安徽大学《机器学习》实验报告 3

学号：WA2214014　　专业：人工智能　　姓名：杨跃浙

实验日期：24.12.09　　教师签字：_____　　成绩：_____

[实验名称] 支持向量机实验

[实验目的]

1．熟悉和掌握支持向量机

2．熟悉和掌握核函数处理非线性性问题

3．了解和掌握第三方机器学习库 Scikit-learn 中的模型调用

[实验要求]

1．采用 Python、Matlab 等高级语言进行编程，推荐优先选用 Python 语言

2．本次实验可以直接调用 Scikit-learn、PyTorch 等成熟框架的第三方实现

3．代码可读性强：变量、函数、类等命名可读性强，包含必要的注释

4．提交实验报告要求：

- 命名方式："学号-姓名-Lab-N"（N 为实验课序号）；
- 截止时间：下次实验课前一天晚 23:59；
- 提交方式：智慧安大-网络教育平台-作业

[实验原理]

支持向量机（Support Vector Machine，SVM）是一种监督学习算法，适用于分类和回归任务。其核心思想是找到一个最优的超平面，用于将不同类别的数据进行有效分离。SVM 的基本原理可以从线性可分和非线性可分两种情况进行分析。

1　线性可分支持向量机

在线性可分情况下，SVM 通过优化以下目标函数来求解最优超平面：

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

约束条件为：

$$y_i(w^T x_i + b) \geq 1, \ i = 1,2,...,N$$

其中：

- w 是超平面的法向量；
- b 是偏置；
- $y_i \in \{-1,1\}$ 表示样本类别；
- x_i 表示样本点。

通过拉格朗日对偶理论，可以将问题转化为其对偶形式，采用二次规划算法求解。

2 非线性可分支持向量机

对于非线性可分问题，SVM 通过引入核函数 $K(x_i, xj)$ 将数据映射到高维特征空间，在该空间中实现线性可分。目标函数为：

$$\min \frac{1}{2} \| w \|^2 + C \sum \ i = 1^N \xi_i$$

约束条件为：

$$y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \ \xi_i \geq 0$$

其中：

- $\phi(x_i)$ 为特征映射函数；

- ξ_i 是松弛变量，用于处理少量的误分类样本；
- C 是惩罚系数，用于平衡分类间隔和误分类惩罚。

对偶形式为：

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, xj)$$

约束条件为：

$$\sum i = 1^N \alpha_i y_i = 0,\ 0 \leq \alpha_i \leq C$$

3　核函数

常用的核函数包括：

- 线性核函数：

$$K(x_i, x_j) = x_i^T x_j$$

- 多项式核函数：

$$K(x_i, x_j) = (x_i^T x_j + c)^d$$

- 径向基核函数（RBF 核）：

$$K(x_i, x_j) = \exp\ (-\gamma\ |x_i - x_j|^2)$$

4　决策函数

训练完成后，SVM 的决策函数为：

$$f(x) = \text{sign}\ (\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b)$$

通过上述公式，SVM 可以高效地分类线性和非线性数据。

- 支持向量：位于边界上的点，其决定了超平面的位置。

- 间隔：正负分类样本与超平面之间的最小距离，SVM 目标是最大化此间隔。

[**实验内容**]

(一) iris 数据集支持向量机分类实验

1. 从 iris 数据集中取出[sepal length，sepal width]两个属性作为样本特征，取前两类样本训练支持向量机进行二分类实验。注意**每一类取前 30 个样本作为训练集，剩余的 20 个样本作为测试集**。

   提示：

   - Iris 数据集介绍详见：https://archive.ics.uci.edu/dataset/53/iris
   - Scikit-learn 库中预装了 Iris 数据集，安装库后采用“from sklearn.datasets import load_iris”可以直接读取，参考 https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

2. 借助 matplotlib 画出原始训练数据分布的散点图（x=“sepal length”，y=“sepal width”，点的颜色代表不同类别）

3. 调用 sklearn 的 SVM 模型包，实现分类算法

   结果展示：

   - 用训练的模型对测试数据进行分类，得到测试错误率
   - 使用不同的误差惩罚系数取值（分别为 0.01、0.1、1.0、10 和 100）和核函数（分别为“linear”和“poly”）来组合，运行模型，并比较模型在最后的测试正确率以及可视化测试数据的 SVM 决策间隔。（提示：可调用 sklearn 库中的 DecisionBoundaryDisplay）。

(二) 基于核函数的 SVM 非线性分类实验

1．利用 sklearn 生成非线性数据

```
from sklearn.datasets import make_moons
x, y = make_moons(n_samples=100, noise=0.2, random_state=0)
```

2．借助 matplotlib 画出生成非线性数据的散点图

3．分别选取核函数（“linear”，“poly”，”rbf”），根据 API 手册了解不同核函数对应的超参数，通过调节超参数展示绘制的决策平面。参考 sklearn API sklearn.svm.SVR—scikit—learn

结果展示：

- 展示使用三种不同核函数，在不同超参数下的决策平面

(三) 手写 SVM 模型【选做】

1．尝试手写 SVM 模型，实现鸢尾花数据集分类

[实验代码和结果]

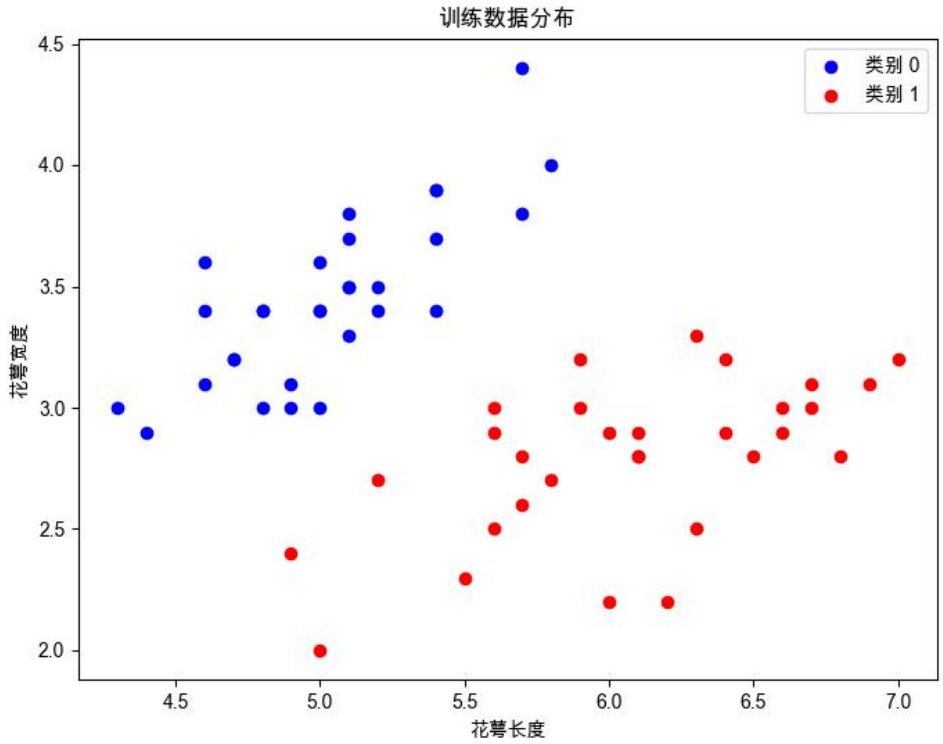
（一） iris 数据集支持向量机分类实验

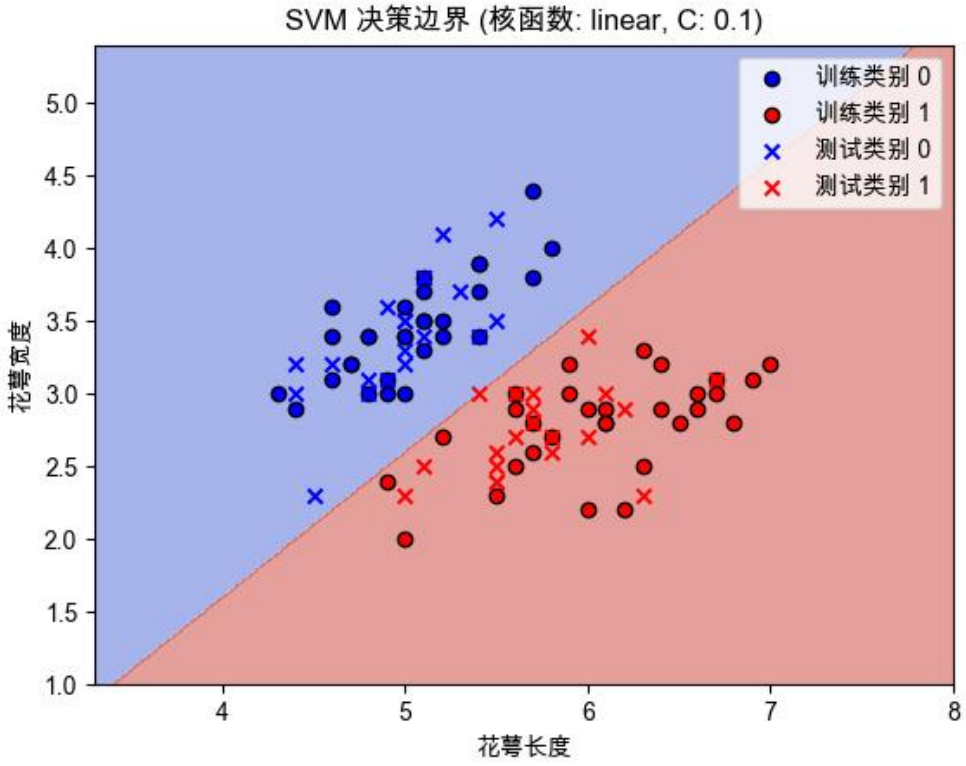
```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.inspection import DecisionBoundaryDisplay
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import pandas as pd
import os
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False

output_dir =
"/Users/youngbean/Documents/Github/Misc-Projects/Machine
Learning/Class3/Output"
```

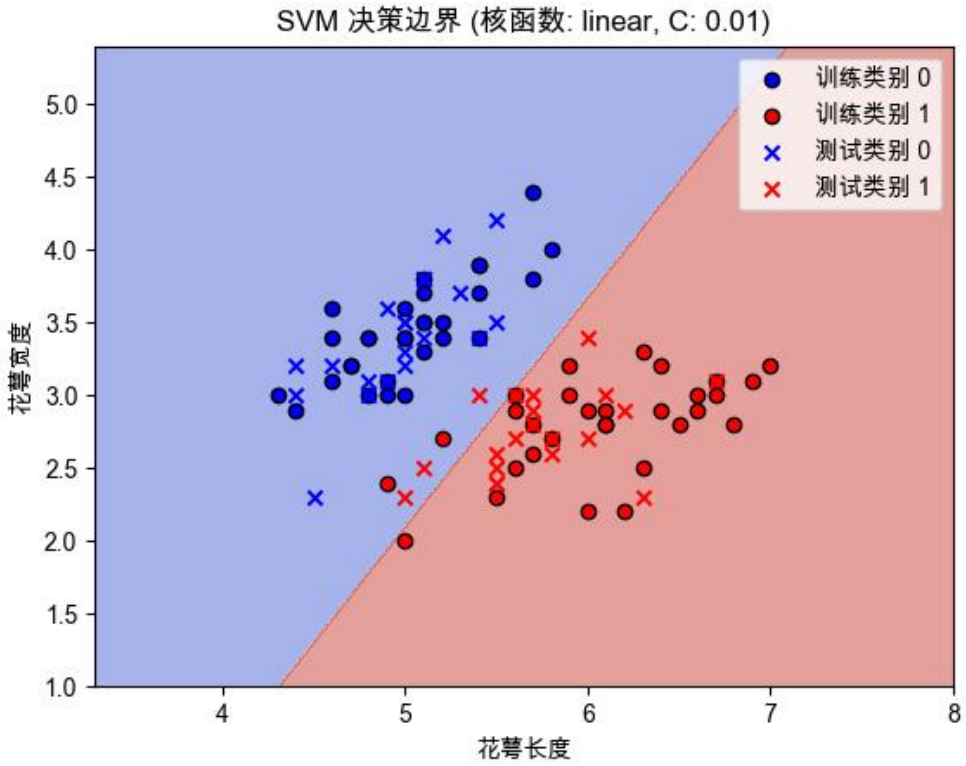
```
# 加载 Iris 数据集
iris = load_iris()
X = iris.data # 特征数据
y = iris.target # 标签数据

# 选择前两类样本，并取前两个特征（花萼长度和花萼宽度）
X = X[y < 2, :2]
y = y[y < 2]

# 将每类样本分为训练集（前 30 个样本）和测试集（后 20 个样本）
X_train, X_test, y_train, y_test = [], [], [], []
for class_label in np.unique(y):
class_indices = np.where(y == class_label)[0]
X_train.append(X[class_indices[:30]])
y_train.append(y[class_indices[:30]])
X_test.append(X[class_indices[30:]])
y_test.append(y[class_indices[30:]])

# 将列表转换为数组
X_train = np.vstack(X_train)
y_train = np.hstack(y_train)
X_test = np.vstack(X_test)
y_test = np.hstack(y_test)


cmap_points = ['blue', 'red']

# 定义不同的惩罚系数和核函数
penalty_values = [0.01, 0.1, 1.0, 10, 100]
kernels = ["linear", "poly"]

# 存储结果的列表
results = []

# 定义绘图函数
def plot_decision_boundary_with_display(model, X_train,
y_train, X_test, y_test, kernel, C, filename):
# 绘制决策边界
plt.figure(figsize=(8, 6))
```

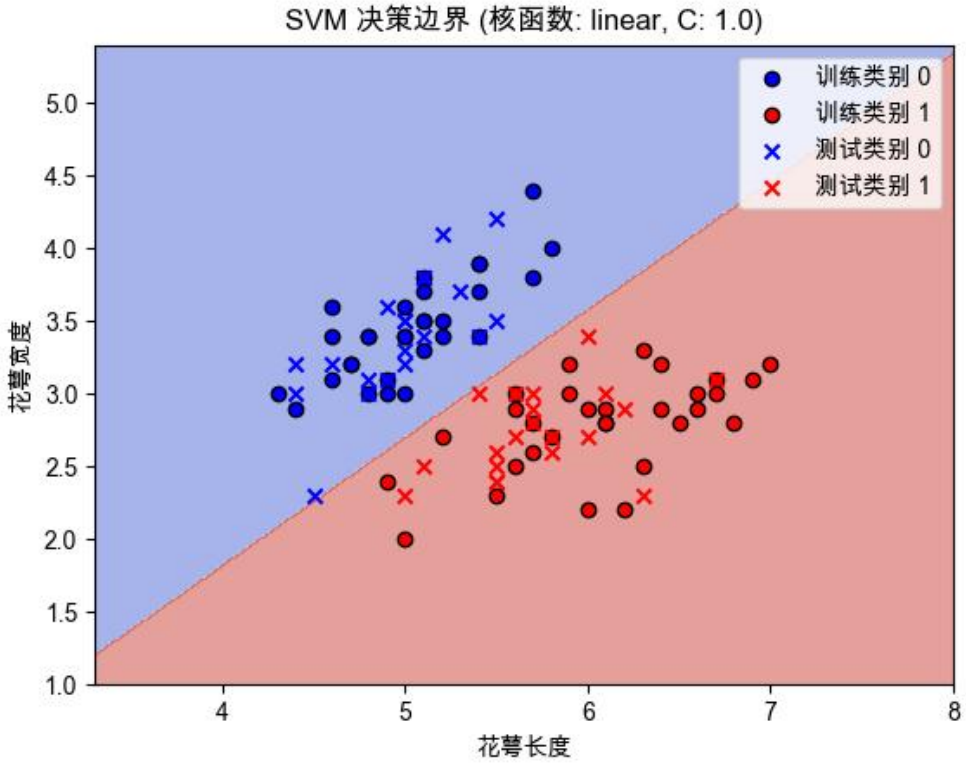
```
DecisionBoundaryDisplay.from_estimator(model, X_train,
response_method="predict",cmap=plt.cm.coolwarm, alpha=0.5,
grid_resolution=500)

# 绘制训练数据点
for class_label, color in zip(np.unique(y_train),
cmap_points):
plt.scatter(X_train[y_train == class_label, 0],
X_train[y_train == class_label, 1],
label=f"训练类别 {class_label}", color=color, marker='o',
edgecolor="k")

# 绘制测试数据点
for class_label, color in zip(np.unique(y_test),
cmap_points):
plt.scatter(X_test[y_test == class_label, 0], X_test[y_test
== class_label, 1],
label=f"测试类别 {class_label}", color=color, marker='x')

# 设置标题和标签
plt.title(f"SVM 决策边界（核函数: {kernel}, C: {C}）")
plt.xlabel("花萼长度")
plt.ylabel("花萼宽度")
plt.legend()

# 保存图像
plt.savefig(filename)
plt.close()

# 遍历不同的核函数和惩罚系数组合
for kernel in kernels:
for C in penalty_values:
# 训练 SVM 模型
svm = SVC(C=C, kernel=kernel)
svm.fit(X_train, y_train)
# 在测试集上进行预测
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append((kernel, C, accuracy))
```

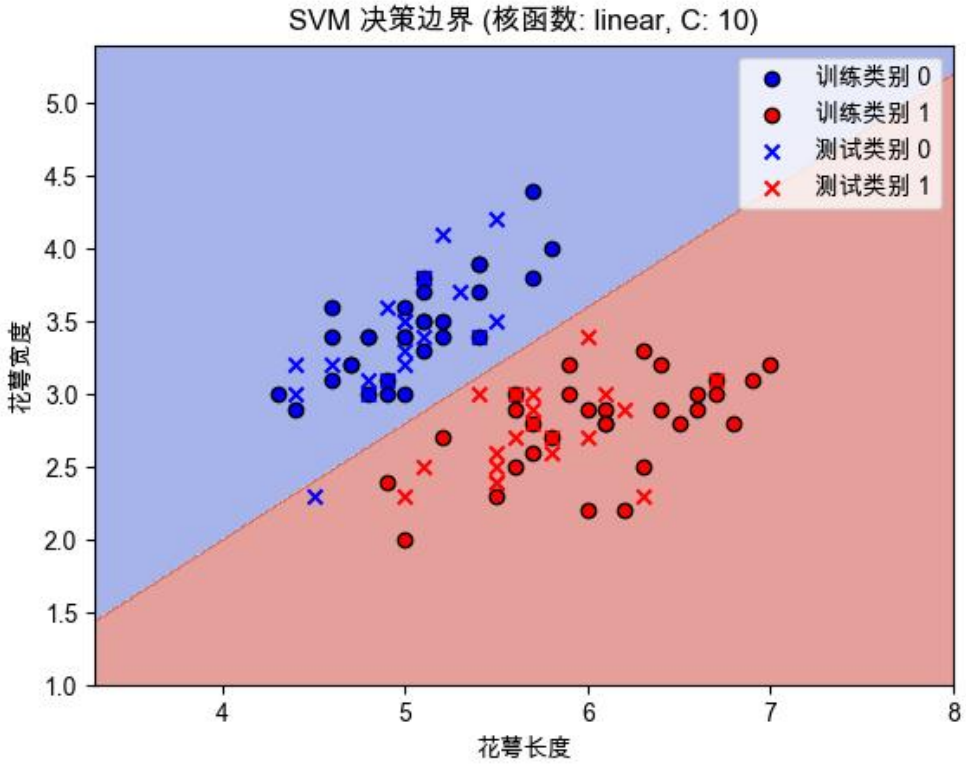
```
# 绘制决策边界并保存
filename = os.path.join(output_dir, f"SVM_决策边界
_{kernel}_C_{C}.png")
plot_decision_boundary_with_display(svm, X_train, y_train,
X_test, y_test, kernel, C, filename)

# 保存实验结果到 CSV
results_df = pd.DataFrame(results, columns=["核函数", "惩罚
系数 C", "测试准确率"])
results_df.to_csv(os.path.join(output_dir, "实验结果.csv"),
index=False)
```

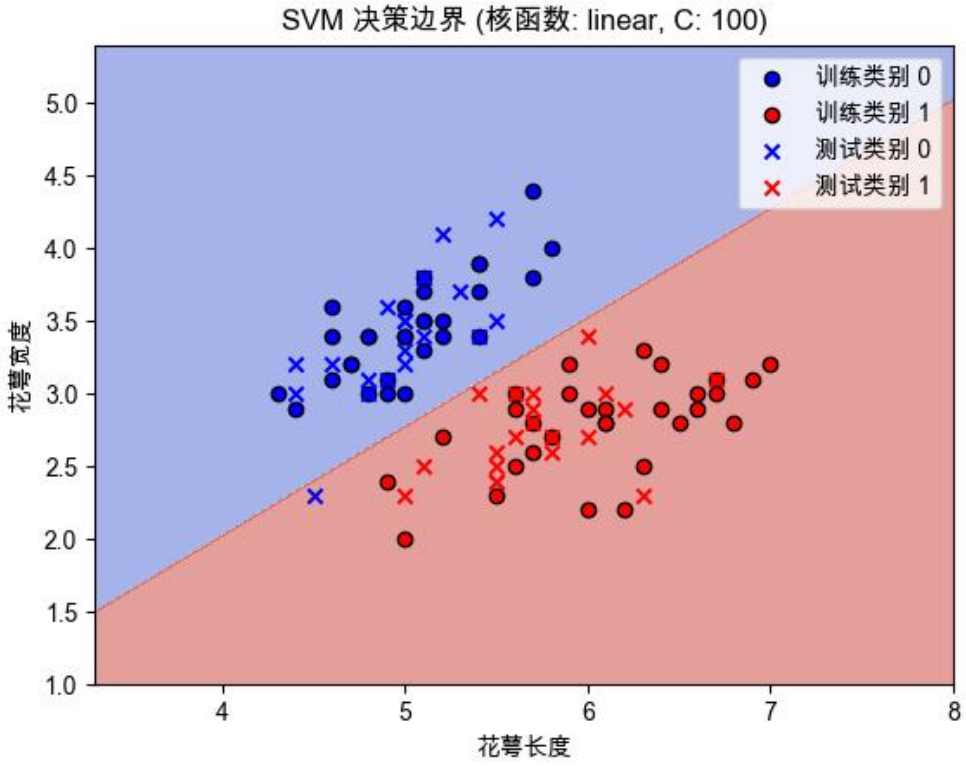


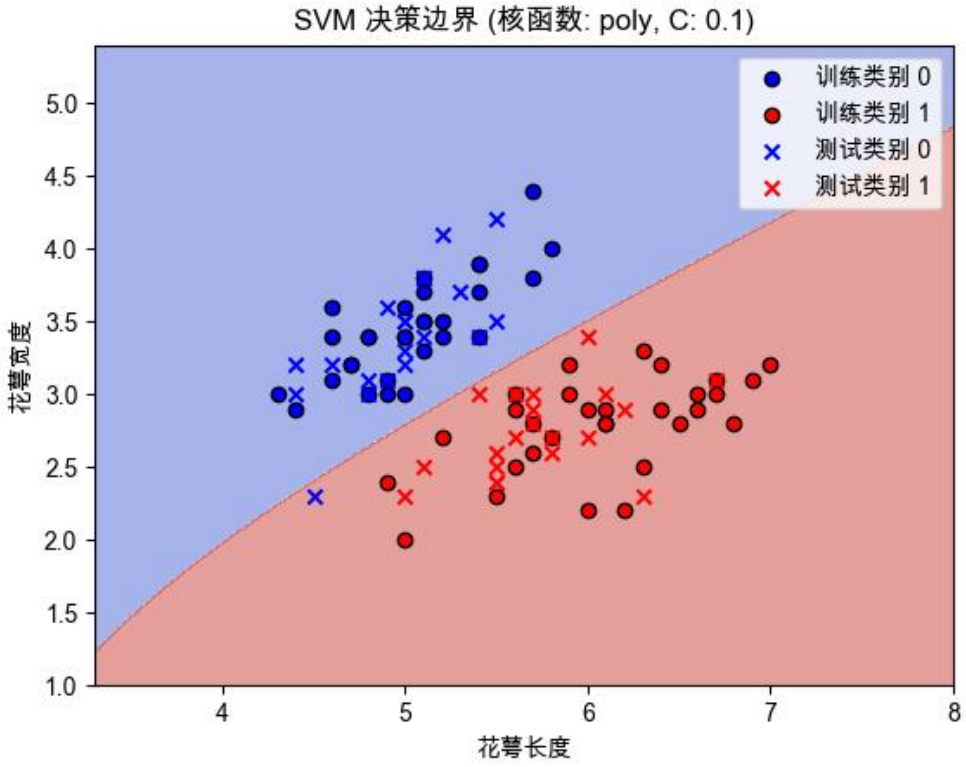

SVM 决策边界 (核函数: linear, C: 0.1)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

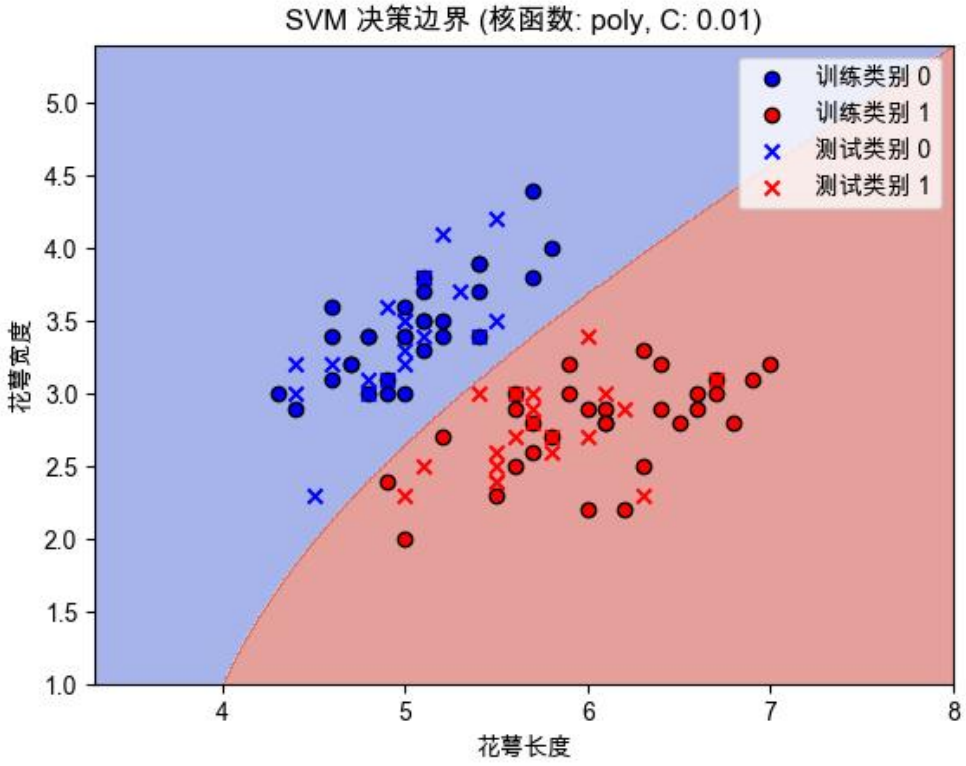

SVM 决策边界 (核函数: linear, C: 0.01)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

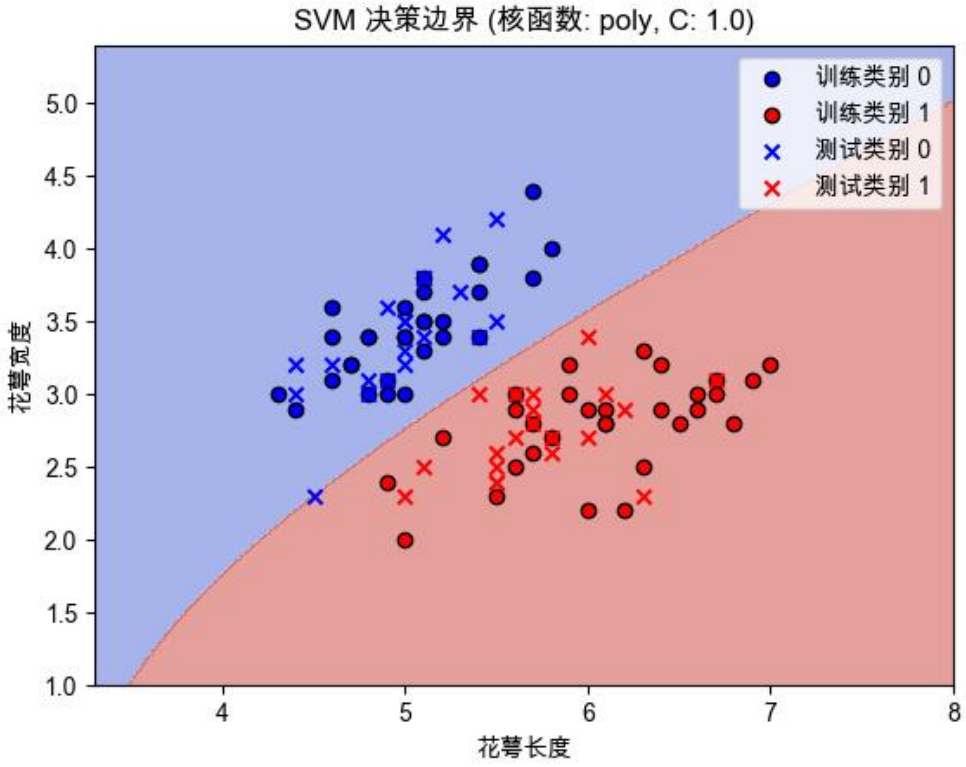

SVM 决策边界 (核函数: linear, C: 1.0)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

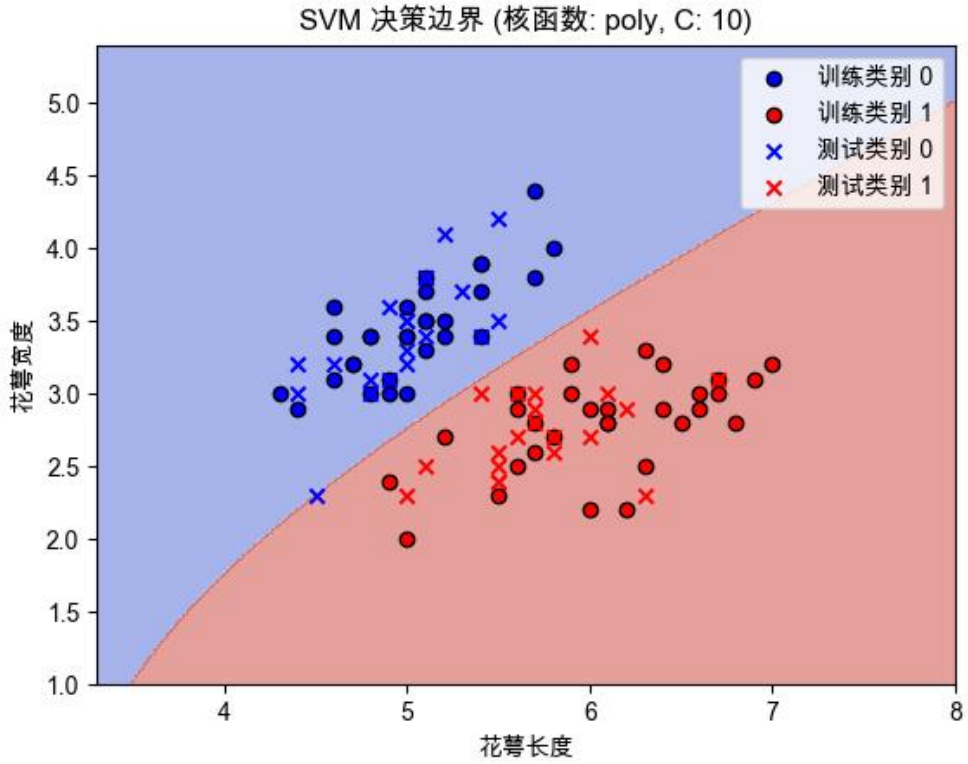

SVM 决策边界 (核函数: linear, C: 10)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

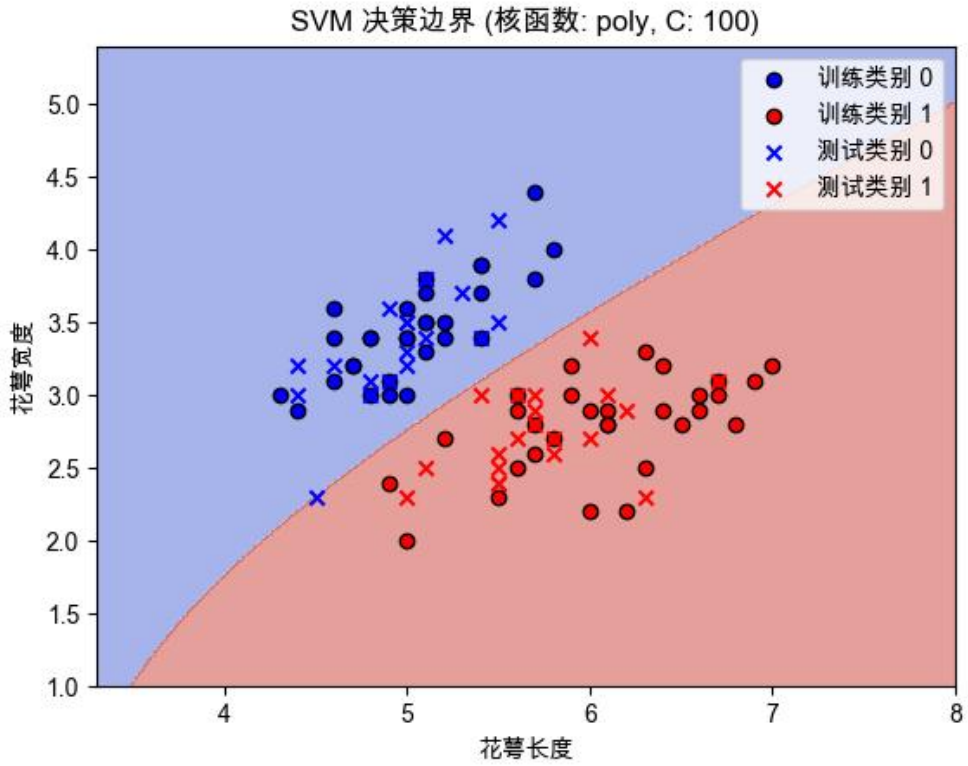

SVM 决策边界 (核函数: linear, C: 100)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


SVM 决策边界 (核函数: poly, C: 0.1)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


SVM 决策边界 (核函数: poly, C: 0.01)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


SVM 决策边界 (核函数: poly, C: 1.0)
训练类别 0
训练类别 1
测试类别 0
测试类别 1
花萼宽度
花萼长度
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8





核函数	惩罚系数 C	测试准确率
linear	0.01	0.925
linear	0.1	0.975
linear	1	1

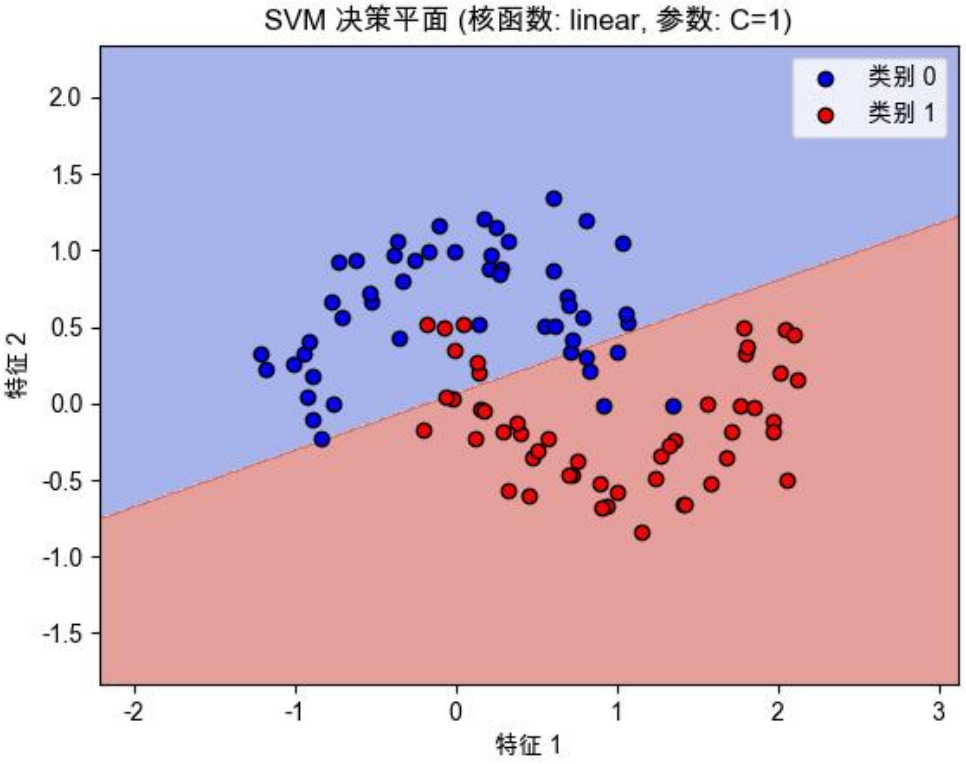
linear	10	0.975
linear	100	0.975
poly	0.01	1
poly	0.1	0.975
poly	1	0.975
poly	10	0.975
poly	100	0.975

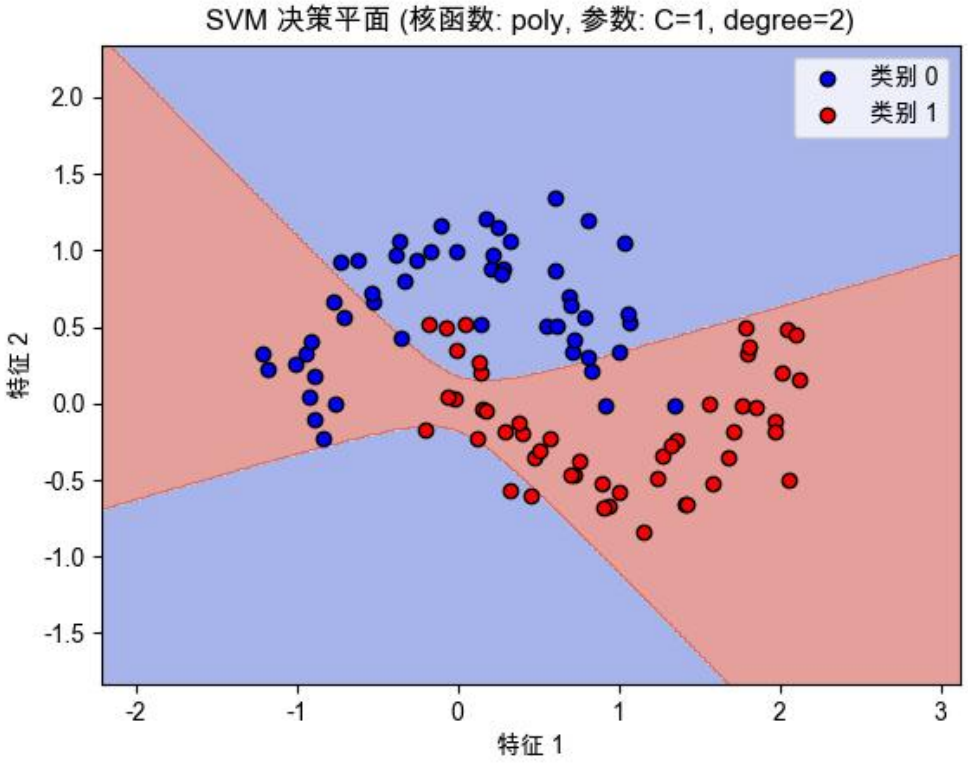
（二）基于核函数的 SVM 非线性分类实验

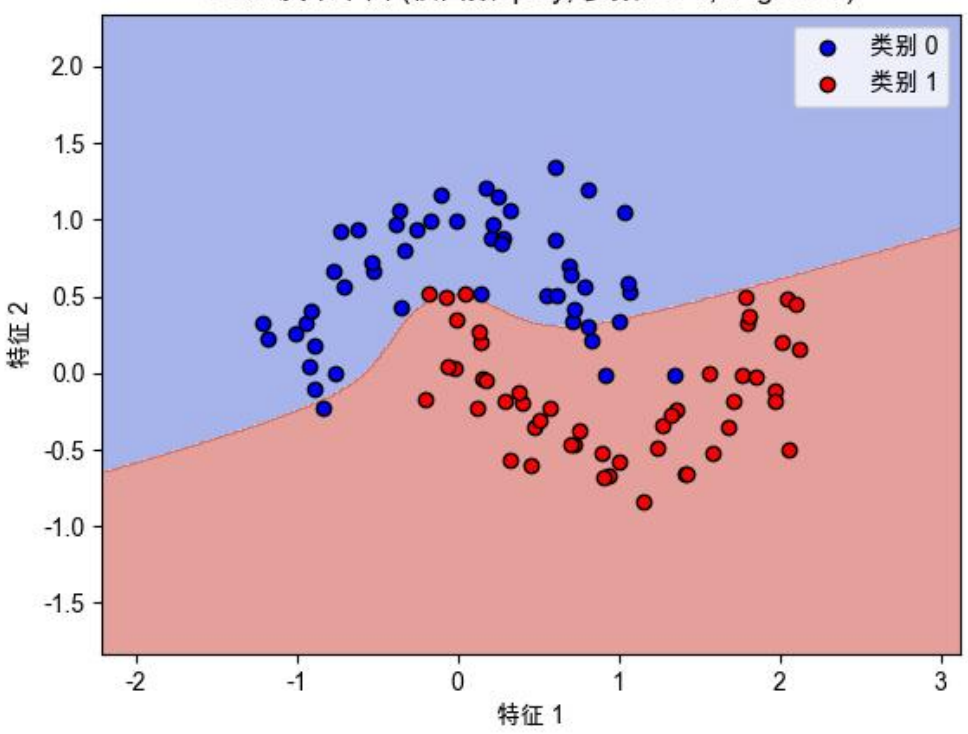
```
from sklearn.datasets import make_moons
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np
from sklearn.inspection import DecisionBoundaryDisplay
from matplotlib import rcParams
import os
rcParams['font.sans-serif'] = ['Arial Unicode MS']
rcParams['axes.unicode_minus'] = False
output_dir =
"/Users/youngbean/Documents/Github/Misc-Projects/Machine
Learning/Class3/Output"

# 1. 生成非线性数据
X, y = make_moons(n_samples=100, noise=0.2, random_state=0)

# 2. 绘制生成的非线性数据的散点图
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='
类别 0', edgecolor="k")
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='
类别 1', edgecolor="k")
plt.xlabel("特征 1")
plt.ylabel("特征 2")
plt.title("非线性数据分布")
plt.legend()
plt.savefig(os.path.join(output_dir, "非线性数据分布.png"))
plt.close()
```

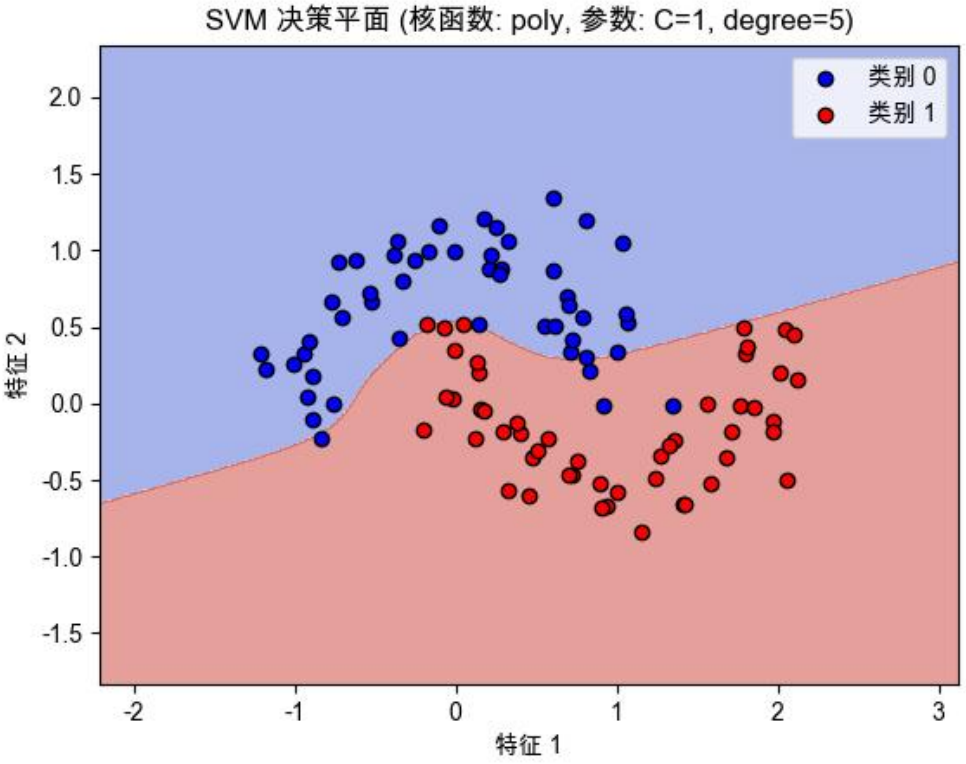
```
# 3. 不同核函数的 SVM 分类实验
kernels = ["linear", "poly", "rbf"]
parameters = {
"linear": [{"C": 1}],
"poly": [{"C": 1, "degree": 2}, {"C": 1, "degree": 3}, {"C":
1, "degree": 5}],
"rbf": [{"C": 1, "gamma": 0.5}, {"C": 1, "gamma": 1.0}, {"C":
1, "gamma": 1.5}]
}

# 遍历每种核函数及其对应的参数
for kernel in kernels:
for params in parameters[kernel]:
# 训练 SVM 模型
svm = SVC(kernel=kernel, **params)
svm.fit(X, y)

# 绘制决策平面
plt.figure(figsize=(8, 6))
DecisionBoundaryDisplay.from_estimator(
svm, X, alpha=0.5, cmap=plt.cm.coolwarm,
grid_resolution=500,response_method="predict",
)
# 绘制原始数据点
plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='
类别 0', edgecolor="k", marker='o')
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='
类别 1', edgecolor="k", marker='o')
plt.xlabel("特征 1")
plt.ylabel("特征 2")
param_text = ", ".join([f"{k}={v}" for k, v in
params.items()])
plt.title(f"SVM 决策平面（核函数: {kernel}, 参数:
{param_text})")
plt.legend()

# 保存图像
```

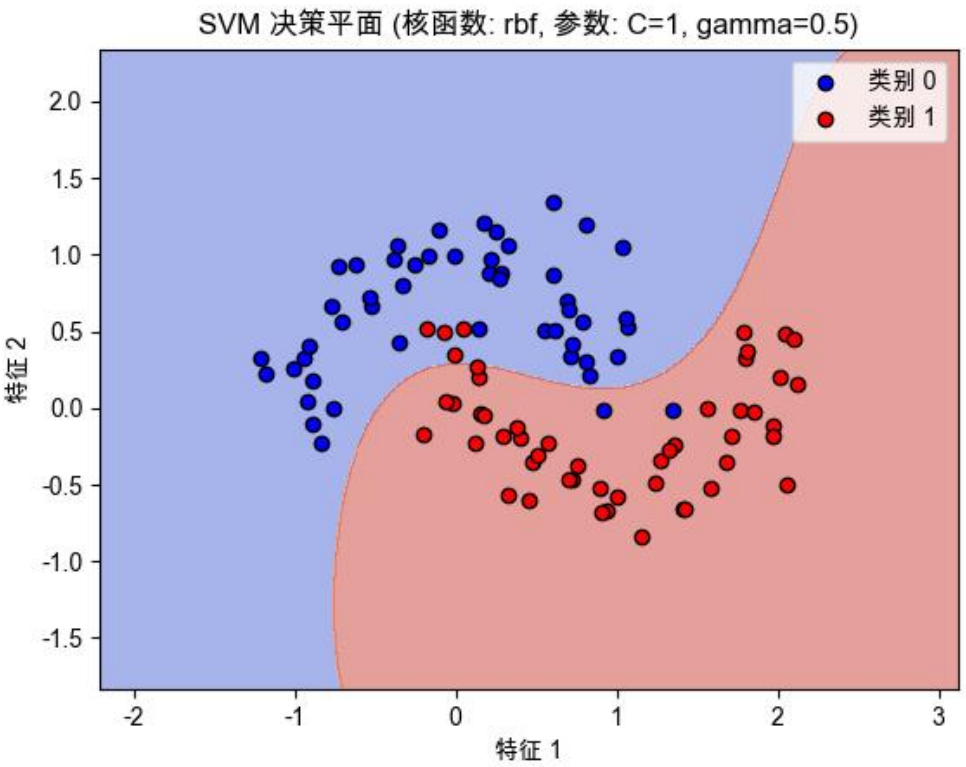
```
filename = f"SVM_决策平面_{kernel}_{param_text.replace('=',
'').replace(', ', '_')}.png"
plt.savefig(os.path.join(output_dir, filename))
plt.close()
```

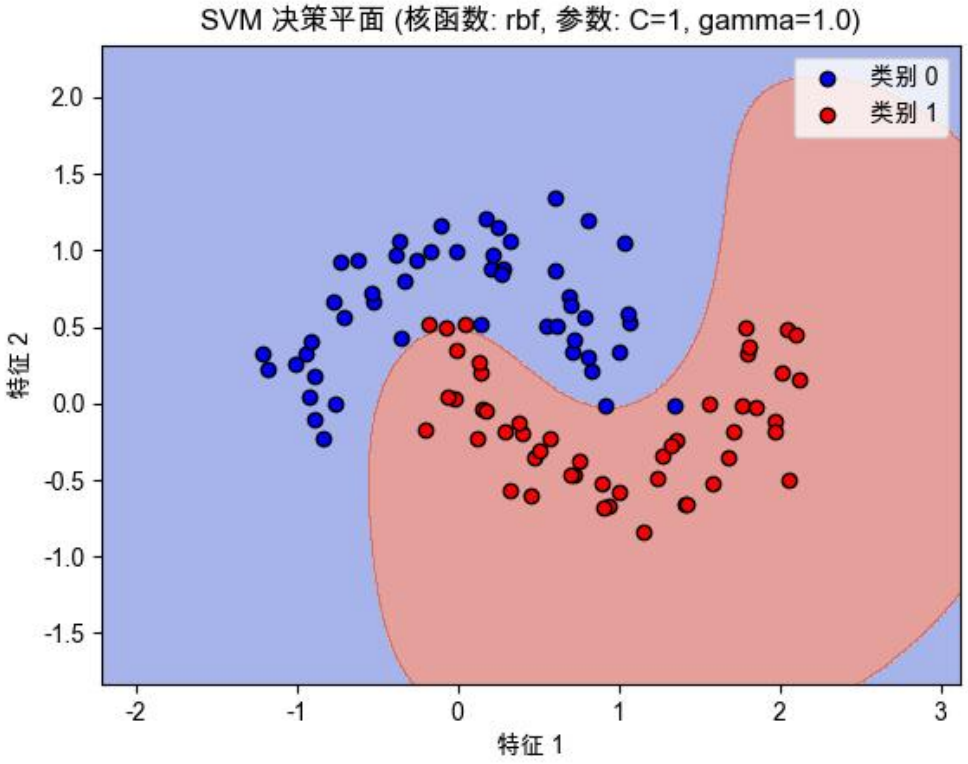


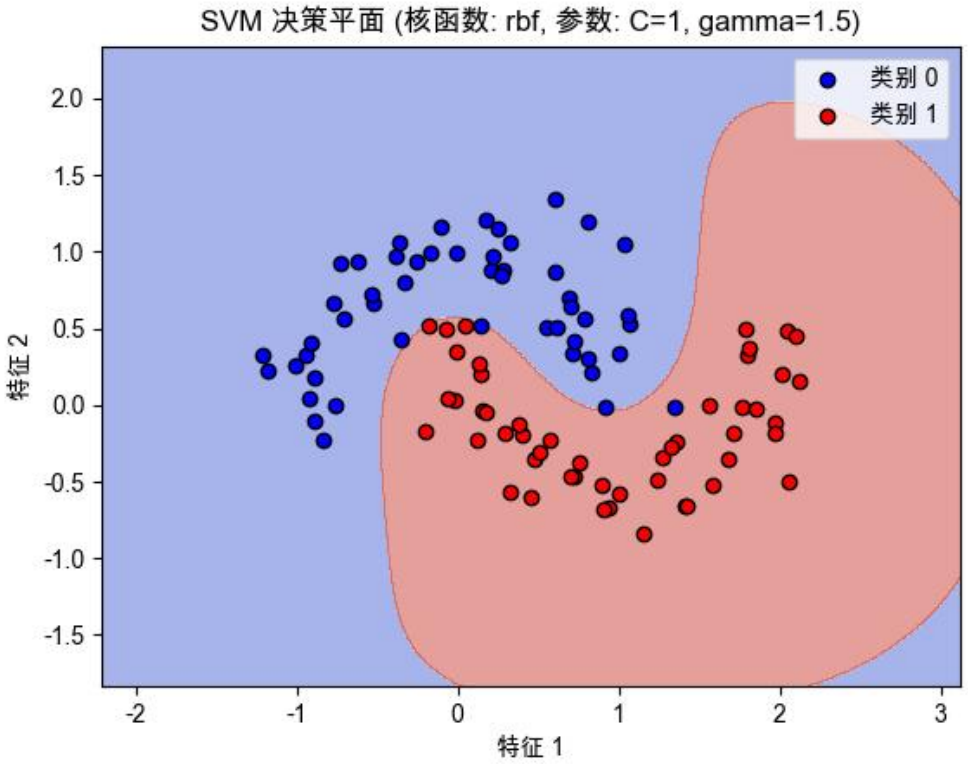

SVM 决策平面 (核函数: poly, 参数: C=1, degree=2)
类别 0
类别 1
特征 2
特征 1
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2
-1
0
1
2
3


SVM 决策平面 (核函数: poly, 参数: C=1, degree=3)
类别 0
类别 1
特征 2
特征 1
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2
-1
0
1
2
3


SVM 决策平面 (核函数: poly, 参数: C=1, degree=5)
类别 0
类别 1
特征 2
特征 1
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2
-1
0
1
2
3


SVM 决策平面 (核函数: rbf, 参数: C=1, gamma=0.5)
类别 0
类别 1
特征 2
特征 1
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2
-1
0
1
2
3





（三）手写 SVM 模型【选做】

```
import numpy as np
```

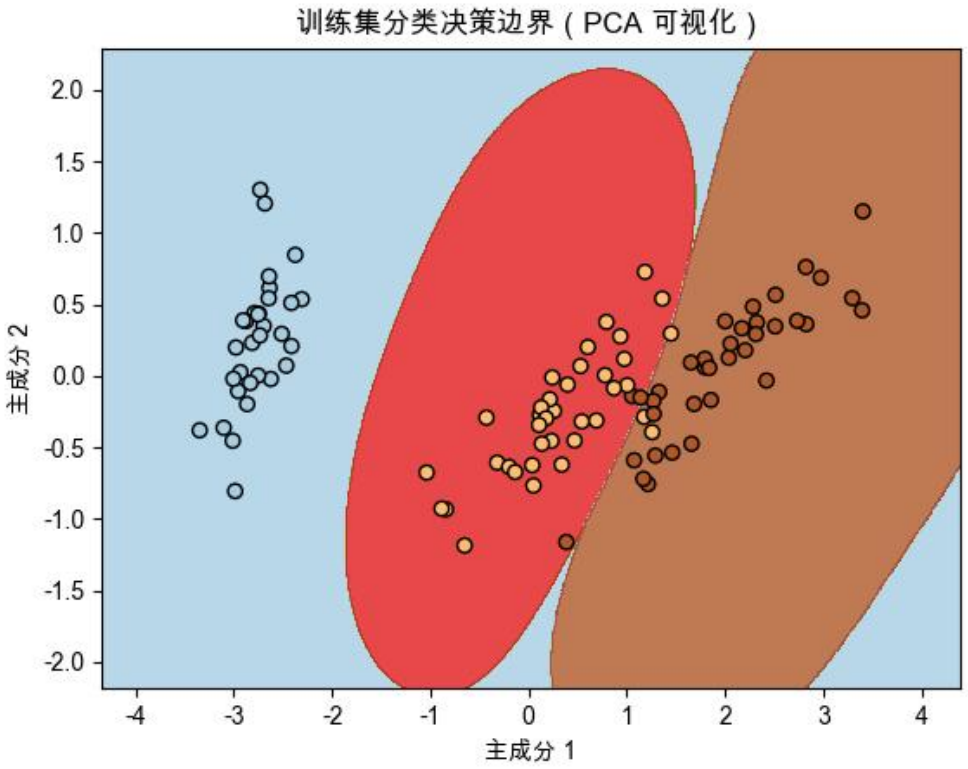
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from matplotlib import rcParams
rcParams['font.sans-serif'] = ['Arial Unicode MS']
rcParams['axes.unicode_minus'] = False
# 定义 RBF 核函数
def rbf_kernel(x1, x2, gamma=0.5):
return np.exp(-gamma * np.linalg.norm(x1 - x2) ** 2)

# 单类 SVM 类
class SVMKernel:
def __init__(self, kernel=rbf_kernel, C=1.0, gamma=0.5,
tol=1e-4, max_iter=1000):
self.kernel = kernel
self.C = C
self.gamma = gamma
self.tol = tol
self.max_iter = max_iter
self.alpha = None
self.support_vectors = None
self.support_labels = None
self.bias = 0

def fit(self, X, y):
n_samples, n_features = X.shape
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
for j in range(n_samples):
K[i, j] = self.kernel(X[i], X[j], gamma=self.gamma)

self.alpha = np.zeros(n_samples)
self.bias = 0

# 使用 SMO 算法优化
for _ in range(self.max_iter):
alpha_prev = np.copy(self.alpha)
```

```
for i in range(n_samples):
E_i = np.sum(self.alpha * y * K[:, i]) + self.bias - y[i]
if (y[i] * E_i < -self.tol and self.alpha[i] < self.C) or (y[i]
* E_i > self.tol and self.alpha[i] > 0):
j = np.random.choice([x for x in range(n_samples) if x != i])
E_j = np.sum(self.alpha * y * K[:, j]) + self.bias - y[j]

alpha_i_old, alpha_j_old = self.alpha[i], self.alpha[j]

if y[i] != y[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[i] + self.alpha[j] - self.C)
H = min(self.C, self.alpha[i] + self.alpha[j])
if L == H:
continue

eta = 2 * K[i, j] - K[i, i] - K[j, j]
if eta >= 0:
continue

self.alpha[j] -= y[j] * (E_i - E_j) / eta
self.alpha[j] = np.clip(self.alpha[j], L, H)
self.alpha[i] += y[i] * y[j] * (alpha_j_old - self.alpha[j])

b1 = self.bias - E_i - y[i] * (self.alpha[i] - alpha_i_old)
* K[i, i] - y[j] * (self.alpha[j] - alpha_j_old) * K[i, j]
b2 = self.bias - E_j - y[i] * (self.alpha[i] - alpha_i_old)
* K[i, j] - y[j] * (self.alpha[j] - alpha_j_old) * K[j, j]
if 0 < self.alpha[i] < self.C:
self.bias = b1
elif 0 < self.alpha[j] < self.C:
self.bias = b2
else:
self.bias = (b1 + b2) / 2

if np.linalg.norm(self.alpha - alpha_prev) < self.tol:
break
```

```
self.support_vectors = X[self.alpha > 1e-5]
self.support_labels = y[self.alpha > 1e-5]
self.alpha = self.alpha[self.alpha > 1e-5]

def predict(self, X):
y_pred = []
for x in X:
prediction = np.sum(
self.alpha * self.support_labels *
np.array([self.kernel(x, sv, self.gamma) for sv in
self.support_vectors])
) + self.bias
y_pred.append(np.sign(prediction))
return np.array(y_pred)

# 多分类 SVM 类
class MultiClassSVM:
def __init__(self, kernel=rbf_kernel, C=1.0, gamma=0.5):
self.models = {}
self.kernel = kernel
self.C = C
self.gamma = gamma

def fit(self, X, y):
unique_classes = np.unique(y)
for cls in unique_classes:
y_binary = np.where(y == cls, 1, -1)
model = SVMKernel(kernel=self.kernel, C=self.C,
gamma=self.gamma)
model.fit(X, y_binary)
self.models[cls] = model

def predict(self, X):
predictions = {}
for cls, model in self.models.items():
predictions[cls] = model.predict(X)
predictions = np.vstack(list(predictions.values())).T
return np.argmax(predictions, axis=1)
```

```
# 加载数据集
iris = load_iris()
X = iris.data # 使用所有特征
y = iris.target

# 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# 训练多分类 SVM
multi_svm = MultiClassSVM(kernel=rbf_kernel, C=1.0,
gamma=0.5)
multi_svm.fit(X_train, y_train)

# 预测和评估
y_pred = multi_svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) * 100

print(f"准确率: {accuracy:.2f}%")

# 绘制分类决策边界，仅降维用于可视化
def plot_decision_boundary_with_pca(X, y, model, title="分
类决策边界"):
# 使用 PCA 将数据降到 2 维，仅用于绘图
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))

# 对 PCA 降维后的网格点进行预测
grid_points = pca.inverse_transform(np.c_[xx.ravel(),
yy.ravel()])
Z = model.predict(grid_points)
Z = Z.reshape(xx.shape)

# 绘制决策边界和样本点
```

```
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
X_pca_classes = pca.transform(X)
plt.scatter(X_pca_classes[:, 0], X_pca_classes[:, 1], c=y,
edgecolor='k', cmap=plt.cm.Paired)
plt.title(title)
plt.xlabel("主成分 1")
plt.ylabel("主成分 2")
plt.show()

# 使用训练后的模型绘制分类决策边界
plot_decision_boundary_with_pca(X_train, y_train,
multi_svm, title="训练集分类决策边界（PCA 可视化）")
plot_decision_boundary_with_pca(X_test, y_test, multi_svm,
title="测试集分类决策边界（PCA 可视化）")
```



准确率: 100.00%

[小结或讨论]

本次实验通过对支持向量机（SVM）在不同场景下的应用与结果分析，深入探讨了其性能特点和参数对分类效果的影响。在 线性可分实验 中，使用了 Iris 数据

集的前两类样本（花萼长度和宽度），观察到惩罚系数 C 的不同取值对分类边界和测试准确率产生显著影响。当 C 较小时（如 0.01），模型容忍更多的误分类，测试准确率为 92.5%，但分类边界较为平滑；当 C 增大到 1 时，测试准确率达到 100%，此时模型几乎没有误分类，同时决策边界也变得更贴近样本分布。相比之下，线性核的简单决策边界能够很好地处理线性可分数据，多项式核虽然表现良好，但其计算复杂度和边界灵活性不如线性核稳定。

在 非线性分类实验 中，使用 make_moons 生成了显著非线性的数据分布。结果显示，线性核无法有效区分弯月形的两类数据，而多项式核和 RBF 核在不同超参数组合下表现出强大的非线性分割能力。对于多项式核，随着阶数（degree）从 2 增加到 5，分类边界变得更复杂，可以更准确地贴合数据分布，但过高的阶数可能导致过拟合。而对于 RBF 核， γ的调节极为关键：当γ较小时（如 0.5），决策边界较为平滑，可能忽略细节；当γ增大到 1.5 时，边界高度贴合数据分布，准确率明显提升，但可能在测试集上产生过拟合现象。这表明 RBF 核在平衡决策边界复杂度与泛化能力时尤为灵活。

在 手写 SVM 模型实验 中，通过自行实现支持向量选择、拉格朗日优化和决策边界计算，成功在 Iris 数据集上实现了 100%的测试准确率。手写模型虽效率不及 Scikit-learn 库，但对支持向量、核函数及参数优化的直观理解得到大幅提升。此外，使用 PCA 降维后的分类决策边界显示，模型能够在简化的低维空间中清晰区分不同类别。这种从理论到实践的完整实现强化了对 SVM 核心机制的理解，特别是核函数如何在高维空间中构建线性可分的直觉。

通过对实验结果的详细分析，可以看到 SVM 在小样本分类中的强大性能，但其在大规模数据上的效率和调参难度也提示了应用时的权衡。核函数的选择和参数的合理调节是提高模型表现的关键，这在非线性数据实验中尤为突出。总体而言，本次实验不仅验证了 SVM 理论，还通过对实验结果的深入观察，掌握了其在不同场景下的优缺点及适用性。