CSE-569 Homework-4 Solution

Name: Kunal Vinay Kumar Suthar
ASURite ID: 1215112535

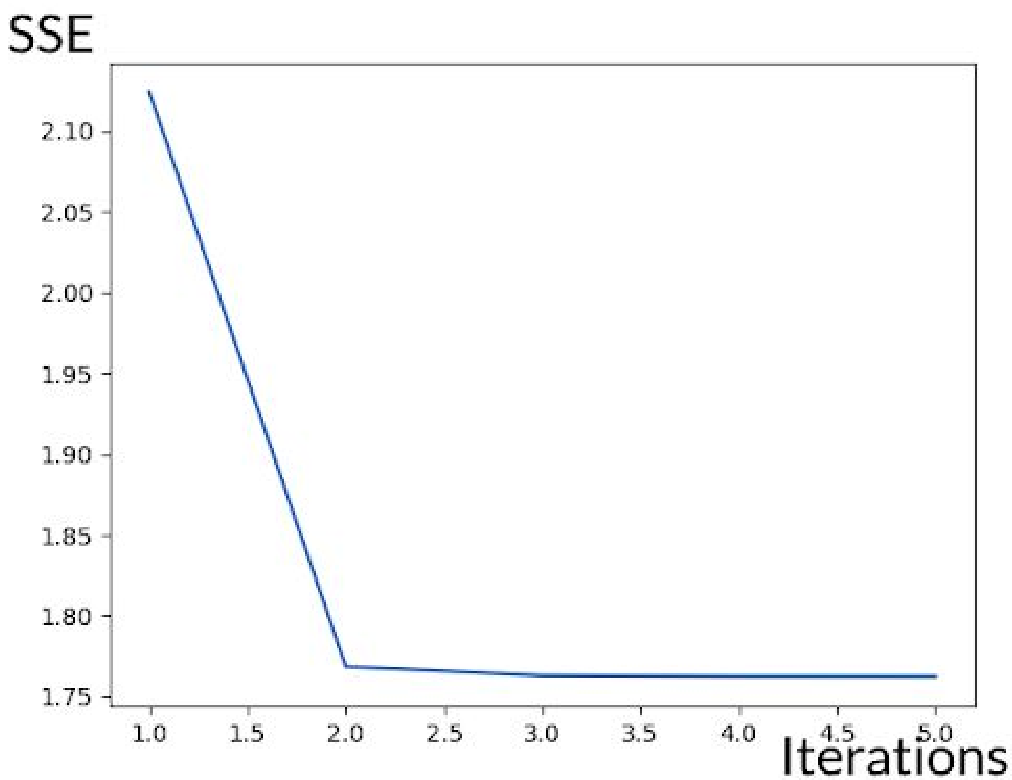
Q1. K-Means

1) A plot of squared sum error (SSE) (divided by the number of samples) as a function of iterations for r = 5 runs for the K-Means algorithm.

Solution 1)

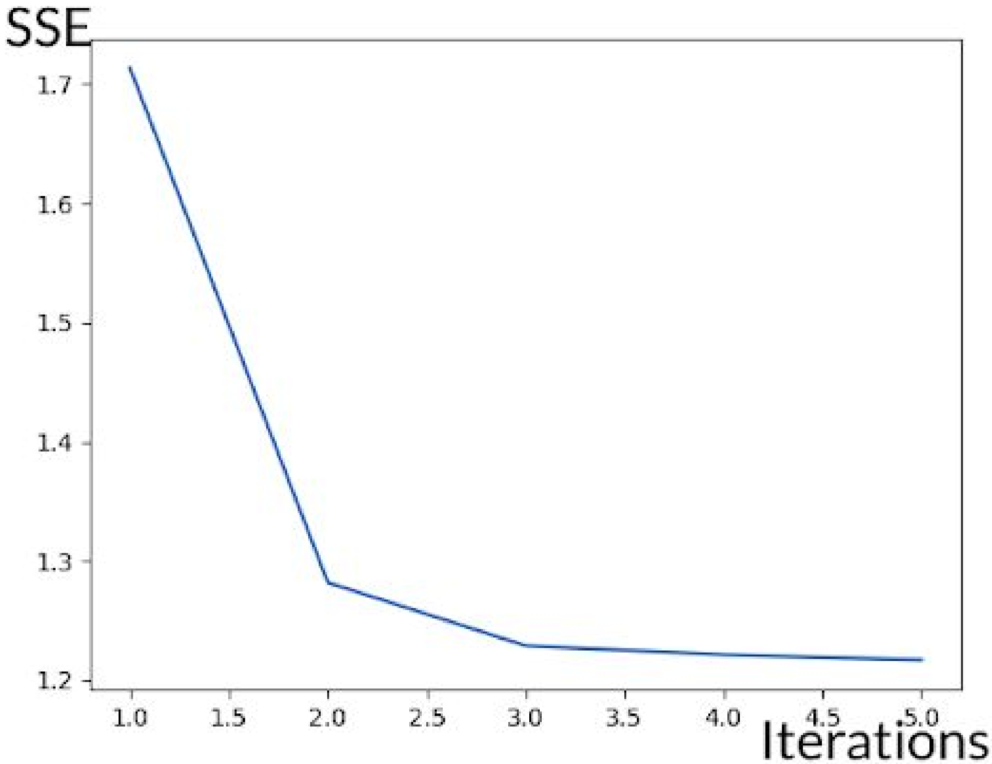
For **r=5**

ON DATASET 1:



Plot of SSE vs iterations

ON DATASET 2:

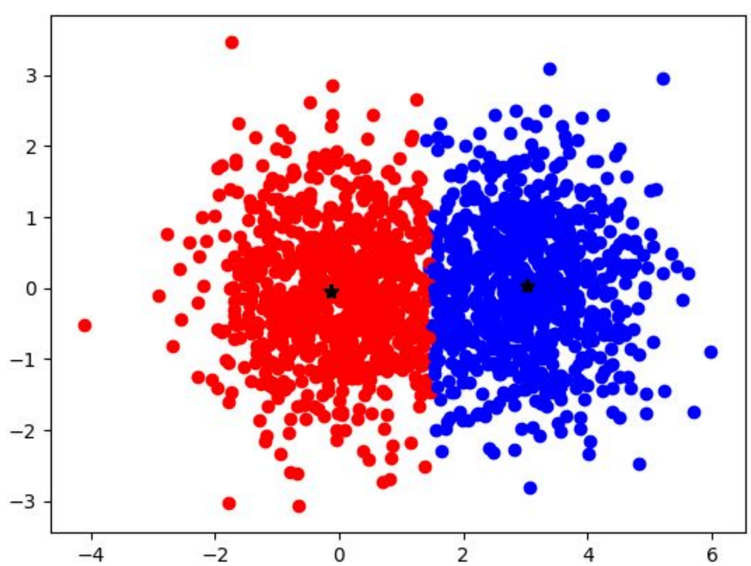


2) Depict the clustering for the lowest SSE among the r trials.

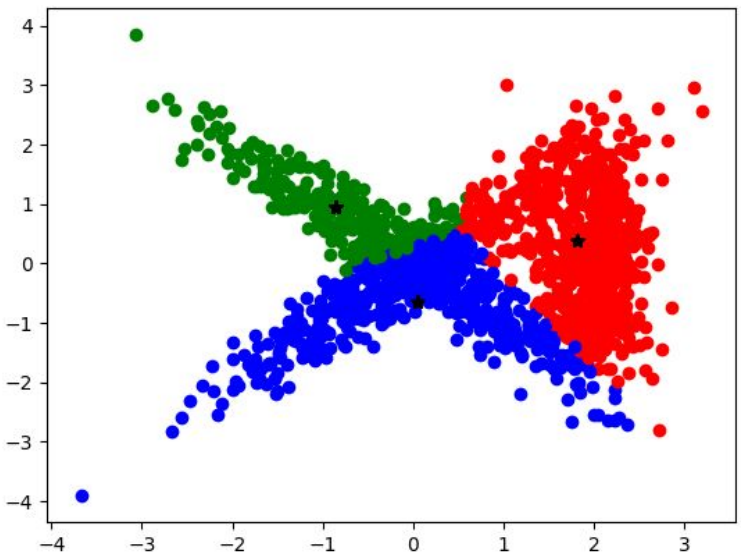
Solution 2)

K for DATASET 1 = 2
K for DATASET 2 = 3

CLUSTER FOR DATASET 1:



CLUSTER FOR DATASET 2:



Q2. Gaussian Mixture Model

3) For both the initializations of the EM algorithm, plot of the log likelihood as a function of iterations for the EM algorithm.

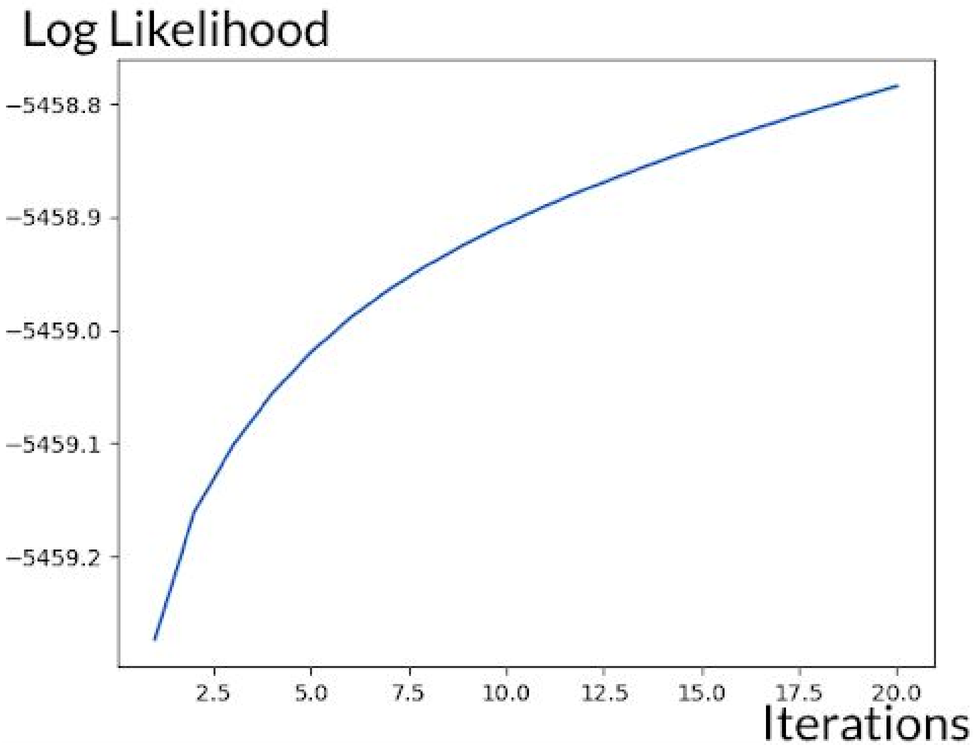
Solution 3

1. ***STRATEGY 1:*** Random initialization of mean vectors and usage of the overall data covariance (multiplied by a random positive factor) as the initial covariance matrices

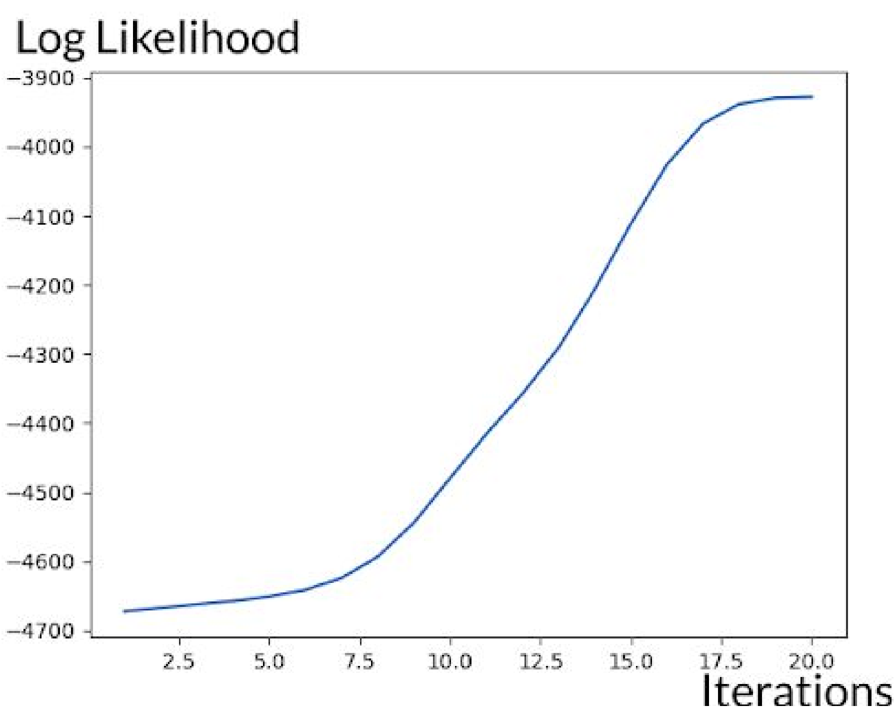
2. ***STRATEGY 2:*** Using the mean vectors and the covariances of the K clusters from K-Means as the initial mean vectors and covariances for the EM algorithm

Using Strategy 1:

Dataset 1:

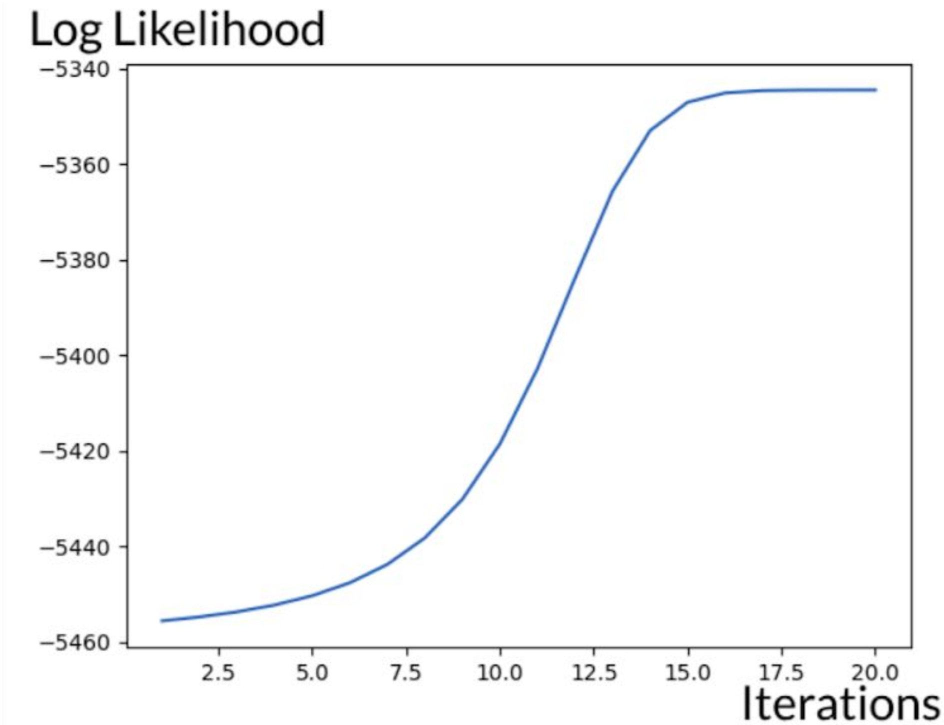


Dataset 2:

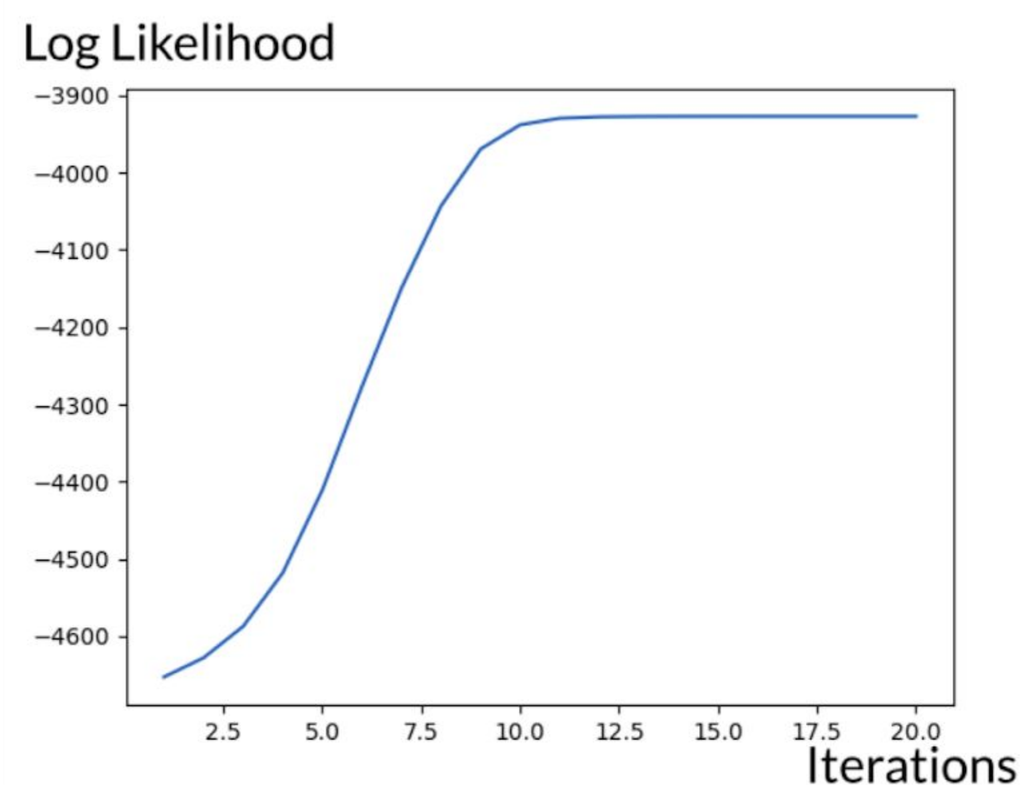


Using Strategy 2:

Dataset 1:



Dataset 2:

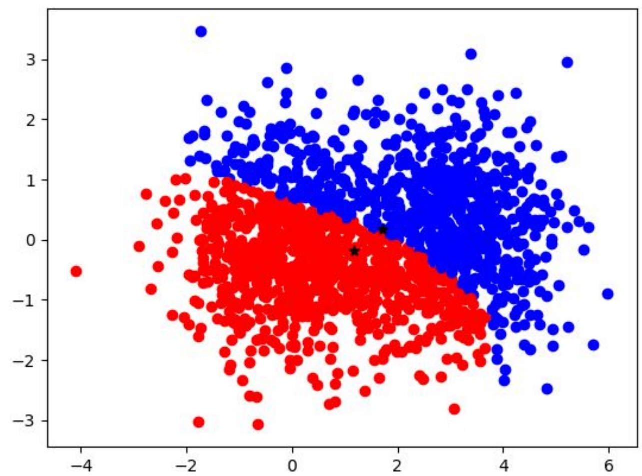


4) For both the initializations of the EM algorithm, depict the clustering.
Solution 4

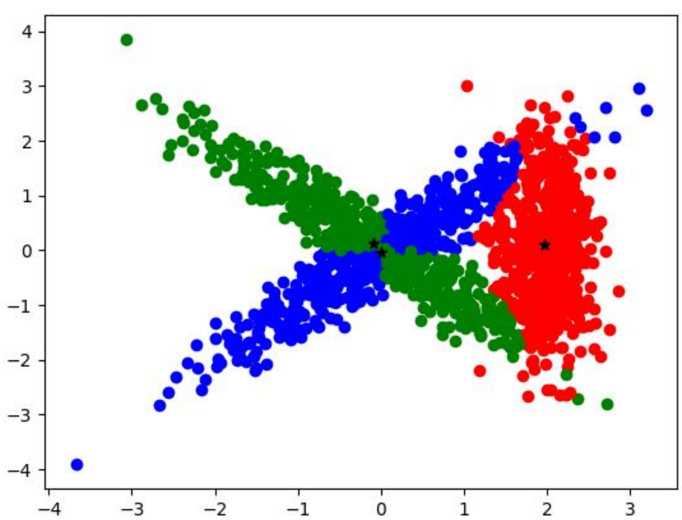
Using Strategy 1:

K for DATASET 1 = 2
K for DATASET 2 = 3

Dataset 1:

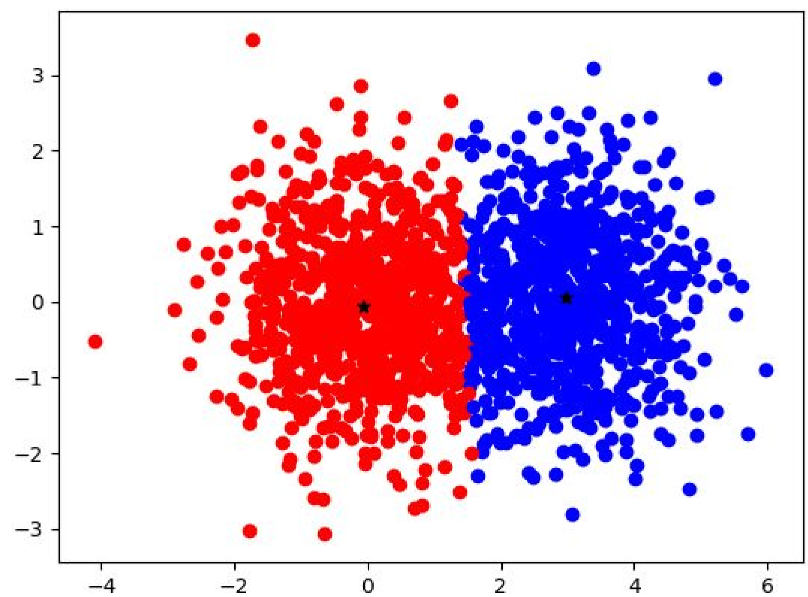


Dataset 2:

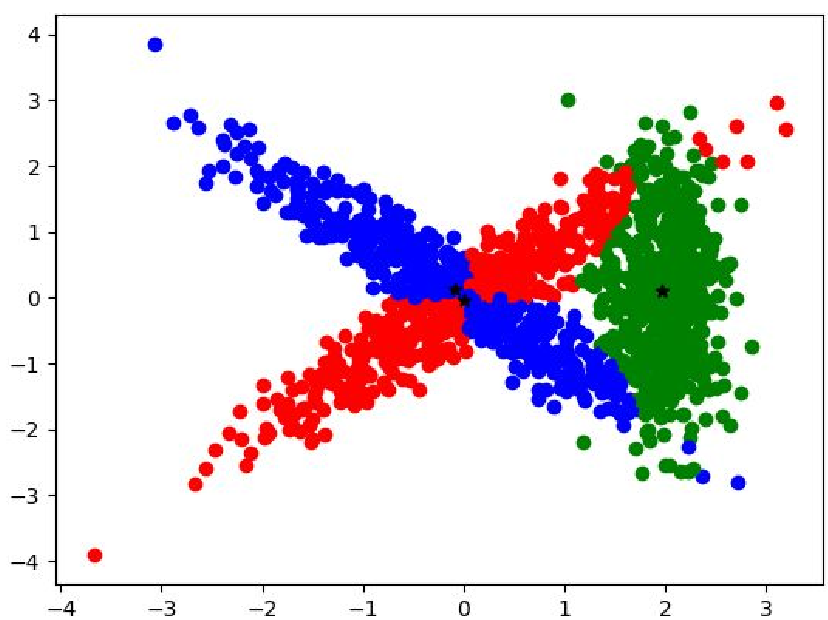


<u>Using strategy 2:</u>

Dataset 1:



Dataset 2:



5) A short discussion on the initialization.

Solution 5

Dataset 1 is composed of points with 2 Gaussian Distributions. Dataset 2 is composed of points with 3 Gaussian Distributions. Points belonging to a multivariate gaussian distribution, form elliptical clusters in a two dimensional plane. From the cluster diagram for Dataset 1 which has random initialization of mean vectors and usage of the overall data covariance as the covariance matrix, we observe that the clusters are not well defined ellipses. This is because of the random initialization of the mean points, which could stray away from the actual cluster center. From the cluster diagram of Dataset 1 with mean vectors and covariances from K-means, we find better elliptical clusters for the two gaussian distributions. This is because the mean vectors from the K-means algorithm is likely to be more accurate than the random initializations. Hence, we get a better result with the latter initialization.

6) Code for K-Means and EM algorithm

Solution 6

CODE FOR K-MEANS:

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import random
from math import pow
datasets_dir = '/home/kunal/Desktop/HW4_CSE569/Dataset_1.txt'

def main():

	x=[]
	y=[]
	with open('Dataset_1.txt') as f:
		for line in f:
			data = line.split()
			x.append(float(data[0]))
			y.append(float(data[1]))

	x=np.asarray(x)
	y=np.asarray(y)
	print x.shape
	print y.shape
	r=5
	K=3
```

```
    u_k,r_nk=apply_Kmeans(x,y,r,K)
    print u_k
    print r_nk

    color=['red','blue','green','pink','purple','black']
    for i in range(0,1600):
        for j in range(0,K):
            if r_nk[i][j]==1:
                plt.scatter(x[i],y[i],c=color[j])
            plt.scatter(u_k[j][0],u_k[j][1],marker='*',color='black')
    plt.show()

# Euclidean Distance
def distance(a, b):
    return np.linalg.norm(a - b)

def apply_Kmeans(x,y,r,K):
    rand_k=random.sample(np.arange(0,1600,1),K)
    print rand_k


    points = np.array(list(zip(x, y)), dtype=np.float32)
    print points.shape

    u_k=points[rand_k]
    print u_k

    SSE=[]
    indices=[]
    r_nk=np.zeros((1600,K))

    for index in range(0,r):

        r_nk=np.zeros((1600,K))
        sse=0
        for i in range(0,1600):
            mind=9999999
            min_index=0
            for j in range(0,K):
                d1=distance(points[i],u_k[j])
                if d1 < mind:
                    mind=d1
                    min_index=j
            r_nk[i][min_index]=1
            sse=sse+pow(mind,2)
        SSE.append(sse)
        ones=np.count_nonzero(r_nk,axis=0)
```

```
            u_k= np.dot(r_nk.T,points)

            for i in range(0,K):
                  u_k[i]=u_k[i]/ones[i];

            print ones
            plt.plot(sse/1600,index)
            plt.show()
            print "uk=",u_k
            print "SSE=",sse/1600

            indices.append(index+1)

            print r_nk

      print indices
      SSE=np.asarray(SSE)
      SSE=SSE/1600
      plt.plot(indices,SSE)
      plt.show()

      return u_k,r_nk


if __name__ == "__main__":
    main()
```

CODE FOR EM ALGORITHM: (FOR STRATEGY 1 FOR INITIALIZATION 1)

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import random
from math import pow
import scipy.stats
datasets_dir = '/home/kunal/Desktop/HW4_CSE569/Dataset_1.txt'

def main():
      x=[]
      y=[]
      with open('Dataset_2.txt') as f:
            for line in f:
                  data = line.split()
                  x.append(float(data[0]))
```

```
                y.append(float(data[1]))

    x=np.asarray(x)
    y=np.asarray(y)
    print x.shape
    print y.shape

    K=3
    iterations=20
    applyEM(K,x,y,iterations)

def applyEM(K,x,y,iterations):

    rand_k=random.sample(np.arange(0,1600,1),K)
    print rand_k

    x_points = np.array(list(zip(x, y)), dtype=np.float32)
    print x_points.shape

    #randomly initializing u_k,cov_k and pi_k
    u_k=x_points[rand_k]
    print u_k

    cov=np.cov(x_points.T)*2

    cov_k=[]
    for i in range(0,K):
        cov_k.append(cov)

    cov_k=np.asarray(cov_k)

    pi_k=[]
    sum=0
    for i in range(0,K):
        pi=random.uniform(0,0.5)
        if i == K-1:
            pi=1-sum

        sum=sum + pi
        pi_k.append(pi)

    pi_k=np.asarray(pi_k)

    log=[]
    itr=[]
    gamma_nk=np.zeros((x_points.shape[0],K))

    for i in range(0,iterations):
        gamma_nk=E(x_points,u_k,cov_k,pi_k)

        u_k,cov_k,pi_k=M(x_points,u_k,cov_k,pi_k,gamma_nk)
        lik=LogLikelihood(x_points,u_k,cov_k,pi_k)
```

```
		print lik,"for iteration",i
		print "u_k=",u_k
		itr.append(i+1)
		log.append(lik)

	print log
	plt.plot(itr,log)
	plt.show()
	color=['red','blue','green','pink','purple','black']

	classes=np.zeros((x_points.shape[0],1))
	for i in range(0,x_points.shape[0]):
		classes[i]=np.argmax(gamma_nk[i])
		print classes[i]

	print x
	print y
	for i in range(0,x_points.shape[0]):
		plt.scatter(x[i],y[i],c=color[int(classes[i][0])])

	for i in range(0,K):
		plt.scatter(u_k[i][0],u_k[i][1],marker='*',color='black')
	plt.show()

def LogLikelihood(x_points,u_k,cov_k,pi_k):
	likelihood=0
	for i in range(0,x_points.shape[0]):
		sumval=0
		for j in range(0,u_k.shape[0]):
			sumval=sumval+pi_k[j]*scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])
		likelihood=likelihood+np.log(sumval)
	return likelihood

def E(x_points,u_k,cov_k,pi_k):

	gamma_nk=np.zeros((x_points.shape[0],u_k.shape[0]))
	for i in range(0,gamma_nk.shape[0]):
		denominator=0
		for j in range(0,gamma_nk.shape[1]):
			denominator=denominator+scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])

		for j in range(0,gamma_nk.shape[1]):
			numerator = scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])
			gamma_nk[i][j]= numerator/denominator


	return gamma_nk


def M(x_points,u_k,cov_k,pi_k,gamma_nk):

	N_k= np.sum(gamma_nk,axis=0)
```

```
        for i in range(0,u_k.shape[0]):
                numerator1=0
                numerator2=np.zeros((cov_k.shape[1],cov_k.shape[2]))


                for j in range(0,x_points.shape[0]):
                        numerator1=numerator1+gamma_nk[j][i]*x_points[j]

                u_k[i]=numerator1/N_k[i]



                for j in range(0,x_points.shape[0]):

numerator2=numerator2+gamma_nk[j][i]*np.dot((x_points[j]-u_k[i]).T.reshape(2,1),(x_points[j]-u_k[i]).reshape(
1,2))

                cov_k[i]=numerator2/N_k[i]

        pi_k= N_k/x_points.shape[0]

        return u_k,cov_k,pi_k

if __name__ == "__main__":
    main()
```

CODES FOR STRATEGY 2 with EM PARAMETERS from KMeans Clustering:

File: Q1_Kmeans.py

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import random
from math import pow
from Q2_GMixModel import applyEM

datasets_dir = '/home/kunal/Desktop/HW4_CSE569/Dataset_2.txt'

def main():


        x=[]
        y=[]
        with open('Dataset_2.txt') as f:
                for line in f:
```

```
                data = line.split()
                x.append(float(data[0]))
                y.append(float(data[1]))

x=np.asarray(x)
y=np.asarray(y)
print x.shape
print y.shape
r=5
K=3
u_k,r_nk=apply_Kmeans(x,y,r,K)

print "rnk=",r_nk.shape
maxi=np.argmax(r_nk,axis=1)

x_points = np.array(list(zip(x, y)), dtype=np.float32)
print maxi.shape

cluster_points0=[]
cluster_points1=[]
cluster_points2=[]

for i in range(0,1500):
        if maxi[i]==0:
                cluster_points0.append(x_points[i])
        elif maxi[i]==1:
                cluster_points1.append(x_points[i])
        else:
                cluster_points2.append(x_points[i])

cluster_points0=np.asarray(cluster_points0)
cluster_points1=np.asarray(cluster_points1)
cluster_points2=np.asarray(cluster_points2)

print "0=",cluster_points0
print "1=",cluster_points1
print "2=",cluster_points2

cov_k=np.zeros((3,2,2))
cov_k[0]=np.cov(cluster_points0.T)
cov_k[1]=np.cov(cluster_points1.T)
cov_k[2]=np.cov(cluster_points2.T)

applyEM(3,x,y,20,u_k,cov_k)


color=['red','blue','green','pink','purple','black']
# cluster_data=[[],[]]
for i in range(0,1500):
        for j in range(0,K):
                if r_nk[i][j]==1:
                        plt.scatter(x[i],y[i],c=color[j])
```

```
                    plt.scatter(u_k[j][0],u_k[j][1],marker='*',color='black')
        plt.show()

# Euclidean Distance
def distance(a, b):
    return np.linalg.norm(a - b)

def apply_Kmeans(x,y,r,K):
        rand_k=random.sample(np.arange(0,1500,1),K)
        print rand_k


        points = np.array(list(zip(x, y)), dtype=np.float32)
        print points.shape

        u_k=points[rand_k]
        print u_k

        SSE=[]
        indices=[]
        r_nk=np.zeros((1600,K))

        for index in range(0,r):

                r_nk=np.zeros((1500,K))
                sse=0
                for i in range(0,1500):
                        mind=9999999
                        min_index=0
                        for j in range(0,K):
                                d1=distance(points[i],u_k[j])
                                if d1 < mind:
                                        mind=d1
                                        min_index=j
                        r_nk[i][min_index]=1
                        sse=sse+pow(mind,2)
                SSE.append(sse)
                ones=np.count_nonzero(r_nk,axis=0)

                u_k= np.dot(r_nk.T,points)

                for i in range(0,K):
                        u_k[i]=u_k[i]/ones[i];

                print ones
                plt.plot(sse/1500,index)
                plt.show()
                print "uk=",u_k
                print "SSE=",sse/1500

                indices.append(index+1)
```

```
            print r_nk

        print indices
        SSE=np.asarray(SSE)
        SSE=SSE/1500
        plt.plot(indices,SSE)
        plt.show()

        return u_k,r_nk


if __name__ == "__main__":
    main()
```

File: Q2_GMixModel.py

```
import numpy as np
import os
import pdb
import matplotlib.pyplot as plt
import random
from math import pow
import scipy.stats
datasets_dir = '/home/kunal/Desktop/HW4_CSE569/Dataset_1.txt'

def applyEM(K,x,y,iterations,u_k,cov_k):

        rand_k=random.sample(np.arange(0,1600,1),K)
        print rand_k

        x_points = np.array(list(zip(x, y)), dtype=np.float32)
        print x_points.shape

        print "u_kshape=",u_k.shape

        cov=np.cov(x_points.T)*2

        cov_k=[]
        for i in range(0,K):
                cov_k.append(cov)

        cov_k=np.asarray(cov_k)

        pi_k=[]
        sum=0
        for i in range(0,K):
```

```
        pi=random.uniform(0,0.5)
        if i == K-1:
            pi=1-sum

        sum=sum + pi
        pi_k.append(pi)

    pi_k=np.asarray(pi_k)

    log=[]
    itr=[]
    gamma_nk=np.zeros((x_points.shape[0],K))

    for i in range(0,iterations):
        gamma_nk=E(x_points,u_k,cov_k,pi_k)

        u_k,cov_k,pi_k=M(x_points,u_k,cov_k,pi_k,gamma_nk)
        lik=LogLikelihood(x_points,u_k,cov_k,pi_k)
        print lik,"for iteration",i
        print "u_k=",u_k
        itr.append(i+1)
        log.append(lik)

    print log
    plt.plot(itr,log)
    plt.show()
    color=['red','blue','green','pink','purple','black']

    classes=np.zeros((x_points.shape[0],1))
    for i in range(0,x_points.shape[0]):
        classes[i]=np.argmax(gamma_nk[i])
        print classes[i]

    print x
    print y
    for i in range(0,x_points.shape[0]):
        plt.scatter(x[i],y[i],c=color[int(classes[i][0])])

    for i in range(0,K):
        plt.scatter(u_k[i][0],u_k[i][1],marker='*',color='black')
    plt.show()

def LogLikelihood(x_points,u_k,cov_k,pi_k):
    likelihood=0
    for i in range(0,x_points.shape[0]):
        sumval=0
        for j in range(0,u_k.shape[0]):
            sumval=sumval+pi_k[j]*scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])
        likelihood=likelihood+np.log(sumval)
    return likelihood
```

```
def E(x_points,u_k,cov_k,pi_k):

       gamma_nk=np.zeros((x_points.shape[0],u_k.shape[0]))
       for i in range(0,gamma_nk.shape[0]):
              denominator=0
              for j in range(0,gamma_nk.shape[1]):
                     denominator=denominator+scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])

              for j in range(0,gamma_nk.shape[1]):
                     numerator = scipy.stats.multivariate_normal(u_k[j],cov_k[j]).pdf(x_points[i])
                     gamma_nk[i][j]= numerator/denominator


       return gamma_nk


def M(x_points,u_k,cov_k,pi_k,gamma_nk):

       N_k= np.sum(gamma_nk,axis=0)

       for i in range(0,u_k.shape[0]):
              numerator1=0
              numerator2=np.zeros((cov_k.shape[1],cov_k.shape[2]))


              for j in range(0,x_points.shape[0]):
                     numerator1=numerator1+gamma_nk[j][i]*x_points[j]

              u_k[i]=numerator1/N_k[i]



              for j in range(0,x_points.shape[0]):

numerator2=numerator2+gamma_nk[j][i]*np.dot((x_points[j]-u_k[i]).T.reshape(2,1),(x_points[j]-u_k[i]).reshape(
1,2))

              cov_k[i]=numerator2/N_k[i]

       pi_k= N_k/x_points.shape[0]

       return u_k,cov_k,pi_k
```