

HW3_report

0616237 廖俊崐

1. Linear Regression:

先從檔案一行一行讀進資料，並利用逗號把每行的 x,y 分開，再用 2 維 list

把 x,y 資料儲存起來，再要求輸入 polynomial base number 並把它儲存

起來。接著再根據 Polynomial base 把 x 從 list 拿出來擴展成 data

num*base number 矩陣 A,再把 y 從 list 拿出來擴展成 data num *1 矩

$$\begin{aligned} Ax &= b \\ x &= (A^T A)^{-1} A^T b \\ x &= A^{-1} b \end{aligned}$$

陣 b,接著利用 PDF 的方程式求出 x。

最後把相關資料 Output 出來。

求反矩陣跟矩陣相乘我各寫了一個 function 實作:

```
def multiply_matrix(matrix_a, matrix_b):
    mul_matrix = [[0 for i in range(len(matrix_b[0]))] \
                   for j in range(len(matrix_a))]

    for i in range(len(matrix_a)):
        for j in range(len(matrix_b[i])):
            for k in range(len(matrix_a[i])):
                mul_matrix[i][j] += matrix_a[i][k] * matrix_b[k][j]

    return mul_matrix
```

矩陣相乘

```

def inverse_matrix(matrix):
    inverse_matrix=[[0 if i!=j else 1 for i in range(len(matrix)) ] \
                    for j in range(len(matrix))]

    for i in range(len(matrix)):
        if i==0:
            div_num=matrix[i][0]
            for j in range(len(matrix[i])):
                matrix[i][j] /=div_num
                inverse_matrix[i][j] /=div_num
            else:
                for k in range(i,len(matrix)):
                    mul= -(matrix[k][i-1]/matrix[i-1][i-1])
                    for j in range(len(matrix[k])):
                        matrix[k][j]=matrix[i-1][j]*mul+matrix[k][j]
                        inverse_matrix[k][j] =inverse_matrix[i-1][j]*mul+invers
e_matrix[k][j]

        final_index=len(matrix)-1
        last=len(matrix[0])-1

        for i in range(len(matrix)):
            index=final_index-i
            if index==final_index:
                div_num=matrix[index][last]
                for j in range(len(matrix[index])):
                    matrix[index][j] /=div_num
                    inverse_matrix[index][j] /=div_num
            else:
                div_num=matrix[index][index]
                for z in range(len(matrix[index])):
                    matrix[index][z] /=div_num
                    inverse_matrix[index][z] /=div_num
                for k in range(i,len(matrix)):
                    ind=final_index-k
                    mul= -(matrix[ind][index+1]/matrix[index+1][index+1])
                    for j in range(len(matrix[k])):
                        matrix[ind][j]=matrix[index+1][j]*mul+matrix[ind][j]
                        inverse_matrix[ind][j] =inverse_matrix[index+1][j]*mul+
inverse_matrix[ind][j]

    return inverse_matrix

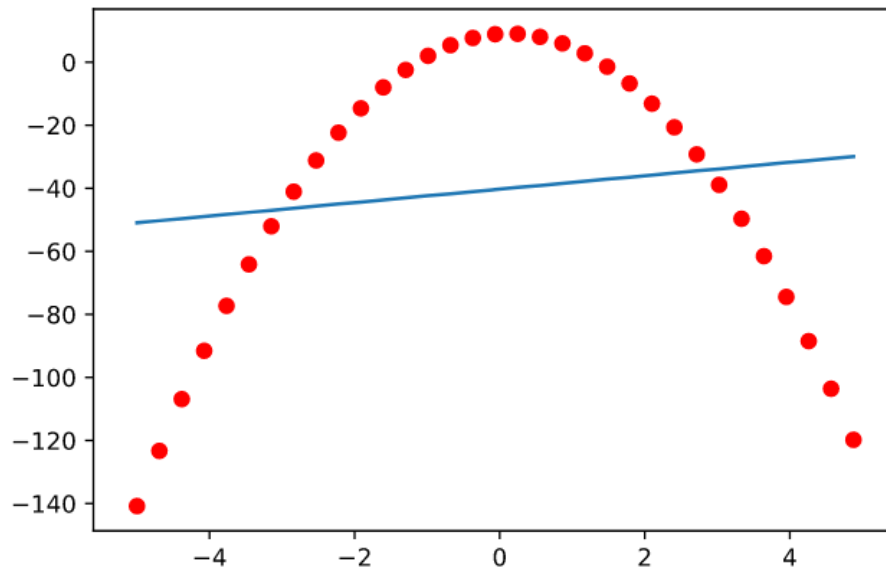
```

計算反矩陣: 利用 Gauss-Jordan elimination

Output:

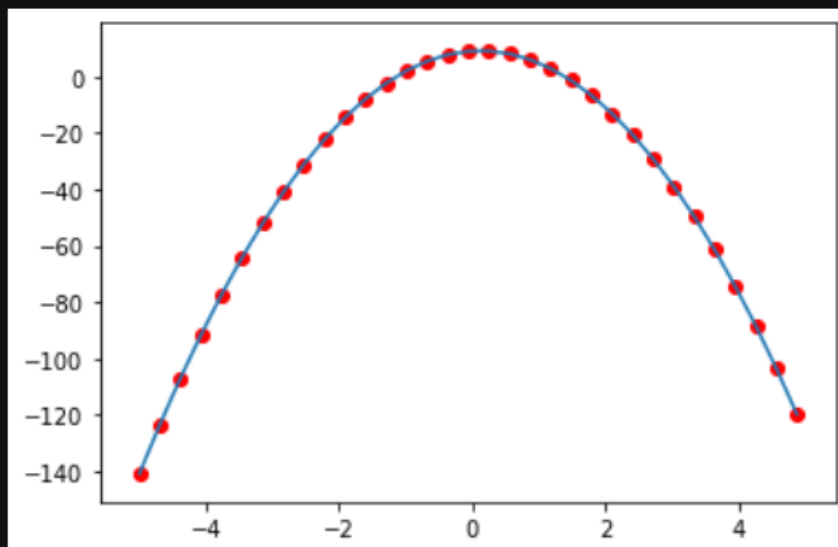
Polynomial base:2

Fitting line: $2.129097X^1 + -40.27912612124098$
Total error: 31993.922878891477



Polynomial base:3

Fitting line: $-5.706656X^2 + 1.431515X^1 + 8.999492270942895$
Total error: $1.4830585018155022e-26$



2. Logistic Regression:

先從檔案一行一行讀進資料，並利用逗號把每行的 x,y 分開，再用 2 維 list

把 x,y 資料儲存起來，再要求輸入 method 代號(L2norm 輸入 0, cross

entropy 輸入 1), 並把它儲存起來。擴增 2 維 List, 讓每航除了 x,y 之外, 再多加了 1 這個元素(for bias), w 是個 3*1 矩陣, 我初始值設 [0.5 ,0.5 ,0.5], 針對每行資料先帶進 sinmoid_fun 裡, 如果回傳值大於 0.5, label 設 1, 否則就設 0, 再設迴圈跑一萬次, 或是 error function 回傳的最大錯誤值小於 0.001 就停止, 每次迴圈利用我們之前選擇的 l2 norm 或 cross entropy 回傳回來的 w'乘上我的 learn rate0.02 更新我的 w, 最後把相關結果 output 出來。

```
def sinmoid_fun(w,x):  
    ans=0  
    for i in range(len(w)):  
        ans+=w[i]*x[i]  
    re=1/(1+math.exp(-ans))  
  
    return re
```

sinmoid function

```
def L2norm(data_arr,w):  
    y_arr=[num[-1] for num in data_arr]  
    w_diff=[0,0,0]  
    max_error=0.0  
    for i in range(len(data_arr)):  
        mx=sinmoid_fun(w,data_arr[i])  
  
        for j in range(len(w)):  
            data_arr[i][4]=abs(y_arr[i]-mx)  
  
            if data_arr[i][4]> max_error:  
                max_error=data_arr[i][4]  
  
            w_diff[j]+=(y_arr[i]-mx)*mx*(1-mx)*data_arr[i][j]  
  
    return w_diff,max_error
```

L2 norm

```

def Cross_Entropy(data_arr,w):
    y_arr=[num[-1] for num in data_arr]
    w_diff=[0.0,0.0,0.0]
    max_error=0.0

    for i in range(len(data_arr)):
        mx=sigmoid_fun(w,data_arr[i])
        #print("mx:",mx,"true",data_arr[i][-1])
        data_arr[i][4]=abs(y_arr[i]-mx)

        if data_arr[i][4] > max_error:
            max_error=data_arr[i][4]

        for j in range(len(w_diff)):
            w_diff[j]+=(y_arr[i]-mx)*data_arr[i][j]

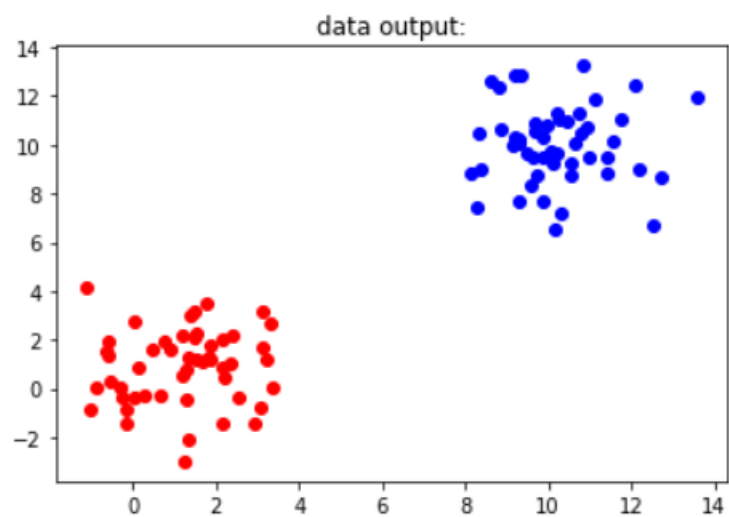
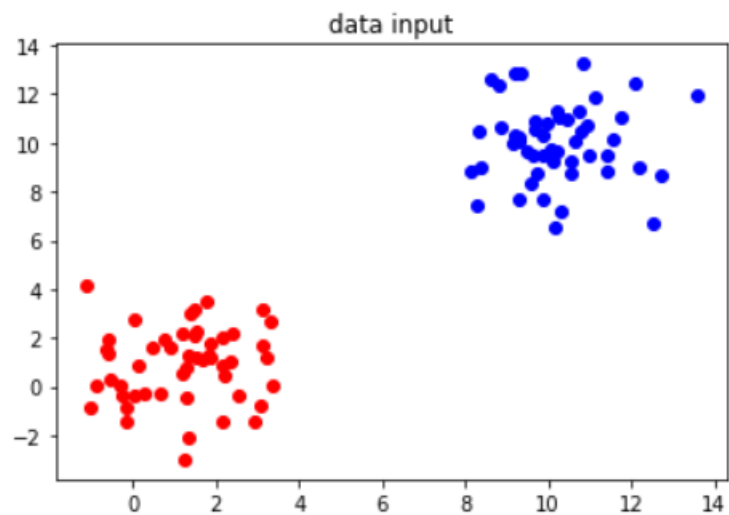
    for j in range(len(w_diff)):
        w_diff[j] /= len(data_arr)

    return w_diff,max_error

```

Cross entropy

Output:



Confusion Matrix:

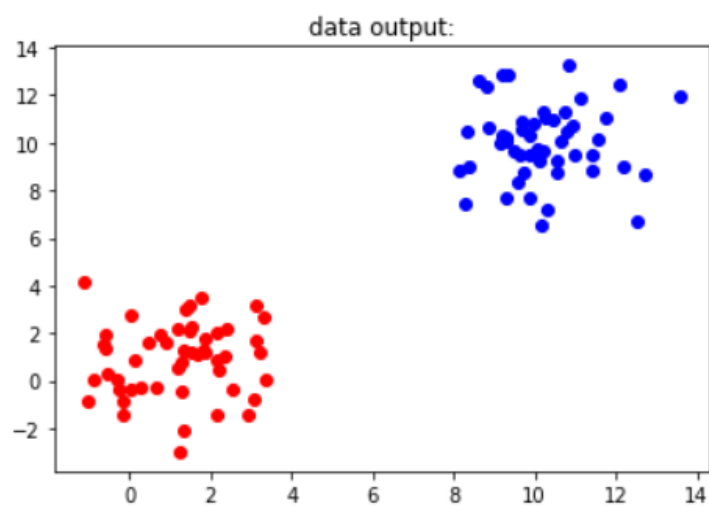
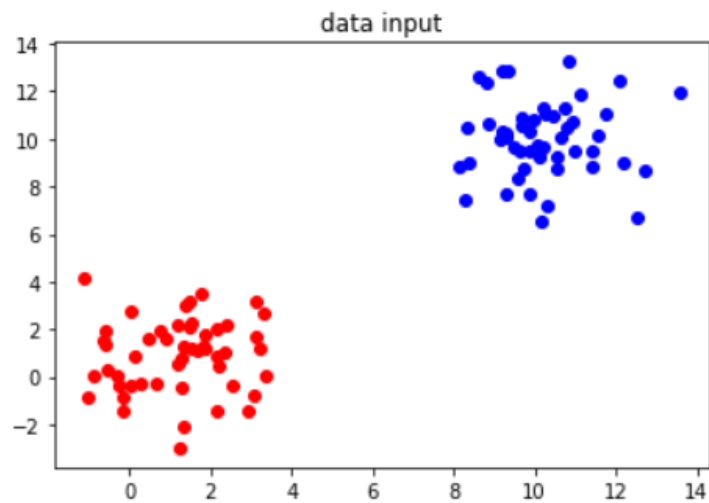
	predict cluster 1	predict cluster 2
Is cluster 1	50	0
Is cluster 2	0	50

Precision : 1.0
Recall: 1.0

The weight: 0.6116571904733943 0.550345521465671 -5.735798948740105

Data: Logistic_data1-1.txt, Logistic_data1-2.txt

method: L2norm



Confusion Matrix:

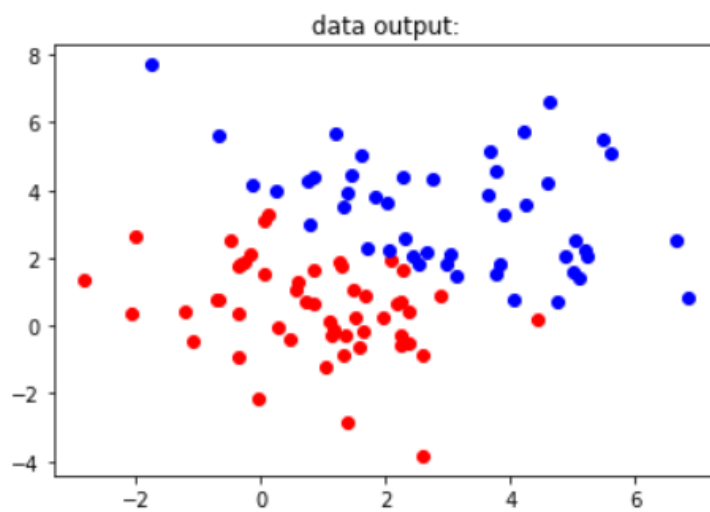
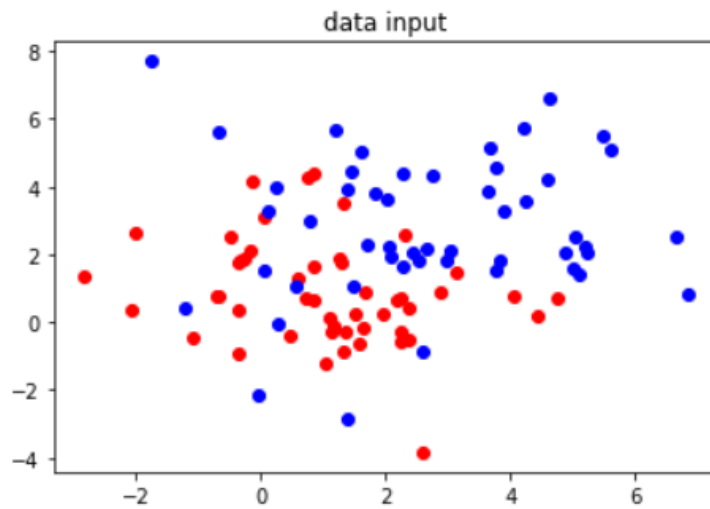
	predict cluster 1	predict cluster 2
Is cluster 1	50	0
Is cluster 2	0	50

Precision : 1.0
Recall: 1.0

The weight:[0.7570849150274537 0.5690173282928859 -7.11707202990714]

Data: Logistic_data1-1.txt, Logistic_data1-2.txt

method: Cross entropy



Confusion Matrix:

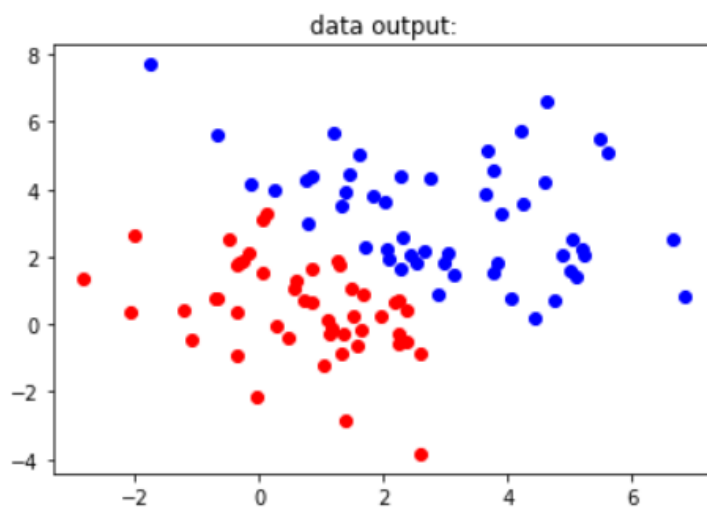
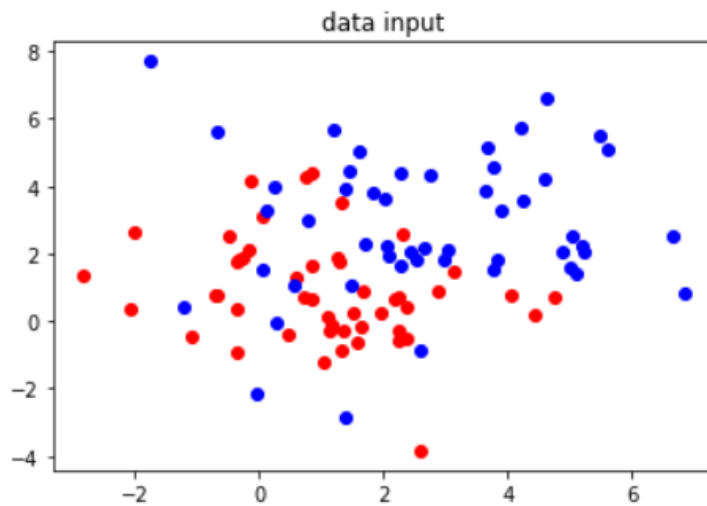
	predict cluster 1	predict cluster 2
Is cluster 1	42	11
Is cluster 2	8	39

Precision : 0.7924528301886793
Recall: 0.84

The weight:[0.6615126520861885 0.9224399823761541 -3.168636866154705]

Data: Logistic_data2-1.txt, Logistic_data2-2.txt

Method: L2norm



Confusion Matrix:

	predict cluster 1	predict cluster 2
Is cluster 1	40	9
Is cluster 2	10	41

Precision : 0.8163265306122449
Recall: 0.8

The weight:[0.5363417728825696 0.5968452199599986 -2.052517622126323]

Data: Logistic_data2-1.txt, Logistic_data2-2.txt

Method: Cross Entropy