Gradient Descent and Hessian Matrix:

$$l(\theta) = \sum_{i=1}^{M} \left(y_i \log \sigma(\theta^T x_i) + (1 - y_i) \log(1 - \sigma(\theta^T x_i)) \right)$$

$$\sigma(z) = (1 + e^{-z})^{-1}$$

$$\log \sigma(\theta^T x_i) = \log \frac{1}{1+e^{-\theta x_i}} = -\log(1+e^{-\theta x_i})$$

$$\log(1 - \sigma(\theta^T x_i)) = \log\left(1 - \frac{1}{1+e^{-\theta x_i}}\right) = \log(e^{-\theta x_i}) - \log(1+e^{-\theta x_i})$$

$$= -\theta x_i - \log(1+e^{-\theta x_i})$$

$$\left[1 = \frac{1+e^{-\theta x_i}}{1+e^{-\theta x_i}} \; ; \; \log(x/y) = \log x - \log y \right]$$

$$\Rightarrow l(\theta) = \sum_{i=1}^{M} \left[y_i (\log(1+e^{-\theta x_i})) + (1 - y_i)(-\theta x_i - \log(1+e^{-\theta x_i})) \right]$$

$$= \sum_{i=1}^{M} \left[y_i \left(\cancel{\log(1+e^{-\theta x_i})} \right) + y_i \theta x_i - \theta x_i - \log(1+e^{-\theta x_i}) + y_i \cancel{\log(1+e} \right.$$

$$= \sum_{i=1}^{M} \left[y_i \theta x_i - \theta x_i - \log(1+e^{-\theta x_i}) \right]$$

Now $\left\{ \left[-\theta x_i - \log(1+e^{-\theta x_i}) \right] = -\left[\theta x_i - \log(1+e^{-\theta x_i}) \right] \right\} = -\log(1+e^{\theta x_i})$

$$\Rightarrow l(\theta) = \sum_{i=1}^{M} \left[y_i \theta x_i - \log(1+e^{\theta x_i}) \right]$$

Partial derivative WRT θ

Gradient: $\Rightarrow \dfrac{\partial l}{\partial \theta} = y_i x_i - \dfrac{x_i e^{\theta x_i}}{1+e^{\theta x_i}}$

$\Rightarrow$ £

$$\underline{\text{Hessian}}: \frac{d^2\ell}{d\theta^2} := -\frac{\partial}{\partial\theta}\left(\frac{x_i e^{\theta x_i}}{1+e^{\theta x_i}}\right)$$

$$= -\left[\frac{x_i^2 e^{\theta x_i}\left(1+e^{\theta x_i}\right) - \left(x_i e^{\theta x_i}\right)\left\{e^{\theta x_i}\cdot x_i\right\}}{\left(1+e^{\theta x_i}\right)^2}\right]$$

$$= -\left[\frac{x_i^2 e^{\theta x_i} + \cancel{\left(x_i e^{\theta x_i}\right)^2} - \cancel{x_i^2\left(e^{\theta x_i}\right)^2}}{\left(1+e^{\theta x_i}\right)^2}\right]$$

$$= -\frac{x_i^2 e^{\theta x_i}}{\left(1+e^{\theta x_i}\right)^2}$$

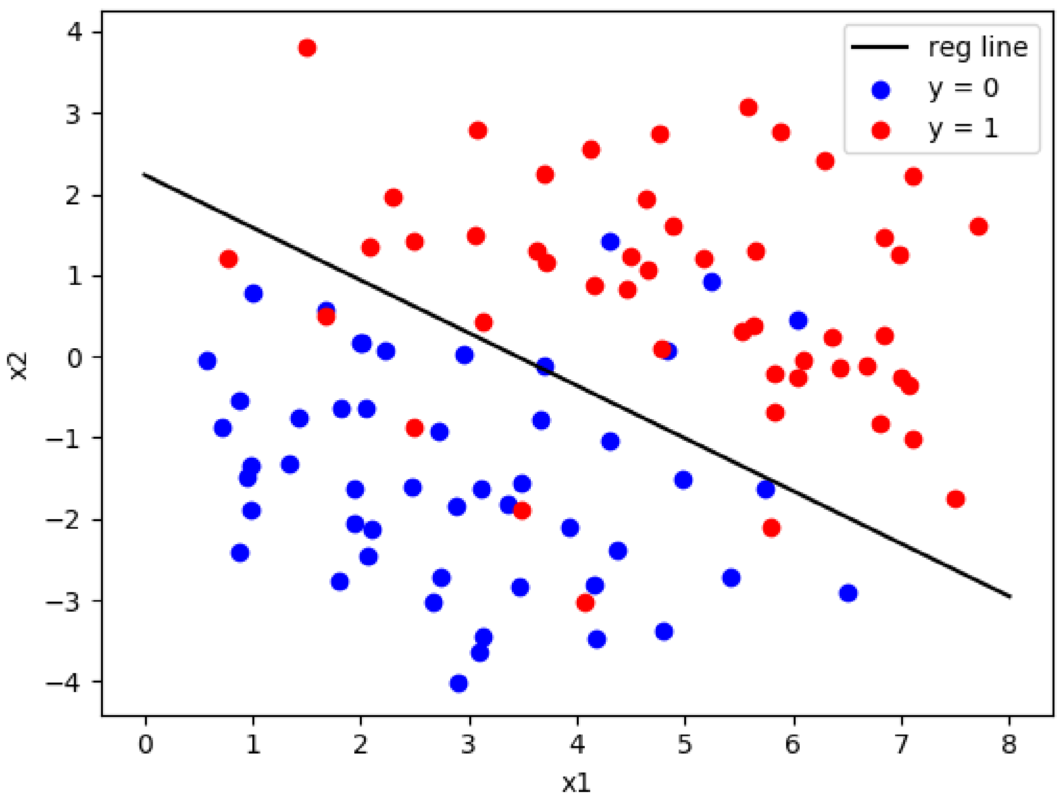
Implementation of Gradient Decent Method:

Learning Rate: 0.01

Values of Coefficients θ: [-2.62051097 0.7603714   1.1719467]

Iterations to Converge: 873

Plot:

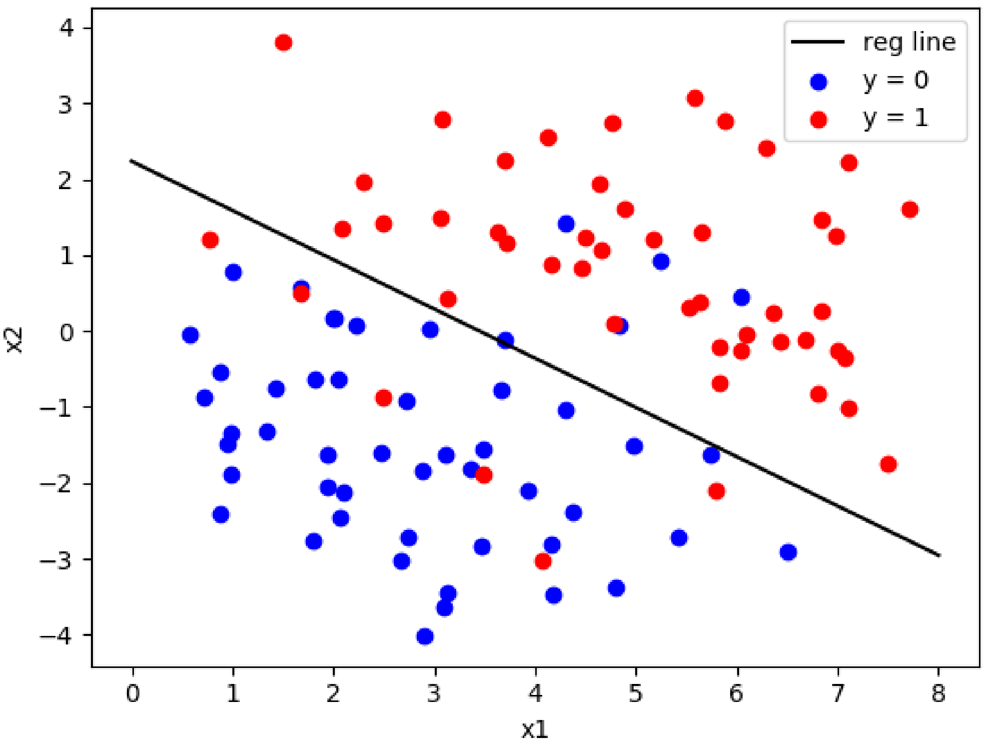


Implementation of Newton Method:

Values of Coefficients W: [-2.6205116   0.76037154 1.17194674]

Iterations to Converge: 8

Plot:



Comparison of Newton Method and Gradient Method:

Overall time taken by Gradient Method: **0.031242847442626953 seconds**

Overall time taken by Newton Method: **0.015625 seconds**

Number of iterations taken by Gradient to Converge: **873**

Number of iterations taken by Newton to Converge: **8**

Newton Method performs better than Gradient Decent method.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot
import time


#Method to find the sigmoid of the function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

#Function to add an Interncept
def addintercept(X):
    intercept = np.ones((X.shape[0], 1))

    return np.concatenate((intercept, X), axis=1)

#Function to find the loss function
def loss(h, y):
    return np.sum((y * np.log(h) + (1 - y) * np.log(1 - h)))

#Function to find the Hessian Matrix
def hessian(X,h):
    return  np.dot(X.T,(np.diag(h*(1-h)).dot(X)))


def fit_hessian(X,y):
    w = np.zeros(X.shape[1])
    lr = 0.01
    lossvalue=0
    for i in range(1000):

        z = np.dot(X, w)
        h = sigmoid(z)
        gradient = np.dot(X.T, (y-h)) #Get the gradient
        hessian_value=np.linalg.inv(hessian(X,h))#Inverse the hessian
        w+=hessian_value.dot(gradient) #Update Weights
        previousloss = lossvalue
        lossvalue=loss(h,y) #Update loss function value



        if lossvalue-previousloss==0: #CHeck if Converged
            print(i)
            break;

    return w

def fit(X,y):

    theta = np.zeros(X.shape[1])
    lr = 0.01
    lossvalue=0;
    for i in range(1000):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (y-h)) #Get the gradient
        previousloss = lossvalue  #Save old Loss function value
        lossvalue=loss(h,y) #Update the loss function value
        theta += lr * gradient  #Update the weigths
        if lossvalue-previousloss==0: #CHeck if Converged
```

```
            print(i)
            break;

    return theta


def plot_gradient(X,y,theta):
    x_class0 = []
    x_class1 = []
    for i in range(len(y)):
        # Seperating Classes
        if (y[i] == 0):
            x_class0.append(X[i, :])
        else:
            x_class1.append(X[i, :])

    x_class0 = np.array(x_class0)
    x_class1 = np.array(x_class1)

    plt.scatter(x_class0[:, 1], x_class0[:, 2], c='b', label='y = 0')
    plt.scatter(x_class1[:, 1], x_class1[:, 2], c='r', label='y = 1')
    x1 = np.linspace(0, 8, 4)

    x2 = -(theta[0] + theta[1] * x1) / theta[2]
    plt.plot(x1, x2, c='k', label='reg line')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()

def plot_hessian(X,y,w):
    x_class0 = []
    x_class1 = []
    for i in range(len(y)):
        # Seperating Classes
        if (y[i] == 0):
            x_class0.append(X[i, :])
        else:
            x_class1.append(X[i, :])

    x_class0 = np.array(x_class0)
    x_class1 = np.array(x_class1)

    plt.scatter(x_class0[:, 1], x_class0[:, 2], c='b', label='y = 0')
    plt.scatter(x_class1[:, 1], x_class1[:, 2], c='r', label='y = 1')
    x1 = np.linspace(0, 8, 4)

    x2 = -(w[0] + w[1] * x1) / w[2]
    plt.plot(x1, x2, c='k', label='reg line')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    #Load the data into X and y
   X= np.genfromtxt("q1x.dat")
   y=np.genfromtxt("q1y.dat")
```

```python
 #Add an Interncept in X
X=addintercept(X)

 #Get weights using Gradient method
start_time = time.time()
theta=fit(X,y)
print("--- %s seconds ---" % (time.time() - start_time))


 #Get weights using Newton
start_time2 = time.time()
w=fit_hessian(X,y)
print("--- %s seconds ---" % (time.time() - start_time2))


 #Plot for Gradient Desecent method
plot_gradient(X,y,theta)
 #Plot for Newton Method
plot_hessian(X,y,w)
```