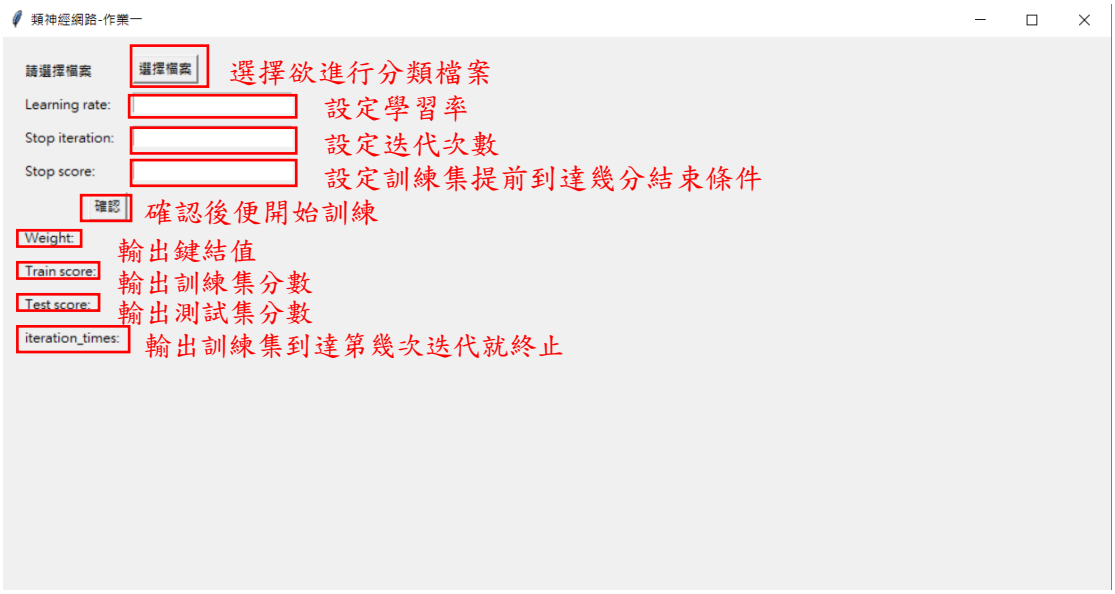
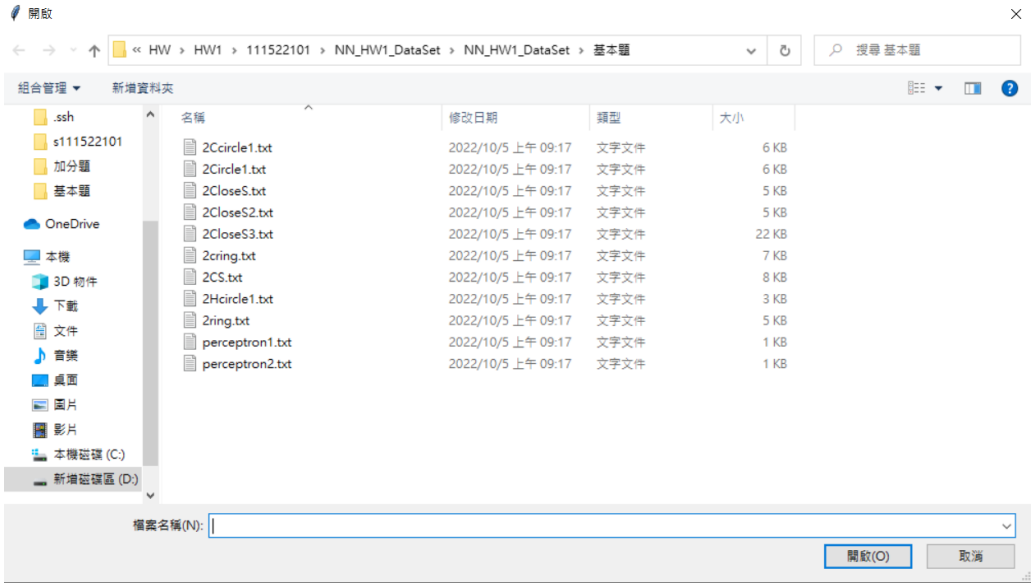
- 程式執行說明

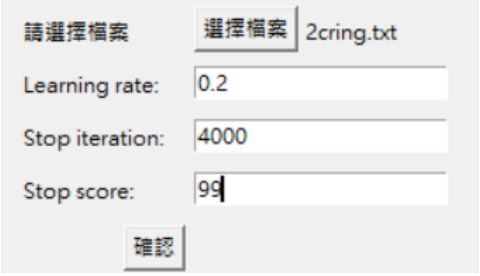
1. 執行/dist 的 main.exe
2. 開啟程式 GUI



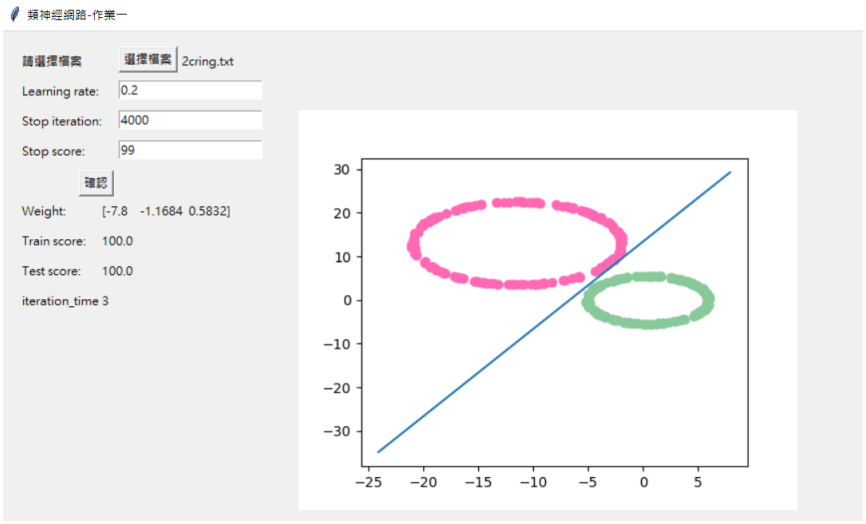
3. 選擇欲分類檔案



4. 依序輸入學習率，迭代次數，訓練集提前結束條件



5. 得到輸出結果



- 程式碼簡介

此程式碼主要含有一個 main 與三個 class，分別為

- main.py
- Preprocessor.py
- Model.py
- Plot.py

以下會依序介紹這四個檔案

1. main.py

   main.py 主要是負責介面的設計與 class 的調動，GUI 主要是使用 tkinter 進行撰寫

```python
def main():
    window = tk.Tk()

    window.geometry("1000x500+200+300")

    window.title('類神經網路-作業一')
    #選擇檔案
    file_name = tk.StringVar()   # 設定 text 為文字變數
    file_name.set('')            # 設定 text 的內容

    tk.Label(window, text='請選擇檔案').place(x = 20,y = 20)

    tk.Button(window, text='選擇檔案',command= lambda: get_file_url(file_name)).place(x = 120,y = 16)

    tk.Label(window, textvariable=file_name).place(x=180, y=20)

    #學習率

    tk.Label(window, text='Learning rate:').place(x = 20,y = 50)
    learning_rate = tk.Entry(window)
    learning_rate.place(x = 120,y = 50)

    #輸入迭代次數

    tk.Label(window, text='Stop iteration:').place(x = 20,y = 80)
    interation = tk.Entry(window)
    interation.place(x = 120,y = 80)


    #Stop rate:
    tk.Label(window, text='Stop score:').place(x = 20,y = 110)
    stop_rate = tk.Entry(window)
    stop_rate.place(x = 120,y = 110)
```

使用者選擇檔案

GUI 使用者輸入部分

```python
    #Weight
    tk.Label(window, text='Weight:').place(x = 20,y = 170)
    weight_text = tk.StringVar()          # 設定 weight 為文字變數
    weight_text.set('')                   # 設定 weight 的內容

    #Train score
    tk.Label(window, text='Train score:').place(x = 20,y = 200)
    Train_score_text = tk.StringVar()     # 設定 Train_score 為文字變數
    Train_score_text.set('')              # 設定 Train_score 的內容

    #Test score
    tk.Label(window, text='Test score:').place(x = 20,y = 230)
    Test_score_text = tk.StringVar()      # 設定 Test_score 為文字變數
    Test_score_text.set('')               # 設定 Test_score 的內容

    #iteration_times
    tk.Label(window, text='iteration_times:').place(x = 20,y = 260)
    iteration_times_text = tk.StringVar()     # 設定 iteration_times 為文字變數
    iteration_times_text.set('')              # 設定 iteration_times 的內容

    #開始預測資料
    tk.Button(window, text='確認',command= lambda: predict_data(
            window,float(learning_rate.get()),
            int(interation.get()),
            float(stop_rate.get()),
            weight_text,
            Train_score_text,
            Test_score_text,
            iteration_times_text
    )).place(x = 80,y = 140)
```

GUI 結果輸出部分

將輸入資料傳入 predict_data 進行資料前處理與預測

```python
def predict_data(window,learning_rate,iteration,stop_score,weight_text,Train_score_text,Test_score_text,iteration_times_text):
    # 1. 處理資料，先將資料 2/3 當作訓練資料，1/3 當做測試資料
    dataProcessor = preprocessor()
    dataset = dataProcessor.readfile(file_url)
    dataset = dataProcessor.convert_label(dataset)
    dataset_train,dataset_test = dataProcessor.split_train_test(dataset)
    dataset_train_feature,dataset_train_label = dataProcessor.split_feature_label(dataset_train)
    dataset_test_feature,dataset_test_label = dataProcessor.split_feature_label(dataset_test)

    # 2. 將訓練資料拿去訓練 => 有訓練完的model
    perceptron_ = perceptron(iteration, len(dataset_train_feature[0]) ,learning_rate)
    iteration_times = perceptron_.train(stop_score,dataset_train_feature,dataset_train_label)

    # 3. 將train data 跟 test data 拿去預測
    train_predict = perceptron_.allDataToPredict(dataset_train_feature)
    test_predict = perceptron_.allDataToPredict(dataset_test_feature)
    train_score = perceptron_.accuracy(train_predict,dataset_train_label)
    test_score = perceptron_.accuracy(test_predict,dataset_test_label)
    print(train_score,test_score)

    weight_text.set(perceptron_.weight)          # 設定 weight 的內容
    Train_score_text.set(train_score)            # 設定 train_score 的內容
    Test_score_text.set(test_score)              # 設定 test_score 的內容
    iteration_times_text.set(iteration_times)    # 設定 iteration_times 的內容
    #將結果印在GUI上
    tk.Label(window, textvariable=weight_text).place(x=100, y=170)

    tk.Label(window, textvariable=Train_score_text).place(x=100, y=200)

    tk.Label(window, textvariable=Test_score_text).place(x=100, y=230)

    tk.Label(window, textvariable=Test_score_text).place(x=100, y=230)

    tk.Label(window, textvariable=iteration_times_text).place(x=100, y=260)

    plot = ploter(window,perceptron_.weight,dataset)
    print(len(dataset_train_feature[0]))
    if(len(dataset_train_feature[0]) == 2):
        #畫2D的圖
        plot.two_dimension_plot()
    elif(len(dataset_train_feature[0]) == 3):
        #畫3D的圖
        plot.three_dimension_plot()
```

資料前處理交給 preprocessor 做

將處理好的資料交給 perceptron 進行訓練

將結果與資料交給 ploter 來印製

2. Preprocessor.py

在負責資料集的讀取、轉換、訓練測試資料集分割以及 shuffle 等功能

```python
class preprocessor:

    def __init__(self):
        pass

    #'./NN_HW1_DataSet/NN_HW1_DataSet/基本題/2cring.txt'
    def readfile(self,dataset_url):
        datas = open(dataset_url, 'r')
        dataset = []
        for line in datas:
            stripped_line = line.strip()
            line_list = stripped_line.split()
            line_list = list(map(float,line_list))
            dataset.append(line_list)

        return dataset

    def convert_label(self,dataset):
        check = dict()
        idx = 0
        for pos in range(len(dataset)):
            if dataset[pos][-1] not in check:
                check[dataset[pos][-1]] = idx
                idx += 1
            dataset[pos][-1] = check[dataset[pos][-1]]

        return dataset
```

讀取資料集，將空白切開轉成 list 格式，並且將資料使用 float 代替

處理 label，假如有兩群就是 0 1 三群就是 0 1 2，以此類推

```python
    def split_train_test(self,dataset):
        #先shuffle資料
        random.shuffle(dataset)
        split_boundary = math.ceil(len(dataset) * 2 / 3)

        return dataset[:split_boundary],dataset[split_boundary:]

    def split_feature_label(self,dataset):
        feature = [data[:-1] for data in dataset]
        label = [data[-1] for data in dataset]
        return np.array(feature),np.array(label)
```

使用 shuffle，並且將資料拆成 2/3 訓練，1/3 測試

將 feature 跟 label 拆開

3. Model.py

主要將使用者輸入的資訊與處理好的資料使用單層感知機進行訓練，並且負責正確率的計算，回傳給使用者

```python
class perceptron:                                   # 初始化，迭代次數，weight 大小，學習率
    def __init__(self,iteration,weight_len,learning_rate):
        self.iteration = iteration
        self.weight = np.zeros(weight_len + 1) #多一個給bias
        self.learning_rate = learning_rate

    def accuracy(self,actual,answer):
        correct = 0.0
        for actual_,answer_ in zip(actual,answer):      # 回傳正確率
            if(actual_ == answer_):
                correct += 1

        return correct / float(len(actual)) * 100.0

    def allDataToPredict(self,dataset_feature):
        test_predict = np.array([])
        for feature in dataset_feature:
            test_predict = np.append(test_predict,self.predict(feature))

        return test_predict

    def predict(self,feature):
        # v = feature * weight + bias
        # v > 0 => prediction 1
        # v < 0 => prediction -1

        prediction = np.dot(feature,self.weight[1:]) + self.weight[0]
        #predict_ans = -1
        if prediction > 0:
            predict_ans = 1
        else:                                           # 進行預測
            predict_ans = 0
        return predict_ans

    def train(self,stop_score,dataset_train_feature,dataset_train_label):
        for i in range(self.iteration):
            train_predict = self.allDataToPredict(dataset_train_feature)
            train_score = self.accuracy(train_predict,dataset_train_label)
            if(train_score >= stop_score):
                return i + 1
            for feature,label in zip(dataset_train_feature,dataset_train_label):
                predict_ans = self.predict(feature)

                if(predict_ans == 1 and label == 0):            # Weight 調整
                    # w(n) - nx(n)
                    self.weight[1:] = self.weight[1:] - (self.learning_rate * feature)
                    self.weight[0] = self.weight[0] - (self.learning_rate)
                elif(predict_ans == 0 and label == 1):
                    # w(n) + nx(n)
                    self.weight[1:] = self.weight[1:] + (self.learning_rate * feature)
                    self.weight[0] = self.weight[0] + (self.learning_rate)
        return self.iteration
```

4. Plot.py

Plot 會根據傳進的維數判斷要印製二維或三維資料，畫線是根據感知機算出的 weight 進行畫線

```python
class ploter:
    def __init__(self,window,weight,dataset):
        self.window = window
        self.weight = weight
        self.dataset = dataset

    def two_dimension_plot(self):
        f = Figure(figsize=(5, 4), dpi=100)
        f_plot = f.add_subplot(111)
        f_plot.clear()
        zero_class_x_feature = np.array([feature[0] for feature in self.dataset if feature[2] != 0])
        zero_class_y_feature = np.array([feature[1] for feature in self.dataset if feature[2] != 0])

        first_class_x_feature = np.array([feature[0] for feature in self.dataset if feature[2] != 1])
        first_class_y_feature = np.array([feature[1] for feature in self.dataset if feature[2] != 1])

        f_plot.scatter(zero_class_x_feature,zero_class_y_feature, color = 'hotpink')
        f_plot.scatter(first_class_x_feature,first_class_y_feature, color = '#88c999')

        x_min = min(min(zero_class_x_feature),min(first_class_x_feature))
        x_max = max(max(zero_class_x_feature),max(first_class_x_feature))
        print(x_min,x_max)
        x = np.arange(x_min - 3,x_max + 3,2)
        y = (-self.weight[1] / self.weight[2]) * x - (self.weight[0] / self.weight[2])
        f_plot.plot(x,y)

        canvs = FigureCanvasTkAgg(f, self.window)

        canvs.draw()

        canvs.get_tk_widget().place(x=300,y=80)
```

```python
    def three_dimension_plot(self):
        f = Figure(figsize=(5, 4), dpi=100)
        f_plot = f.add_subplot(111,projection='3d')
        f_plot.clear()
        zero_class_x_feature = np.array([feature[0] for feature in self.dataset if feature[3] != 0])
        zero_class_y_feature = np.array([feature[1] for feature in self.dataset if feature[3] != 0])
        zero_class_z_feature = np.array([feature[2] for feature in self.dataset if feature[3] != 0])

        first_class_x_feature = np.array([feature[0] for feature in self.dataset if feature[3] != 1])
        first_class_y_feature = np.array([feature[1] for feature in self.dataset if feature[3] != 1])
        first_class_z_feature = np.array([feature[2] for feature in self.dataset if feature[3] != 1])


        f_plot.scatter(zero_class_x_feature,zero_class_y_feature,zero_class_z_feature, color = 'hotpink')
        f_plot.scatter(first_class_x_feature,first_class_y_feature,first_class_z_feature, color = '#88c999')

        x = np.linspace(0, 2, 2)
        y = np.linspace(0, 2, 2)
        X, Y = np.meshgrid(x, y)

        f_plot.plot_surface(
            X,
            Y = -(self.weight[0] / self.weight[2]) - (self.weight[1] / self.weight[2]) - (self.weight[3] / self.weight[2]),
            Z = Y,
            color='g',
            alpha=0.6
        )

        f_plot.set_xlabel('X')
        f_plot.set_ylabel('Y')
        f_plot.set_zlabel('Z')

        canvs = FigureCanvasTkAgg(f, self.window)
        canvs.draw()
        canvs.get_tk_widget().place(x=300,y=80)
```

- 實驗結果

  ➢ 基本題

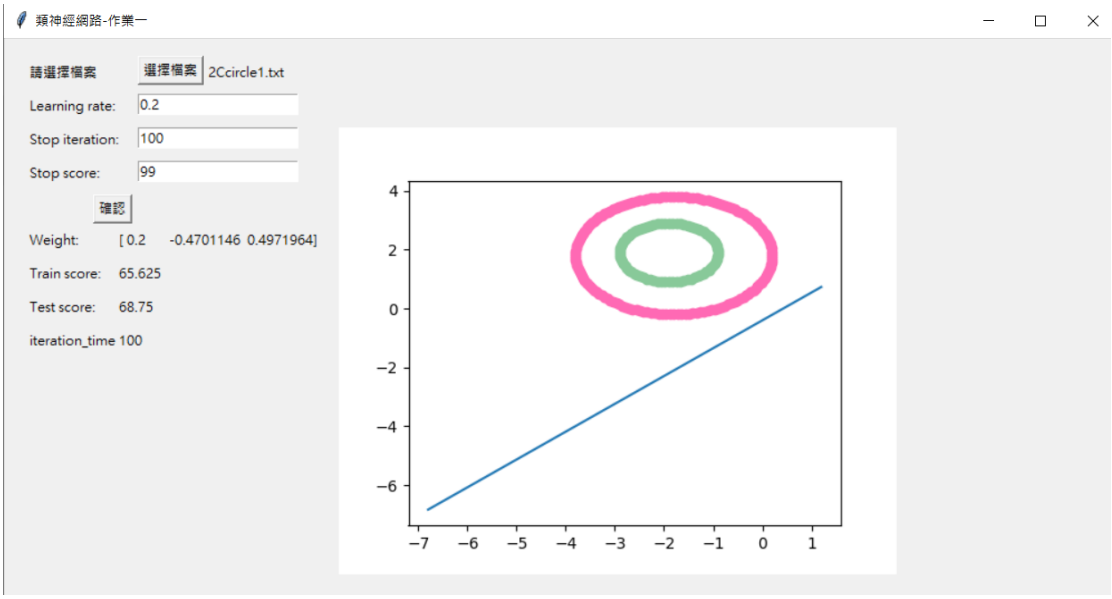
1. 2Ccircle1.txt
   - 使用者輸入資料

選擇檔案	2Ccircle1.txt
學習率	0.2
迭代次數	100
訓練集提前結束分數	99

   - 輸出結果

Weight	[0.2,-0.4701146,0.4971964]
訓練集分數	65.625
測試集分數	68.75
訓練集總迭代訓練次數	100

   - 實驗截圖



   - 實驗敘述

     在資料屬於有出現圓中圓的情況時，在二維的情況，對於單層感知器無法去分類，或許需要多層感知機讓分類器可以針對圓型去進行分割才有機會可以辨識出來

2. 2Circle1.txt
   - 使用者輸入資料

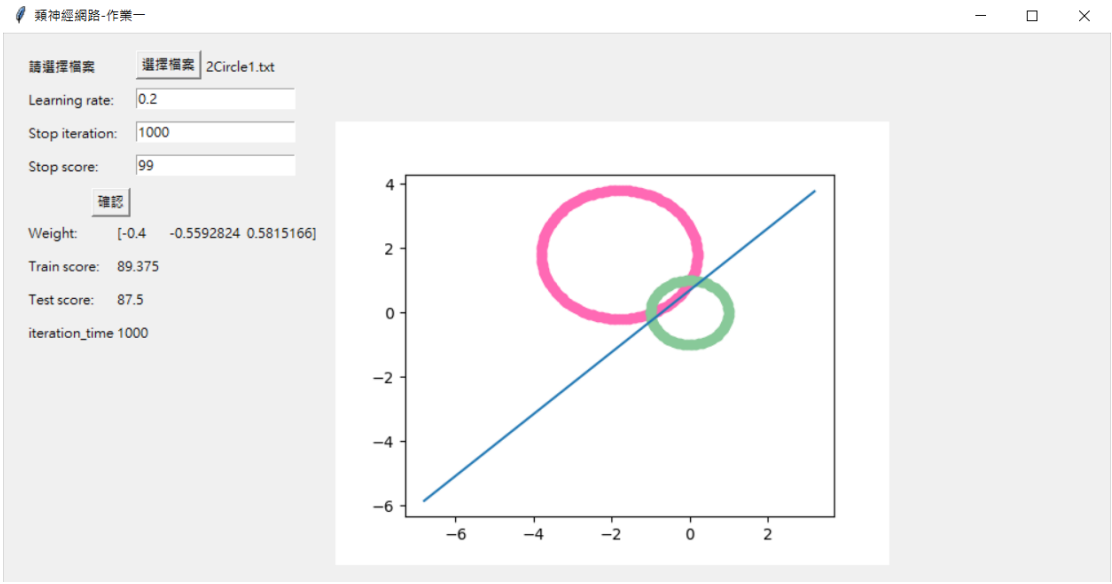
選擇檔案	2Circle1.txt

學習率	0.2
迭代次數	1000
訓練集提前結束分數	99

- 輸出結果

Weight	[0.4, -0.5592824, 0.5815166]
訓練集分數	89.375
測試集分數	87.5
訓練集總迭代訓練次數	1000

- 實驗截圖



- 實驗描述

  此情況比 2Ccircle1.txt 好上不少，因為在二維的情況下，資料沒有相交時通常都可以分的出來，但在這個情況仍然有些許部分重疊，所以在辨識率上仍然沒辦法達到 100%。

3. 2CloseS.txt

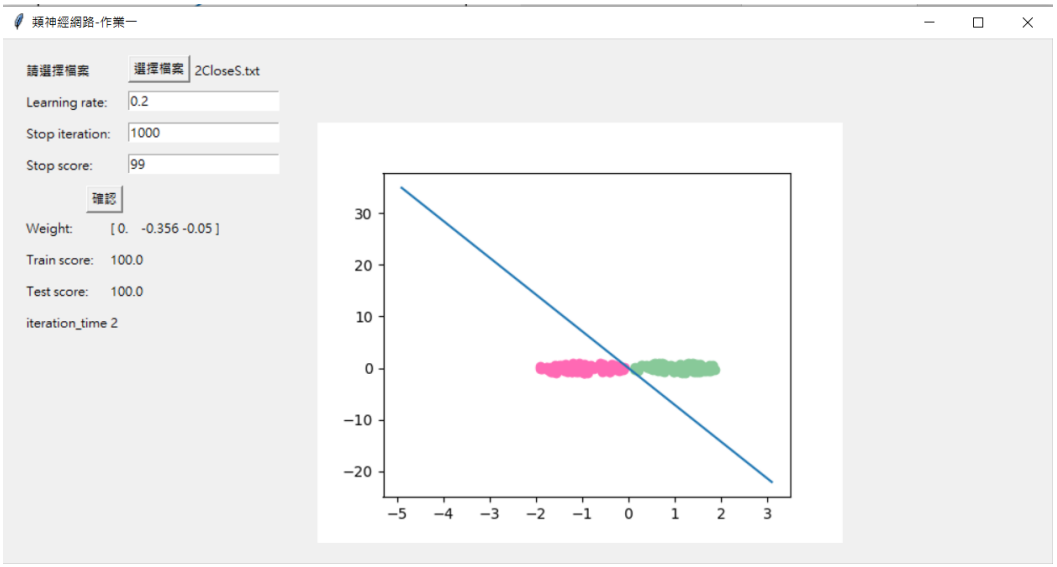
- 使用者輸入資料

選擇檔案	2CircleS.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99

- 輸出結果

Weight	0. ,-0.356 , -0.05]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	2

- 實驗截圖



4. 2CloseS2.txt

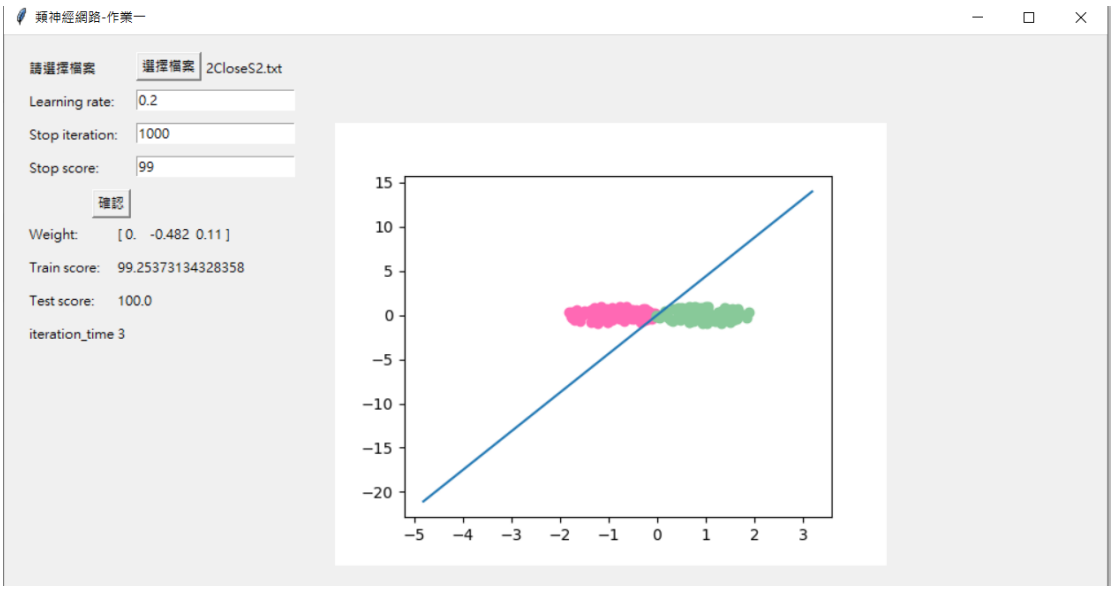
- 使用者輸入資料

選擇檔案	2CircleS2.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99

- 輸出結果

Weight	[0. , -0.482 ,0.11]
訓練集分數	99.2537313428358
測試集分數	100.0
訓練集總迭代訓練次數	3

- 實驗截圖



- 實驗描述

  除了資料量夠多以外，以資料集來說只有出現一點交集的情況，因此，在取測試集時，不一定每次都 100%，但相交的部分不多，算是分的不錯。

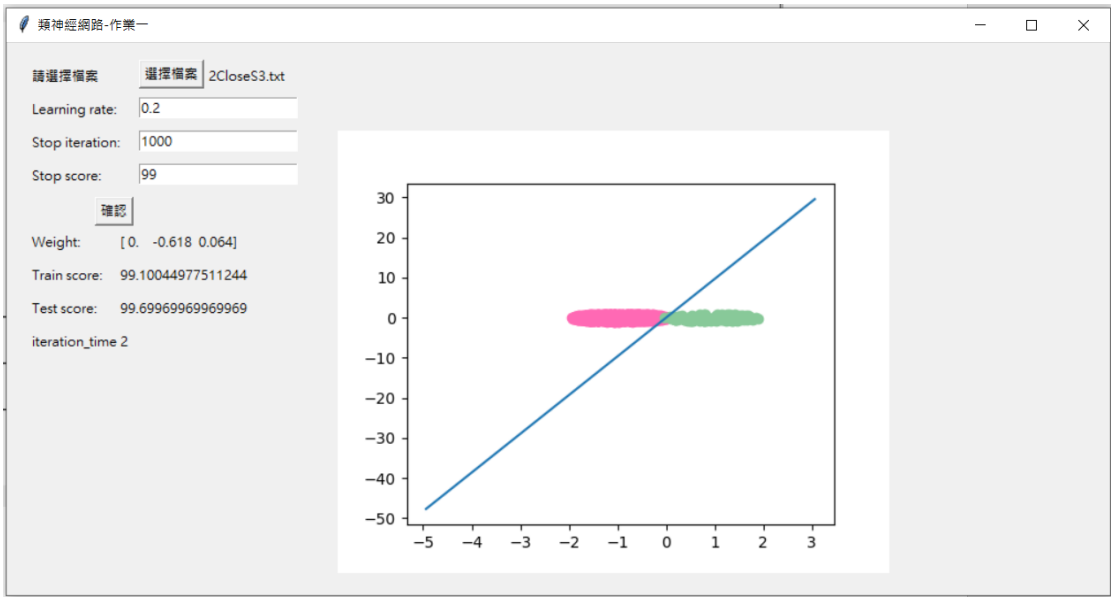
5. 2CloseS3.txt
   - 使用者輸入資料

選擇檔案	2CircleS3.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99

   - 輸出結果

Weight	[0 , -0.618 , 0.064]
訓練集分數	99.1004977511244
測試集分數	99.69969969969969
訓練集總迭代訓練次數	2

   - 實驗截圖



   - 實驗描述

     此資料集有出現部分資料相交的情況，因此準確率就很難達到 100%，因為在取 1/3 時，有非常高的機率會取到辨識錯誤的資料。

6. 2cring.txt
   - 使用者輸入資料

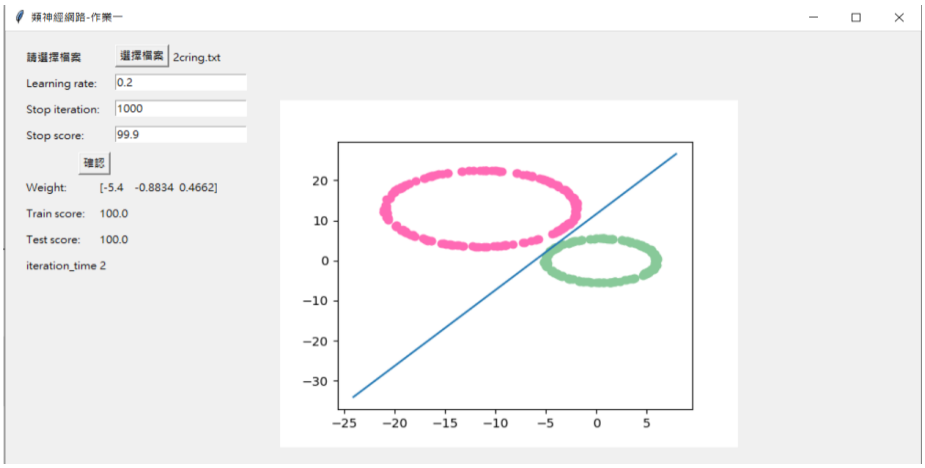
選擇檔案	2cring.txt
學習率	0.2
迭代次數	1000

訓練集提前結束分數	99

- 輸出結果

Weight	[-5.4, -0.8834, 0.4662]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	2

- 實驗截圖



- 實驗截圖

  在這個例子上就可以看的出來，比較 2Ccircle.txt 圓已經沒有出現交集的情況，所以感知機在一次 Epoch 就可以正確進行分類。

7.2CS.txt

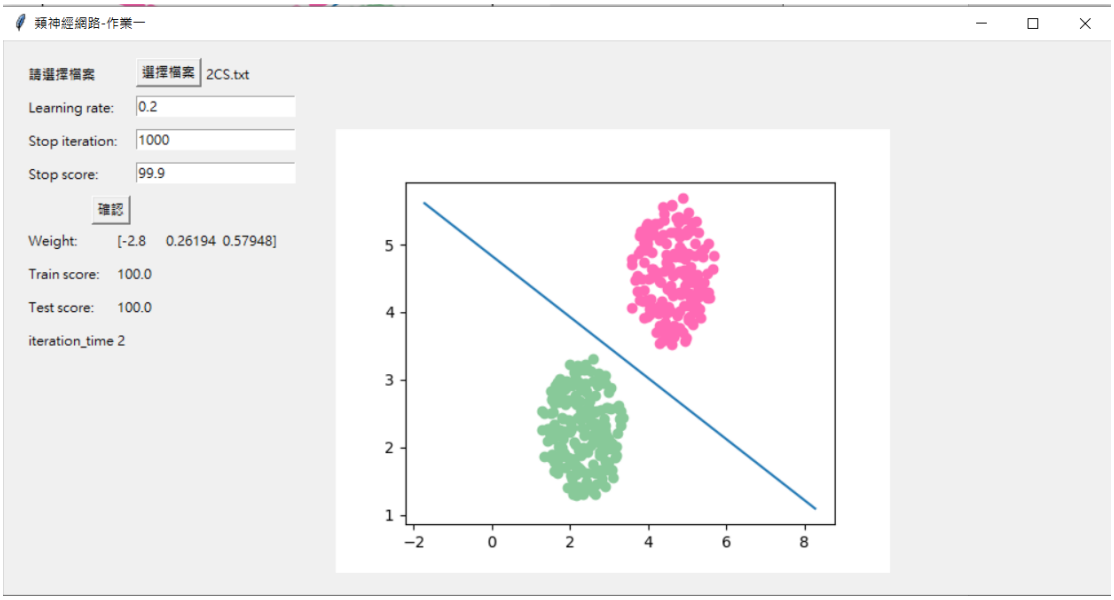
- 使用者輸入資料

選擇檔案	2CS.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99.9

- 輸出結果

Weight	[-2.8, 0.26194, 0.57948]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	2

- 實驗截圖



- 實驗截圖

  在這個例子上就可以看的出來，比較 2Ccircle.txt 圓已經沒有出現交集的情況，所以感知機在一次 Epoch 就可以正確進行分類。

8.2Hcircle1.txt

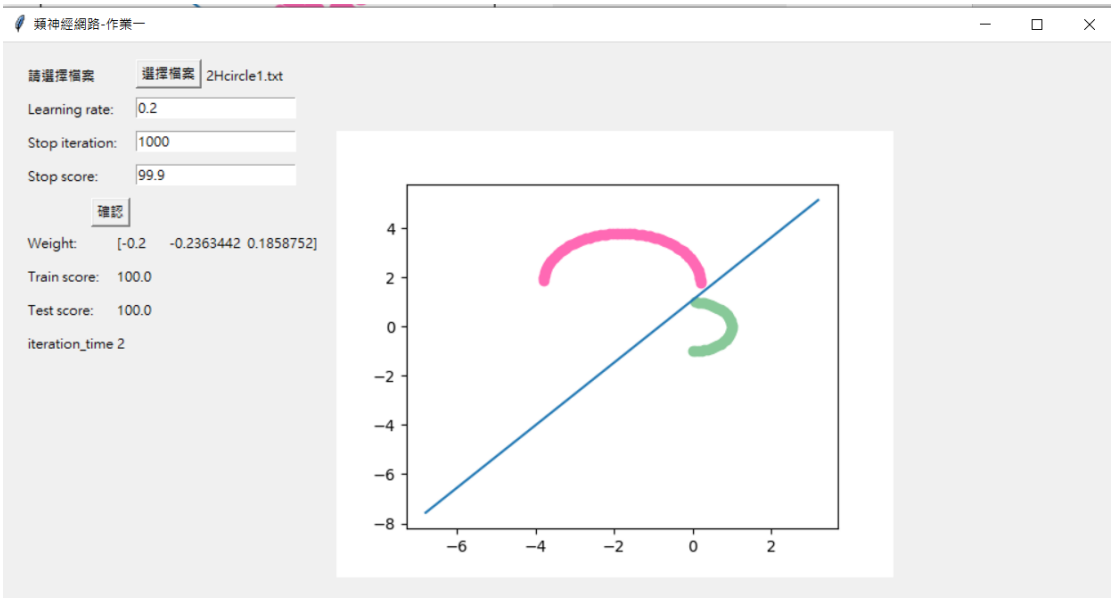
- 使用者輸入資料

選擇檔案	2Hcircle1.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99.9

- 輸出結果

Weight	[-0.2, -0.2363442, 0.1858752]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	2

- 實驗截圖



- 實驗描述

  同 2CS.txt

9.2ring.txt

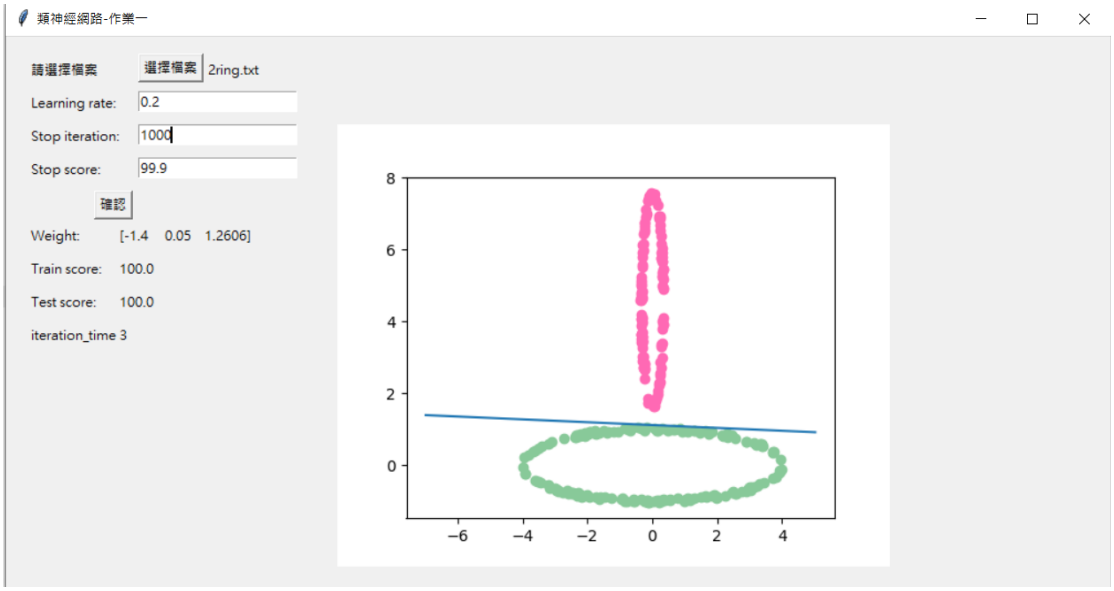
- 使用者輸入資料

選擇檔案	2ring.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99

- 輸出結果

Weight	[-1.4, 0.05 , 1.2606]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	3

- 實驗截圖



- 實驗描述

  同 2CS.txt

10.perceptron1.txt

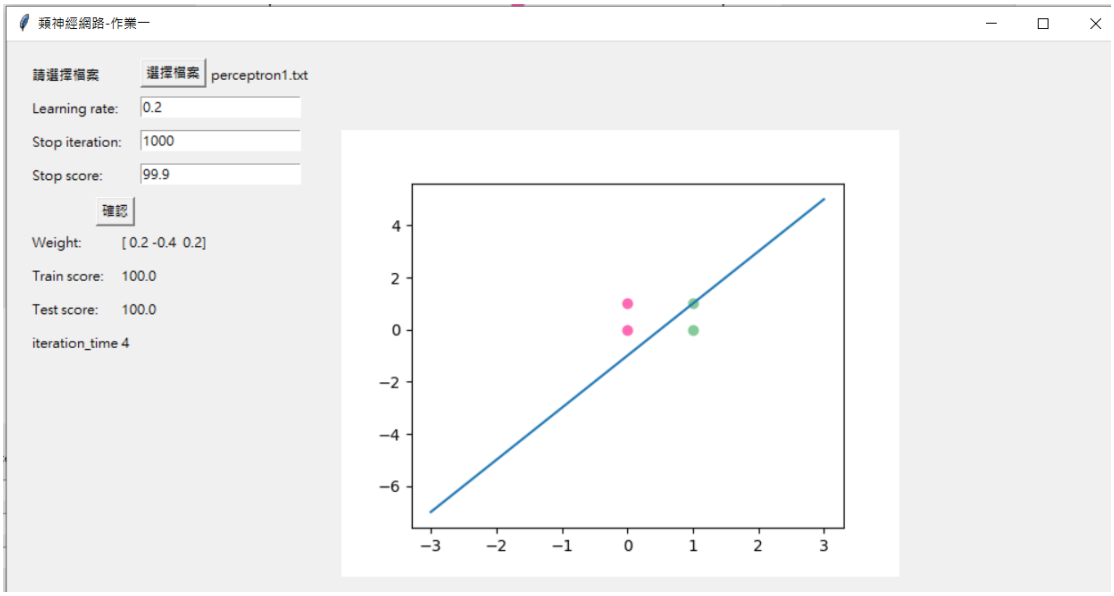
- 使用者輸入資料

選擇檔案	Perceptron1.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	99.9

- 輸出結果

Weight	[0.2, -0.4, 0.2]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	4

- 實驗截圖



- 實驗描述

  在此例子中可以發現，由於點的樣本數少又拆分成訓練、測試資料的關係，測試準確率常會很低，不論迭代次數和學習率是多少，因為訓練資料只有兩個點可以分類。

11. perceptron2.txt

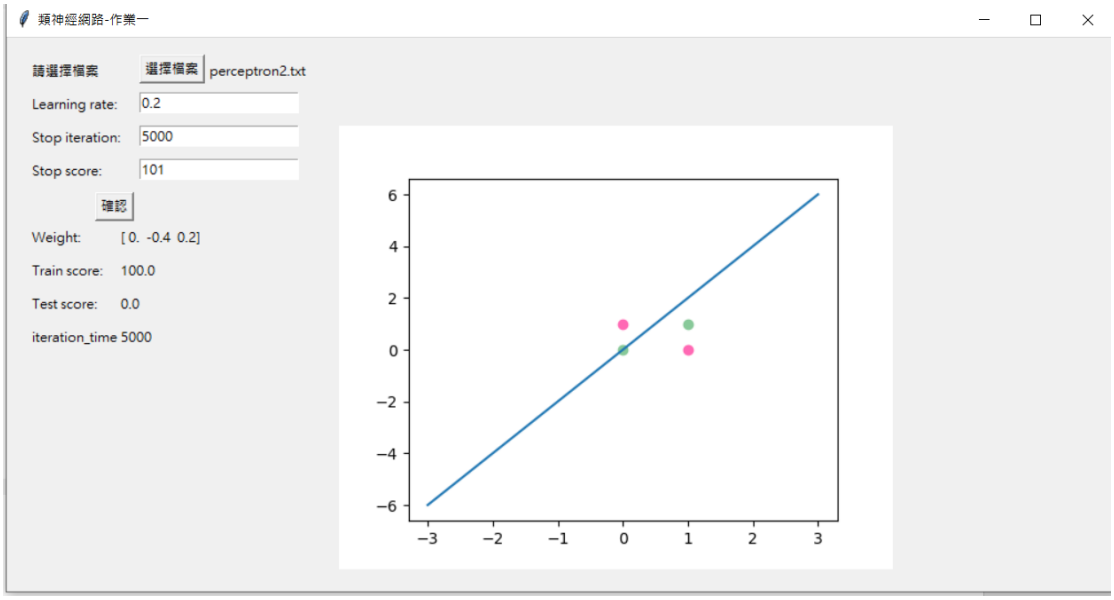
- 使用者輸入資料

選擇檔案	Perceptron2.txt
學習率	0.2
迭代次數	5000
訓練集提前結束分數	101

- 輸出結果

Weight	[0.2,-0.4701146,0.4971964]
訓練集分數	100.0
測試集分數	0.0
訓練集總迭代訓練次數	5000

- 實驗截圖



- 實驗描述

  在對角各別有資料的時候沒辦法使用單層感知機找到正確的答案，因為此情況沒辦法一分為二。

➢ 加分題

1. perceptron3.txt

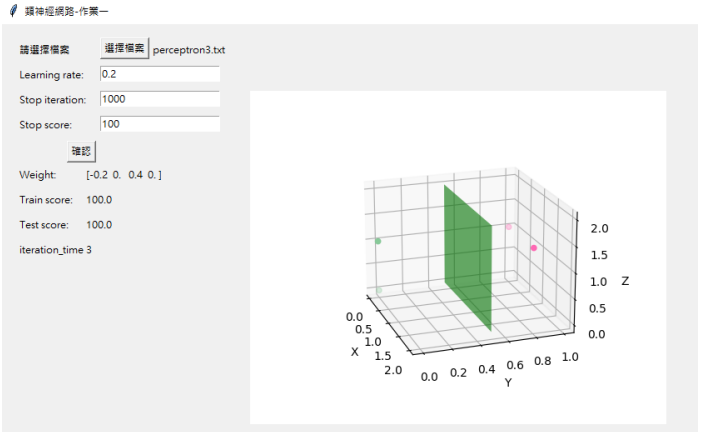
- 使用者輸入資料

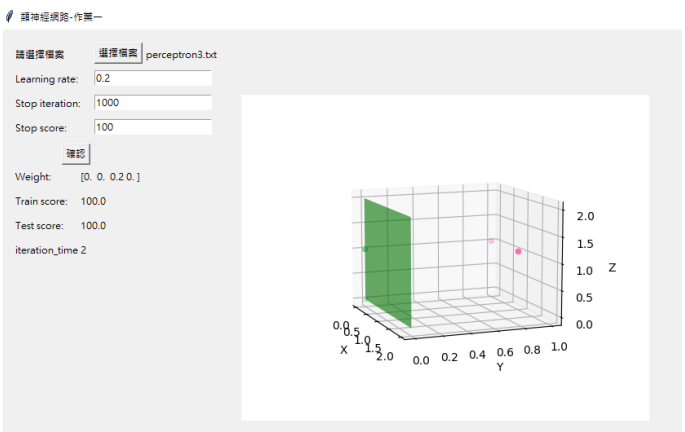
選擇檔案	Perceptron3.txt
學習率	0.2
迭代次數	1000
訓練集提前結束分數	100

- 輸出結果

Weight	[-0.2, -0. , 0.4, 0.]
訓練集分數	100.0
測試集分數	100.0
訓練集總迭代訓練次數	3

- 實驗截圖





- 實驗描述

  在處理三維的資料上可以使用平面的方式還區分正確的資料，不過有時候會有錯誤(平面切錯)，推測可能是因為資料量不夠，導致 weight 調整不夠。

- 實驗結果分析及討論。

1. 辨識不佳的情況

➢ 圓中圓

在 2Ccircle1.txt , 2Circle1.txt 中有圓中圓，沒有辦法在同一個圓上進行實踐分類，因為只能一分為二，因此通常有資料交疊的情況就會辨識很不好。

➢ 對角線各別有同一類圖形

對角線類別如 perceptron2.txt，在辨識的時候效果很不好，因為硬分解器也無法找到一條線分出兩群，除非用多層感知機，可以直接 and 或 xor 就可以正確分出這種類別。

➢ 資料集數量不足，效果不好

在 perceptron1.txt 只有四個點，發現我在取資料集 2/3 時是取 celi，所以訓練集有三個資料，測試集只有一個資料，因為訓練結果有時候會有圖畫不出來，還有測試集辨識率不是 100 就是 0 分的情況發生，個人認為當資料集的個數變大，那訓練出來的感知機效果也會好不少。

2. 學習率大小與迭代次數對於硬分解器差異不大

在每一個測資下，在迭代次數固定為 100 的情況下，不論學習率調成 10000000 或是 0.00000001，效果都跟 0.2 差異不大，反之，在學習率固定為 0.2 的情況下，迭代次數設為 5 或是 1000000，都得出差不多的結果，回去看實驗結果，可以發現通常好分類的資料在 1-5 次內就可以辨識出來，而難分辨的資料(圓中圓，perceptron2.txt)就不論迭代到幾次通常都會到達某個上限之後就只會有細微的變化。