HomeWork 2#

Name: Tianze Wang
USCID: 3993275888

Q1:

(a)

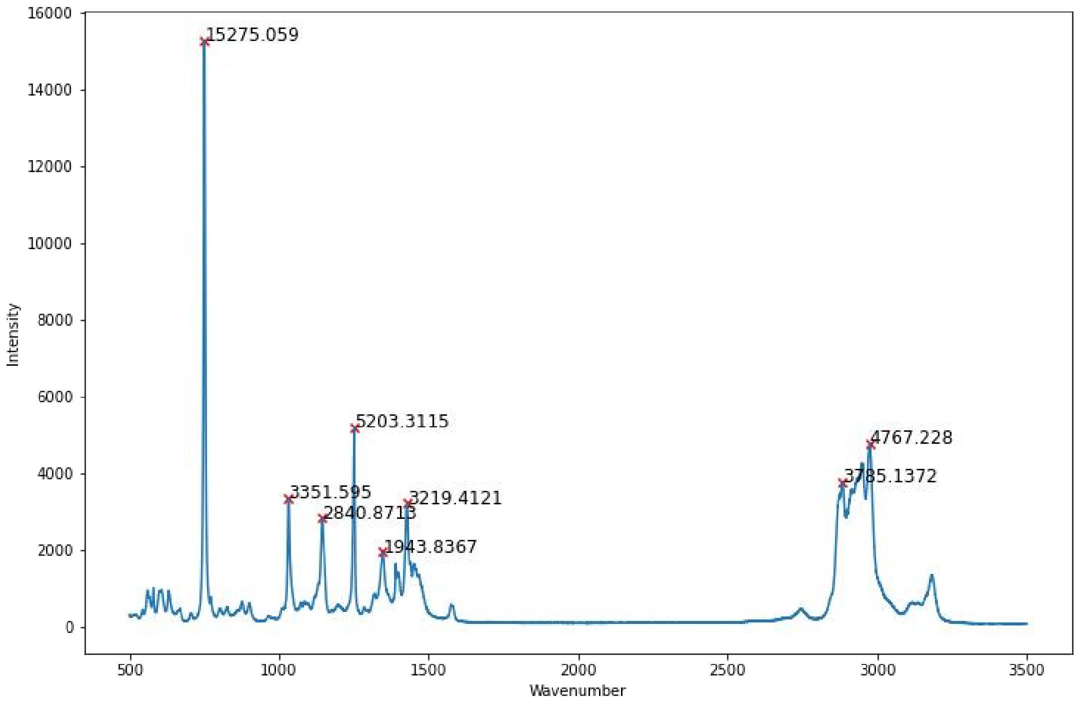
Wavenumber estimates for the eight largest spectral peak:

(750.42657, 1250.886, 2975.9155, 2884.5649, 1031.9596, 1427.287, 1145.3602, 1346.5677)

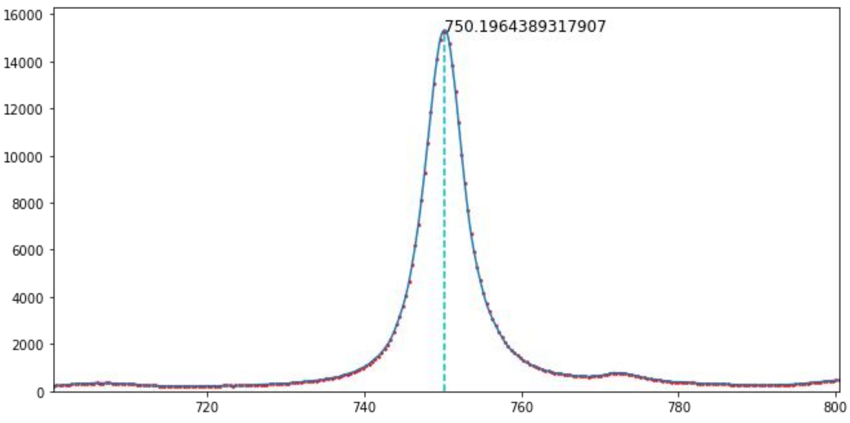
Peaks’ definition:

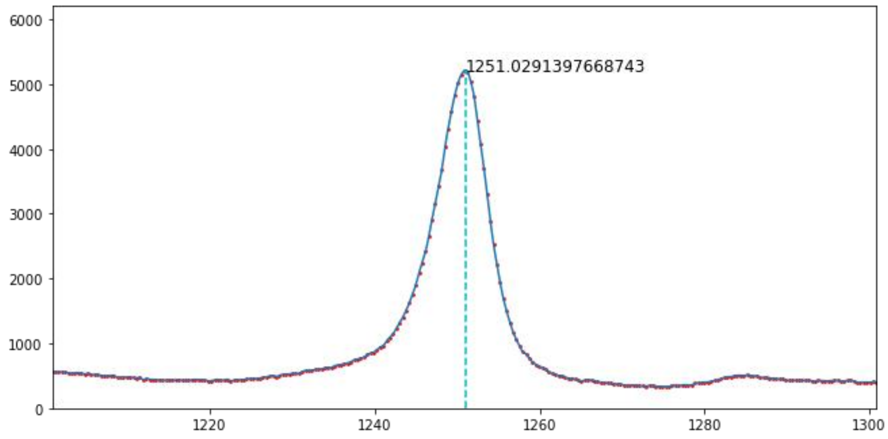
Peaks' intensity height > 1000, distance between 2 peaks more than 200 sample

(b)

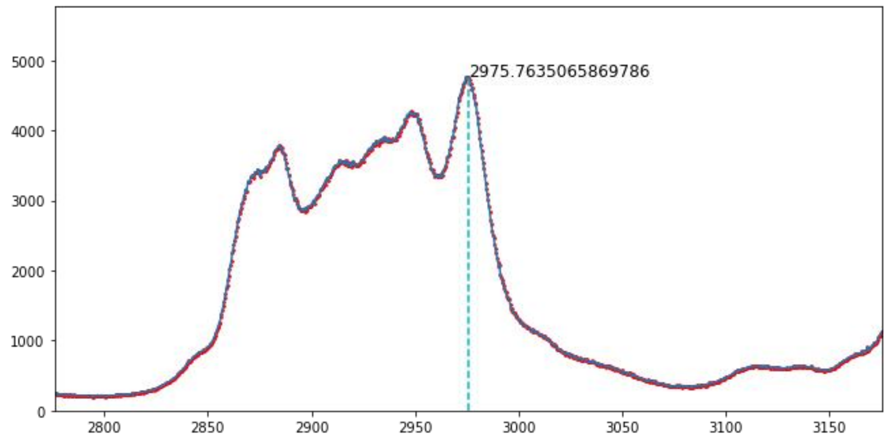


(c)

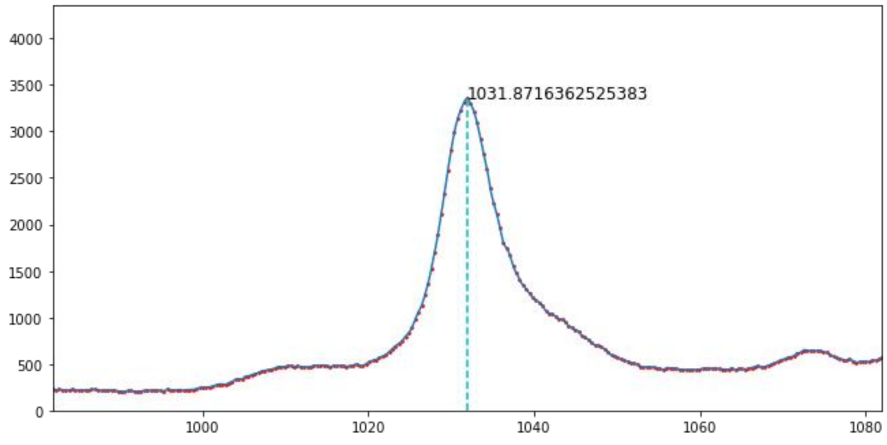




1251.0291397668743



2975.7635065869786

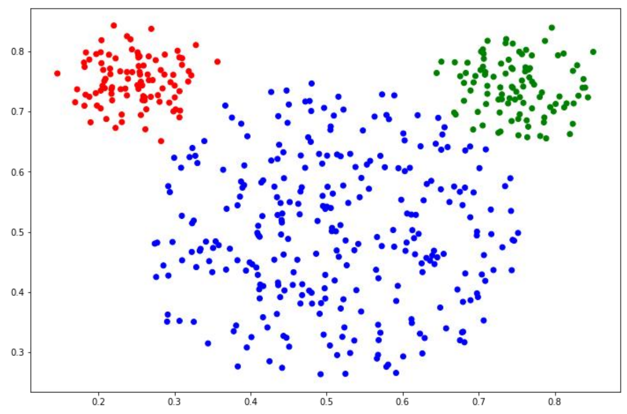


1031.8716362525383

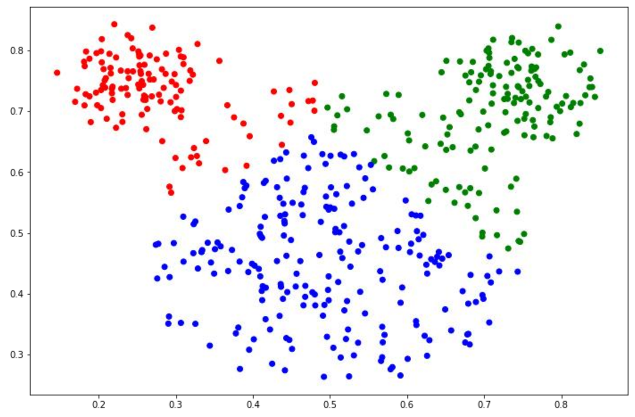
Q2

(a)

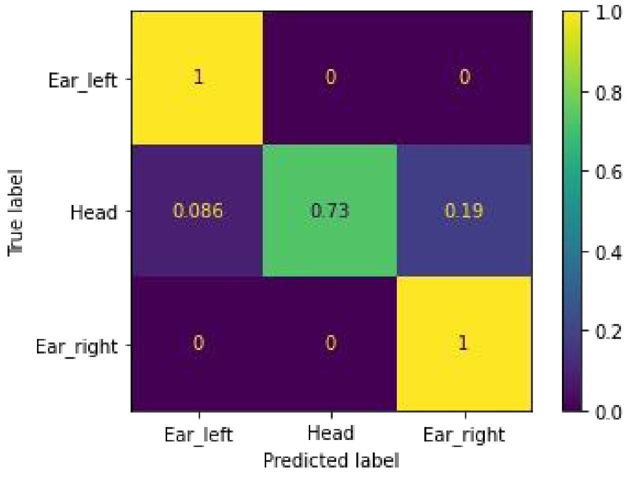
True



Predict:

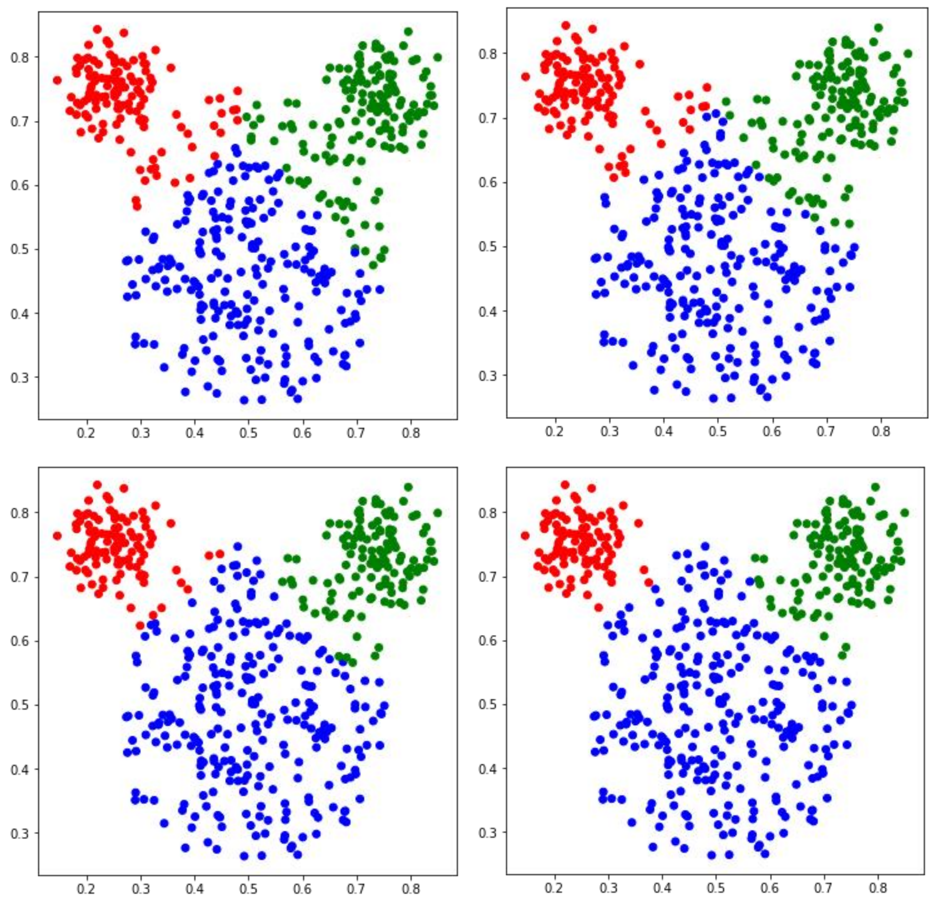


Confusion matrix

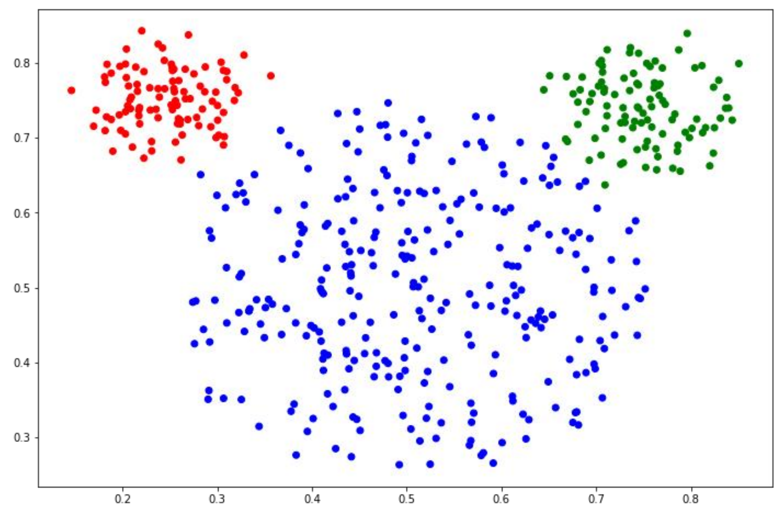


(b)

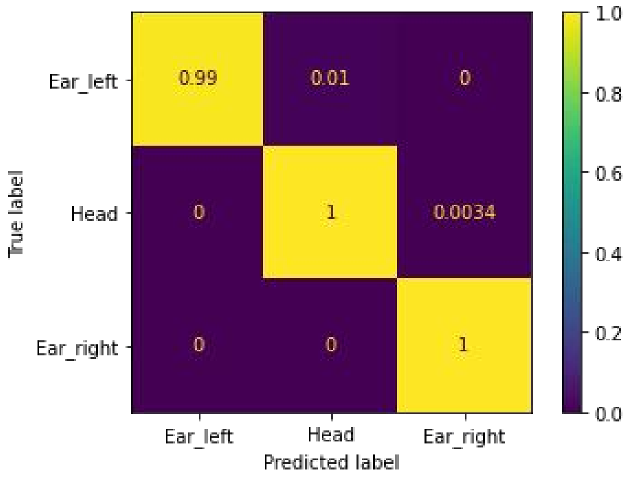
Firs 4 iterations:



Predict after 28 iterations:



Confusion matrix:



Clustering using GMM model behaves much better than K-means method.
It has more precision on bordering area while K-means only has 73% accuracy on head area.
After 28 iterations, GMM can have pretty good accuracy.

Code

Q1:

```
import matplotlib.pyplot as plt
import scipy.signal
import numpy as np
from scipy import interpolate as it
raman = "raman.txt"

wavenum, intensity = [],[]
with open(raman, 'r') as f:
    lines = f.readlines()
    for data in lines:
        value = [float(s) for s in data.split()]
        wavenum.append(value[0])
        intensity.append(value[1])
```

```
peaks, _ = scipy.signal.find_peaks(intensity, height = 1000, distance = 200)
peaks_wav = [wavenum[i] for i in peaks]
peaks_intensity = [intensity[i] for i in peaks]
Z = zip(peaks_wav, peaks_intensity)
Z = sorted(Z, key = lambda x: x[1], reverse = True)
peaks_wav, peaks_intensity = zip(*Z)
# Peak definition:
#         peaks' intensity height > 1000, distance between 2 peaks more than 200 samples
print(peaks_wav[0:8])
print(peaks_intensity[0:8])
fig, ax = plt.subplots(figsize=(12,8))
plt.plot(wavenum, intensity)
plt.scatter(peaks_wav[0:8], peaks_intensity[0:8], marker = "x", color = "red")
for i in range(8):
    ax.text(peaks_wav[i],peaks_intensity[i], peaks_intensity[i], size=12)
plt.xlabel("Wavenumber")
plt.ylabel("Intensity")
plt.show()

tck = it.splrep(wavenum, intensity, s=0)
xfit = np.arange(500, 3500, np.pi/50)
yfit = it.splev(xfit, tck, der=0)

deriv = it.splev(xfit, tck, der = 1)

zero_cross = []
for i in range(len(deriv) - 1):
    if deriv[i] * deriv[i+1] < 0 and yfit[i] > 1000:
        if xfit[i] > 740 and xfit[i] < 760:
            zero_cross.append(i)
        elif xfit[i] > 1240 and xfit[i] < 1260:
            zero_cross.append(i)
        elif xfit[i] > 2950 and xfit[i] < 3000 and yfit[i] > 4600:
            zero_cross.append(i)
        elif xfit[i] > 1020 and xfit[i] < 1040:
            zero_cross.append(i)

fig = plt.figure(figsize = (24,12))
ax1 = fig.add_subplot(221)
ax1.scatter(wavenum, intensity, s = 3,color = 'red')
ax1.plot(xfit, yfit)
ax1.vlines(xfit[zero_cross[0]], 0, peaks_intensity[0], colors = "c", linestyles = "dashed")
ax1.text(xfit[zero_cross[0]], yfit[zero_cross[0]], xfit[zero_cross[0]], size=12)
```

```
ax1.set_xlim(peaks_wav[0] - 50, peaks_wav[0] + 50)
ax1.set_ylim(0, peaks_intensity[0] + 1000)

fig = plt.figure(figsize = (24,12))
ax2 = fig.add_subplot(222)
ax2.scatter(wavenum, intensity, s = 3,color = 'red')
ax2.plot(xfit, yfit)
ax2.vlines(xfit[zero_cross[2]], 0, peaks_intensity[1], colors = "c", linestyles = "dashed")
ax2.text(xfit[zero_cross[2]], yfit[zero_cross[2]], xfit[zero_cross[2]], size=12)
ax2.set_xlim(peaks_wav[1] - 50, peaks_wav[1] + 50)
ax2.set_ylim(0, peaks_intensity[1] + 1000)

fig = plt.figure(figsize = (24,12))
ax3 = fig.add_subplot(223)
ax3.scatter(wavenum, intensity, s = 3,color = 'red')
ax3.plot(xfit, yfit)
ax3.vlines(xfit[zero_cross[5]], 0, peaks_intensity[2], colors = "c", linestyles = "dashed")
ax3.text(xfit[zero_cross[5]], yfit[zero_cross[5]], xfit[zero_cross[5]], size=12)
ax3.set_xlim(peaks_wav[2] - 200, peaks_wav[2] + 200)
ax3.set_ylim(0, peaks_intensity[2] + 1000)

fig = plt.figure(figsize = (24,12))
ax4 = fig.add_subplot(224)
ax4.scatter(wavenum, intensity, s = 3,color = 'red')
ax4.plot(xfit, yfit)
ax4.vlines(xfit[zero_cross[1]], 0, peaks_intensity[4], colors = "c", linestyles = "dashed")
ax4.text(xfit[zero_cross[1]], yfit[zero_cross[1]], xfit[zero_cross[1]], size=12)
ax4.set_xlim(peaks_wav[4] - 50, peaks_wav[4] + 50)
ax4.set_ylim(0, peaks_intensity[4] + 1000)
```

Q2:

```
import numpy as np
import matplotlib.pyplot as plt
import re
import math
from scipy.cluster.vq import kmeans, vq, whiten
from sklearn import metrics
from scipy.stats import multivariate_normal as mn
cluster = "cluster.txt"

#Read data
x, y, pairs, cls = [], [], [], []
with open(cluster, 'r') as f:
        lines = f.readlines()
        for line in lines:
                if re.match('[1-9]\d*.\d*|0.\d*[1-9]\d*[1-9]\d*.\d*|0\.\d*[1-9]\d*[A-Z]\w+', line, flags
= 0):
                        data = line.split()
                        x.append(float(data[0]))
                        y.append(float(data[1]))
                        pairs.append([float(data[0]), float(data[1])])
                        if data[2] == 'Head':
                                cls.append(1)
                        elif data[2] == "Ear_left":
                                cls.append(0)
                        elif data[2] == "Ear_right":
                                cls.append(2)

#Q2(a) predict
centroids,_ = kmeans(pairs,3)

centx = [centroids[i][0] for i in range(3)]
Z = zip(centroids, centx)
Z = sorted(Z, key = lambda x: x[1])
centroids, centx = zip(*Z)

km_res,_ = vq(pairs,centroids)
colors = ['red', 'blue', 'green']
col = []
for i in range(len(km_res)):
        col.append(colors[km_res[i]])
fig, ax = plt.subplots(figsize=(12,8))
plt.scatter(x, y, c = col)
plt.show()

#Q2(a) true
col2 = []
for i in range(len(cls)):
        if(cls[i] == 1):
                col2.append('blue')
        elif(cls[i] == 0):
                col2.append('red')
        elif(cls[i] == 2):
                col2.append('green')
fig, ax = plt.subplots(figsize=(12,8))
plt.scatter(x, y, c = col2)
plt.show()
```

```
#Confusion matrix
metrics.ConfusionMatrixDisplay.from_predictions(cls, km_res, normalize = 'true', display_labels =
['Ear_left', 'Head', 'Ear_right'])
plt.show()

#M-step
def update_mu(pairs, gamma, n_sample, n_class):
        mu = np.zeros((n_class,2))
        pairs = np.array(pairs)
        for i in range(n_class):
                top = 0
                bot = 0
                for j in range(n_sample):
                        top += gamma[j][i] * pairs[j]
                        bot += gamma[j][i]
                mu[i] = top / bot
        return mu

def update_covar(pairs, gamma, mu, n_sample, n_class):
        covar = np.zeros((n_class,2))
        pairs = np.array(pairs)
        for i in range(n_class):
                top = 0
                bot = 0
                for j in range(n_sample):
                        top += gamma[j][i] * ((pairs[i] - mu[i])**2)
                        bot += gamma[j][i]
                covar[i] = top / bot
        return covar

def update_covar1(pairs, gamma, mu, n_sample, n_class):
        covar = np.zeros((n_class,2))
        for i in range(n_class):
                covar[i] = np.average((pairs - mu[i])**2, axis = 0,weights = gamma[:,i])
        return covar

def update_w(gamma):
        w = gamma.sum(axis = 0) / gamma.sum()
        return w

#E-step
def gaussian_pdf(pairs, mu, covar, num, cls, dim):
        f = np.exp(-(1/2) * np.transpose(pairs[num] - mu[cls]) * np.linalg.inv(cov) * (pairs[num] -
mu[cls])) / (np.sqrt(np.linalg.det(covar)((2*math.pi)**dim)))
        return f

def update_gamma(pairs, mu, covar, w, n_sample, n_class):
        pdf = np.zeros((n_sample, n_class))
        for i in range(n_class):
                pdf[:,i] = w[i] * mn.pdf(pairs, mean = mu[i], cov = np.diag(covar[i]))
                gamma = pdf / pdf.sum(axis = 1).reshape(-1, 1)
        return gamma

def neg_log_like(paris, mu, covar, w, n_sample, n_class):
        pdf = np.zeros((n_sample, n_class))
        for i in range(n_class):
                pdf[:,i] = w[i] * mn.pdf(pairs, mean = mu[i], cov = np.diag(covar[i]))
        l = np.log(pdf.sum(axis = 1))
        return np.mean(l)

#Initialize
gamma = []
w = np.zeros(3)
for i in range(len(km_res)):
        if(km_res[i] == 0):
                temp = [1, 0, 0]
                w[0] += 1
                gamma.append(temp)
        if(km_res[i] == 1):
                temp = [0, 1, 0]
                w[1] += 1
                gamma.append(temp)
        if(km_res[i] == 2):
                temp = [0, 0, 1]
                w[2] += 1
                gamma.append(temp)
w = w / 490
gamma = np.array(gamma)
mu = update_mu(pairs, gamma, 490, 3)
covar = update_covar1(pairs, gamma, mu, 490, 3)

#First 4 iterations:
gmm_res = []
for i in range(490):
        gmm_res.append(np.argmax(gamma[i]))
col2 = []
for i in range(len(gmm_res)):
        col2.append(colors[gmm_res[i]])
fig, ax = plt.subplots(figsize = (6, 6))
plt.scatter(x, y, c = col2)
plt.show()
for n in range(3):
        gamma = update_gamma(pairs, mu, covar, w, 490, 3)
        w = update_w(gamma)
        mu = update_mu(pairs, gamma, 490, 3)
        covar = update_covar1(pairs, gamma, mu, 490, 3)
        gmm_res = []
        for i in range(490):
                gmm_res.append(np.argmax(gamma[i]))
        col2 = []
        for i in range(len(gmm_res)):
                col2.append(colors[gmm_res[i]])
        fig, ax = plt.subplots(figsize = (6, 6))
        plt.scatter(x, y, c = col2)
        plt.show()

#Iteration
prev_l = 100
count = 0
while True:
        count += 1
        gamma = update_gamma(pairs, mu, covar, w, 490, 3)
        w = update_w(gamma)
        mu = update_mu(pairs, gamma, 490, 3)
        covar = update_covar1(pairs, gamma, mu, 490, 3)
        l = neg_log_like(pairs, mu, covar, w, 490, 3)
        if(abs(prev_l - l) < 10**-15):
                print("Iteration : ", count)
                break
        else:
                prev_l = l

#Plot
gmm_res = []
for i in range(490):
        gmm_res.append(np.argmax(gamma[i]))
col2 = []
for i in range(len(gmm_res)):
        col2.append(colors[gmm_res[i]])
fig, ax = plt.subplots(figsize=(12,8))
plt.scatter(x, y, c = col2)
plt.show()

#Confusion matrix
metrics.ConfusionMatrixDisplay.from_predictions(cls, gmm_res, normalize = 'true', display_labels
= ['Ear_left', 'Head', 'Ear_right'])
plt.show()
```