機器學習 HW3

班級:電機三甲
姓名:林士恩
學號:B043011031

一、完整程式碼:

1.Linear regression with multiple variables

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

#func to compute the total cost error
def computeCost(X, y, theta) :    #X represents training sets
    inner = np.power(((X * theta.T) - y), 2)   #inner is also a matrix object (brocasting)
    return np.sum(inner) / (2 * X.shape[0])

def gradientDescent(theta, X, y, alpha, iters) :
    temp = np.matrix(np.zeros(theta.shape))  #numpy.zeros會回傳一個限定維度的array object
    parameters = int(theta.ravel().shape[1])  #matrix.ravel()會回傳個整個攤平的matrix(即row vector)
    cost = np.zeros(iters)  
    current = 0
    for i in range(iters):
        error = (X * theta.T) - y      #error為97 x 1
        for j in range(parameters):      #parameters即為θ的數量, 進行θ0, θ1的運算  
            term = np.multiply(error, X[:,j])    #矩陣乘法, X[:, j]為 97 x 2矩陣的column 0, 1 (bitwise product)
            temp[0,j] = theta[0,j] - ((alpha / len(X)) * np.sum(term))   #temp為 1 x 2 matrix
            
        term1 = np.multiply(error, X[:,0])
        temp[0,0] = theta[0,0] - ((alpha / len(X)) * np.sum(term1))
        term2 = np.multiply(error, X[:,1])
        temp[0,1] = theta[0,1] - ((alpha / len(X)) * np.sum(term2)) 
        term3 = np.multiply(error, X[:,2])
        temp[0,1] = theta[0,2] - ((alpha / len(X)) * np.sum(term3))         
        
        theta = temp
        cost[i] = computeCost(X, y, theta)
        current += 1
        if (cost[i - 1] - cost[i] < 10**-10) & (i > 0):
            print(cost[i - 1])
            print(cost[i])
            print("Early Stop at %d iters" % current)
            break
        
    print("θ0: %f θ1: %f" %(theta[0, 0], theta[0, 1]))    
    return theta, cost   

path = os.getcwd() + '\ex1data2.txt'
data = pd.read_csv(path, header = None, names =['Size', 'Bedrooms', 'Price'])
```

```python
#print(data)

data = (data - data.mean()) / data.std()  #DataFrame return standard deviation
mean = data.mean()       #mean is DataFrame Object
standardDeviation = data.std()
#print(data)

data.insert(0, 'Ones', 1)     #插入column insert(loc, column, value, allow_duplicates=False)
columns = data.shape[1]       #data現在為100 X 4
x = data.iloc[:, 0:(columns -1)]          #利用slice取出處理後的DataFrame
y = data.iloc[:, (columns - 1):columns]
X = np.matrix(x.values)
Y = np.matrix(y.values)
theta = np.matrix(np.array([0, 0, 0]))

alpha = 0.0001
iters = 10000

g, cost = gradientDescent(theta, X, Y, alpha, iters)

fig, ax = plt.subplots(figsize = (6, 3))
ax.plot(np.arange(iters), cost, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Error vs. Training Epoch')

```

Console:

θ0: -0.000000 θ1: 0.221931

2. Logistic Regression

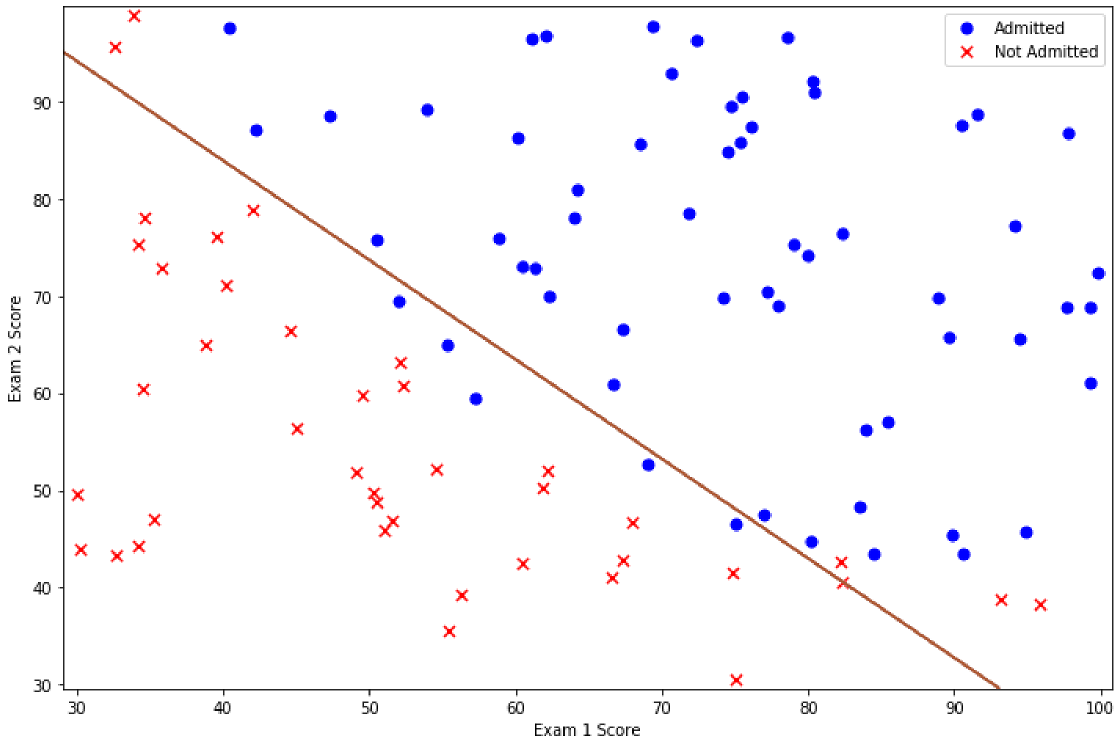
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

#define sigmoid func
def sigmoid(z) :
    return 1 / (1 + np.exp(-z))  #np.exp() Calculate the exponential of all elements in the input array

#define a func to compute the cost
def computeCost(theta, X, Y) :
    theta = np.matrix(theta)
    s1 = np.multiply(Y, np.log(sigmoid(X * theta.T)))    #elementwise product
    s2 = np.multiply((1 - Y), np.log(1 - sigmoid(X * theta.T)))
    return -(1 / len(X)) * np.sum(s1 + s2)

#define a func to compute gradient
def computeGradient(theta, X, Y) :
    theta = np.matrix(theta)
    parameters = int(theta.ravel().shape[1])
    gradient = np.zeros(theta.T.shape[0])
    error = sigmoid(X * theta.T) - Y
    
    for i in range(parameters) :
        term = np.multiply(error, X[:,i])
        gradient[i] = np.sum(term) / len(X)
        
    return gradient

#define a func to use our model to predict the result
def predict(X, theta) :
    prob = sigmoid(X * theta.T)
    return [1 if x >= 0.5 else 0 for x in prob]
        
path = os.getcwd() + '\data\ex2data1.txt'
data = pd.read_csv(path, header = None, names =['Exam 1', 'Exam 2', 'Admitted'])
#print(data)

positive = data[data['Admitted'].isin([1])]  #查看共有多少人錄取
negative = data[data['Admitted'].isin([0])]
```

```python
#劃出點圖，s代表點的scalar
#fig1, ax1 = plt.subplots(figsize=(12, 8))
#ax1.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='Admitted')  #藍色代表錄取
#ax1.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='Not Admitted') #紅色代表
#ax1.set_xlabel('Exam 1 Score')
#ax1.set_ylabel('Exam 2 Score')

#test the sigmoid func
#nums = np.arange(-10, 10, 1)
#fig2, ax2 = plt.subplots(figsize = (12, 8))
#ax2.plot(nums, sigmoid(nums), 'b')

data.insert(0, 'Ones', 1)     #插入column insert(loc, column, value, allow_duplicates=False)
data.insert(4, 'Square_1', data['Exam 1'] **3 )
#data.insert(5, 'Square_2', data['Exam 2'] **3 )
print(data)
columns = data.shape[1]       #data現在為100 X 5


x = data.iloc[:, [0, 1, 2, 4]]
y = data.iloc[:, 3:4]
#x = data.iloc[:, 0:(columns -1)]           #利用slice取出處理後的DataFrame
#y = data.iloc[:, (columns - 1):columns]
X = np.matrix(x.values)
Y = np.matrix(y.values)
theta = np.zeros(4)



result = opt.fmin_tnc(func = computeCost, x0 = theta, fprime = computeGradient, args=(X, Y))   #theta要的是arr
#print(result)       #result為一個array包含theta, iters
#print(computeCost(result[0], X, Y))

#===============================test our model==================================

theta_min = np.matrix(result[0])
predictions = predict(X, theta_min)
correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (a, b) in zip(predictions, Y)]
accuracy = (sum(map(int, correct)) % len(correct))
print("accuracy = %f" %accuracy)
print("iters : %d" %result[1])

h = 0.02 # step size in the mesh
#create a mesh to plot in
x_min, x_max = X[:, 1].min() - 1, X[:, 1].max() + 1
y_min, y_max = X[:, 2].min() - 1, X[:, 2].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = predict(np.c_[np.ones(xx.ravel().shape[0]).ravel(), xx.ravel(), yy.ravel(), (xx**3).ravel()], theta_min)
Z = np.matrix(Z).reshape(xx.shape)
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='Admitted')
ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='Not Admitted')
ax.contour(xx, yy, Z, cmap=plt.cm.Paired)
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')
```

（註：這時 hypothesis 最高有 3 次項）

Console:

```
accuracy = 89.000000
iters : 36
```



二、問題討論:

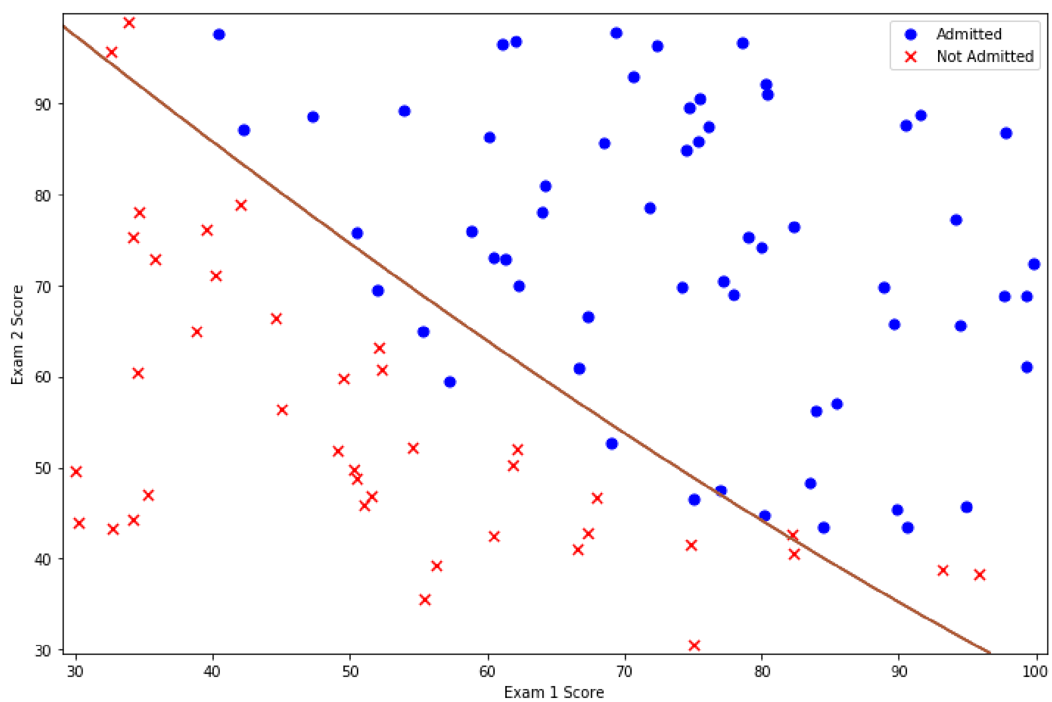
1. 持續增加模型的複雜度對於 accuracy 的影響:

若將 hypothesis 的複雜度增加，加入了兩個 feature 的二次項，如下圖

```
data.insert(4, 'Square_1', data['Exam 1'] **2 )
data.insert(5, 'Square_2', data['Exam 2'] **2 )
print(data)
columns = data.shape[1]       #data現在為100 X 5


x = data.iloc[:, [0, 1, 2, 4, 5]]
y = data.iloc[:, 3:4]
#x = data.iloc[:, 0:(columns -1)]           #利用slice取出處理後的DataFrame
#y = data.iloc[:, (columns - 1):columns]
X = np.matrix(x.values)
Y = np.matrix(y.values)
theta = np.zeros(5)
```

訓練完後可以發現正確率上升了 1%，產生出的 decision boundary 與線性的 hypothesis 沒有太大的差別

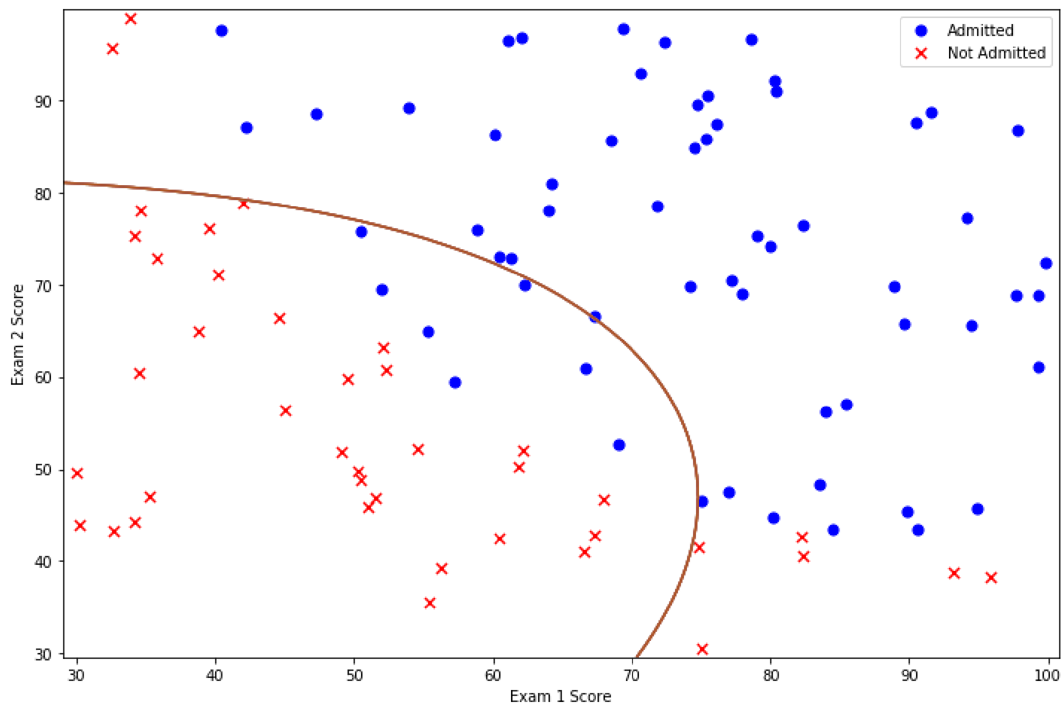
```
accuracy = 90.000000
iters : 100
accuracy = 90.000000
```



若再將兩 2 多加入的 feature 改成 3 次方時，會發現正確率反而下降，且 iters 次

數也下降到了 40 次，如下圖所示

```
accuracy = 85.000000
iters : 40
```

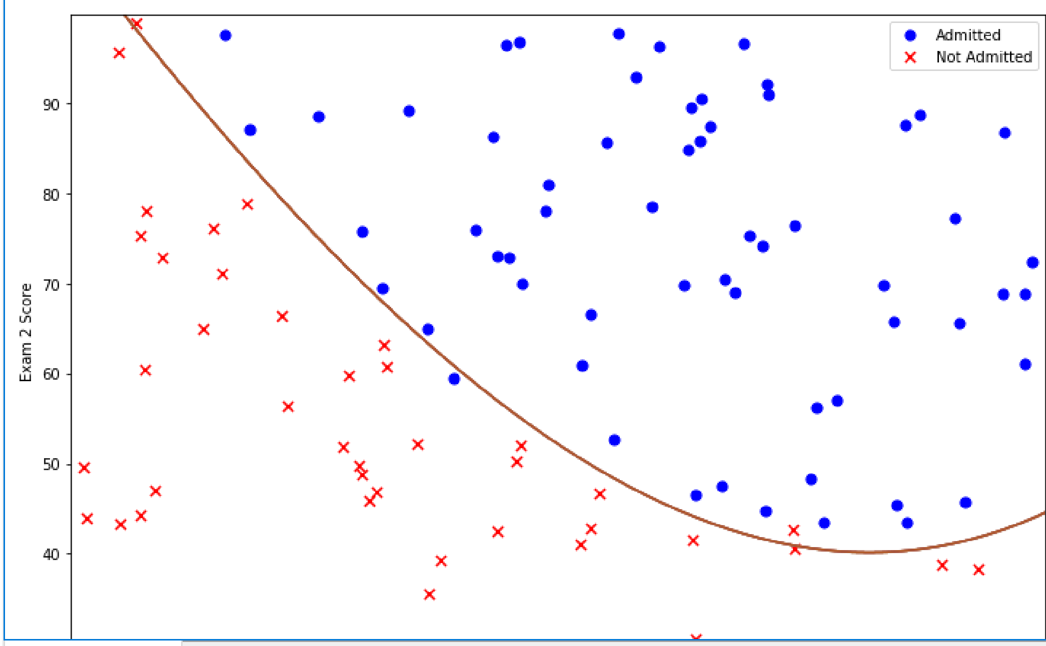


若只加入 1 個”Exam 1”的 3 次項 feature，如下圖所示

```python
data.insert(0, 'Ones', 1)     #插入column insert(loc, column, value, allow_duplicates=False)
data.insert(4, 'Square_1', data['Exam 1'] **3 )
#data.insert(5, 'Square_2', data['Exam 2'] **3 )
print(data)
columns = data.shape[1]       #data現在為100 X 5


x = data.iloc[:, [0, 1, 2, 4]]
y = data.iloc[:, 3:4]
#x = data.iloc[:, 0:(columns -1)]           #利用slice取出處理後的DataFrame
#y = data.iloc[:, (columns - 1):columns]
X = np.matrix(x.values)
Y = np.matrix(y.values)
theta = np.zeros(4)
```

最後會發現正確率提升到了 97%，iters 次數也上升到了 100 次，如下圖所示

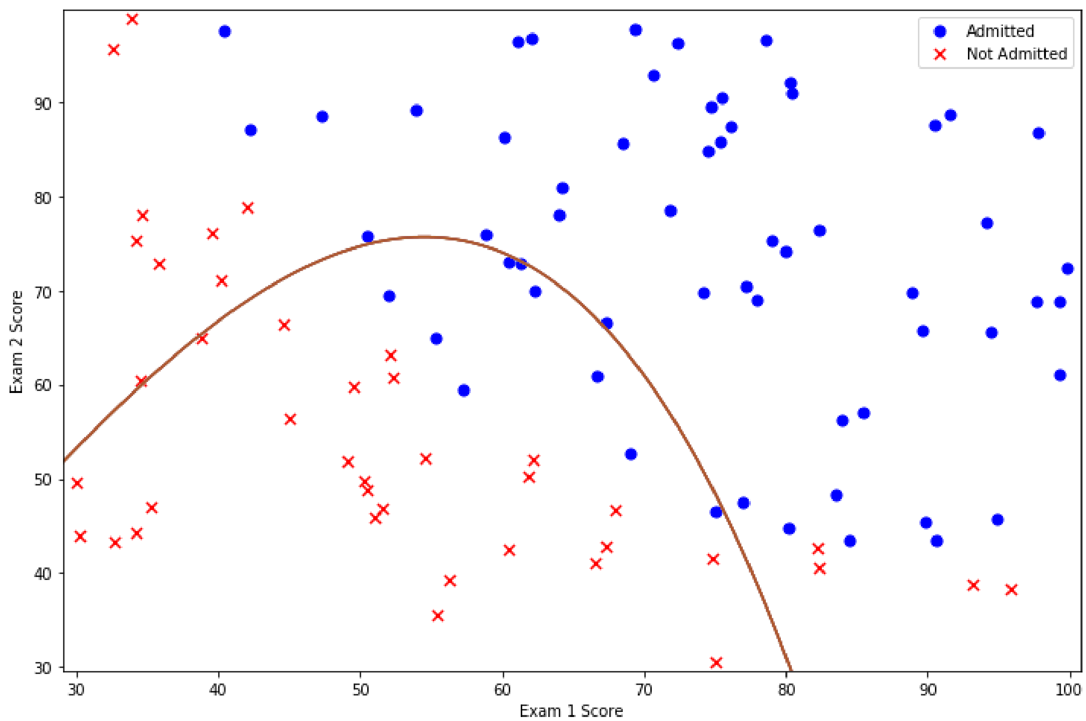
```
accuracy = 97.000000
iters : 100
C:/Users/user1/Desktop/Dick Learning_HW/HW_3_2.py:21: RuntimeWarning: divide by zero
encountered in log
  s2 = np.multiply((1 - Y), np.log(1 - sigmoid(X * theta.T)))
C:/Users/user1/Desktop/Dick Learning_HW/HW_3_2.py:21: RuntimeWarning: invalid value
encountered in multiply
  s2 = np.multiply((1 - Y), np.log(1 - sigmoid(X * theta.T)))
```



IPython console History log

下圖為有 4 次方的 feature

```
[100 rows x 5 columns]
accuracy = 78.000000
iters : 20
```



由前面幾次的觀察可以發現，對於一組所有的訓練資料，一般認為模型越複雜越能變化去近似資料，但是，增加 hypothesis 的複雜度去訓練未必是一件好事，有時候反而會降低準確度，所以再決定模型時，先觀察資料的散佈情形是很重要的，資料的散佈情況會進而決定 decision boundary 應該設在哪才會提升正確率。

也可以觀察到當 hypothesis 越複雜，所進行的 iters 也越少，因為次方數越多，theta 只要走一點點就可以使整個 hypothesis 的值變大很多，即越容易收斂。

2. 為什麼不直接以 accuracy 當作是 cost function:

不利用 accuracy 來當作 cost function 的原因在於，目前選定 cost function 後，藉由資料的測試可以得到 accuracy(正確率)，就像這次的範例，是用訓練時的資料再去驗證正確率，這個 decision boundary 就是針對現在的 cost function 所畫出的最佳解，所以用訓練時的資料去測試的話一定是最高的正確率(最佳解)，但是如果今天再用這個模型沒有訓練過的資料去測試，accuracy 就可能會下降或上升，代表 cost function 一樣可以求得最佳解，若今天只用 accuracy 當作 cost function 就代表失去了自己求的最佳解的功能，只要給的資料先前沒有看過，就可能沒辦法找到最佳解。所以 accuary 只適合用來當作評斷一個模組在某些場合是否有效，並

不適合直接當作 cost function。