```
import pdb
import numpy as np
import random
import matplotlib.pyplot as plt

dataOR = [( 1,  1,  1,  1,  1),
          ( 1,  1, -1,  1,  1),
          ( 1, -1,  1,  1,  1),
          ( 1, -1, -1,  1,  1),
          (-1,  1,  1,  1,  1),
          (-1,  1, -1,  1,  1),
          (-1, -1,  1,  1,  1),
          (-1, -1, -1,  1, -1)]


dataXOR = [( 1,  1,  1,  1, -1),
           ( 1,  1, -1,  1,  1),
           ( 1, -1,  1,  1,  1),
           ( 1, -1, -1,  1,  1),
           (-1,  1,  1,  1,  1),
           (-1,  1, -1,  1,  1),
           (-1, -1,  1,  1,  1),
           (-1, -1, -1,  1, -1)]

class perceptron:
  def __init__(self, lr, iters, prt, recTrainHist):
    self.lr = lr
    self.iters = iters
    self.af = self.sign
    self.prt = prt
    #self.loss = L
    self.recTrainHist = recTrainHist
    if self.recTrainHist:
      self.WHist = list()
      self.bHist = list()
      self.XHist = list()
      self.YHist = list()
      self.YestHist = list()
      self.ErrHist = list()
      self.deltaWHist = list()
      self.updateHist = list()
      self.accHist = list()
    self.W = None
    self.b = None
    return

  def sign(self, val):
    return np.where(val>0, 1, -1)

  def train(self, X, Y):
    self.nSamples, self.mFeatures = X.shape
    self.W = np.random.uniform(low=-.1, high=.1, size=self.mFeatures)
    self.b = 0

    XY = np.concatenate((X, Y), axis=1)
    XY_tmp = np.ndarray.copy(XY)
    for _ in range(self.iters):
      i = random.randint(0, len(XY_tmp)-1)
      Xdatum = XY_tmp[i][:-1]
```

```
      Ydatum = XY_tmp[i][-1]
      XY_tmp = np.delete(XY_tmp, i, axis=0)
      #pdb.set_trace()
      linOutput = np.dot(Xdatum, self.W) + self.b
      y_ = self.af(linOutput)
      # apply the Perceptron rule
      error = Ydatum - y_
      update = self.lr * error
      delta_W = update * Xdatum
      self.W += delta_W
      self.b += error
      if len(XY_tmp)==0: XY_tmp = np.ndarray.copy(XY)
      if self.prt==True:
        print("iter:{:<4d}".format(_)," |X:{:2}".format(Xdatum[0]), \
          "{:2}".format(Xdatum[1]), "{:2}".format(Xdatum[2]), \
          "{:2}".format(Xdatum[3]), " |Y:{:2}".format(Ydatum), \
          " |y_:{:2}".format(y_), " | err:{:2}".format(error), \
          " | W:{:3.2f}".format(self.W[0]), "{:3.2f}".format(self.W[1]), \
          "{:3.2f}".format(self.W[2]), '{:3.2f}'.format(self.W[3]))
      if self.recTrainHist==True:
        self.WHist.append(self.W)
        self.bHist.append(self.b)
        self.XHist.append(Xdatum)
        self.YHist.append(Ydatum)
        self.YestHist.append(y_)
        self.ErrHist.append(error)
        self.deltaWHist.append(delta_W)
        #self.updateHist.append(update)
        self.accHist.append(self.getacc(self.YHist, self.YestHist))
    return

  def predict(self, X):
    dotprod = np.dot(X, self.W) + self.b
    return self.sign(dotprod)

  def getacc(self, groundtruth, prediction):
    correct = 0
    for i in range(len(groundtruth)):
      if groundtruth[i] == prediction[i]:
        correct += 1
    return correct / float(len(groundtruth)) * 100.0

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




if __name__ == '__main__':

  print('-->> Training and testing with OR')
```

```
  X, Y = get_data(dataOR)
  pOR = perceptron( 0.1, 1000, True, True)
  pOR.train(X,Y)
  plt.style.use('ggplot')
  plt.plot(range(len(pOR.accHist)), pOR.accHist, label='OR Perceptron Taining Ac
c')
  #plt.show()
  #figOR = plt.figure()
  #ax = figOR.add_subplot(111, projection='3d')



  print("\n\n")
  print('-->> Training and testing with XOR')
  xorX, xorY = get_data(dataXOR)
  pXOR = perceptron( 0.1, 1000, True, True)
  pXOR.train(xorX,xorY)
  #plt.style.use('fivethirtyeight')

  plt.plot(range(len(pXOR.accHist)), pXOR.accHist, label='XOR Perceptron Taining
 Acc')

  plt.xlabel('Training iterations')
  plt.ylabel('Training accuracy [%]')
  plt.title('OR vs. XOR training accuracy')
  plt.legend()
  #plt.grid(True)
  #plt.tight_layout()
  plt.show()
```

Learning rate is passed in.