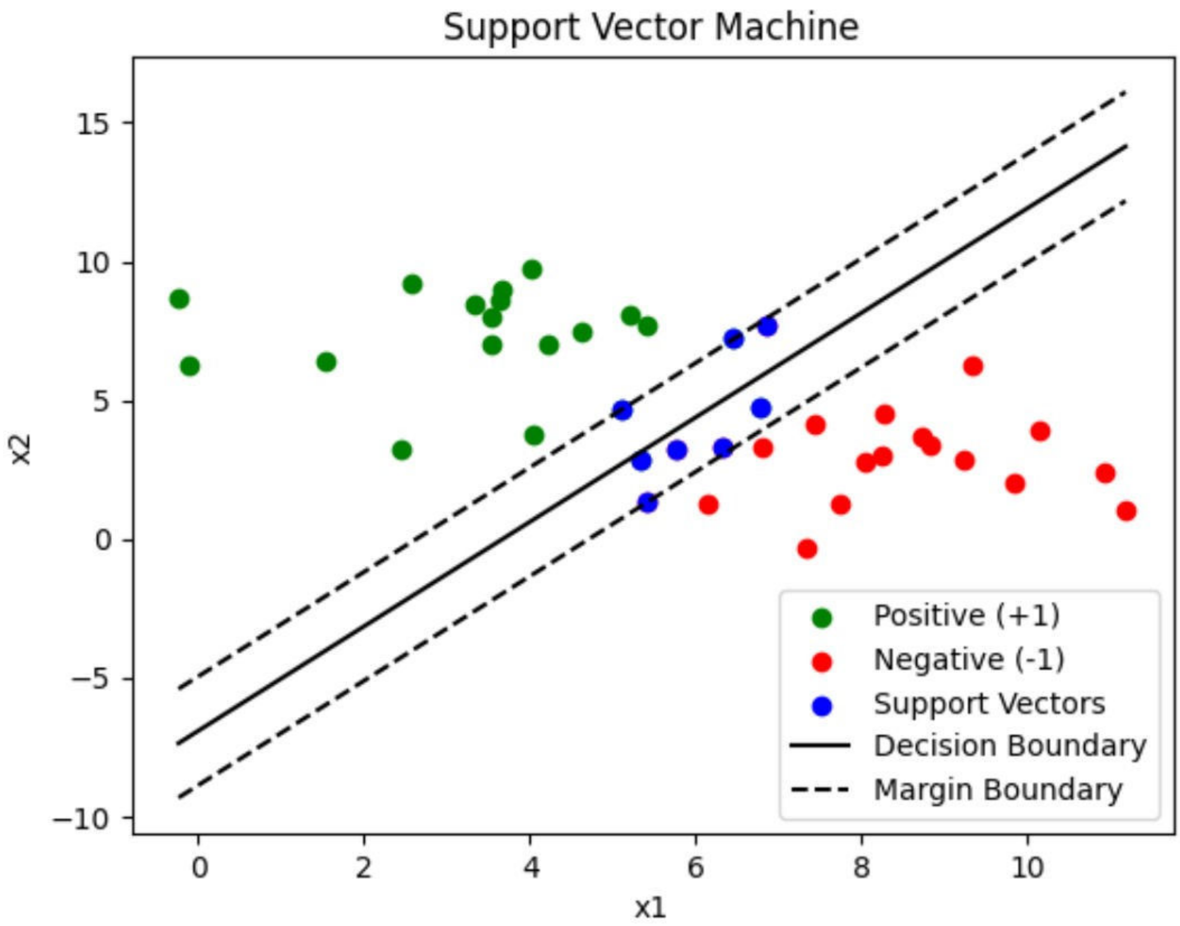
注意：筛除接近0和接近C(1)的分量是利用bool索引来删除的。

（5） 绘制散点图和直线，决策边界

4) 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$\boldsymbol{\omega}^{*T}\boldsymbol{x} + b = -1$。

```python
# 4) 画出数据集散点图和决策边界
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', label='Positive (+1)')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='r', label='Negative (-1)')
plt.scatter(X[sv_idx, 0], X[sv_idx, 1], c='b', label='Support Vectors')
x1 = np.linspace(np.min(X[:,0]), np.max(X[:,0]), 100)
x2 = (-w[0]*x1 - b) / w[1]  # Decision boundary
x2_margin1 = (-w[0]*x1 - b + 1) / w[1]  # Margin boundary
x2_margin2 = (-w[0]*x1 - b - 1) / w[1]
plt.plot(x1, x2, 'k', label='Decision Boundary')
plt.plot(x1, x2_margin1, 'k--', label='Margin Boundary')
plt.plot(x1, x2_margin2, 'k--')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Support Vector Machine')
plt.legend()
plt.show()
```

运行结果：



3. 非线性支持向量机与核函数

思路：这道题是软间隔的plus版，在第二题的基础上需要通过核函数来对P矩阵做变换，对核函数的选择就成为了重点，这里我选择高斯核，一是高斯核是通用核函数，二是高斯核映射到无穷维空间：高斯核函数实际上是将数据映射到了无限维的特征空间，这样可以更好地处理非线性可分的数据集，提高了模型的拟合能力，三是高斯核参数灵活性（便于调参）：高斯核函数有一个参数用来控制高斯函数的宽度，通过合理调整这个参数可以更好地适应不同数据的分布特点。这种参数的灵活性使得高斯核能够应对多样化的数据集。通过预览数据，发现数据量不是很大，使用高斯核并不会造成过大的计算负担，因此就选择高斯核。

（1） 数据预处理

1) 读入训练数据集'Raisin_train.csv',把数据类型都转换成np.double类型。

```
# ---- Your code here ----
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# 1）读入训练数据集'Raisin_train.csv'，将数据类型转换为np.double类型
train_data = pd.read_csv('Raisin_train.csv')
X_train = train_data.iloc[:, :-1].values.astype(np.double)
y_train = train_data.iloc[:, -1].values

# 特征归一化处理
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
```

（2） 高斯核（这里的调参对后面的准确率预测至关重要，过大会导致过拟合）

2) 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）：

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)-\sum_{i=1}^{m}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{m}\alpha_i y_i = 0$$

$$\boldsymbol{0}\le\boldsymbol{\alpha}\le C$$

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$\boldsymbol{P}$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：$K(\boldsymbol{x},\boldsymbol{z})=exp(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2})$
- 拉普拉斯核：$K(\boldsymbol{x},\boldsymbol{z})=exp(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|}{\sigma})$
- Sigmoid核：$K(\boldsymbol{x},\boldsymbol{z})=tanh(\beta\boldsymbol{x}^T\boldsymbol{z}+\theta)$

则$\boldsymbol{P}$是一个$m\times m$的矩阵，其中$P_{ij}=y_iy_jK(\boldsymbol{x_i},\boldsymbol{x_j})$。

```
# ---- Your code here ----
# 2) 选择核函数 K(x, z) 和参数C, 求解对偶问题
# 选择高斯核作为核函数
svm = SVC(kernel='rbf', C=1.0, gamma=0.7)  # gamma是高斯核的一个参数，可以自己调整
svm.fit(X_train, y_train)
alpha = svm.dual_coef_
support_vectors = svm.support_vectors_
y_sv = svm.dual_coef_.reshape(-1, 1) * y_train[svm.support_]
```

（3） 求解b*（按照公式往里套）

3) 求出$b^*=y_j-\sum_{i=1}^{m}\alpha_i^*y_iK(\boldsymbol{x_i},\boldsymbol{x_j})$, 其中$j$为$\alpha^*$中的一个正分量$0<\alpha_j^*<C$的下标。

```
# ---- Your code here ----
# 3) 求解 b*
b_values = []
for j in range(len(support_vectors)):
    b = y_sv[j]
    for i in range(len(support_vectors)):
        b -= alpha[0][i] * y_sv[i] * np.exp(-0.7 * np.linalg.norm(support_vectors[i] - support_vectors[j]) **
    b_values.append(b)
b_star = np.mean(b_values)
```

（4） 把测试集带入进模型，求解准确率

4) 读入测试数据集'Raisin_test.csv',用分类决策函数$f(\boldsymbol{x}) = sign(\sum_{i=1}^{m} \alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b^*)$（注意这里的$m, \alpha_i^*, y_i, \boldsymbol{x}_i$是训练集的, $\boldsymbol{x}$是测试集的）进行预测，输出预测准确率。

```
# ---- Your code here ----
# 4) 读入测试数据集'Raisin_test.csv'，进行预测
test_data = pd.read_csv('Raisin_test.csv')
X_test = test_data.iloc[:, :-1].values.astype(np.double)
y_test = test_data.iloc[:, -1].values
X_test = scaler.transform(X_test)  # 使用相同的缩放参数

# 使用分类决策函数进行预测
y_pred = svm.predict(X_test)

# 计算预测准确率
accuracy = accuracy_score(y_test, y_pred)
print("预测准确率：", accuracy)
```

```
预测准确率： 0.8888888888888888
```

求出准确率的大致范围在0.8~0.9之间波动。

心得体会

调参的时候在网上查找参数应该是在0.5左右，参数过小会使得过拟合，样本数据的细微波动也会造成模型的变动（过于敏感），参数过大可能会产生欠拟合，没有了泛化能力，性能都不是很好，这里最优参数应该是0.7，在这个状态下系统的拟合效果最佳。