```
import numpy as np
#import matplotlib
from matplotlib import pyplot as plt
import pdb
#from random import random

plt.style.use('ggplot')


D = [( 3,  4, -1),
     ( 2,  3, -1),
     ( 2,  1,  1),
     ( 1,  2,  1),
     ( 1,  3,  1),
     ( 4,  4, -1)]

class svm:
  def __init__(self, lr=0.01, lp=0.01, iters=1000, prt=False, log=False):
    self.lr = lr # learning rate
    self.lp = lp # lambda parameter
    self.iters = iters
    self.prt = prt
    self.w = None
    self.b = None
    self.log = log
    if self.log:
      self.xHist = list()
      self.yHist = list()
      self.yEstHist =list()
      self.wHist = list()
      self.bHist = list()
      self.wUpdHist = list()
      self.bUpdHist = list()
      # add logs
    return

  def train(self, X, Y):
    nSamples, mfeatures = X.shape
    self.w = np.zeros(mfeatures)
    self.b = 0
    print('In training -->>')
    XY = np.concatenate((X, Y), axis=1)

    XY_tmp = np.ndarray.copy(XY)
    for j in range(self.iters):
      a = np.random.randint(0, len(XY_tmp))

      xDatum = XY_tmp[a][:-1]
      yDatum = XY_tmp[a][-1]
      XY_tmp = np.delete(XY_tmp, a, axis=0)
      yEst = yDatum * (np.dot(xDatum, self.w) - self.b) >= 1
      #pdb.pprint('XY_tmp', XY_tmp)
      #pdb.set_trace()
      if yEst: # binary classification only
        wUpdate = self.lr * (2 * self.lp * self.w)
        bUpdate = 0.0
      else:
        wUpdate = self.lr * (2 * self.lp * self.w - np.dot(xDatum, yDatum))
        bUpdate = self.lr * yDatum
      self.w -= wUpdate
```

```
      self.b -= bUpdate
      if len(XY_tmp)==0: XY_tmp = np.ndarray.copy(XY) # reload!!
      if self.prt:
        print('iter:{:3d}'.format(j))
        print('XY_tmp:')
        print(XY_tmp)
        print('yDatum:', yDatum)
        print('yEst:', yEst)
        print('wUpdate:', wUpdate)
        print('bUpdate:', bUpdate)
        print()
        print()
      if self.log: # data logger!!
        self.xHist.append(xDatum)
        self.yHist.append(yDatum)
        self.yEstHist.append(yEst)
        self.wUpdHist.append(wUpdate)
        self.bUpdHist.append(bUpdate)
        self.wHist.append(self.w)
        self.bHist.append(self.b)
    return

  def predict(self, X):
    linOutput = np.dot(X, self.w) - self.b
    return np.sign(linOutput)

  def getHyperPlaneVal(self, x, w, b, offset):
    #res = (-w[0] * x + b + offset)/w[1]
    res = np.dot(x, self.w) - self.b + offset
    return res

  def visualize(self, X, Y):
    fig = self.fig
    #plott = fig.add_subplot(2,2,2)
    #plt.scatter(X[:,0], X[:,1], marker='x', color='red', label='input')

    x01 = np.amin(X[:,0])
    x02 = np.amax(X[:,0])
    x11 = self.getHyperPlaneVal(x01, self.w, self.b, 0)
    x12 = self.getHyperPlaneVal(x02, self.w, self.b, 0)
    x11p = self.getHyperPlaneVal(x01, self.w, self.b, 1)
    x12p = self.getHyperPlaneVal(x02, self.w, self.b, 1)
    x11n = self.getHyperPlaneVal(x01, self.w, self.b, -1)
    x12n = self.getHyperPlaneVal(x02, self.w, self.b, -1)
    plt.plot([x01, x02], [x11, x12], 'y--') # draw mid line
    plt.plot([x01, x02], [x11p, x12p], color="r", label='+ class bound') # draw
p class boundary
    plt.plot([x01, x02], [x11n, x12n], color='b', label='- class bound') # draw
n class boundary
    #x1max = np.amax(X[:,1])
    #x1min = np.amin(X[:,1])
    #plott.set_ylim()
    #self.fig.legend()
    plt.show()
    return


def get_data(data):
  X = list()
  Y = list()
```

```
  for i in range(len(data)):
    X.append(np.asarray(data[i][:-1]))
    Y.append(np.asarray(data[i][-1]))
  X = np.asarray(X)
  Y = [1 if i > 0 else -1 for i in Y]
  Y = np.asarray(Y)
  Y = np.expand_dims(Y,axis=1)
  return X, Y

from SVM import SVM
if __name__ == '__main__':

  X, Y = get_data(D)
  fig = plt.figure()
  #subfig = fig.add_subplot(1,1,1)

  for i in range(len(X)):
    if Y[i]>=0:
      plt.scatter(X[i][0], X[i][1], marker='x', color='r')
    else:
      plt.scatter(X[i][0], X[i][1], marker='o', color='b')
  #pdb.set_trace()
  plt.title('Data Classification')
  plt.xlabel('X1')
  plt.ylabel('X2')
  fig.legend()
  #subfig.show()
  mySVM = svm(lr=0.01, lp=0.01, iters=1000, prt=True, log=True)
  mySVM.fig = fig
  mySVM.train(X, Y)
  mySVM.visualize(X, Y)


  # test using
  #model = SVM(max_iter=1000, kernel_type='linear', C=1000, epsilon=0.001)
  #model.fit(X, Y)
  #y_hat = model.predict(X)
```