Assignment 4

Contents

1 Problem Statement

Using the data set of two examination results design a predictor using logistic regression with delta learning rule using Newtons method for predicting whether a student can get an admission in the institution and compare the results with Logistic regression(using gradient descent).

2 Solution

2.1 Pre processing

2.1.1 Normalizing the data

Min - Max Normalisation Technique is used here, ie. $\frac{x-min}{max-min}$

```
x = np.array(data.iloc[:,[0,1]])
y = np.array(data.iloc[:,2]).reshape(-1,1)
ran = x.max(axis=0) - x.min(axis=0)
x = (x - x.min(axis=0)) / ran
x = np.insert(x, 0, 1,axis = 1)
x[:5]
```

```
array([[1.        , 0.06542784, 0.69465488],
       [1.        , 0.00326632, 0.19470455],
       [1.        , 0.08296784, 0.61961779],
       [1.        , 0.43176427, 0.81600135],
       [1.        , 0.7019434 , 0.65539214]])
```

Figure 1: min max normalisation.

2.1.2 Adding Polynomial Features

Added X_1^2, X_2^2 and X_1X_2 terms

```
x = np.insert(x,3,x[:,1]**2,axis=1)
x = np.insert(x,4,x[:,2]**2,axis=1)
x = np.insert(x,5,x[:,1] * x[:,2],axis=1)
```

Figure 2: Adding Polynomial Features.

2.1.3 Test train Split

Split the data to 70 - 30 for test and train set

```
d = int(y.shape[0] * 0.7)
xtrain = x[:d]
xtest = x[d:]
ytrain = y[:d]
ytest = y[d:]
```

Figure 3: test and train split.

2.2 Logistic Regression cost Function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \text{ if y} = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \text{ if y} = 0$$

```
def cost(x,y,w,lamb=0):
    h = 1/ (1 + np.exp(- np.matmul(x,w)))
    n = np.where(y==0)
    p = np.where(y==1)
    cost = - np.log(h[p]).sum() -  np.log(1 - h[n]).sum()
    return (cost + lamb * np.dot(w.T,w)[0,0]) / y.shape[0]
```

Figure 4: Logistic Regression cost Function.

Both the conditions can be combined as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left[-y^{(i)}\log(h) - (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right]$$

2.3 Newton's Method

In logistic regression, the gradient and the Hessian are

$$\nabla_\theta J = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$H = \frac{1}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)})\left(h_\theta(1-x^{(i)})\right)x^{(i)}\left(x^{(i)}\right)^T\right]$$

The update rule for Newton's method is

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

```
def newton_method(x, y, lamb=0.01, iterations=1):
    m = y.shape[0]
    cost_list = []
    w_list = []
    Lamb = lamb * np.eye(x.shape[1])
    Lamb[0,0] = 0
    np.random.seed(10)
    #w = np.ones((x.shape[1],1))
    w = np.random.randn(x.shape[1]).reshape(-1,1)
    for i in range(iterations):
        h = 1/ (1 + np.exp(- np.matmul(x,w)))
        grad = (np.dot(x.T,(h - y)) + Lamb @ w) / m
        hessian = ((((h * (1 - h) ) * x).T @ x) + Lamb) / m
        h_inv = np.linalg.pinv(hessian)
        w_list.append(grad)
        w = w -  (h_inv @ grad)
        cost_list.append(cost(x,y,w,0))
    return cost_list,w_list, w
```

Figure 5: Newton's Method.

2.3.1 Decision Boundary of Newton's Method

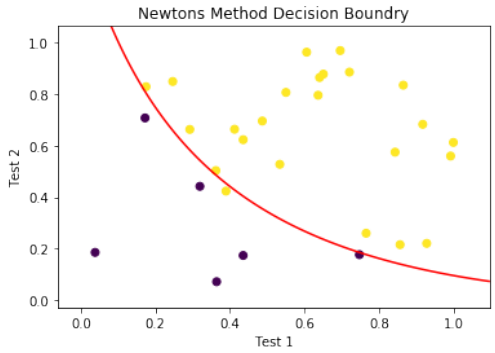

Figure 6: Decision Boundary obtained using Newton's Method.

Accuracy: 96.66666666666667%

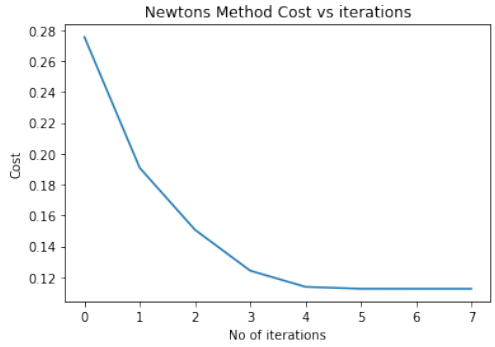

Figure 7: Cost vs no of iterations graph.

As you can see Newton's Method converge in 4 iterations.

2.4 Gradient Descent Method

The update rule for Gradient descent is

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J$$

In logistic regression, the gradient is

$$\nabla_\theta J = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

```
def gradient_descent(x, y, alpha, iterations, lamb=0):
    m = y.shape[0]
    cost_list = []
    w_list = []
    w = np.random.randn(x.shape[1]).reshape(-1,1)
    #w = np.zeros((x.shape[1],1))
    for i in range(iterations):
        h = 1/ (1 + np.exp(- np.matmul(x,w)))
        w1 = np.array(w,copy=True)
        w1[0,0]= 0
        grad = np.dot(x.T,(h - y)) + lamb * w1
        w_list.append(np.array(w,copy=True))
        w = w - (alpha/m) * grad
        if (np.abs(grad) > 0.01).sum() == 0:
            break
        cost_list.append(cost(x,y,w,lamb))
    return cost_list, w_list, w
```

Figure 8: Gradient Descent Method.

2.4.1 Decision Boundary of Gradient Descend Method

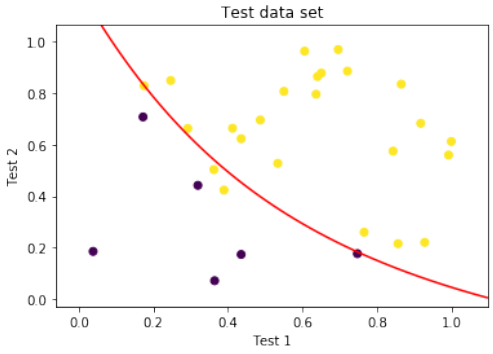

Figure 9: Decision Boundary obtained using GD Method.

Accuracy: 90%

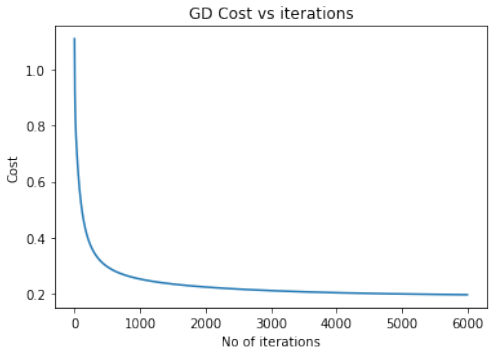

Figure 10: Cost vs no of iterations graph.

As you can see Gradient Descent took more than 6000 iterations to obtain same accuracy as Newton's Method.

2.5 Comparison

Table 1: Comparison of no of epocs.

Method	Iterations for Convergence	Accuracy
Gradient Descend	6000	90 %
Newtons Method	4	96.67 %
Gradient Descend	20000	96.67 %

Table 2: Difference between Newton's Gradient descent Method.

Gradient Descend	Newtons Method
Needs a parameter α	doesn't need α
Require Larger Number of iterations for convergence	Converges really fast
No need to compute Hessian	It Needs to Compute Hessian and its inverse
Each iteration is relatively cheap	Each iteration is relatively costly as it have to co