重庆科技学院学生实验报告

课程名称	机器学习		实验项目名称	支持向量机	
开课学院及实验室	计算机科学与工程学院			实验日期	2025 年 10 月 16 日
学生姓名	刘志贵	学号	2023440221	专业班级	智科 2023-02
指导教师	易军		实验成绩		

一、实验目的和要求

1. 理解支持向量机的基本原理。
2. 通过可视化直观理解不同核函数（线性核、RBF 核）对决策边界的影响。
3. 将 SVM 应用于实际问题，学会通过调整参数来优化模型性能。

二、实验内容

1. 用提供的二维数据集 testSet.txt，testSetRBF.txt，testSetRBF2.txt，然后用 scikit-learn 库复现直观展示不同核函数和参数的效果。
2. 将 SVM 应用于 digits 文件夹中的图像数据，完成指定的手写数字（9 和非 9）二分类任务，最后通过错误率来评估模型性能。

三、实验步骤与结果

1. 收集数据

根据实验的要求，第一步是处理提供的数据。首先需要定义能够读取并解析这些文件的函数。下面的代码块定义了处理三种不同类型数据集 testSet.txt，testSetRBF.txt 的二维数据还有 digits 文件夹下的图像数据的函数。

先是导入需要的库：

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import svm
from sklearn.metrics import accuracy_score
```

图 1 导入库函数

然后定义加载数据集的函数：

```
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    with open(fileName) as fr:
        for line in fr.readlines():
            lineArr = line.strip().split('\t')
            dataMat.append([float(lineArr[0]), float(lineArr[1])])
            labelMat.append(float(lineArr[2]))
    return np.array(dataMat), np.array(labelMat)
```

图 2 数据加载函数

之后是将 32x32 的二进制图像矩阵转换为 1x1024 的向量的函数：

```python
def img2vector(filename):
    returnVect = np.zeros((1,1024))
    with open(filename) as fr:
        for i in range(32):
            lineStr = fr.readline()
            for j in range(32):
                returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect
```

图 3 转换图像向量函数

最后是从目录加载手写数字图像数据的函数：

```python
def loadImages(dirName):
    hwLabels = []
    trainingFileList = os.listdir(dirName)
    m = len(trainingFileList)
    trainingMat = np.zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        # 将问题定义为二分类
        if classNumStr == 9:
            hwLabels.append(-1)
        else:
            hwLabels.append(1)
        trainingMat[i,:] = img2vector('%s/%s' % (dirName, fileNameStr))
    return trainingMat, np.array(hwLabels)
```

图 4 加载手写数字图像函数

到这里数据准备和辅助函数定义就已经完成了。

2. 准备数据

可以看到我把数据集都放在一个目录下面：

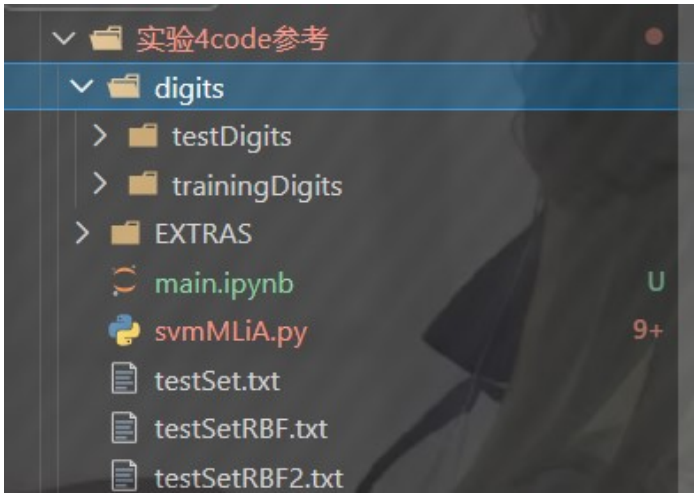


图 5 数据目录图

然后我吧目录的训练集和测试集全部进行加载和分配，并在最后输出数据集的维度用来验证加载结果：

```
X_linear, y_linear = loadDataSet('testSet.txt')
print(f"线性数据集testSet.txt数据维度: {X_linear.shape}")

X_rbf_train, y_rbf_train = loadDataSet('testSetRBF.txt')
X_rbf_test, y_rbf_test = loadDataSet('testSetRBF2.txt')
print(f"RBF训练集testSetRBF.txt数据维度: {X_rbf_train.shape}")
print(f"RBF测试集testSetRBF2.txt数据维度: {X_rbf_test.shape}")

X_train_digits, y_train_digits = loadImages('digits/trainingDigits')
print(f"手写数字训练数据维度: {X_train_digits.shape}")

X_test_digits, y_test_digits = loadImages('digits/testDigits')
print(f"手写数字测试数据维度: {X_test_digits.shape}")
```

✓ 0.1s

```
线性数据集testSet.txt数据维度: (100, 2)
RBF训练集testSetRBF.txt数据维度: (100, 2)
RBF测试集testSetRBF2.txt数据维度: (100, 2)
手写数字训练数据维度: (402, 1024)
手写数字测试数据维度: (186, 1024)
```

图 6 加载数据图

3. 分析数据

这里主要是完成对图像的目测，我通过了一个辅助函数把向量数据还原成图像然后打印出来，这样直观的看出是否被正确的加载和解析：

```
def print_digit(vector):
    matrix = vector.reshape(32, 32)
    for row in matrix:
        print(''.join(['#' if pixel > 0 else '.' for pixel in row]))

print("目测分析:打印一个非'9'的数字样本，标签为 1:")
print_digit(X_train_digits[0])

nine_index = np.where(y_train_digits == -1)[0][0]
print("\n目测分析:打印一个数字'9'的样本，标签为 -1:")
print_digit(X_train_digits[nine_index])
```

✓ 0.0s

图 7 目测代码图

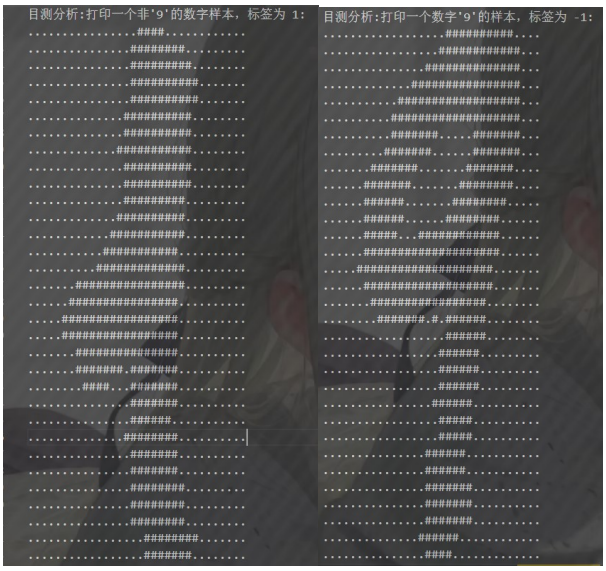


图 8 目测结果输出

采用的是 txt 形式输出，分看解析效果无误，数据加载完成后就可以开始训练了。

4. 训练算法

下面展示了使用线性核函数和 RBF 核函数进行的训练和可视化操作，并且我进行了对径向基核函数采用不同的设置来运行 SMO 算法。

先展示线性核函数的训练和可视化：

```
| 解释 | 添加注释
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

| 解释 | 添加注释
def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

print("线性核模型")
clf_linear = svm.SVC(kernel='linear', C=1.0).fit(X_linear, y_linear)
fig, ax = plt.subplots(figsize=(8, 6))
xx, yy = make_meshgrid(X_linear[:, 0], X_linear[:, 1])
plot_contours(ax, clf_linear, xx, yy, cmap=plt.cm.plasma, alpha=0.8)
ax.scatter(X_linear[:, 0], X_linear[:, 1], c=y_linear, cmap=plt.cm.plasma, s=30, edgecolors='k')
ax.set_title('SVC with Linear Kernel')
plt.show()
✓ 0.1s
```

图 9 线性核函数代码

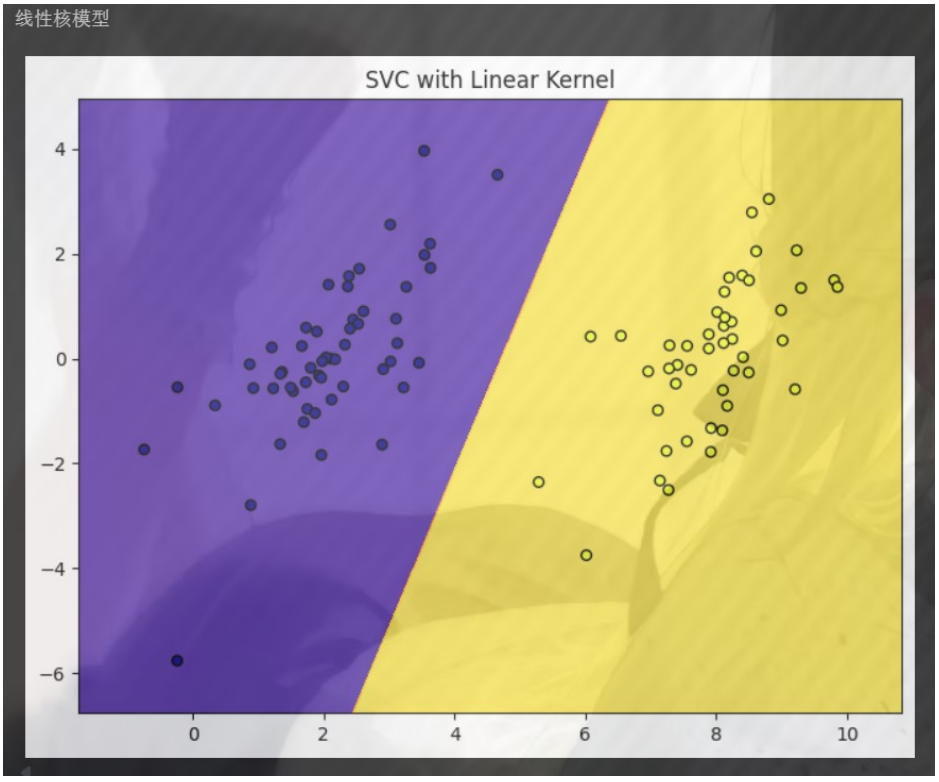


图 10 线性核函数可视化图像

然后是 RBF 核函数的训练与可视化代码和结果：

```
print("\n不同gamma参数的RBF核模型")
C = 200.0
models = (svm.SVC(kernel='rbf', gamma=0.1, C=C),
          svm.SVC(kernel='rbf', gamma=1, C=C),
          svm.SVC(kernel='rbf', gamma=50, C=C))
models = (clf.fit(X_rbf_train, y_rbf_train) for clf in models)
titles = ('gamma = 0.1', 'gamma = 1', 'gamma = 50')

fig, sub = plt.subplots(1, 3, figsize=(15, 4))
xx_rbf, yy_rbf = make_meshgrid(X_rbf_train[:, 0], X_rbf_train[:, 1])

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx_rbf, yy_rbf, cmap=plt.cm.plasma, alpha=0.8)
    ax.scatter(X_rbf_train[:, 0], X_rbf_train[:, 1], c=y_rbf_train, cmap=plt.cm.plasma, s=20, edgecolors='k')
    ax.set_title(title)
plt.show()
✓ 0.4s
```

图 11 RBF 核函数训练代码

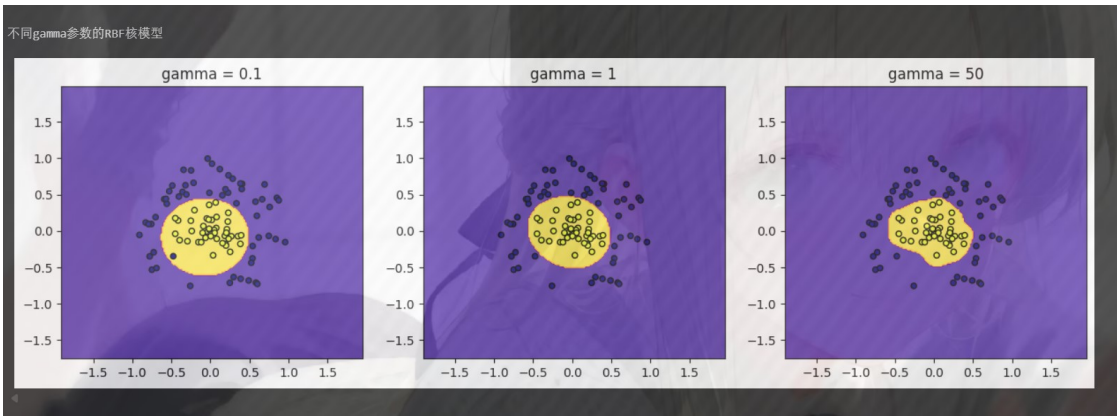


图 12 不同 gamma 可视化图

上面清晰的展示了两个核函数的结果图像，可以发现线性核函数比较单一，能比较好的处理简单的数据，RBF 核函数就比较厉害，不想线性核函数一样单一的分割，可以处理比较复杂的数据。

然后就是在高维的手写数字数据上训练模型，如下：

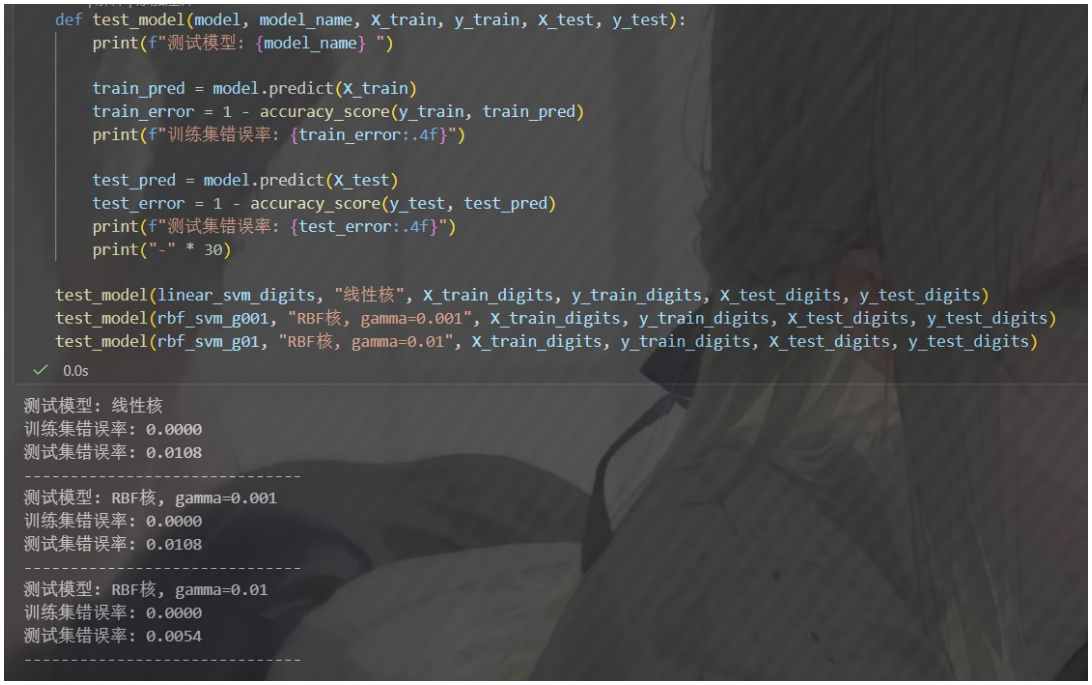
```
print("训练线性核模型")
linear_svm_digits = SVC(kernel='linear', C=200).fit(X_train_digits, y_train_digits)

print("\n训练RBF核模型 (gamma=0.001)")
rbf_svm_g001 = SVC(kernel='rbf', C=200, gamma=0.001).fit(X_train_digits, y_train_digits)
print("训练RBF核模型 (gamma=0.01)")
rbf_svm_g01 = SVC(kernel='rbf', C=200, gamma=0.01).fit(X_train_digits, y_train_digits)
✓ 0.0s
训练线性核模型

训练RBF核模型 (gamma=0.001)
训练RBF核模型 (gamma=0.01)
```

图 13 训练手写数据

这里还是使用的线性核函数和 RBF 核函数，并且用了不同的参数训练。

5. 测试算法

在上面训练好了之后就可以测试一下效果怎么样了，可以发现效果最好的是 RBF 核函数且 gamma=0.01 的时候，因为线性核函数能够处理的数据肯定不能太复杂，所以 RBF 的效果好是正常的。



```
def test_model(model, model_name, X_train, y_train, X_test, y_test):
    print(f"测试模型: {model_name} ")

    train_pred = model.predict(X_train)
    train_error = 1 - accuracy_score(y_train, train_pred)
    print(f"训练集错误率: {train_error:.4f}")

    test_pred = model.predict(X_test)
    test_error = 1 - accuracy_score(y_test, test_pred)
    print(f"测试集错误率: {test_error:.4f}")
    print("-" * 30)

test_model(linear_svm_digits, "线性核", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
test_model(rbf_svm_g001, "RBF核, gamma=0.001", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
test_model(rbf_svm_g01, "RBF核, gamma=0.01", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
```

✓ 0.0s

```
测试模型: 线性核
训练集错误率: 0.0000
测试集错误率: 0.0108
------------------------------
测试模型: RBF核, gamma=0.001
训练集错误率: 0.0000
测试集错误率: 0.0108
------------------------------
测试模型: RBF核, gamma=0.01
训练集错误率: 0.0000
测试集错误率: 0.0054
------------------------------
```

图 14 测试算法

四、实验结果

结果我用一个表格直观展示：

核函数类型	核参数（gamma）	训练集错误率	测试集错误率
线性核	-	0.0000	0.0108
RBF 核	0.001	0.0000	0.0108
RBF 核	0.01	0.0000	0.0054

表 1 测试算法结果

上面的表格就可以清晰的显示这次实验训练算法的结果了。

五、模型评价

通过对几个数据集的可视化能够看到线性核生成直线边界，但只能适用于线性可分问题；而 RBF 核能够生成灵活的曲线边界，适合用于复杂的非线性问题。

在参数调整方面，实验我使用了 gamma=0.001，这时候测试错误率最低。但 gamma 增大到 0.01 的时候，测试错误率上升了，说明修改参数对算法也有很大的影响。

附件

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import svm
from sklearn.metrics import accuracy_score

def loadDataSet(fileName):
    dataMat = []; labelMat = []
    with open(fileName) as fr:
        for line in fr.readlines():
            lineArr = line.strip().split('\t')
            dataMat.append([float(lineArr[0]), float(lineArr[1])])
            labelMat.append(float(lineArr[2]))
    return np.array(dataMat), np.array(labelMat)

def img2vector(filename):
    returnVect = np.zeros((1,1024))
    with open(filename) as fr:
        for i in range(32):
            lineStr = fr.readline()
            for j in range(32):
                returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect

def loadImages(dirName):
    hwLabels = []
    trainingFileList = os.listdir(dirName)
    m = len(trainingFileList)
    trainingMat = np.zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        # 将问题定义为二分类
        if classNumStr == 9:
            hwLabels.append(-1)
        else:
```

```
                hwLabels.append(1)
            trainingMat[i,:] = img2vector('%s/%s' % (dirName, fileNameStr))
    return trainingMat, np.array(hwLabels)

X_linear, y_linear = loadDataSet('testSet.txt')
print(f"线性数据集 testSet.txt 数据维度: {X_linear.shape}")

X_rbf_train, y_rbf_train = loadDataSet('testSetRBF.txt')
X_rbf_test, y_rbf_test = loadDataSet('testSetRBF2.txt')
print(f"RBF 训练集 testSetRBF.txt 数据维度: {X_rbf_train.shape}")
print(f"RBF 测试集 testSetRBF2.txt 数据维度: {X_rbf_test.shape}")

X_train_digits, y_train_digits = loadImages('digits/trainingDigits')
print(f"手写数字训练数据维度: {X_train_digits.shape}")

X_test_digits, y_test_digits = loadImages('digits/testDigits')
print(f"手写数字测试数据维度: {X_test_digits.shape}")

def print_digit(vector):
    matrix = vector.reshape(32, 32)
    for row in matrix:
        print(''.join(['#' if pixel > 0 else '.' for pixel in row]))

print("目测分析:打印一个非'9'的数字样本，标签为 1:")
print_digit(X_train_digits[0])

nine_index = np.where(y_train_digits == -1)[0][0]
print("\n 目测分析:打印一个数字'9'的样本，标签为 -1:")
print_digit(X_train_digits[nine_index])

def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

print("线性核模型")
clf_linear = svm.SVC(kernel='linear', C=1.0).fit(X_linear, y_linear)
fig, ax = plt.subplots(figsize=(8, 6))
xx, yy = make_meshgrid(X_linear[:, 0], X_linear[:, 1])
plot_contours(ax, clf_linear, xx, yy, cmap=plt.cm.plasma, alpha=0.8)
ax.scatter(X_linear[:, 0], X_linear[:, 1], c=y_linear, cmap=plt.cm.plasma, s=30,
edgecolors='k')
ax.set_title('SVC with Linear Kernel') #testSet.txt
plt.show()

print("\n 不同 gamma 参数的 RBF 核模型")
C = 200.0
models = (svm.SVC(kernel='rbf', gamma=0.1, C=C),
          svm.SVC(kernel='rbf', gamma=1, C=C),
          svm.SVC(kernel='rbf', gamma=50, C=C))
models = (clf.fit(X_rbf_train, y_rbf_train) for clf in models)
titles = ('gamma = 0.1', 'gamma = 1', 'gamma = 50')

fig, sub = plt.subplots(1, 3, figsize=(15, 4))
xx_rbf, yy_rbf = make_meshgrid(X_rbf_train[:, 0], X_rbf_train[:, 1])

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx_rbf, yy_rbf, cmap=plt.cm.plasma, alpha=0.8)
    ax.scatter(X_rbf_train[:, 0], X_rbf_train[:, 1], c=y_rbf_train, cmap=plt.cm.plasma,
s=20, edgecolors='k')
    ax.set_title(title)
plt.show()

print("训练线性核模型")
linear_svm_digits = SVC(kernel='linear', C=200).fit(X_train_digits, y_train_digits)

print("\n 训练 RBF 核模型 (gamma=0.001)")
rbf_svm_g001 = SVC(kernel='rbf', C=200, gamma=0.001).fit(X_train_digits, y_train_digits)
print("训练 RBF 核模型 (gamma=0.01)")
```

```
rbf_svm_g01 = SVC(kernel='rbf', C=200, gamma=0.01).fit(X_train_digits, y_train_digits)

def test_model(model, model_name, X_train, y_train, X_test, y_test):
    print(f"测试模型: {model_name} ")

    train_pred = model.predict(X_train)
    train_error = 1 - accuracy_score(y_train, train_pred)
    print(f"训练集错误率: {train_error:.4f}")

    test_pred = model.predict(X_test)
    test_error = 1 - accuracy_score(y_test, test_pred)
    print(f"测试集错误率: {test_error:.4f}")
    print("-" * 30)

test_model(linear_svm_digits, "线性核", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
test_model(rbf_svm_g001, "RBF 核, gamma=0.001", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
test_model(rbf_svm_g01, "RBF 核, gamma=0.01", X_train_digits, y_train_digits, X_test_digits, y_test_digits)
```