機器學習 HW4

班級:電機三甲

姓名: 林士恩

學號: B043011031

一、原始程式碼:

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

def predict(X, theta) :
    prob = sigmoid(X * theta.T)
    return [1 if x >= 0.5 else 0 for x in prob]      #串列生成

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

def costReg(theta, X, Y, lambda1, reg_on = True):
    theta = np.matrix(theta)
    X = np.matrix(X)
    Y = np.matrix(Y)
    seg1 = np.multiply(Y, np.log(sigmoid(X * theta.T)))   #elementwise product
    seg2 = np.multiply((1 - Y), np.log(1 - sigmoid(X * theta.T)))
    if(reg_on == True) :
        reg = (lambda1 /(2 * len(X))) * np.sum(np.power(theta, 2))
        return np.sum(- (seg1 + seg2)) / len(X) + reg
    else :
        return np.sum(- (seg1 + seg2)) / len(X)

def gradientReg(theta, X, Y, lambda1):     #算gradient
    theta = np.matrix(theta)
    X = np.matrix(X)
    Y = np.matrix(Y)
    parameters = int(theta.ravel().shape[1])
    gradient = np.zeros(parameters)        #用來存新的theta
    error = sigmoid(X * theta.T) - Y
    
    for i in range(parameters) :
        term = np.multiply(error, X[:, i])
        if(i == 0) :
            gradient[i] = sum(term) / len(X)
        else :
```

```python
            gradient[i] =  sum(term) / len(X) + (lambda1 / len(X)) * theta[:, i]
    return gradient
    
    

path = os.getcwd() + '\exercise4-data\ex2data2.txt'
data = pd.read_csv(path, header = None, names =['Test 1', 'Test 2', 'Accepted'])
#print(data)

positive = data[data['Accepted'].isin([1])]  #查看共有多少產品錄取
negative = data[data['Accepted'].isin([0])]
#劃出點圖，s代表點的scalar
#fig1, ax1 = plt.subplots(figsize=(12, 8))
#ax1.scatter(positive['Test 1'], positive['Test 2'], s=50, c='b', marker='o', label='Accepted')  #藍色代表錄取
#ax1.scatter(negative['Test 1'], negative['Test 2'], s=50, c='r', marker='x', label='Rejected') #紅色代表不錄取
#ax1.set_xlabel('Test 1 Score')
#ax1.set_ylabel('Test 2 Score')

degree = 6
x1 = data['Test 1']
x2 = data['Test 2']
data.insert(3 ,'Ones', 1)      #加入x0項

for i in range(1, degree + 1):
    for j in range(0, i + 1):
        data['Feat.' + str(i - j) + str(j)] = np.power(data['Test 1'], i - j) * np.power(data['Test 2'], j)

data.drop('Test 1', axis = 1, inplace = True)
data.drop('Test 2', axis = 1, inplace = True)

cols = data.shape[1]
X = data.iloc[:, 1 : cols]    #取column 1 ~ 28
Y = data.iloc[:, 0 : 1]

theta = np.zeros(cols - 1)
X = np.array(X.values)
Y = np.array(Y.values)
lambda1 = 1
```

```python
data = data.sample(frac = 1)
X2 = data.iloc[:, 1 : cols]    #取column 1 ~ 28
Y2 = data.iloc[:, 0 : 1]
train_num = int(data.shape[0] * 0.7)
val_num = int(data.shape[0] * 0.2)
X_train = X2.iloc[: train_num, :]        #用來做訓練
Y_train = Y2.iloc[: train_num, :]
X_valid = X2.iloc[train_num : train_num + val_num, :]   #用來測量效用
Y_valid = Y2.iloc[train_num : train_num + val_num, :]
X_test = X2.iloc[train_num + val_num :, :]             #用來測試真實情況
Y_test = Y2.iloc[train_num + val_num :, :]


lambdaArray = np.arange(0, 10, 1)
cost = np.zeros(lambdaArray.shape[0])
theta_all = list()

for i in range(lambdaArray.shape[0]):
    theta = np.zeros(X_train.shape[1])
    result = opt.fmin_tnc(func = costReg, x0 = theta, fprime = gradientReg, args=(X_train, Y_train, lambdaArray[i]), )
    theta_min = np.matrix(result[0])
    cost[i] = costReg(theta_min, X_valid, Y_valid, lambdaArray[i], reg_on = False)     #不需要再做regulation的cost運算
    theta_all.append(theta_min.ravel())

fig, ax = plt.subplots(figsize = (9, 8))
ax.set_xlabel("Lambda")
ax.set_ylabel("Cost")
ax.plot(lambdaArray, cost)
index_min = cost.argmin()
lambda1 = lambdaArray[index_min]

```

```python
result = opt.fmin_tnc(func = costReg, x0 = theta, fprime = gradientReg, args=(X, Y, 1), )   #theta要的是array，不然
print(result)
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)) * 100
print("accuracy with training data and lambda: %f = %f" %(1, accuracy))

###################################畫圖##############################

h = 0.02
xx, yy = np.meshgrid(np.arange(-1, 1.5, h), np.arange(-1, 1.5, h)) #Return coordinate matrices from coordinate vect
X2_plot = np.ones(xx.ravel().shape[0]).ravel()        #Return a new array of given shape and type, filled with ones
                                                      #X2_plot用來儲存decision boundary
for i in range(1, degree + 1):
    for j in range(0, i + 1):
        term = np.power(xx.ravel(), i - j) * np.power(yy.ravel(), j)
        X2_plot = np.c_[X2_plot, term.ravel()]  #Translates slice objects to concatenation along the second axis.

Z = np.matrix(predict(np.matrix(X2_plot), theta_min)).reshape(xx.shape)
fig2, ax2 = plt.subplots(figsize = (12, 9))
ax2.scatter(positive['Test 1'], positive['Test 2'], s = 50, c = 'b', marker = 'o', label = "Accepted")
ax2.scatter(negative['Test 1'], negative['Test 2'], s = 50, c = 'r', marker = 'x', label = "Rejected")
ax2.contour(xx, yy, Z, cmap = plt.cm.Paired)
ax2.legend()          #加上右上角的標註
ax2.set_xlabel("Test1 Score")
ax2.set_ylabel("Test2 Score")


theta_optimun = theta_all[index_min]
X_test = np.matrix(X_test)
predictions = predict(X_test, theta_optimun)
Y_test = np.array(Y_test)
correct2 = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (a, b) in zip(predictions, Y_test)]
accuracy = (sum(map(int, correct2)) / len(correct2)) * 100
print("Real accuracy with lambda: %f = %f"%(lambda1 , accuracy))

```

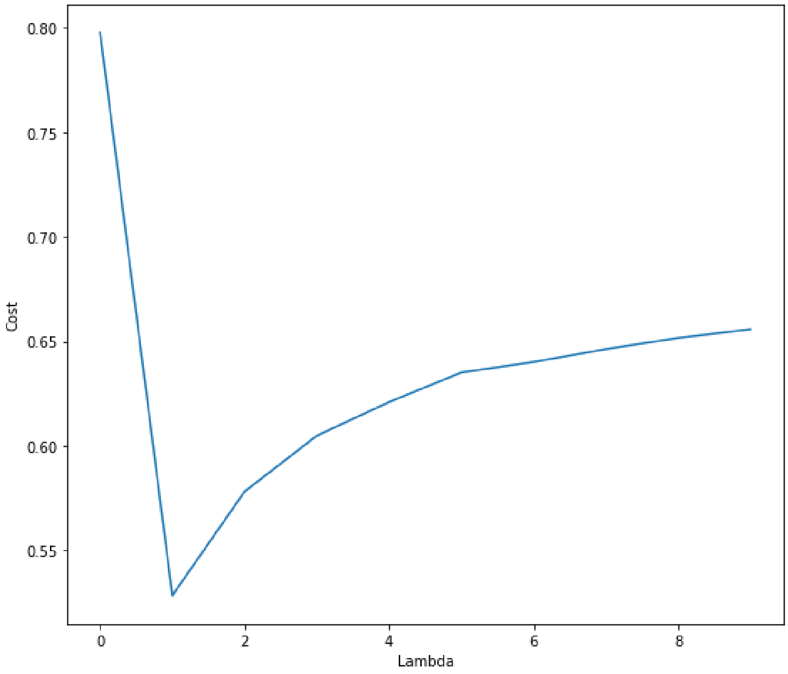
```python
###################################畫圖##############################

h = 0.02
xx, yy = np.meshgrid(np.arange(-1, 1.5, h), np.arange(-1, 1.5, h)) #Return coordinate matrices from coordinate vectors
X3_plot = np.ones(xx.ravel().shape[0]).ravel()        #Return a new array of given shape and type, filled with ones.
                                                      #X2_plot用來儲存decision boundary
for i in range(1, degree + 1):
    for j in range(0, i + 1):
        term = np.power(xx.ravel(), i - j) * np.power(yy.ravel(), j)
        X3_plot = np.c_[X3_plot, term.ravel()]  #Translates slice objects to concatenation along the second axis.

Z2 = np.matrix(predict(np.matrix(X3_plot), theta_optimun)).reshape(xx.shape)
fig3, ax3 = plt.subplots(figsize = (12, 9))
ax3.scatter(positive['Test 1'], positive['Test 2'], s = 50, c = 'b', marker = 'o', label = "Accepted")
ax3.scatter(negative['Test 1'], negative['Test 2'], s = 50, c = 'r', marker = 'x', label = "Rejected")
ax3.contour(xx, yy, Z2, cmap = plt.cm.Paired)
ax3.legend()          #加上右上角的標註
ax3.set_xlabel("Test1 Score")
ax3.set_ylabel("Test2 Score")
```

Console:

Regulation的作法:

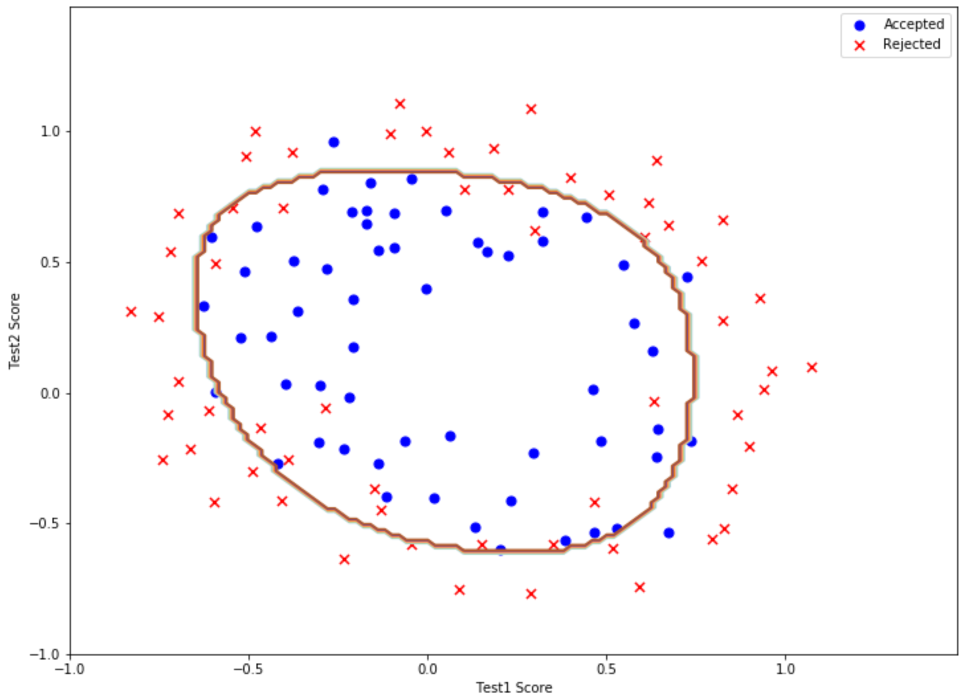
產生許多不同lambda值，並針對所以lambda進行gradient descent，並比較每個lambda解出的theta所對應出的cost(不需再加上regulation項)，即可找到optimal lambda，同時也解決了overfitting的問題

```
accuracy with training data and lambda: 1.000000 = 83.050847
Real accuracy with lambda: 1.000000 = 53.846154
```




```python
for i in range(lambdaArray.shape[0]):
    theta = np.zeros(X_train.shape[1])
    result = opt.fmin_tnc(func = costReg, x0 = theta, fprime = gradientReg, args=(X_train, Y_train, lambdaArray[i]), )
    theta_min = np.matrix(result[0])
    cost[i] = costReg(theta_min, X_valid, Y_valid, lambdaArray[i], reg_on = False)     #不需要再做regulation的cost運算
    theta_all.append(theta_min.ravel())

```

Lambda = 1時的decision boundary



二、問題與討論:

1.把資料分群的影響、Lambda 的選用:

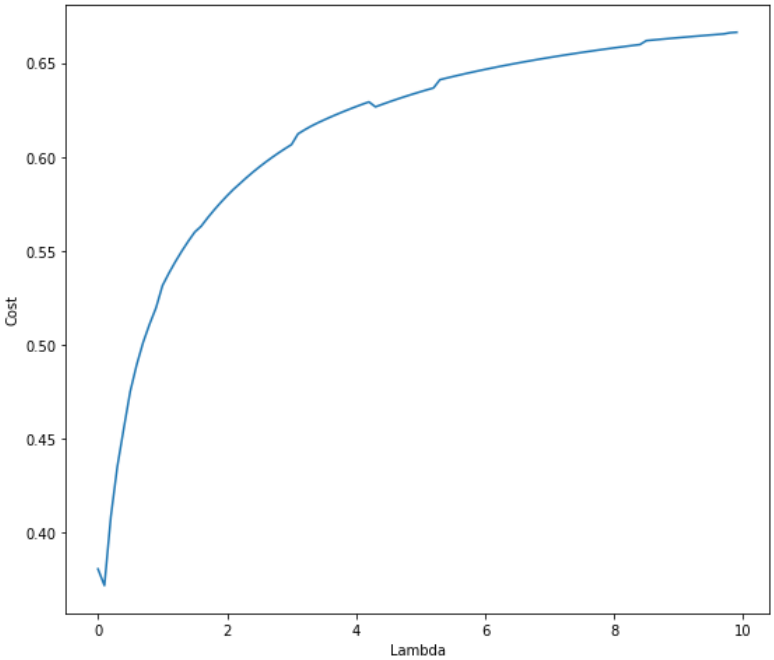
若將全部資料打亂，並分成三種:訓練資料、比較用的資料、測試模型的資料，並嘗試多個 lambda 值，去找出最少 cost 的 lambda 值以及對應的 theta，下三張圖即顯示某隨機的訓練資料對應到的 lambda 以及 cost 關係，也可以發現每次測試的結果都不同（因為訓練資料是隨機的，代表最佳解也會變動）。
但是這次的資料不夠多，所以可能會造成實際正確率與本來的正確率相差甚多（lambda 為 0.0 ~ 1.0）

```
data = data.sample(frac = 1)
X2 = data.iloc[:, 1 : cols]    #取column 1 ~ 28
Y2 = data.iloc[:, 0 : 1]
train_num = int(data.shape[0] * 0.7)
val_num = int(data.shape[0] * 0.2)
X_train = X2.iloc[: train_num, :]        #用來做訓練
Y_train = Y2.iloc[: train_num, :]
X_valid = X2.iloc[train_num : train_num + val_num, :]   #用來測量效用
Y_valid = Y2.iloc[train_num : train_num + val_num, :]
X_test = X2.iloc[train_num + val_num :, :]              #用來測試真實情況
Y_test = Y2.iloc[train_num + val_num :, :]
```

	Feat.14	Feat.05	Feat.60	Feat.51	Feat.42
39	6.593500e-03	-1.180628e-02	1.474191e-04	-2.639676e-04	4.726587e-04
97	1.828187e-01	3.794792e-01	3.908597e-03	8.113129e-03	1.684053e-02
78	1.009075e-01	-2.702217e-01	5.640054e-04	-1.510359e-03	4.044613e-03
93	4.065234e-02	6.550082e-01	3.439766e-08	5.542300e-07	8.929996e-06
86	-5.355994e-03	2.058197e-03	1.854825e-01	-7.127706e-02	2.739030e-02
101	1.077340e-04	1.007523e-05	1.508320e+00	1.410573e-01	1.319160e-02
108	-1.305404e-05	-1.452010e-06	5.212125e-02	5.797485e-03	6.448585e-04
43	1.097638e-08	2.963681e-10	9.505233e-03	2.566465e-04	6.929594e-06
50	1.507949e-02	6.301358e-02	6.808256e-06	2.845007e-05	1.188861e-04
36	-4.982022e-04	-4.507073e-04	1.760921e-04	1.593048e-04	1.441179e-04
116	-6.317918e-03	9.963553e-01	6.472253e-14	-1.020695e-11	1.609667e-09
24	-2.874016e-03	-9.853026e-03	2.408803e-06	8.258129e-06	2.831144e-05
23	-2.327289e-03	-1.506979e-03	5.574266e-03	3.609479e-03	2.337229e-03
26	3.841235e-02	-4.416666e-02	1.024169e-02	-1.177593e-02	1.354000e-02
51	-1.619687e-04	1.022081e-02	6.472253e-14	-4.084228e-12	2.577297e-10
114	-3.562096e-02	2.968243e-02	4.387691e-02	-3.656200e-02	3.046659e-02
34	-3.450028e-03	2.962631e-03	2.306126e-03	-1.980332e-03	1.700564e-03
14	3.089336e-02	2.755401e-02	2.668725e-02	2.380255e-02	2.122966e-02
44	3.972067e-04	1.004446e-04	6.093288e-02	1.540855e-02	3.896474e-03
104	-2.748088e-03	-6.720616e-03	1.155103e-05	2.824873e-05	6.908398e-05
30	-2.281110e-01	8.269895e-01	3.506532e-04	-1.271252e-03	4.608772e-03
81	-5.364723e-03	-1.806653e-02	5.549899e-06	1.869014e-05	6.294192e-05
52	-8.695759e-03	5.188464e-02	6.362953e-07	-3.796558e-06	2.265277e-05
79	2.790602e-02	-2.455160e-01	3.997646e-07	-3.517112e-06	3.094341e-05
64	4.869068e-02	3.194040e-02	2.012989e-01	1.320492e-01	8.662243e-02
82	-1.174919e-02	-1.180628e-02	4.719600e-03	4.742532e-03	4.765575e-03
63	7.687077e-02	7.597518e-02	4.867916e-02	4.811202e-02	4.755148e-02

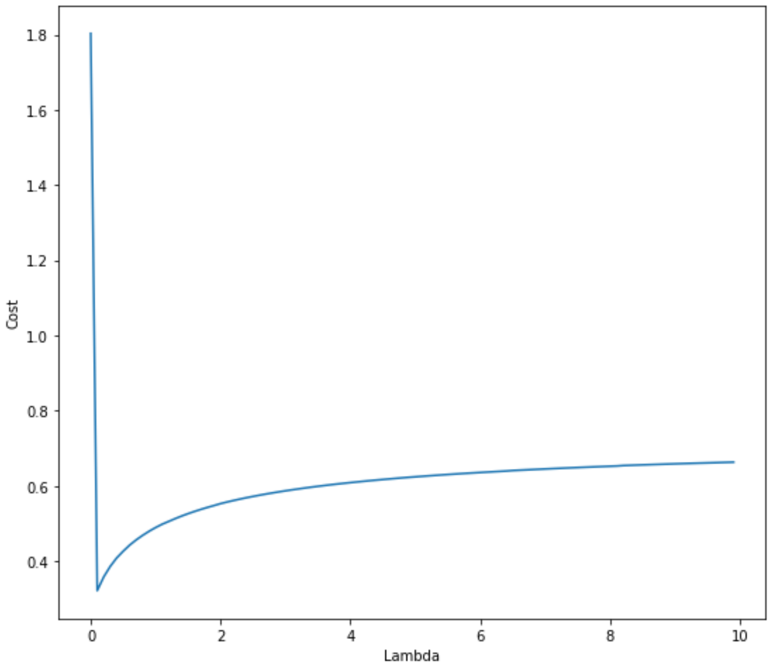
（資料打亂，row 的順序不固定）

```
accuracy with training data and lambda: 0.100000 = 83.898305
Real accuracy with lambda: 0.100000 = 92.307692
```



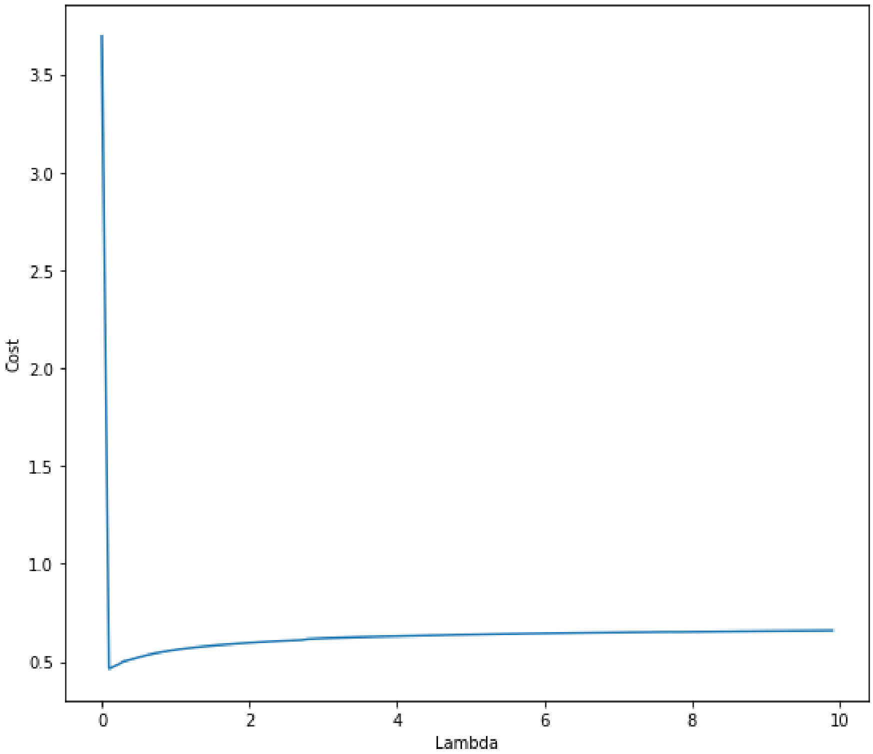
上圖可以發現實際的正確率較本來的正確率大了許多(有可能是因為測試資料不夠多)。

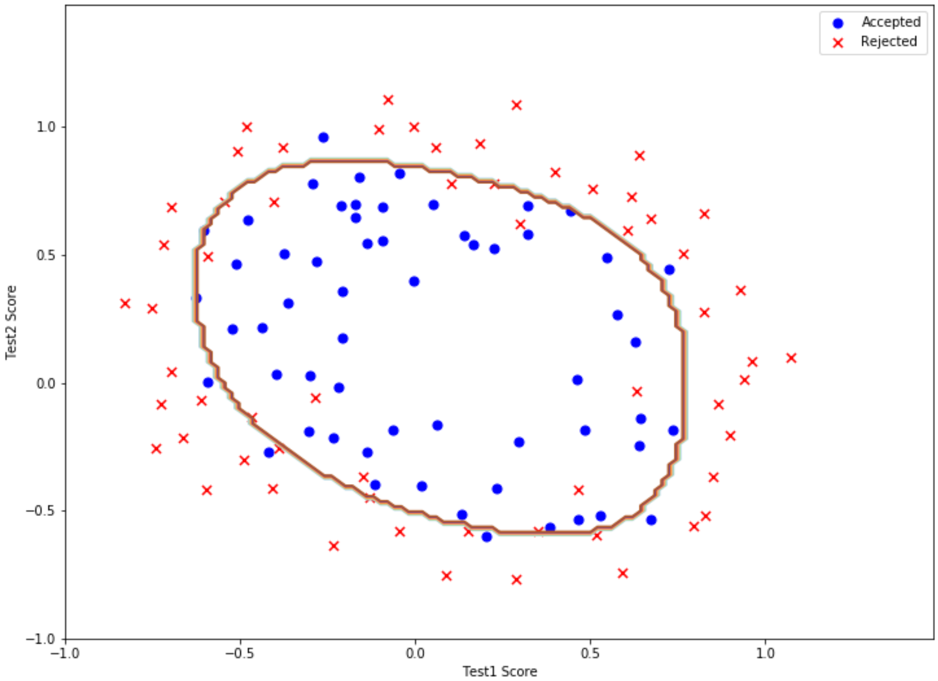
```
accuracy with training data and lambda: 0.100000 = 83.898305
Real accuracy with lambda: 0.100000 = 69.230769
```

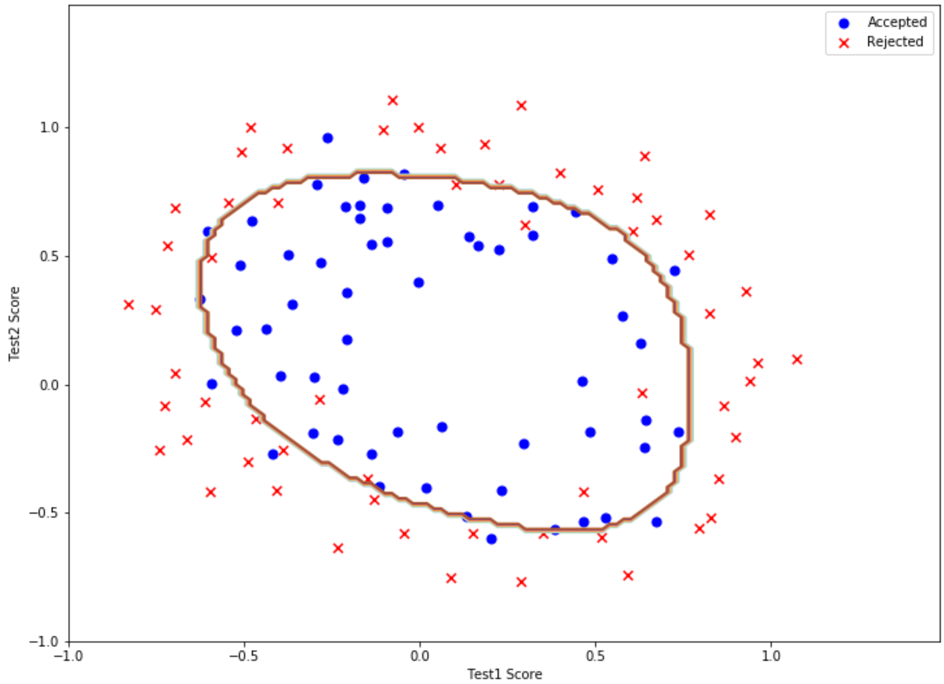


上圖可以觀察到實際的正確率較本來的正確率下降許多(有可能是因為測試資料不夠多)。

```
accuracy with training data and lambda: 0.100000 = 83.898305
Real accuracy with lambda: 0.100000 = 84.615385
```







由上面 3 張圖發現實際的正確率跟原本的正確率也可能差不多(有可能是因為測試資料不夠多)。

而且同一區間之下，大部分的 lambda 都相同

所以我們可以得知，當用所有的訓練資料所測試出的正確率並不可靠，因為在碰到陌生的資料後效能可能就會有所下降或上升。

對於每一次 lambda 都不一樣的情況下，到底要選取哪一個 lambda:

1. 可以做多種測試，並將所有的 lambda 做平均(類似期望值)，就可以包含所有可能會出現 lambda 值，再將此 lambda 值帶入 cost func，就可達到解決 overfitting 以及做平均 lambda 的功用。

如下圖所示

```
lambdaArray = np.arange(0, 10, 0.1)
cost = np.zeros(lambdaArray.shape[0])
theta_all = list()
theta_optimal_list = list()

for iters in range(100) :
    for i in range(lambdaArray.shape[0]):
        theta = np.zeros(X_train.shape[1])
        result = opt.fmin_tnc(func = costReg, x0 = theta, fprime = gradientReg, args=(X_train, Y_train, lambdaArray[i]), )
        theta_min = np.matrix(result[0])
        cost[i] = costReg(theta_min, X_valid, Y_valid, lambdaArray[i], reg_on = False)     #不需要再做regulation的cost運算
        theta_all.append(theta_min.ravel())
        
    theta_optimal_list.append(lambdaArray[cost.argmin()])

lambda_average = np.array(theta_optimal_list).average()
```