Assignment 2 - Logistic Regression

> **FULL MARKS = 10**

In this assignment, you are going to implement your own logistic Regression function. Please notice **no** library versions of logistic regression are allowed. Follow the instructions, you will need to fill the blanks to make it functional. The process is similar to the previous assignment.

Initialization

No more library allowed

In [1]:
```
# load required library
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
import scipy.optimize as opt
```

Load data *(Do not modify)*

We use 100 samples and 2 features.

In [2]:
```
x, y = load_iris(return_X_y=True)
```

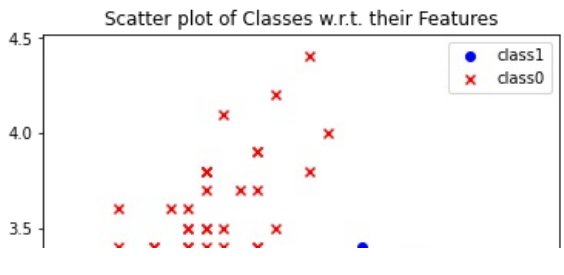
In [3]:
```
x=x[:100, :2] # class 0 and 1 balanced
y=y[:100]
```

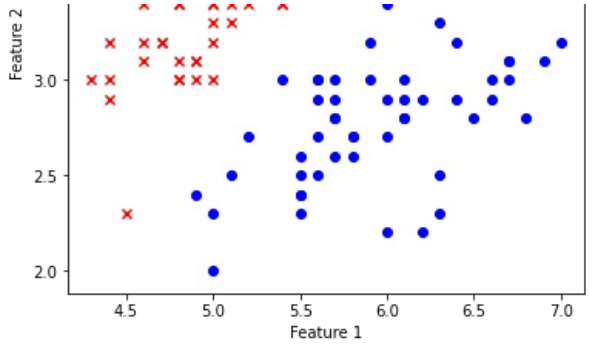
Visualize data

In [4]:
```
# draw raw data
def draw_data(x,y):

    ##########################################################################
    # Full Mark: 1                                                           #
    # TODO:                                                                  #
    # 1. make a scatter plot of the raw data                                 #
    # 2. set title for the plot                                              #
    # 3. set label for x,y axis                                              #
    # Note, this scatter plot has two different type of points               #
    ##########################################################################

    # split data into classes
    stacked = np.column_stack((x,y))
    class0 = stacked[stacked[:,-1]==0, :]
    class1 = stacked[stacked[:,-1]==1, :]

    # scatter plot both the classes
    plt.figure(figsize=(6,6))
    plt.scatter(class1[:,0],class1[:,1], c='blue', label = 'class1')
    plt.scatter(class0[:,0],class0[:,1], c='red', marker='x', label = 'class0')

    # label the plot
    plt.title('Scatter plot of Classes w.r.t. their Features')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()

    ##########################################################################
    #                        END OF YOUR CODE                                #
    ##########################################################################

    # show plot
    plt.show()
```

Your plot should be similar to the example below

In [5]:
```
draw_data(x,y)
```





Sigmoid function

In [6]:
```
# please do not modify this cell
x = np.concatenate((np.array([np.ones(len(y))]).T, x), axis=1)
theta = np.zeros(x.shape[1])
```

You can decide by yourselves whether to split the dataset to training and testing. The training and testing datasets spliting is not a part of the assignment 2 as you have done that in assignmnet 1. Without spliting, you can just report the accuracy on training dataset.

In [7]:
```
# define sigmoid function
# math: refer to https://en.wikipedia.org/wiki/Sigmoid_function or slides
def sigmoid(theta, X):
    ###########################################################################
    # Full Mark: 1                                                            #
    # TODO:                                                                   #
    # 1. implement the sigmoid function over input theta and X                #
    ###########################################################################

    # sigmoid activation = 1 / ( 1 + e^-z )
    s = 1 / (1 + np.exp(-np.dot(X, theta)))

    ###########################################################################
    #                       END OF YOUR CODE                                  #
    ###########################################################################

    return s
```

Cost function

In [8]:
```
# define cost function with sigmoid function
def cost(theta, X, y):
    ###########################################################################
    # Full Mark: 2                                                            #
    # TODO:                                                                   #
    # 1. implement the cross entropy loss function with sigmoid               #
    ###########################################################################

    # sigmoid
    sig = sigmoid(theta, x)

    # cross entropy loss = -Σ( y * log(sigmoid) + 1-y * log(1-sigmoid) ) / N
    co = -(1/x.shape[0]) * np.sum(y*np.log(sig + 1e-15) + (1 - y)*np.log(1 - sig + 1e-15))

    # added 1e-15 to avoid log(0) error

    ###########################################################################
    #                       END OF YOUR CODE                                  #
    ###########################################################################
    return co
```

Calculate gradients

In [9]:
```
# the gradient of the cost is a vector of the same length as θ where the jth element (for j = 0, 1, . . . , n)
def gradient(theta, X, y):
    ###########################################################################
    # Full Mark: 2                                                            #
    # TODO:                                                                   #
    # 1. calculate the gradients using theta and sigmoid                      #
    # Hint: X may need to be transposed to do matrix operation                #
    ###########################################################################

    # gradient descent
    grad = (1 / x.shape[0]) * np.dot(X.T, sigmoid(theta,X) - y)
```

```
    ###########################################################################
    #                       END OF YOUR CODE                                  #
    ###########################################################################
    return grad
```

Predicting

In [10]:
```
# predict for new X
def predict(theta, X):
    ###########################################################################
    # Full Mark: 1                                                            #
    # TODO:                                                                   #
    # 1. predict the value using theta and sigmoid                            #
    # 2. convert the predicted value to 0/1                                   #
    # That's how it is called Logistic regression                             #
    ###########################################################################

    # sort predicted labels into 0/1 classes
    predict_labels = sigmoid(theta, X)
    predict_labels[predict_labels < 0.5] = 0
    predict_labels[predict_labels >= 0.5] = 1

    ###########################################################################
    #                       END OF YOUR CODE                                  #
    ###########################################################################

    return predict_labels
```

Calculate accuracy

In [11]:
```
# calculate accuracy
def accurate(predictions, y):
    ###########################################################################
    # Full Mark: 1                                                            #
    # TODO:                                                                   #
    # 1. calculate the accuracy value                                         #
    # Note that you coud not import extra library                             #
    ###########################################################################

    # find mean accuracy
    accuracy_score = np.mean(predictions == y)

    ###########################################################################
    #                       END OF YOUR CODE                                  #
    ###########################################################################
    return accuracy_score
```

Calling functions

In [12]:
```
# please do not modify this cell
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(x, y))
final_theta = result[0]
final_cost = cost(final_theta, x, y)
predictions = predict(final_theta, x)
accuracy = accurate(predictions, y)
print("final cost is " + str(final_cost))
print("accuracy is " + str(accuracy))
```

```
final cost is 6.434551869277412e-07
accuracy is 1.0
```

Decsion boudary

In [13]:
```
# draw decision boudary
def draw_decision_boudary(final_theta,x,y):
    ###########################################################################
    # Full Mark: 2                                                            #
    # TODO:                                                                   #
    # 1. plot the decision boudary on the raw data                            #
    # 2. set title for the plot                                               #
    # 3. set label for x,y axis                                               #
    # Note, this scatter plot has two different type of points                #
    ###########################################################################

    # split data into classes
    stacked = np.column_stack((x,y))
    class0 = stacked[stacked[:,-1]==0, :]
    class1 = stacked[stacked[:,-1]==1, :]
```

```
    # scatter plot both the classes
    plt.figure(figsize=(6,6))
    plt.scatter(class1[:,1],class1[:,2], c='blue', label = 'class1')
    plt.scatter(class0[:,1],class0[:,2], c='red', marker='x', label = 'class0')

    # plot the decision boundary
    x_values = [np.min(x[:,1])-0.5, np.max(x[:,2])+2.5]
    y_values = - (final_theta[0] + np.dot(final_theta[1], x_values)) / final_theta[2]
    plt.plot(x_values, y_values, label = 'Decision Boundary')

    # label the plot
    plt.title('Scatter plot of Classes w.r.t. their Features + Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()

    ###########################################################################
    #                         END OF YOUR CODE                                #
    ###########################################################################

    # show plot
    plt.show()
```

Your plot should be similar to the example below

In [14]:
```
draw_decision_boudary(final_theta,x,y)
```