```
import numpy as np
import random as rnd
import matplotlib.pyplot as plt
%matplotlib inline

def gen_line():
    '''
    Generate boundary line for classification

    Returns
    -------
    2 2-dimensional array consisting of your line in form [w0, w1, w2]
and [w0, w1_norm, w2_norm]

    '''
    [x1,x2,y1,y2] = [rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0),
rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0)]
    xA,yA,xB,yB = [rnd.uniform(-1, 1) for i in range(4)]
    w = np.array([x2*y1-y2*x1, y2-y1, x1-x2])
    w_norm = np.array([1, -w[1]/w[2], -w[0]/w[2]])
    return w, w_norm

def gen_pts(n, d, w=None, w_norm=None):
    '''
    Generates random points from a uniform distribution over -1,1

    Parameters
    ----------
    n : number of points
    d : dimension of image

    Returns
    -------
    d-dimensional array consisting of n-number of uniform, random
points, and a clean slate sign
    '''
    if w is None:
        w, w_norm = gen_line()

    d_ = np.random.uniform(-1.0, 1.0,(d,n))
    x_ = np.append(np.ones(n), d_).reshape((d+1,n))
    y = np.sign(np.dot(w.T,x_))
    d_ = np.append(x_, y).reshape((d+2,n))
    return x_, y, w, d_, w_norm

def pick_pt(y_, y):
    '''
    Find misclassified points and pick one at random.

    Parameters
    ----------
    y_ : list of all output points from our updated weight
    y  : list of correct output points

    Returns
    -------
    index of random point, number of misclassified points
    '''
    mc_pts = []
    for i in xrange(len(y)):
        if y_[i] != y[i]:
            mc_pts.append(i)

    try:
        index = rnd.choice(mc_pts)
    except IndexError:
        index = 0

    return index, len(mc_pts)


def update(xi, yi_, w_):
    '''
    Takes a misclassified point and updates the weight to correctly
classify point

    Parameters
    ----------
    xi   : incorrectly classified point
    yi_  : correct sign for point
    w_   : current weight

    Returns
    -------
    updated weight
    '''
    w_ += yi_ * xi

    return w_

def pre_process(n, d):
    '''
    Creates the necessary datasets and solutions needed to run a PLA
classification

    Parameters
    ----------
    n : number of data points
    d : dimensions of dataset
    Returns
    -------
    x_  : coordinates or feature information (1, x1, x2)
    y   : solution from sign function
    w   : true weights (w0, w1, w2)
    d_  : entire dataset (incl. solution)
    w_n : normalized weights, ie. (w0=1, w1, w2)
    '''
    x_, y, w, d_, w_n = gen_pts(n,d)

    return x_, y, w, d_, w_n

#     xp = d_[:-1,d_[-1]>0]
#     xm = d_[:-1,d_[-1]<0]

#     fig = plt.figure(figsize=(6, 6))
#     plt.plot( xp[1], xp[2], 'bo')
#     plt.plot( xm[1], xm[2], 'ro' )
#     x = np.linspace(-1,1)
#     plt.plot(x, w_n[1]*x + w_n[2], color='black')
#     plt.title('PLA Classification')
#     plt.ylim([-1,1])


def pla():
    w_ = np.zeros(3)
    y_ = np.sign(np.dot(w_.T,x_))
    i = 0 # iterations

    while np.array_equal(y, y_) != True:
        index, total_mc_pts= pick_pt(y_,y)
    #     print 'i:', index, ' y_:', y_[index], ' y:', y[index]
#         print '# of misclassified points after iter %i: %i' %(i, 
total_mc_pts)
        w_ = update(x_[:,index], y[index], w_)
        y_ = np.sign(np.dot(w_.T, x_))
        i += 1
#         plt.plot(x, w_[1]*x + w_[2], color='g', alpha=0.1)
        if i%1000 == 0:
            break
    w_n = np.array([1, -w_[1]/w_[2], -w_[0]/w_[2]])
#     print 'updated:', y_, ' sol:', y
#     print w
#     print w_n

#     plt.plot(x, w_[1]*x + w_[2], color='g')
    return i, w_n
```