**第5章 支持向量机**

**一、简述数据预处理相关典型技术**（逻辑要通，**要有小标题**，要排版清晰，请不要网络大幅摘抄）。

1. **数据清洗**
   1. **将部分数据突变值，超出范围的值进行删除；**
   2. **对于缺失数据使用中位数、均值众数等数字特征进行填充缺失数据；**
   3. **去除重复值；**
2. **数据集成**
   1. **合并来自于不同数据源的数据**
   2. **处理不一致的数据格式等**
3. **数据变换**
   1. **归一化/标准化：将数据缩放到特定范围（如0到1）或转换为均值为0、标准差为1的分布。**
   2. **离散化：将连续数据分段为离散区间。**

**二、简述支持向量机**的思路、关键操作（不超过300字，逻辑要通，**要有小标题**，要排版清晰，请不要网络大幅摘抄）

**支持向量机（Support Vector Machine, SVM）是一种监督学习算法，广泛应用于分类和回归问题。其核心思想是通过找到一个最优超平面，将不同类别的数据点分开，并且在分类问题中最大化类别之间的间隔。SVM的目标是找到一个超平面，使得不同类别的数据点能够被正确分类，并且距离超平面最近的点的距离（即间隔）最大化。这些距离超平面最近的点被称为支持向量。**

**概念介绍：**

**超平面：指将两类样本点划分的直线或者平面称为超平面。**

**线性可分 ：指简单样本集合中可以将两类不同的样本，通过一个超平面进行分离，即正例和负例分别分布在超平面的两侧。**

**感知机模型：即对于样本点特征值，根据所给出的函数将该样本点的特征值输入后，通过函数输出其分类（或称之为正例或者反例）的模型。**

**三、利用SVM解决问题（回归或分类）。**

附：Sklearn常用数据集

|  |  |  |
| --- | --- | --- |
| 序号 | 数据集 | 学号尾号 |
| 1 | 鸢尾花数据集：load\_iris（） | 全做 调参 |
| 2 | 手写数字数据集：load\_digits（） | 0 1 2 |
| 3 | 乳腺癌数据集load-barest-cancer（） | 6 7 |
| 4 | 糖尿病数据集：load-diabetes（） | 8 9 3 |
| 5 | 波士顿房价数据集：load-boston（） | 4 5 |

利用SVM对以上数据集（学号尾号不同采用不同数据集）进行分析，找出最佳参数下的SVM更适合本数据集（准确率更高，不同回归、不同参数下的不同准确率要列出）。

要求：

1）针对当前数据、模型，**详叙数据预处理、模型参数调整过程及效果比对**；

2）代码要列出，重点代码加注释说明，特别是自己调试过程中的自我理解；

3）运行结果要截图，结果要文字说明；

4）注意排版

鸢尾花

*from* sklearn *import* svm  *# 导入支持向量机模型*

*import* numpy *as* np  *# 导入数值计算库*

*import* matplotlib.pyplot *as* plt  *# 导入绘图库*

*import* matplotlib *as* mpl  *# 导入matplotlib库*

*from* matplotlib *import* colors  *# 导入颜色管理模块*

*from* sklearn *import* model\_selection  *# 导入模型选择模块，用于数据集划分*

*from* sklearn.datasets *import* load\_iris  *# 导入鸢尾花数据集*

def load\_data():

*"""*

*加载鸢尾花数据集并进行预处理*

*返回：数据特征矩阵和标签向量*

*"""*

    data\_set = []

    data\_X = []

    data\_y = []

    iris = load\_iris()  *# 加载鸢尾花数据集*

    data\_X.append(iris.data)  *# 获取特征数据*

    data\_y.append(iris.target)  *# 获取目标分类标签*

    data\_X = np.array(data\_X)  *# 转换为numpy数组*

    data\_y = np.array(data\_y)  *# 转换为numpy数组*

    data\_X = np.reshape(data\_X, (150, 4))  *# 重塑特征数据形状为(150, 4)*

    data\_y = np.reshape(data\_y, (150))  *# 重塑标签数据形状为(150,)*

*# data\_X = data\_X[1:3,:]*

*return* data\_X, data\_y

def split\_data(*data\_X*, *data\_y*):

*"""*

*将数据集分割为训练集和测试集*

*参数：*

*data\_X: 特征矩阵*

*data\_y: 标签向量*

*返回：训练集特征、测试集特征、训练集标签、测试集标签*

*"""*

*# 使用train\_test\_split函数划分数据集，随机种子为42，测试集占30%，训练集占50%*

    X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(*data\_X*, *data\_y*, *random\_state*=42, *test\_size*=0.3, *train\_size*=0.5)

*# print(y\_train)*

*# 将特征数据转换为浮点数类型*

*for* index, item *in* enumerate(X\_train):

        X\_train[index] = list(map(float, item))

*for* index, item *in* enumerate(X\_test):

        X\_test[index] = list(map(float, item))

*# print(X\_train)*

    X\_train = np.array(X\_train)  *# 转换为numpy数组*

    X\_test = np.array(X\_test)  *# 转换为numpy数组*

*return* X\_train, X\_test, y\_train, y\_test

def show\_data(*X\_train*, *y\_train*):

*"""*

*可视化训练数据的散点图*

*参数：*

*X\_train: 训练集特征数据*

*y\_train: 训练集标签数据*

*"""*

*# 根据不同类别分离数据点索引*

    c0 = [i *for* i *in* range(len(*y\_train*)) *if* *y\_train*[i] == 1]  *# Iris-virginica类别索引*

    c1 = [i *for* i *in* range(len(*y\_train*)) *if* *y\_train*[i] == 0]  *# Iris-setosa类别索引*

    c2 = [i *for* i *in* range(len(*y\_train*)) *if* *y\_train*[i] == 2]  *# Iris-versicolor类别索引*

    plt.rcParams['font.sans-serif'] = ['SimHei']  *# 设置字体以显示中文标签*

    print(*X\_train*[c0, 0])

*# 绘制三种不同类别的散点图*

    plt.scatter(*x*=*X\_train*[c0, 0], *y*=*X\_train*[c0, 1], *color*='r', *marker*='s', *label*='Iris-virginica')

    plt.scatter(*x*=*X\_train*[c1, 0], *y*=*X\_train*[c1, 1], *color*='g', *marker*='o', *label*='Iris-setosa')

    plt.scatter(*x*=*X\_train*[c2, 0], *y*=*X\_train*[c2, 1], *color*='b', *marker*='v', *label*='Iris-versicolor')

*# 添加图表标签和标题*

    plt.xlabel("花萼宽度")

    plt.ylabel("花瓣长度")

    plt.title("各数据类型的散点图")

    plt.legend(*loc*='upper left')  *# 添加图例*

    plt.show()  *# 显示图表*

*if* \_\_name\_\_ == '\_\_main\_\_':

*# 加载数据集*

    data\_X, data\_y = load\_data()

*# 分割数据集为训练集和测试集*

    X\_train, X\_test, y\_train, y\_test = split\_data(data\_X, data\_y)

    show\_data(X\_train, y\_train)

*# 创建并训练不同核函数的SVM模型*

    clf1 = svm.SVC(*C*=1, *kernel*='linear', *decision\_function\_shape*='ovr').fit(X\_train, y\_train)  *# 线性核函数SVM*

    clf2 = svm.SVC(*C*=1, *kernel*='rbf', *gamma*=1).fit(X\_train, y\_train)  *# 径向基函数(RBF)核SVM*

    clf3 = svm.SVC(*kernel*='poly').fit(X\_train, y\_train)  *# 多项式核函数SVM*

    clf4 = svm.SVC(*kernel*='sigmoid').fit(X\_train, y\_train)  *# Sigmoid核函数SVM*

*# 评估线性核函数SVM模型性能*

    print("linear线性核函数-训练集：", clf1.score(X\_train, y\_train))  *# 在训练集上的准确率*

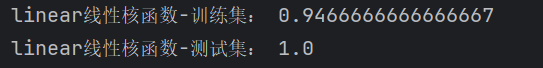
    print("linear线性核函数-测试集：", clf1.score(X\_test, y\_test))  *# 在测试集上的准确率*

*# 输出模型预测相关信息*

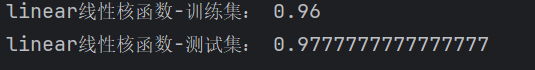
    print(clf1.decision\_function(X\_train))  *# 输出决策函数值*

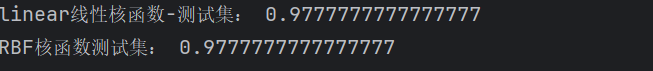
    print(clf1.predict(X\_train))  *# 输出预测结果*

**实验结果截图：**

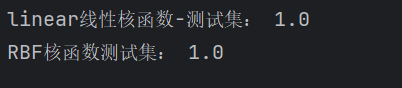
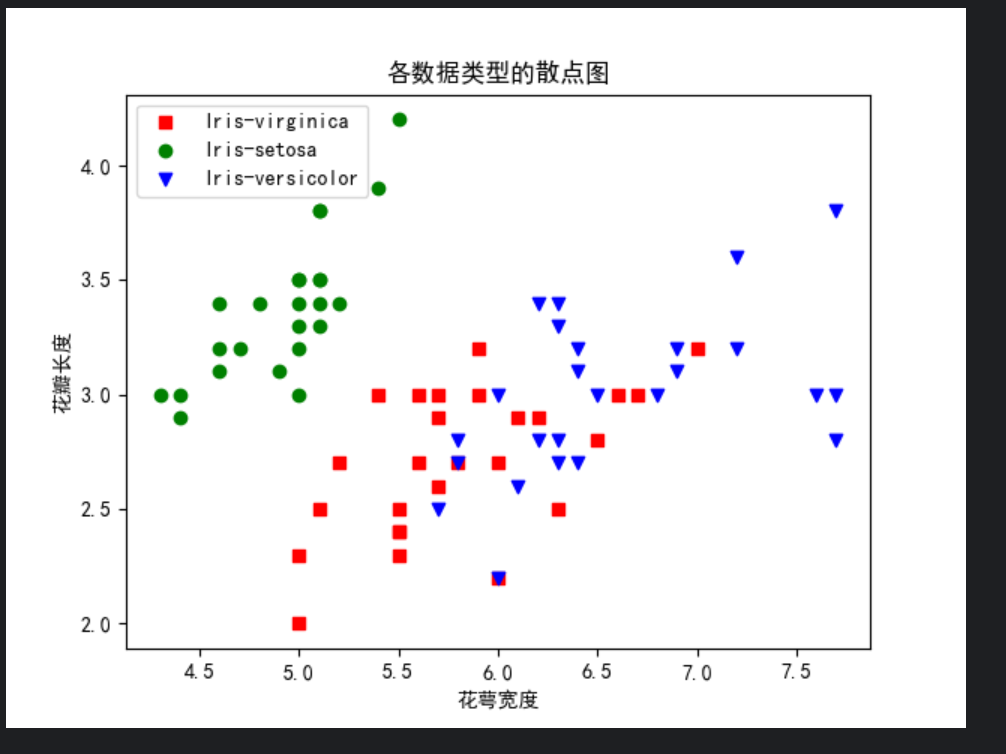
clf1 = svm.SVC(C=1,kernel='linear', decision\_function\_shape='ovr').fit(X\_train,y\_train)  
clf2 = svm.SVC(C=1, kernel='rbf', gamma=1).fit(X\_train,y\_train)****

clf1 = svm.SVC(C=12,kernel='linear', decision\_function\_shape='ovr').fit(X\_train,y\_train)  
clf2 = svm.SVC(C=2, kernel='rbf', gamma=0.4).fit(X\_train,y\_train)

****

clf1 = svm.SVC(C=10,kernel='linear', decision\_function\_shape='ovr').fit(X\_train,y\_train)  
clf2 = svm.SVC(C=23, kernel='rbf', gamma=0.7).fit(X\_train,y\_train) ****

clf1 = svm.SVC(C=1,kernel='linear', decision\_function\_shape='ovr').fit(X\_train,y\_train)  
clf2 = svm.SVC(C=1, kernel='rbf', gamma=0.7).fit(X\_train,y\_train)

****

手写数字识别

*from* sklearn *import* svm  *# 导入支持向量机模型*

*import* numpy *as* np  *# 导入数值计算库*

*import* matplotlib.pyplot *as* plt  *# 导入绘图库*

*import* matplotlib *as* mpl  *# 导入matplotlib库*

*from* matplotlib *import* colors  *# 导入颜色管理模块*

*from* sklearn *import* model\_selection  *# 导入模型选择模块，用于数据集划分*

*from* sklearn.datasets *import* load\_digits  *# 导入手写数字数据集*

def show\_num() :

*"""*

*显示手写数字数据集中的前四个样本图像*

*"""*

    digits = load\_digits()  *# 加载手写数字数据集*

    plt.gray()  *# 设置灰度模式显示图像*

    plt.matshow(digits.images[0])  *# 显示第1个数字图像*

    plt.matshow(digits.images[1])  *# 显示第2个数字图像*

    plt.matshow(digits.images[2])  *# 显示第3个数字图像*

    plt.matshow(digits.images[3])  *# 显示第4个数字图像*

    plt.show()  *# 展示图像*

def load\_data():

*"""*

*加载手写数字数据集并进行预处理*

*返回：数据特征矩阵和标签向量*

*"""*

    data\_set = []

    data\_X = []

    data\_y = []

    digits = load\_digits()  *# 加载手写数字数据集*

*# print(digits)*

    data\_X.append(digits.data)  *# 获取特征数据*

    data\_y.append(digits.target)  *# 获取目标分类标签*

    data\_X = np.array(data\_X)  *# 转换为numpy数组*

    data\_X = np.reshape(data\_X, (data\_X.shape[1], data\_X.shape[2]))  *# 重塑特征数据形状*

    data\_y = np.array(data\_y)  *# 转换为numpy数组*

    data\_y = np.reshape(data\_y, data\_y.shape[1])  *# 重塑标签数据形状*

*# 每一个存在8\*8 = 64 个特征值*

*# 其中的特征值代表的含义是 其灰度值*

*return* data\_X, data\_y

def split\_data(*data\_X*, *data\_y*) :

*"""*

*将数据集分割为训练集和测试集*

*参数：*

*data\_X: 特征矩阵*

*data\_y: 标签向量*

*返回：训练集特征、测试集特征、训练集标签、测试集标签*

*"""*

*# 使用train\_test\_split函数划分数据集，测试集占20%，随机种子为0*

    X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(*data\_X*, *data\_y*, *test\_size*=0.2, *random\_state*=0)

*# 将特征数据转换为浮点数类型*

*for* index, item *in* enumerate(X\_train) :

        X\_train[index] = list(map(float, item))

*for* index, item *in* enumerate(X\_test):

        X\_test[index] = list(map(float, item))

*# 转换为numpy数组*

    X\_train = np.array(X\_train)

    X\_test = np.array(X\_test)

    y\_train = np.array(y\_train)

    y\_test = np.array(y\_test)

*return* X\_train, X\_test, y\_train, y\_test

*if* \_\_name\_\_ == '\_\_main\_\_':

*# 加载数据集*

    data\_X, data\_y = load\_data()

*# 分割数据集为训练集和测试集*

    X\_train, X\_test, y\_train, y\_test = split\_data(data\_X, data\_y)

*# 创建并训练SVM模型*

*# RBF核函数SVM模型，C=1，gamma=0.001*

    clf1 = svm.SVC(*C*=1, *kernel*='rbf', *gamma*=0.001, *random\_state*=0).fit(X\_train, y\_train)

*# 线性核函数SVM模型，C=12，使用"一对其余"(OVR)决策函数形状*

    clf2 = svm.SVC(*C*=12, *kernel*='linear', *decision\_function\_shape*='ovr').fit(X\_train, y\_train)

*# 评估模型性能*

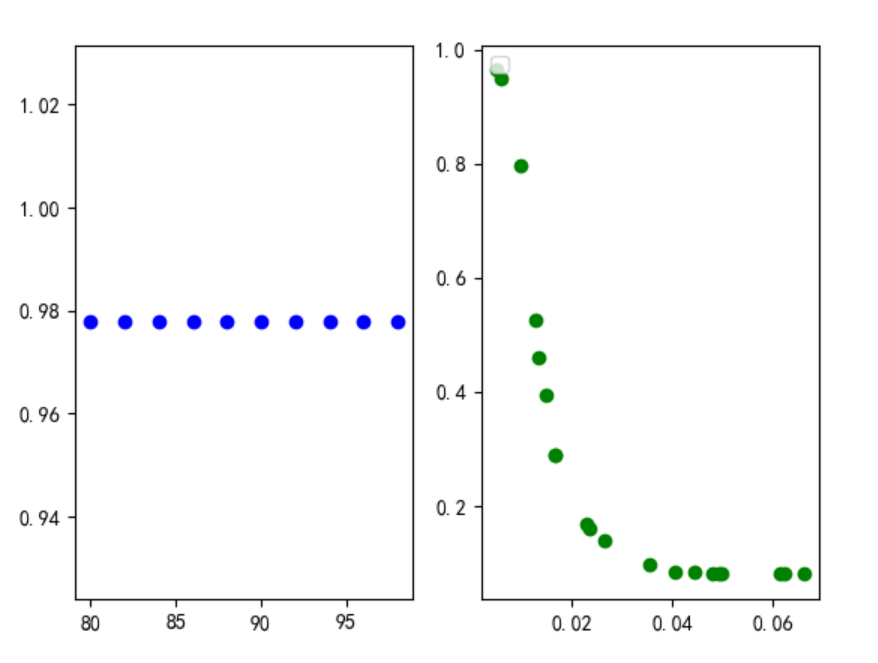
    print("linear 线性核函数-训练集", clf2.score(X\_test, y\_test))  *# 输出线性核函数在测试集上的准确率*

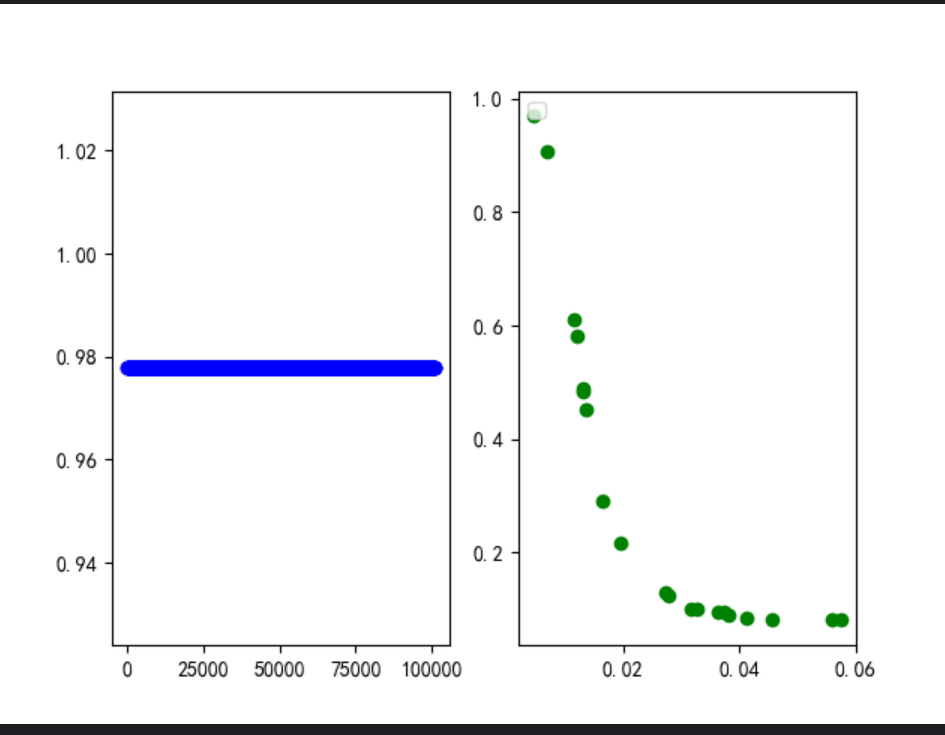
    print("RBF 核函数-训练集", clf1.score(X\_test, y\_test))  *# 输出RBF核函数在测试集上的准确率*

clf1 = svm.SVC(C=1,kernel='rbf',gamma=0.001,random\_state=0).fit(X\_train,y\_train)  
clf2 = svm.SVC(C=12,kernel='linear',decision\_function\_shape='ovr').fit(X\_train,y\_train)



左图为线性核函数（其x轴为c），右图为REF核函数（gamma值为x轴，其c=1）：





**四、[附加题].**利用支持向量机解决**自己遇到的问题**。

[解题要求同题2]

*from* sklearn *import* svm  *# 导入sklearn的SVM实现*

*import* numpy *as* np  *# 导入NumPy用于数值计算*

*import* matplotlib.pyplot *as* plt  *# 导入绘图工具*

*import* matplotlib *as* mpl  *# 导入matplotlib库*

*from* matplotlib *import* colors  *# 导入颜色模块*

*from* sklearn *import* model\_selection  *# 导入模型选择工具*

*from* cvxopt *import* matrix, solvers  *# 导入凸优化求解器*

def train(*X\_train*, *y*, *C*):

*"""*

*使用凸优化方法训练线性SVM模型*

*参数：*

*X\_train: 训练集特征矩阵*

*y: 训练集标签向量*

*C: 正则化参数*

*返回：*

*w\_pre: 计算得到的权重向量*

*b\_pre: 计算得到的偏置*

*"""*

*# 计算训练样本间的内积矩阵*

    m = []

*for* i *in* range(*X\_train*.shape[0]):

        m.append([])

*for* j *in* range(*X\_train*.shape[0]):

            m[i].append(np.inner(*X\_train*[i], *X\_train*[j]))

    m = np.array(m)

*# 计算标签向量的内积*

    r = np.inner(*y*, *y*)

*# 定义凸优化问题的目标函数和约束条件*

    p = matrix(r \* m)  *# 目标函数的二次项*

    q = matrix(np.ones(40) \* -1)  *# 目标函数的一次项*

    A = matrix(*y*.reshape(1, -1))  *# 等式约束条件*

    b = matrix(0.)  *# 等式约束的常数项*

*# 不等式约束条件*

    g = matrix(np.vstack((np.eye(40) \* -1, np.eye(40))))

    h = matrix(np.vstack((np.zeros(len(*y*)).reshape(-1, 1), np.ones(len(*y*)).reshape(-1, 1) \* *C*)))

*# 求解凸优化问题*

    solution = solvers.qp(p, q, g, h, A, b)

*# 提取拉格朗日乘子*

    lamda = np.ravel(solution['x'])

*# 计算权重向量w和偏置b*

    w\_pre = np.sum(lamda.reshape(-1, 1) \* *y* \* *X\_train*, *axis*=0)

    b\_pre = 0

*for* i *in* range(*X\_train*.shape[0]):

        b\_pre += *y*[i] - np.sum(*y* \* lamda.reshape(-1, 1) \* np.inner(*X\_train*, *X\_train*[i].T).reshape(-1, 1))

    b\_pre = b\_pre / *X\_train*.shape[0]

*return* w\_pre, b\_pre

*# 生成二维正态分布的样本数据*

mu = np.array([3, 5])  *# 第二类数据的均值*

sigma = np.array([[1, 0], [0,2]])  *# 第二类数据的协方差矩阵*

L = np.linalg.cholesky(sigma)  *# 协方差矩阵的Cholesky分解*

sigma=np.array([2,4])

X1 = np.random.normal(0,1,(20,2))  *# 生成第一类数据：均值为0的标准正态分布*

*# 绘制第一类数据的散点图*

plt.plot(X1[:,0], X1[:,1], 'o')

X2 = np.dot(np.random.randn(20, 2), L.T) + mu  *# 生成第二类数据：均值为mu的正态分布*

plt.plot(X2[:,0], X2[:,1], '+')  *# 绘制第二类数据的散点图*

*# 构建标签向量：第一类为1，第二类为-1*

y=[]

*for* i *in* range(len(X1)):

    y.append(1)

*for* i *in* range(len(X2)):

    y.append(-1)

*# 合并两类数据*

X=[]

*for* i *in* range(len(X1)):

    X.append(X1[i])

*for* i *in* range(len(X2)):

    X.append(X2[i])

*# 转换为NumPy数组*

X\_train=np.array(X)

y\_train1=np.array(y)

y\_p1 = np.ones(len(X)//2)[:,np.newaxis]

y\_p2 = (np.ones(len(X)//2)\*-1)[:,np.newaxis]

y\_train=np.vstack((y\_p1,y\_p2))

*# 使用自定义的train函数训练SVM模型*

w, b = train(X\_train,y\_train,1000000000)  *# 使用非常大的C值接近硬间隔SVM*

*# 计算分离超平面*

X = np.linspace(-4, 8 , 40)  *# 用于绘制分离超平面的x轴数据*

y = (-w[0]/w[1]\*X - b/w[1]).ravel()  *# 计算分离超平面的y轴数据*

*# 绘制数据点和分离超平面*

[[plt.scatter(x[0], x[1], *color* = \"r\", s = 50)] for x in X\_train[:20]]  # 绘制第一类数据点（红色）

[[plt.scatter(x[0], x[1], *color* = \"b\", s = 50)] for x in X\_train[20:]]  # 绘制第二类数据点（蓝色）

plt.plot(X, y, *color*=\"g\")  # 绘制分离超平面（绿色线）

plt.show()  *# 显示图形*

**数据结果：**

0:  4.3867e+16 -1.4946e+19  3e+19  4e-01  4e-06

1:  1.1149e+17 -2.4665e+18  4e+18  4e-02  2e+00

2:  1.1204e+17 -3.3988e+17  6e+17  5e-03  2e-01

3:  2.9354e+16 -5.5375e+16  8e+16  4e-08  1e-06

4:  4.6327e+15 -5.8015e+15  1e+16  5e-08  7e-07

5:  6.6733e+14 -7.4886e+14  1e+15  2e-08  2e-07

6:  9.5823e+13 -1.0629e+14  2e+14  7e-09  8e-08

7:  1.3741e+13 -1.5150e+13  3e+13  2e-09  5e-08

8:  1.9696e+12 -2.1672e+12  4e+12  1e-09  1e-08

9:  2.8226e+11 -3.1035e+11  6e+11  2e-10  3e-09

10:  4.0449e+10 -4.4461e+10  8e+10  3e-10  2e-09

11:  5.7962e+09 -6.3707e+09  1e+10  2e-10  7e-10

12:  8.3055e+08 -9.1290e+08  2e+09  3e-12  2e-10

13:  1.1900e+08 -1.3083e+08  2e+08  5e-12  1e-10

14:  1.7049e+07 -1.8752e+07  4e+07  9e-12  3e-11

15:  2.4414e+06 -2.6887e+06  5e+06  5e-13  1e-11

16:  3.4920e+05 -3.8592e+05  7e+05  3e-13  6e-12

17:  4.9798e+04 -5.5544e+04  1e+05  7e-14  2e-12

18:  7.0424e+03 -8.0521e+03  2e+04  2e-14  9e-13

19:  9.7154e+02 -1.1903e+03  2e+03  2e-14  3e-13

20:  1.2251e+02 -1.8592e+02  3e+02  2e-14  2e-13

21:  7.9552e+00 -3.4395e+01  4e+01  8e-15  4e-14

22: -5.6601e+00 -2.5035e+01  2e+01  3e-15  3e-14

23: -1.9538e+01 -2.3397e+01  4e+00  4e-15  5e-14

24: -2.0823e+01 -2.0864e+01  4e-02  5e-16  6e-14

25: -2.0837e+01 -2.0837e+01  4e-04  4e-15  5e-14

26: -2.0837e+01 -2.0837e+01  4e-06  3e-15  7e-14

