Question 3

February 10, 2020

1 COL744 : Machine Learning (Assignment 1)

1.1 Question 3

1.1.1 Part (a) : Implementing logistic regression using newton's method

In this part I have implemented logistic regression using newton't update method to find optimal theta.

[1]:

```
import numpy as np
import numpy as np

import matplotlib as mp
import matplotlib.pyplot as plt

from tqdm import tqdm
import math
```

[2]:

```
X_unnormalized=np.genfromtxt('./ass1_data/data/q3/logisticX.csv',␣
 ↪delimiter=',') #Loading X
Y=np.genfromtxt('./ass1_data/data/q3/logisticY.csv') #Loading Y

X_=(X_unnormalized-X_unnormalized.mean(axis=0))/X_unnormalized.std(axis=0)␣
 ↪#Normalizing X
X=np.hstack((np.ones((X_.shape[0],1)),X_)) # Adding X0
```

[3]:

```
def g(x):
    '''Logistic function'''
    return 1/(1+((math.e)**(-1*x)))

def newton(X,Y):
    '''
    Newton's update method to find theta

    ---Parameters
    * X,Y - Training examples

    ---Returns
    theta

    '''
    theta_lst=[]
    theta = np.zeros(X.shape[1])
    for i in range(100):
        g_thetaX = g(X.dot(theta))
        grad = (X.T).dot(Y)-(X.T).dot(g_thetaX)
        diag = np.diagflat((g_thetaX.T)*(np.ones(g_thetaX.shape)-g_thetaX))
        H = -(((X.T).dot(diag))).dot(X)
        theta_next = theta - (np.linalg.inv(H)).dot(grad)
        theta_lst.append(theta)
        theta=theta_next
        if(i>2 and np.sum(np.abs(theta_lst[-1]-theta_lst[-2]))<1e-10):
            print('converged in %d iterations'%(i))
            break
    return theta_lst[-1]
```

[4]:

```
theta=newton(X,Y)

print(theta)
```

```
converged in 9 iterations
[ 0.40125316  2.5885477  -2.72558849]
```

1.1.2 Part (b) : Ploting decision surface

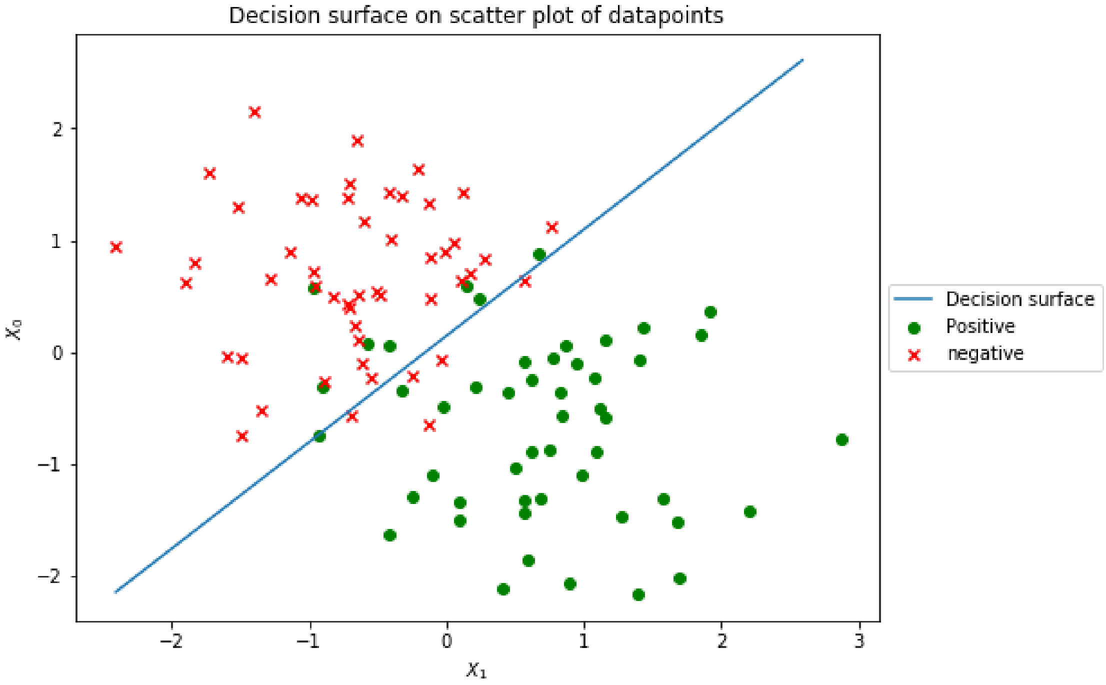
[5]:

```
plt.figure(figsize=(8,6))

##### Plotting datapoints
pos_X = X[np.where(Y==1)] #Getting datapoints with positive label
neg_X = X[np.where(Y==0)] #Getting datapoints with negative label
plt.scatter(pos_X[:,1], pos_X[:,2], marker='o', c='green', label='Positive')
plt.scatter(neg_X[:,1], neg_X[:,2], marker='x', c='red', label='negative')

##### Plotting decision surface
X_line = np.arange(X[:,1].min(), X[:,1].max())
Y_line = -1*(1/theta[2])*((theta[1]*X_line)+theta[0])
plt.plot(X_line, Y_line, label='Decision surface')


plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xlabel('$X_1$')
plt.ylabel('$X_0$')
plt.title('Decision surface on scatter plot of datapoints')
plt.show()
```



1.1.3 Observations:

- Here we can see that decision boundary found by our algorithm nicely separates positive and negative points.