In [1]:

```
import numpy as np
import scipy.optimize as opt
import random
```

In [4]:

```
def sigmoid(z):
    '''
    z should be a scaler
    '''
    return 1/(1+np.exp(-z))
```

In [7]:

```
def costFunction(theta,X,y,Lambda):
    '''
    X is a (n,d) matrix
    y is a (n,)vector
    w is a (n+1,) => (b,w1,w2,w3,....)
    Lambda is a scalar
    '''
    n,d = X.shape
    X_tilda =np.c_[np.ones((n,1)),X]
    J = np.zeros((n,))
    for i in range(n):
        # J should be (n,)
        J[i] = np.log(1 + np.exp(-y[i] * (X_tilda[i,:] @ theta)))

    reg = Lambda * theta[1:].T @ theta[1:]
    return 1/n * np.sum(J) + reg
```

In [2]:

```
def gradF(theta,X,y,Lambda):
    '''
    X is a (n,d) matrix
    y is a (n,)vector
    w is a (n+1,) => (b,w1,w2,w3,....)
    Lambda is a scalar
    '''

    n,d = X.shape
    X_tilda =np.c_[np.ones((n,1)),X]
#    mu = np.zeros((n,))
#     for i in range(d):
#         mu[i] = sigmoid(y[i] * X[i,:] @ w)
    #mu should be a n-by-1
    mu = 1/(1+ np.exp(np.multiply(-y.T , (X_tilda @ theta))))
    del_J = np.zeros(d+1)
    # derivative wrt. d
    del_J[0] =  1/n * (1-mu).T @ (-y)
    # derivative wrt w=(w0,w1,w2...)
    for i in range(1,d+1):
        del_J[i] = 1/n * (1-mu).T @ (-np.multiply(y,X_tilda[:,i])) + 2*Lambda*the
ta[i]
    return del_J
```

In [1]:

```
def gradient_descent(theta_init,gradient,step_size,tolerence):
    theta_list = [theta_init]
    theta = theta_init
    curr_grad = gradient(theta)
    num_iteration = 0
    #print("curr gradient is {}".format(np.max(np.abs(curr_grad))))
    while(np.max(np.abs(curr_grad)) > tolerence and num_iteration < 100000):
        #print("**************************************************************")
        #print("curr gradient is {}".format(np.max(np.abs(curr_grad))))
        #print("current iteration {}" .format(num_iteration))
        num_iteration += 1
        theta = theta - step_size*curr_grad
        theta_list.append(theta)
        curr_grad = gradient(theta)
    if (num_iteration == 10000):
         error("not converging")
    print("it takes {0} iterations to converge".format(num_iteration))
    return theta,theta_list,num_iteration
```

In [4]:

```
def selective_gradF(theta,X,y,Lambda,m):
    '''
    X is a (n,d) matrix
    y is a (n,)vector
    w is a (n+1,) => (b,w1,w2,w3,....)
    Lambda is a scalar
    m is the number of batches, m <= n
    '''

    n,d = X.shape
    selector = random.sample(range(0,n), m) # max is n-1

    X_tilda =np.c_[np.ones((m,1)),np.copy(X[selector, :])]
    y_tilda = np.copy(y[selector])

    mu = 1/(1+ np.exp(np.multiply(-y_tilda.T , (X_tilda @ theta))))
    del_J = np.zeros(d+1)

    del_J[0] =  1/m * (1-mu).T @ (-y_tilda)

    for i in range(1,d+1):
        del_J[i] = 1/m * (1-mu).T @ (-np.multiply(y_tilda,X_tilda[:,i])) + 2*Lamb
da*theta[i]
    return del_J
```

In [ ]: