实验三 感知器准则算法实验（iris数据）

**一、实验目的**

1.进一步理解感知器算法的原理和实现方法。

2.理解最小平方误差判别方法的原理和实现方法。

3.掌握多类分类器的实现方法

**二、实验内容及要求**

1．实验数据：iris数据，分为三种类型：分别为w1，w2和w3类，每种类型中包括50个四维的向量，各类别出现的概率相等。

2实验要求

1）从iris数据的每个样本中取出三个特征作为分类特征，并且将样本点画出；

2）从每个类别的数据中抽取45个样本作为训练样本，5个样本作为测试样本，

3）用感知器批处理的方法实现w1类和w2类之间，w2类和w3类，w1类和w3类之间分类器的设计，写出判别函数，画出分类面，并记录收敛的次数。

4）用感知器单步处理的方法实现w1类和w2类之间，w2类和w3类，w1类和w3类之间分类器的设计，写出判别函数，画出分类面，并记录收敛的次数。

5）对于前面用感知器算法无法迭代出权向量的类别，使用最小平方误差判别的方法求出权向量，写出判别函数，画出分类面，并记录迭代次数。

6）用多类分类器的逐步修正的方法对三个类别进行分类，写出每个类别的判别函数，画出分类面。

7）将测试样本分别应用在分类器上，对测试样本进行判别，将判别结果进行显示。

8）使用python语言来完成实验

**三、实验原理**

感知器是一种神经网络模型,对于两类线性可分的模式类w1和w2,首先对样本进行规范化处理,即将w2类的全部样本都乘以(-1),这样对于两类的所有模式样本,判别函数性质描述为:

d(y)=ａTy>0

其中,ａ=[ａ1,ａ2,…,an,an+1]T,y=[x1,x2,…,xn,xn+1]T

感知器算法通过对已知类别的训练样本集的学习,寻找一个满足上式的权向量。具体步骤如下：

1.选择 N个分属于w1和 w2类的模式样本构成的训练样本集，将训练样本写成增广向量的形式，并进行规范化处理。

2.用全部训练样本进行一轮迭代。每输入一个样本X，计算一次判别函数aTX，在一轮样本全部计算之后，若结果中有小于或等于零的情况，此时要对权向量进行修正，并且迭代次数加1。

假设进行到第k次迭代时，输入的样本中有ｍ个样本的判别函数小于或等于零，说明这m个样本被错分，这时要对权向量进行更新，即用m个错分样本值和对权向量进行修正：



3.分析结果,在这一轮的迭代中只要有一个样本的分类发生了错误,则回到步骤(2)进行下一轮迭代,用全部样本再训练一次,建立新的a(k+1),直至用全部样本进行训练都获得了正确的分类结果,迭代结束。这时的权向量值即为算法结果。

逐步修正法可以对多类进行分类，对样本进行增广化处理后，采用与感知器算法类似的单样本修正法来求解线性分类器，给每一类样本一个判别函数，算法参见教材80-81页。

**四、实验程序及结果：**

1.感知器批处理（针对12 13类样本）+伪逆矩阵法最小平方误差（针对23样本）

【batch\_and\_pinv.py】

#-\*- coding: utf-8 -\*-

import numpy as np

from matplotlib import pyplot as plt

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

# 将列表中的数据切片读入矩阵

def Read(lines,m,n):

    A = np.zeros((m, n))

    A\_row = 0  # 表示矩阵的行，从0行开始

    for line in lines:  # 把lines中的数据逐行读取出来

        list = line.strip('\n').split('\t')  # 处理逐行数据：strip表示把头尾的'\n'去掉，split表示以空格来分割行数据，然后把处理后的行数据返回到list列表中

        A[A\_row:] = list[0:5]  # 把处理后的数据放到方阵A中。list[0:4]表示列表的0,1,2,3列数据放到矩阵A中的A\_row行

        A\_row += 1  # 然后方阵A的下一行接着读

    return A

# 计算准确率

def count\_accuracy(true,false,id,Id):

    if id==Id:

        true+=1

    else:

        false+=1

    return true,false

# 利用批处理的梯度下降算法设计的感知器

def processing\_batch(A1,A2,ID,del\_line,w,p):

    '''

    (1)去除无用的数据

    例如目前需要绘制第一类和第二类的分类面，选取前三个特征进行分类

    就需要去除第一行（标签）和第五行的数据（第四个特征）

    '''

    A1 = np.delete(A1,[0,del\_line],axis=1)

    A2 = np.delete(A2,[0,del\_line],axis=1)

    '''

    (2)样本增广化、规范化处理

    '''

    # 样本增广化

    One = np.ones(25) # 1\*25的行向量值均为1

    A1 = np.insert(A1,0,values=One,axis=1) # 在第0列插入One向量（变成了列向量）

    A2 = np.insert(A2,0,values=One,axis=1)

    # 样本规范化

    A = np.concatenate((A1,-1\*A2),axis=0) # A1与A2矩阵拼接在一起

    # print(A,'\n')

    '''

    (3)采用批处理的梯度下降算法进行迭代运算

    计算权向量w和迭代次数k

    w为4\*1的向量,初始化时设定为[1,1,1,1]^T

    any(data<=0 for data in y) 意思为在y矩阵里是否存在元素<=0

    设定学习率p

    迭代的同时绘制出迭代次数与惩罚值的散点图来描绘其收敛过程

    '''

    k = 1 # 迭代次数

    # 开始迭代！！

    print("采用批处理的梯度下降算法迭代计算第%d类和%d类的权向量:"%(ID[0],ID[1]))

    while 1:

        J = 0;

        y = np.dot(A,w) # 计算样本训练值(错误样本训练值为负数)

        '''找到结果为负数的结果，求和后再取相反数作为惩罚J'''

        for i in range(0,50):

            item = y[i]

            if item <= 0:

                J = J - item

                w = w + p \* np.mat(A[i,:]).transpose()

        print("第%d次迭代中,求解惩罚值为%d"%(k,J))

        print("更新权向量值为:",w.transpose())

        # print(w.transpose)

        '''判断是否迭代完毕'''

        if all(data>0 for data in y):

            print("迭代完毕！")

            break

        '''下一轮的迭代'''

        k = k + 1

    print("经过%d次迭代后找到最终权向量求解值为:"%k,w.transpose(),"\n")

    return w

# 基于MSE设计的感知器(伪逆矩阵法)

def MSE\_pinv(A1,A2,ID,del\_line,w,b):

    '''

    (1)去除无用的数据

    例如目前需要绘制第一类和第二类的分类面，选取前三个特征进行分类

    就需要去除第一行（标签）和第五行的数据（第四个特征）

    '''

    A1 = np.delete(A1,[0,del\_line],axis=1)

    A2 = np.delete(A2,[0,del\_line],axis=1)

    '''

    (2)样本增广化、规范化处理

    '''

    # 样本增广化

    One = np.ones(25) # 1\*25的行向量值均为1

    A1 = np.insert(A1,0,values=One,axis=1) # 在第0列插入One向量（变成了列向量）

    A2 = np.insert(A2,0,values=One,axis=1)

    # 样本规范化

    A = np.concatenate((A1,-1\*A2),axis=0) # A1与A2矩阵拼接在一起

    '''

    (3)采用最小平方误差判别进行运算(伪逆矩阵法)

    '''

    w = np.linalg.pinv(A).dot(b)

    print("采用伪逆矩阵法对第%d类与第%d类分类找到最终权向量求解值为:"%(ID[0],ID[1]),w.transpose())

    return w

# 绘图函数

def Plot(A1,A2,w,ID,flag):

    fig = plt.figure()

    ax = plt.axes(projection='3d')

    plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

    plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

    '''(1)绘制散点图'''

    point1 = ax.scatter3D(A1[:,0].tolist(),A1[:,1].tolist(),A1[:,2].tolist(),c='red',marker = '+')  # type: ignore

    point2 = ax.scatter3D(A2[:,0].tolist(),A2[:,1].tolist(),A2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

    '''(2)绘制分类面'''

    '''

    三个特征的权向量格式为w=[w0,w1,w2,w3]

    则可得分类面方程为 : w1\*x + w2\*y + w3\*z + w0 = 0

    推导可得 -w3\*z = w1\*x + w2\*y + w0

    z = -(w1\*x + w2\*y + w0)/w3

    '''

    x = np.linspace(-1,8,3)

    y = np.linspace(-1,8,3)

    X,Y = np.meshgrid(x,y)

    w = w.A # 矩阵单个元素也属于一个独立矩阵，因此将matrix矩阵类型转为ndarray

    Z = -1\*(w[1]\*X+w[2]\*Y+w[0])/w[3]

    ax.plot\_surface(X,Y,Z,color='aliceblue')  # type: ignore

    ax.set(xlabel="X", ylabel="Y", zlabel="Z")

    if(flag==True):

        str1 = "训练集中为第%d类"%ID[0]

        str2 = "训练集中为第%d类"%ID[1]

    else:

        str1 = "被判别为第%d类"%ID[0]

        str2 = "被判别为第%d类"%ID[1]

    plt.legend([point1,point2],[str1,str2])

# 测试集测试函数

def Test(B1,B2,del\_line,ID,w,flag):

    true,false =0,0

    # 去除无用的数据

    B1 = np.delete(B1,del\_line,axis=1)

    B2 = np.delete(B2,del\_line,axis=1)

    # 样本增广化处理

    One = np.ones(25) # 1\*25的行向量值均为1

    B1 = np.insert(B1,1,values=One,axis=1)

    B2 = np.insert(B2,1,values=One,axis=1)

    B = np.concatenate((B1,B2),axis=0) # B1与B2矩阵拼接在一起

    res1,res2 = [],[]

    for i in range(0,50):

        B\_row = B[i] # 取出第i行

        id = B\_row[0] # 取出测试集实际标号

        B\_row = np.delete(B\_row,0) # 矩阵中删除类别号

        y = np.dot(B\_row,w)

        if y > 0:

            res1.append(B\_row)

            Id = ID[0]

        else:

            res2.append(B\_row)

            Id = ID[1]

        true,false = count\_accuracy(true,false,id,Id)

    if flag==1:

        str='感知器批处理方法'

    elif flag==2:

        str='感知器单步处理方法'

    else:

        str='感知器伪逆矩阵法的最小平方误差判别方法'

    print("基于%s对第%d类和第%d类进行分类："%(str,ID[0],ID[1]))

    print("正确个数：",true)

    print("错误个数：",false)

    print("准确率：",true/(true+false),'\n')

    # 将分类结果转换为矩阵

    res1 = np.mat(res1)

    res2 = np.mat(res2)

    # 将第一列增广列删除

    res1 = np.delete(res1,0,axis=1)

    res2 = np.delete(res2,0,axis=1)

    return res1,res2

'''

1.读取训练集和测试集

注:在此处统计行数是为了兼容不同的样本集,

因为理论上说我们事先不会知晓有多少组数据

'''

f1 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\train.txt') # 打开训练集

f2 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\test.txt') # 打开测试集

lines1 = f1.readlines() # 把全部数据文件读到一个列表lines中

lines2 = f2.readlines()

Line1 = len(lines1) # 读取训练集行数

Line2 = len(lines2) # 读取训练集列数

A = Read(lines1,Line1,5)

B = Read(lines2,Line2,5)

'''

2.将三类样本拆分

'''

# 提取三类训练集

A1,A2,A3 = A[0:25],A[25:50],A[50:75]

# 提取三类测试集

B1,B2,B3 = B[0:25],B[25:50],B[50:75]

'''

3.利用批处理的梯度下降算法设计的感知器

A1--第一类 A2--另一类

del\_line--要删除的列

ID--存储着两类的类别号

w--权向量 p--学习率

'''

w = np.ones((4,1)) # 初始权向量

del\_line = 4 # 我们选择前三个特征，因此第五列(第四个特征)删掉

p = 0.95 # 学习率

# 求解第一类和第二类的权向量w

w1 = processing\_batch(A1,A2,[1,2],del\_line,w,p)

# 求解第一类和第三类的权向量w

w2 = processing\_batch(A1,A3,[1,3],del\_line,w,p)

# 求解第二类和第三类的权向量w(采用伪逆矩阵法的最小平方误差)

# w3 = processing\_batch(A2,A3,[2,3],del\_line,w,p)

b = np.mat(np.ones((50,1)))

w3 = MSE\_pinv(A2,A3,[2,3],del\_line,w,b)

'''

4.绘制训练集的散点图以及分类面

flag为1则为训练集;flag为0则为测试集

'''

a = np.delete(A,[0,del\_line],axis=1)

a1,a2,a3 = a[0:25],a[25:50],a[50:75] # 未增广的数据集

# 第一类和第二类的分类面以及训练集散点图

Plot(a1,a2,w1,[1,2],flag=1)

plt.title('第一类和第二类训练集的分布')

# 第一类和第三类的分类面以及训练集散点图

Plot(a1,a3,w2,[1,3],flag=1)

plt.title('第一类和第三类训练集的分布')

# 第二类和第三类的分类面以及训练集散点图

Plot(a2,a3,w3,[2,3],flag=1)

plt.title('第二类和第三类训练集的分布')

'''

5.导入测试集对分类面进行检验

'''

# 检验第一类和第二类的准确率

res12,res21 = Test(B1,B2,del\_line,[1,2],w1,1)

# 检验第一类和第三类的准确率

res13,res31 = Test(B1,B3,del\_line,[1,3],w2,1)

# 检验第二类和第三类的准确率

res23,res32 = Test(B2,B3,del\_line,[2,3],w3,3)

'''

6.绘制测试集的散点图以及分类面

flag为1则为训练集;flag为0则为测试集

'''

# 第一类和第二类的分类面以及测试集集散点图

Plot(res12,res21,w1,[1,2],flag=0)

plt.title('第一类和第二类测试集的分布')

# 第一类和第三类的分类面以及测试集散点图

Plot(res13,res31,w2,[1,3],flag=0)

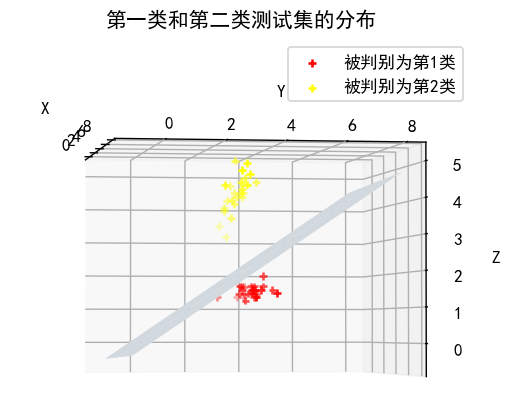
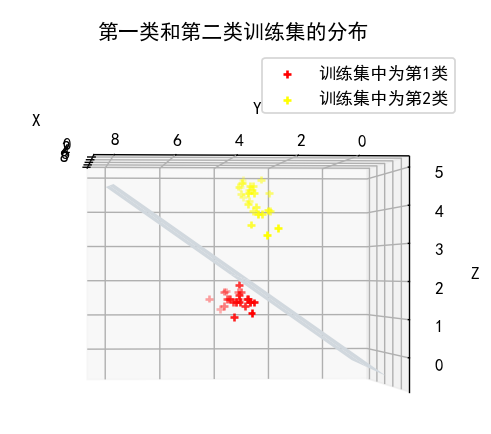
plt.title('第一类和第三类测试集的分布')

# 第二类和第三类的分类面以及测试集散点图

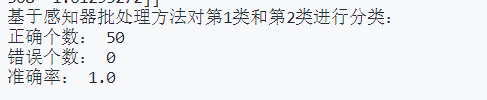
Plot(res23,res32,w3,[2,3],flag=0)

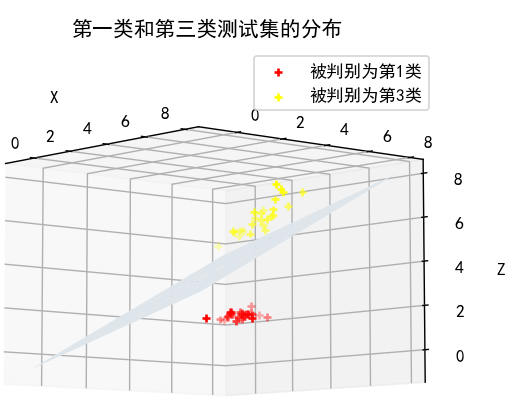
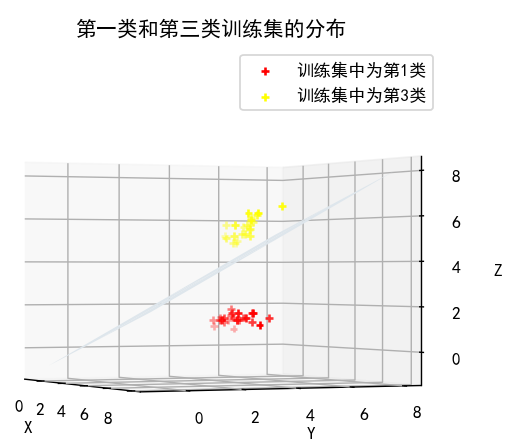
plt.title('第二类和第三类测试集的分布')

plt.show()

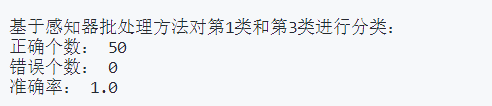


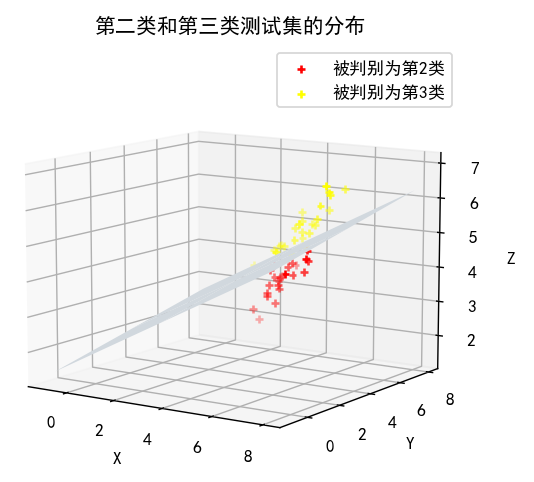
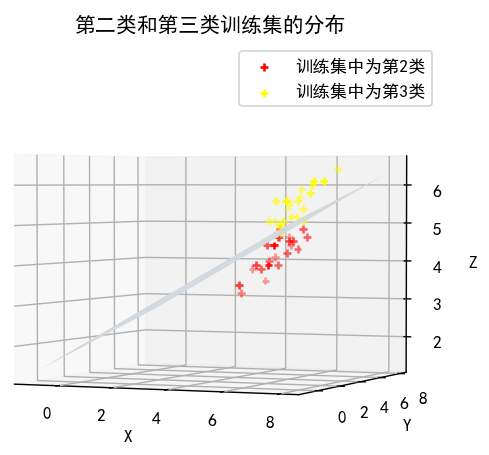






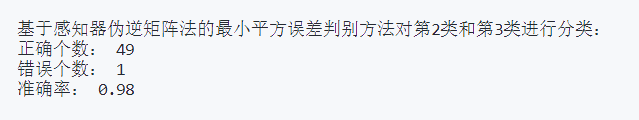








采用伪逆矩阵法不需迭代一步即可求解。



2.感知单步处理（针对12 13类样本）+单步处理最小平方误差（针对23样本）

【process\_by\_step.py】

#-\*- coding: utf-8 -\*-

import numpy as np

from matplotlib import pyplot as plt

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

# 将列表中的数据切片读入矩阵

def Read(lines,m,n):

    A = np.zeros((m, n))

    A\_row = 0  # 表示矩阵的行，从0行开始

    for line in lines:  # 把lines中的数据逐行读取出来

        list = line.strip('\n').split('\t')  # 处理逐行数据：strip表示把头尾的'\n'去掉，split表示以空格来分割行数据，然后把处理后的行数据返回到list列表中

        A[A\_row:] = list[0:5]  # 把处理后的数据放到方阵A中。list[0:4]表示列表的0,1,2,3列数据放到矩阵A中的A\_row行

        A\_row += 1  # 然后方阵A的下一行接着读

    return A

# 计算准确率

def count\_accuracy(true,false,id,Id):

    if id==Id:

        true+=1

    else:

        false+=1

    return true,false

# 利用单步修正的梯度下降算法设计的感知器

def processing\_step(A1,A2,ID,del\_line,w,p):

    '''

    (1)去除无用的数据

    例如目前需要绘制第一类和第二类的分类面，选取前三个特征进行分类

    就需要去除第一行（标签）和第五行的数据（第四个特征）

    '''

    A1 = np.delete(A1,[0,del\_line],axis=1)

    A2 = np.delete(A2,[0,del\_line],axis=1)

    '''

    (2)样本增广化、规范化处理

    '''

    # 样本增广化

    One = np.ones(25) # 1\*25的行向量值均为1

    A1 = np.insert(A1,0,values=One,axis=1) # 在第0列插入One向量（变成了列向量）

    A2 = np.insert(A2,0,values=One,axis=1)

    # 样本规范化

    A = np.concatenate((A1,-1\*A2),axis=0) # A1与A2矩阵拼接在一起

    '''

    (3)采用单步处理的梯度下降算法进行迭代运算

    计算权向量w和迭代次数k

    w为4\*1的向量,初始化时设定为[1,1,1,1]^T

    设定学习率p

    迭代的同时绘制出迭代次数与惩罚值的散点图来描绘其收敛过程

    '''

    k = 1 # 迭代次数

    count = 0 # 记录数据集里连续分类正确的个数，count=50代表全部分类成功

    i = 0 # 目前在用第i个数据检验正确性

    print("采用单步处理的梯度下降算法迭代计算第%d类和%d类的权向量:"%(ID[0],ID[1]))

    # '''开始迭代！！'''

    while 1:

        y = np.dot(A[i,:],w)

        if y <= 0:

            J = -y

            w = w + p \* np.mat(A[i,:]).transpose()

            count = 0 # 分类错误，count重置

            print("第%d次迭代中,求解惩罚值为%d"%(k,J))

            print("更新权向量值为:",w.transpose())

        else:

            J = 0

            count = count + 1

            print("第%d次迭代中,该权向量对目前所检验的样本分类正确"%k)

            print("权向量值依旧为:",w.transpose())

        '''判断是否迭代完毕'''

        if count == 50: # 连续50个样本分类正确

            print("迭代完毕！")

            break

        '''下一轮的迭代'''

        k = k + 1

        i = (i + 1) % 50

    print("经过%d次迭代后找到最终权向量求解值为:"%k,w.transpose())

    return w

# 基于MSE设计的感知器(单样本修正法/LMS):

def LMS(A1,A2,ID,del\_line,w,b,p,n\_interations):

    '''

    (1)去除无用的数据

    例如目前需要绘制第一类和第二类的分类面，选取前三个特征进行分类

    就需要去除第一行（标签）和第五行的数据（第四个特征）

    '''

    A1 = np.delete(A1,[0,del\_line],axis=1)

    A2 = np.delete(A2,[0,del\_line],axis=1)

    '''

    (2)样本增广化、规范化处理

    '''

    # 样本增广化

    One = np.ones(25) # 1\*25的行向量值均为1

    A1 = np.insert(A1,0,values=One,axis=1) # 在第0列插入One向量（变成了列向量）

    A2 = np.insert(A2,0,values=One,axis=1)

    # 样本规范化

    A = np.concatenate((A1,-1\*A2),axis=0) # A1与A2矩阵拼接在一起

    A = np.mat(A)

    '''

    (3)采用最小平方误差判别进行运算(单样本修正法LMS)

    '''

    k = 0 # 迭代次数

    i = 0 # 目前在用第i个数据检验正确性

    '''开始迭代！！'''

    # for k in range(n\_interations):

    while 1:

        gradients = np.dot(A[i,:].transpose(),np.dot(A[i,:],w)-b[i]) # 计算样本梯度

        J = p \* gradients # 平方误差作为惩罚

        w = w - J

        Error = np.linalg.norm(J)

        if Error<= 1e-10:

            break

        '''下一轮的迭代'''

        i = (i + 1) % 50

        k = k+1

    print("采用单样本修正法对第%d类与第%d类分类,经过%d次迭代后找到最终权向量求解值为:"%(ID[0],ID[1],k+1),w.transpose())

    return w

# 绘图函数

def Plot(A1,A2,w,ID,flag):

    fig = plt.figure()

    ax = plt.axes(projection='3d')

    plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

    plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

    '''(1)绘制散点图'''

    point1 = ax.scatter3D(A1[:,0].tolist(),A1[:,1].tolist(),A1[:,2].tolist(),c='red',marker = '+')  # type: ignore

    point2 = ax.scatter3D(A2[:,0].tolist(),A2[:,1].tolist(),A2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

    '''(2)绘制分类面'''

    '''

    三个特征的权向量格式为w=[w0,w1,w2,w3]

    则可得分类面方程为 : w1\*x + w2\*y + w3\*z + w0 = 0

    推导可得 -w3\*z = w1\*x + w2\*y + w0

    z = -(w1\*x + w2\*y + w0)/w3

    '''

    x = np.linspace(-1,8,3)

    y = np.linspace(-1,8,3)

    X,Y = np.meshgrid(x,y)

    w = w.A # 矩阵单个元素也属于一个独立矩阵，因此将matrix矩阵类型转为ndarray

    Z = -1\*(w[1]\*X+w[2]\*Y+w[0])/w[3]

    ax.plot\_surface(X,Y,Z,color='aliceblue')  # type: ignore

    ax.set(xlabel="X", ylabel="Y", zlabel="Z")

    if(flag==True):

        str1 = "训练集中为第%d类"%ID[0]

        str2 = "训练集中为第%d类"%ID[1]

    else:

        str1 = "被判别为第%d类"%ID[0]

        str2 = "被判别为第%d类"%ID[1]

    plt.legend([point1,point2],[str1,str2])

# 测试集测试函数

def Test(B1,B2,del\_line,ID,w,flag):

    true,false =0,0

    # 去除无用的数据

    B1 = np.delete(B1,del\_line,axis=1)

    B2 = np.delete(B2,del\_line,axis=1)

    # 样本增广化处理

    One = np.ones(25) # 1\*25的行向量值均为1

    B1 = np.insert(B1,1,values=One,axis=1)

    B2 = np.insert(B2,1,values=One,axis=1)

    B = np.concatenate((B1,B2),axis=0) # B1与B2矩阵拼接在一起

    res1,res2 = [],[]

    for i in range(0,50):

        B\_row = B[i] # 取出第i行

        id = B\_row[0] # 取出测试集实际标号

        B\_row = np.delete(B\_row,0) # 矩阵中删除类别号

        y = np.dot(B\_row,w)

        if y > 0:

            res1.append(B\_row)

            Id = ID[0]

        else:

            res2.append(B\_row)

            Id = ID[1]

        true,false = count\_accuracy(true,false,id,Id)

    if flag==1:

        str='感知器批处理方法'

    elif flag==2:

        str='感知器单步处理方法'

    else:

        str='感知器最小平方误差判别方法'

    print("基于%s对第%d类和第%d类进行分类："%(str,ID[0],ID[1]))

    print("正确个数：",true)

    print("错误个数：",false)

    print("准确率：",true/(true+false),'\n')

    # 将分类结果转换为矩阵

    res1 = np.mat(res1)

    res2 = np.mat(res2)

    # 将第一列增广列删除

    res1 = np.delete(res1,0,axis=1)

    res2 = np.delete(res2,0,axis=1)

    return res1,res2

'''

1.读取训练集和测试集

注:在此处统计行数是为了兼容不同的样本集,

因为理论上说我们事先不会知晓有多少组数据

'''

f1 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\train.txt') # 打开训练集

f2 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\test.txt') # 打开测试集

lines1 = f1.readlines() # 把全部数据文件读到一个列表lines中

lines2 = f2.readlines()

Line1 = len(lines1) # 读取训练集行数

Line2 = len(lines2) # 读取训练集列数

A = Read(lines1,Line1,5)

B = Read(lines2,Line2,5)

'''

2.将三类样本拆分

'''

# 提取三类训练集

A1,A2,A3 = A[0:25],A[25:50],A[50:75]

# 提取三类测试集

B1,B2,B3 = B[0:25],B[25:50],B[50:75]

'''

3.利用批处理的梯度下降算法设计的感知器

A1--第一类 A2--另一类

del\_line--要删除的列

ID--存储着两类的类别号

w--权向量 p--学习率

'''

w = np.ones((4,1)) # 初始权向量

del\_line = 4 # 我们选择前三个特征，因此第五列(第四个特征)删掉

p = 0.95 # 学习率

# 求解第一类和第二类的权向量w

w1 = processing\_step(A1,A2,[1,2],del\_line,w,p)

# 求解第一类和第三类的权向量w

w2 = processing\_step(A1,A3,[1,3],del\_line,w,p)

# 求解第二类和第三类的权向量w

w = 0.01\*np.ones((4,1)) # 初始权向量

p = 0.001 # 学习率

b = 0.001\*np.mat(np.ones((50,1)))

n\_interations = int(1e5) # 迭代次数

w3 = LMS(A2,A3,[2,3],del\_line,w,b,p,n\_interations)

'''

4.绘制训练集的散点图以及分类面

flag为1则为训练集;flag为0则为测试集

'''

a = np.delete(A,[0,del\_line],axis=1)

a1,a2,a3 = a[0:25],a[25:50],a[50:75] # 未增广的数据集

# 第一类和第二类的分类面以及训练集散点图

Plot(a1,a2,w1,[1,2],flag=1)

plt.title('第一类和第二类训练集的分布')

# 第一类和第三类的分类面以及训练集散点图

Plot(a1,a3,w2,[1,3],flag=1)

plt.title('第一类和第三类训练集的分布')

# 第二类和第三类的分类面以及训练集散点图

Plot(a2,a3,w3,[2,3],flag=1)

plt.title('第二类和第三类训练集的分布')

'''

5.导入测试集对分类面进行检验

'''

# 检验第一类和第二类的准确率

res12,res21 = Test(B1,B2,del\_line,[1,2],w1,2)

# 检验第一类和第三类的准确率

res13,res31 = Test(B1,B3,del\_line,[1,3],w2,2)

# 检验第二类和第三类的准确率

res23,res32 = Test(B2,B3,del\_line,[2,3],w3,3)

'''

6.绘制测试集的散点图以及分类面

flag为1则为训练集;flag为0则为测试集

'''

# 第一类和第二类的分类面以及测试集集散点图

Plot(res12,res21,w1,[1,2],flag=2)

plt.title('第一类和第二类测试集的分布')

# 第一类和第三类的分类面以及测试集散点图

Plot(res13,res31,w2,[1,3],flag=2)

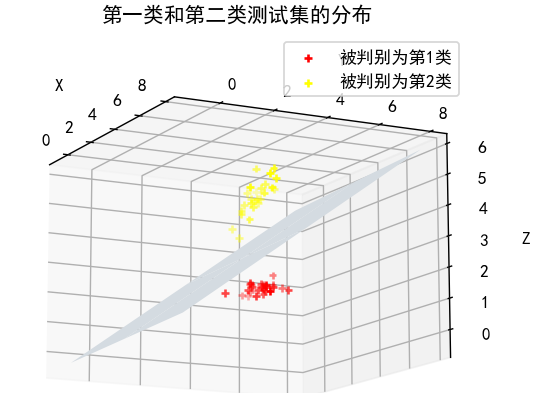
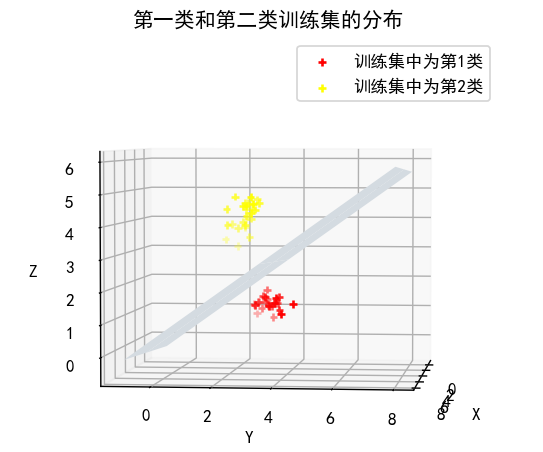
plt.title('第一类和第三类测试集的分布')

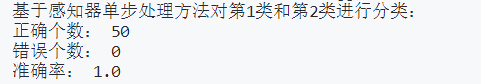
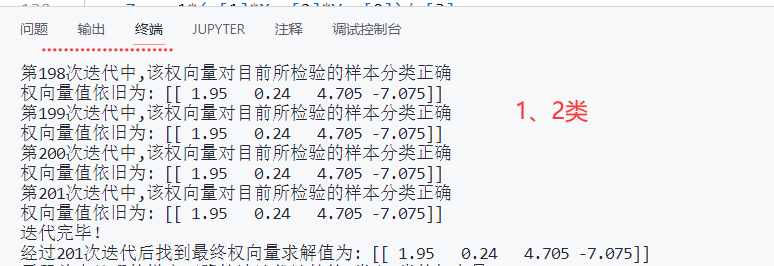
# 第二类和第三类的分类面以及测试集散点图

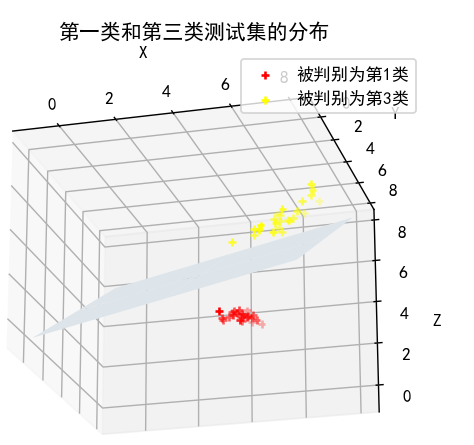
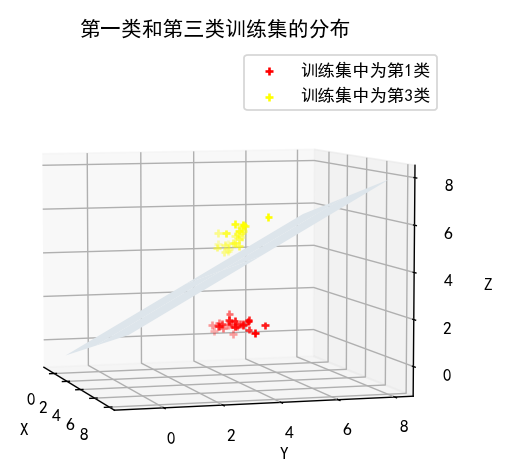
Plot(res23,res32,w3,[2,3],flag=0)

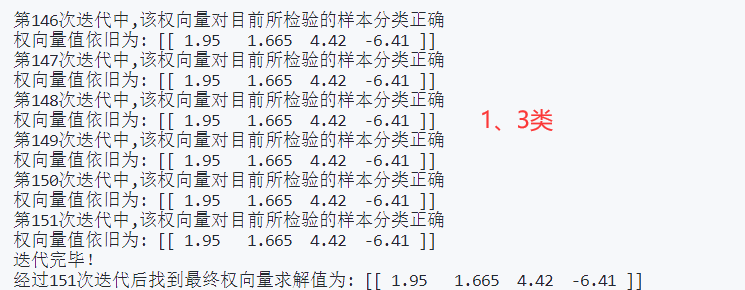
plt.title('第二类和第三类测试集的分布')

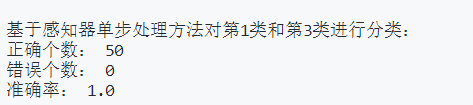
plt.show()

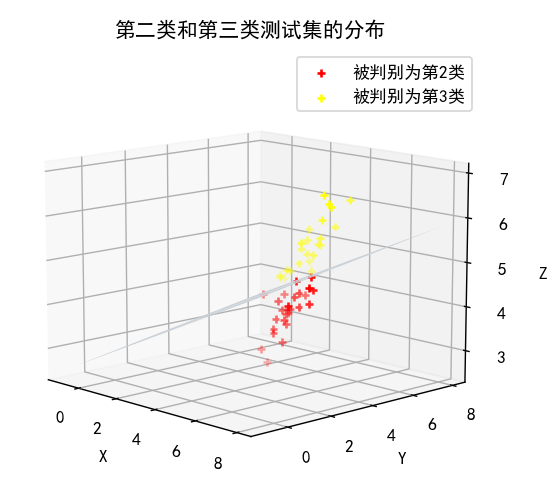
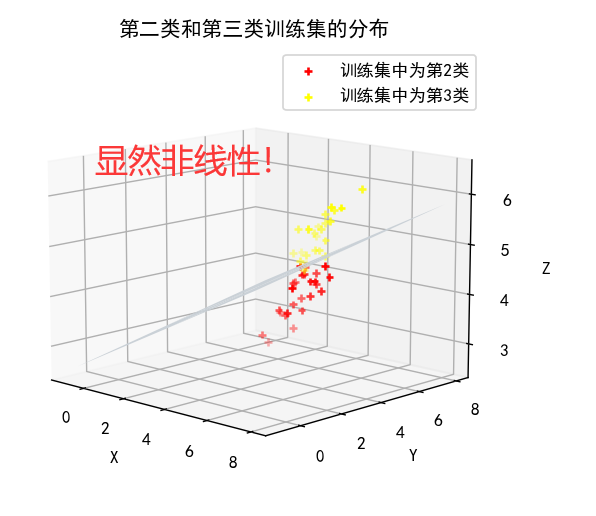






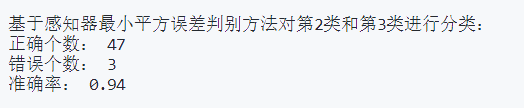






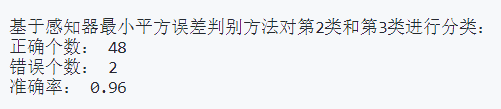


由于迭代次数过多，没有输出其过程仅仅展示结果



**实际上，将迭代次数限制在100000次的话，准确率会提高2%（避免了过拟合）**





3.多类分类器：对12、13样本线性分类；2、3样本设置误差值满足即可结束迭代

【MultiClassifation.py】

#-\*- coding: utf-8 -\*-

import numpy as np

from matplotlib import pyplot as plt

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

# 将列表中的数据切片读入矩阵

def Read(lines,m,n):

    A = np.zeros((m, n))

    A\_row = 0  # 表示矩阵的行，从0行开始

    for line in lines:  # 把lines中的数据逐行读取出来

        list = line.strip('\n').split('\t')  # 处理逐行数据：strip表示把头尾的'\n'去掉，split表示以空格来分割行数据，然后把处理后的行数据返回到list列表中

        A[A\_row:] = list[0:5]  # 把处理后的数据放到方阵A中。list[0:4]表示列表的0,1,2,3列数据放到矩阵A中的A\_row行

        A\_row += 1  # 然后方阵A的下一行接着读

    return A

# 输出权向量

def show\_w(w):

    print("第一类:",w[0].transpose())

    print("第二类:",w[1].transpose())

    print("第三类:",w[2].transpose())

# 计算准确率

def count\_accuracy(true,false,id,Id):

    if id==Id:

        true+=1

    else:

        false+=1

    return true,false

# 对样本的训练函数

'''

对样本的训练函数

A--某一类的训练样本

w--存储着三类目前的权向量，为列表[w1,w2,w3]

right\_id,该样本正确的id值

count--数据集里连续分类正确的个数

k--迭代次数 p--学习率

'''

def Train(A,w,right\_id,count,k,p,i,max\_k,Error):

    y = [np.dot(A[i,:],w[0]),np.dot(A[i,:],w[1]),np.dot(A[i,:],w[2])]

    # print("aaaaaaaaaaaaaaaaaaaaaaa",y)

    # 寻找最大值的下角标

    cur\_id = y.index(max(y))

    # 如果此时最大值的下角标有cur\_id + 1 = right\_id，即判别类别与实际类别相同

    if cur\_id + 1 == right\_id:

        count = count + 1

        # print("第%d次迭代中,该权向量对目前所检验的样本分类正确"%k)

        # print("三类权向量值依旧为")

    # 如果此时分类错误，但是迭代次数足够多且两个样本分类函数值之间差小于0.2，我们也认为是正确的

    # （因为第二类和第三类是明显的非线性，无法通过线性判别求解，允许少量误差的存在）

    elif k>=max\_k and abs(y[1]-y[2]) < Error:

        count = count + 1

        # print("第%d次迭代中,虽然分类错误，但是误差已足够小，因此也认为正确"%k)

        # print("三类权向量值依旧为")

    # 其他情况只能认为是分类错误

    else:

        temp = p \* np.mat(A[i,:]).transpose()

        w[cur\_id] = w[cur\_id] - temp

        # 因为y[right\_id-1]不是最大的，所以需要增加w[right\_id-1]

        w[right\_id - 1] = w[right\_id - 1] + temp

        count = 0 # 分类错误，count重置

        # print("第%d次迭代中,该权向量对目前所检验的样本错分成第%d类"%(k,cur\_id+1))

        # print("三类权向量值更新为")

    # show\_w(w)

    return w,count

'''

1.读取训练集和测试集

注:在此处统计行数是为了兼容不同的样本集,

因为理论上说我们事先不会知晓有多少组数据

'''

f1 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\train.txt') # 打开训练集

f2 = open('F:\\Code\\Mode Regonization\\Iris\\Fisher\\test.txt') # 打开测试集

lines1 = f1.readlines() # 把全部数据文件读到一个列表lines中

lines2 = f2.readlines()

Line1 = len(lines1) # 读取训练集行数

Line2 = len(lines2) # 读取训练集列数

A = Read(lines1,Line1,5)

B = Read(lines2,Line2,5)

'''

2.将三类样本拆分

'''

# 提取三类训练集

A1,A2,A3 = A[0:25],A[25:50],A[50:75]

# 提取三类测试集

B1,B2,B3 = B[0:25],B[25:50],B[50:75]

# A2 = A3

'''

3.去除无用的数据

例如目前需要绘制第一类和第二类的分类面，选取前三个特征进行分类

就需要去除第一列（标签）和第五列的数据（第四个特征）

'''

del\_line = 1

A1 = np.delete(A1,[0,del\_line],axis=1)

A2 = np.delete(A2,[0,del\_line],axis=1)

A3 = np.delete(A3,[0,del\_line],axis=1)

a1,a2,a3 = A1,A2,A3 # 寄存后续绘制散点图使用

'''

4.样本增广化处理

'''

One = np.ones(25) # 1\*25的行向量值均为1

A1 = np.insert(A1,0,values=One,axis=1) # 在第0列插入One向量（变成了列向量）

A2 = np.insert(A2,0,values=One,axis=1)

A3 = np.insert(A3,0,values=One,axis=1)

'''

5.采用单步处理的梯度下降算法进行迭代运算

计算权向量w和迭代次数k

w为4\*1的向量,初始化时设定为[1,1,1,1]^T

设定学习率p

'''

w = [np.ones((4,1)),np.ones((4,1)),np.ones((4,1))]

k = 1 # 迭代次数

p = 0.1 # 学习率

count = 0 # 记录数据集里连续分类正确的个数，count=75代表全部分类成功

i = 0 # 目前在用第i个数据检验正确性

max\_k = 10000 # 我们认为足够大的迭代次数，为了应对非线性关系的存在而设置

Error = 1.5 # 非线性关系中分类允许的感知器误差

# '''开始迭代！！'''

'''思路：为了使三类样本均受到相同周期的训练，

在迭代时每一个while循环均训练三个样本,

而这三个样本就是从三类中分别取出一个进行训练

避免一口气训练完成第一类样本后，结果对二三类样本严重违和

保证三类的权向量变化幅度相对较小，更快收敛到正确值。'''

while 1:

    # 对第一类样本的第i+1组数据训练

    w,count = Train(A1,w,1,count,k,p,i,max\_k,Error)

    k = k + 1

    # print("count:",count)

    '''判断是否迭代完毕'''

    if count>=75: # 连续75个样本分类正确

        # print("迭代完毕！")

        break

    # 对第二类样本的第i+1组数据训练

    w,count = Train(A2,w,2,count,k,p,i,max\_k,Error)

    k = k + 1

    # print("count:",count)

    '''判断是否迭代完毕'''

    if count>=75: # 连续75个样本分类正确

        # print("迭代完毕！")

        break

    # 对第三类样本的第i+1组数据训练

    w,count = Train(A3,w,3,count,k,p,i,max\_k,Error)

    k = k + 1

    # print("count:",count)

    '''判断是否迭代完毕'''

    if count>=75: # 连续75个样本分类正确

        # print("迭代完毕！")

        break

    '''下一轮的迭代'''

    i = (i + 1) % 25

print("经过%d次迭代后找到最终权向量求解值为:"%k)

show\_w(w)

'''

6.导入测试集对分类面进行检验

'''

true,false =0,0

# 去除无用的数据

B = np.delete(B,del\_line,axis=1)

# 样本增广化处理

One = np.ones(75) # 1\*75的行向量值均为1

B = np.insert(B,1,values=One,axis=1)

# res1,res2,res3 = [],[],[]

res = [[],[],[]]

for i in range(0,75):

    B\_row = B[i] # 取出第i行

    id = B\_row[0] # 取出测试集实际标号

    B\_row = np.delete(B\_row,0) # 矩阵中删除类别号

    y = [np.dot(B\_row,w[0]),np.dot(B\_row,w[1]),np.dot(B\_row,w[2])]

    # 最大值的下角标+1就是我们测试估计出的类别号

    # print("bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",y)

    Id = y.index(max(y)) + 1

    # 放入对应类别的列表

    res[Id - 1].append(B\_row)

    true,false = count\_accuracy(true,false,id,Id)

print("基于多类分类器对三类样本进行分类：")

print("正确个数：",true)

print("错误个数：",false)

print("准确率：",true/(true+false),'\n')

'''

7.绘制测试集的散点图以及分类面

'''

# 将分类结果转换为矩阵

res1,res2,res3 = np.mat(res[0]),np.mat(res[1]),np.mat(res[2])

# 将第一列增广列删除

res1,res2,res3 = np.delete(res1,0,axis=1),np.delete(res2,0,axis=1),np.delete(res3,0,axis=1)

fig = plt.figure()

ax = plt.axes(projection='3d')

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

'''(1)绘制散点图'''

point1 = ax.scatter3D(res1[:,0].tolist(),res1[:,1].tolist(),res1[:,2].tolist(),c='red',marker = '+')  # type: ignore

point2 = ax.scatter3D(res2[:,0].tolist(),res2[:,1].tolist(),res2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

point3 = ax.scatter3D(res3[:,0].tolist(),res3[:,1].tolist(),res3[:,2].tolist(),c='blue',marker = '+')  # type: ignore

plt.legend([point1,point2,point3],["测试集中判定为第1类","测试集中判定为第2类","测试集中判定为第3类"])

'''(2)绘制分类面

三个特征的权向量格式为w=[w0,w1,w2,w3]

则可得分类面方程为 : w1\*x + w2\*y + w3\*z + w0 = 0

推导可得 -w3\*z = w1\*x + w2\*y + w0

z = -(w1\*x + w2\*y + w0)/w3

'''

x = np.linspace(-1,8,3)

y = np.linspace(-1,8,3)

X,Y = np.meshgrid(x,y)

# 矩阵单个元素也属于一个独立矩阵，因此将matrix矩阵类型转为ndarray

w1,w2,w3 = w[0].A,w[1].A,w[2].A # type: ignore

Z1 = -1\*(w1[1]\*X+w1[2]\*Y+w1[0])/w1[3] # 第一类分类面

Z2 = -1\*(w2[1]\*X+w2[2]\*Y+w2[0])/w2[3] # 第二类分类面

Z3 = -1\*(w3[1]\*X+w3[2]\*Y+w3[0])/w3[3] # 第三类分类面

ax.plot\_surface(X,Y,Z1,color='mistyrose')  # type: ignore

ax.plot\_surface(X,Y,Z2,color='lightyellow')  # type: ignore

ax.plot\_surface(X,Y,Z3,color='cyan')  # type: ignore

'''

8.因分类面相交容易将点遮住，现单独输出三类样本的分类面

'''

# 第一类

fig = plt.figure()

ax = plt.axes(projection='3d')

point1 = ax.scatter3D(res1[:,0].tolist(),res1[:,1].tolist(),res1[:,2].tolist(),c='red',marker = '+')  # type: ignore

point2 = ax.scatter3D(res2[:,0].tolist(),res2[:,1].tolist(),res2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

point3 = ax.scatter3D(res3[:,0].tolist(),res3[:,1].tolist(),res3[:,2].tolist(),c='blue',marker = '+')  # type: ignore

plt.legend([point1,point2,point3],["测试集中判定为第1类","测试集中判定为第2类","测试集中判定为第3类"])

Z1 = -1\*(w1[1]\*X+w1[2]\*Y+w1[0])/w1[3] # 第一类分类面

ax.plot\_surface(X,Y,Z1,color='red')  # type: ignore

# 第二类

fig = plt.figure()

ax = plt.axes(projection='3d')

point1 = ax.scatter3D(res1[:,0].tolist(),res1[:,1].tolist(),res1[:,2].tolist(),c='red',marker = '+')  # type: ignore

point2 = ax.scatter3D(res2[:,0].tolist(),res2[:,1].tolist(),res2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

point3 = ax.scatter3D(res3[:,0].tolist(),res3[:,1].tolist(),res3[:,2].tolist(),c='blue',marker = '+')  # type: ignore

plt.legend([point1,point2,point3],["测试集中判定为第1类","测试集中判定为第2类","测试集中判定为第3类"])

Z2 = -1\*(w2[1]\*X+w2[2]\*Y+w2[0])/w2[3] # 第二类分类面

ax.plot\_surface(X,Y,Z2,color='yellow')  # type: ignore

# 第三类

fig = plt.figure()

ax = plt.axes(projection='3d')

point1 = ax.scatter3D(res1[:,0].tolist(),res1[:,1].tolist(),res1[:,2].tolist(),c='red',marker = '+')  # type: ignore

point2 = ax.scatter3D(res2[:,0].tolist(),res2[:,1].tolist(),res2[:,2].tolist(),c='yellow',marker = '+')  # type: ignore

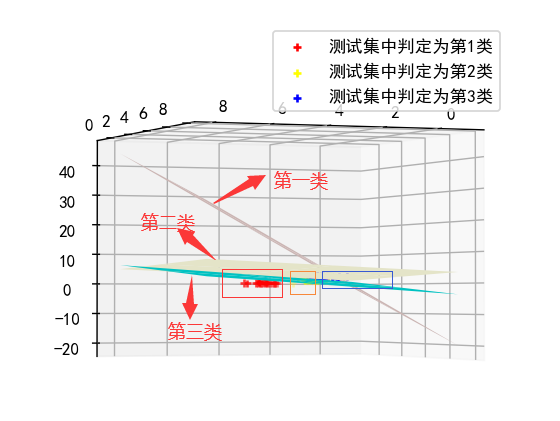
point3 = ax.scatter3D(res3[:,0].tolist(),res3[:,1].tolist(),res3[:,2].tolist(),c='blue',marker = '+')  # type: ignore

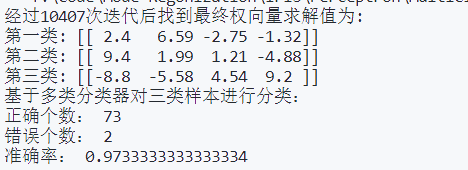
plt.legend([point1,point2,point3],["测试集中判定为第1类","测试集中判定为第2类","测试集中判定为第3类"])

Z3 = -1\*(w3[1]\*X+w3[2]\*Y+w3[0])/w3[3] # 第三类分类面

ax.plot\_surface(X,Y,Z3,color='blue')  # type: ignore

plt.show()





错分样本必为23类样本之间相互错分

**因分类面相交容易将点遮住，现单独输出三类样本的分类面**

**【平面颜色和测试集点颜色相对应】**

