```
#!/usr/bin/python

import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sp
from mpl_toolkits.mplot3d import Axes3D
import copy
import pprint
from matplotlib.backends.backend_pdf import PdfPages



class analyZeData(object):
    def __init__(self, dataSetFile):
        # Load in the dataSet into an NP array
        # It retains the row column convention given the file
        # i j Zij Xij(1) Xij(2)
        self.NGROUPS=4
        self.dataSet=np.loadtxt(dataSetFile, comments="#")
        self.w=np.ones((self.dataSet.shape[0],self.NGROUPS))
        self.ones=np.ones((self.dataSet.shape[0],))
        self.zeros=np.zeros((self.dataSet.shape[0],))
        self.preBias=np.ones((self.dataSet.shape[0],2))
        self.LL=[]

    def bcMultiply(self, ynx2, xnx1):
        [m, n] = ynx2.shape
        result = copy.deepcopy(ynx2)
        for i in range(m):
            result[i]=(ynx2[i]*xnx1[i])
        return result



    def weightedMean(self,w,x):
        [mu]=np.sum(self.bcMultiply(x,w), axis=0, keepdims=True) / np.sum(w)
        return mu

    def weightedCov(self,w,x,mu):
        [m, n] = self.w.shape
        sigma= np.mat(self.bcMultiply((x-mu),w)).T*(x-mu) / np.sum(w)
        return sigma


    def EStep(self, mu0, mu1, sigma0, sigma1, phi, lembda):
        self.w=np.ones((self.dataSet.shape[0],self.NGROUPS))

        y0=sp.multivariate_normal(mu0, sigma0)
        y1=sp.multivariate_normal(mu1, sigma1)
        [m, n] = self.w.shape
        self.w=np.array([(1-phi)*lembda*y0.pdf(self.X), (1-phi)*(1-lembda)*y1.pdf(se
lf.X), (phi)*(1-lembda)*y0.pdf(self.X), (phi)*lembda*y1.pdf(self.X)]).T
        s=np.sum(self.w, axis=1)
        self.LL.append(np.sum(np.log(np.sum(self.w, axis=1))))
        one=np.ones(s.shape)
        s=np.divide(one,s)
        self.w=self.bcMultiply(self.w, s)
        #print "EStep", self.w[0]

    def precintBIAS(self, mu0, mu1, sigma0, sigma1, phi, lembda):
        y0=sp.multivariate_normal(mu0, sigma0)
        y1=sp.multivariate_normal(mu1, sigma1)
        p=np.array([(1-phi)*lembda*y0.pdf(self.X), (1-phi)*(1-lembda)*y1.pdf(self.X)
, (phi)*(1-lembda)*y0.pdf(self.X), (phi)*lembda*y1.pdf(self.X)]).T
        p0=np.sum(p[:,0:2], axis=1, keepdims=True)/(1-phi)
        p1=np.sum(p[:,2:4], axis=1, keepdims=True)/phi
        s=p0+p1
        self.preBias=np.ones((self.dataSet.shape[0],2))
        nPrecint=int(self.dataSet[-1,0])
        nVoterPerPrecint=int(self.dataSet[-1,1])
        print "####################### Precint Preference Table ##################"
        print "---------------+--------------------+----------+-------------------"
        print "%-15s|%-20s|%-15s " %("Precint ID", "P(Yi=1/Xi)", "> 0.5")
        print "---------------+--------------------+----------+-------------------"
        for i in range(nPrecint):
            p_1=1
            p_0=1
            for j in range(nVoterPerPrecint):
                p_1*=p0[i*nVoterPerPrecint+j]
                p_0*=p1[i*nVoterPerPrecint+j]
            s=p_1+p_0
            self.preBias[i*nVoterPerPrecint:(i+1)*nVoterPerPrecint,0]*=p_0/s
            self.preBias[i*nVoterPerPrecint:(i+1)*nVoterPerPrecint,1]*=p_1/s
            print  "%-15s|%-20s|%-15s " %(i+1, float(p_1/s), (p_1/s> 0.5).flatten())
        print "---------------+--------------------+----------+-------------------"




    def initStep(self,labeled=False):
        if labeled==True:
            self.X=self.dataSet[:,3:5]
            self.Z=self.dataSet[:,2]
            self.Y=copy.deepcopy(self.ones)
            N=self.dataSet.shape[0]
            precinct=0
            pointer=0
            for i in range(N):
                if self.dataSet[i,0] != precinct :
                    if np.sum(self.Z[pointer:i]) < 10 and i != 0 :
                        self.Y[pointer:i] = 0 *self.ones[pointer:i]
                    precinct=self.dataSet[i,0]
                    pointer=i
            if np.sum(self.Z[precinct:N]) < 10 :
                self.Y[pointer:N] = 0 *self.ones[pointer:N]
            self.Z=self.dataSet[:,2]
            self.w[:,0]=np.array(((self.Z+self.Y) == self.zeros), dtype=int)
            self.w[:,1]=np.array(((self.Z-self.Y) == self.ones), dtype=int)
            self.w[:,2]=np.array(((self.Y-self.Z) == self.ones), dtype=int)
            self.w[:,3]=np.array((np.multiply(self.Z,self.Y) == self.ones), dtype=in
t)
            return 0
        else:
            # weight Values Would be set in E-Step
            self.X=self.dataSet[:,2:4]
            [mu0]=np.mean(self.X, axis=0, keepdims=True)
            sigma0=np.cov(self.X.T)
            mu1=mu0
            sigma1=sigma0
            phi=0.5
            lembda=0.5
            return [mu0.T, mu1.T, sigma0, sigma1, phi, lembda]

    def MStep(self):
        return self.MLEstimator()

    def MLEstimator(self):
        mu0=self.weightedMean((self.w[:,0]+self.w[:,2]),self.X)
        mu1=self.weightedMean((self.w[:,1]+self.w[:,3]),self.X)
        sigma0=self.weightedCov((self.w[:,0]+self.w[:,2]),self.X,mu0)
        sigma1=self.weightedCov((self.w[:,1]+self.w[:,3]),self.X,mu1)
        phi=sum((self.w[:,2]+self.w[:,3]))/self.w.shape[0]
        lembda=sum((self.w[:,0]+self.w[:,3]))/self.w.shape[0]
        #self.precintBIAS(mu0, mu1, sigma0, sigma1, phi, lembda)

        return [mu0, mu1, sigma0, sigma1, phi, lembda]


    def visualizeData(self, ax, mu0, mu1, sigma0, sigma1, phi, lembda):
        [m, n] = self.w.shape
        y0=sp.multivariate_normal(mu0, sigma0)
        y1=sp.multivariate_normal(mu1, sigma1)
        #print "DEBUG:",mu0
        #print "DEBUG:",mu1
        #print "DEBUG:",sigma0
        #print "DEBUG:",sigma1
        #print "DEBUG:",lembda, phi
        #Defining Mesh for contour Plot
        x, y = np.mgrid[-3:3:0.1, -3:3:0.1]
        pos = np.empty(x.shape + (2,))
        pos[:, :, 0] = x; pos[:, :, 1] = y
        p_z_1=y1.pdf(self.X)*(1-lembda)
        p_z_2=y0.pdf(self.X)*lembda
        s=(p_z_1+p_z_2)
        p_z_1=np.divide(p_z_1,s)
        p_z_2=np.divide(p_z_1,s)
        p_z_1=np.multiply(p_z_1,self.preBias[:,0])
        p_z_2=np.multiply(p_z_2, self.preBias[:,1])
        self.Z=np.array(((p_z_1+p_z_2)> 0.5), dtype=int)
        #self.Z=np.array((p_z_1 > p_z_0), dtype=int)



        nPrecint=int(self.dataSet[-1,0])
        nVoterPerPrecint=int(self.dataSet[-1,1])


        plt.scatter(self.X[:,0], self.X[:,1] ,c=self.Z, alpha=0.8)
        plt.scatter(mu0[0], mu0[1], s=70, c='yellow')
        plt.scatter(mu1[0], mu1[1], s=70, c='yellow')

        pie=np.mean(self.Z, axis=0, keepdims=True)

        # Mixture Contour
        plt.contour(x,y, (pie*y1.pdf(pos)+ (1-pie)* y0.pdf(pos)) )
        #ax.plot_surface(x,y, (pie*y1.pdf(pos)+ (1-pie)* y0.pdf(pos)) )
        #return [mu0, mu1, sigma0, sigma1, phi, lembda]

    def runEM(self, itr, initFromMLE=False, epsilon=1):
        self.LL=[]
        print "####################### START EM Training on UNabled Set ############
##########"
        [ mu0, mu1, sigma0, sigma1, phi, lembda]=self.initStep()
        if initFromMLE:
            mle.initStep(labeled=True)
            [ mu0, mu1, sigma0, sigma1, phi, lembda]=mle.MLEstimator()

        mu0*=epsilon
        mu1*=epsilon
        #sigma0*=epsilon
        #sigma1*=epsilon
        phi*=epsilon
        lembda*=epsilon

        for i in range(itr):
            self.EStep(mu0, mu1, sigma0, sigma1, phi, lembda)
            [mu0, mu1, sigma0, sigma1, phi, lembda]=self.MStep()
            #print "DEBUG:", "ITR", i

        self.printEstimates(mu0, mu1, sigma0, sigma1, phi, lembda, epsilon)

        print "####################### END   EM Training on UNabled Set ############
##########"
        return [mu0, mu1, sigma0, sigma1, phi, lembda]

    def printEstimates(self,mu0, mu1, sigma0, sigma1, phi, lembda, epsilon=1.0) :
        print "######################  EM Estimates EPSILON=", epsilon, "###########
##########"
        print "[LEMDA, PHI]: ",
        pprint.pprint([lembda, phi])
        print "mu0: ",
        pprint.pprint(mu0)
        print "mu1: ",
        pprint.pprint(mu1)
        print "sigma0: "
        pprint.pprint(sigma0)
        print "sigma1: "
        pprint.pprint(sigma1)
        print "#####################################################################
##########"



pp=PdfPages('./Model2.pdf')
fig=plt.figure()

# Train
print "####################### START MLE Training on Labled Set ####################
##"
mle=analyZeData('./surveylabeled.dat')
mle.initStep(labeled=True)
[mu0, mu1, sigma0, sigma1, phi, lembda]=mle.MLEstimator()
print "####################### END Train Parameters on Labled Set ##################
##"
print ""
print "######################  Maximum Liklihood Estimates #########################
##"
mle.printEstimates(mu0, mu1, sigma0, sigma1, phi, lembda)
print "#############################################################################
##"

# Predict/Label
print "####################### Prediction on Unlabeled Set #########################
##"
mle2=analyZeData('./surveyunlabeled.dat')
mle2.initStep(labeled=False)
mle2.precintBIAS(mu0, mu1, sigma0, sigma1, phi, lembda)
ax=fig.add_subplot(121)
plt.title("MODEL-2: Distribution - MLE")
mle2.visualizeData(ax, mu0, mu1, sigma0, sigma1, phi, lembda)
print "#############################################################################
##"
print ""
#ax=fig.add_subplot(122, projection='3d')



#exit()

em=analyZeData('./surveyunlabeled.dat')
[mu0, mu1, sigma0, sigma1, phi, lembda]=em.runEM(10, initFromMLE=True)
print ""

em.precintBIAS(mu0, mu1, sigma0, sigma1, phi, lembda)
ax=fig.add_subplot(122)
plt.title("MODEL2: Distribution - EM")
em.visualizeData(ax, mu0, mu1, sigma0, sigma1, phi, lembda)
plt.savefig(pp, format='pdf')
plt.show()
plt.title("MODEL-2: Log Liklihood Over Iterations")
plt.plot(em.LL, label="INIT=MLE")
em.runEM(10, initFromMLE=True, epsilon=0.9)
plt.plot(em.LL, label="INIT=MLE - 10%")
em.runEM(10, initFromMLE=True, epsilon=0.6)
plt.plot(em.LL, label="INIT=MLE - 40%")

plt.legend()
plt.savefig(pp, format='pdf')
plt.show()
pp.close()
```