**实验二** **基于Fisher准则的线性分类器设计**

1. **实验目的：**

1. 进一步了解分类器的设计概念，能够根据自己的设计对线性分类器有更深刻地认识；

2. 理解Fisher准则方法确定最佳线性分界面方法的原理，以及拉格朗日乘子求解的原理。

1. **实验内容及要求**

实验数据：IRIS数据。分为三种类型，每种类型中包括50个四维的向量。

实验模型：假设IRIS数据是正态分布的。

实验准备：在每种类型中，选择45个向量作为训练样本，5个作为测试样本。

实验要求：

1. 使用fisher判别准则，从每个类别的4个特征中选取3个特征，分别针对其中的两种类别设计线性分类器，即设计第一类和第二类，第二类和第三类，第一类和第三类之间的分类面。
2. 并用测试样本计算分类器的性能。
3. 在确定w0时可以尝试使用不同的经验值，比较分类结果是否不同？
4. 要求画出投影前的样本点，以及投影后的测试样本点、投影方向及分类面。

5.使用python语言来完成实验

**三、实验原理：**

设有一个集合包含N个d维样本，其中个属于类，个属于类。线性判别函数的一般形式可表示成,其中。

根据Fisher选择投影方向的原则，即使原样本向量在该方向上的投影能兼顾类间分布尽可能分开，类内样本投影尽可能密集的要求，用以评价投影方向的函数为：





其中：

  为类中的第个样本

为类内离散度，定义为：

为类间离散度，定义为：

上面的公式是使用Fisher准则求最佳法线向量的解，我们称这种形式的运算为线性变换，其中是一个向量，是的逆矩阵，如是维，和都是×维，得到的也是一个维的向量。

向量就是使Fisher准则函数达极大值的解，也就是按Fisher准则将维空间投影到一维空间的最佳投影方向，该向量的各分量值是对原维特征向量求加权和的权值。

以上讨论了线性判别函数加权向量的确定方法，并讨论了使Fisher准则函数极大的维向量的计算方法，但是判别函数中的另一项尚未确定，一般可采用以下几种方法确定如



或者

或当与已知时可用



当确定之后，则可按以下规则分类，





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

import numpy as np

from matplotlib import pyplot as plt

# 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

# 封装起来的Fisher线性判别函数

def Fisher(del\_line,A1,A2,B1,B2,ID):

    '''

    (1)删除矩阵中无用的数据

    '''

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

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

    '''

    (2)计算两类均值向量

    axis=0代表对矩阵的每一列求均值

    所得的mean1和mean2均为1\*3向量

    '''

    mean1 = np.mean(A1,axis=0)

    mean2 = np.mean(A2,axis=0)

    # print(mean1)

    # print(mean2)

    '''

    (3)计算总的类内离散度矩阵

    A1-mean1代表数据集每一组对应的(x,y)向量对与三个特征的均值向量相减

    该矩阵与自身转置相乘得到的n\*n矩阵就是该类别的类内离散度矩阵

    在本题是3\*3,类内离散度矩阵求和变为总类内离散度矩阵

    '''

    s1 = A1-mean1

    s1 = np.dot(s1.transpose(),s1)

    s2 = A2-mean2

    s2 = np.dot(s2.transpose(),s2)

    s = s1+s2

    # print(s)

    '''

    (4)计算投影方向和阈值

    投影方向:

        Sw^-1\*(m1-m2)

        后面的均值向量应是列向量

    阈值:

        w0 = -0.5\*(~m1+~m2)-(1/(N1+N2-2))\*ln[P(w1)/p(w2)]

        在这里我们知道在这两类中,先验概率均为0.5

        因此-(1/(N1+N2-2))\*ln[P(w1)/p(w2)]必为0

        得出:w0 = -0.5\*(~m1+~m2)

        其中~m代表所有样本在投影后的均值

        ~m1 = 投影方向 \* mean1【矩阵相乘】

        下面给~m1变量取名为mm1

    '''

    Mean = mean1-mean2

    direction = np.dot(np.linalg.inv(s),Mean.transpose()) # 投影方向

    mm1 = np.dot(mean1,direction) # 第一类在投影后的均值

    mm2 = np.dot(mean2,direction)

    w0 = -0.5\*(mm1 + mm2) # 阈值

    # print(mm1)

    # print(direction)

    # print(w0)

    '''

    (5)对测试数据进行分类

    B\_test存储着测试集，每一行均为一对特征向量

    将每一对特征向量向投影方向做投影w^T\*x

    w^T\*x+w0 > 0  则为第一类

    反之为第二类

    '''

    true,false = 0,0

    # 删除第5列(第四个特征)

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

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

    B\_test = np.zeros([50,4])

    B\_test[0:25] = B1

    B\_test[25:50] = B2

    # 遍历测试集

    res1,res2 = [],[]

    for i in range(0,50):

        B\_row = B\_test[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,direction)+w0 # 投影值

        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)

    print("基于Fisher线性判别对第%d类和第%d类进行分类："%(ID[0],ID[1]))

    print("正确个数：",true)

    print("错误个数：",false)

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

    '''

    (6)计算测试集在投影方向这条直线上的点

    将方向向量归一成单位向量

    dir\_point代表投影到直线上的点

    50\*3 \* 3\*1 \*1\*3

    '''

    dire = np.zeros((3,1))

    # 计算方向向量的单位向量，分母为向量的模

    dire[0] = direction[0]/(np.linalg.norm(direction,ord=2,axis=None,keepdims=False))

    dire[1] = direction[1]/(np.linalg.norm(direction,ord=2,axis=None,keepdims=False))

    dire[2] = direction[2]/(np.linalg.norm(direction,ord=2,axis=None,keepdims=False))

    B\_test = np.delete(B\_test,0,axis=1) # 删除第一列(类别号)

    # 计算测试集在投影方向这条直线上的点

    dire\_tran = dire.transpose()

    dir\_point1 = np.dot(np.dot(B\_test[0:25],dire),dire\_tran) # 第一类测试集的投影点

    dir\_point2 = np.dot(np.dot(B\_test[25:50],dire),dire\_tran) # 另一类测试集的投影点

    # print(direction)

    '''(7)对分类结果进行绘图'''

    fig = plt.figure()

    ax = fig.gca(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

    ax.plot3D([-5\*direction[0],2\*direction[0]],[-5\*direction[1],2\*direction[1]],[-5\*direction[2],2\*direction[2]])#画出最佳投影方向

    # 将分类结果转换为矩阵

    res1 = np.mat(res1)

    res2 = np.mat(res2)

    '''

    (1)画出分别属于两类的各点

    其中ax.scatter3D是绘制3D散点图，np.tolist为矩阵转列表

    '''

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

    point2 = ax.scatter3D(res2[:,0].tolist(),res2[:,1].tolist(),res2[:,2].tolist(),marker = '+')

    '''

    (2)画出阈值在投影前的点以及阈值投影在直线上的点

    并将两点连线，画出分类面。

    '''

    mean12 = 0.5\*(mean1+mean2) # 实际空间的阈值点

    mean21 = np.dot(np.dot(mean12,dire),dire\_tran) # 阈值在投影方向的投影点坐标

    point\_mean1 = ax.scatter3D(mean12[0],mean12[1],mean12[2])

    point\_mean2 = ax.scatter3D(mean21[0],mean21[1],mean21[2])

    # ax.plot3D([mean12[0],mean21[0]],[mean12[1],mean21[1]],[mean12[2],mean21[2]],c='black',linestyle='dashed',label='分类面')#画出分类面

    X=np.arange(0,10,0.1)

    Y=np.arange(0,10,0.1)

    X,Y=np.meshgrid(X,Y)

    Z=(-dire[0]/dire[2])\*(X+mean21[2])-(dire[1]/dire[2])\*(Y+mean21[1])-mean21[0]

    ax.plot\_surface(X,Y,Z)

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

    '''

    (3)画出测试集各点在投影方向上的投影点

    '''

    p1 = ax.scatter3D(dir\_point1[:,0].tolist(),dir\_point1[:,1].tolist(),dir\_point1[:,2].tolist(),marker='\*')

    p2 = ax.scatter3D(dir\_point2[:,0].tolist(),dir\_point2[:,1].tolist(),dir\_point2[:,2].tolist(),marker='\*')

    '''

    (4)对所绘制图像增添一些图文符号解释

    '''

    str1 = "判别为第%d类"%ID[0]

    str2 = "判别为第%d类"%ID[1]

    str3 = "测试集第%d类的投影点"%ID[0]

    str4 = "测试集第%d类的投影点"%ID[1]

    plt.legend([point1,point2,point\_mean1,point\_mean2,p1,p2],[str1,str2,'阈值点','阈值的投影点',str3,str4])

'''

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.使用封装好的Fisher判别函数

第一个参数为要删除的列号，因为我们只选择三个特征，总会有一个特征要删掉

第二、三参数为训练集的两类

第二、三参数为测试集的两类

最后一个列表参数传入的是我们想要判别类别的实际ID,即1 2 3中其中两个

'''

Fisher(4,A1,A2,B1,B2,[1,2]) # 第一类和第二类分类

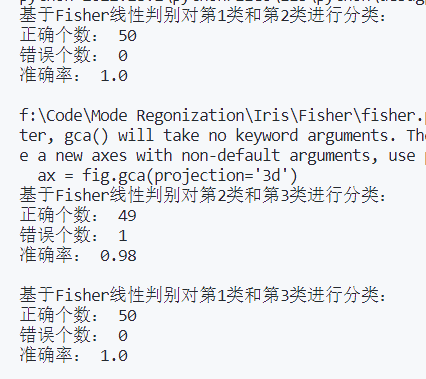
Fisher(4,A2,A3,B2,B3,[2,3]) # 第二类和第三类分类

Fisher(4,A1,A3,B1,B3,[1,3]) # 第一类和第三类分类

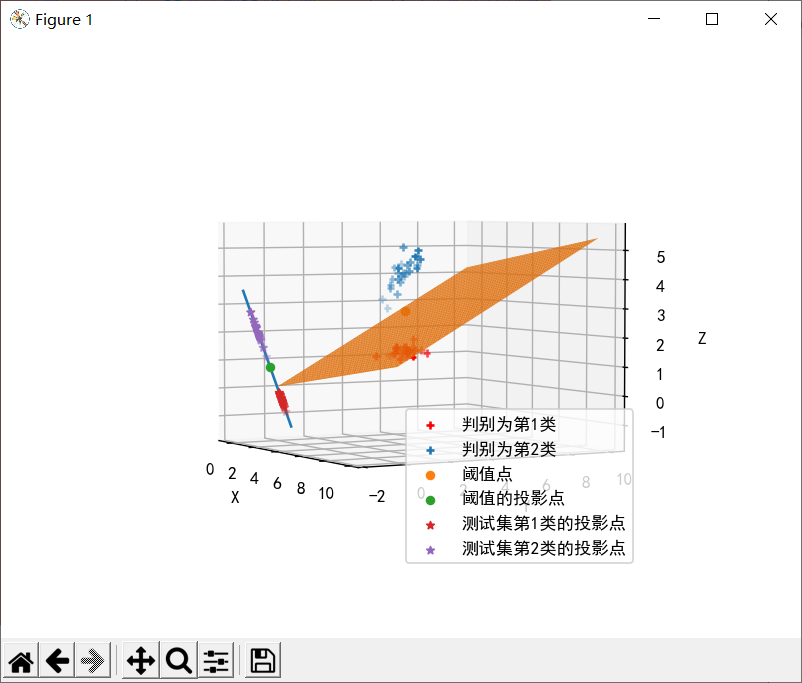
plt.show()

1. **实验代码及结果**

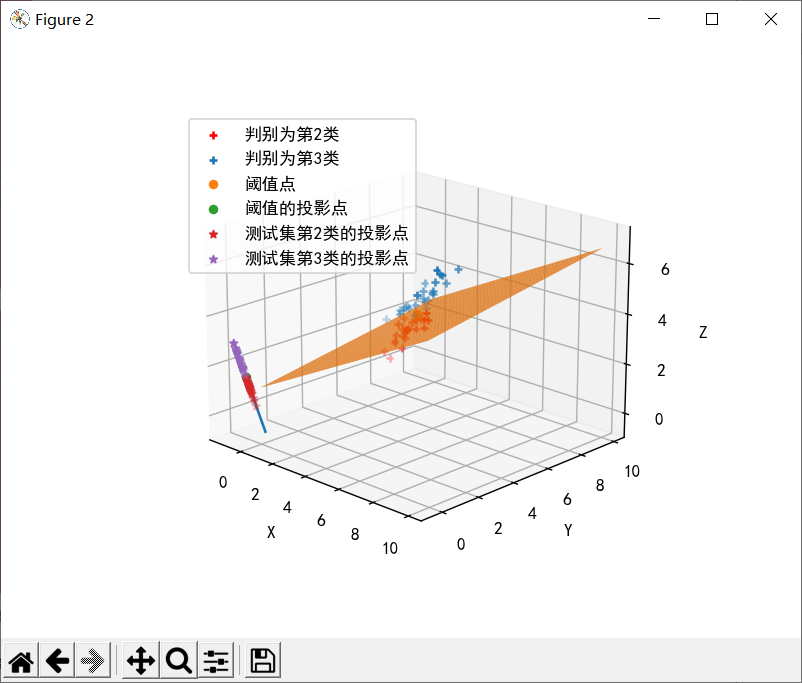
实验结果如下图所示：



在测试第二类和第三类的分类面分类效果时发现有一分错数据，准确率为98%，其他两组分类准确率均为100%。下面用可视化绘图描述一下分类效果：

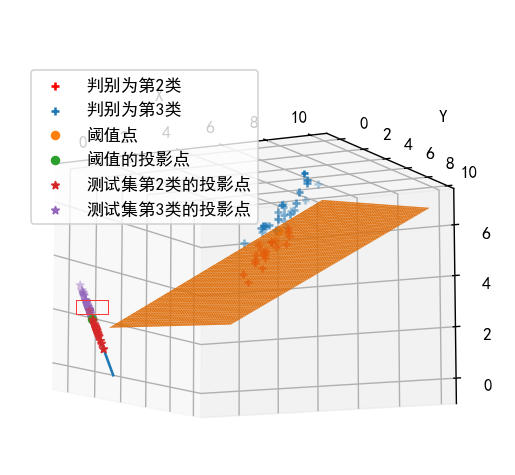


第一类与第二类样本的分类结果

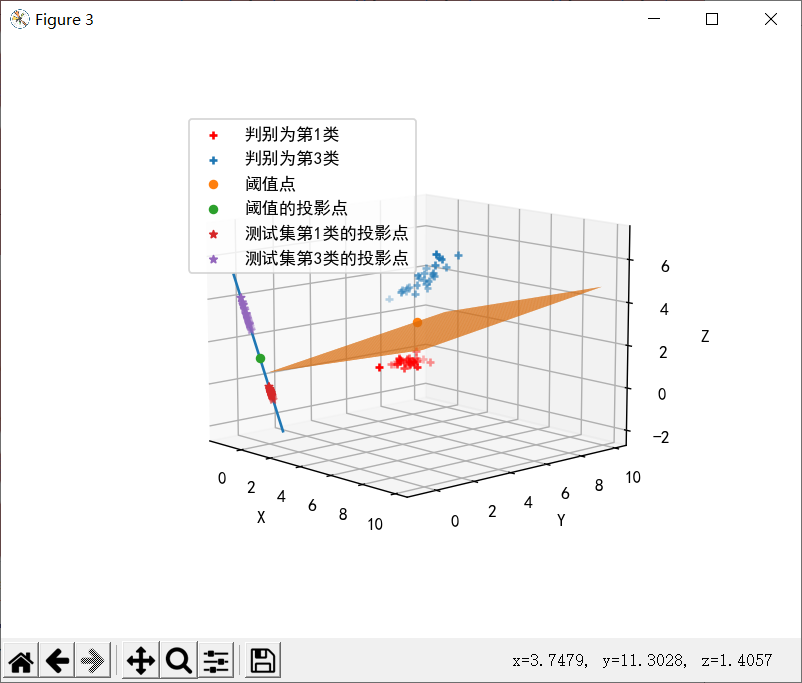


**第二类与第三类样本的分类结果**

为了更加可视化观察错分的那个样本位置，我们将坐标轴稍作旋转，如下图所示：



**第二类与第三类样本的分类结果**



**第一类与第三类样本的分类结果**