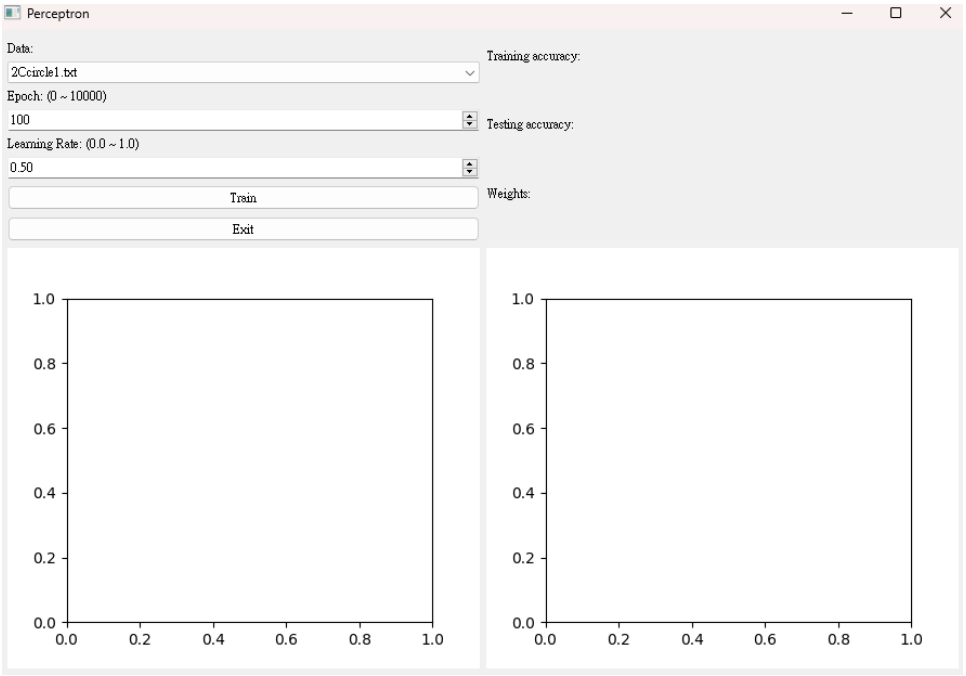
Neural Network Homework 1 Report

113522041 蔡秉睿

A. GUI 功能說明



GUI 的左上部分分別是資料、Epoch、Learning Rate 選擇，以及開始訓練和退出的按鈕。

右上部分為訓練完的準確率和權重以及跑過測試資料後的準確率。

下方為訓練資料以及測試資料的辨識結果。

B. 程式碼簡介

```
10  def sign(z):    #activation function
11      if z >= 0:
12          return 1
13      else:
14          return -1
15
```

左圖是激勵函數

```python
class Perceptron():     #perceptron
    def __init__(self):
        self.filename = ''
        self.epoch = 100
        self.lr = 0.5
        self.data = []
        self.x_num = []
        self.fin_weights = []
        self.train_data = []
        self.test_data = []
        self.train_accuracy = 0.0
        self.test_accuracy = 0.0
```

接著是感知機的基本功能，首先是初始化(上圖)

```python
    def readfile(self):     #readfile function
        # self.filename = input()
        file = open('basic/' + self.filename, "r")

        for line in file.readlines():
            line = line.rstrip('\n')
            line = '-1 ' + line
            line = line.split(' ')
            line = np.float_(line)
            # print(type(line))
            # print(line)
            self.data.append(line)

        file.close()
        # print(data)
```

再來是感知機的讀檔(上圖)

```python
def preprocess(self):   #preprocess function
    self.x_num = len(self.data[0]) - 1
    class1 = self.data[0][-1]
    for i in self.data:     #change the class of the input data
        if i[-1] == class1:
            i[-1] = -1
        else:
            i[-1] = 1

    train_data, test_data = train_test_split(self.data, test_size=0.33)     #train data and test data 2:1
    self.train_data = np.array(train_data)
    self.test_data = np.array(test_data)
    # print(type(train_data))
```

感知機的預處理(上圖)，其中包含幫資料分類以及把資料分成訓練集和測試集。

```
def train(self):    #training function
    w = np.random.uniform(0, 1, self.x_num)

    for epoch in range(self.epoch):
        for i in self.train_data:
            y = np.dot(w, i[:self.x_num])   #y = w * x
            d = sign(y)     #call activation function
            if y >= 0 and i[-1] != d:       #adjust the weights
                w = w - self.lr * i[:-1]
            elif y < 0 and i[-1]!= d:
                w = w + self.lr * i[:-1]
        print("epoch: ", epoch, "  weights: ", w)

    self.fin_weights.append(w)
    # print(self.fin_weights)
```

感知機訓練(上圖)

```
def predict(self):      #predict the accuracy of the train data and test data
    train_predictions = []
    train_classes = []
    weights = np.array(self.fin_weights)
    for i in self.train_data:
        y = np.dot(weights, i[:self.x_num])
        d = sign(y)
        train_predictions.append(d)
        train_classes.append(i[-1])

    test_predictions = []
    test_classes = []
    for i in self.test_data:
        y = np.dot(weights, i[:self.x_num])
        d = sign(y)
        test_predictions.append(d)
        test_classes.append(i[-1])

    self.train_accuracy = accuracy_score(train_classes, train_predictions)
    self.test_accuracy = accuracy_score(test_classes, test_predictions)
    print("Train Accuracy: ", self.train_accuracy)
    print("Test Accuracy: ", self.test_accuracy)
```

預測訓練以及測試結果(上圖)

```python
class MyWidget(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.setObjectName("MyWidget")
        self.setWindowTitle("Perceptron")
        self.resize(900, 600)
        self.ui()
```

接著是 GUI 設計部分(上圖)

```python
def ui(self):
    #layout
    main_layout = QtWidgets.QVBoxLayout(self)
    left_layout = QtWidgets.QVBoxLayout()
    right_layout = QtWidgets.QVBoxLayout()
    top_layout = QtWidgets.QHBoxLayout()
    bottom_layout = QtWidgets.QHBoxLayout()

    #data file
    data_label = QtWidgets.QLabel("Data:")
    left_layout.addWidget(data_label)
    self.file_choosing = QtWidgets.QComboBox(self)
    data_folder = "basic"
    for f in os.listdir(data_folder):
        self.file_choosing.addItem(f)
    left_layout.addWidget(self.file_choosing)

    #epoch
    epoch_label = QtWidgets.QLabel("Epoch: (0 ~ 10000)")
    left_layout.addWidget(epoch_label)
    self.epoch_input = QtWidgets.QSpinBox(self)
    self.epoch_input.setMinimum(0)
    self.epoch_input.setMaximum(10000)
    self.epoch_input.setValue(100)
    left_layout.addWidget(self.epoch_input)

    #learning rate
    lr_label = QtWidgets.QLabel("Learning Rate: (0.0 ~ 1.0)")
    left_layout.addWidget(lr_label)
    self.lr_input = QtWidgets.QDoubleSpinBox(self)
    self.lr_input.setMinimum(0.0)
    self.lr_input.setMaximum(1.0)
    self.lr_input.setValue(0.5)
    left_layout.addWidget(self.lr_input)

    #train button
    self.train_button = QtWidgets.QPushButton(self)
    self.train_button.setText("Train")
    self.train_button.clicked.connect(self.train_perceptron)
    left_layout.addWidget(self.train_button)
```

```python
    #accuracy label & weights label
    self.train_accuracy_label = QtWidgets.QLabel(self)
    self.train_accuracy_label.setText("Training accuracy:")
    self.train_accuracy_result = QtWidgets.QLabel(self)
    self.test_accuracy_label = QtWidgets.QLabel(self)
    self.test_accuracy_label.setText("Testing accuracy:")
    self.test_accuracy_result = QtWidgets.QLabel(self)
    self.weight_label = QtWidgets.QLabel(self)
    self.weight_label.setText("Weights:")
    self.weights_result = QtWidgets.QLabel(self)
    right_layout.addWidget(self.train_accuracy_label)
    right_layout.addWidget(self.train_accuracy_result)
    right_layout.addWidget(self.test_accuracy_label)
    right_layout.addWidget(self.test_accuracy_result)
    right_layout.addWidget(self.weight_label)
    right_layout.addWidget(self.weights_result)

    #exit button
    self.exit_button = QtWidgets.QPushButton(self)
    self.exit_button.setText("Exit")
    self.exit_button.clicked.connect(self.close)
    left_layout.addWidget(self.exit_button)

    #add layouts on the top_layout
    top_layout.addLayout(left_layout)
    top_layout.addLayout(right_layout)

    #plotting canvas
    self.train_canvas = PlotCanvas(self)
    bottom_layout.addWidget(self.train_canvas)
    self.test_canvas = PlotCanvas(self)
    bottom_layout.addWidget(self.test_canvas)

    #add top_layout & bottom_layout on the main_layout
    main_layout.addLayout(top_layout)
    main_layout.addLayout(bottom_layout)
```

上面兩張圖為 UI 的基本設計

```python
def resultRender(self, perceptron):     #show the result
    self.train_canvas.plot(perceptron.train_data, perceptron.fin_weights, "Train data")
    self.test_canvas.plot(perceptron.test_data, perceptron.fin_weights, "Test data")
    self.train_accuracy_result.setText(str(perceptron.train_accuracy))
    self.test_accuracy_result.setText(str(perceptron.test_accuracy))
    weights_string = ""
    for i in perceptron.fin_weights:
        weights_string += str(i) + " "
    self.weights_result.setText(weights_string)

def train_perceptron(self):     #start training
    print("Start training")
    perceptron = Perceptron()
    perceptron.filename = str(self.file_choosing.currentText())
    perceptron.epoch = self.epoch_input.value()
    perceptron.lr = self.lr_input.value()
    perceptron.readfile()
    perceptron.preprocess()
    perceptron.train()
    perceptron.predict()
    self.resultRender(perceptron)
```

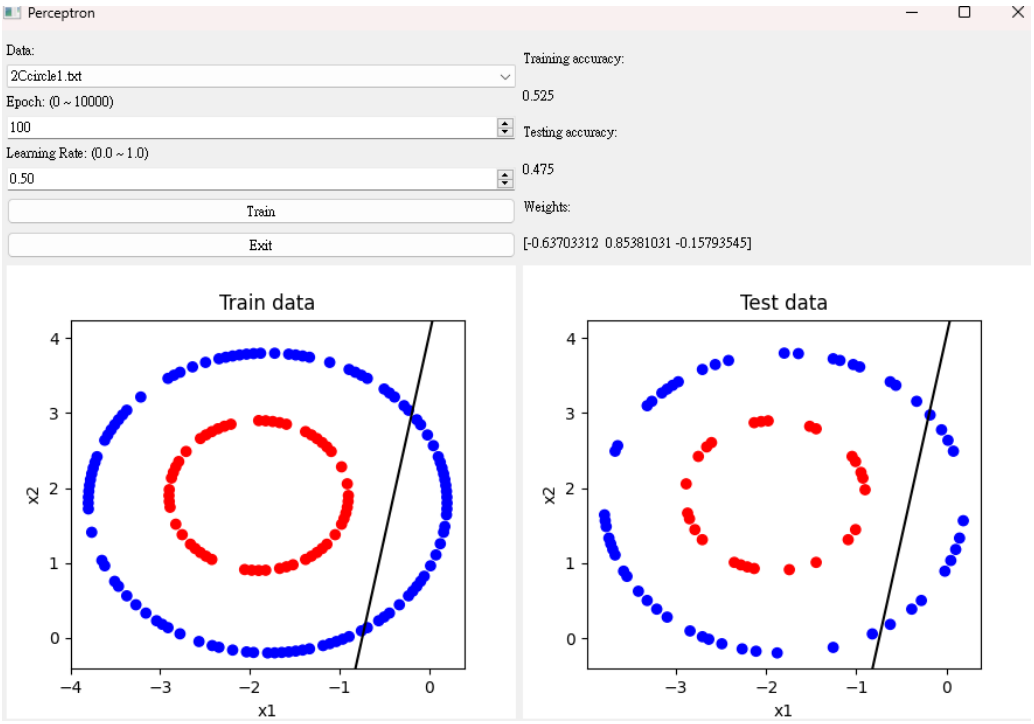
train_perceptron 是呼叫上面感知機的 function 做運算

resultRender 是把結果回傳出來(顯示為二維圖)

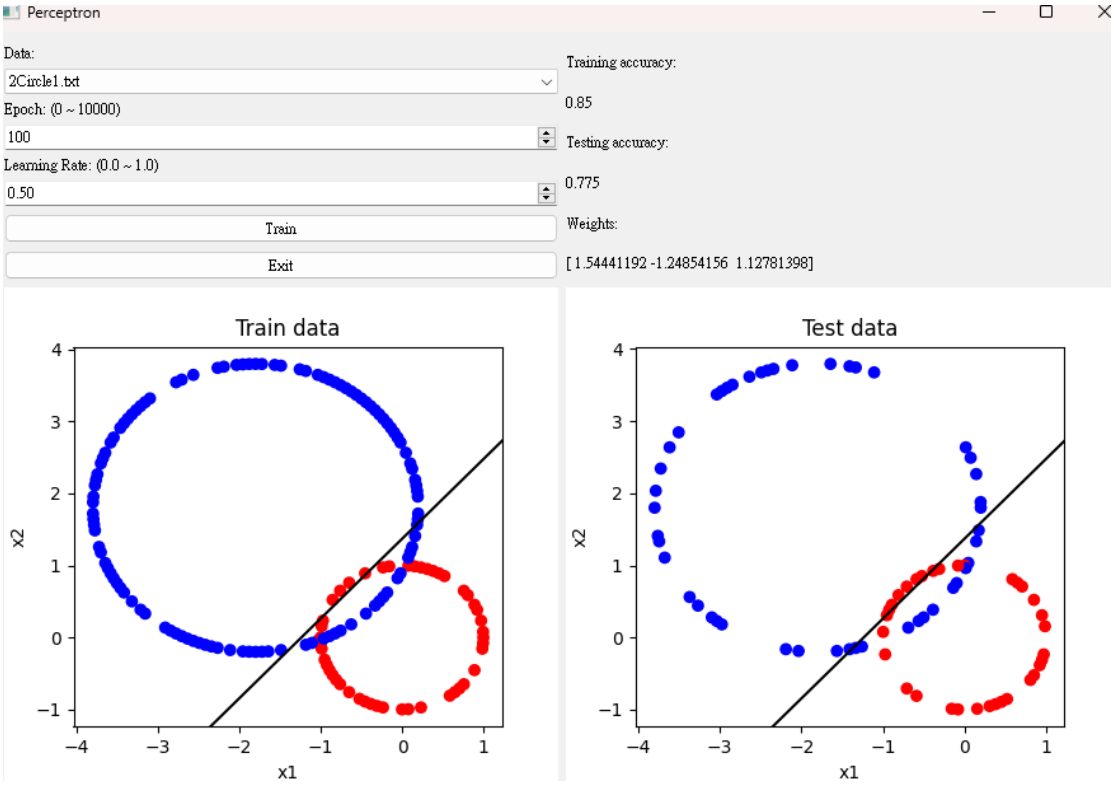
```python
class PlotCanvas(FigureCanvas):     #plot train and test results
    def __init__(self, parent=None, width=12, height=4, dpi=100):
        fig, self.axes= plt.subplots(figsize=(width, height), dpi=dpi)
        FigureCanvas.__init__(self, fig)
        # self.setParent(parent)

    def plot(self, data, weight, title):
        self.axes.clear()
        self.axes.set_title(title)
        colors = []
        for i in data:      #classified by colorize
            if i[-1] == -1:
                colors.append('r')
            else:
                colors.append('b')
        self.axes.scatter(data[:, 1], data[:, 2], c=colors)
        self.axes.axline((0, weight[0][0] / weight[0][2]), (weight[0][0] / weight[0][1], 0), color="black")     # w0 = w1 * x1 + w2 * x2
        self.axes.set_xlabel('x1')
        self.axes.set_ylabel('x2')
        self.draw()
```

上圖為畫出二維圖的 function

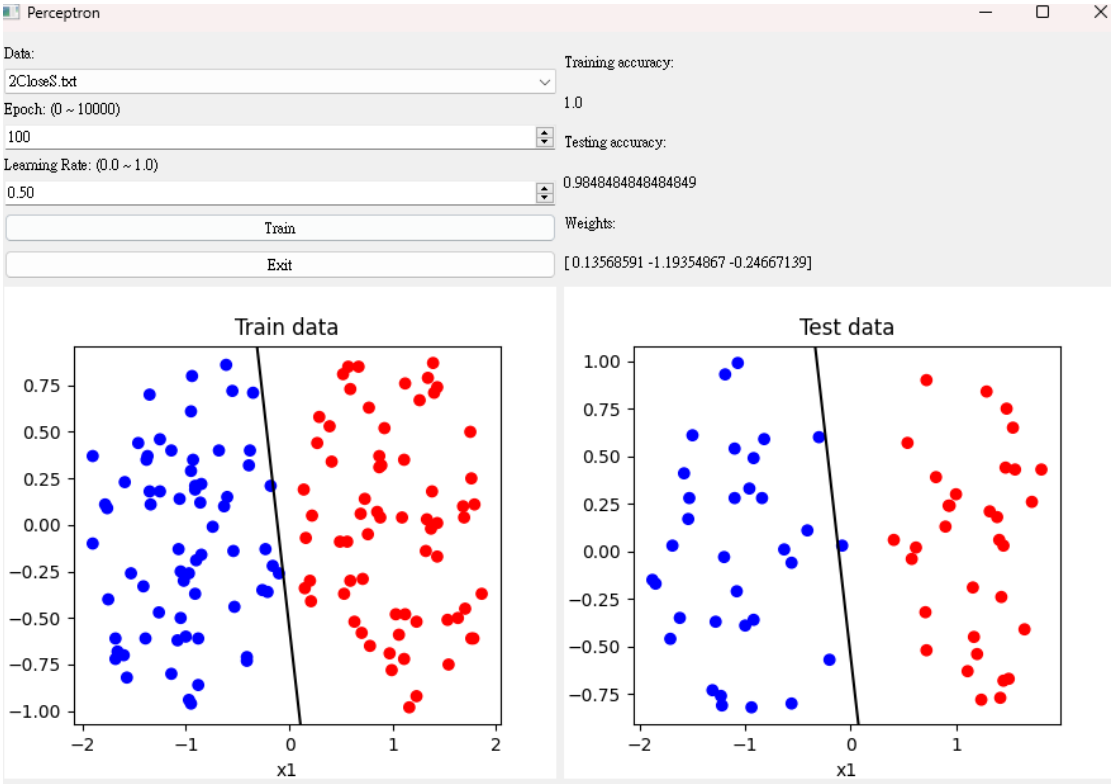
C. 實驗結果



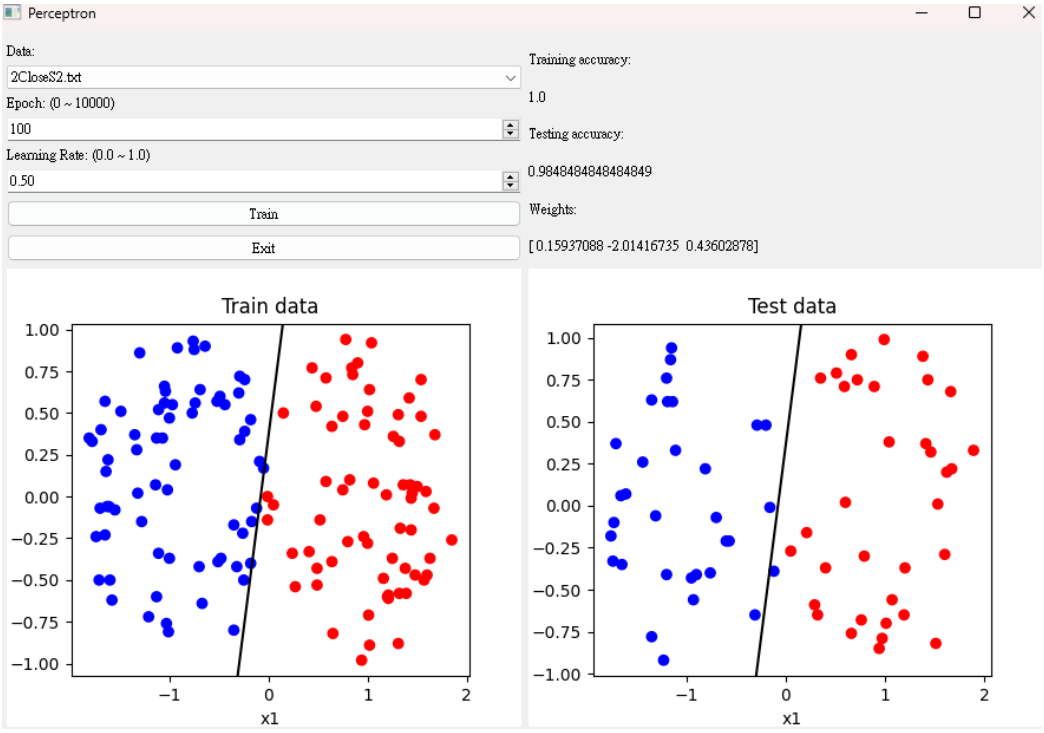
2Ccircle1.txt：資料為非線性分割，結果都不好。



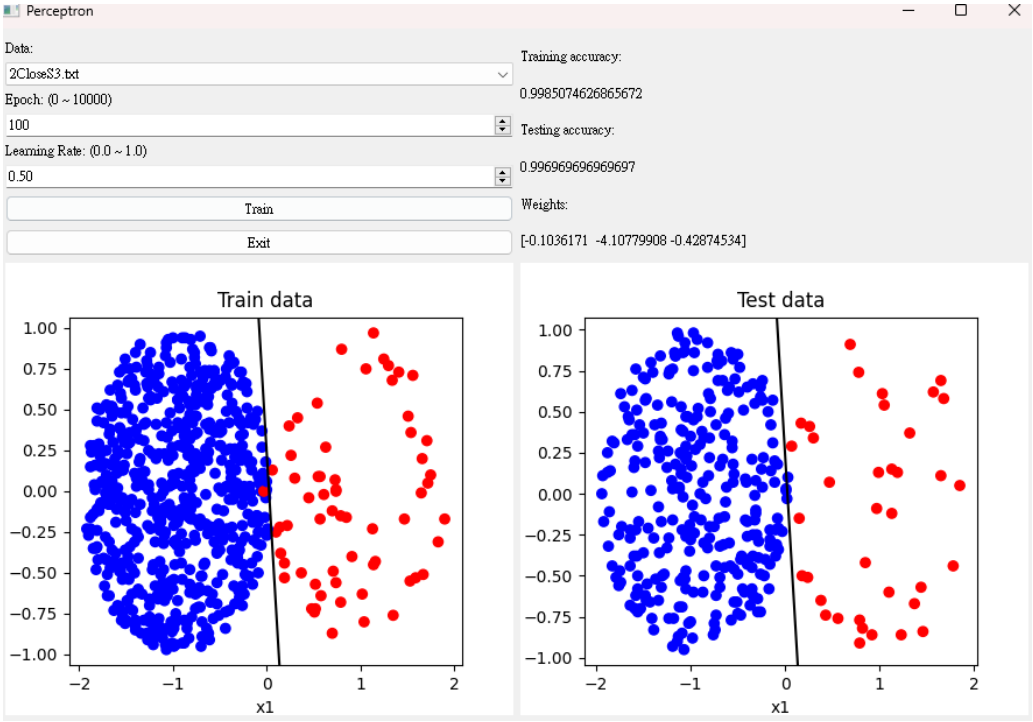
2Circle1.txt：資料沒辦法完全分割，但準確率有提高。



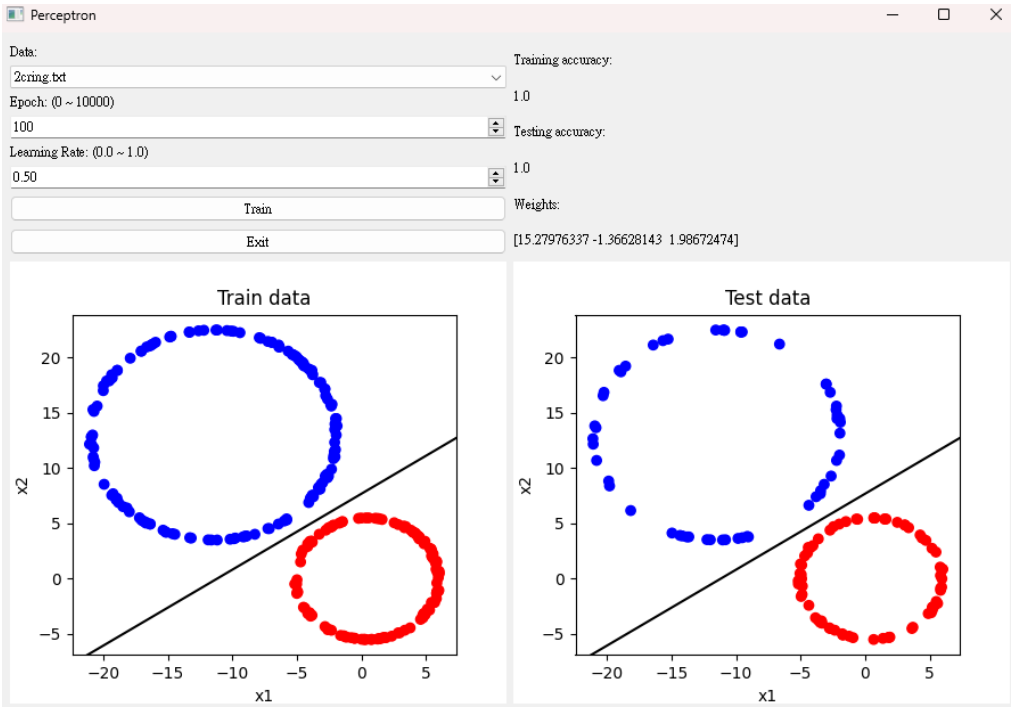
2CloseS.txt：資料幾乎可完全線性分割。



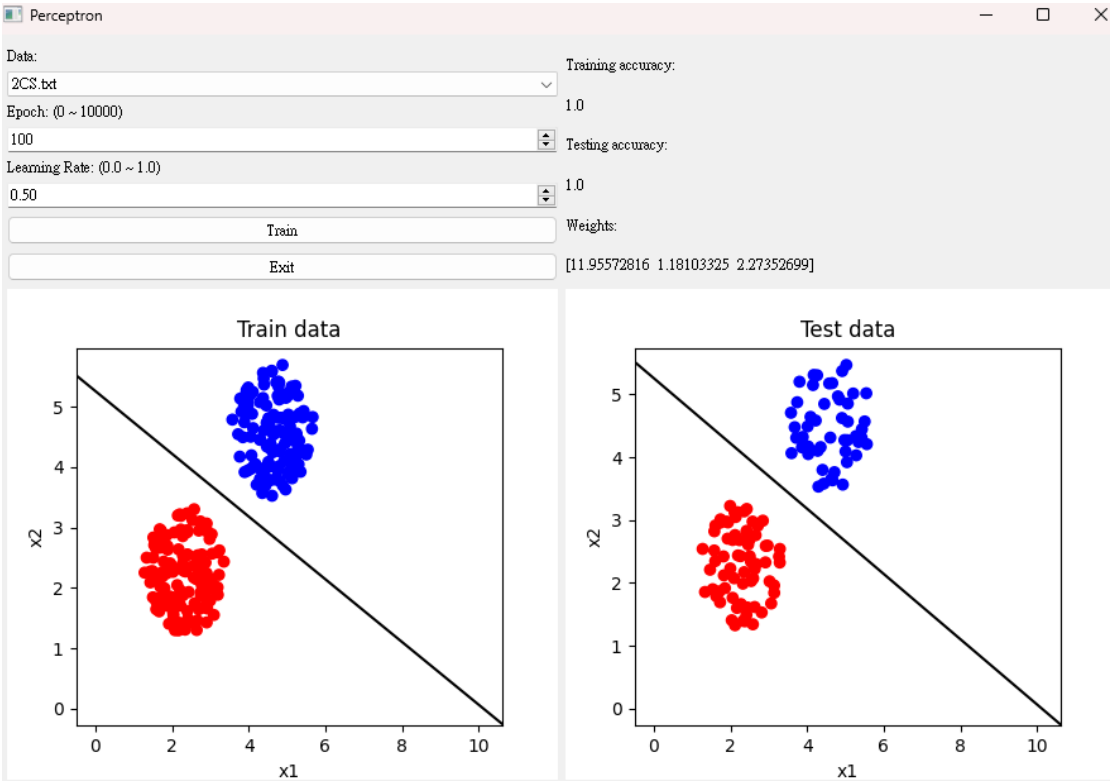
2CloseS2.txt：和上一個一樣，資料幾乎可完全分割。



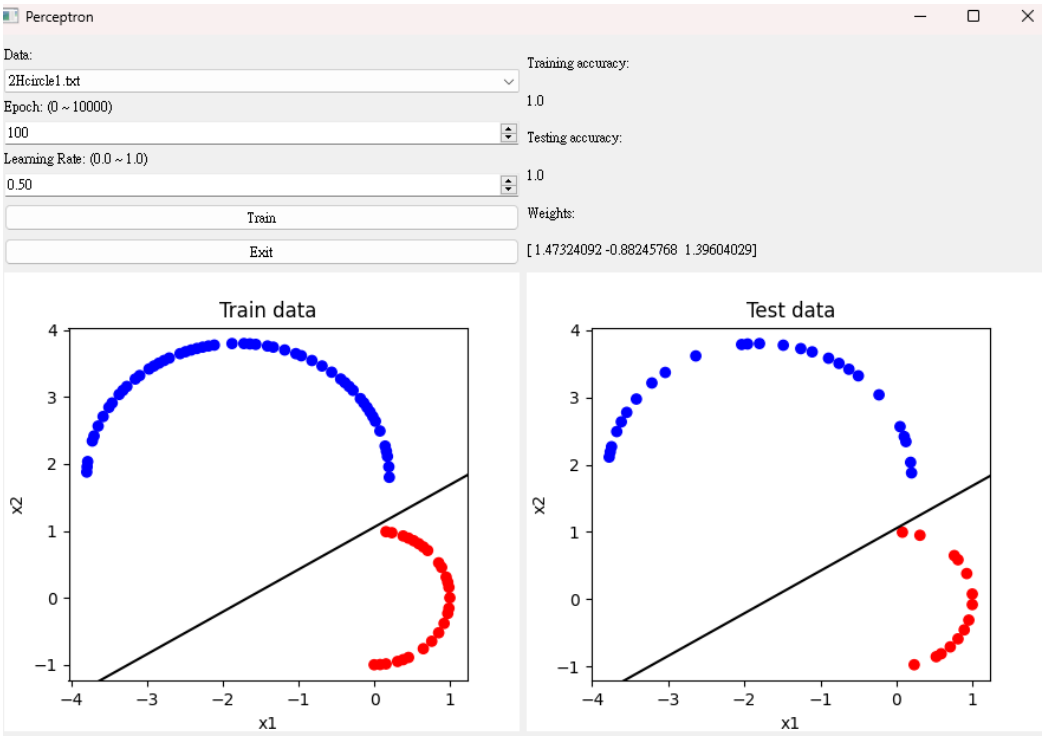
2CloseS3.txt：資料幾乎可完全線性分割，2CloseS 系列幾乎準確率都達到 100%。



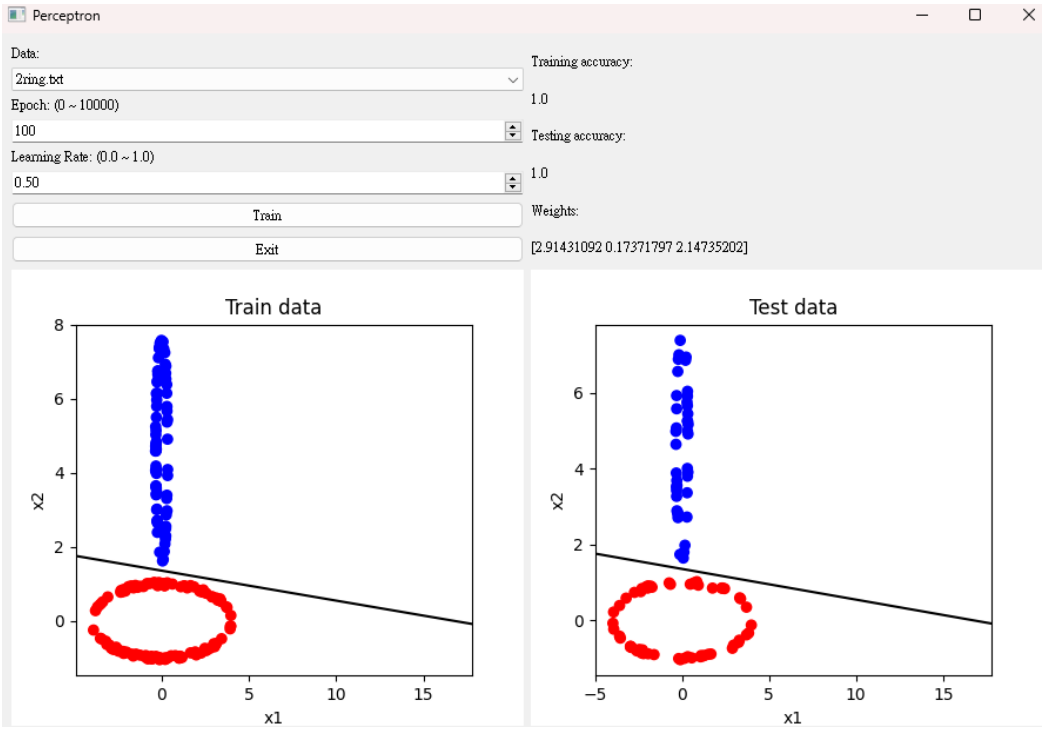
2cring.txt：資料可完全線性分割，準確率為 100%。



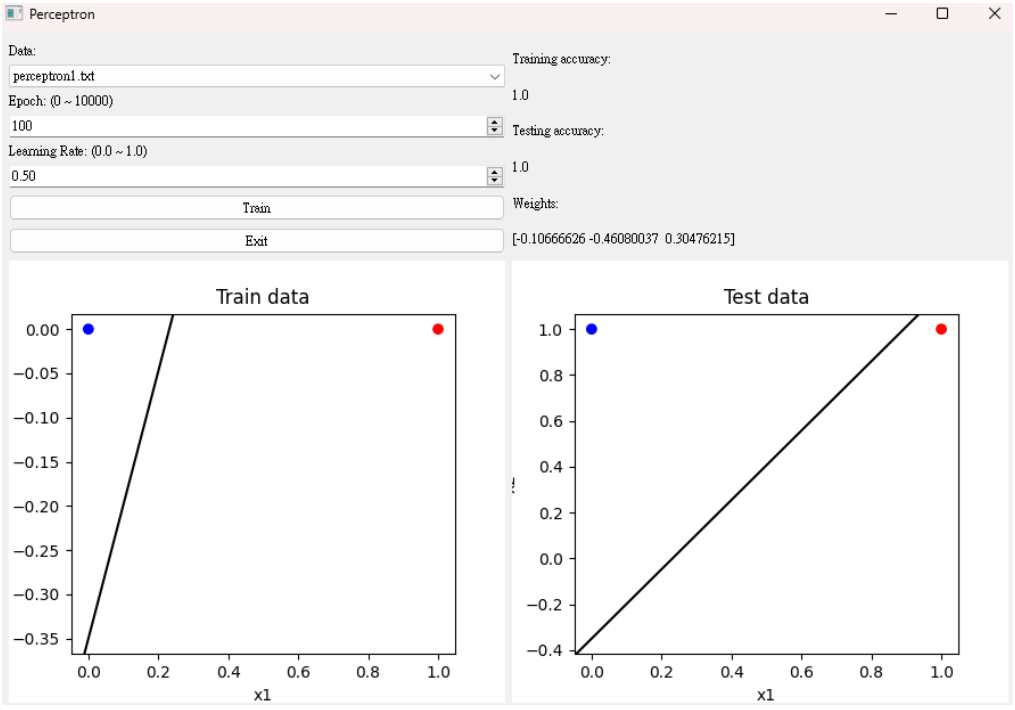
2CS.txt：資料可完全線性分割，準確率為 100%。



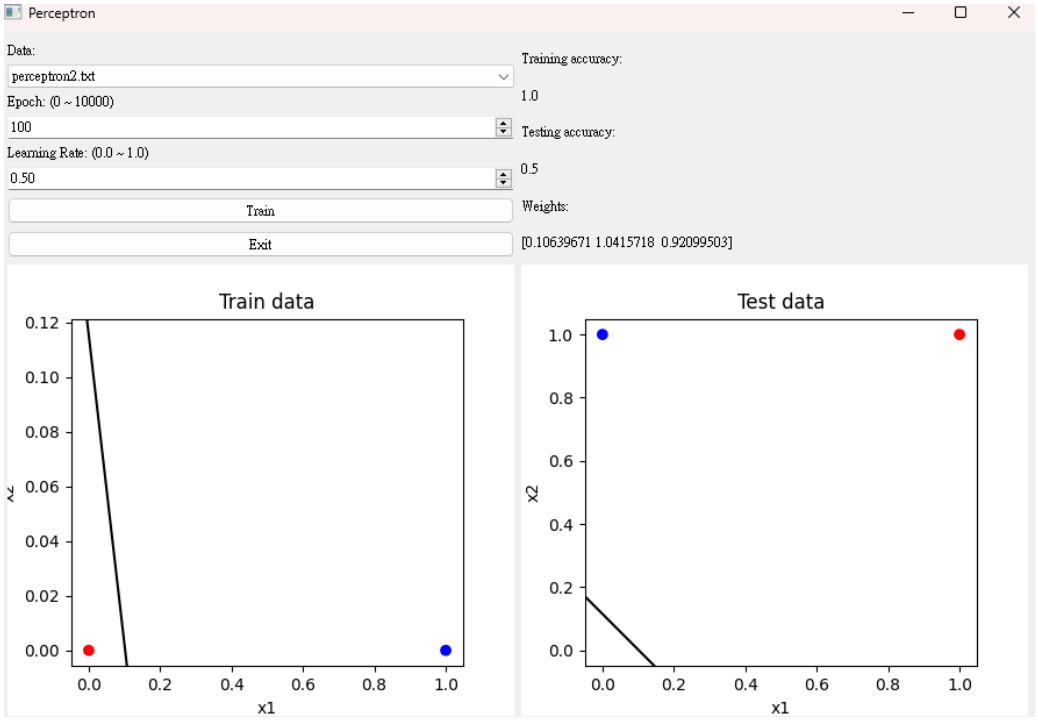
2Hcircle1.txt：資料可完全線性分割，準確率為 100%。



2ring.txt：資料可完全線性分割，準確率為 100%。



perceptron1.txt：資料量太少。有時準確率可以 100%，有時可以 50%，有時卻是 0%。



perceptron2.txt：資料為 XOR 加上一個 NOT GATE，為不可線性分割的資料。

D. 實驗結果分析及討論

在資料集中，如果兩類資料是分很開且可線性分割的話，比較容易做感知機的分類；相對的，無法做線性分割的資料不管增加幾次訓練，結果一樣部會改善。

另外原本認為降低 Epoch 次數(10)、提高 Learning rate(0.8)會影響準確率，不過似乎是給我們的資料的分類很清楚明白，結果依然很好，同時我也發現原本設為 Epoch 100 時，後半段的 weights 數據都是一樣的，代表很早就分類完成了。

然後程式部分分為感知機實作以及 GUI 的製作兩個部分。

感知機的部分我覺得比較麻煩的是 readfile，原本是用 read().split('\n ')來做，後來發現給我們的資料中有幾個在最後一筆都多一行，所以後來換成 readlines()並且處理多的換行字元。

GUI 的部分是我認為最麻煩的部分，好多東西都是第一次碰到，不過也認識滿多新工具。

最後是在打包成執行檔後，不知道為什麼執行檔有 2GB 左右，導致開起來都要比較久，都要大概 2、30 秒。