Gradient decend algorithm used in the entire submission

In [ ]:

```
"""Encapsulating all the model parameters"""
class regression:
    """Setting up basic parameters for the training for model"""
    def __init__(self, X, Y, ratio,eqn,dJ,J,intercept=True):
        # init class for initialsing model parameters
        self.eqn = eqn
        self.J = J
        self.dJ = dJ

        #taking care of the intercept
        if intercept:
            X_temp = np.hstack((np.ones((X.shape[0], 1)), X))
            self.n_para = X.shape[-1] + 1
        else:
            X_temp = X
            self.n_para = X.shape[-1]

        data = np.hstack((X_temp, Y))
        np.random.shuffle(data)
        self.N_data = int(Y.shape[0] * ratio)
        self.data_train = data[:self.N_data, :]

        self.X = data[:, 0:self.n_para]
        self.Y = data[:, self.n_para:]

        #decding the validation set
        if ratio < 1:
            self.X_valid = data[self.N_data:, 0:self.n_para]
            self.Y_valid = data[self.N_data:, self.n_para:]
        else:
            self.X_valid = data[:self.N_data, 0:self.n_para]
            self.Y_valid = data[:self.N_data, self.n_para:]
```

executed in 387ms, finished 19:06:48 2018-02-12

In [ ]:

```
    """To create batches for stocastic gradient descend
        For the purpose of this assignment b_ratio is set to 1 therefore it will act as Batch Gradient descend
    """
    def batches(self, b_ratio):
        np.random.shuffle(self.data_train)
        self.X_train = self.data_train[:, 0:self.n_para]
        self.Y_train = self.data_train[:, self.n_para:]
        b_size = int(self.N_data*b_ratio)

        total_batches = [
            int(self.N_data / b_size) + 1
            if self.N_data % b_size else int(self.N_data / b_size)
        ][0]
        for b in range(0, total_batches):
            yield (self.Y_train[b * b_size:(b + 1) * b_size, :],
                   self.X_train[b * b_size:(b + 1) * b_size, :])
```

executed in 387ms, finished 19:06:48 2018-02-12

In [ ]:

```
    """Function will evaluate error at each interation"""
    def _J(self, theta,X,Y,Xobs=None):
        return eval(self.J)

    """Function to evaluate gradient"""
    def _dJ(self,theta,X,Y):
        return eval(self.dJ)
```

executed in 387ms, finished 19:06:48 2018-02-12

In [1]:

```

    """Trains model parameters
        lr: learning rate
        b_ratio: batch ratio
        iter: total number of iterations
        flag: To indicate the use of interactive mode
        Xobs: observed point for weighted linear regression
        thresh: Convergence criteria for the model. If |Error_T-1 - Error_T| then 
model is set to be converged
    """
    def _train_(self, lr=0.001, b_ratio=1, iter=100, thresh=0.00001,flag=False, 
Xobs = None):

        self.lr = lr
        self.iter = iter
        if flag and self.valid:
            self.plot_set()
        theta = np.zeros((1, self.n_para))
        J_old = self._J(theta,self.X_valid,self.Y_valid,Xobs=Xobs)
        history = []
        start_time = time()
        for i in range(1, iter):
            for (Y, X) in self.batches(b_ratio):

                Gd_sum = eval(self.dJ)
                theta = theta - lr*Gd_sum
                J = self._J(theta,self.X_valid,self.Y_valid,Xobs=Xobs)

                if (time()-start_time >= 0.2):
                    if flag and self.valid:
                        self.int_plot(theta,J,i,self.eqn,self.X,self.Y)
                    start_time=time()
                    history.append(np.hstack((theta, J)))

                if np.abs(J_old-J) < thresh:
                    print("Model converged at epoch %d" % (i))
                    self.theta = theta
                    self.history = history
                    return self.theta
                J_old=J
        self.theta = theta
        self.history = history
        return self.theta
```

executed in 387ms, finished 19:06:48 2018-02-12

Modifying codes for linear regression

In [ ]:

```
"""Building linear regression class for interactive plotting"""
class linear(regression):
    def __init__(self,X,Y,ratio,eqn,dJ,J,intercept=True):
        regression.__init__(self,X,Y,ratio,eqn,dJ,J,intercept=True)

        self.valid=False
        if X.shape[-1]==1:
            self.valid=True
```

In [ ]:

```
    """Initial setup for the model graphs"""
    def plot_set(self):
        plt.figure(num=4,figsize=(40,40),dpi=30)
        sp1 = plt.subplot(221,projection='3d')
        sp1.set_xlabel(r'$\theta_{0}$',fontsize=40,labelpad=40)
        sp1.set_ylabel(r'$\theta_{1}$',fontsize=40,labelpad=40)
        sp1.set_zlabel(r'$J(\theta)$',fontsize=40,labelpad=40)
        sp1.set_title('Gradient Descend',fontsize=40)
        sp1.tick_params(labelsize=20)
        plt.gca().invert_xaxis()

        sp2 = plt.subplot(222)
        sp2.set_xlabel(r'$\theta_{0}$',fontsize=40,labelpad=5)
        sp2.set_ylabel(r'$\theta_{1}$',fontsize=40,labelpad=5)
        sp2.set_title('Contour Curve',fontsize=40,y=0.9)
        sp2.tick_params(labelsize=20)

        sp3 = plt.subplot(223)
        sp3.set_xlabel(r'$epoch$',fontsize=40,labelpad=40)
        sp3.set_ylabel(r'$J(\theta)$',fontsize=40)
        sp3.set_title('Error V/s Time',fontsize=40,y=0.9)
        sp3.tick_params(labelsize=20)

        sp4 = plt.subplot(224)
        sp4.tick_params(labelsize=20)
        self.sp1 = sp1
        self.sp2 = sp2
        self.sp3 = sp3
        self.sp4 = sp4
```

executed in 245ms, finished 19:19:14 2018-02-12

In [3]:

```
    """Interactive plotting the model parameters"""
    def int_plot(self,theta,z,i,eqn,X,Y):

        sp4 = self.sp4
        x=theta[0,0]
        y=theta[0,1]
        self.sp1.scatter(x,y,z,c='b', marker='o',s=80.,alpha= i/self.iter)
        self.sp2.scatter(x,y,c='r', marker='o',s=80.,alpha= i/self.iter)
        self.sp3.scatter(i,z,c='g', marker='o',s=80.)

        sp4.cla()
        sp4.set_title('Current Model',fontsize=40,y=0.9)
        sp4.set_xlabel(r'$X$',fontsize=40,labelpad=40)
        sp4.set_ylabel(r'$Y$',fontsize=40,labelpad=40)
        Y_predicted = eval(eqn)
        sp4.scatter(X[:,1],Y,c='b', marker='o',s=80.,label='Actual Data')
        sp4.plot(X[:,1],Y_predicted,'r-',label='model',linewidth=5.0)
        sp4.legend(bbox_to_anchor=(0.7, 0.2), loc=2, borderaxespad=0.,fontsize =
30)

        plt.draw()
        plt.pause(0.01)
```

executed in 245ms, finished 19:19:14 2018-02-12

Modifying codes for Logistic regression

In [ ]:

```
"""Building logistic regression class for interactive plotting"""
class logistic(regression):
    def __init__(self,X,Y,ratio,eqn,dJ,J,intercept=True):
        regression.__init__(self,X,Y,ratio,eqn,dJ,J,intercept)
        self.valid=False
        if X.shape[-1]==2:
            self.valid=True
```

In [ ]:

```
    """Initial setup for the model graphs"""
    def plot_set(self):
        plt.figure(num=4,figsize=(60,20),dpi=30)

        sp2 = plt.subplot(221,projection='3d')
        sp2.set_xlabel(r'$\theta_{0}$',fontsize=40,labelpad=20)
        sp2.set_ylabel(r'$\theta_{1}$',fontsize=40,labelpad=20)
        sp2.set_zlabel(r'$\theta_{2}$',fontsize=40,labelpad=20)
        sp2.set_title('Contour Curve',fontsize=40,y=1.08)
        plt.gca().invert_xaxis()
        sp2.tick_params(labelsize=20)

        sp3 = plt.subplot(222)
        sp3.set_xlabel(r'$epoch$',fontsize=40,labelpad=40)
        sp3.set_ylabel(r'$J(\theta)$',fontsize=40)
        sp3.set_title(r'$LL(\theta) V/s Time$',fontsize=40,y=0.9)
        sp3.tick_params(labelsize=20)

        sp4 = plt.subplot(223)
        sp4.tick_params(labelsize=20)

        self.sp2 = sp2
        self.sp3 = sp3
        self.sp4 = sp4
```

executed in 235ms, finished 19:19:33 2018-02-12

In [4]:

```
    """Interactive plotting the model parameters"""
    def int_plot(self,theta,z,i,eqn,X,Y):

        sp4 = self.sp4

        self.sp2.scatter(theta[0,0],theta[0,1],theta[0,2],c='r',
marker='o',s=80.,alpha= i/self.iter)
        self.sp3.scatter(i,z,c='g', marker='o',s=80.)

        sp4.cla()
        sp4.set_title('Current Model',fontsize=40,y=0.9)
        sp4.set_xlabel(r'$X$',fontsize=40,labelpad=40)
        sp4.set_ylabel(r'$Y$',fontsize=40,labelpad=40)

        Y_predicted = eval(eqn)

        sp4.scatter(X[Y[:,0]==1][:,1],X[Y[:,0]==1][:,2],c='b',
marker='o',s=80.,label='Y=1')
        sp4.scatter(X[Y[:,0]==0][:,1],X[Y[:,0]==0][:,2],c='r',
marker='^',s=80.,label='Y=0')

        sp4.plot(X[:,1],Y_predicted,'r-',label='model',linewidth=5.0)
        sp4.legend(bbox_to_anchor=(0.7, 0.3), loc=2, borderaxespad=0.,fontsize =
30)

        plt.draw()
        plt.pause(0.01)
```

executed in 235ms, finished 19:19:33 2018-02-12