```
            predict += a * s * np.sign(x[d] - theta)

        result.append( np.sign(predict) )

    return np.array(result)
def main():

    (X, Y) = read_data('hw6_train.dat.txt')
    D = X.shape[1]          # dimension of X
    N = X.shape[0]          # data size
    print('X shape:', X.shape)
    print('Y shape:', Y.shape)

    T = 1                       # iteration
    G = []                        # all g
    a = np.zeros(T)               # weight of gt (alpha)
    Ein_g = []                    # 0/1 errors of each iteration
    Ein_G = []                    # 0/1 errors of all iteration
    Epsilon = []                # all epsilon of each iteration
    u = np.array( [ 1/N ] * N )   # weight of each sample
    U = [1]                       # sum of weights u
    sorted_index = []
    unsorted_index = []

    for d in range(D):
        index = np.argsort(X[:, d])
        sorted_index.append( index )
        unsorted_index.append( np.argsort(index) )

    for t in range(T):
        print('\rt = %d' % (t+1), end='', flush=True)

        best_abs_sum, best_s, best_i, best_d = 0, 1, -1, 0
        for d in range(D):

            index = sorted_index[d]
            left, right = 0, np.sum(Y * u)
            abs_sum = abs(right - left)

            if abs_sum > best_abs_sum:
                best_abs_sum = abs_sum
                best_s = 1 if right >= left else -1
                best_i, best_d = -1, d

            Y_tmp, u_tmp = Y[index], u[index]
            for i, y in enumerate(Y_tmp):

                right -= y * u_tmp[i]
                left += y * u_tmp[i]
                abs_sum = abs(right - left)

                if abs_sum > best_abs_sum:
                    best_abs_sum = abs_sum
                    best_s = 1 if right >= left else -1
                    best_i, best_d = i, d

        index = sorted_index[best_d]
        # best division (theta)
        X_tmp = X[index][:, best_d]
        if best_i < 0:
            theta = -np.inf
        elif best_i >= N-1:
            theta = np.inf
        else:
            x1 = X_tmp[best_i]
            x2 = X_tmp[best_i+1]
            theta = (x2 + x1) / 2

        g = (best_s, best_d, theta)

        # predict by small gt
        predict_g = predict([g], [1], X)
        error01_g = abs(predict_g - Y) / 2
        epsilon_g = np.sum(error01_g * u) / u.sum()
        scale = np.sqrt( (1-epsilon_g) / epsilon_g )
        # update u
        incorrect = np.where(error01_g == 1)[0]
        correct = np.where(error01_g == 0)[0]
        u[incorrect] *= scale
        u[correct] /= scale
        U.append( u.sum() )

        a[t] = np.log(scale)
        Ein_g.append( np.sum(error01_g) / N )
        Epsilon.append( epsilon_g )
        G.append(g)

        # predict by big Gt
        predict_G = predict(G, a, X)
        error01_G = np.sum( abs(predict_G - Y) / 2 ) / N
        Ein_G.append(error01_G)

    print('')
    print('Ein(g1):', Ein_g[0])

    print('max Ein(gt):', max(Ein_g))

    for i in range(T):
        if Ein_G[i] <= 0.05:
            print('Ein(G) < 0.05 , t = ' ,i+1)
            break
    (X_test, Y_test) = read_data('hw6_test.dat.txt')
    N_test = X_test.shape[0]
    Eout_g = []
    guni = np.zeros(N_test)
    for i, g in enumerate(G):
            print('\r%d' % (i+1), end='', flush=True)
            predict_g = predict([g], [1], X_test)
            guni +=predict_g
            error01_g = np.sum( abs(predict_g - Y_test) / 2 ) / N_test
            Eout_g.append(error01_g)
```