**机器学习课程**

**大 作 业**

**学 号:**  921127940122

**姓 名:**  顾翔

**专 业:** 数据科学与大数据技术

**南京理工大学网络空间安全学院**

**2023年 4 月 24 日**

**说明：共2个设计题目，每题50分，合计100，占课程成绩的50%**

**要求: 1.独立完成；2. 报告要条理清晰、图文并茂；3.双面打印，4月24日上交**

**本报告的所有代码及数据可以在我的github项目：<https://github.com/eminentgu/ML_learning_notes/tree/main/> 查看、下载**

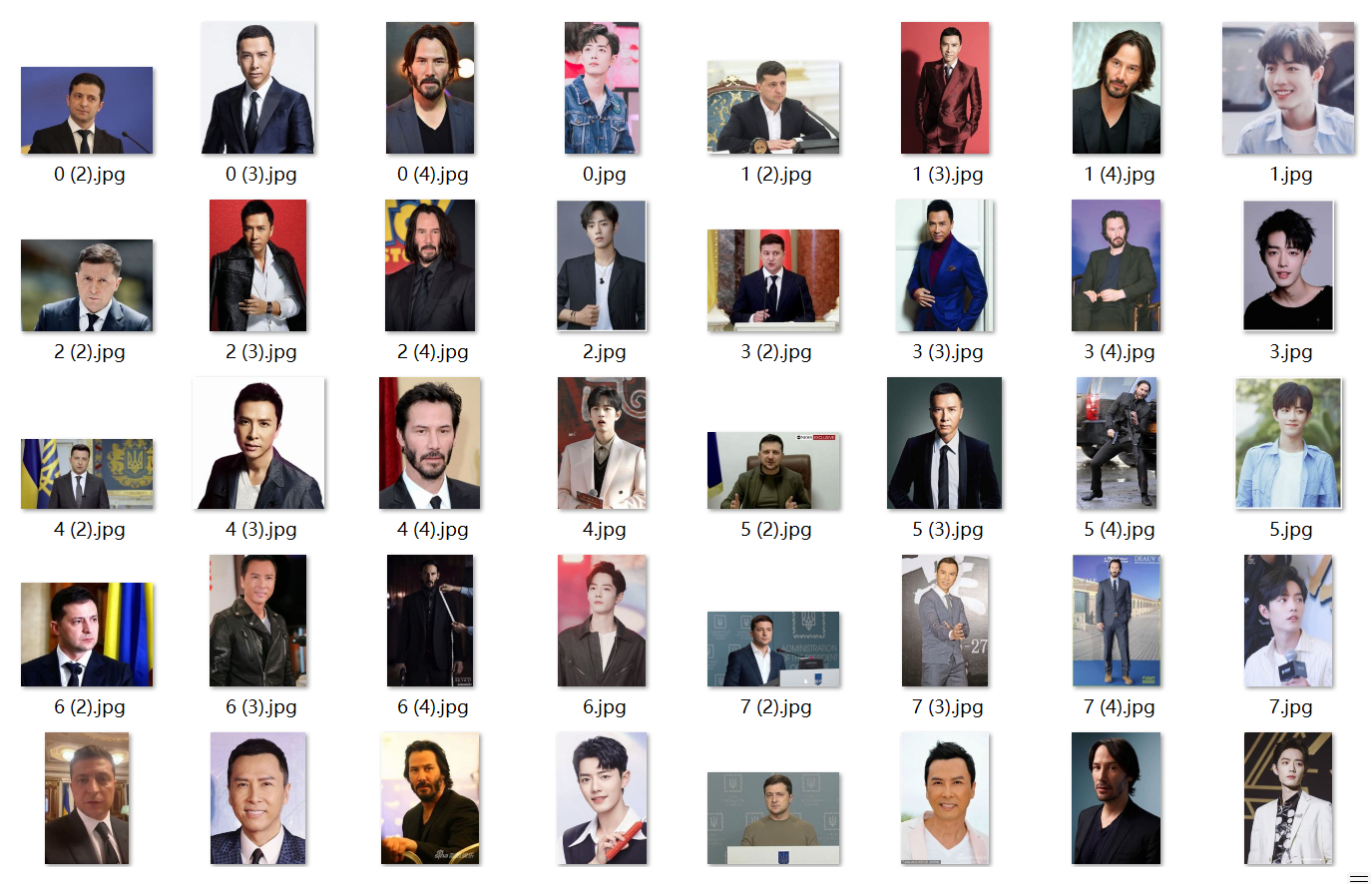
设计题目一：人脸相似性比对

**要求**：选取10个人正面头像（如选班级同学）若干张（如10张图片）。基于这些图像进行PCA降维。利用降维后的特征数据进行图像之间的相似性计算。现选取10人中任一人的头像（不能从10张中取），计算出与谁最相似。试一试100人。

本题目在两个数据集上使用两种方式实现PCA进行作答，第一个数据集4.6日之前完成的，由网络图片搜集的100+10张图片组成的数据集，通过手写PCA方式实现。在4.6日，课程群下发了学生人脸数据集，因此在原本已经完成的回答上，附加了在新的学生人脸数据集使用scikit-learn库的PCA实现。

### 一、数据搜集、标注、预处理

**数据搜集**：从网络中选取10个影视明星（肖战、泽连斯基、甄子丹、John Wick、StarLight、Lance、Homelander、A-Train、Billy、Luke Skywalker）的各10张照片，共100张作为训练集，另各选一张共10张作为测试集，共110张照片。



**图片处理**：由PreProcess.ipynb完成，其中，图像处理部分代码参考了DANNALI35的代码，使用opencv和face\_recognition库识别并修正照片中的人脸，进行人脸检测与对齐，最后将人脸照片转化为50\*50的灰度图片，储存在train和test文件夹中。



至此，完成了对数据的收集和初步处理

**数据读取与预处理：**从train文件夹以此读取灰度图，将他们展开成一维向量，再拼接起来。

from PIL import Image

import numpy as np

import os

def LoadData(path):

    files = os.listdir(path)

    flag = 1

    for i in range(len(files)):

        currentFile = path+str(i)+".jpg"

*#print(currentFile)*

        img = Image.open(currentFile)

        imgGray = img.convert('L')

        imgArray =  np.array(imgGray)

        if (flag == 1):

            flag = 0

            data = imgArray.reshape([1, -1])

        else:

            data = np.append(data, imgArray.reshape([1, -1]), axis = 0)

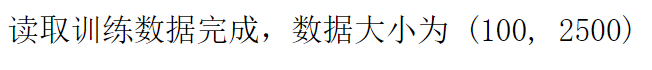
    return data

path = "DataProcessed/train/"

data = LoadData(path)

print("读取训练数据完成，数据大小为", data.shape)

**运行结果如下：**



### 二、PCA主成分分析

主成分分析（PCA）是常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中，并期望在所投影的维度上数据的方差最大，以此使用较少的维度，同时保留较多原数据的维度。

从算法上来讲，首先需要对原始数据进行去中心化，接着计算协方差矩阵，求出协方差矩阵的特征值和特征向量，提取前k个特征，即获得基向量,利用基向量，与原数据相乘即可获得降维后的数据。按照上述思路，设计PCA函数。值得一提的是，我在设计PCA函数的时候，将基向量也加入了返回值，这是因为考虑到后续将测试集数据进行降维或是可视化PCA降维结果时，需要用到这个基向量。

手写的PCA降维的函数如下

def PCA(X, components):

*# 去中心化*

    XMeaned = X - np.mean(X, axis=0)

*# 计算协方差矩阵*

    covMat = np.cov(XMeaned, rowvar=False)

*# 计算特征值和特征向量*

    eigenValues, eigenVectors = np.linalg.eig(covMat)

*# 对特征值排序*

    sortedIndex = np.argsort(eigenValues)[::-1]

    sortedEigenvalue = eigenValues[sortedIndex]

    sortedEigenvectors = eigenVectors[:,sortedIndex]

*# 取前 num\_components 个特征向量*

    eigenvectorSubset = sortedEigenvectors[:, 0:components]

*# 转换数据到新空间*

    XReduced = np.dot(eigenvectorSubset.T, XMeaned.T).T

    return XReduced,eigenvectorSubset

processedTrain,eigenvectorSubset = PCA(data, 50)

print("PCA完成，选取前50特征，降维后的数据集大小为：", processedTrain.shape)

**运行结果如下：**



接下来我做了一些可视化的工作来直观的展示PCA降维的效果，我设计了InversePCA函数来将PCA后的图片转换回原始空间，并随机选取了一张照片显示可视化结果：

import matplotlib.pyplot as plt

*# 支持中文，高清显示*

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

%matplotlib inline

%config InlineBackend.figure\_format='svg'

def InversePCA(XReduced, eigenvectors, XMean):

*# 将数据从新空间转换回原始空间*

    XOriginal = np.dot(eigenvectors, XReduced.T).T

*# 重新添加均值*

    XOriginal = XOriginal + XMean

    return np.real(XOriginal)

rawImg = data[30].reshape(50,50)

rawImg = Image.fromarray(rawImg)

rawImg = rawImg.convert('L')

inverseData = InversePCA(processedTrain, eigenvectorSubset, np.mean(data, axis=0))

inverseImg = inverseData[30].reshape(50,50)

*#print(inverseImg)*

inverseImg = Image.fromarray(inverseImg)

inverseImg = inverseImg.convert('L')

fig = plt.figure()

ax0 = fig.add\_subplot(221)

ax0.axis('off')

ax0.set\_title("训练集：原图")

ax0.imshow(rawImg, cmap = "gray")

ax1 = fig.add\_subplot(222)

ax1.axis('off')

ax1.set\_title("训练集：降维后的图片")

ax1.imshow(inverseImg, cmap = "gray")

**运行结果如下：**

****

### 三、在测试集上对PCA进行评估测试

在测试集上，首先利用计算好基向量（top k特征值）特征值集合，对新的图片数据进行PCA降维，接着，使用欧氏距离，将图片与训练集图片进行对比，其中欧氏距离最小时，即为预测图片的类型。

对新图片进行PCA降维函数如下：

def TransformPCA(X, eigenvectors, XMean):

    return np.dot(X-XMean, eigenvectors)

欧式距离计算函数如下：

def d\_ou(v1,v2):  *#np.dot(v1,v2) == np.sum(v1\*v2)*

    v1=np.array(v1)

    v2=np.array(v2)

    return np.sqrt(np.dot((v1-v2),(v1-v2)))

*#return np.sqrt(np.sum((v1-v2)\*(v1-v2)))*

计算欧氏距离、预测并可视化代码如下：

*#使用测试集图片进行测试*

nameList = ["肖战", "泽连斯基", "甄子丹", "John Wick", "StarLight", "Lance", "Homelander", "A-Train", "Billy", "Luke Skywalker"]

name = 4

img = Image.open("DataProcessed\_Famous/test/" + str(name) + ".jpg")

imgGray = img.convert('L')

test =  np.array(imgGray).reshape([1, -1])

processedTest = TransformPCA(test, eigenvectorSubset, np.mean(data, axis=0))

*#print(processedTest.shape)*

verify = processedTest

*#测试照片可视化*

rawImg = test.reshape(50,50)

rawImg = Image.fromarray(rawImg)

rawImg = rawImg.convert('L')

inverseData = InversePCA(processedTest, eigenvectorSubset, np.mean(data, axis=0))

inverseImg = inverseData[0].reshape(50,50)

inverseImg = Image.fromarray(inverseImg)

inverseImg = inverseImg.convert('L')

fig = plt.figure()

ax0 = fig.add\_subplot(221)

ax0.axis('off')

ax0.set\_title("测试集：原图")

ax0.imshow(rawImg, cmap = "gray")

ax1 = fig.add\_subplot(222)

ax1.axis('off')

ax1.set\_title("测试集：降维后的图")

ax1.imshow(inverseImg, cmap = "gray")

minDis = 1e9

dis = []

for i in range(processedTrain.shape[0]):

    dis.append(d\_ou(processedTrain[i],verify[0]))

averageDis = 0

for i in range(len(dis)):

    averageDis = averageDis + dis[i]

*#print(i)*

    if((i + 1) % 10 == 0):

        averageDis = averageDis/10

        print("测试照片与", nameList[int((i + 1) / 10) - 1],"的平均欧式距离是：", averageDis)

        if(averageDis < minDis):

            minDis = averageDis

            ansName = int((i + 1) / 10) - 1

        averageDis = 0

print("\n欧氏距离最小为：",minDis,",即测试照片预测为：",nameList[ansName], "\n测试照片实际上是", nameList[name])

**运行结果如下：可以看到，程序预测时在大多数情况下是正确的。**



### 四、学生数据集下实现

4.6日，课程群下发了学生人脸数据集，因此在原本已经完成的回答上，附加了在新的学生人脸数据集使用scikit-learn库的PCA实现。

其中，数据处理部分，观察到训练数据已经是人脸了，因此只要简单的转化为灰度图，而在测试数据中，41011.jpg、41012.jpg、41013.jpg这三张图片需要裁剪人脸，其他图片转化为灰度图即可

数据加载、欧氏距离计算等与在名人数据集下相同

以下为使用sklearn库进行PCA、验证测试集、可视化的代码

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

pca.fit(data)

processedTrain = pca.transform(data)

print("PCA完成，选取前2特征，降维后的数据集大小为：", processedTrain.shape)

print("开始测试")

name = 0

img = Image.open("DataProcessed\_Student/test/" + str(name) + ".jpg")

imgGray = img.convert('L')

test =  np.array(imgGray).reshape([1, -1])

processedTest = pca.transform(test)

*#print(processedTest)*

verify = processedTest

dis = []

minDis = 1e9

for i in range(processedTrain.shape[0]):

    dis.append(d\_ou(processedTrain[i],verify[0]))

    print("与训练集的第", i, "张照片的欧氏距离为：", dis[i])

    if (dis[i] < minDis):

        minDis = dis[i]

        ansName = i

print("与测试图片欧氏距离最小的是", ansName, ".jpg")

*#可视化*

rawImg = test.reshape(50,50)

rawImg = Image.fromarray(rawImg)

rawImg = rawImg.convert('L')

inverseData = pca.inverse\_transform(processedTest)

inverseImg = inverseData[0].reshape(50,50)

inverseImg = Image.fromarray(inverseImg)

inverseImg = inverseImg.convert('L')

trainImg = Image.open("DataProcessed\_Student/train/" + str(ansName) + ".jpg")

fig = plt.figure()

ax0 = fig.add\_subplot(231)

ax0.axis('off')

ax0.set\_title("测试集：原图")

ax0.imshow(rawImg, cmap = "gray")

ax1 = fig.add\_subplot(232)

ax1.axis('off')

ax1.set\_title("测试集：降维后的图")

ax1.imshow(inverseImg, cmap = "gray")

ax2 = fig.add\_subplot(233)

ax2.axis('off')

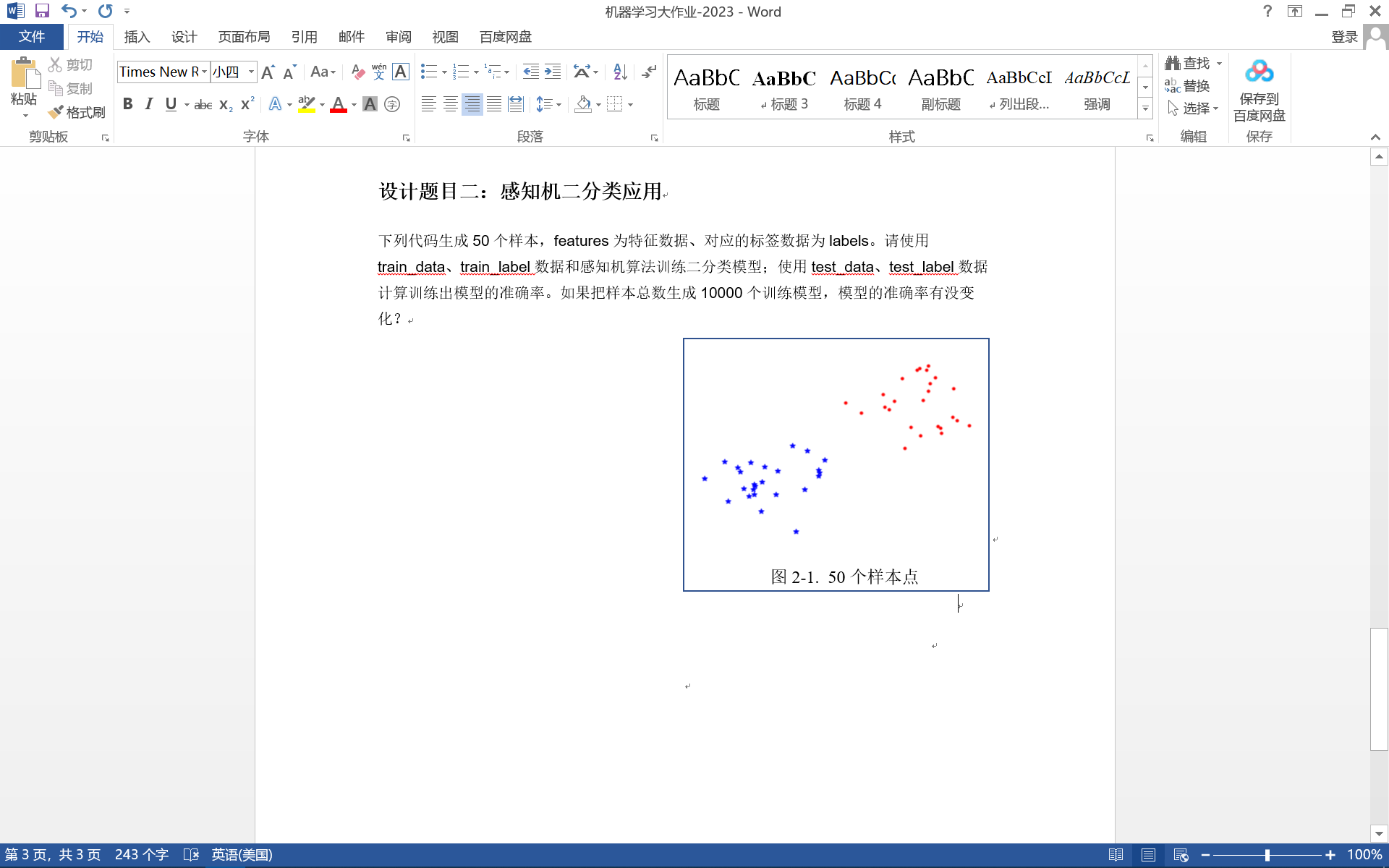
ax2.set\_title("训练集："+ str(ansName) + ".jpg")

ax2.imshow(trainImg, cmap = "gray")

**运行结果如下：可以看到，程序预测时在大多数情况下是正确的。**

****

设计题目二：感知机二分类应用

**要求**：下列代码生成50个样本，features为特征数据、对应的标签数据为labels。请使用train\_data、train\_label数据训练一个二分类感知机模型；

（1）使用test\_data、test\_label数据计算训练出模型的准确率。如果把样本总数生成10000个训练模型，模型的准确率有没变化？

（2）针对10000个样本，请分别使用贝叶斯模型、支持向量机模型进行分类，与感知机比较一下分类效果。

### 数据生成、可视化等相关前期准备

*#数据生成*

import numpy as np

import matplotlib.pyplot as plt

*# 支持中文，高清显示*

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

%matplotlib inline

%config InlineBackend.figure\_format='svg'

numbers=50

N=numbers\*70//100

class1 = np.random.randn(numbers//2,2)

*#生成第一类25个2维特征数据*

class2 = np.random.randn(numbers//2,2)+np.array([5,5])

*#生成第二类25个2维特征数据*

label1 = [0]\*(numbers//2)  *#第一类标签0*

label2 = [1]\*(numbers-numbers//2) *#第二类标签1*

features = np.vstack((class1,class2))

labels = label1+label2

np.random.seed(100)

np.random.shuffle(features)

np.random.seed(100)

np.random.shuffle(labels)

train\_data=features[:N]

train\_label=labels[:N]

test\_data=features[N:]

test\_label=labels[N:]

plt.figure()

plt.title("生成的" + str(numbers) + "个数据点")

plt.scatter(class1[:,0],class1[:,1],c='b',marker='\*')

plt.scatter(class2[:,0],class2[:,1],c='r',marker='.')

plt.show()

*#可视化*

def VisTrainData(trainDataX,trainDataY):

*#在训练集上可视化*

    xTrainSet1=[]; yTrainSet1=[]

    xTrainSet2=[]; yTrainSet2=[]

    for i,v in enumerate(trainDataX):

        if trainDataY[i]==0:

            xTrainSet1.append(v[0])

            yTrainSet1.append(v[1])

        else:

            xTrainSet2.append(v[0])

            yTrainSet2.append(v[1])

    x\_points = np.linspace(-2, 10, 12)

    plt.scatter(xTrainSet1, yTrainSet1,c='skyblue',marker='.', label = "类别1")

    plt.scatter(xTrainSet2, yTrainSet2,c='pink',marker='.', label = "类别2")

    plt.title("在训练集（" + str(len(train\_data)) + "个点)上的可视化")

    plt.legend()

    return plt

*#plt.show()*

def VisTestData(test\_data,test\_label,pred\_label):

    wrong\_data = []

    for i in range(len(test\_label)):

        if (pred\_label[i] != test\_label[i]):

            wrong\_data.append(test\_data[i])

*#在测试集上可视化*

    xTestSet1=[]; yTestSet1=[]

    xTestSet2=[]; yTestSet2=[]

    for i,v in enumerate(test\_data):

        if test\_label[i]==0:

            xTestSet1.append(v[0])

            yTestSet1.append(v[1])

        else:

            xTestSet2.append(v[0])

            yTestSet2.append(v[1])

    wrongX = []

    wrongY = []

    for i in range(len(wrong\_data)):

        wrongX.append(wrong\_data[i][0])

        wrongY.append(wrong\_data[i][1])

    plt.scatter(xTestSet1, yTestSet1,c='skyblue',marker='.', label = "类别1")

    plt.scatter(xTestSet2, yTestSet2,c='pink',marker='.', label = "类别2")

    plt.scatter(wrongX, wrongY,c='none',marker='o',edgecolors='r', label = "预测错误的点")

    plt.title("在测试集（" + str(len(test\_data)) + "个点)上的可视化")

    plt.legend()

    return plt

*#plt.show()*

### 二、感知机训练算法

感知机的训练算法如下：

初始化参数w = 0, b = 0

repeat

从训练集随机采样一个样本(xi, yi)

计算感知机的输出 t = f(WTX + b)

如果t ≠ yi：

w′ ← w + n(t- yi)\*xi

b′ ← b + n(t- yi)

until 训练次数达到要求

输出：分类网络参数w和b

其中n为学习率。

感知机训练代码实现如下：

*#感知机训练算法*

import numpy as np

import random

from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

%matplotlib inline

%config InlineBackend.figure\_format='svg'

*# 样本点集*

trainDataX = train\_data

*# 样本标签*

trainDataY = train\_label

*# 权重*

weight = [0, 0]

*# 偏置量*

bias = 0

*# 学习速率,控制每一步调整权的幅度*

learnRate = 0.2

*# 输入最大训练次数*

trainNum =5000

*# 激活函数*

def fun(n):

    if n > 0:

        return 1

    else:

        return 0

def training():

    global weight, bias

    for j in range(trainNum):

*# 从点集中随机取数*

        i = random.randint(0, len(trainDataX) - 1)

*# 获取分类结果*

        result = fun(np.dot(trainDataX[i], weight) + bias)

*# 若点分类正确,则后面代码不需要执行*

        if result == trainDataY[i]:

            continue

*# 更新权重*

*# 等价于下面两个式子*

        weight = weight + np.dot((trainDataY[i] - result), trainDataX[i]) \

                 \* learnRate

*# weight[0] = weight[0] + (trainDataY[i] - result) \* \*

*#             trainDataX[i][0] \* learnRate*

*# weight[1] = weight[1] + (trainDataY[i] - result) \* \*

*#             trainDataX[i][1] \* learnRate*

*#print(str(result) + '    ' + str(trainDataY[i]))*

*# 更新偏移量*

        bias = bias + (trainDataY[i] - result) \* learnRate

*#print("Update weight and bias: ")*

    print("训练完成,weight: ", weight, "bias ", bias)

training()

pred\_label = []

for i in range(test\_data.shape[0]):

    pred = fun(weight[0] \* test\_data[i][0] + weight[1] \* test\_data[i][1] + bias)

    pred\_label.append(pred)

wrong\_data = []

for i in range(len(test\_label)):

    if (pred\_label[i] != test\_label[i]):

        wrong\_data.append(test\_data[i])

accuracy = (len(test\_data)-len(wrong\_data))/len(test\_data)

print("在测试集的准确率为：", accuracy)

plt = VisTrainData(trainDataX,trainDataY)

x\_points = np.linspace(-2, 10, 12)

y\_ = -(weight[0] \* x\_points + bias) / weight[1]

plt.plot(x\_points, y\_, label = "拟合")

plt.legend()

plt.show()

plt = VisTestData(test\_data,test\_label,pred\_label)

x\_points = np.linspace(-2, 10, 12)

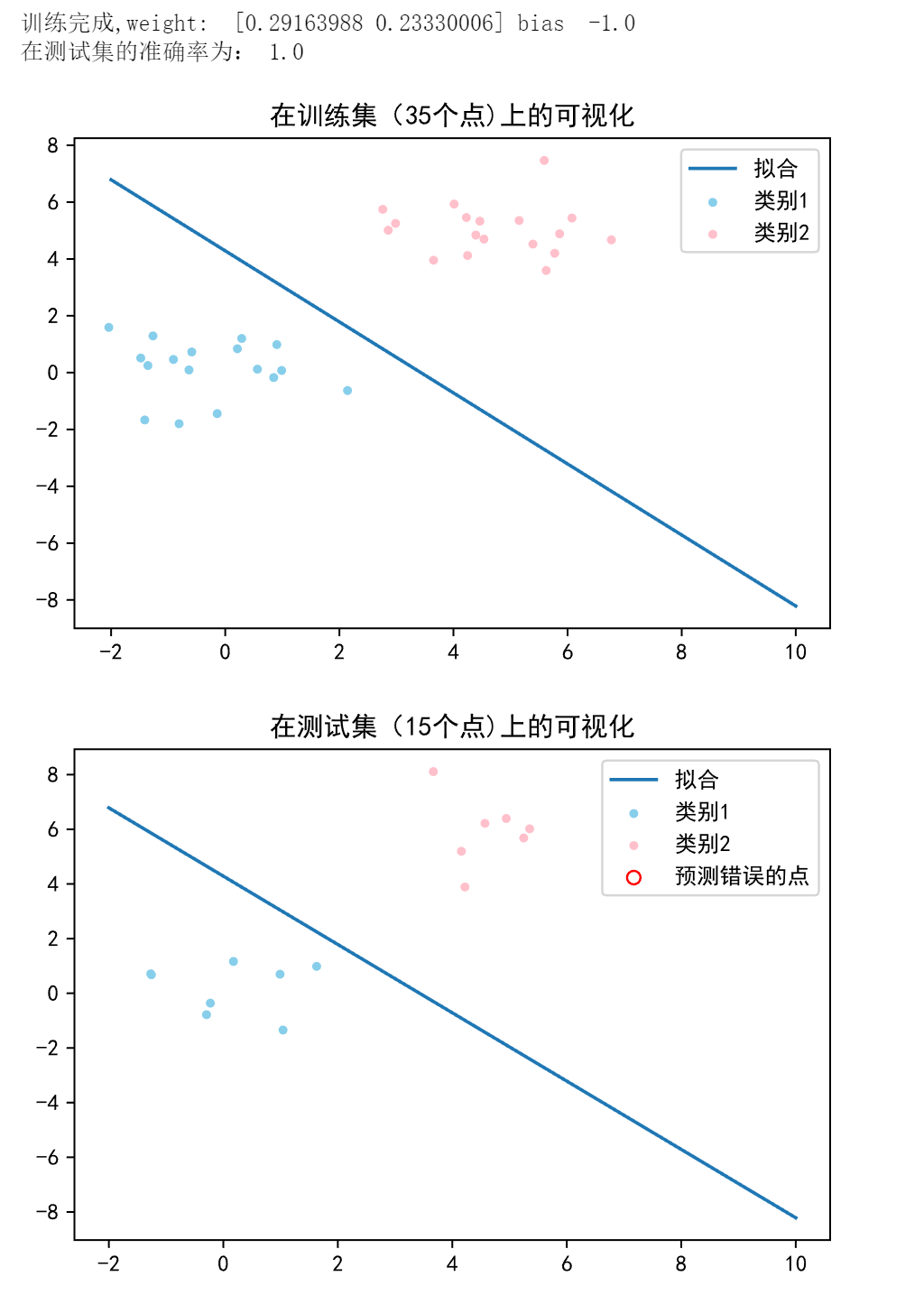
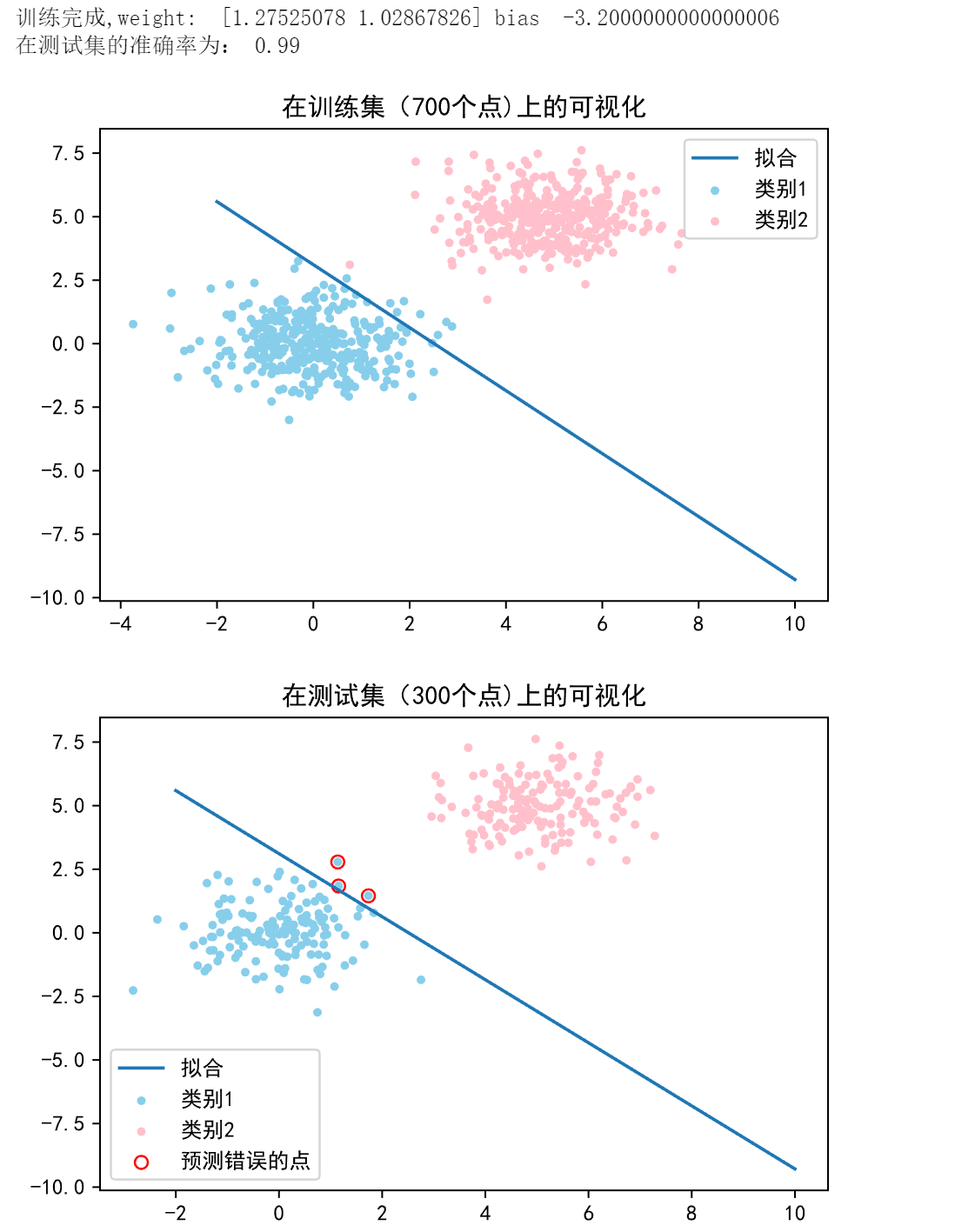
y\_ = -(weight[0] \* x\_points + bias) / weight[1]

plt.plot(x\_points, y\_, label = "拟合")

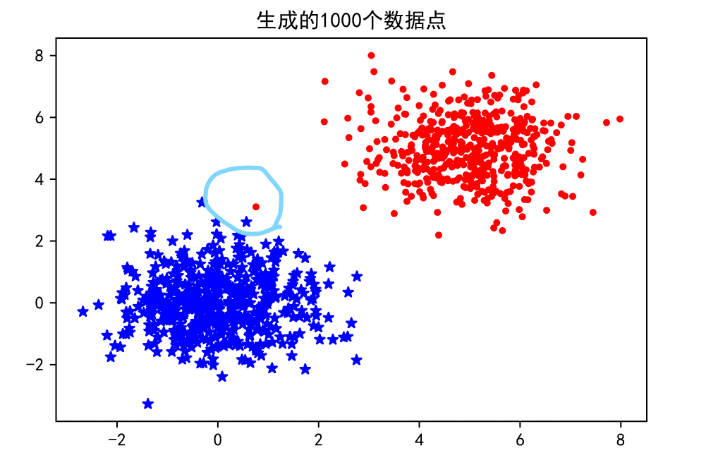
plt.legend()

plt.show()

**在总数据集大小为50（上）和1000（下）个点的情况下，运行结果分别如下：**

** **

可以看到，在1000个数据点下的准确性比50个点下的较低，因为根据题目中提供的生成数据点的方法，当生成的点较多时，可能会出现一些类别的点之间的距离很近(如图)，这对感知机的分类造成了一定的影响，准确度下降。



### 三、使用贝叶斯分类器实现二分类

贝叶斯分类器实现原理如下：

1. 统计训练集中每个类别的先验概率𝑃(𝑌=𝑦)，即在所有训练样本中，类别𝑦出现的概率。

2. 对于每个特征𝑥𝑖，计算在给定类别𝑦下的条件概率𝑃(𝑋=𝑥𝑖|𝑌=𝑦)，即在所有属于类别𝑦的训练样本中，特征𝑥𝑖出现的概率。

3. 对于一个新的测试样本𝑥=(𝑥1,𝑥2,...,𝑥𝑛)，计算其属于每个类别的后验概率𝑃(𝑌=𝑦|𝑥)，即给定特征𝑥，该样本属于类别𝑦的概率。

4. 选择后验概率最大的类别作为预测结果。

以下是在10000个点下的贝叶斯分类器实现代码：

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

*# 创建模型并拟合数据*

model = GaussianNB()

model.fit(train\_data, train\_label)

*# 预测新数据*

pred\_label = model.predict(test\_data)

*# 计算准确率*

accuracy = accuracy\_score(np.array(test\_label), pred\_label)

*#print("预测结果：", y\_pred)*

print("准确率：", accuracy)

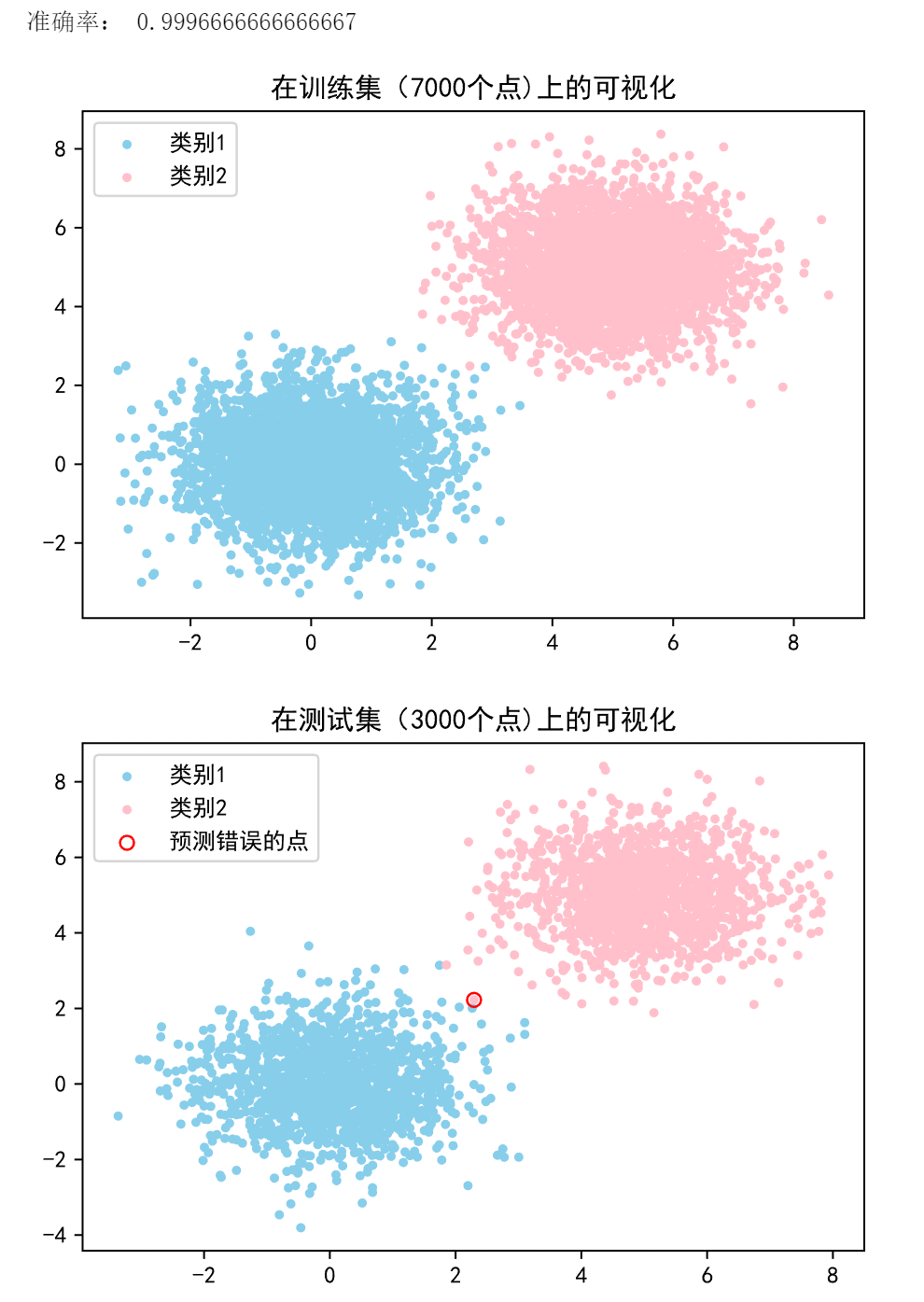
plt = VisTrainData(train\_data, train\_label)

plt.show()

plt = VisTestData(test\_data,test\_label,pred\_label)

plt.show()

贝叶斯分类器在测试集分类结果如下：



### 四、使用SVM实现二分类

支持向量机（SVM）是一种常用的分类算法，它的基本思想是找到能够最大化不同类别之间边界的超平面，从而实现分类。

SVM的实现代码如下：

*#SVM*

from sklearn import svm

from sklearn.metrics import accuracy\_score

*# 创建模型并拟合数据*

model = svm.SVC(kernel='linear')

model.fit(train\_data, train\_label)

*# 预测新数据*

pred\_label = model.predict(test\_data)

*# 计算准确率*

accuracy = accuracy\_score(np.array(test\_label), pred\_label)

*#print("预测结果：", y\_pred)*

print("准确率：", accuracy)

*#获取超平面相关参数*

w = model.coef\_[0]

a = -w[0] / w[1]

xx = np.linspace(-4, 10)

yy = a \* xx - (model.intercept\_[0]) / w[1]

b = model.support\_vectors\_[0]

yy\_down = a \* xx + (b[1] - a \* b[0])

b = model.support\_vectors\_[-1]

yy\_up = a \* xx + (b[1] - a \* b[0])

plt = VisTrainData(train\_data, train\_label)

plt.plot(xx,yy\_up,c = 'pink',linestyle = '--')

plt.plot(xx,(yy\_up+yy\_down)/2,c = "red",linestyle = '--')

plt.plot(xx,yy\_down,c = 'skyblue',linestyle = '--')

plt.show()

plt.show()

plt = VisTestData(test\_data,test\_label,pred\_label)

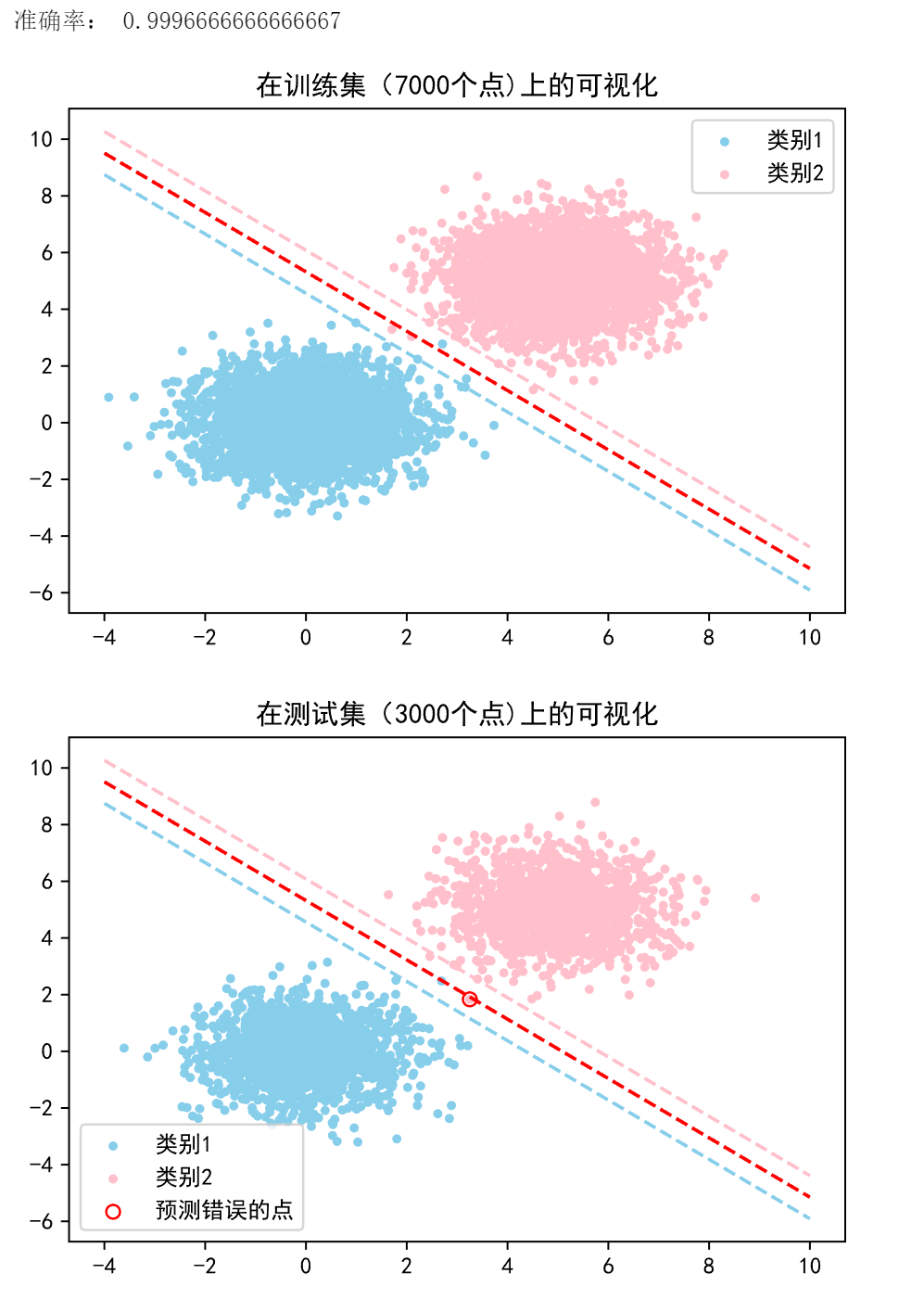
plt.plot(xx,yy\_up,c = 'pink',linestyle = '--')

plt.plot(xx,(yy\_up+yy\_down)/2,c = "red",linestyle = '--')

plt.plot(xx,yy\_down,c = 'skyblue',linestyle = '--')

plt.show()

SVM在测试集分类结果如下：



经过了100次10000个测试点下进行的实验对比后，获得了感知机、贝叶斯分类器、SVM三种方法的准确率，可以看到，在这个二分类问题上，单层感知机的准确率表现略逊于贝叶斯和SVM。

