A. 程式執行說明 (GUI 功能說明)

填上學習率、和訓練次數，再讀取資料，按 training 後則可得到圖示，左上角為訓練資料，右上角為驗證資料，另外，可得到分別的準確率和鍵結值。

B. 程式碼簡介

```
import tkinter as tk
from tkinter import filedialog, messagebox
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

plt.ioff()  # 关闭弹出的图形窗口

class Perceptron:
# 初始化 weight 和 bias ，學習率和疊代次數
    def __init__(self, input_size, X_mean, Y_mean, learning_rate, n_iters):
        self.weights = X_mean
        self.bias = Y_mean
        self.learning_rate = learning_rate
        self.n_iters = n_iters
# 活化函數，大於 0 是 1，小於 0 是 0
    def activation_function(self, x):
        return np.where(x >= 0, 1, 0)
# 得到預測值
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.activation_function(linear_output)
# 疊代後根據誤差調整 weight 和 bias
    def fit(self, X, y, ax_train, ax_test, fig):
        for iteration in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_pred = self.activation_function(linear_output)
                update = self.learning_rate * (y[idx] - y_pred)
                self.weights += update * x_i
```

```
                self.bias += update

            # 每次迭代后绘制分类线
            self.plot_decision_boundary(X, y, ax_train, ax_test, iteration, fig)

        # 训练结束后计算准确率
        return self.weights, self.bias
#畫圖，決策邊界和資料點
    def plot_decision_boundary(self, X_train, y_train, ax_train, ax_test, iteration,
fig):
        ax_train.clear()
        ax_test.clear()

        # 决策边界
        x1_min, x1_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
        x2_min, x2_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
        x1_values = np.array([x1_min, x1_max])

        # 绘制训练集的分类线
        ax_train.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='bwr')
        if self.weights[1] != 0:
            x2_values = -(self.weights[0] / self.weights[1]) * x1_values - (self.bias /
self.weights[1])
            ax_train.plot(x1_values, x2_values, 'k-', label=f'Iteration {iteration +
1}')
        ax_train.set_title(f'Training Set (Iteration {iteration + 1})')

        # 绘制测试集的分类线（与训练集一致）
        ax_test.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='bwr')
        if self.weights[1] != 0:
            ax_test.plot(x1_values, x2_values, 'k-', label=f'Iteration {iteration + 1}')
        ax_test.set_title(f'Test Set (Iteration {iteration + 1})')

        fig.canvas.draw()

    def calculate_accuracy(self, X, y):
        predictions = self.predict(X)
        accuracy = np.mean(predictions == y)
```

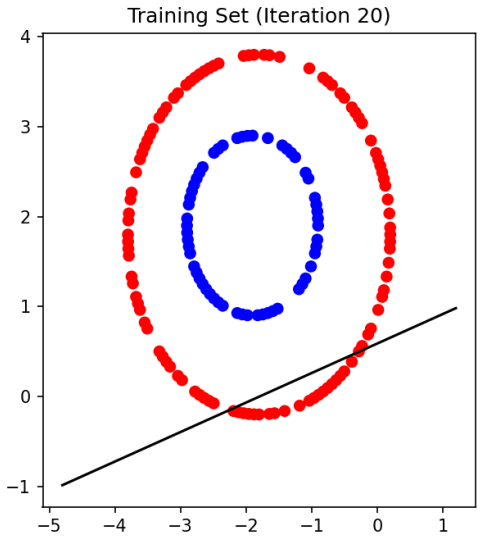
```
            return accuracy
# 訓練資料，並且計算準確度
def start_training():
    try:
        learning_rate = float(learning_rate_entry.get())
        n_iters = int(epoch_entry.get())

        if not hasattr(start_training, "data"):
            messagebox.showerror("Error", "Please load a dataset first!")
            return

        X, y = start_training.data
        y = y - np.min(y)  # 处理标签

        global X_train, X_test, y_train, y_test
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

        # 创建感知机并进行训练
        perceptron = Perceptron(input_size=2, X_mean=np.mean(X_train, axis=0),
Y_mean=np.mean(y_train), learning_rate=learning_rate, n_iters=n_iters)
        weights, bias = perceptron.fit(X_train, y_train, ax_train, ax_test, fig)

        # 计算训练集和测试集的准确率
        train_accuracy = perceptron.calculate_accuracy(X_train, y_train)
        test_accuracy = perceptron.calculate_accuracy(X_test, y_test)

        # 更新界面上的显示
        train_accuracy_label.config(text=f"Training Accuracy: {train_accuracy:.2f}")
        test_accuracy_label.config(text=f"Test Accuracy: {test_accuracy:.2f}")
        weights_label.config(text=f"Weights: {weights}, Bias: {bias:.2f}")
    except ValueError as e:
        messagebox.showerror("输入错误", f"无效输入: {e}")
# 讀取資料，得到 X,Y
def load_dataset():
    filepath = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
    if not filepath:
        return
```

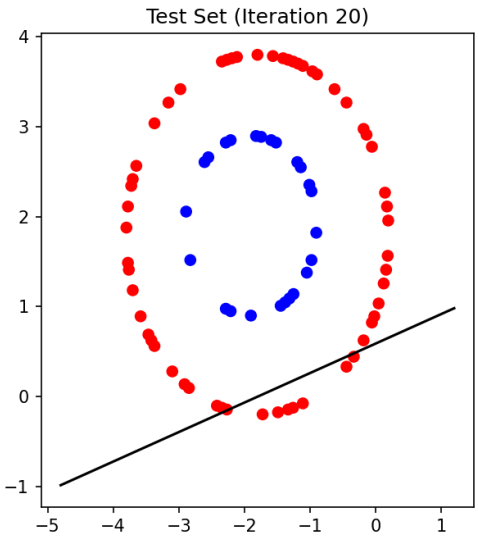
```
        dataset_name.set(filepath.split('/')[-1])
        column_names = ['X1', 'X2', 'Y']
        data = pd.read_csv(filepath, delim_whitespace=True, header=None,
names=column_names)
        X = np.array(data.iloc[:, :-1])
        y = np.array(data.iloc[:, -1])
        start_training.data = (X, y)


#創界圖形介面

# 创建主窗口
root = tk.Tk()
root.title("感知机设置")

# 创建图形显示区域
fig, (ax_train, ax_test) = plt.subplots(1, 2, figsize=(10, 5))  # 创建两个子图
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().grid(row=0, column=0, columnspan=4)

# 显示数据集名称
dataset_name = tk.StringVar()
tk.Label(root, text="Dataset:").grid(row=1, column=0)
tk.Label(root, textvariable=dataset_name).grid(row=1, column=1)
load_button = tk.Button(root, text="Load Dataset", command=load_dataset)
load_button.grid(row=1, column=2)

# 学习率
tk.Label(root, text="Learning rate:").grid(row=2, column=0)
learning_rate_entry = tk.Entry(root)
learning_rate_entry.grid(row=2, column=1)

# 迭代次数（收敛条件）
tk.Label(root, text="Epoch:").grid(row=3, column=0)
epoch_entry = tk.Entry(root)
epoch_entry.grid(row=3, column=1)
```

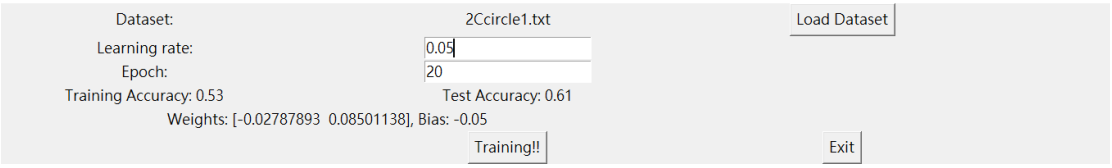
```
# 显示训练集和测试集的准确率
train_accuracy_label = tk.Label(root, text="Training Accuracy: ")
train_accuracy_label.grid(row=4, column=0)
test_accuracy_label = tk.Label(root, text="Test Accuracy: ")
test_accuracy_label.grid(row=4, column=1)

# 显示权重值
weights_label = tk.Label(root, text="Weights: ")
weights_label.grid(row=5, column=0, columnspan=2)

# 开始训练按钮
train_button = tk.Button(root, text="Training!!", command=start_training)
train_button.grid(row=6, column=1)

# 退出按钮，关闭整个窗口
exit_button = tk.Button(root, text="Exit", command=root.destroy)
exit_button.grid(row=6, column=2)

# 运行 GUI 主循环
root.mainloop()
```

C.實驗結果

2Ccircle1.txt

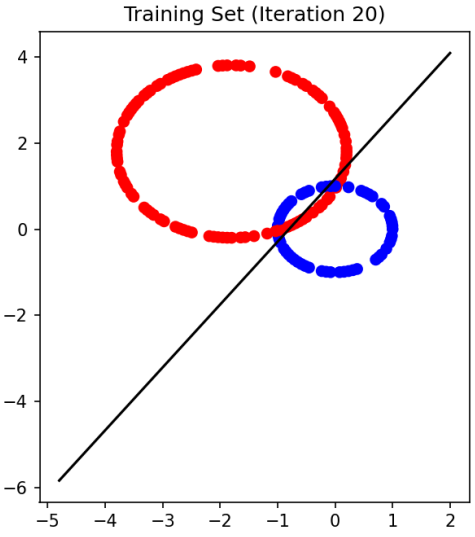


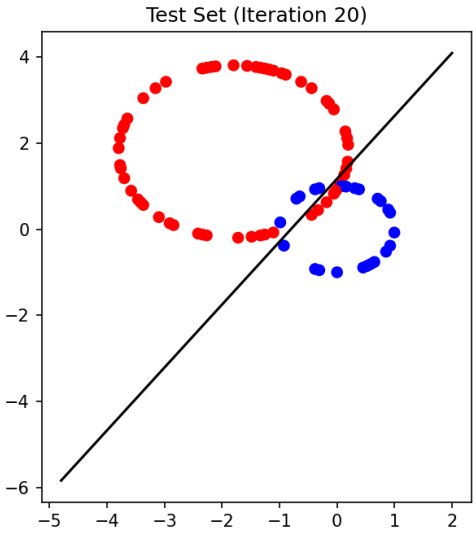


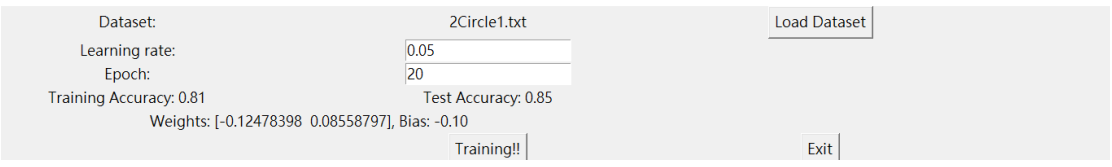


由於非線性可分割，無法做到 100%

2Circle1.txt

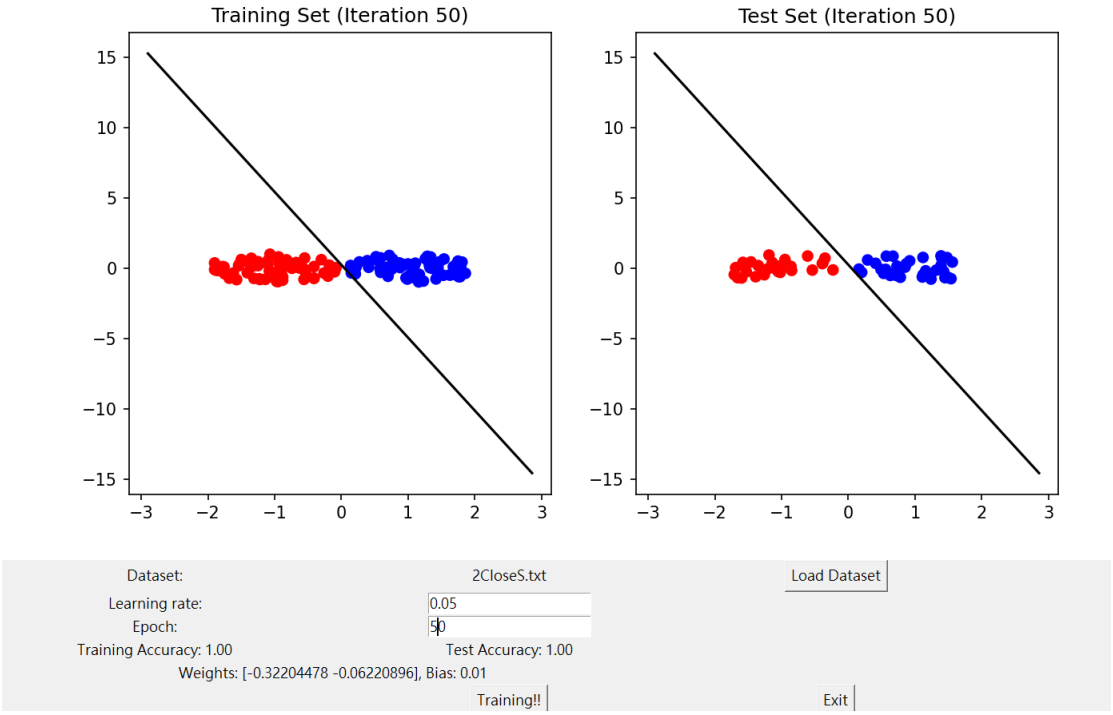






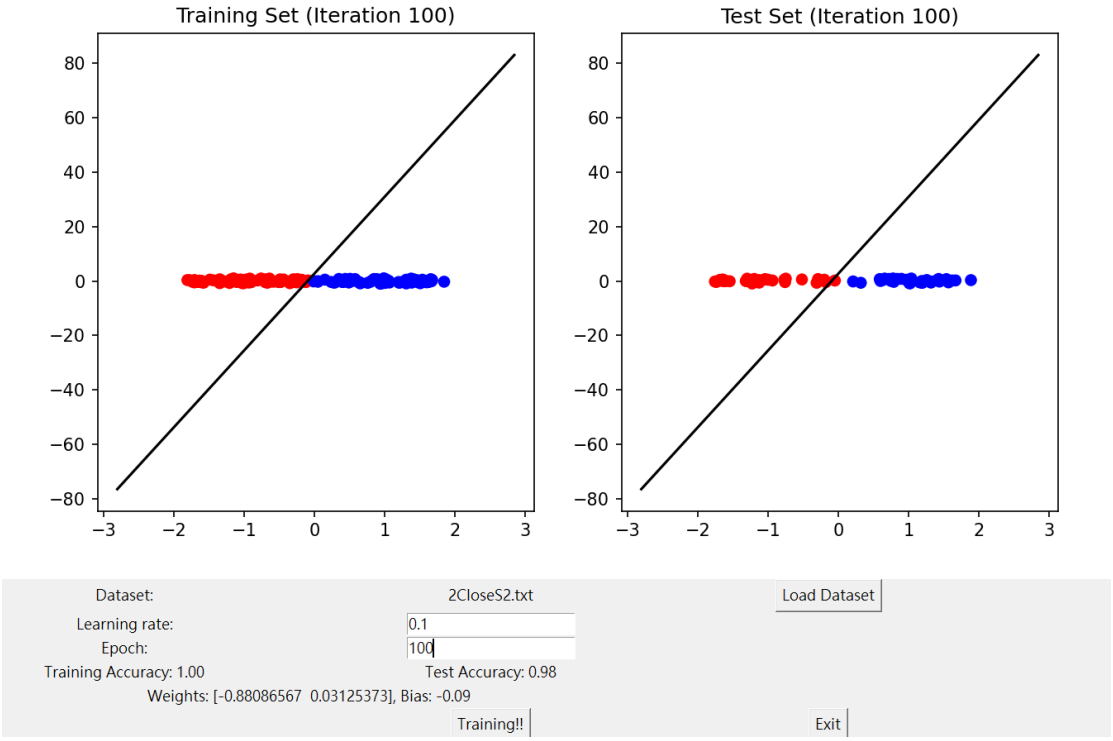
較易分割，準確率較高

2CloseS.txt



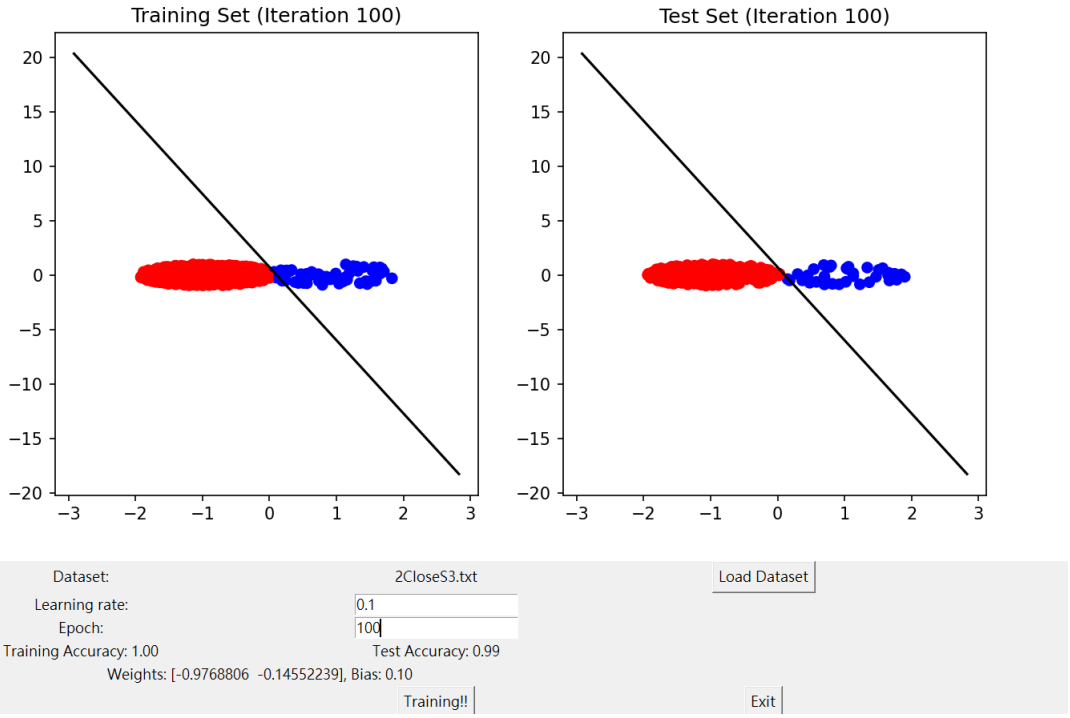
完全分開，為線性可分割，準確率高

2CloseS2.txt



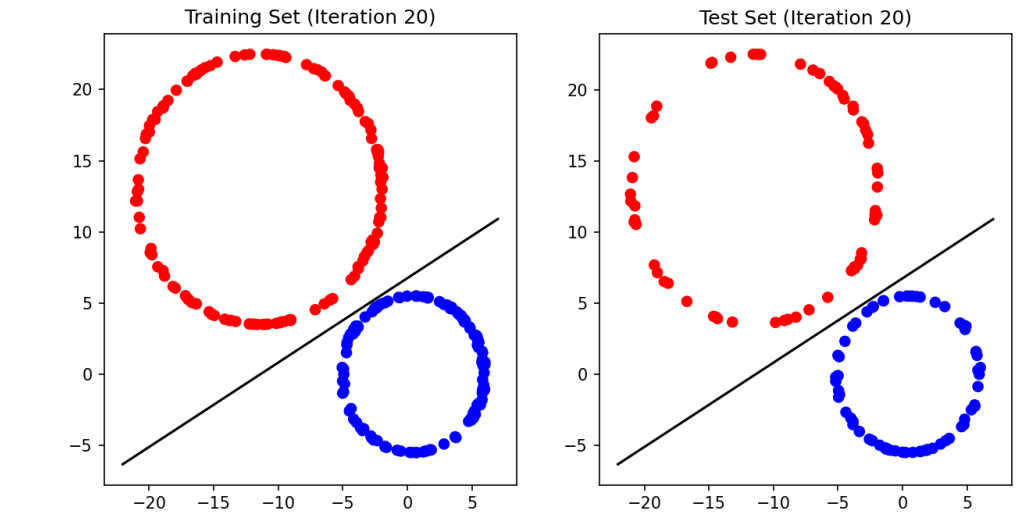
由於資料非常靠近，在切分資料時，test data 有些資料點超出 training data 範圍，導致 test data 的準確率有些下降。
而且，感知機是只要分對了就不會繼續修正到最佳決策邊界，不像 SVM 會到最佳決策邊界。

2CloseS3.txt



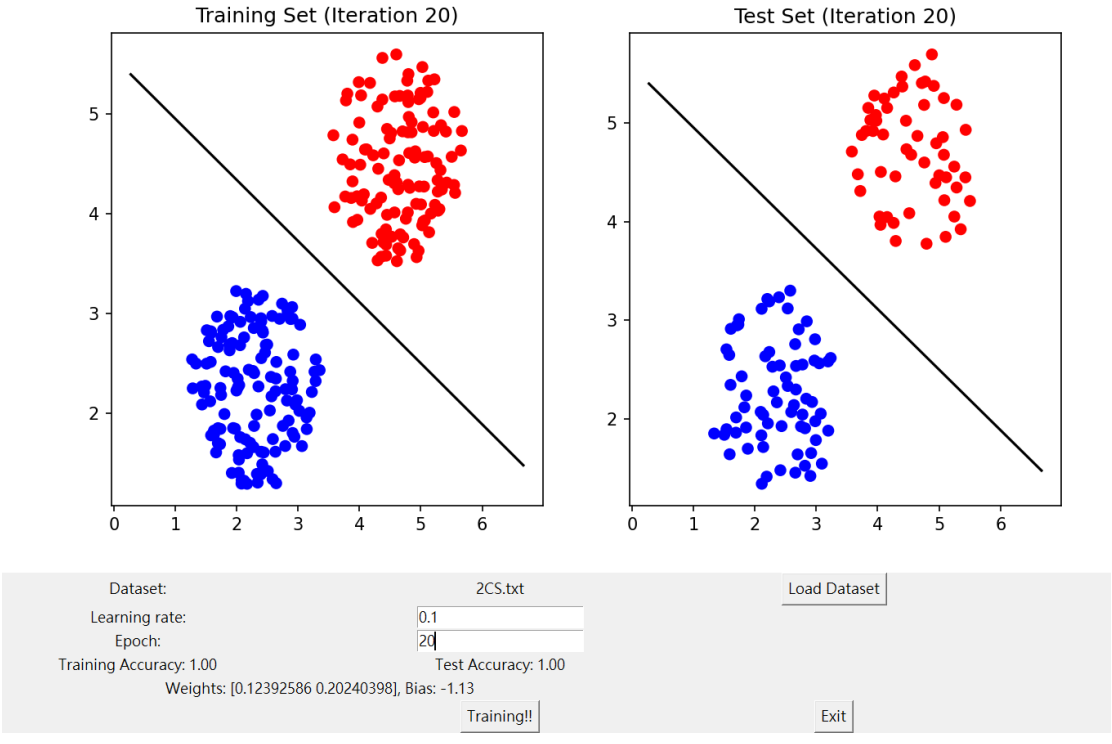
Test 不是 100% 準確儘管線性可分割，理由同 2CloseS2.txt

2cring.txt



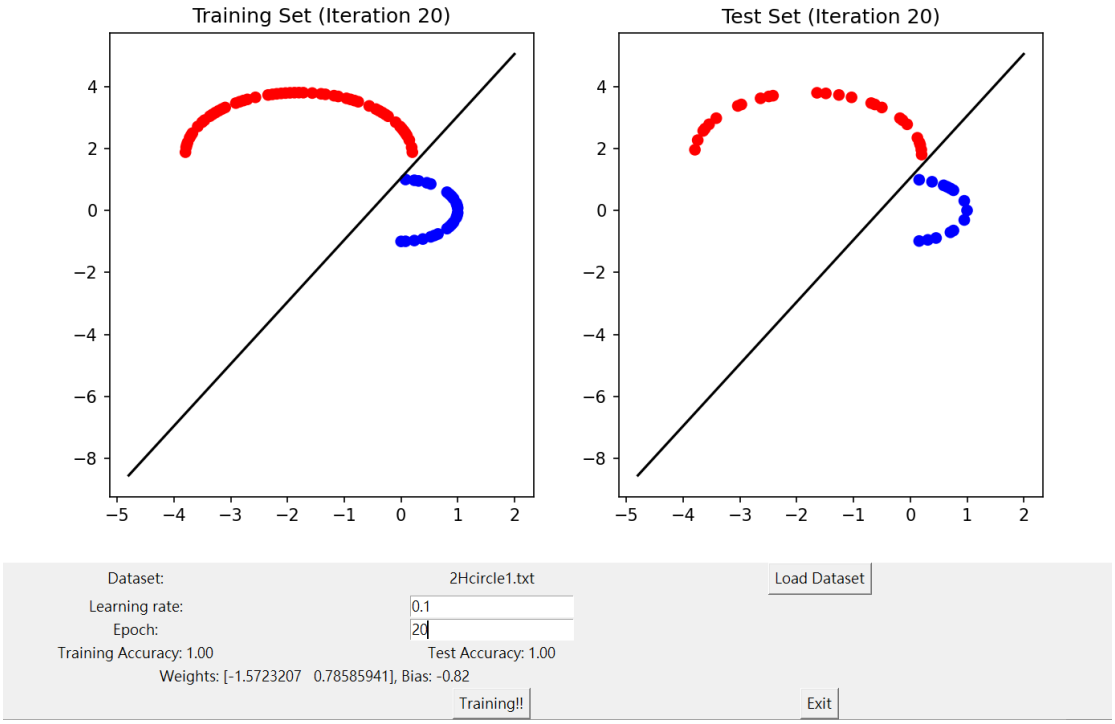
線性可分割

2CS.txt



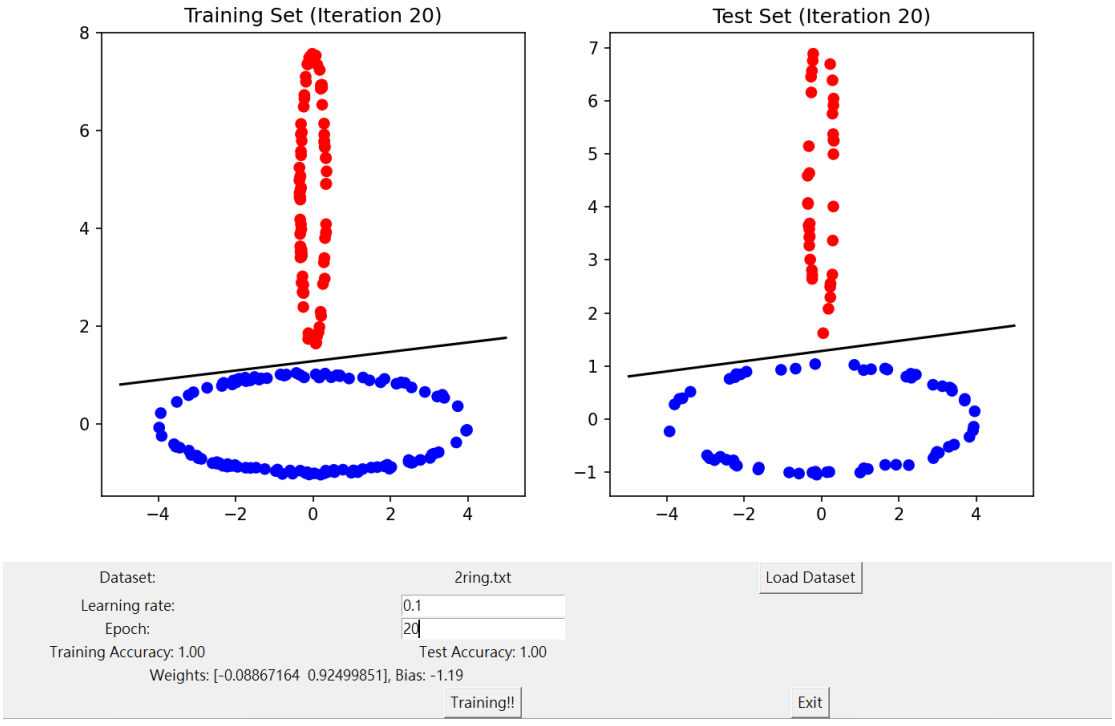
線性可分割

2Hcircle1.txt



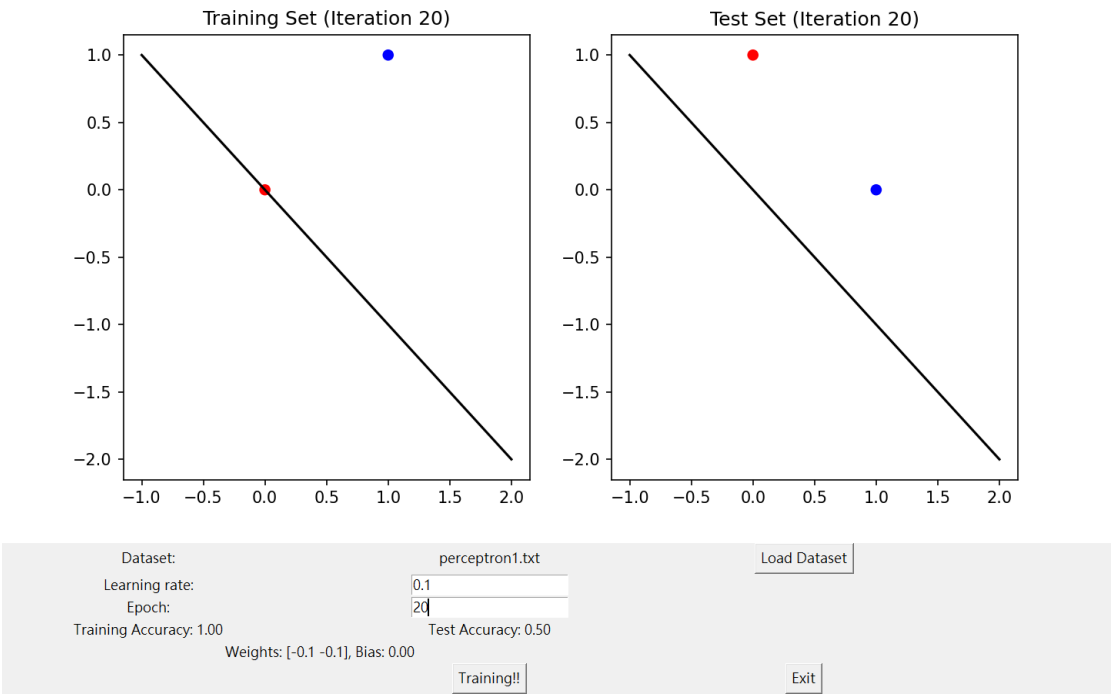
線性可分割

2ring.txt



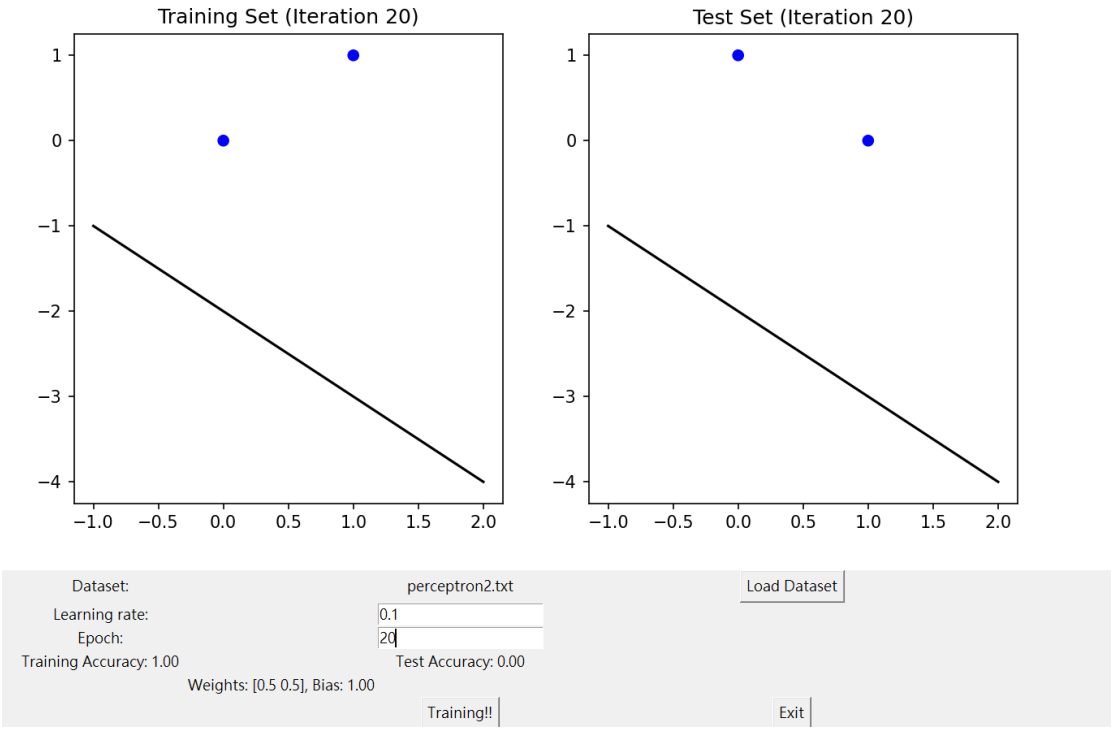
線性可分割

perceptron1.txt



資料點過少，無法切分 training data 和 test data，切分結果導致模型不具代表性，test data 準確率不佳。

perceptron2.txt



理由同 perceptron1.txt

D. 實驗結果分析及討論

基本上學習率不要大到太誇張，都可以得到好的結果。
訓練次數 大約 20 次即可得到最後結果，再大效果差不多，不過主要還是配合學習率為多少。
訓練正確率，只要線性可分割，訓練正確率基本上 100%
測試正確率，如果是線性可分割，但是沒有達到 100% ，代表資料點太靠近，所以 test 資料會出現 training data 訓練時無法預料的狀況，如果決策邊界是像 SVM 一樣，是要切在最中間的位置，可能結果會好一點。
資料點不夠多會產生模型適應性不足的狀況，測試資料效果一般很差。
鍵結值就是決定決策邊界，基本上到了決策邊界後，就不太會調整了。