Name: Janmejay Mohanty

CS 559-B Homework Assignment 4

Solution 1:

Given:

The three cluster centers after the first round of execution.

The three clusters are , so calculating the Euclidean distance between of each point from all the three clusters.

First Iteration:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Centroid:1 |  |  |  |  |  |  |
| Centroid:2 |  |  |  |  |  |  |
| Centroid:3 |  |  |  |  |  |  |

seed1

seed2

seed3

1)

The 3 clusters with cluster points are:

Cluster 1

Cluster 2

Cluster 3

2)

Calculating the center after the first round:

Center1

Center2

Center3

Solution 3:

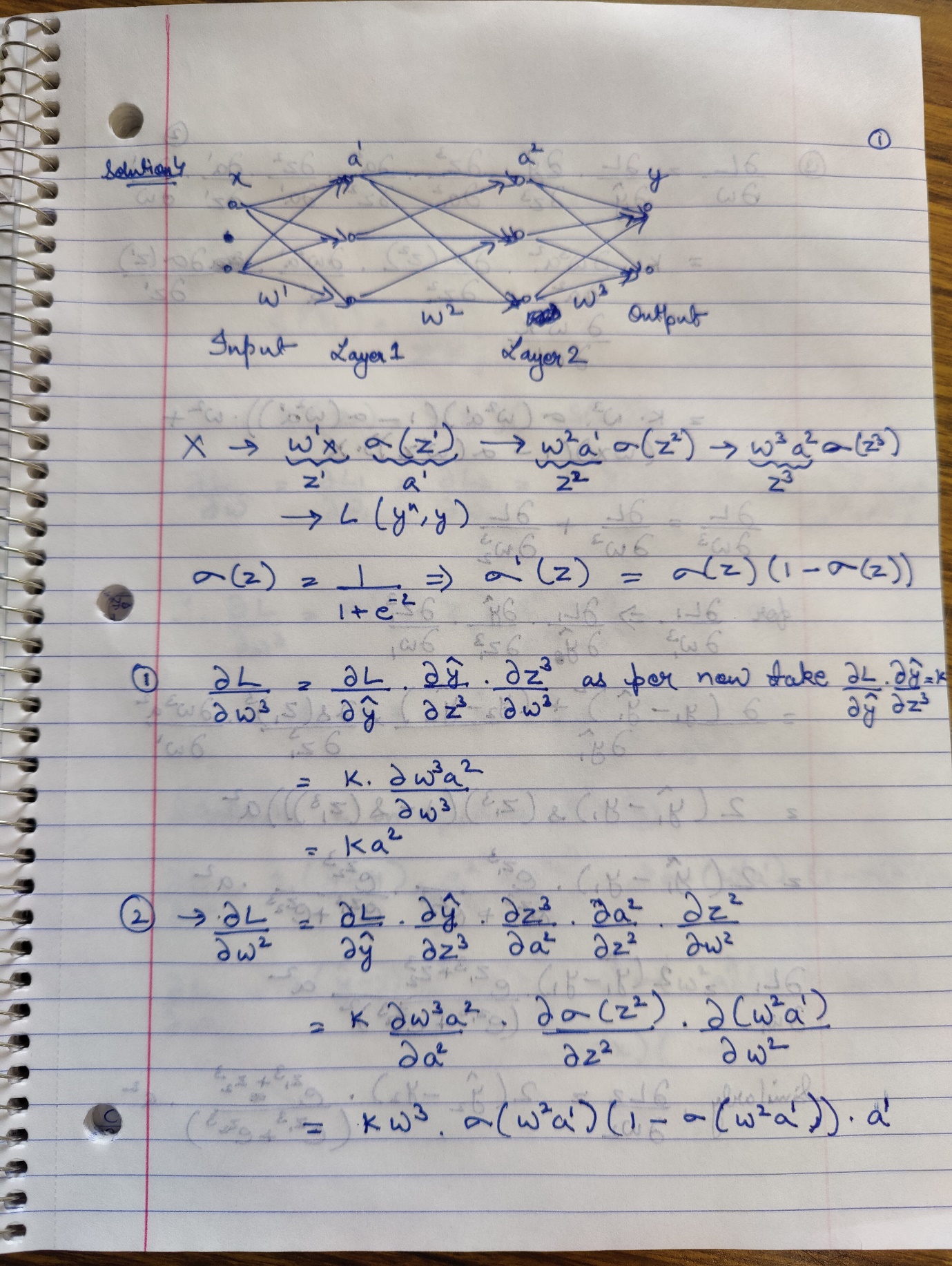
1)

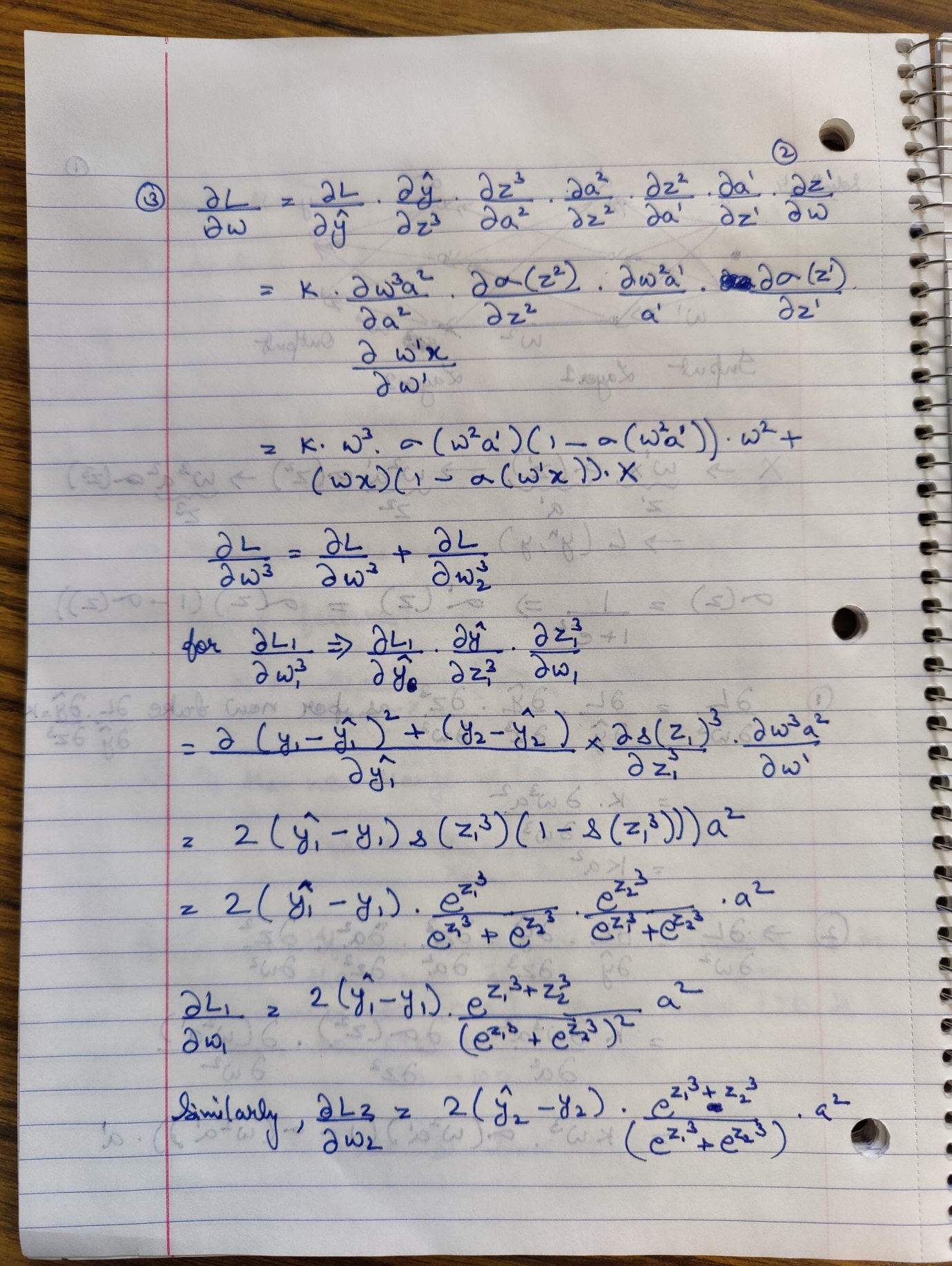
Since , the new test example is classified as C1

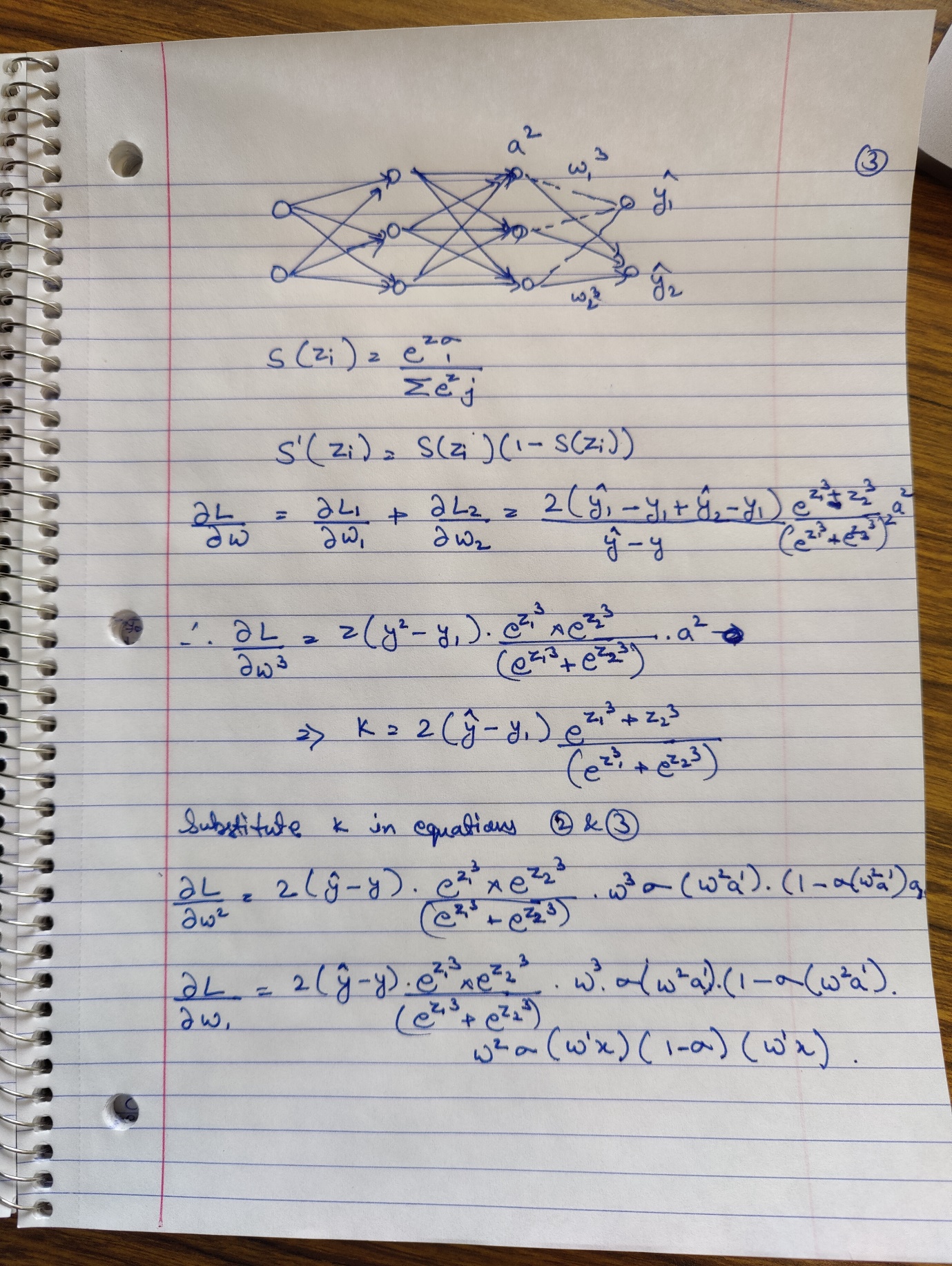
2)

=> C1

Solution 4:







Solution 2:

import numpy as np

import pandas as pd

from scipy.stats import multivariate\_normal

from scipy.stats import mode

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

X = pd.read\_csv('points.dat.txt', sep = " ", header=None)

#Reference from http://www.oranlooney.com/post/ml-from-scratch-part-5-gmm/

class GMM:

def \_\_init\_\_(self, k, max\_iter=5):

self.k = k

self.max\_iter = int(max\_iter)

def initialize(self, X):

self.shape = X.shape

self.n, self.m = self.shape

self.phi = np.full(shape=self.k, fill\_value=1/self.k)

self.weights = np.full( shape=self.shape, fill\_value=1/self.k)

random\_row = np.random.randint(low=0, high=self.n, size=self.k)

self.mu = [ X[row\_index,:] for row\_index in random\_row ]

self.sigma = [ np.cov(X.T) for \_ in range(self.k) ]

def e\_step(self, X):

self.weights = self.predict\_proba(X)

self.phi = self.weights.mean(axis=0)

def m\_step(self, X):

for i in range(self.k):

weight = self.weights[:, [i]]

total\_weight = weight.sum()

self.mu[i] = (X \* weight).sum(axis=0) / total\_weight

self.sigma[i] = np.cov(X.T,

aweights=(weight/total\_weight).flatten(),

bias=True)

def fit(self, X):

self.initialize(X)

for iteration in range(self.max\_iter):

self.e\_step(X)

self.m\_step(X)

def predict\_proba(self, X):

likelihood = np.zeros( (self.n, self.k) )

for i in range(self.k):

distribution = multivariate\_normal(

mean=self.mu[i],

cov=self.sigma[i])

likelihood[:,i] = distribution.pdf(X)

numerator = likelihood \* self.phi

denominator = numerator.sum(axis=1)[:, np.newaxis]

weights = numerator / denominator

return weights

def predict(self, X):

weights = self.predict\_proba(X)

return np.argmax(weights, axis=1)

gmm = GMM(10,5)

gm = gmm.fit(X)

