《机器学习》课程

实验报告

班级：电子 214

姓名：刘伟航

学号：213876

实验一 线性回归

一、实验目的

1. 掌握线性回归的基本原理
2. 编程实现简单的线性回归
3. 能够用线性回归解决实际问题
4. 熟悉 python 软件的使用

二、实验步骤

1. 生成数据集
2. 初始化参数值
3. 梯度下降优化参数
4. 得出结果并分析

三、实验原理

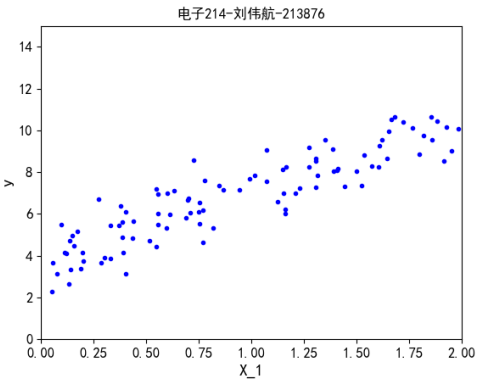
使用 Python 语言，PycharmIDE，Python3.9 编译器，使用 numpy、matplotlib 库和 sklearn 的 iris 数据集进行实验。实验分为四部分 1：random 生成一组线性分布的数据样本，叠加一组 random 随机数，构成一组样本并绘制其分布图；2：根据生成的随机样本，计算回归方程参数，绘制回归方程直线；3：根据一组简单样本构建回归直线并对两个未知点进行预测 4：调用 iris 数据集中的 50 条样本，对花瓣长度、花瓣宽度做成二维数据集，绘制样本点并做出回归曲线。

四、实验过程、结果及分析

1、 构造数据

采用 rand 函数构建一组线性分布的随机样本，首先构造 X 坐标为取值在[0,2)随机分布的 100×1 行向量，给定 Y 与 X 的线性关系，叠加上取值[0,1)的随机数，构成了一组数据量为 100 的样本。用 matplotlib 中的 plot 方法，所设定绘制图像的取值，坐标名、图名、坐标轴范围并绘制样本点。

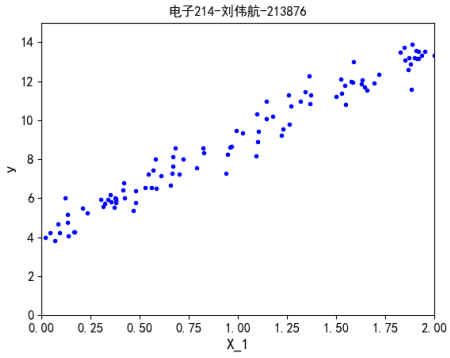
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

X = 2 * np.random.rand(100, 1)
y = 3*X + 4 + np.random.randn(100, 1)
# 构造线性方程，加入随机抖动

plt.plot(X, y, 'b.')
# b 指定为蓝色,.指定线条格式
plt.xlabel('X_1')
plt.ylabel('y')
plt.axis([0, 2, 0, 15])
plt.rcParams['font.sans-serif']    =
['SimHei']
plt.title('电子 214-刘伟航-213876')
plt.show()
```



改变 w 与 b 的值样本分布趋势发生变化。结果如图所示：

通过实验结果可以看出，构造的数据分布大致按照设定的直线参数分布，改变直线参数使样本分布的斜率、截距发生变化；通过改变 rand 函数的

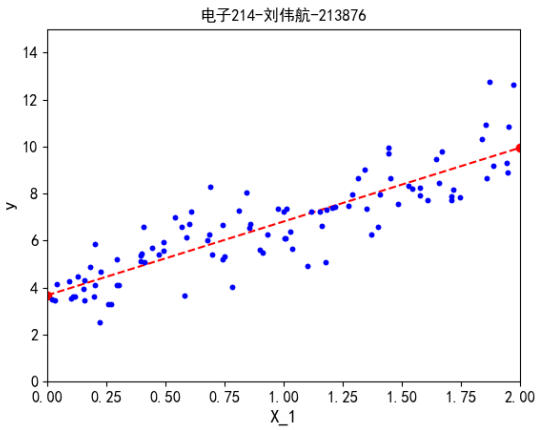
权重可以使样本的分布更加离散或紧密。



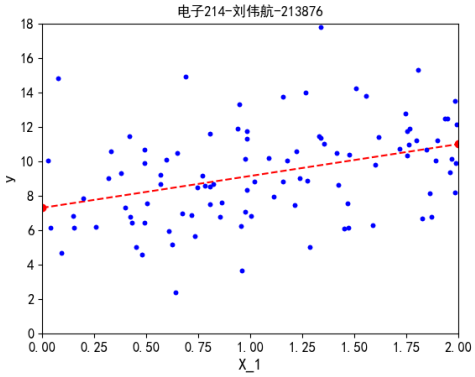
2、 线性方程实现回归

运用 numpy 的矩阵运算，X_b 构建了一个长为 100 的全 1 数组与 X 连接，Theta_best 代表了利用公式$\theta = (X^T X)^{-1} X^T y$计算出的最佳回归方程的参数。最终的回归方程是由 0、2 两点 x 计算该参数对应下 y 的预测值，两点确定一条直线绘制出来的。改变 X 与 y 的数据构建时的参数可以改变随机样本，并绘制其回归曲线。#与’’’分别为单行注释和多行注释的起始符号。

```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

X = 2 * np.random.rand(100, 1)
y = 3*X + 4 + np.random.randn(100, 1)
# 构造线性方程，加入随机抖动
X_b = np.c_[(np.ones((100, 1)), X)]
# np.linalg.inv:矩阵求逆
theta_best =
np.linalg.inv(X_b.T.dot(X_b)).dot(X_
b.T).dot(y)
print(theta_best)

# 测试数据
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)),
X_new]
# 预测结果
y_predict = X_new_b.dot(theta_best)
print(y_predict)

plt.plot(X, y, 'b.')        # b 指定为
蓝色,.指定线条格式
plt.xlabel('X_1')
plt.ylabel('y')
plt.axis([0, 2, 0, 15])
plt.rcParams['font.sans-serif'] =
['SimHei']
plt.title('电子 214-刘伟航-213876')
plt.plot(X_new, y_predict, 'r--o')
# 指定红色和线条
plt.plot(X, y, 'b.')        # 指定蓝色
和点
plt.show()
```



改变生成样本的参数，构造一组样本数据，绘制回归曲线如图：

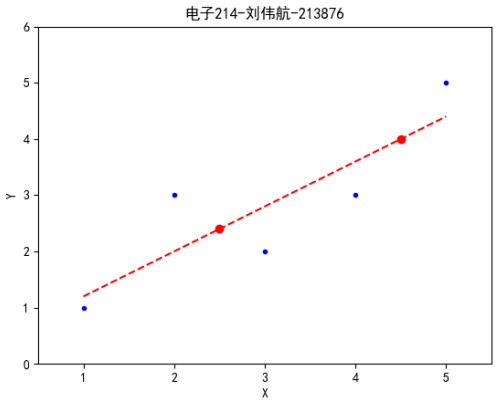


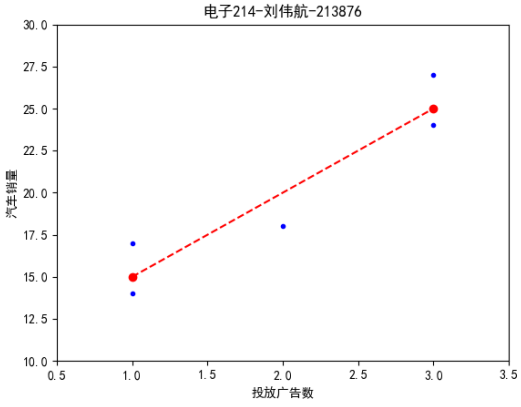
3、 简单线性回归

根据给定的一组数据，采用最小二乘法计算线性回归参数，并绘制图像

$$\begin{cases} y = b_0 x + b_1 \\ b_0 = \dfrac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \\ b_1 = \bar{y} - b_0 \bar{x} \end{cases}$$

```
import numpy as np
import matplotlib.pyplot as plt
x=np.array([1,2,3,4,5])
y=np.array([1,3,2,3,5])
p,q=0,0
ax=sum(x)/len(x)
ay=sum(y)/len(y)
for i in range(len(x)):
    p+=(x[i]-ax)*(y[i]-ay)
    q+=(x[i]-ax)**2
b0=p/q
b1=ay-b0*ax
xx=np.array([min(x),max(x),2.5,4.5])
yy=xx.dot(b0)+b1
plt.plot(x,y,'b.')
plt.plot(xx[:2],yy[:2],'r--')
plt.plot(xx[2:],yy[2:],'ro')
plt.axis([0.5, 5.5, 0, 6])
plt.rcParams['font.sans-serif']     =
['SimHei']
plt.title('电子 214-刘伟航-213876')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

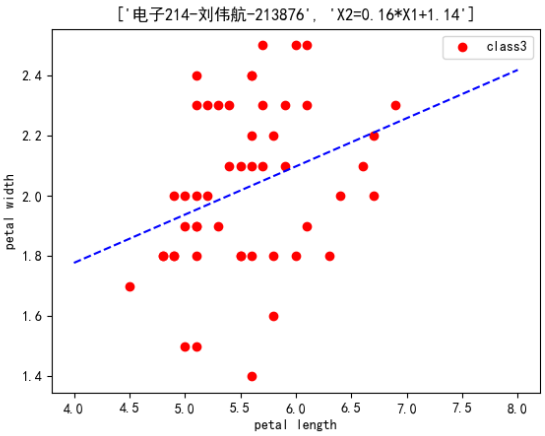




4、 实现鸢尾花数据集给定数据线性回归

从 sklearn 库导入鸢尾花数据集，取数据集后 50 组数据，将花瓣长度、花瓣宽度分别赋值给 x1、x2 数组，利用公式$\theta = (X^T X)^{-1} X^T y$计算回归方程参数，并绘制样本点和回归曲线。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import
load_iris

# 导入鸢尾花数据集
iris = load_iris()
X = iris.data[100:150, 2:4]  ##鸢尾
花数据集后 50 个样本,只取特征空间中后
两个维度
print(len(X))
print(X)
x1=X[0:50, 0]  #第一列数据
print(x1.shape) # 第一列大小为 50
行,1 列
x2=X[0:50, 1]  #第二列数据
print(x2.shape) # 第二列大小为 50
行,1 列
plt.scatter(x1, x2, c="red",
marker='o', label='class3')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc=1)   #图例放置位置,右
上为 1,逆时针排序
plt.rcParams['font.sans-serif'] =
['SimHei']
X_b = np.c_[(np.ones((50, 1)), x1)]
theta_best =
np.linalg.inv(X_b.T.dot(X_b)).dot(X_
b.T).dot(x2)
print(theta_best)
X_new = np.array([[4], [8]])
X_new_b = np.c_[np.ones((2, 1)),
X_new]
y_predict = X_new_b.dot(theta_best)
print(y_predict)
plt.plot(X_new, y_predict, 'b--')
plt.title(['电子 214-刘伟航-
213876','X2='+str(round(theta_best[1
],2))+'*X1+'+str(round(theta_best[0]
,2))])
plt.show()
```



五、实验心得

实验一对线性回归类问题进行了练习，线性回归是机器学习中的有监督学习，回归的目的是招到最符合样本变化趋势规律的曲线方程，通过方程来进行预测或者分类判别。内容主要包括运用 random 生成随机样本，运用最小二乘法求回归方程，运用 matplotlib 库绘图添加图例等，最后练习从库中导入 IRIS 数据集。本次实验让我对线性回归的各个参数的求解方法和意义有了更深的理解，绘图方法和 numpy 数组的运用掌握，numpy 库的数组可以进行更多样更方便的数组或矩阵操作。本次实验实现的都是线性回归，除了线性模型外还可以使用非线性模型。线性模型形式简单、易于建模，但却蕴涵着机器学习中一些重要的基本思想，许多机器学习算法都是这样以一个中心思想逐渐扩展算法实现更加丰富高级的功能。

实验二 Fisher 线性判别

一、实验目的

1. 熟悉 Python 的基础使用
2. 理解和掌握 Fisher 线性判别法的基本原理和实现过程
3. 利用 Fisheries 判别法解决实际问题

二、实验步骤

1. 输入数据集
2. 计算关键参数值
3. 对测试数据集进行分类判别
4. 绘制结果

三、实验原理

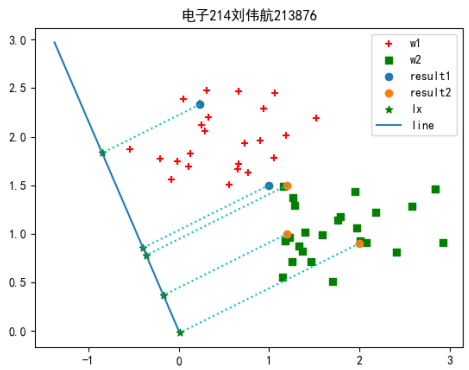
Fisher 线性判别的基本思想是：把 d 维空间的所有样本投影到一条直线上，就能把维数压缩到 1。关键在于要找到这样一条最优的投影方向，使这些模式的投影能较好地区分开。

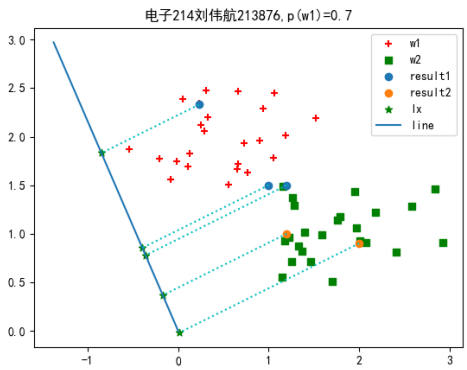
两类样本投影的均值彼此间相距尽可能大；使同类样本的投影彼此间尽可能密集(类内离散度越小越好)。类内紧，类间开

Fisher 线性判别分析，就是通过给定的训练数据，确定投影方向 w 和阈值 yo，即确定线性判别函数，然后根据这个线性判别函数，对测试数据进行测试，得到测试数据的类别。

四、实验过程、结果及分析

任务一：





（2）写出各个参数（如均值，类内类间离散度矩阵等）值的含义

均值:m1、m2,为样本中所有数据相加除以样本的总个数，表示改组样本 dev 中心值；

类内离散度矩阵:s1、s2,类内协方差矩阵，衡量类内样本的聚集程度，我们

希望其值越小越好；
最优投影方向:w_new,按照此方向画出的投影直线，将两类样本点投影到这条直线上后，与其他投影方向相比，分得最开即分类效果最好。
(3)阈值采用代码 w0=(m1_new+m2_new)/2-math.log(pw1/pw2)/(24+24-2)计算
代码：

```
#task1
import numpy as np
import math
import matplotlib.pyplot as plt
x1=[0.23,1.52,0.65,0.77,1.05,1.19,0.29,0.25,
0.66,0.56,0.90,0.13,-0.54,0.94,-0.21,0.05,-
0.08,0.73,0.33,1.06,-0.02,0.11,0.31,0.66]
y1=[2.34,2.19,1.67,1.63,1.78,2.01,2.06,2.12,
2.47,1.51,1.96,1.83,1.87,2.29,1.77,2.39,1.56,
1.93,2.20,2.45,1.75,1.69,2.48,1.72]
x2=[1.40,1.23,2.08,1.16,1.37,1.18,1.76,1.97,
2.41,2.58,2.84,1.95,1.25,1.28,1.26,2.01,2.18,
1.79,1.33,1.15,1.70,1.59,2.93,1.46]
y2=[1.02,0.96,0.91,1.49,0.82,0.93,1.14,1.06,
0.81,1.28,1.46,1.43,0.71,1.29,1.37,0.93,1.22,
1.18,0.87,0.55,0.51,0.99,0.91,0.71]
#将矩阵整合为 w1、w2
w1=[[0 for i in range(2)]for i in range(24)]
w2=[[0 for i in range(2)]for i in range(24)]
for i in range(24):
    w1[i][0]=x1[i]
    w1[i][1]=y1[i]
    w2[i][0]=x2[i]
    w2[i][1]=y2[i]
print('整合矩阵 w1 w2')
print('w1=',w1)
print('w2=',w2)
#计算两类均值向量
m1=np.mean(w1,0)
#mean(matrix,axis=0),matrix 填写一个矩阵，
axis 0 代表：压缩行，对各列求均值
m2=np.mean(w2,0)
#axis 1 代表：压缩列，对各行求均值
print('计算两类均值向量')
print('m1=',m1)
print('m2=',m2)

#计算总的类内离散度矩阵 Sw=s1+s2
s10=[0,0]
s20=[0,0]
s1=[[0 for i in range(2)]for j in range(2)]#2*2
s2=[[0 for i in range(2)]for j in range(2)]
for i in range(24):#这里要注意矩阵的转置
    s10[0]=(w1[i][0]-m1[0])
    s10[1]=(w1[i][1]-m1[1])
    s11=np.mat(s10)#将 list 变为矩阵
    s1+=np.mat((s11.T)*s11)#这里和书上
公式相反，因为设置的时候和书上不一样，
想到得到 2*2 的矩阵就必须换个方向
    s20[0]=(w2[i][0]-m2[0])
    s20[1]=(w2[i][1]-m2[1])
    s22=np.mat(s20)
    s2+=np.mat((s22.T)*s22)
print('s1')
print(s1)
print('s2')
print(s2)
sw=s1+s2
print('sw')
print(sw)

#计算投影方向和阈值
w_new=(np.mat(sw)).I*(np.mat((m1-m2)).T)

print(w_new)
#这里因为考虑先验概率
m1_new=m1*w_new#这里的顺序很重要，
因为前面设置的时候没有注意，所以写的
时候要注意一下
m2_new=m2*w_new
pw1=0.7
pw2=0.3
w0=(m1_new+m2_new)/2-
math.log(pw1/pw2)/(24+24-2)
print('w0')
print(w0)

#对测试数据进行分类判别
```

```
x=[[1,1.5],[1.2,1.0],[2.0,0.9],[1.2,1.5],[0.23,
2.33]]
result1=[]
result2=[]
for i in range(5):
    y=np.mat(x[i])*w_new#这里的顺序依然要小心
    if y>w0[0][0]:
        result1.append(x[i])
    else:
        result2.append(x[i])
print('result1')
print(result1)
print('result2')
print(result2)

#计算试验点在 w_new 方向上的点
w_k=np.mat(np.zeros((2,1)))#归一化
w_k[0]=w_new[0]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))# 使用二范数进行归一化
w_k[1]=w_new[1]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))
print(w_k)
wd=np.mat(np.zeros((2,5)))
for i in range(5):
    wd[:,i]=(np.mat(x[i])*(w_k*w_k.T)).T
print('wd')
print(wd)
#显示分类结果
mw1=np.mat(w1)
mw2=np.mat(w2)
mr1=np.mat(result1)
mr2=np.mat(result2)
p1=plt.scatter(mw1[:,0].tolist(),mw1[:,1].toli
st(),c='red',marker='+')#画出 w1 类的各点
p2=plt.scatter(mw2[:,0].tolist(),mw2[:,1].toli
st(),c='green',marker='s')#画出 w2 类的各点
p3=plt.scatter(mr1[:,0].tolist(),mr1[:,1].tolist
())#画出测试集中属于 w1 的各点
p4=plt.scatter(mr2[:,0].tolist(),mr2[:,1].tolist
())#画出测试集中属于 w2 的各点
p5,
=plt.plot([0,10*w_new[0,0]],[0,10*w_new[1
,0]])#画出最佳投影方向
p6=plt.scatter(wd.T[:,0].tolist(),wd.T[:,1].tol
ist(),c='g',marker='*')#画出测试集各点在投影方向上的投影点 for i in range(5):
for i in range(5):

p7=plt.plot([x[i][0],wd.T[i,0].tolist()],[x[i][1
],wd.T[i,1].tolist()],'c:')
plt.legend([p1,p2,p3,p4,p6,p5],['w1','w2','res
ult1','result2','lx','line'])
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title('电子 214 刘伟航 213876,p(w1)=0.7')
plt.show()
```

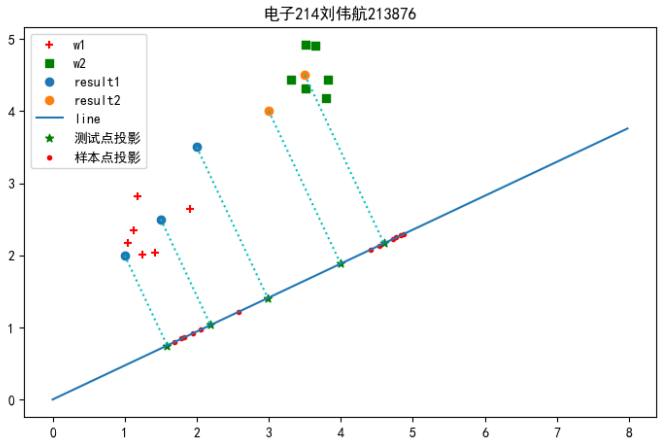
任务二：

代码：

```
#task2
import numpy as np
import math
import matplotlib.pyplot as plt
x1=[1.1233,1.1839,1.2400,1.4173,1.0497,1.
90278]
y1=[2.3532,2.8212,2.0154,2.0430,2.1690,2.
64918]
x2=[3.3063,3.5085,3.5108,3.8176,3.7948,3.
64437]
y2=[4.4357,4.3111,4.9234,4.4302,4.1848,4.
90490]
#将矩阵整合为 w1、w2
w1=[[0 for i in range(2)]for i in range(6)]
w2=[[0 for i in range(2)]for i in range(6)]
for i in range(6):
```

```
        w1[i][0]=x1[i]
        w1[i][1]=y1[i]
        w2[i][0]=x2[i]
        w2[i][1]=y2[i]
print('整合矩阵 w1 w2')
print('w1=',w1)
print('w2=',w2)
#计算两类均值向量
m1=np.mean(w1,0)
#mean(matrix,axis=0),matrix 填写一个矩阵，
axis 0 代表：压缩行，对各列求均值
m2=np.mean(w2,0)
#axis 1 代表：压缩列，对各行求均值
print('计算两类均值向量')
print('m1=',m1)
print('m2=',m2)

#计算总的类内离散度矩阵 Sw=s1+s2
s10=[0,0]
s20=[0,0]
s1=[[0 for i in range(2)]for j in range(2)]#2*2
s2=[[0 for i in range(2)]for j in range(2)]
for i in range(5):#这里要注意矩阵的转置
        s10[0]=(w1[i][0]-m1[0])
        s10[1]=(w1[i][1]-m1[1])
        s11=np.mat(s10)#将 list 变为矩阵
        s1+=np.mat((s11.T)*s11)#这里和书上
公式相反，因为设置的时候和书上不一样，
想到得到 2*2 的矩阵就必须换个方向
        s20[0]=(w2[i][0]-m2[0])
        s20[1]=(w2[i][1]-m2[1])
        s22=np.mat(s20)
        s2+=np.mat((s22.T)*s22)
print('s1')
print(s1)
print('s2')
print(s2)
sw=s1+s2
print('sw')
print(sw)

#计算投影方向和阈值
w_new=(np.mat(sw)).I*(np.mat((m1-m2)).T)
print('w_new')
print(w_new)
#这里因为考虑先验概率
m1_new=m1*w_new#这里的顺序很重要，
因为前面设置的时候没有注意，所以写的
时候要注意一下
m2_new=m2*w_new
pw1=0.5
pw2=0.5
w0=(m1_new+m2_new)/2-
math.log(pw1/pw2)/(6+6-2)
print('w0')
print(w0)

#对测试数据进行分类判别
x=[[1.5,2.5],[1,2],[2,3.5],[3,4],[3.5,4.5]]
result1=[]
result2=[]
for i in range(5):
        y=np.mat(x[i])*w_new#这里的顺序依
然要小心
        if y>w0[0][0]:
                result1.append(x[i])
        else:
                result2.append(x[i])
print('result1')
print(result1)
print('result2')
print(result2)
#计算试验点在 w_new 方向上的点
w_k=np.mat(np.zeros((2,1)))#归一化
w_k[0]=w_new[0]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))# 使用二
范数进行归一化
w_k[1]=w_new[1]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))
print(w_k)
wd=np.mat(np.zeros((2,5)))
for i in range(5):
        wd[:,i]=(np.mat(x[i])*(w_k*w_k.T)).T
print('wd')
print(wd)

samples=[[x1[i],y1[i]] for i in
range(len(x1))]+[[x2[i],y2[i]] for i in
```

```
range(len(x2))]#计算所有样本点在投影方向上的投影点
F=np.mat(np.zeros((2,1)))#归一化
F[0]=w_new[0]/(np.linalg.norm(w_new,ord=2,axis=None,keepdims=False))#使用二范数进行归一化
F[1]=w_new[1]/(np.linalg.norm(w_new,ord=2,axis=None,keepdims=False))
ws=np.mat(np.zeros((2,12)))
for i in range(12):
    ws[:,i]=(np.mat(samples[i])*(F*F.T)).T


#显示分类结果
mw1=np.mat(w1)
mw2=np.mat(w2)
mr1=np.mat(result1)
mr2=np.mat(result2)
p1=plt.scatter(mw1[:,0].tolist(),mw1[:,1].tolist(),c='red',marker='+')#画出 w1 类的各点
p2=plt.scatter(mw2[:,0].tolist(),mw2[:,1].tolist(),c='green',marker='s')#画出 w2 类的各点
p3=plt.scatter(mr1[:,0].tolist(),mr1[:,1].tolist())#画出测试集中属于 w1 的各点
p4=plt.scatter(mr2[:,0].tolist(),mr2[:,1].tolist())#画出测试集中属于 w2 的各点
p5,          =plt.plot([0,-1*w_new[0,0]],[0,-1*w_new[1,0]])#画出最佳投影方向
p6=plt.scatter(wd.T[:,0].tolist(),wd.T[:,1].tolist(),c='g',marker='*')#画出测试集各点在投影方向上的投影点
for i in range(5):
p7=plt.plot([x[i][0],wd.T[i,0].tolist()],[x[i][1],wd.T[i,1].tolist()],'c:')#画出测试点投影线
p8=plt.scatter(ws.T[:,0].tolist(),ws.T[:,1].tolist(),c='r',marker='.')
plt.legend([p1,p2,p3,p4,p5,p6,p8],['w1','w2','result1','result2','line','测试点投影','样本点投影'])
plt.gca().set_aspect('equal')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title('电子 214 刘伟航 213876')
plt.show()
```



任务三：

利用样本中的前两个特征（长度和宽度特征分别作为横轴和纵轴，用不同的颜色标记不同品种的花）

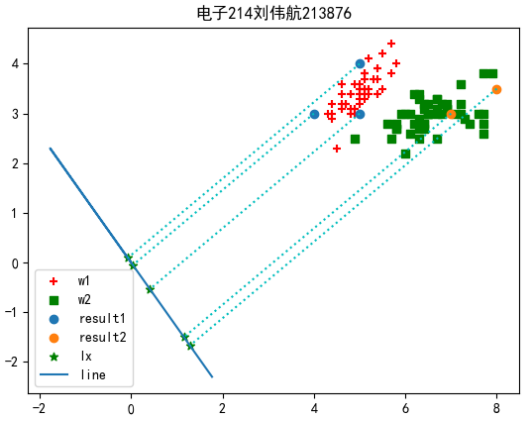
要求：提取数据库中的前 50 个数据和后 50 个数据做为两类数据集，绘图。设计 fisher 判别器，将两类分开，显示结果，要求画出样本集、投影。

代码：

```
#task3
from sklearn.datasets import load_iris
# 导入鸢尾花数据集
iris = load_iris()
import numpy as np
import math
import matplotlib.pyplot as plt
#将矩阵整合为 w1、w2
x1=iris.data[0:50,0]
y1=iris.data[0:50,1]
x2=iris.data[100:150,0]
```

```
y2=iris.data[100:150,1]
w1=[[0 for i in range(2)]for i in range(50)]
w2=[[0 for i in range(2)]for i in range(50)]
for i in range(50):
    w1[i][0]=x1[i]
    w1[i][1]=y1[i]
    w2[i][0]=x2[i]
    w2[i][1]=y2[i]
print('整合矩阵 w1 w2')
print('w1=',w1)
print('w2=',w2)
#计算两类均值向量
m1=np.mean(w1,0)
#mean(matrix,axis=0),matrix 填写一个矩阵，
axis 0 代表：压缩行，对各列求均值
m2=np.mean(w2,0)
#axis 1 代表：压缩列，对各行求均值
print('计算两类均值向量')
print('m1=',m1)
print('m2=',m2)

#计算总的类内离散度矩阵 Sw=s1+s2
s10=[0,0]
s20=[0,0]
s1=[[0 for i in range(2)]for j in range(2)]#2*2
s2=[[0 for i in range(2)]for j in range(2)]
for i in range(24):#这里要注意矩阵的转置
    s10[0]=(w1[i][0]-m1[0])
    s10[1]=(w1[i][1]-m1[1])
    s11=np.mat(s10)#将 list 变为矩阵
    s1+=np.mat((s11.T)*s11)#这里和书上
公式相反，因为设置的时候和书上不一样，
想到得到 2*2 的矩阵就必须换个方向
    s20[0]=(w2[i][0]-m2[0])
    s20[1]=(w2[i][1]-m2[1])
    s22=np.mat(s20)
    s2+=np.mat((s22.T)*s22)
print('s1')
print(s1)
print('s2')
print(s2)
sw=s1+s2
print('sw')
print(sw)

#计算投影方向和阈值
w_new=(np.mat(sw)).I*(np.mat((m1-m2)).T)
print('w_new')
print(w_new)
#这里因为考虑先验概率
m1_new=m1*w_new#这里的顺序很重要，
因为前面设置的时候没有注意，所以写的
时候要注意一下
m2_new=m2*w_new
pw1=0.5
pw2=0.5
w0=(m1_new+m2_new)/2-
math.log(pw1/pw2)/(24+24-2)
print('w0')
print(w0)

#对测试数据进行分类判别
x=[[4,3],[7,3],[8,3.5],[5,4],[5,3]]
result1=[]
result2=[]
for i in range(5):
    y=np.mat(x[i])*w_new#这里的顺序依
然要小心
    if y>w0[0][0]:
        result1.append(x[i])
    else:
        result2.append(x[i])
print('result1')
print(result1)
print('result2')
print(result2)

#计算试验点在 w_new 方向上的点
w_k=np.mat(np.zeros((2,1)))#归一化
w_k[0]=w_new[0]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))# 使用二
范数进行归一化
w_k[1]=w_new[1]/(np.linalg.norm(w_new,o
rd=2,axis=None,keepdims=False))
print(w_k)
wd=np.mat(np.zeros((2,5)))
for i in range(5):
    wd[:,i]=(np.mat(x[i])*(w_k*w_k.T)).T
```

```
print('wd')
print(wd)

#显示分类结果
mw1=np.mat(w1)
mw2=np.mat(w2)
mr1=np.mat(result1)
mr2=np.mat(result2)
p1=plt.scatter(mw1[:,0].tolist(),mw1[:,1].tolist(),c='red',marker='+')#画出 w1 类的各点
p2=plt.scatter(mw2[:,0].tolist(),mw2[:,1].tolist(),c='green',marker='s')#画出 w2 类的各点
p3=plt.scatter(mr1[:,0].tolist(),mr1[:,1].tolist())#画出测试集中属于 w1 的各点
p4=plt.scatter(mr2[:,0].tolist(),mr2[:,1].tolist())#画出测试集中属于 w2 的各点
p5, =plt.plot([0,10*w_new[0,0],-10*w_new[0,0]],[0,10*w_new[1,0],-10*w_new[1,0]])#画出最佳投影方向
p6=plt.scatter(wd.T[:,0].tolist(),wd.T[:,1].tolist(),c='g',marker='*')#画出测试集各点在投影方向上的投影点
for i in range(5):
    p7=plt.plot([x[i][0],wd.T[i,0].tolist()],[x[i][1],wd.T[i,1].tolist()],'c:')
plt.legend([p1,p2,p3,p4,p6,p5],['w1','w2','result1','result2','lx','line'])
#plt.legend([p5],['line'])
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title('电子 214 刘伟航 213876')
plt.gca().set_aspect('equal')
plt.show()
```



五、实验心得

实验二对 FISHER 线性判别进行了练习，线性判别法的基本思想是寻找一个投影方向将高维问题降低到一维空间来解决，线性判别也是最具代表性的有监督学习方法，本次实验对样本进行了很多变化，包括将样本点投影在某一方向上。代码中给出的最佳投影方向是一个单位长度为 1 的二维向量，因此在绘制可视化的方向时使直线过原点并对直线进行延长，在自定义样本点时就出现因为对向量延长的长度太长导致绘图不美观，需要不断调整参数，此外我将测试样本投影方向上的投影点与样本点相连，起初发现投影方向并不与连线垂直且每一点都存在相同的偏移，经过排查是否存在错误的计算过程，计算误差，计算方法都没有发现错误，我注意到绘图区的横纵坐标单位长度不同，plt.gca().set_aspect('equal')，加入该条命令后使得坐标等长，就可以观察到垂直投影了。

Fisher 线性判别法计算简单，物理意义明确，当两类样本数据同先验，满足高斯分布且协方差相等时，可达到最优分类。同时 Fisher 线性判别法也存在缺点，没有利用样本分布的信息

实验三 支持向量机

一、实验目的

1. 掌握支持向量机的基本原理
2. 编程实现简单的支持向量机
3. 掌握支持向量机里的关键语句
4. 能够用支持向量机解决实际间题

二、实验步骤

1.生成数据集
2.调用支持向量机函数
3.绘图
4.得出结果并分析不同参数对结果的影响

三、实验原理

支持向量机是一种分类监督机器学习算法，可用于分类或回归。 然而，它主要用于分类问题。在这个算法中，我们将每一个数据项作为一个点在 n 维空间中。作为一个点，每一个特征值都是一个特定坐标的值。然后，通过查找区分这两个类的超平面来进行分类。SVM 学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。w•x+b=0 即为分离超平面，几何间隔最大的分离超平面却是唯一的。

四、实验过程、结果及分析

任务一：自定义数据实现 SVM 分类

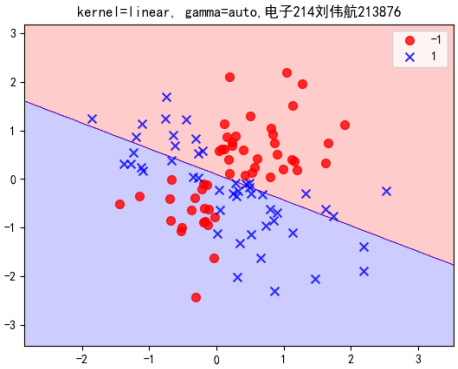
（1）仔细阅读下面程序，运行程序，理解程序实现的功能；

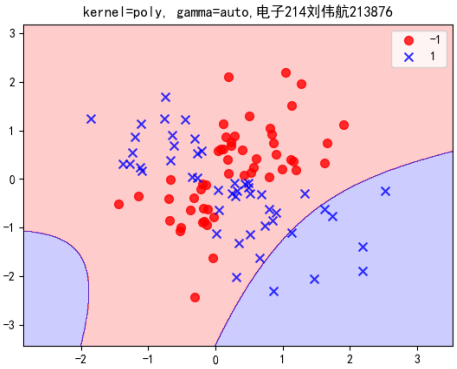
代码首先设置随机种子生成了 100 个二维样本，用异或函数，将数据按一三象限和二四象限分开为两类，构建决策边界，使用高斯核函数作为支持向量机的核函数将两类样本进行划分，将类别两两之间进行划分，用二分类的方法模拟多分类的结果。一三象限类的数据用红色圈出来，二四象限类的数据用蓝色圈出来。

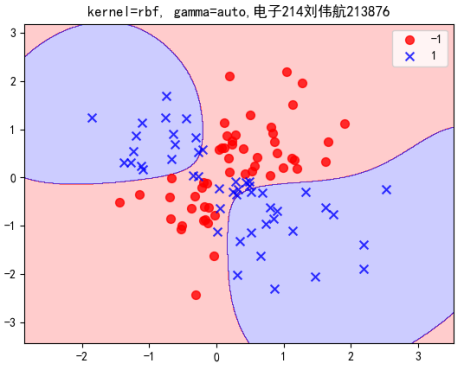
（2）SVC 的参数 kernel 选择核函数；gamma 为核函数系数，只对'rbf'、'poly'、'sigmoid'三种核函数起作用，默认值 auto 为样本特征数的倒数；C 为为惩罚系数，C 越大代表这个分类器对在边界内的噪声点的容忍度越小，分类准确率高，但是容易过拟合，泛化能力差

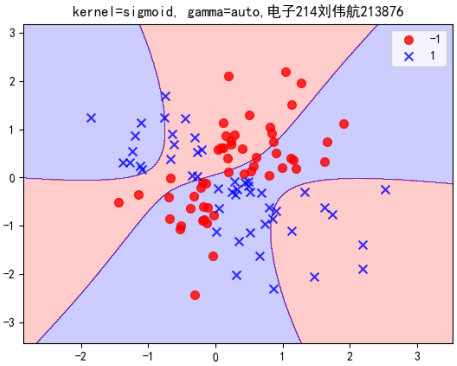
（3）程序和结果如下：

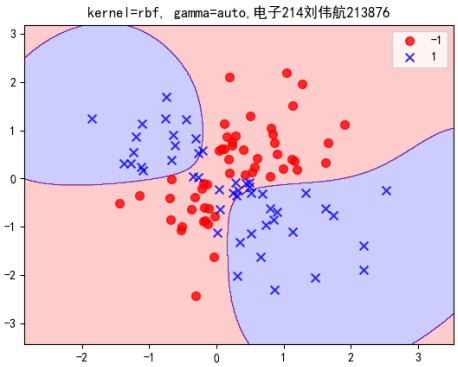
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import
ListedColormap
from sklearn import datasets
from sklearn.svm import SVC
# 以下为创建数据 X 和标签 y
np.random.seed(1)
# 创建 100 个二维数组，即 100 个 2 个
特征的样本
X = np.random.randn(100, 2)
# np.logical_xor(bool1, bool2)，异或逻辑
运算，如果 bool1 和 bool2 的结果相同则
为 False，否则为 True
# ++和--为一三象限，+-和-+为二四象
限，如此做则 100 个样本必定线性不可
分
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)
# 对 X 数组人为分类,二四象限为 True，
即为 1 类；一三象限为 False，即为-1 类
y = np.where(y, 1, -1)
# 构建决策边界,通用程序，后面可直接
使用
def plot_decision_regions(X, y,
classifier=None):
    marker_list = ['o', 'x', 's']
    color_list = ['r', 'b', 'g']
    cmap =
ListedColormap(color_list[:len(np.unique(y)
)])
    x1_min, x1_max = X[:, 0].min() - 1,
X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1,
X[:, 1].max() + 1
    t1 = np.linspace(x1_min, x1_max, 666)
    t2 = np.linspace(x2_min, x2_max, 666)
    x1, x2 = np.meshgrid(t1, t2)
    y_hat =
classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T)
    y_hat = y_hat.reshape(x1.shape)
    plt.contourf(x1, x2, y_hat, alpha=0.2,
cmap=cmap)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    for ind, clas in
enumerate(np.unique(y)):
        plt.scatter(X[y == clas, 0], X[y ==
clas, 1], alpha=0.8, s=50,
                c=color_list[ind],
marker=marker_list[ind], label=clas)

# 以下为关键语句
# for kernel in ['rbf','linear','poly','sigmoid']:
kernel='rbf'
# gamma = 'auto'
for gamma in[10,50,100]:
    svm = SVC(kernel=kernel,
gamma=gamma, C=1, random_state=1)
    svm.fit(X, y)
    plot_decision_regions(X, y,
classifier=svm)
    SVC(C=1, cache_size=200,
class_weight=None,
coef0=0.0,decision_function_shape='ovr',
degree=3, gamma='auto',
kernel='rbf',max_iter=-1, probability=False,
random_state=1, shrinking=True,tol=0.001,
verbose=False)
    plt.title('kernel=%s,
gamma=%s,%s'%(kernel,gamma,'电子 214
刘伟航 213876'))
    plt.legend()
    plt.rcParams['font.sans-serif'] =
['SimHei']
    plt.rcParams['axes.unicode_minus'] =
False
    plt.show()
```

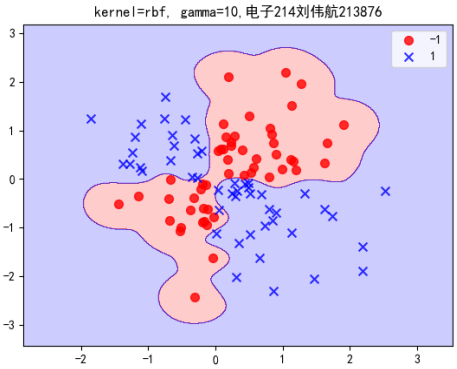

kernel=linear, gamma=auto,电子214刘伟航213876
-1
1

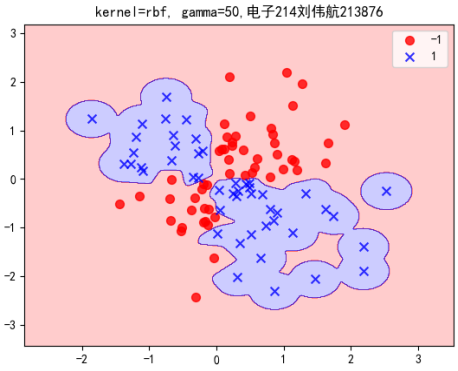

kernel=poly, gamma=auto,电子214刘伟航213876
-1
1

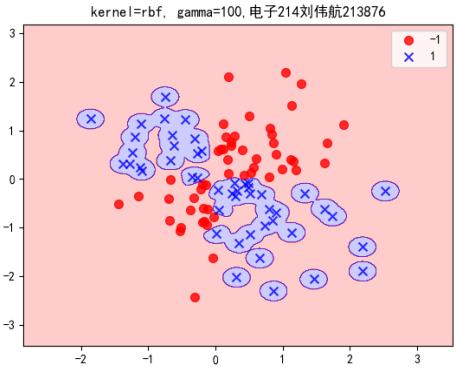

kernel=rbf, gamma=auto,电子214刘伟航213876
-1
1


kernel=sigmoid, gamma=auto,电子214刘伟航213876
-1
1


kernel=rbf, gamma=auto,电子214刘伟航213876
-1
1

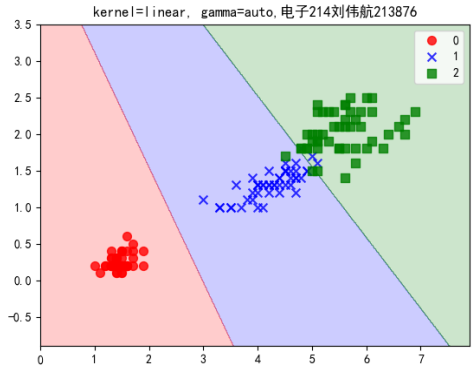

kernel=rbf, gamma=10,电子214刘伟航213876
-1
1

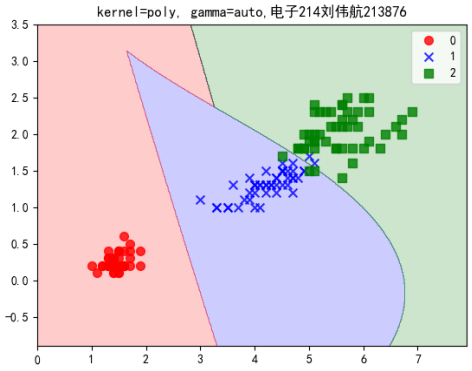

kernel=rbf, gamma=50,电子214刘伟航213876
-1
1

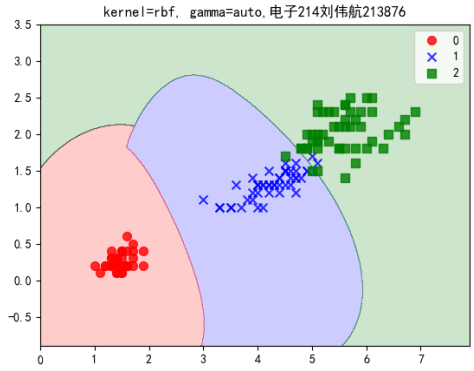

kernel=rbf, gamma=100,电子214刘伟航213876
-1
1

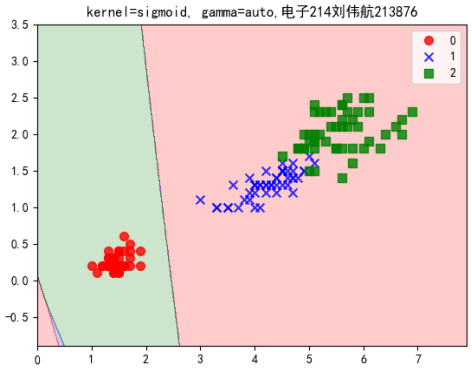
任务二：

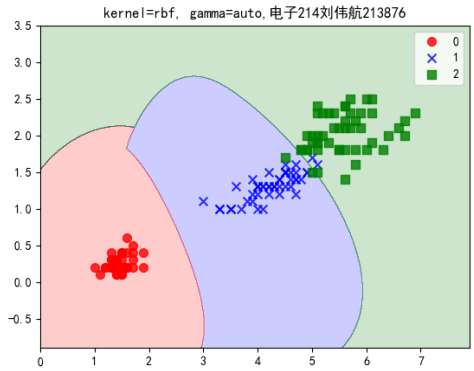
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import
ListedColormap
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.datasets import load_iris
# 导入鸢尾花数据集
iris = load_iris()
# 以下为创建数据 X 和标签 y
X = iris.data[:, 2:4]
y = iris.target[:]
def plot_decision_regions(X, y,
classifier=None):
    marker_list = ['o', 'x', 's']
    color_list = ['r', 'b', 'g']
    cmap =
ListedColormap(color_list[:len(np.unique(y)
)])
    x1_min, x1_max = X[:, 0].min() - 1,
X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1,
X[:, 1].max() + 1
    t1 = np.linspace(x1_min, x1_max, 666)
    t2 = np.linspace(x2_min, x2_max, 666)
    x1, x2 = np.meshgrid(t1, t2)
    y_hat =
classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T)
    y_hat = y_hat.reshape(x1.shape)
    plt.contourf(x1, x2, y_hat, alpha=0.2,
cmap=cmap)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    for ind, clas in
enumerate(np.unique(y)):
        plt.scatter(X[y == clas, 0], X[y ==
clas, 1], alpha=0.8, s=50,
                    c=color_list[ind],
marker=marker_list[ind], label=clas)

# 以下为关键语句
# for kernel in ['rbf','linear','poly','sigmoid']:
kernel='rbf'
    # gamma = 'auto'
for gamma in[10,50,100]:
    svm = SVC(kernel=kernel,
gamma=gamma, C=1, random_state=1)
    svm.fit(X, y)
    plot_decision_regions(X, y,
classifier=svm)
    SVC(C=1, cache_size=200,
class_weight=None,
coef0=0.0,decision_function_shape='ovr',
degree=3, gamma='auto',
kernel='rbf',max_iter=-1, probability=False,
random_state=1, shrinking=True,tol=0.001,
verbose=False)
    plt.title('kernel=%s,
gamma=%s,%s'%(kernel,gamma,'电子 214
刘伟航 213876'))
    plt.legend()
    plt.rcParams['font.sans-serif'] =
['SimHei']
    plt.rcParams['axes.unicode_minus'] =
False
    plt.show()
```

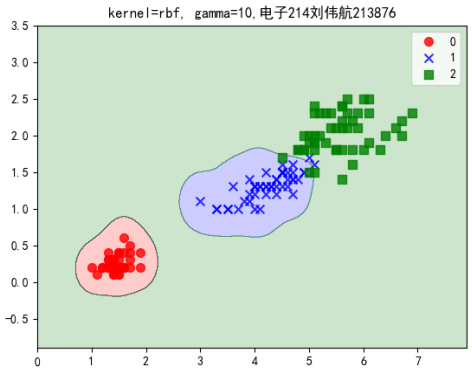

kernel=linear, gamma=auto,电子214刘伟航213876
0
1
2

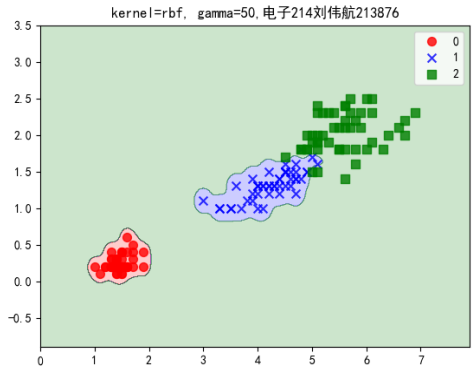

kernel=poly, gamma=auto,电子214刘伟航213876
0
1
2

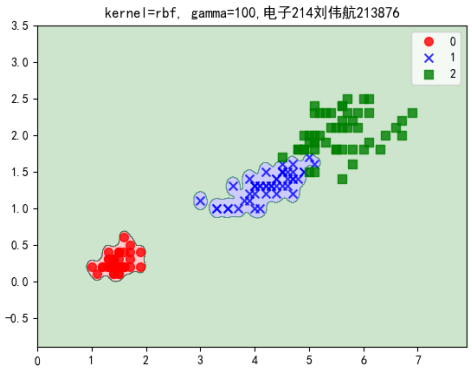

kernel=rbf, gamma=auto,电子214刘伟航213876
0
1
2


kernel=sigmoid, gamma=auto,电子214刘伟航213876
0
1
2


kernel=rbf, gamma=auto,电子214刘伟航213876
0
1
2


kernel=rbf, gamma=10,电子214刘伟航213876
0
1
2


kernel=rbf, gamma=50,电子214刘伟航213876
0
1
2


kernel=rbf, gamma=100,电子214刘伟航213876
0
1
2

五、实验心得

实验三练习了支持向量级的实现方法，支持向量机 SVM 是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机

实验直接使用了 sklearn 库的 svm 方法。同时运用核函数可是实现十分灵活的非线性问题分类。

通过在样本空间中找到一个划分超平面，将不同类别的样本分开，同时使得两个点集到此平面的最小距离最大，两个点集中的边缘点到此平面的距离最大。即最优分类超平面要满足两个条件：能够正确分类，超平面要距离两类样本中最近的样本最远。支持向量机的解具有稀疏性，训练完成后大部分的训练样本都不需保留，最终模型仅予支持向量有关。

在实验中我还观察到虽然样本时随机生成的但由于设置了相同的随机种子，每次生成的样本都相同。

本实验通过不断改变参数 kernel，C，Gamma 的值，来观察他们对实验有什么影响，通过观察图像的变化，可以得到 linear 以及 poly 的测试集精度相对于训练集来说降低的很少，泛化能力较好，而 rbf 核降低的较多，泛化能力差。C 的值对训练集精度和测试机精度影响不是很大，当 C 增大时，训练集精度和测试机精度稍有降低。gamma 越大，分类效果越来越好，但当 gamma 过大时容易出现过拟合的现象。

实验四 K 均值聚类

一、实验目的

1. 熟悉 Python 的基础使用
2. 熟悉几种常见的聚类算法
3. 理解分类和聚类的区别
4. 理解和掌握 k-均值聚类的基本原理和实现过程
5. 能够利用 k-均值聚类解决实际问题

二、实验步骤

1. 导入实验所需数据集
2. 比较两个案例中数据集的不同
3. 比较两个案例解决问题的区别
4. k-均值聚类解决实际问题

三、实验原理

K-means 算法首先需要选择 K 个初始化聚类中心，计算每个数据对象到 K 个初始化聚类中心的距离，将数据对象分到距离聚类中心最近的那个数据集中，当所有数据对象都划分以后，就形成了 K 个数据集（即 K 个簇），接下来重新计算每个簇的数据对象的均值，将均值作为新的聚类中心，最后计算每个数据对象到新的 K 个初始化聚类中心的距离，重新划分，每次划分以后，都需要重新计算初始化聚类中心，一直重复这个过程，直到所有的数据对象无法更新到其他的数据集中。

四、实验过程、结果及分析

任务一：

(1) 运行调试程序（复制时注意缩近问题）,输出结果

(2) 针对下面的指令，改变划线部分的数据取值，重新生成数据集完成聚类，保存结果，说明这四个参数对结果图形的影响。

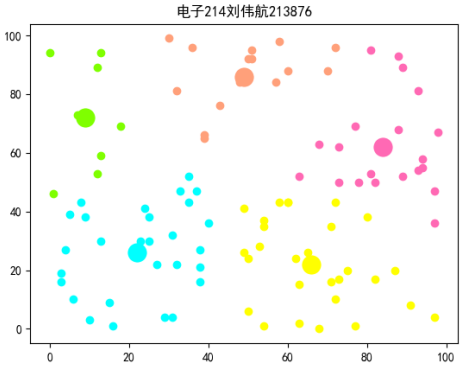
该程序生成随机整数，参数分别为随机范围 1-100 的整数，生成样本的规格 100 个 1 行 2 列样本。结果如后图所示。

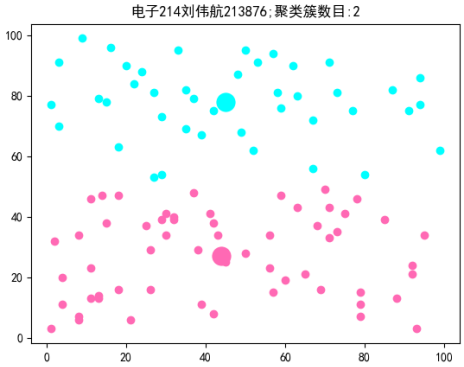
(3) 改变聚类簇数目，分别取 2，3，4，6，观察结果的不同

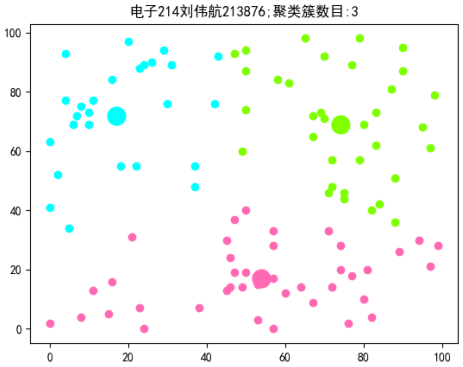
(4) 查阅资料，不用颜色，用不同的离散点型区分簇，如方型、十字星等，并加图名和图例

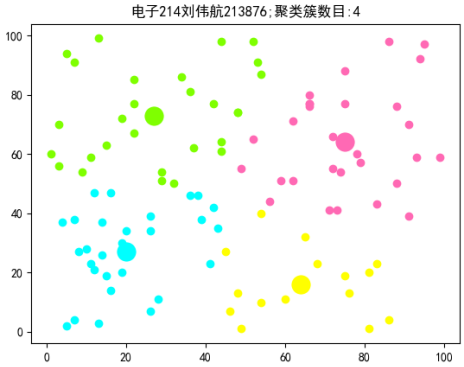
```
import numpy as np
import matplotlib.pyplot as plt
# 两点距离
def distance(e1, e2):
    return np.sqrt((e1[0]-e2[0])**2+(e1[1]-e2[1])**2)
# 集合中心
def means(arr):
    return np.array([np.mean([e[0] for e in arr]), np.mean([e[1] for e in arr])])
# arr 中距离 a 最远的元素，用于初始化聚类中心
def farthest(k_arr, arr):
    f = [0, 0]
    max_d = 0
    for e in arr:
        d = 0
        for i in range(k_arr.__len__()):
            d = d + np.sqrt(distance(k_arr[i], e))
        if d > max_d:
```

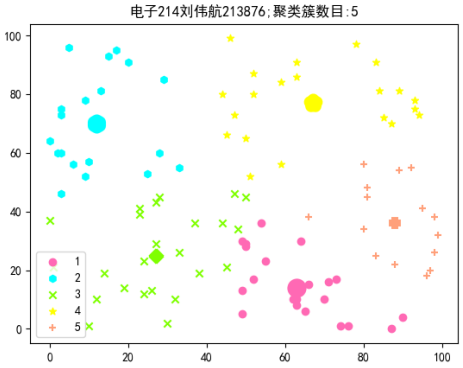
```
                max_d = d
                f = e
    return f
# arr 中距离 a 最近的元素，用于聚类
def closest(a, arr):
    c = arr[1]
    min_d = distance(a, arr[1])
    arr = arr[1:]
    for e in arr:
        d = distance(a, e)
        if d < min_d:
            min_d = d
            c = e
    return c
if __name__=="__main__":
    ## 生成二维随机坐标，手上有数据集的朋友注意，理解 arr 改起来就很容易了
    ## arr 是一个数组，每个元素都是一个二元组，代表着一个坐标
    ## arr 形如：[ (x1, y1), (x2, y2), (x3, y3) ... ]
    arr = np.random.randint(100, size=(100, 1, 2))[:, 0, :]
    ## 初始化聚类中心和聚类容器
    m = 5
    r = np.random.randint(arr.__len__() - 1)
    k_arr = np.array([arr[r]])
    cla_arr = [[]]
    for i in range(m-1):
        k = farthest(k_arr, arr)
        k_arr = np.concatenate([k_arr, np.array([k])])
        cla_arr.append([])
    ## 迭代聚类
    n = 20
    cla_temp = cla_arr
    for i in range(n): # 迭代 n 次
        for e in arr: # 把集合里每一个元素聚到最近的类
            ki = 0 # 假定距离第一个中心最近
            min_d = distance(e, k_arr[ki])
            for j in range(1, k_arr.__len__()):
                if distance(e, k_arr[j]) < min_d: # 找到更近的聚类中心
                    min_d = distance(e, k_arr[j])
                    ki = j
            cla_temp[ki].append(e)
        # 迭代更新聚类中心
        for k in range(k_arr.__len__()):
            if n - 1 == i:
                break
            k_arr[k] = means(cla_temp[k])
            cla_temp[k] = []
    ## 可视化展示
    col = ['HotPink', 'Aqua', 'Chartreuse', 'yellow', 'LightSalmon','black']
    dot = ['o','h','x','*','+','v']
    pl = []
    for i in range(m):
        plt.scatter(k_arr[i][0], k_arr[i][1], linewidth=10, color=col[i],marker=dot[i])
        p=plt.scatter([e[0] for e in cla_temp[i]], [e[1] for e in cla_temp[i]],color=col[i],marker=dot[i])
        pl.append(p)
    plt.title('电子 214 刘伟航 213876;聚类簇数目:{}'.format(m))
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.legend(pl,[1,2,3,4,5])
    plt.show()
```

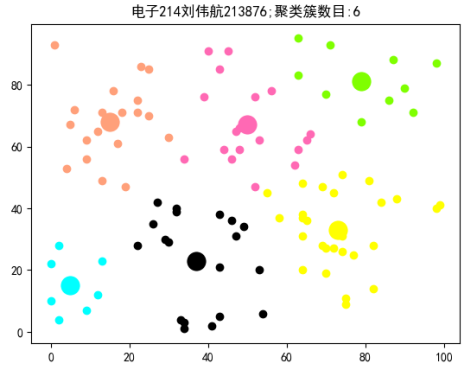

电子214刘伟航213876

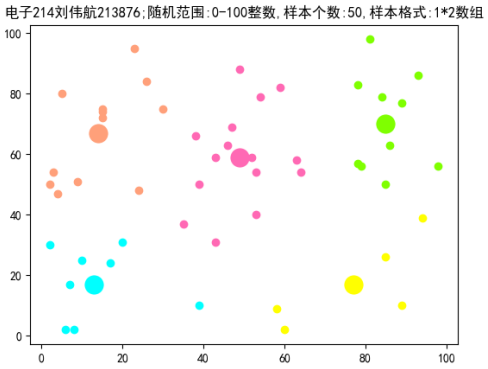

电子214刘伟航213876;聚类簇数目:2

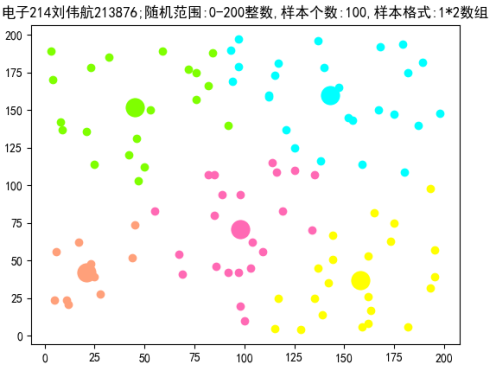

电子214刘伟航213876;聚类簇数目:3

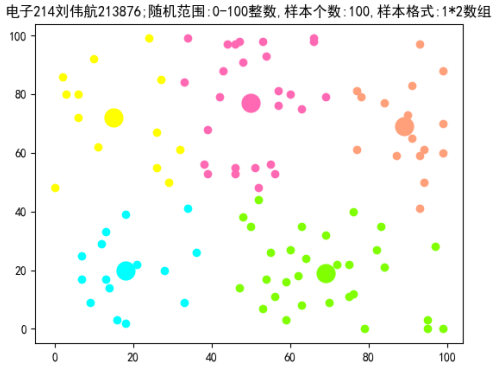

电子214刘伟航213876;聚类簇数目:4


电子214刘伟航213876;聚类簇数目:5
1
2
3
4
5


电子214刘伟航213876;聚类簇数目:6


电子214刘伟航213876;随机范围:0-100整数, 样本个数:50, 样本格式:1*2数组


电子214刘伟航213876;随机范围:0-200整数, 样本个数:100, 样本格式:1*2数组



任务二:给定数据集,内含 80 个二维数据,完成聚类

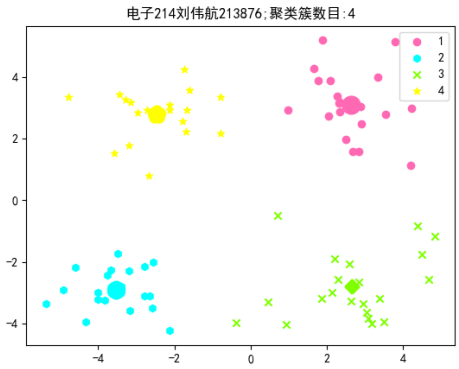
(1) 查阅资料,读入 datas.txt 文件,然后把数据可视化

(2) 聚类数目为 4,输出并保存聚类图形

(3) 用不同的离散点型区分簇,如方型、十字星等，并加图名和图例

```
import numpy as np
import matplotlib.pyplot as plt

# 两点距离
def distance(e1, e2):
    return np.sqrt((e1[0]-e2[0])**2+(e1[1]-e2[1])**2)
# 集合中心
def means(arr):
    return np.array([np.mean([e[0] for e in arr]), np.mean([e[1] for e in arr])])
# arr 中距离 a 最远的元素，用于初始化聚类中心
def farthest(k_arr, arr):
    f = [0, 0]
    max_d = 0
    for e in arr:
        d = 0
        for i in range(k_arr.__len__()):
            d = d + np.sqrt(distance(k_arr[i], e))
        if d > max_d:
            max_d = d
            f = e
    return f
# arr 中距离 a 最近的元素，用于聚类
def closest(a, arr):
    c = arr[1]
    min_d = distance(a, arr[1])
    arr = arr[1:]
    for e in arr:
        d = distance(a, e)
        if d < min_d:
            min_d = d
            c = e
    return c
if __name__=="__main__":
    ## 生成二维随机坐标，手上有数据集的朋友注意，理解 arr 改起来就很容易了
    ## arr 是一个数组，每个元素都是一个二元组，代表着一个坐标
    ## arr 形如：[ (x1, y1), (x2, y2), (x3, y3) ... ]
    # arr = np.random.randint(100, size=(100, 1, 2))[:, 0, :]
    arr = np.loadtxt('juleidatas.txt')
    ## 初始化聚类中心和聚类容器
    m = 4
    r = np.random.randint(arr.__len__() - 1)
    k_arr = np.array([arr[r]])
    cla_arr = [[]]
    for i in range(m-1):
        k = farthest(k_arr, arr)
        k_arr = np.concatenate([k_arr, np.array([k])])
        cla_arr.append([])
    ## 迭代聚类
    n = 20
    cla_temp = cla_arr
    for i in range(n): # 迭代 n 次
```

```
            for e in arr: # 把集合里每一个元素聚到最近的类
                ki = 0 # 假定距离第一个中心最近
                min_d = distance(e, k_arr[ki])
                for j in range(1, k_arr.__len__()):
                    if distance(e, k_arr[j]) < min_d: # 找到更近的聚类中心
                        min_d = distance(e, k_arr[j])
                        ki = j
                cla_temp[ki].append(e)
            # 迭代更新聚类中心
            for k in range(k_arr.__len__()):
                if n - 1 == i:
                    break
                k_arr[k] = means(cla_temp[k])
                cla_temp[k] = []
        ## 可视化展示
        col = ['HotPink', 'Aqua', 'Chartreuse', 'yellow', 'LightSalmon','black']
        dot = ['o','h','x','*','+','v']
        pl = []
        for i in range(m):
            plt.scatter(k_arr[i][0], k_arr[i][1], linewidth=10, color=col[i],marker=dot[i])
            p=plt.scatter([e[0] for e in cla_temp[i]], [e[1] for e in cla_temp[i]],color=col[i],marker=dot[i])
            pl.append(p)
        plt.title('电子 214 刘伟航 213876;聚类簇数目:{}'.format(m))
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False
        plt.legend(pl,[1,2,3,4,5])
        plt.show()
```



五、实验心得

本次实验练习了 kmeans 程序实现方法，k 均值聚类算法是一种迭代求解的聚类分析算法，先随机选取 K 个对象作为初始的聚类中心，然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。聚类中心以及分配给它们的对象就代表一个聚类。一旦全部对象都被分配了，每个聚类的聚类中心会根据聚类中现有的对象被重新计算。这个过程将不断重复直到满足某个终止条件。

因此 kmeans 需要指定聚类的簇数，并且随机选取的初始均值对聚类结果有很大影响，不同初始值可能有不同的结果。

聚类是一种无监督学习，在无监督学习任务中研究最多应用最广，聚类的目标是将数据集中的样本划分为若干个通常不相交的子集。同一簇的样本尽可能彼此相似，不同簇的样本尽可能不同。K 均值算法属于原形聚类，此类算法假设聚类结构能通过一组原型刻画，算法对原型进行初始化，再对原型进行迭代更新求解。

实验五 密度聚类

一、实验目的

1. 熟悉 Python 的基础使用
2. 掌握 sklearn. datasets 数据生成
3. 理解和掌握 DBSCAN 聚类的基本原理和实现过程
4. 体会 DBSCAN 聚类时 eps,min_Pts 参数对结果的影响

二、实验步骤

1. 了解 DBSCAN 聚类原理及实现
2. 自制数据集
3. 用 DBSCAN 聚类解决实际问题
4. 体会 k-均值聚类和 DBSCAN 聚类的区别

三、实验原理

DBSCAN 是一种基于密度的聚类算法，这类密度聚类算法一般假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间的紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。

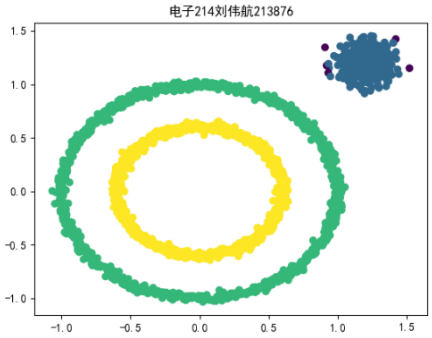
通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，就得到了最终的所有聚类类别结果。

四、实验过程、结果及分析

（1）打开原文链接，看懂并运行程序，运行结果以“班级姓名”为名原文链接:https://blog.csdn. net/xyisv/article/details/88918448

```
from sklearn import datasets
import numpy as np
import random
import matplotlib.pyplot as plt
import time
import copy


def find_neighbor(j, x, eps):
    N = list()
    for i in range(x.shape[0]):
        temp =
np.sqrt(np.sum(np.square(x[j] - x[i])))  #
计算欧式距离
        if temp <= eps:
            N.append(i)
    return set(N)



def DBSCAN(X, eps, min_Pts):
    k = -1
    neighbor_list = []  # 用来保存每个数
据的邻域
    omega_list = []  # 核心对象集合
    gama = set([x for x in range(len(X))])
# 初始时将所有点标记为未访问
    cluster = [-1 for _ in range(len(X))]  #
聚类
    for i in range(len(X)):

neighbor_list.append(find_neighbor(i, X,
eps))
        if len(neighbor_list[-1]) >=
min_Pts:
            omega_list.append(i)  # 将
样本加入核心对象集合
    omega_list = set(omega_list)  # 转化
为集合便于操作
    while len(omega_list) > 0:
        gama_old = copy.deepcopy(gama)
        j =
random.choice(list(omega_list))  # 随机选
取一个核心对象
        k = k + 1
        Q = list()
        Q.append(j)
        gama.remove(j)
```

```
            while len(Q) > 0:
                q = Q[0]
                Q.remove(q)
                if len(neighbor_list[q]) >=
min_Pts:
                    delta = neighbor_list[q]
& gama
                    deltalist = list(delta)
                    for i in range(len(delta)):

Q.append(deltalist[i])
                        gama = gama -
delta
            Ck = gama_old - gama
            Cklist = list(Ck)
            for i in range(len(Ck)):
                cluster[Cklist[i]] = k
            omega_list = omega_list - Ck
        return cluster


X1, y1 =
datasets.make_circles(n_samples=2000,
factor=.6, noise=.02)
X2, y2 =
datasets.make_blobs(n_samples=400,
n_features=2, centers=[[1.2, 1.2]],
cluster_std=[[.1]], random_state=9)
X = np.concatenate((X1, X2))
eps = 0.08
min_Pts = 10
begin = time.time()
C = DBSCAN(X, eps, min_Pts)
end = time.time()
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=C)
plt.title('电子 214 刘伟航 213876')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.show()
```



（2）程序中的数据是由下面两条语句得到的。

利用这两条语句,编写一个数据集可视化的程序;分析两条语句的作用;

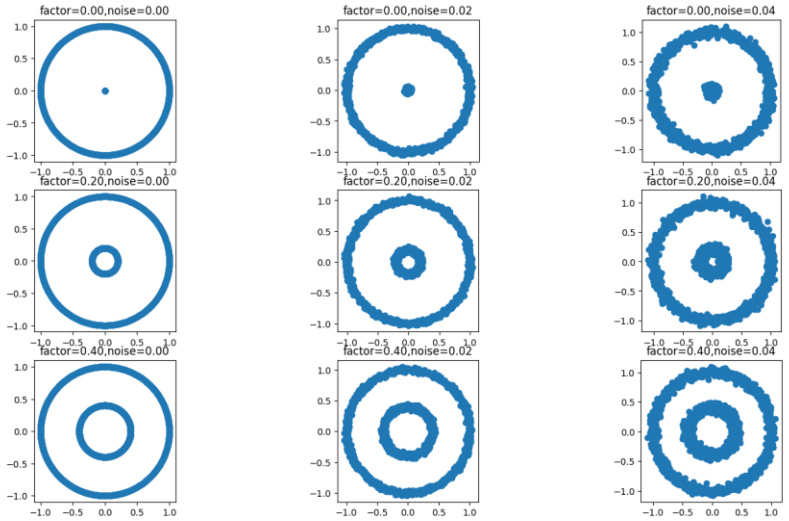
改变语句中的参数，分析参数作用

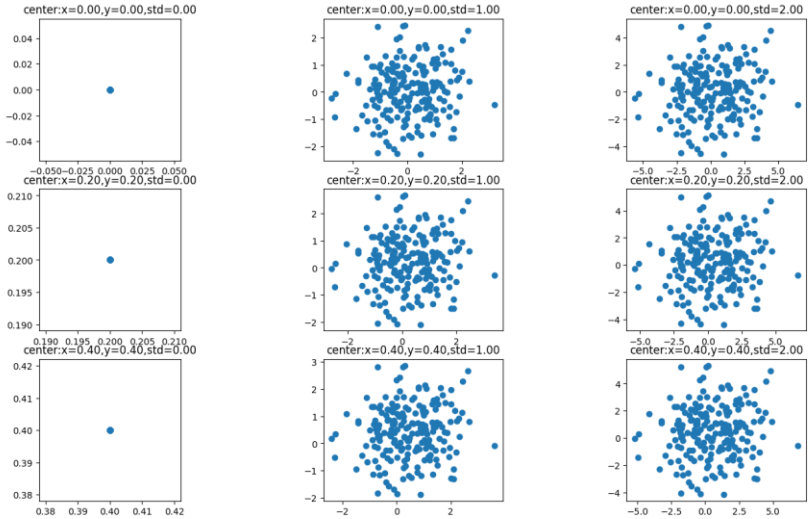
语句 1 是产生环形样本点的程序，参数分别为样本点个数，内外环半径之比，噪声大小影响样本点方差

语句 2 产生球状簇状样本点，参数分别为样本点个数，样本点维度，样本中心点期望均值，样本点标准差，随机状态相同随机状态生成的随机点每次都相同。

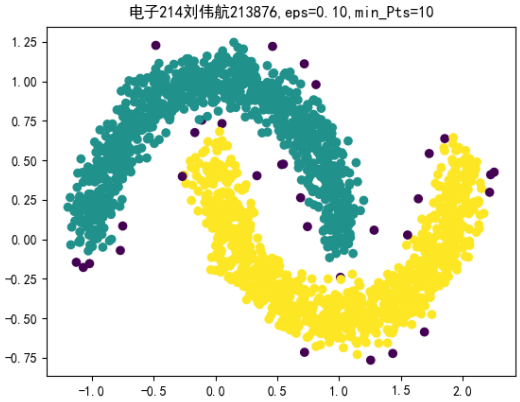
保存运行结果（多图，对比展示，用图名标注参数数值）

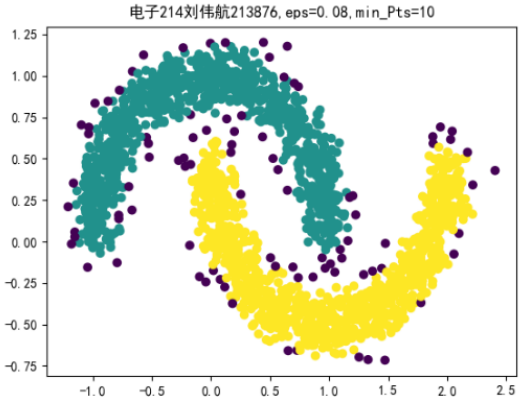
```
n_samples=200
for i in range(3):
    x=0.2*i
    for j in range(3):
        cluster_std = j
        # X1, y1 =
datasets.make_circles(n_samples=n_samples
, factor=factor, noise=noise)
        X2, y2 =
datasets.make_blobs(n_samples=n_samples,
n_features=2, centers=[[x, x]],
cluster_std=[[cluster_std]], random_state=9)
            plt.subplot(3,3,i*3+j+1)
            # plt.scatter(X1[:,0],X1[:,1])
            plt.scatter(X2[:,0],X2[:,1])

plt.title('center:x=%.2f,y=%.2f,std=%.2f'%(x
,x,cluster_std))
            plt.gca().set_aspect(1)
plt.show()
```

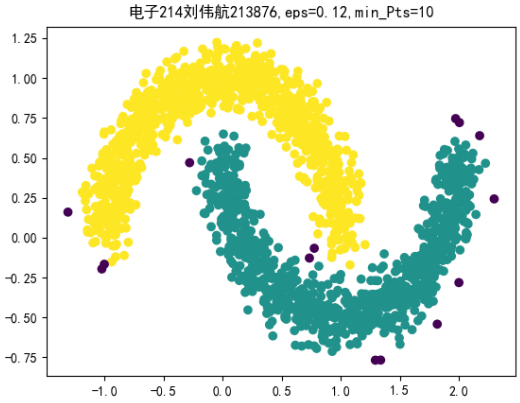




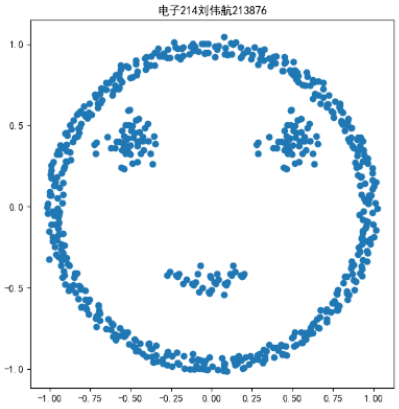

```
X1, y1 =
datasets.make_moons(n_samples=2000,
shuffle=True, noise=0.1,
random_state=None)
eps = 0.12
min_Pts = 10
begin = time.time()
C = DBSCAN(X1, eps, min_Pts)
end = time.time()
plt.figure()
plt.scatter(X1[:, 0], X1[:, 1], c=C)
plt.title('电子 214 刘伟航
213876,eps=%.2f,min_Pts=%.0f'%(eps,min_
Pts))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.show()
```

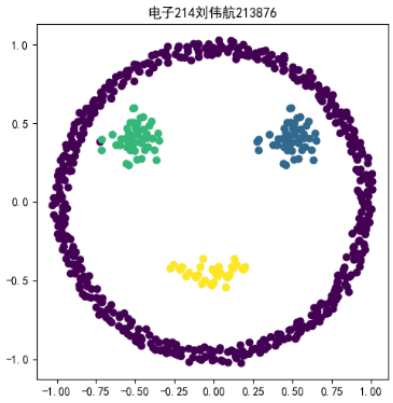







```
X1, y1 =
datasets.make_circles(n_samples=500,factor
=.95,noise=.015)
X2, y2 =
datasets.make_blobs(n_samples=50,
n_features=2, centers=[[-0.5, 0.4]],
cluster_std=[[.08]], random_state=9)
X3, y3 =
datasets.make_blobs(n_samples=50,
n_features=2, centers=[[0.5, 0.4]],
cluster_std=[[.08]], random_state=9)
X4, y4 =
datasets.make_blobs(n_samples=10,
n_features=2, centers=[[0, -0.5]],
cluster_std=[[.05]], random_state=9)
X5, y5 = datasets.make_blobs(n_samples=8,
n_features=2, centers=[[0.1, -0.45]],
cluster_std=[[.05]], random_state=9)
X6, y6 = datasets.make_blobs(n_samples=8,
n_features=2, centers=[[-0.1, -0.45]],
cluster_std=[[.05]], random_state=9)
X7, y7 = datasets.make_blobs(n_samples=5,
n_features=2, centers=[[-0.2, -0.4]],
cluster_std=[[.05]], random_state=9)
X8, y8 = datasets.make_blobs(n_samples=5,
n_features=2, centers=[[0.2, -0.4]],
cluster_std=[[.05]], random_state=9)
X = np.concatenate((X1, X2, X3,
X4,X5,X6,X7,X8))
# eps = 0.12
# min_Pts = 8
# C = DBSCAN(X, eps, min_Pts)
# plt.figure()
# plt.scatter(X[:, 0], X[:, 1], c=C)
plt.scatter(X[:, 0], X[:, 1])
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title('电子 214 刘伟航 213876')
plt.gca().set_aspect(1)
plt.show()
```





五、实验心得

（实验中的收获、意见和建议等，不少于 200 字。）

本次实验练习了密度聚类的实现方法，密度聚类 DBSCAN 是具有噪声的基于密度的聚类方法，也是一种基于密度的空间聚类算法。其核心思想就是先发现密度较高的点，然后把相近的高密度点逐步都连成一片，进而生成各种簇。

密度聚类对于任意形状可进行聚类，不受噪声影响，可解释性也强于 k 均值聚类。同时也具有很多缺点，需要不断调整聚类参数才能产生比较好的聚类效果，不能指定聚类簇的数目，计算量大，计算复杂度高，如果数据密度分布不均匀，则可能导致聚类效果不佳。

实验中主要练习了不同形状样本点的生成方法，掌握了对 plot 方法的子图绘制，通过循环控制参数调整，子图对比参数改变产生的变化。

实验六 KNN

一、实验目的

1.熟悉 Python 的基础使用

2.理解和掌握 k 近邻的基本原理和实现过程

3.利用 k 近邻解决实际问题

二、实验步骤

1. 绘制实验所需数据集
2. 运行 k 近邻程序
3. 对测试数据进行分类判别，并显示分类结果

三、实验原理

KNN（k 邻近法），最初由 Cover 和 Hart 于 1968 年提出，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一，它的适用面很广，并且在样本量足够大的情况下准确度很高，多年来得到了很多的关注和研究。

它的工作原理是：存在一个样本数据集合，也称作为训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一个数据与所属分类的对应关系。输入没有标签的新数据后，将新的数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本最相似数据(最近邻)的分类标签。一般来说，我们只选择样本数据集中前 k 个最相似的数据，这就是 k-近邻算法中 k 的出处，通常 k 是不大于 20 的整数。最后，选择 k 个最相似数据中出现次数最多的分类，作为新数据的分类。

（按指导书中的任务要求顺序写，关键地方要辅以程序、图片等说明，组织语言注意条理性。凡是任务中涉及到编程实现的，要附上完整的程序和结果。程序部分要求白底黑字，有适当的注释）

任务 1:

有以下数据点: ([[2, 1], [3, 2], [4, 2], [1, 3], [1.5, 4], [1.7, 3], [2.6,5], [3. 4, 3], [3, 6], [1, 7], [4, 5], [1. 2, 6], [1.8, 7], [2. 2, 8], [3.7, 7], [4.8,5]])

其类别为([0,0,0,0,0,0,0,0,1, 1, 1,1,1,1, 1, 1])

设计 KNN 算法进行分类判断，测试数据为([3.2, 5.4])

任务要求：

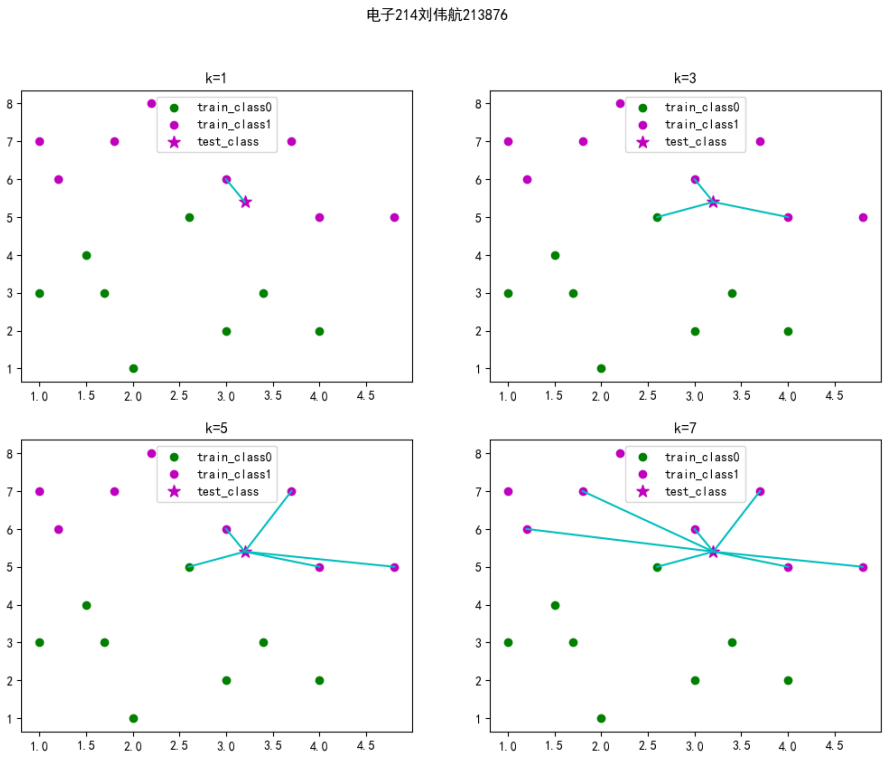
（1）读懂程序，尤其是排序和遍历得到前 K 个数值的过程；

（2）让 k=1，3，5，7，观察结果的不同。用 subplot 指令将四个结果用子图的形式保存成 2 行 2 列，每个子图用 k 值命名；

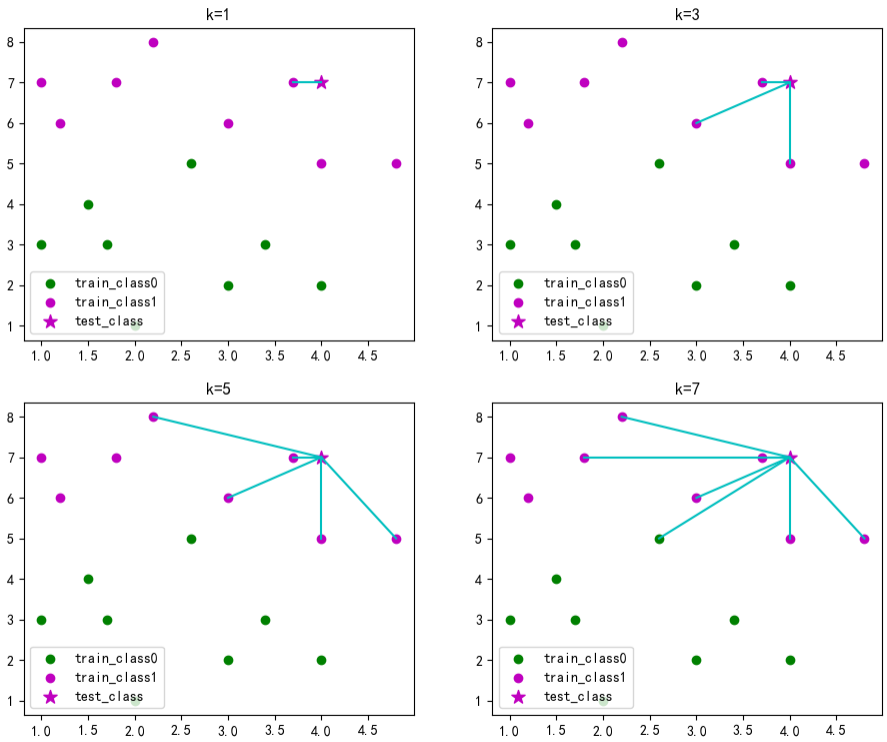
（3）多改变几个测试数据的值，观察结果的不同。

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from sklearn.neighbors import
KNeighborsClassifier   # k 邻近算法模型
import matplotlib.pyplot as plt
x_train = np.array(
      [[2, 1], [3, 2], [4, 2], [1, 3], [1.5, 4],
[1.7, 3], [2.6, 5], [3.4, 3], [3, 6], [1, 7], [4, 5],
[1.2, 6], [1.8, 7],
       [2.2, 8], [3.7, 7], [4.8, 5]])
```

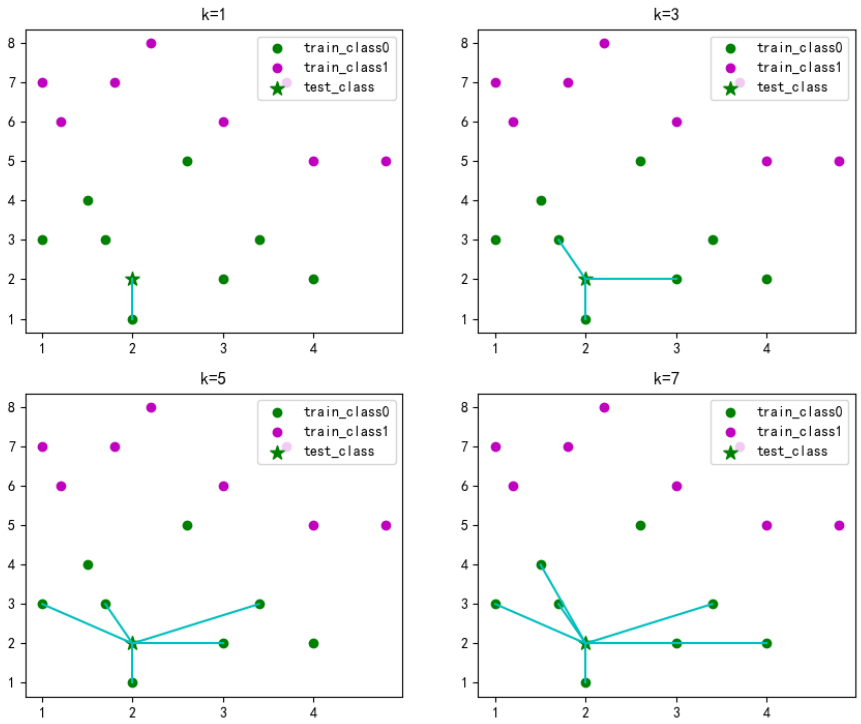
```
y_train = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
x_test = np.array([[2, 5]])  # X_test 为测试样本

X_train_0 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 0])
# 将 class1 一类的样本点放到 X_train_1 中
X_train_1 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 1])
# 绘制所有样本点并采用不同的颜色分别标记 class0 以及 class1
fig = plt.figure()
fig.suptitle("电子 214 刘伟航 213876, test:{}".format(x_test))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
for k in [1, 3, 5, 7]:  # k 为邻居数
    plt.subplot(2, 2, (k + 1) // 2)
    plt.title("k=%s" % (k))

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    knn.score(x_train, y_train)
    y_test = knn.predict(x_test)
    y_c = 'g' if y_test == 0 else 'm'
    plt.scatter(X_train_0[:, 0], X_train_0[:, 1], c='g', marker='o', label='train_class0')
    plt.scatter(X_train_1[:, 0], X_train_1[:, 1], c='m', marker='o', label='train_class1')
    plt.scatter(x_test[0][0], x_test[0][1], c=y_c, marker='*', s=100, label='test_class')
    print(x_train, y_train, x_test)
    print('预测分类：', knn.predict(x_test))
    plt.legend(loc='best')
plt.show()
```



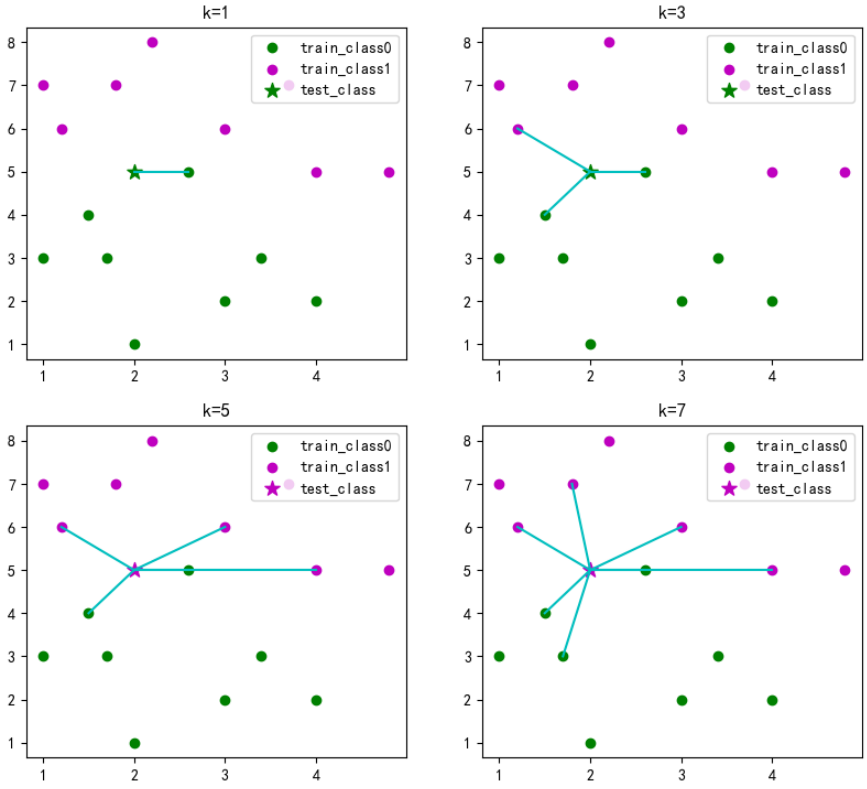
电子214刘伟航213876, test:[4 7]



电子214刘伟航213876, test:[2 2]



电子214刘伟航213876, test:[2 5]



任务 3：直接调用 KNN 库

任务要求：

打开原文链接 https://www.cnblogs.com/angle6-liu/p/10416736.html

（1）运行 1. 数据蓝蝴蝶和 2. 根据身高、体重、鞋子尺码，预测性别的程序。

（2）直接调用 KNN 库，完成任务 1，比较结果。

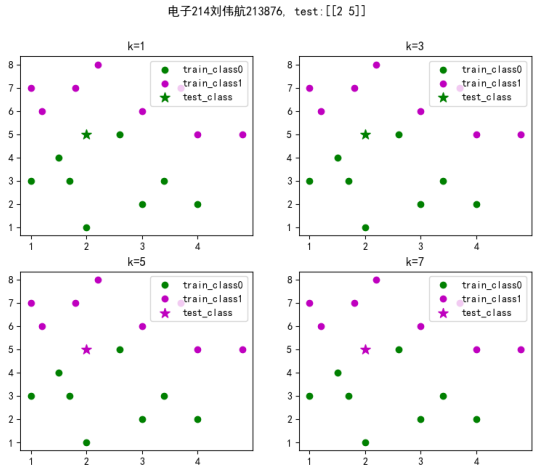
```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from sklearn.neighbors import KNeighborsClassifier   # k 邻近算法模型
import matplotlib.pyplot as plt

x_train = np.array(
    [[2, 1], [3, 2], [4, 2], [1, 3], [1.5, 4], [1.7, 3], [2.6, 5], [3.4, 3], [3, 6], [1, 7], [4, 5], [1.2, 6],
[1.8, 7],
     [2.2, 8], [3.7, 7], [4.8, 5]])
y_train = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1])
x_test = np.array([[2, 5]])   # X_test 为测试样本

X_train_0 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 0])
# 将 class1 一类的样本点放到 X_train_1 中
X_train_1 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 1])
# 绘制所有样本点并采用不同的颜色分别标记 class0 以及 class1
fig = plt.figure()
fig.suptitle("电子 214 刘伟航 213876, test:{}".format(x_test))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
for k in [1, 3, 5, 7]:   # k 为邻居数
    plt.subplot(2, 2, (k + 1) // 2)
    plt.title("k=%s" % (k))

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    knn.score(x_train, y_train)
    y_test = knn.predict(x_test)
    y_c = 'g' if y_test == 0 else 'm'
    plt.scatter(X_train_0[:, 0], X_train_0[:, 1], c='g', marker='o', label='train_class0')
    plt.scatter(X_train_1[:, 0], X_train_1[:, 1], c='m', marker='o', label='train_class1')
    plt.scatter(x_test[0][0], x_test[0][1], c=y_c, marker='*', s=100, label='test_class')
    print(x_train, y_train, x_test)
    print('预测分类：', knn.predict(x_test))
    plt.legend(loc='best')
plt.show()
```

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from sklearn.neighbors import KNeighborsClassifier   # k 邻近算法模型
import matplotlib.pyplot as plt

x_train = np.array(
      [[2, 1], [3, 2], [4, 2], [1, 3], [1.5, 4], [1.7, 3], [2.6, 5], [3.4, 3], [3, 6], [1, 7], [4, 5], [1.2, 6],
[1.8, 7],
       [2.2, 8], [3.7, 7], [4.8, 5]])
y_train = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1])
x_test = np.array([[2, 5]])   # X_test 为测试样本

X_train_0 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 0])
# 将 class1 一类的样本点放到 X_train_1 中
X_train_1 = np.array([x_train[i, :] for i in range(len(y_train)) if y_train[i] == 1])
# 绘制所有样本点并采用不同的颜色分别标记 class0 以及 class1
fig = plt.figure()
fig.suptitle("电子 214 刘伟航 213876, test:{}".format(x_test))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
for k in [1, 3, 5, 7]:   # k 为邻居数
      plt.subplot(2, 2, (k + 1) // 2)
      plt.title("k=%s" % (k))

      knn = KNeighborsClassifier(n_neighbors=k)
      knn.fit(x_train, y_train)
      knn.score(x_train, y_train)
      y_test = knn.predict(x_test)
      y_c = 'g' if y_test == 0 else 'm'
      plt.scatter(X_train_0[:, 0], X_train_0[:, 1], c='g', marker='o', label='train_class0')
      plt.scatter(X_train_1[:, 0], X_train_1[:, 1], c='m', marker='o', label='train_class1')
      plt.scatter(x_test[0][0], x_test[0][1], c=y_c, marker='*', s=100, label='test_class')
      print(x_train, y_train, x_test)
      print('预测分类：', knn.predict(x_test))
      plt.legend(loc='best')
plt.show()
```

```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\ex6_3_1.py
预测分类： [0 2 1 2 1 2 2 1 2 0]
真实分类： [0 2 1 2 1 2 2 1 2 0]

Process finished with exit code 0
```

```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\ex6_3_2.py
['男']

Process finished with exit code 0
```

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from sklearn.neighbors import
KNeighborsClassifier   # k 邻近算法模型
import matplotlib.pyplot as plt

x_train = np.array(
      [[2, 1], [3, 2], [4, 2], [1, 3], [1.5, 4],
[1.7, 3], [2.6, 5], [3.4, 3], [3, 6], [1, 7], [4, 5],
[1.2, 6], [1.8, 7],
       [2.2, 8], [3.7, 7], [4.8, 5]])
y_train = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1])
x_test = np.array([[2, 5]])   # X_test 为测
试样本

X_train_0 = np.array([x_train[i, :] for i in
range(len(y_train)) if y_train[i] == 0])
# 将 class1 一类的样本点放到 X_train_1
中
X_train_1 = np.array([x_train[i, :] for i in
range(len(y_train)) if y_train[i] == 1])
# 绘制所有样本点并采用不同的颜色分别
标记 class0 以及 class1
fig = plt.figure()
fig.suptitle("电子 214 刘伟航 213876,
test:{}".format(x_test))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
for k in [1, 3, 5, 7]:   # k 为邻居数
      plt.subplot(2, 2, (k + 1) // 2)
      plt.title("k=%s" % (k))

      knn =
KNeighborsClassifier(n_neighbors=k)
      knn.fit(x_train, y_train)
      knn.score(x_train, y_train)
      y_test = knn.predict(x_test)
      y_c = 'g' if y_test == 0 else 'm'
      plt.scatter(X_train_0[:, 0], X_train_0[:,
1], c='g', marker='o', label='train_class0')
      plt.scatter(X_train_1[:, 0], X_train_1[:,
1], c='m', marker='o', label='train_class1')
      plt.scatter(x_test[0][0], x_test[0][1],
c=y_c, marker='*', s=100, label='test_class')
      print(x_train, y_train, x_test)
      print('预测分类：', knn.predict(x_test))
      plt.legend(loc='best')
plt.show()
```



任务 2:思考分析（先做任务 3，最后做这个）
从图中看出,K 近邻找到的貌似不全是距离测试样本最近的点，阅读程序中关于距离计算的部分，读懂程序，分析可能的原因，并给出修改建议。
程序中有关距离的技术按使用的是欧式距离，且每次计算距离后都进行了排序，于是考虑是否有距离以外的其他因素决定 knn 结果，然而只有距离影响决策结

果，考虑什因素让可见的最近样本点没有参与到决策中，发现样本点横纵坐标单位长度并不等长，导致肉眼观测的最近样本点并非绝对距离最近的样本点，解决办法可以在绘图中加入语句：plt.gca().set_aspect(1)

五、实验心得

本次实验主要练习了 K 近邻的实现。k 近邻算法是一种基本分类和回归方法。算法的核心思想是，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的 K 个实例，这 K 个实例的多数属于某个类，就把该输入实例分类到这个类中。算法简单，容易理解，

k 近邻法三要素：k 值的选择，分类决策规则，距离度量

KNN 算法不仅可以用于分类，还可以用于回归。通过找出一个样本的 k 个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权值，如权值与距离成反比。

knn 的另一个缺点为计算量大，因为对每一个待分类的样本都要计算它到全体已知样本的距离，才能求得它的 K 个最近邻点。可以对已知样本点进行剪辑，事先去除对分类作用不大的样本。该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

实验七 PCA

一、实验目的

1.掌握主成分分析的基本原理
2.编程实现简单的主成分分析
3.掌握主成分分析里的关键语句
4.能够用主成分分析解决实际问题

二、实验步骤

1.生成或调用数据集
2.生成或调用 PCA 函数
3.绘图
4.得出结果并分析不同参数对结果的影响

三、实验原理

PCA 是一种常用的降维技术，它通过线性变换将高维数据映射到低维空间，同时保留数据的主要特征。

数据中心化：首先，对原始数据进行中心化操作，即将每个维度的数据减去该维度上的均值，使得数据的均值为零。中心化操作可以消除数据的平移影响。

计算协方差矩阵：然后，计算中心化后的数据的协方差矩阵。协方差矩阵描述了不同维度之间的线性关系。协方差矩阵的元素表示了两个维度之间的相关性。

特征值分解：对协方差矩阵进行特征值分解，得到特征值和对应的特征向量。特征值表示了数据在特征向量方向上的方差，而特征向量则表示了数据在新的特征空间中的方向。

四、实验过程、结果及分析

任务 1：编程实现 PCA

（1）结合附件程序,写出 PCA 实现原理；

原理：标准化数据：首先，将原始数据进行标准化，使得每个特征具有零均值和单位方差。这是为了消除不同特征之间的量纲差异，确保各个特征对降维结果的影响相同。计算协方差矩阵：然后，计算标准化后数据的协方差矩阵。协方差度量了不同特征之间的相关性，协方差矩阵的元素表示了两个特征之间的协方差。特征值分解：对协方差矩阵进行特征值分解，得到特征值和对应的特征向量。特征值表示了数据在特征向量方向上的方差，而特征向量表示了数据投影到该方向的权重。选择主成分：根据特征值的大小，选择前 k 个特征值对应的特征向量作为主成分。这些主成分代表了数据中最重要的结构信息，对应的特征向量即为主成分向量。数据投影：将原始数据投影到选择的主成分上，得到降维后的低维数据。投影的计算可以通过将原始数据与选择的主成分向量相乘来实现。

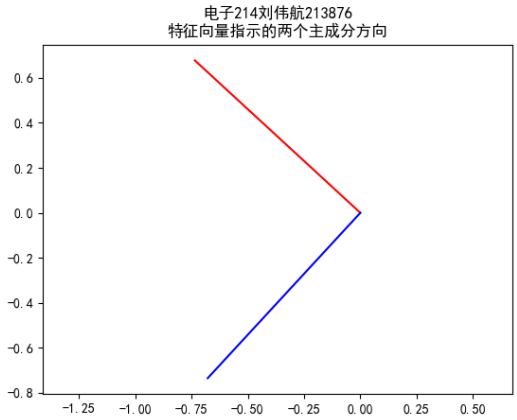
分析最后的结果图中红、蓝色实线，和红、蓝、绿离散点所代表的意义：

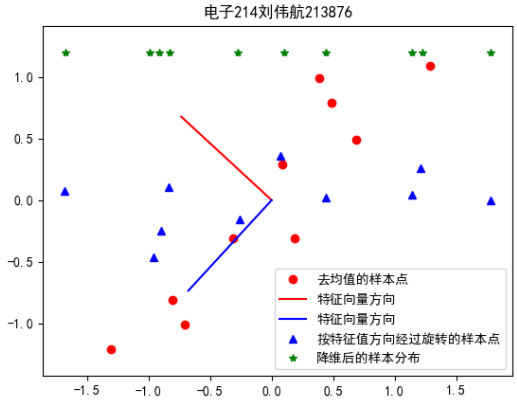
红色和蓝色的实线分别代表了原始数据的两个主成分特征向量：红色实线对应的特征向量是第一个主成分，具有最大的特征值，表示数据中方差最大的方向。蓝色实线对应的特征向量是第二个主成分，具有次大的特征值，表示数据中方差次大的方向。

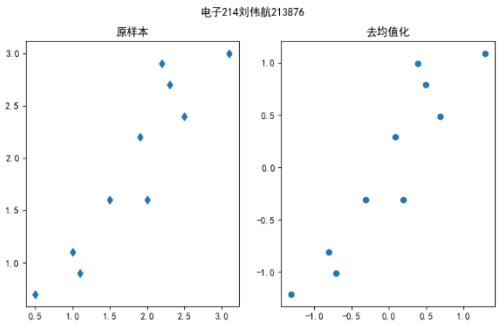
红色、蓝色的离散点表示了原始数据在新的特征空间中的投影：红色离散点对应的是第一个主成分方向上的投影，蓝色离散点对应的是第二个主成分方向上的投影，绿色星号表示了降维后的数据。

```
import numpy as np
import matplotlib.pyplot as plt
# 样本点
x=np.array([2.5,0.5,2.2,1.9,3.1,2.3,2,1,1.5,1.1])
y=np.array([2.4,0.7,2.9,2.2,3,2.7,1.6,1.1,1.6,0.9])
# 计算样本均值
mean_x=np.mean(x)
mean_y=np.mean(y)
# 样本去均值
scaled_x=x-mean_x
scaled_y=y-mean_y
# 样本点矩阵化
data=np.matrix([[scaled_x[i],scaled_y[i]] for i in range(len(scaled_x))])

# 一行两列子图，第一幅绘制样本点，第二幅绘制去均值的样本点
fig=plt.figure()
fig.suptitle("电子 214 刘伟航 213876")
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.subplot(121)
plt.title("原样本")
plt.plot(x,y,'d')
plt.subplot(122)
plt.title("去均值化")
plt.plot(scaled_x,scaled_y,'o')
plt.show()

# 求协方差矩阵
cov=np.cov(scaled_x,scaled_y)
print('cov=',cov)
# 计算特征值、特征向量；输出
eig_val, eig_vec = np.linalg.eig(cov)
print('eig_val=',eig_val)
print('eig_vec=',eig_vec)

## 根据样本点范围设置绘图区域 X，Y 坐标范围
# xmin ,xmax = scaled_x.min(), scaled_x.max()
# ymin, ymax = scaled_y.min(), scaled_y.max()
# dx = (xmax - xmin) * 0.2
# dy = (ymax - ymin) * 0.2
# plt.xlim(xmin - dx, xmax + dx)
# plt.ylim(ymin - dy, ymax + dy)

# 特征向量与样本点做矩阵乘法，进行降维，转置
new_data=np.transpose(np.dot(eig_vec,np.transpose(data)))
# 分别绘制特征向量矩阵的两列向量的方向
plt.axis('equal')
plt.title("电子 214 刘伟航 213876\n 特征向量指示的两个主成分方向")
plt.plot([eig_vec[:,0][0],0],[eig_vec[:,0][1],0],color='red')
plt.plot([eig_vec[:,1][0],0],[eig_vec[:,1][1],0],color='blue')
plt.show()

# 改变数据维度
eig_pairs = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(len(eig_val))]
# 根据特征值从大到小，对特征向量排序
eig_pairs.sort(reverse=True)
# 取出特征值最大对应的特征向量 feature
feature=eig_pairs[0][1]
# 降到一维
new_data_reduced=np.transpose(np.dot(feature,np.transpose(data)))
print('new_data_reduced = ')
print(new_data_reduced)

# 绘制去均值的样本点
p1,
=plt.plot(scaled_x,scaled_y,'o',color='red')
# 绘制降维的特征向量方向
p2,
=plt.plot([eig_vec[:,0][0],0],[eig_vec[:,0][1],0],color='red')
p3,
=plt.plot([eig_vec[:,1][0],0],[eig_vec[:,1][1],0],color='blue')
```

```
# 绘制降维后的数据点
p4,
=plt.plot(new_data[:,0],new_data[:,1],'^',col
or='blue')
p5,
=plt.plot(new_data_reduced[:,0],[1.2]*10,'*',
color='green')
plt.legend([p1,p2,p3,p4,p5],["去均值的样本
点","特征向量方向","特征向量方向","按
特征值方向经过旋转的样本点","降维后
的样本分布"])
#
plt.plot(new_data_reduced[:,0]*feature[0],ne
w_data_reduced[:,0]*feature[1],'.',color='ma
roon')
plt.axis('equal')
plt.title("电子 214 刘伟航 213876")
plt.show()
```







分析程序中蓝色语句的作用，如果去掉对结果有什么影响，说明原因。

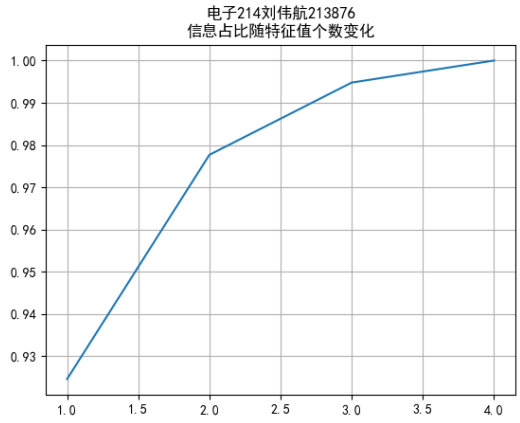
```
xmin ,xmax = scaled_x.min(), scaled_x.max()
ymin, ymax = scaled_y.min(), scaled_y.max()
dx = (xmax - xmin) * 0.2
dy = (ymax - ymin) * 0.2
plt.xlim(xmin - dx, xmax + dx)
plt.ylim(ymin - dy, ymax + dy)
```

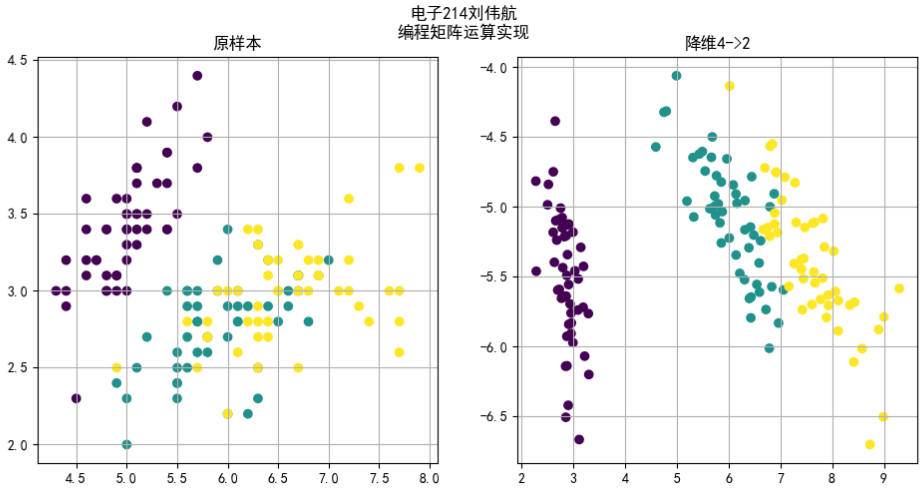
该语句设置了绘图的坐标轴范围，如果去掉这段代码，将会导致绘制的散点图在默认的坐标轴范围内显示，可能会出现以下情况：

1. 数据点分布不够集中：如果散点图所示的数据分布在整个坐标轴范围内，那么可能会导致数据点过于分散，无法清晰地观察到数据的分布特征；

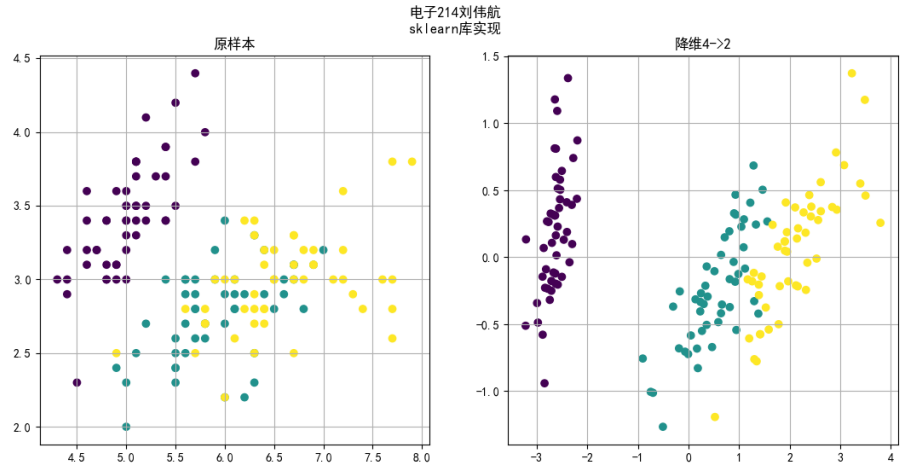
2. 部分数据点被裁剪：如果散点图所示的数据超出了默认的坐标轴范围，部分数据点可能会被裁剪，导致无法完整地展示所有的数据点。

```
X, y = load_iris(return_X_y=True)
# plt.scatter(X[:,0], X[:, 1],c=y)
# plt.grid()
# plt.show()

cov_mat = np.cov(X.T)
print("协方差矩阵:\n",cov_mat)
eig_val_cov, eig_vec_cov =
np.linalg.eig(cov_mat)  # 特征值，特征向量
vec = eig_vec_cov[:, :2]
print("特征向量:\n",vec)
X_new = np.dot(X, vec)  # 矩阵相乘
print(X_new.shape)

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.suptitle("电子 214 刘伟航\n 编程矩阵运算实现")
plt.subplot(1,2,1)
plt.title("原样本")
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.grid()

plt.subplot(1,2,2)
plt.title("降维 4->2")
plt.scatter(X_new[:, 0], X_new[:, 1], c=y)
plt.grid()
plt.show()

# 4
print(eig_val_cov)
print(eig_val_cov / eig_val_cov.sum())
x = eig_val_cov / eig_val_cov.sum()
plt.figure()
plt.plot(range(1, 5), np.cumsum(x))
plt.grid()
plt.title("电子 214 刘伟航 213876\n 信息占比随特征值个数变化")
plt.show()
```





```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.suptitle("电子 214 刘伟航\nsklearn 库实现")

plt.subplot(1,2,1)
plt.title("原样本")
X, y = load_iris(return_X_y=True)
pca=PCA(n_components=2)                    # n_components 主成分的个数
X_new_sk=pca.fit_transform(X)
print("特征向量:\n",pca.components_)        #  查看特征向量
plt.scatter(X[:,0], X[:, 1],c=y)
plt.grid()

plt.subplot(1,2,2)
plt.title("降维 4->2")
plt.scatter(X_new_sk[:,0], X_new_sk[:, 1],c=y)
plt.grid()

plt.show()
```



五、实验心得

本次实验练习使用了 PCA，主成分分析（PCA）是一种基于变量协方差矩阵对数据进行压缩降维、去噪的有效方法：将方差最大的方向作为主要特征，并且在各个正交方向上数据没有相关性。目的是从一组特征中重新计算出一组按重要性从大到小排列的新特征，它们是原有特征的线性组合，并且相互之间是不相关的。

PCA 局限是可以很好地解决线性相关，但是对于高阶相关性就没有办法了。对于存在高阶相关性的数据，可以考虑 Kernel PCA，通过 Kernel 将非线性相关转化为线性相关。PCA 假设数据各特征分布在正交方向上，如果在非正交方向上存在几个方差较大的方向，PCA 的效果就大打折扣。PCA 是一种无参数技术，也就是说面对同样的数据，谁来做结果都一样，没有主观参数的介入，所以 PCA 便于通用实现，但是本身没有个性化的优化。

实验过程中可以看出 PCA 对数据的压缩效果很好，PCA 作为算法的一种预处理方法可以大大降低算法处理数据集的维度，时间。实验中运用了协方差矩阵和特征值特征向量等矩阵运算方法。让我对数据的处理有了更深的理解。

实验八 决策树

一、实验目的

1. 熟悉 Python 的基础使用，会调用.txt 文件
2. 理解和掌握决策树的基本原理和实现过程
3. 掌握信息熵、信息增益的计算

二、实验步骤

1. 了解决策树原理
2. 运行决策树程序
3. 移植程序，用决策树解决实际问题

三、实验原理

决策树(Decision Tree)在机器学习中是比较常见的一种算法，属于监督学习中的一种。相比其他算法比如支持向量机或神经网络，决策树的概念更加直观。优点：计算复杂度不高，输出结果易于理解，对中间值的缺失值不敏感，可以处理不相关特征数据。

四、实验过程、结果及分析

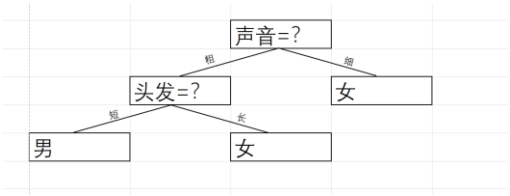
任务 1:手动编程，输出决策树模型

（1）打开原文链接：https://www.jianshu.com/p/de0ad1c793d6

（2）运行文中程序，理解决策树工作原理，会计算信息增益和信息熵；

```
from math import log
from math import log
import operator
filename ="tennis.txt"
with open(filename,"r") as file:
    lines=file.readlines()
arr=[]

def calcShannonEnt(dataSet):  # 计算数据的熵(entropy)
    numEntries=len(dataSet)  # 数据条数
    labelCounts={}
    for featVec in dataSet:
        currentLabel=featVec[-1] # 每行数据的最后一个字（类别）
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel]=0
        labelCounts[currentLabel]+=1  # 统计有多少个类以及每个类的数量
    shannonEnt=0
    for key in labelCounts:
        prob=float(labelCounts[key])/numEntries  # 计算单个类的熵值
        shannonEnt-=prob*log(prob,2)  # 累加每个类的熵值
    return shannonEnt

def createDataSet1():    # 创造示例数据
    dataSet = [['长', '粗', '男'],
               ['短', '粗', '男'],
               ['短', '粗', '男'],
               ['长', '细', '女'],
               ['短', '细', '女'],
               ['短', '粗', '女'],
               ['长', '粗', '女'],
               ['长', '粗', '女']]
    labels = ['头发','声音']  #两个特征
    return dataSet,labels

def splitDataSet(dataSet,axis,value): # 按某个特征分类后的数据
    retDataSet=[]
```

```
    for featVec in dataSet:
        if featVec[axis]==value:
            reducedFeatVec
=featVec[:axis]

reducedFeatVec.extend(featVec[axis+1:])

retDataSet.append(reducedFeatVec)
    return retDataSet

def chooseBestFeatureToSplit(dataSet):  #
选择最优的分类特征
    numFeatures = len(dataSet[0])-1
    baseEntropy = calcShannonEnt(dataSet)
# 原始的熵
    bestInfoGain = 0
    bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example
in dataSet]
        uniqueVals = set(featList)
        newEntropy = 0
        for value in uniqueVals:
            subDataSet =
splitDataSet(dataSet,i,value)
            prob
=len(subDataSet)/float(len(dataSet))
            newEntropy
+=prob*calcShannonEnt(subDataSet)     #
按特征分类后的熵
        infoGain = baseEntropy -
newEntropy  # 原始熵与按特征分类后的
熵的差值
        if (infoGain>bestInfoGain):     #
若按某特征划分后，熵值减少的最大，则
次特征为最优分类特征
            bestInfoGain=infoGain
            bestFeature = i
    return bestFeature

def majorityCnt(classList):     #按分类后
类别数量排序，比如：最后分类为 2 男 1
女，则判定为男；
    classCount={}
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote]=0
        classCount[vote]+=1
    sortedClassCount =
sorted(classCount.items(),key=operator.item
getter(1),reverse=True)
    return sortedClassCount[0][0]

def createTree(dataSet,labels):
    classList=[example[-1] for example in
dataSet]  # 类别：男或女
    if
classList.count(classList[0])==len(classList):
        return classList[0]
    if len(dataSet[0])==1:
        return majorityCnt(classList)

bestFeat=chooseBestFeatureToSplit(dataSet)
#选择最优特征
    bestFeatLabel=labels[bestFeat]
    myTree={bestFeatLabel:{}} # 分类结
果以字典形式保存
    del(labels[bestFeat])
    featValues=[example[bestFeat]     for
example in dataSet]
    uniqueVals=set(featValues)
    for value in uniqueVals:
        subLabels=labels[:]

myTree[bestFeatLabel][value]=createTree(s
plitDataSet\

(dataSet,bestFeat,value),subLabels)
    return myTree


if __name__=='__main__':
    dataSet, labels=createDataSet1()  # 创
造示列数据
    print(createTree(dataSet,  labels))    #
输出决策树模型结果
```




```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\ex8.py
{'声音': {'粗': {'头发': {'短': '男', '长': '女'}}, '细': '女'}}

Process finished with exit code 0
```

（3）根据输出结果，画出决策树；

（4）改变数据集，解决经典的打网球问题，画出决策树。

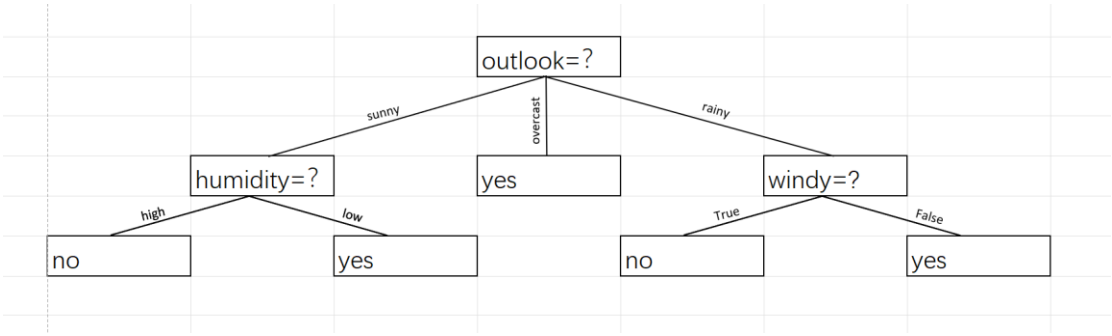
注意，样本数少时可以手动输入，当样本数较多时，应使用批量读取数据，可参考任务 2 中的.txt 文件读取。

{'outlook': {'overcast': 'yes', 'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}, 'rainy': {'windy': {'FALSE': 'yes', 'TRUE': 'no'}}}}

问题描述：今天是否去打网球（play）主要由天气（outlook）、温度（temperature）、湿度（humidity）、是否有风（windy）来确定。样本中共 14 条数据。

```
from math import log
import operator
import numpy as np
data = np.loadtxt("tennis.txt", dtype=str)
# converters  这个是一列，

def calcShannonEnt(dataSet):    # 计算数据的熵(entropy)
    numEntries=len(dataSet)    # 数据条数
    labelCounts={}
    for featVec in dataSet:
        currentLabel=featVec[-1]    # 每行数据的最后一个字（类别）
        if currentLabel not in labelCounts.keys():

labelCounts[currentLabel]=0

labelCounts[currentLabel]+=1    # 统计有多少个类以及每个类的数量
    shannonEnt=0
    for key in labelCounts:

prob=float(labelCounts[key])/numEntries # 计算单个类的熵值
        shannonEnt-=prob*log(prob,2) # 累加每个类的熵值
    return shannonEnt


def splitDataSet(dataSet,axis,value): # 按某个特征分类后的数据
    retDataSet=[]
    for featVec in dataSet:
        if featVec[axis]==value:
            reducedFeatVec=featVec[:axis]

reducedFeatVec.extend(featVec[axis+1:])

retDataSet.append(reducedFeatVec)
    return retDataSet

def chooseBestFeatureToSplit(dataSet): # 选择最优的分类特征
    numFeatures = len(dataSet[0])-1
    baseEntropy =
```

```
calcShannonEnt(dataSet)  # 原始的熵
    bestInfoGain = 0
    bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)
        newEntropy = 0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet,i,value)
            prob =len(subDataSet)/float(len(dataSet))
            newEntropy +=prob*calcShannonEnt(subDataSet)  # 按特征分类后的熵
        infoGain = baseEntropy - newEntropy  # 原始熵与按特征分类后的熵的差值
        if (infoGain>bestInfoGain):  # 若按某特征划分后，熵值减少的最大，则次特征为最优分类特征
            bestInfoGain=infoGain
            bestFeature = i
    return bestFeature

def majorityCnt(classList):      #按分类后类别数量排序，比如：最后分类为 2 男 1 女，则判定为男；
    classCount={}
    for vote in classList:
        if vote not in classCount.keys():


if __name__=='__main__':
    dataSet= [list(d)[1:] for d in

    print(createTree(dataSet, labels))
            classCount[vote]=0
        classCount[vote]+=1
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    return sortedClassCount[0][0]

def createTree(dataSet,labels):
    classList=[example[-1] for example in dataSet]  # 类别：男或女
    if classList.count(classList[0])==len(classList):
        return classList[0]
    if len(dataSet[0])==1:
        return majorityCnt(classList)

bestFeat=chooseBestFeatureToSplit(dataSet) #选择最优特征
    bestFeatLabel=labels[bestFeat]
    myTree={bestFeatLabel:{}}  # 分类结果以字典形式保存
    del(labels[bestFeat])
    featValues=[example[bestFeat] for example in dataSet]
    uniqueVals=set(featValues)
    for value in uniqueVals:
        subLabels=labels[:]

myTree[bestFeatLabel][value]=createTree(splitDataSet\

(dataSet,bestFeat,value),subLabels)
    return myTree
data[1:]]# 创造示列数据
    print(data)
    labels = list(data[0][1:-1])

# 输出决策树模型结果
```




```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\ex8_1.py
[['NO.' 'outlook' 'temperature' 'humidity' 'windy' 'play']
 ['1' 'sunny' 'hot' 'high' 'FALSE' 'no']
 ['2' 'sunny' 'hot' 'high' 'TRUE' 'no']
 ['3' 'overcast' 'hot' 'high' 'FALSE' 'yes']
 ['4' 'rainy' 'mild' 'high' 'FALSE' 'yes']
 ['5' 'rainy' 'cool' 'normal' 'FALSE' 'yes']
 ['6' 'rainy' 'cool' 'normal' 'TRUE' 'no']
 ['7' 'overcast' 'cool' 'normal' 'TRUE' 'yes']
 ['8' 'sunny' 'mild' 'high' 'FALSE' 'no']
 ['9' 'sunny' 'cool' 'normal' 'FALSE' 'yes']
 ['10' 'rainy' 'mild' 'normal' 'FALSE' 'yes']
 ['11' 'sunny' 'mild' 'normal' 'TRUE' 'yes']
 ['12' 'overcast' 'mild' 'high' 'TRUE' 'yes']
 ['13' 'overcast' 'hot' 'normal' 'FALSE' 'yes']
 ['14' 'rainy' 'mild' 'high' 'TRUE' 'no']]
{'outlook': {'rainy': {'windy': {'TRUE': 'no', 'FALSE': 'yes'}}, 'overcast': 'yes', 'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

Process finished with exit code 0
```

任务 2：直接调用决策树库，进行分类

参考原文链接,完成以下任务。

https://blog.csdn.net/zhuiyunzhugang/article/details/105544014

（1）手动生成打网球的.txt 文件(可直接复制)；

（2）运行后如果报错显示没有 pydotplus 模块，可以不安装，就把程序中“4.把决策树结构写入文件”下面对应的代码屏蔽掉，不影响结果；

（3）逐条解释运行结果；

运行结果：首先输出了决策信息，选择决策树划分标准为信息增益，测试集样本类别和预测类别准确率，

（4）解释程序中 def 函数的作用；

该程序中 def 定义了若干函数,在导入 txt 文本决策信息时使用这些函数将各种属性的取值转化为数字。

（5）详细解释下列三行语句的作用(包括各个参数)；

①data=np.loadtxt("play.tennies.txt",delimiter="",dtype=str,converters={0:outlook_type,1:temperature,2:humidity,3:windy,4:play_type})

②x, y = np.split(data,(4,),axis=1)

③x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

1 的作用时导入数据集;参数分别为文件名称,分隔符,数据存储类型为字符串,将字符串转化为对应属性取值数字的函数名称

2 的作用是对数据集进行分割 np.split：NumPy 中的函数，用于将数组在指定位置进行分割。data：要分割的数组。

3 的作用是将数据集划分为训练集和测试集： x,y：要进行分割的特征矩阵和标

签。test_size=0.3：指定测试集所占的比例，这里为 0.3，表示测试集占数据集的 30%。

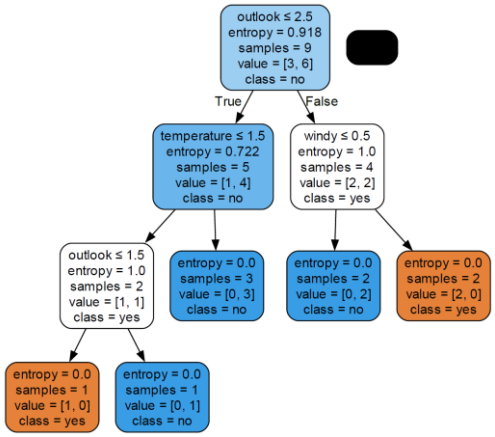
(6)添加以下两行程序，运行，写出输出结果，并解释。

```
from sklearn import metrics
print(metrics.classification_report(y_test, answer))
```

```
# 对原始数据进行分为训练数据和测试数据
import numpy as np
import graphviz
from sklearn import tree
from sklearn.model_selection import train_test_split
import pydotplus


def outlook_type(s):
    # print(s)
    it = {b'sunny': 1, b'overcast': 2, b'rainy': 3}
    return it[s]

def temperature(s):
    it = {b'hot': 1, b'mild': 2, b'cool': 3}
    return it[s]

def humidity(s):
    it = {b'high': 1, b'normal': 0}
    return it[s]

def windy(s):
    it = {b'TRUE': 1, b'FALSE': 0}
    return it[s]

def play_type(s):
    it = {b'yes': 1, b'no': 0}
    return it[s]

play_feature_E = 'outlook', 'temperature', 'humidity', 'windy'
play_class = 'yes', 'no'

#1、读入数据，并将原始数据中的数据转换为数字形式
data = np.loadtxt("play.tennies.txt", delimiter=" ", dtype=str,
                  converters={0: outlook_type, 1: temperature, 2: humidity, 3: windy,
                              4: play_type})  # converters 这个是一列，
# 第 0 列用 out—look_type 操作。
print(data)

x, y = np.split(data, (4,), axis=1)

#2、拆分训练数据与测试数据，为了进行交叉验证
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=2)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

#3、使用信息熵作为划分标准，对决策树进行训练
clf = tree.DecisionTreeClassifier(criterion='entropy')
print(clf)
clf.fit(x_train, y_train)

#4、把决策树结构写入文件
dot_data = tree.export_graphviz(clf, out_file=None,
feature_names=play_feature_E,
class_names=play_class,

filled=True, rounded=True, special_characters=True)
graph =
```

```
pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('play1.pdf')

# 系数反映每个特征的影响力。越大表示该特征在分类中起到的作用越大
print(clf.feature_importances_)

#5、使用训练数据预测，预测结果完全正确
answer = clf.predict(x_train)
y_train = y_train.reshape(-1)
print(answer)
print(y_train)
print(np.mean(answer == y_train))

#6、对测试数据进行预测，准确度较低，说明过拟合
answer = clf.predict(x_test)
y_test = y_test.reshape(-1)
print(answer)
print(y_test)
print(np.mean(answer == y_test))
```

```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\ex8_2.py
[['1' '1' '1' '0' '0']
 ['1' '1' '1' '1' '0']
 ['2' '1' '1' '0' '1']
 ['3' '2' '1' '0' '1']
 ['3' '3' '0' '0' '1']
 ['3' '3' '0' '1' '0']
 ['2' '3' '0' '1' '1']
 ['1' '2' '1' '0' '0']
 ['1' '3' '0' '0' '1']
 ['3' '2' '0' '0' '1']
 ['1' '2' '0' '1' '1']
 ['2' '2' '1' '1' '1']
 ['2' '1' '0' '0' '1']
 ['3' '2' '1' '1' '0']]
DecisionTreeClassifier(criterion='entropy')
[0.3212499  0.19476179 0.         0.48398831]
['1' '1' '1' '1' '1' '1' '0' '0' '0']
['1' '1' '1' '1' '1' '1' '0' '0' '0']
1.0
['1' '1' '1' '0' '1']
['1' '0' '1' '0' '1']
0.8

Process finished with exit code 0
```



任务 3：实际应用

结合西瓜数据集 2.0，任选一种方法，生成决策树模型或分类结果，和课本结果做比较。

watermelon2_0.txt

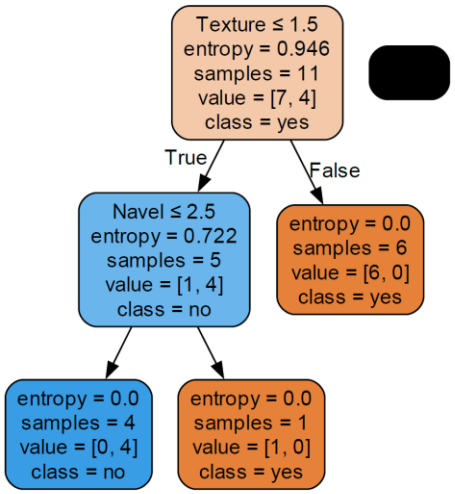
1	1	1	1	1	1	1	1
2	2	1	2	1	1	1	1
3	2	1	1	1	1	1	1
4	1	1	2	1	1	1	1
5	3	1	1	1	1	1	1
6	1	2	1	1	2	2	1
7	2	2	1	2	2	2	1
8	2	2	1	1	2	1	1
9	2	2	2	2	2	1	0
10	1	3	3	1	3	2	0
11	3	3	3	3	3	1	0
12	3	1	1	3	3	2	0
13	1	2	1	2	1	1	0
14	3	2	2	2	1	1	0
15	2	2	1	1	2	2	0
16	3	1	1	3	3	1	0
17	1	1	2	2	2	1	0

运用决策树库，进行分类，输入西瓜数据集 2.0 如上图，分类结果以及程序源码如下：

```
# 色泽{青绿：1，乌黑：2，浅白：3}        # 根蒂{蜷缩：1，稍蜷：2，硬挺：3}
```

```
# 敲声{浊响：1，沉闷：2，清脆：3}
# 纹理{清晰：1，稍糊：2，模糊：3}
# 脐部{凹陷：1，稍凹：2，平坦：3}
# 触感{硬滑：1，软粘：2}
# 好瓜{是：1，否：0}
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split
import pydotplus

melon_feature_E = 'Color', 'Root', 'Knocking', 'Texture', 'Navel', 'Touch'
melon_class = 'yes', 'no'

#1、读入数据，并将原始数据中的数据转换为数字形式
data = np.loadtxt("watermelon2_0.txt", delimiter="\t", dtype=str)
# 第 0 列用 out—look_type 操作。
print(data)

x, y = np.split(data, (6,), axis=1)

#2、拆分训练数据与测试数据，为了进行交叉验证
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=2)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

#3、使用信息熵作为划分标准，对决策树进行训练
clf = tree.DecisionTreeClassifier(criterion='entropy')
print(clf)
clf.fit(x_train, y_train)

#4、把决策树结构写入文件
dot_data = tree.export_graphviz(clf, out_file=None,
feature_names=melon_feature_E,
class_names=melon_class,

filled=True, rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('melon.pdf')

# 系数反映每个特征的影响力。越大表示该特征在分类中起到的作用越大
print(clf.feature_importances_)

#5、使用训练数据预测，预测结果完全正确
answer = clf.predict(x_train)
y_train = y_train.reshape(-1)
print(answer)
print(y_train)
print(np.mean(answer == y_train))

#6、对测试数据进行预测，准确度较低，说明过拟合
answer = clf.predict(x_test)
y_test = y_test.reshape(-1)
print(answer)
print(y_test)
print(np.mean(answer == y_test))
```

```
D:\Documents\Python\MLEX4\venv\Scripts\python.exe D:\Documents\Python\MLEX4\8_3.py
[['1' '1' '1' '1' '1' '1' '1']
 ['2' '1' '2' '1' '1' '1' '1']
 ['2' '1' '1' '1' '1' '1' '1']
 ['1' '1' '2' '1' '1' '1' '1']
 ['3' '1' '1' '1' '1' '1' '1']
 ['1' '2' '1' '1' '2' '2' '1']
 ['2' '2' '1' '2' '2' '2' '1']
 ['2' '2' '1' '1' '2' '1' '1']
 ['2' '2' '2' '2' '2' '1' '0']
 ['1' '3' '3' '1' '3' '2' '0']
 ['3' '3' '3' '3' '3' '1' '0']
 ['3' '1' '1' '3' '3' '2' '0']
 ['1' '2' '1' '2' '1' '1' '0']
 ['3' '2' '2' '2' '1' '1' '0']
 ['2' '2' '1' '1' '2' '2' '0']
 ['3' '1' '1' '3' '3' '1' '0']
 ['1' '1' '2' '2' '2' '1' '0']]
DecisionTreeClassifier(criterion='entropy')
[0.       0.       0.       0.6529947 0.3470053 0.      ]
['1' '1' '0' '1' '0' '0' '0' '0' '0' '1' '0']
['1' '1' '0' '1' '0' '0' '0' '0' '0' '1' '0']
1.0
['1' '1' '1' '0' '1' '0']
['1' '0' '1' '0' '1' '1']
0.6666666666666666

Process finished with exit code 0
```



五、实验心得

本次实验使用决策树对数据集进行分类判别，决策树学习的目的是为了产生一棵泛化能力强，既处理未见示例能力强的决策树。决策树算法中学习简单的决策规则建立决策树模型的过程非常容易理解，模型可以可视化，非常直观，应用范围广，可用于分类和回归，而且非常容易做多类别的分类，能够处理数值型和连续的样本特征。决策树学习的关键在于如何选择最优划分属性，我们希望决策树的分支节点所包含的样本尽可能属于同一类别。可以通过息增益和增益率判别，信息增益对可取数目较多的属性有所偏好，增益率准则对可取数目较少的属性有所偏好。

实验中没有设计剪枝问题，但决策树容易发生过拟合和欠拟合的问题，解决的方法有预剪枝和后剪枝。预剪枝分支没有展开，降低过拟合风险，显著减少训练时间和测试时间开销。但是增加了欠拟合风险，有些分支的当前划分虽然不能提升泛化性能，但在其基础上进行的后续划分却有可能导致性能显著提高。预剪枝基于“贪心”本质禁止这些分支展开，带来了欠拟合风险。后剪枝比预剪枝保留了更多的分支，欠拟合风险小，泛化性能往往优于预剪枝决策树，但是其训练时间开销大，后剪枝过程是在生成完全决策树之后进行的，需要自底向上对所有非叶结点逐一考察。

实验中使用了 graphviz 库对分类结果进行了可视化的处理，运用库的实现方法和函数编程的两种实现方法，在实验中出现了每次运行结果不相同的状况，是每次训练集和测试集的选取不同造成的。