```python
#!/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
from matplotlib.backends.backend_pdf import PdfPages
import pprint



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=2
        self.dataSet=np.loadtxt(dataSetFile, comments="#")
        self.w=np.ones((self.dataSet.shape[0],self.NGROUPS))

    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)
        #print "MU_COMP", 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, pie):
        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-pie)*y0.pdf(self.X), pie*y1.pdf(self.X)]).T
        self.LL.append(np.sum(np.log(np.sum(self.w, axis=1))))
        s=np.sum(self.w, axis=1).reshape((m,1))
        self.w=np.divide(self.w, s)
        #print "ESTEP", y0.pdf(self.X)[:5]

    def partialLL(self):
        #y0=sp.multivariate_normal(mu0, sigma0)
        y1=0
        #print np.sum(self.w, axis=1).shape
        #print self.LL[-1]

    def initStep(self,labeled=False):
        if labeled==True:
            self.X=self.dataSet[:,3:5]
            self.Z=self.dataSet[:,2]
            self.w[:,1]=self.Z
            self.w[:,0]=(1-self.Z)
            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
            pie=0.5
            return [mu0.T, mu1.T, sigma0, sigma1, pie]

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

    def MLEstimator(self):
        #print "BEFORE", self.X[0]
        mu0=self.weightedMean(self.w[:,0],self.X)
        #print "After", self.X[0]
        mu1=self.weightedMean(self.w[:,1],self.X)
        sigma0=self.weightedCov(self.w[:,0],self.X,mu0)
        sigma1=self.weightedCov(self.w[:,1],self.X,mu1)
        pie=sum(self.w[:,1])/self.w.shape[0]

        #print "MSTEP:", mu0, mu1, self.w[:,0].shape
        #print "MSTEP:",
        #print  [mu0, mu1]
        return [mu0, mu1, sigma0, sigma1, pie]

    def visualizeData(self, ax):
        #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

        self.Z=np.array((self.w[:,1] > self.w[:,0]), dtype=int)
        #plt.scatter(self.X[(self.Z==0)][:,0], self.X[(self.Z==0)][:,1], c='red', al
pha=0.5)
        #plt.scatter(self.X(self.Z==1)[:,0], self.X[(self.Z==1)][:,1] , c='black', a
lpha=0.5)
        plt.scatter(self.X[:,0], self.X[:,1] ,c=self.Z, alpha=0.7)

        #Following should seems like Mixture of Gaussian is after all not a bad idea
        #plt.show()


        #Expected value of sufficient Statistics (X) -> mu0 for the Z=0 group
        if self.X[(self.Z==0)].shape[0] !=0:
            [mu0]=np.mean(self.X[(self.Z==0)], axis=0, keepdims=True)
            #Expected value of sufficient Statistics (X^2) -> sigma0 for the Z=0 gro
up

            sigma0=np.cov(self.X[(self.Z==0)].T)
            y0=sp.multivariate_normal(mu0, sigma0)
           # plt.contour(x,y, y0.pdf(pos))

        if self.X[(self.Z==1)].shape[0] !=0:
            #Expected value of sufficient Statistics (X) -> mu1 for the Z=1 group
            [mu1]=np.mean(self.X[(self.Z==1)], axis=0, keepdims=True)
            #Expected value of sufficient Statistics (X^2) -> sigma1 for the Z=1 gro
up

            sigma1=np.cov(self.X[(self.Z==1)].T)
            y1=sp.multivariate_normal(mu1, sigma1)
            #plt.contour(x,y, y1.pdf(pos))


        plt.scatter(mu0[0], mu0[1], s=70, c='yellow')
        plt.scatter(mu1[0], mu1[1], s=70, c='yellow')
        #Expected value of sufficient statistics Z
        pie=np.mean(self.Z, axis=0, keepdims=True)

        # Mixture Contour
        if self.X[(self.Z==1)].shape[0] !=0 and self.X[(self.Z==0)].shape[0] !=0:
            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)) )

        #print " HIT", mu0, mu1, sigma0, sigma1, pie
        return [mu0, mu1, sigma0, sigma1, pie]

    def runEM(self, itr, initFromMLE=False, epsilon=1.0):
        self.LL=[]
```

```
        print "####################### START EM Training on UNabled Set ############
##########"
        [ mu0, mu1, sigma0, sigma1, pie]=self.initStep()
        if initFromMLE:
            mle.initStep(labeled=True)
            [ mu0, mu1, sigma0, sigma1, pie]=mle.MLEstimator()

        mu0*=epsilon
        mu1*=epsilon
        sigma0*=epsilon
        sigma1*=epsilon

        for i in range(itr):
            #print "ITR", i, ":",
            self.EStep(mu0, mu1, sigma0, sigma1, pie)
            self.partialLL()
            [mu0, mu1, sigma0, sigma1, pie]=self.MStep()
        self.printEstimates(mu0, mu1, sigma0, sigma1,pie, epsilon)
        print "####################### END   EM Training on UNabled Set ############
##########"
        return [mu0, mu1, sigma0, sigma1, pie]

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




pp=PdfPages('./Model1.pdf')
fig=plt.figure()
print "####################### START MLE Training on Labled Set ####################
##"
ax=fig.add_subplot(121)
plt.title("MODEL-1: MLE: Labled Set")
#ax=fig.add_subplot(121, projection='3d')

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

mle.visualizeData(ax)

ax=fig.add_subplot(122)
plt.title("MODEL-1: EM Labled Unlabled Set")
#ax=fig.add_subplot(122, projection='3d')


em=analyZeData('./surveyunlabeled.dat')
em.runEM(10, initFromMLE=True)
em.visualizeData(ax)
plt.savefig(pp, format='pdf')
plt.show()
```

```
plt.title("MODEL-1: 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()
```