Question 3: Logistic Regression

Data

In the current problem we are given m values of attributes such that for each i^{th} value of the attribute ($\alpha^i \epsilon R^2$) we have $Y^i \epsilon R^1$. We define X^i such that for each i^{th} sample X^i $=< 1, \alpha^i >$ to accomodate intercept term where $X \epsilon R^{1+2}$.

Equations used in training of model parameters: θ ($\theta \epsilon R^{1+1}$)

Model : $h_\theta(X) = \frac{1}{1+e^{-\theta^T X}}$

Log Likelyhood: $LL(\theta) = \sum_{i=1}^{m} \left(Y^i log(h_\theta(X^i)) + (1 - Y^i)(log 1 - h_\theta(X^i)) \right)$

Gradient: $\bigtriangledown_\theta (J(\theta)) = \sum_{i=1}^{m} X^i (Y^i - h_\theta(X^i))$

Newton's method algorithm: $\theta^{t+1} = \theta^t - H^{-1} \bigtriangledown_\theta (J(\theta))$

$H = \bigtriangledown_\theta^2 (J(\theta)) = -\sum_{i=1}^{m} h_\theta(X^i) * (1 - h_\theta(X^i)) X^{i^T} X^i$

In [5]:

```
import sys
import numpy as np
sys.path.append('../')
from lib.ml import logistic
from lib.ml import norm
import matplotlib
import matplotlib.pyplot as plt
#%matplotlib
```

executed in 9ms, finished 22:58:33 2018-02-12

Loading testing and training data

In [6]:

```
X = norm(np.loadtxt(open('logisticX.csv'), delimiter=",").reshape(100, 2))
Y = (np.loadtxt(open('logisticY.csv'), delimiter=",")).reshape(-1, 1)
```

executed in 532ms, finished 22:58:34 2018-02-12

Function to plot model estimates

In [7]:

```
def _plot_model(self,X,Y):
    plot_eqn = '(X[:,0]*self.theta[0,1]+self.theta[0,0])*-1/self.theta[:,2]'
    plt.figure(1)
    plt.title("Model for learning rate %f"%(self.lr),fontsize=10,y=0.9)
    Y_predicted = eval(plot_eqn)
    plt.plot(X[Y[:,0]==1][:,0],X[Y[:,0]==1][:,1],'b.',label='Y=1')
    plt.plot(X[Y[:,0]==0][:,0],X[Y[:,0]==0][:,1],'r^',label='Y=0')
    plt.plot(X[:,0],Y_predicted,'g-',label='model')
    plt.legend(bbox_to_anchor=(0.7, 0.3), loc=2, borderaxespad=0.)
    plt.xlabel(r'$X_0$')
    plt.ylabel(r'$X_1$')
    plt.draw()

def _plot_contours(self):
    history = np.asarray(self.history)
    t_hist = history.shape[0]
    x = (history[:,:,0].T)[0]
    y = (history[:,:,1].T)[0]
    z = (history[:,:,2].T)[0]
    alpha = [(t_hist-x)/t_hist for x in range(0,t_hist)]
    plt.figure(3)
    sp1 = plt.subplot(111,projection='3d')
    sp1.set_title('Contours for learning rate %f'%(self.lr))
    sp1.plot(x,y,z,c='k', marker='')
    sp1.scatter(x,y,z,c=alpha, marker='o',cmap='gray')
    sp1.set_xlabel(r'$\theta_{0}$')
    sp1.set_ylabel(r'$\theta_{1}$')
    sp1.set_zlabel(r'$\theta_{2}$')
    plt.draw()
```

executed in 200ms, finished 22:58:34 2018-02-12

Training model to estimate parameter θ

Note: to see how gardient descend is working use the interactive mode by replacing

```
model._train_(lr=0.0001,b_ratio=1,iter=20000,thresh=1e-100)
```

with

```
model._train_(lr=0.0001,b_ratio=1,iter=20000,thresh=1e-100,flag=True)
```

In [8]:

```
eqn = 'np.dot(X[:,0:2],(theta[:,0:2].T))*-1/theta[:,2]'
J = 'np.sum(Y*np.log((1/(1+np.exp(-np.dot(X, theta.T)))))+(1-Y)*np.log((1-
(1/(1+np.exp(-np.dot(X, theta.T))))))).reshape(1,1)'
dJ = 'np.dot((Y - (1/(1+np.exp(-np.dot(X, theta.T))))).T, X)'
H = 'np.dot(X.T,-1*((1/(1+np.exp(-np.dot(X, theta.T))))*(1-(1/(1+np.exp(-
np.dot(X, theta.T))))))*X)'
HdJ = 'np.dot(%s,np.linalg.pinv(%s))'%(dJ,H)
model = logistic(X,Y,0.8,eqn,HdJ,J)
theta  = model._train_(lr=0.0001,b_ratio=1,iter=20000,thresh=1e-100)
```

executed in 13.1s, finished 22:58:47 2018-02-12

Y = np.dot(X[:,0:2],(theta[:,0:2].T))*-1/theta[:,2]

LL(θ) = np.sum(Ynp.log((1/(1+np.exp(-np.dot(X, theta.T)))))+(1-Y)np.log((1-(1/(1+np.exp(-np.dot(X, theta.T))))))).reshape(1,1)

H = np.dot(X.T,-1((1/(1+np.exp(-np.dot(X, theta.T))))(1-(1/(1+np.exp(-np.dot(X, theta.T))))))*X)

Estimated value of θ = [0.04378951 1.4109988 -1.3280764]

Functions to plot model estimates

- _plot_model(model,X,Y)
- _plot_contours(model)

Plotting model with data