Python实现“感知机”算法实验报告

2213530 张禹豪 智能科学与技术

一、实验目的

本次实验旨在深入理解机器学习理论课程中的感知机模型，充分掌握感知机模型的算法原理与编程实现。

二、实验原理

感知机是二类分类的线性分类模型，其由输入空间到输出空间对应以下函数：

$$f(x) = sign(w\cdot x + b)$$

其中，w和b为感知机模型参数，w叫做权值或权值向量，b叫做偏置。

感知机存在如下的几何解释：线性方程

$$w\cdot x + b = 0$$

对应于特征空间R^n中的一个超平面S，其中w是超平面的法向量，b是超平面的截距。位于两部分的点（特征向量）被超平面划为正负两类。所以只要找到一个合适的w和b将两个类别分开即实现了二分分类。

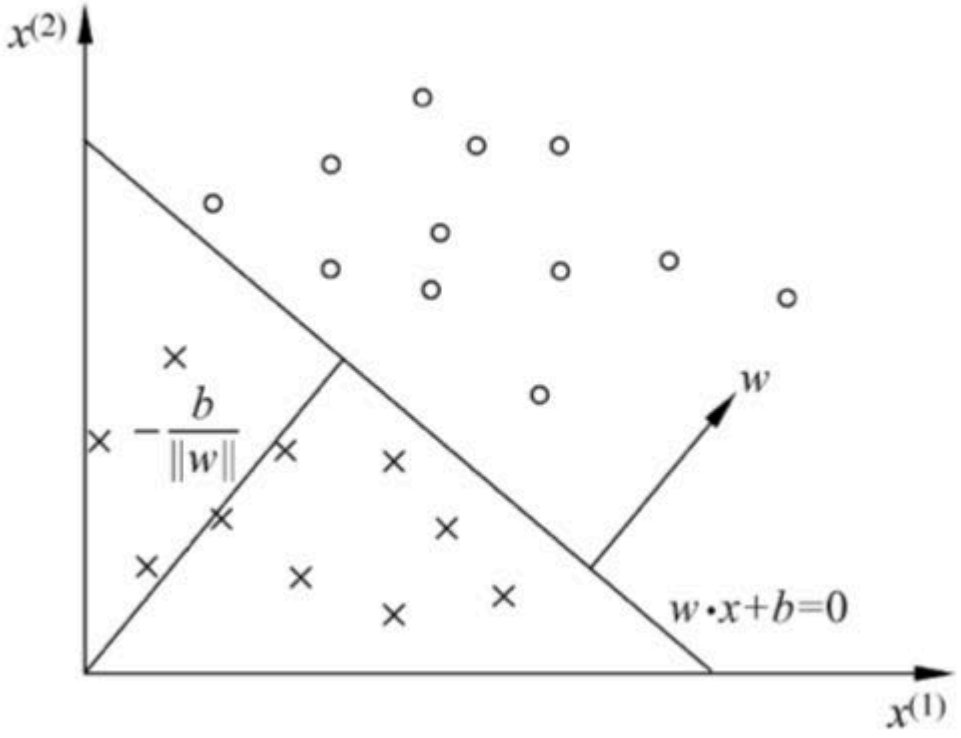


图 2.1　感知机模型

感知机模型的学习过程即为求得模型参数w和b的过程，其学习策略为定义（经验）损失函数并将损失函数极小化。

感知机所采用的损失函数是误分类点到超平面S的总距离。其损失函数定义为

$$L(w,b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

其中，M为误分类点的集合。感知机学习算法是误分类驱动的，具体采用随机梯度下降法。首先，任意选取一个超平面w_0,b_0，然后用梯度下降法不断极小化损失函数。这个过程中每次随机选取一个误分类点使其梯度下降。

$$w \leftarrow w + \eta y_i x_i$$

$$b \leftarrow b + \eta y_i$$

式中η是步长，它是一个超参数，由我们自己调整。这样通过迭代可以期待损失函数不断减小，在一定次数之后，我们选取损失函数最小的那次所对应的参数w，b作为最终感知机模型参数。

三、实验要求与步骤

要求1 使用给定的训练数据训练感知机模型

依照上述实验原理，我们选择误分类点到超平面S的总距离作为损失函数，并通过随机梯度下降法选择若干迭代次数之后它取最小时所对应的参数：

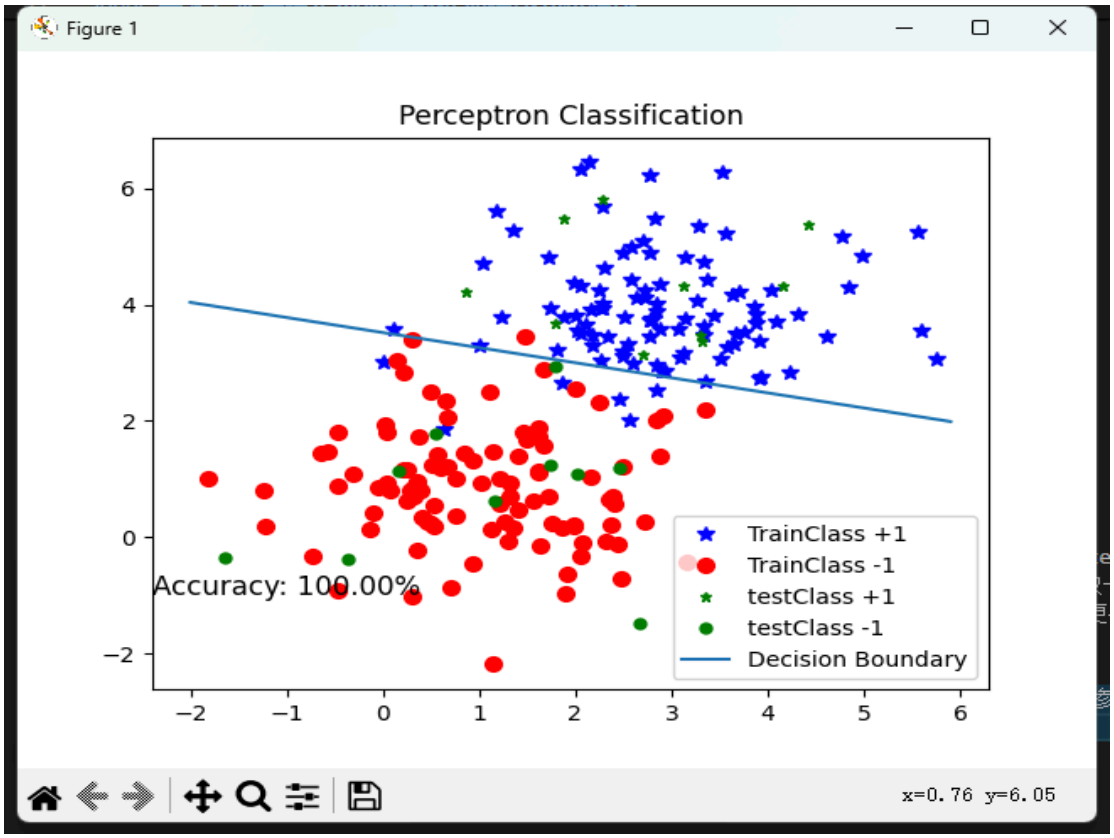
```
    def train(self):

        #具有最小总损失的参数
        best_w = self.w.copy()
        best_b = self.b
        min_total_loss = float('inf')

        count =0
        while self.iteration_num > 0:
            # 计算误分类点到超平面的距离
            distances = self.train_y * (np.dot(self.train_x, self.w.T) + self.b)
            misclassified_indices = np.where(distances <= 0)[0]          #所有误分类点的集合

            misclassified_distances = -distances[misclassified_indices] #误分类点到超平面的距离
            # 计算损失函数（距离之和）
            total_loss = np.sum(misclassified_distances)
            # 如果此时的损失函数最小，就记录下这次的w，b参数
            if total_loss < min_total_loss and total_loss > 0:
                best_b = self.b
                best_w = self.w.copy()
                min_total_loss = total_loss
```

```
            # 打乱顺序，随机取出一个误分类点进行参数更新
            np.random.shuffle(misclassified_indices)
            x = self.train_x[misclassified_indices[0]]
            y = self.train_y[misclassified_indices[0]]

            self.w += self.eta * y * x
            self.b += self.eta * y
            # 更新迭代次数
            self.iteration_num -= 1
            count +=1
            print("已迭代",count,"次")

        self.w = best_w.copy()
        self.b = best_b
        return self.w, self.b
```

在上述代码中我们设定模型中的超参数步长η即self.eta = 1，迭代次数self.iteration_num = 1000，在这种情况下，每次迭代一次我们都从误分类点的集合misclassified_indices中随机取一个点，并通过这个点进行梯度下降。同时每次迭代都将本次的损失函数与上一次的相比较，如果本次的损失函数更小，则将此时的参数b，w记录为最佳参数。若干次迭代之后选取最佳参数作为我们的训练结果。

要求2 在二维平面中画出感知机模型

我们调用python中matplotlib库的用法将训练集、测试集、根据上面训练得到的参数而得到的超平面S以及测试集准确率等信息画在二维平面中。



```
def draw(self):
    # 训练集点，正例蓝色，负例红色
    x1 = self.train_x[np.where(self.train_y > 0)][:, 0]
    y1 = self.train_x[np.where(self.train_y > 0)][:, 1]
    x2 = self.train_x[np.where(self.train_y < 0)][:, 0]
    y2 = self.train_x[np.where(self.train_y < 0)][:, 1]
    # 测试集点，正负例绿色
    x4 = self.test_x[np.where(self.test_y > 0)][:, 0]
    y4 = self.test_x[np.where(self.test_y > 0)][:, 1]
    x5 = self.test_x[np.where(self.test_y < 0)][:, 0]
    y5 = self.test_x[np.where(self.test_y < 0)][:, 1]
    # 超平面S
    x3 = np.arange(-2, 6, 0.1)
    y3 = (self.w[0] * x3 + self.b) / -self.w[1]

    plt.plot(x1, y1, 'b*', label='TrainClass +1',markersize = 7)
    plt.plot(x2, y2, 'ro', label='TrainClass -1',markersize = 7)
    plt.plot(x4, y4, 'g*', label='testClass +1', markersize=5)
    plt.plot(x5, y5, 'go', label='testClass -1', markersize=5)
    plt.plot(x3, y3, label='Decision Boundary')
    # 添加准确率文本
    acc_text = "Accuracy: {:.2f}%".format(self.accuracy() * 100)
    plt.text(-1, -1, acc_text, fontsize=12, ha='center')
    plt.title('Perceptron Classification')
    plt.legend()
    plt.show()
```

要求3 使用得到的感知机模型分类测试数据并报告分类准确率

首先我们生成测试数据，两类测试数据分别以(1,1)和(3,4)成正态随机分布。

```
# 测试数据集
n_test = 10  # 样本量大小
test_X = np.zeros((2 * n_test, 2))  # 训练数据坐标
test_Y = np.zeros(2 *n_test)  # 训练数据标签
center1 = [1,1]  #第一类数据中心
center2 = [3,4]  #第二类数据中心
for i in range(0, n_test):
    test_X[i] = np.random.normal(center1, 1, 2)
for i in range(n_test, 2 * n_test):
    test_X[i] = np.random.normal(center2, 1, 2)
```

```
    for i in range(0, n_test):
        test_Y[i] = -1
    for i in range(n_test, 2 * n_test):
        test_Y[i] = 1
```

然后我们设定预测函数：将测试点坐标代入超平面函数，根据结果返回不同的预测值。

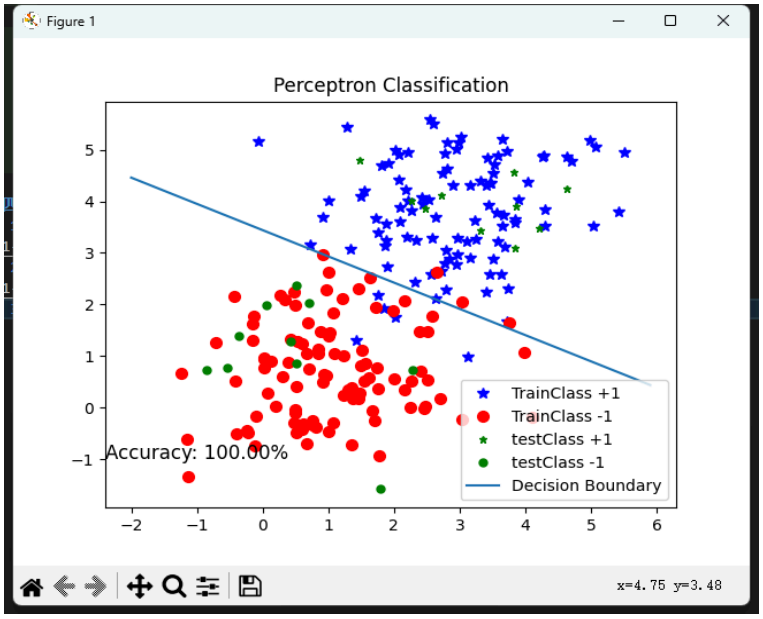
```
# 预测函数
    def predict(self,x):
        if np.dot(x, self.w.T) + self.b > 0:
            return 1
        else:
            return -1
```

最后我们计算准确率，将上述预测值与测试数据本身标签进行对比并计算得准确率。

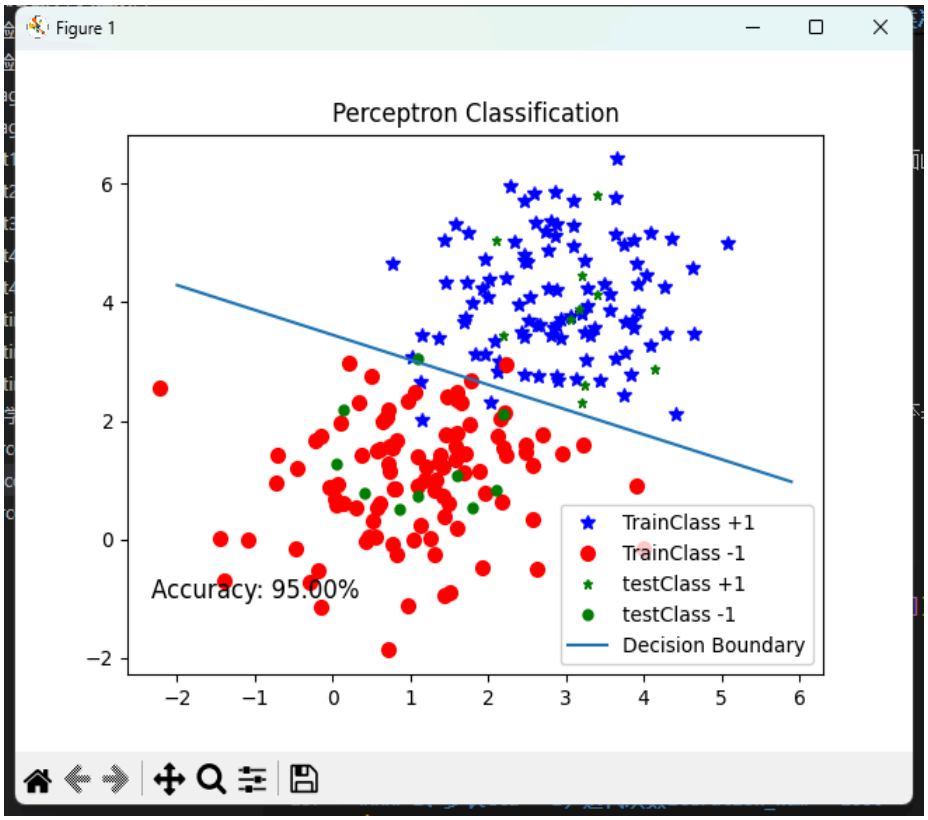
```
# 预测函数
        def accuracy(self):

        correct_count = 0
        for i in range(self.n_test *2):
            prediction = self.predict(self.test_x[i])
            if prediction == self.test_y[i]:
               correct_count += 1
        accuracy = correct_count / (self.n_test*2)
        return accuracy
```

四、实验结果展示

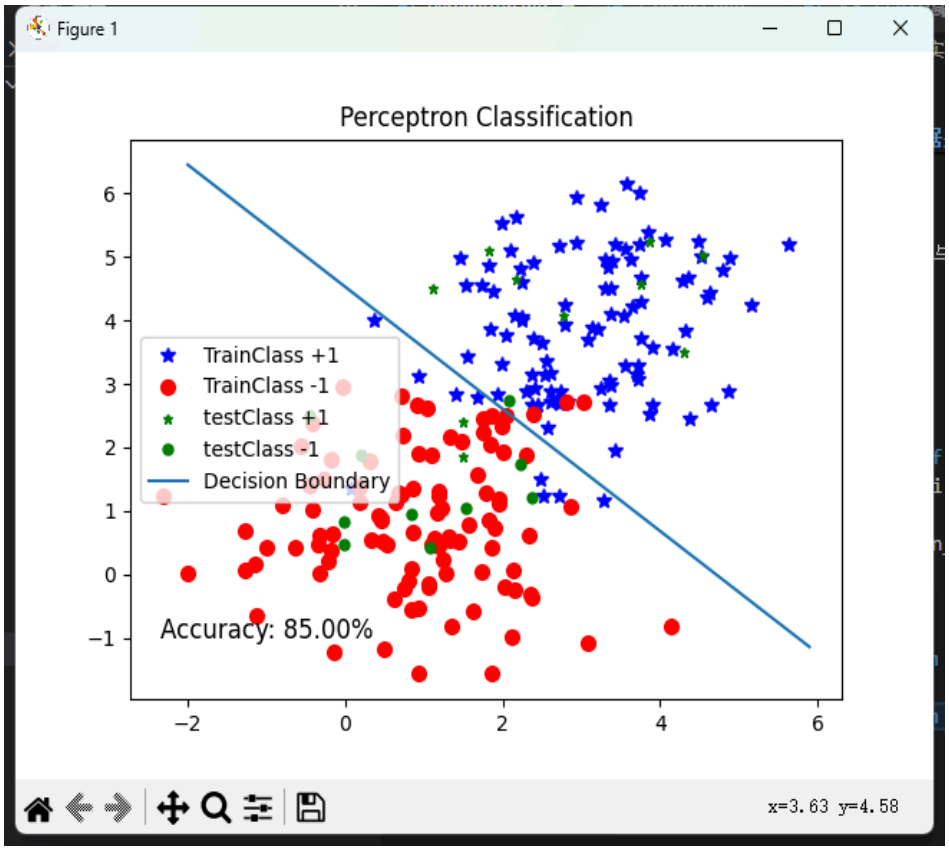
1、步长eta = 1，迭代次数iteration_num = 1000：



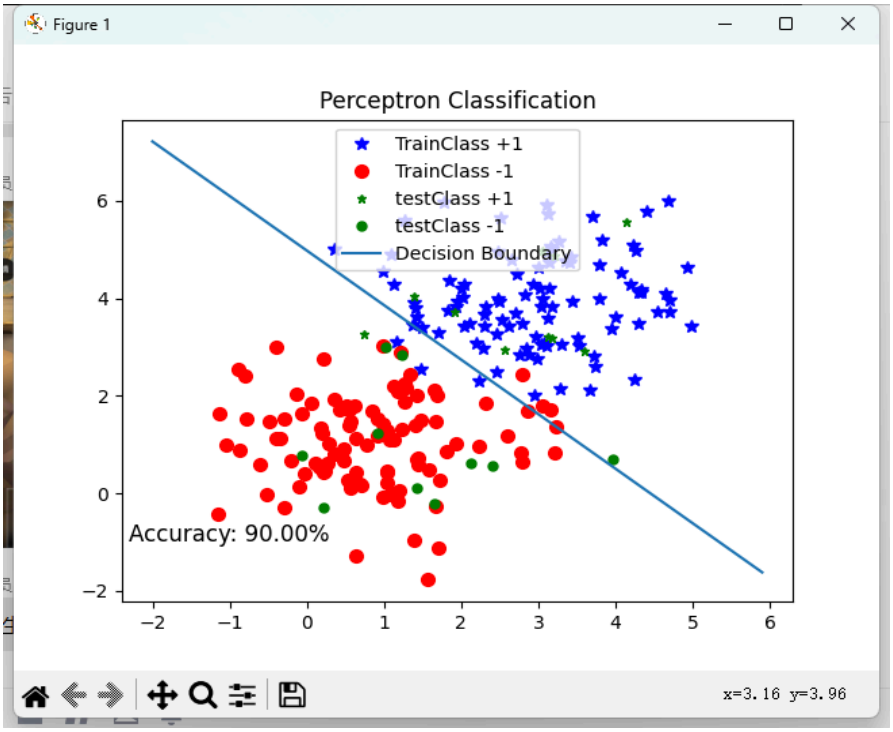
2、步长eta = 1，迭代次数iteration_num = 100：



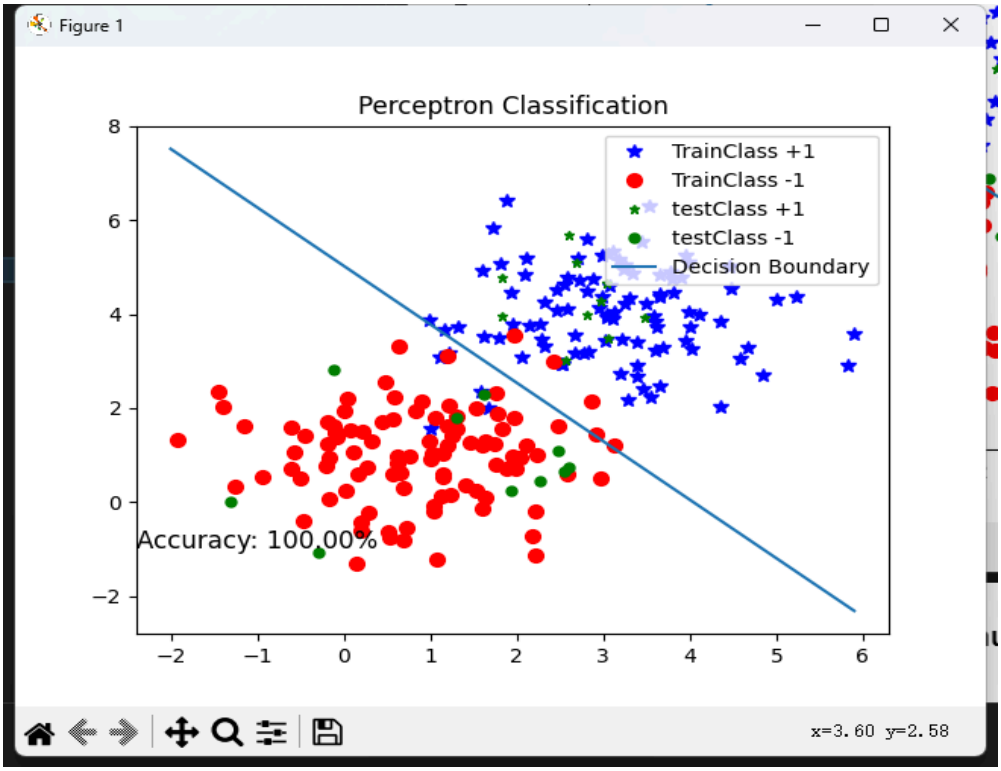
3、步长eta = 1，迭代次数iteration_num = 50：



4、步长eta = 0.5，迭代次数iteration_num = 50：



5、步长eta = 0.1，迭代次数iteration_num = 50：



五、附加题

附加题1：通过改变随机梯度下降法的步长(η)及迭代次数，观察优化算法快慢，直观感受优化算法对机器学习模型的影响

上述实验结果1，2，3分别展示了在相同步长的情况下迭代次数分别为1000、100、50的情况，其准确率分别为100%、95%、85%。由此可见在相同条件下，迭代次数越大，算法的准确度越高，但同时由于迭代次数的增加算法运行时间也会相应增长。

上述实验结果3，4，5分别展示了在相同迭代次数的情况下步长分别为1、0.5、0.1的情况，其准确率分别为85%、90%、100%。由此可见在相同条件下，步长越短，算法的准确度越高，但同时由于步长的减小导致需要足够多的迭代次数才能使得算法损失函数梯度下降到最低。

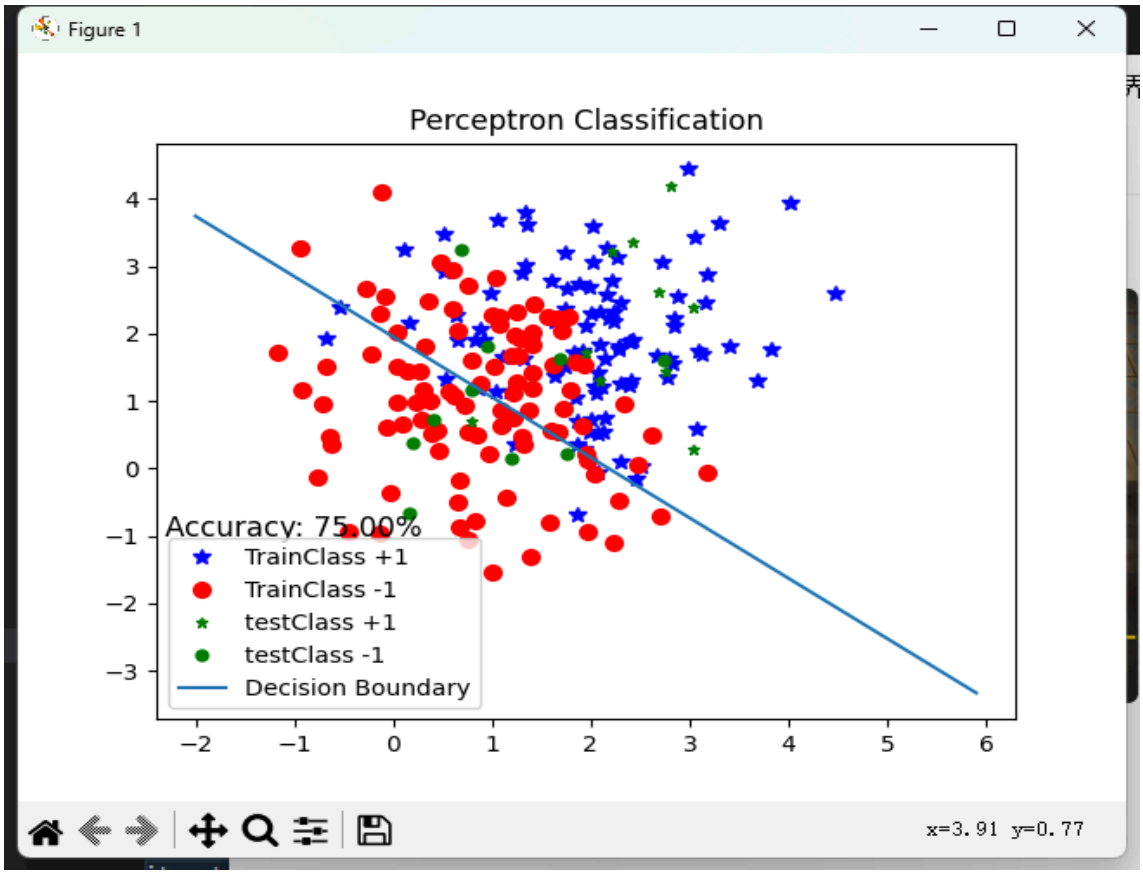
附加题3：课本中所给算法，如果去掉下面语句，结果如何?

$$if \quad y_i(w^T x_i + b) <= 0$$

这句代码的作用是只有误分类点才会对参数进行梯度下降，如果去掉这一句将使得所有点都会对所求参数产生影响，在这种情况下，感知机算法将无法收敛到一个合理结果。

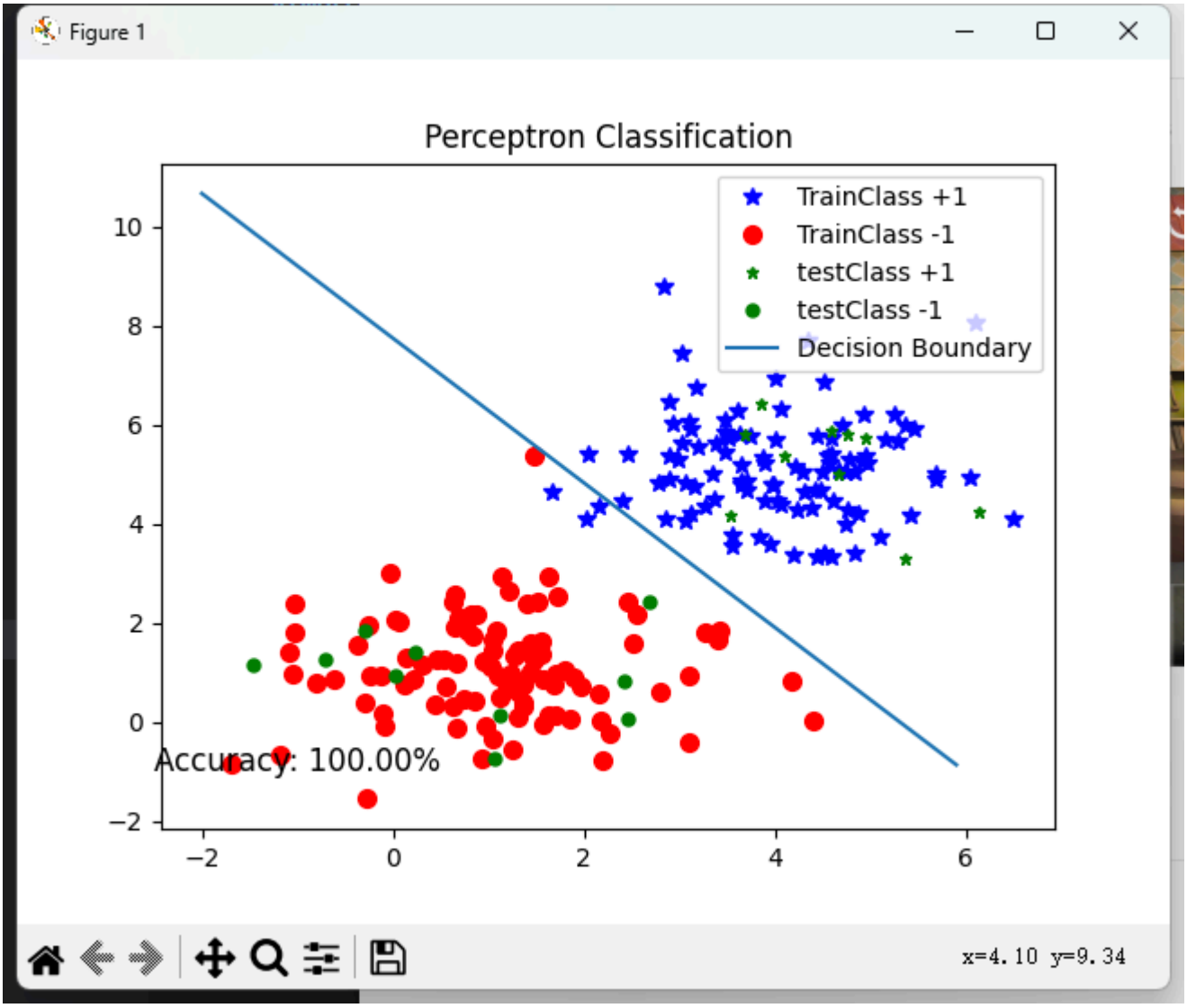
附加题4：通过改变数据中心，使两类数据更近或更远，观察分类效果

我将数据集的一个中心点由[3,4]改为了[2,2]这样使得两类数据离得更近，依旧在eta = 1，iteration_num = 1000的情况下结果如下：



与实验结果1中100%的准确率相比两类数据离得更近的话其准确率明显降低。这与我们的直觉相同：两类数据相隔越远则越容易分类。

我们将一个中心点坐标再次改为[4,5]后结果如下，保持高准确率，符合上述结论。



六、实验结果与总结

通过本次实验我详细掌握并熟悉了感知机算法及代码实现，并直观地学习到算法步长η和迭代次数对算法快慢和算法准确性的影响。

七、附录-实验代码(python)

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

class Perceptron:

    def __init__(self, train_x, train_y,test_x,test_y, eta=1):
        # 超参数：步长
        self.eta = eta
        # 初始化参数w，b
        self.w = np.zeros(train_x.shape[1])
        self.b = 0
        # 保存训练集
        self.train_x = train_x
        self.train_y = train_y
        # 保存测试集
        self.test_x = test_x
        self.test_y = test_y
        self.n_test = 10
        # 迭代次数
        self.iteration_num = 1000

    def train(self):

        #具有最小总损失的参数
        best_w = self.w.copy()
        best_b = self.b
        min_total_loss = float('inf')

        count =0
        while self.iteration_num > 0:
            # 计算误分类点到超平面的距离
            distances = self.train_y * (np.dot(self.train_x, self.w.T) + self.b)
            misclassified_indices = np.where(distances <= 0)[0]          #所有误分类点的集合

            misclassified_distances = -distances[misclassified_indices] #误分类点到超平面的距离
            # 计算损失函数（距离之和）
            total_loss = np.sum(misclassified_distances)
            # 如果此时的损失函数最小，就记录下这次的w，b参数
            if total_loss < min_total_loss and total_loss > 0:
                best_b = self.b
                best_w = self.w.copy()
```

```
            min_total_loss = total_loss

        # 打乱顺序，随机取出一个误分类点进行参数更新
        np.random.shuffle(misclassified_indices)
        x = self.train_x[misclassified_indices[0]]
        y = self.train_y[misclassified_indices[0]]

        self.w += self.eta * y * x
        self.b += self.eta * y
        # 更新迭代次数
        self.iteration_num -= 1
        count +=1
        print("已迭代",count,"次")

    self.w = best_w.copy()
    self.b = best_b
    print("Accuracy: {:.2f}%".format(self.accuracy() * 100))
    return self.w, self.b


# 预测函数
def predict(self,x):

    if np.dot(x, self.w.T) + self.b > 0:
        return 1
    else:
        return -1

def accuracy(self):

    correct_count = 0
    for i in range(self.n_test *2):
        prediction = self.predict(self.test_x[i])
        if prediction == self.test_y[i]:
           correct_count += 1
    accuracy = correct_count / (self.n_test*2)
    return accuracy

# 绘制分类结果
def draw(self):
    # 训练集点，正例蓝色，负例红色
    x1 = self.train_x[np.where(self.train_y > 0)][:, 0]
    y1 = self.train_x[np.where(self.train_y > 0)][:, 1]
```

```
        x2 = self.train_x[np.where(self.train_y < 0)][:, 0]
        y2 = self.train_x[np.where(self.train_y < 0)][:, 1]
        # 测试集点，正负例绿色
        x4 = self.test_x[np.where(self.test_y > 0)][:, 0]
        y4 = self.test_x[np.where(self.test_y > 0)][:, 1]
        x5 = self.test_x[np.where(self.test_y < 0)][:, 0]
        y5 = self.test_x[np.where(self.test_y < 0)][:, 1]
        # 超平面S
        x3 = np.arange(-2, 6, 0.1)
        y3 = (self.w[0] * x3 + self.b) / -self.w[1]

        plt.plot(x1, y1, 'b*', label='TrainClass +1',markersize = 7)
        plt.plot(x2, y2, 'ro', label='TrainClass -1',markersize = 7)
        plt.plot(x4, y4, 'g*', label='testClass +1', markersize=5)
        plt.plot(x5, y5, 'go', label='testClass -1', markersize=5)
        plt.plot(x3, y3, label='Decision Boundary')
        # 添加准确率文本
        acc_text = "Accuracy: {:.2f}%".format(self.accuracy() * 100)
        plt.text(-1, -1, acc_text, fontsize=12, ha='center')
        plt.title('Perceptron Classification')
        plt.legend()
        plt.show()

def main():
    # 训练数据集
    n = 100          #样本量大小
    center1 = [1,1]  #第一类数据中心
    center2 = [3,4]  #第二类数据中心
    train_X = np.zeros((2*n,2))  #训练数据坐标
    train_Y = np.zeros(2*n)  #训练数据标签

    for i in range(0,n):
        train_X[i] = np.random.normal(center1,1,2)
    for i in range(n,2*n):
        train_X[i] = np.random.normal(center2,1,2)

    for i in range(0,n):
        train_Y[i] = -1
    for i in range(n, 2*n):
        train_Y[i] = 1

    # 测试数据集
    n_test = 10  # 样本量大小
```

```
    test_X = np.zeros((2 * n_test, 2))  # 训练数据坐标
    test_Y = np.zeros(2 *n_test)  # 训练数据标签

    for i in range(0, n_test):
        test_X[i] = np.random.normal(center1, 1, 2)
    for i in range(n_test, 2 * n_test):
        test_X[i] = np.random.normal(center2, 1, 2)

    for i in range(0, n_test):
        test_Y[i] = -1
    for i in range(n_test, 2 * n_test):
        test_Y[i] = 1


    perceptron = Perceptron(train_X, train_Y,test_X,test_Y)
    perceptron.train()
    perceptron.draw()



if __name__ == "__main__":
    main()
```